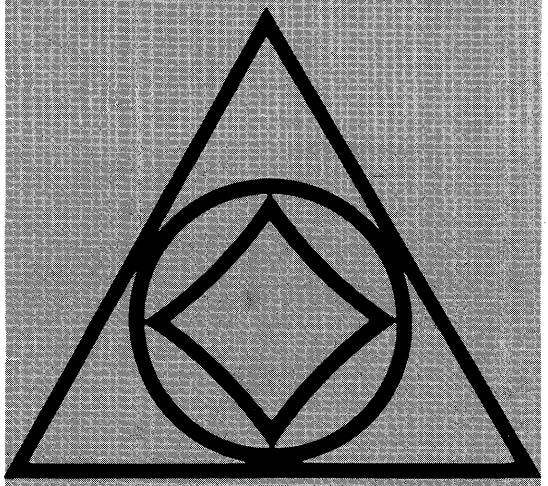


# AFIPS

## CONFERENCE PROCEEDINGS

## VOLUME 27 PART 1

# 1965

## FALL JOINT COMPUTER CONFERENCE

1965
SPARTAN BOOKS
Washington, D. C.
MACMILLAN AND CO., LTD.
London

The ideas and opinions expressed herein are solely those of the authors and are not necessarily representative of or endorsed by the 1965 Fall Joint Computer Conference Committee or the American Federation of Information Processing Societies.

Library of Congress Catalog Card Number 55–44701
Spartan Books, Div. of
Books, Inc.
1250 Connecticut Avenue, N. W.
Washington, D. C.



Sold distributors in Great Britain, the British Commonwealth, and the Continent of Europe:

Macmillan and Co., Ltd.
4 Little Essex Street
London W. C. 2

## PREFACE

This volume records in part the technical material presented at the 1965 Fall Joint Computer Conference. Contained in this publication are the formal papers selected from a record number of contributions submitted to the Technical Program Committee. No attempt has been made to incorporate the material presented at panels and tutorial sessions of the Conference, nor have the invited papers presented on the final day of the Conference been included. The Conference Committee hopes that a subsequent volume will emerge to catch the living spirit of such deliberations.

Still, the size of this volume is large, just as the scope of the Conference is broad. This is, in part, deliberate, since the Conference attempted to provide the opportunity for professional communication on every level. Recognizing the increasing degree of specialization in the hardware and software fields, the Technical Program Committee added a third information channel to the Conference to focus attention on management and applications. These sessions dealt with questions of marketing and economics as well as applications in the scientific and humanistic fields. Thus to the *orientation* in hardware and software was added the *direction* of applications and management in the disciplines that are concerned with information processing.

The most distinctive feature of this Conference, however, must be the five "discuss-only" sessions for which preprints were available before the Conference. Hopefully, new dimensions were added to the papers through a searching examination of the material on the floor of the Conference. We regret that we cannot record the results and evaluate the technique.

The real and permanent contribution of the 1965 Fall Joint Computer Conference is still the technical material presented in this volume. The credit goes to the authors with greatful appreciation of the role of the Technical Program Committee and Session Chairmen who engineered the structure. Behind them are the contributions of many others who, as members of the various committees, made the Conference possible.

ROBERT W. RECTOR, *General Chairman*
*1965 Fall Joint Computer Conference*

# CONTENTS


| | | Page |
|---|---|---|
| Preface | | iii |
| **SESSION 1: PROGRAMMING LANGUAGES** | | |
| Universal Programming Languages and Processors: A Brief Summary and New Concepts | WALTER H. BURKHARDT | 1 |
| Digital Simulation Languages: A Critique and A Guide | JOHN J. CLANCY<br>MARK S. FINEBERG | 23 |
| Automatic Simplification of Mathematical Expressions — The Formac Algorithm | R. G. TOBEY<br>R. J. BOBROW<br>S. N. ZILLES | 37 |
| The New Block Diagram Compiler for Simulation of Sampled-Data Systems | B. J. KARAFIN | 53 |
| Two-Dimensional Programming | MELVIN KLERER<br>JACK MAY | 63 |
| **SESSION 2: ADVANCES IN COMPUTER ORGANIZATION** | | |
| Microprogramming for Data Acquisition and Control | W. C. MCGEE<br>H. E. PETERSEN | 77 |
| Picoprogramming: A New Approach to Internal Computer Control | R. E. BRILEY | 93 |
| A Precession Pattern in a Delay Line Memory | S. P. FRANKEL<br>J. HERNANDEZ | 99 |
| An Associative Parallel Processor with Application to Picture Processing | R. M. BIRD<br>R. H. FULLER | 105 |
| Computer Organization for Array Processing | D. N. SENZIG<br>R. V. SMITH | 117 |
| **SESSION 3: EFFICIENCY AND MANAGEMENT OF COMPUTER INSTALLATIONS** | | |
| Management Problems of Aerospace Computer Centers | G. A. GARRETT | 129 |
| The Multi-Discipline Approach: A Marketing Application | B. G. MENDELSON<br>R. V. MONAGHAN | 139 |
| Organizational Philosophy and the Computer Center | M. H. GOTTERER<br>A. W. STALNAKER | 145 |

| | | Page |
|---|---|---|
| Planning for Generalized Business Systems | R. V. HEAD | 153 |
| Computer Systems Design and Analysis Through Simulation | G. K. HUTCHINSON<br>J. N. MAGUIRE | 161 |
| Basic Concepts for Planning an Electronic Data Processing System | A. F. MORAVEC | 169 |
| **SESSION 6: A NEW REMOTE ACCESSED MAN-MACHINE SYSTEM** | | |
| Introduction and Overview of the Multics System | F. J. CORBATÓ<br>V. A. VYSSOTSKY | 185 |
| System Design of a Computer for Time-Sharing Applications | E. L. GLASER<br>G. A. OLIVER | 197 |
| Structure of the Multics Supervisor | V. A. VYSSOTSKY<br>F. J. CORBATÓ<br>R. M. GRAHAM | 203 |
| A General-Purpose File System for Secondary Storage | R. C. DALEY<br>P. G. NEUMANN | 213 |
| Communications and Input-Output Switching in a Multiplex Computing System | J. F. OSSANNA<br>L. MIKUS<br>S. D. DUNTEN | 231 |
| Some Thoughts About the Social Implications of Accessible Computing | E. E. DAVID, JR.<br>R. M. FANO | 243 |
| **SESSION 7: APPLICATIONS OF SIMULATION** | | |
| Structure and Dynamics of Military Simulations | E. LEVINE | 249 |
| Analogue-Digital Data Processing of Respiratory Parameters | T. W. MURPHY | 253 |
| Computer Simulation: A Solution Technique for Management Problems | A. J. ROWE | 259 |
| The Role of the Computer in Humanistic Scholarship | E. A. BOWLES | 269 |
| The Structure and Character of Useful Information-Processing Simulations | L. FEIN | 277 |
| **SESSION 8: NATURAL LANGUAGE PROCESSING** | | |
| Catalogs: A Flexible Data Structure for Magnetic Tape | MARTIN KAY<br>THEODORE ZIEHE | 283 |

| | | Page |
|---|---|---|
| Information Search Optimization and Iterative Retrieval Techniques | J. J. Rocchio<br>G. Salton | 293 |
| An Economical Program for Limited Parsing of English | D. C. Clarke<br>R. E. Wall | 307 |
| The Mitre Syntactic Analysis Procedure for Transformational Grammars | Arnold M. Zwicky<br>Joyce Friedman<br>Barbara C. Hall<br>Donald E. Walker | 317 |

SESSION 9: CELLULAR TECHNIQUES FOR LOGIC, MEMORY AND SYSTEMS

| | | |
|---|---|---|
| Cobweb Cellular Arrays | R. C. Minnick | 327 |
| Two-Dimensional Iterative Logic | R. H. Canaday | 343 |
| Two-Rail Cellular Cascades | R. H. Short | 355 |
| Associative Memory Structure | B. T. McKeever | 371 |

SESSION 11: THE REVOLUTION IN WRITTEN COMMUNICATION

| | | |
|---|---|---|
| Computer Editing, Typesetting and Image Generation | M. V. Mathews<br>Joan E. Miller | 389 |
| The Left Hand of Scholarship: Computer Experiments with Recorded Text as a Communication Medium | Glenn E. Roudabush<br>Charles R. T. Bacon<br>R. Bruce Briggs<br>James A. Fierst<br>Dale W. Isner<br>Hiroshi A. Noguni | 399 |

SESSION 12: ON-LINE INTERACTIVE SOFTWARE SYSTEMS

| | | |
|---|---|---|
| MATHLAB: A Program for On-Line Machine Assistance in Symbolic Computations | C. Engelman | 413 |
| An Integrated Computer System for Engineering Problem Solving | D. Roos | 423 |
| AESOP: A Prototype for On-Line User Control of Organizational Data Storage, Retrieval and Processing | E. Bennett<br>E. Haines<br>J. Summers | 435 |
| Structuring Programs for Multi-Program Time-Sharing On-Line Applications | K. Lock | 457 |
| Interactive Machine Language Programming | B. W. Lampson | 473 |

| | | Page |
|---|---|---|
| Responsive Time-Sharing Computer in Business — Its Significance and Implications | CHARLES W. ADAMS | 483 |

SESSION 13: HIGH SPEED COMPUTER LOGIC CIRCUITS

| | | |
|---|---|---|
| Circuit Implementation of High Speed Pipeline Systems | LEONARD W. COTTON | 489 |
| High Speed Logic Circuit Considerations | W. H. HOWE | 505 |
| Crosstalk and Reflections in High Speed Digital Systems | A. FELLER<br>H. R. KAUPP<br>J. J. DIGIACOMO | 511 |

SESSION 14: COMPUTERS IN THE BIOLOGICAL AND SOCIAL SCIENCES

| | | |
|---|---|---|
| Integrating Computers into Behavioral Science Research | HAROLD BORKO | 527 |
| Data Analysis in the Social Sciences | GEOFFREY H. BALL | 533 |
| Nonlinear Regression Models in Biology | JOSEPH A. STEINBORN | 561 |
| Computer Correlation of Intracellular Neuronal Responses | FREDERICK F. HILTZ | 567 |
| Information Processing of Cancer Chemotherapy Data | ALICE R. HOLMES<br>ROBERT K. AUSMAN | 583 |

SESSION 18: TIME-SHARED COMPUTER SYSTEMS: SOFTWARE/HARDWARE CONSIDERATIONS

| | | |
|---|---|---|
| A Facility for Experimentation in Man-Machine Interaction | R. WAYNE LICHTENBERGER<br>MELVIN W. PIRTLE | 589 |
| A Time- and Memory-Sharing Executive Program for Quick-Response On-Line Applications | JAMES W. FORGIE | 599 |
| Design for a Multiple User Multiprocessing System | JAMES D. MCCULLOUGH<br>KERMITH H. SPEIERMAN<br>FRANK W. ZURCHER | 611 |
| A Computing System Design for User Service | WEBB T. COMFORT | 619 |

SESSION 19: SCRATCHPAD MEMORIES

| | | |
|---|---|---|
| Design Considerations for a 25-Nsec Tunnel Diode Memory | D. J. CRAWFORD<br>R. L. MOORE<br>J. A. PARISI<br>J. K. PICCIANO<br>W. D. PRICER | 627 |
| SMID: A New Memory Element | R. P. SHIVELY | 637 |

| | | Page |
|---|---|---|
| An Experimental Sixty-Five Nanosecond Thin Film Scratchpad Memory System | G. J. AMMON<br>C. NEITZERT | 649 |
| Impact of Scratchpads in Design; Multi-Functional Scratchpad Memories in the Burroughs B8500 | S. E. GLUCK | 661 |
| Scratchpad Oriented Designs in the RCA Spectra 70 | A. T. LING | 667 |
| Scratchpad Memories at Honeywell: Past, Present and Future | N. NISENOFF | 679 |
| **SESSION 20: ARITHMETIC TECHNIQUES AND SYSTEMS** | | |
| A Bounded Carry Inspection Adder for Fast Parallel Arithmetic | EMANUEL KATELL | 689 |
| A Fast Conditional Sum Adder Using Carry Bypass Logic | JOSEPH F. KRUY | 695 |
| A Checking Arithmetic Unit | RICHARD A. DAVIS | 705 |
| Serial Arithmetic Techniques | M. LEHMAN<br>D. SENZIG<br>J. LEE | 715 |
| **SESSION 23: SIMULATION OF HUMAN BEHAVIOR** | | |
| Simulation Models for Psychometric Theories | C. E. HELM | 727 |
| Human Decision Making Under Uncertainty and Risk: Computer-Based Experiments and a Heuristic Simulation Program | N. V. FINDLER | 737 |
| Computer Experiments in Motor Learning | G. R. BUSSEY | 753 |
| **SESSION 24: HIGH SPEED READ ONLY MEMORIES** | | |
| A Survey of Read Only Memories | MORTON H. LEWIN | 775 |
| A High-Speed, Woven Read-Only Memory | M. TAKASHIMA<br>H. MEADA<br>A. J. KOLK JR. | 789 |
| A Thin Magnetic Film Computer Memory Using a Resonant Absorption Non-Destructive Read-Out Technique | M. MAY<br>J. L. ARMSTRONG<br>W. W. POWEL | 801 |
| Development of an E-Core Read-Only Memory | P. S. SIDHU<br>B. BUSSELL | 809 |

Page

SESSION 25: INPUT/OUTPUT EQUIPMENT
FOR CLOSER MAN-MACHINE INTERFACE

MAGIC: A Machine for Automatic Graphics Interface to a Computer — D. E. Rippy, D. E. Humphries, J. A. Cunningham — 819

A Magnetic Device for Computer Graphic Input — M. H. Lewin — 831

Graphic I — A Remote Graphical Display Console System — W. H. Ninke — 839

The Beam Pen: A Novel High Speed, Input/Output Device for Cathode-Ray-Tube Display Systems — D. R. Haring — 847

Voice Output from IBM System/360 — A. B. Urquhart — 857

SESSION 26: INDUSTRIAL APPLICATIONS

Corrugator Plant Operating System — Walter J. Koch — 867

Real Time Programming and Athena Support at White Sands Missile Range — William G. Davidson — 871

Quality Evaluation of Test Operation via Electronic Data Processing — A. A. Daush — 879

The Introduction of Man-Computer Graphics into the Aerospace Industry — S. H. Chasen — 883

SESSION 27: HYBRID COMPUTERS FOR FUTURE SYSTEMS

Hybrid Computation for Lunar Excursion Module Studies — Arthur Burns — 893

Optimum Design and Error Analysis of Digital Integrators for Discrete System Simulation — Roger W. Burt, Andrew P. Sage — 903

Sequential Analog-Digital Computer (SADC) — Herman Schmid — 915

Design of a High Speed DDA — M. W. Goldman — 929

SESSION 28: COMPUTER DIMENSIONS IN LEARNING

Engineering Mathematics via Computers — John Staudhammer — 951

The Computer: Tutor and Research Assistant — Robert J. Meyer — 959

WOSP: A Word-Organized Stored-Program Training Aid — M. Raspanti — 965

CASE: A Program for Simulation of Concept Learning — Frank B. Baker — 979

Page

SESSION 29: MEMORIES FOR FUTURE COMPUTERS

| | | |
|---|---|---|
| A 375 Nanosecond Main Memory System Utilizing 7 Mil Cores | G. E. WERNER<br>R. M. WHALEN | 985 |
| Monolithic Ferrite Memories | I. ABEYTA<br>M. KAUFMAN<br>P. LAWRENCE | 995 |
| High Speed Ferrite 2-½D Memory System | T. J. GILLIGAN | 1011 |
| Design and Fibrication of a Magnetic Thin Film Integrated Circuit Memory | T. J. MATCOVICH<br>W. FLANNERY | 1023 |
| Batch Fabricated Matrix Memories | T. L. MCCORMACK<br>C. R. BRITTARD<br>H. W. FULLER | 1035 |
| Integrated Semi-Conductor Memory System | HARLEY A. PERKINS<br>JACK D. SCHMIDT | 1053 |

SESSION 30: COMPUTER-AIDED DESIGN & MAINTENANCE

| | | |
|---|---|---|
| Strobes-Shared Time Repair of Big Electronic Systems | J. T. QUATSE | 1065 |
| A Self-Diagnosable Computer | R. E. FORBES<br>D. H. RUTHERFORD<br>C. B. STIEGLITZ<br>L. H. TUNG | 1073 |
| An Automated Interconnect Design System | W. E. PICKRELL | 1087 |
| Systematic Design of Automata | J. P. ROTH | 1093 |

# UNIVERSAL PROGRAMMING LANGUAGES AND PROCESSORS: A BRIEF SURVEY AND NEW CONCEPTS

Walter H. Burkhardt
*Computer Control Company, Inc.*
*Framingham, Massachusetts*

## INTRODUCTION

Progress in any field depends on the materialization of new ideas. But before this is possible, these ideas have to be found, investigated, developed and adapted to the changing world.

In computing, i.e., the use of computers for the solution of problems, new ideas are available everywhere, although the implications behind them and the influence on the state-of-the-art are generally not very well understood. Therefore it is often difficult to separate the wheat from the chaff.

But even valuable ideas are not always useful and welcome. That is especially the case when the basis for them is not adquately prepared. To know which ideas are useful at present, it is necessary to evaluate the state-of-the-art to determine how developments in the field will proceed. There are other reasons. One might be to give the nonspecialist a fast orientation; another is to readjust the basis in a fast growing and changing field.

The last decade brought a tremendous gain in overall computer power and for a unit outlay as well. Therefore, it is not too surprising if many old values have to be changed and new ones appear.

The advent of computers gave a very useful tool for the solution of many tasks for which the statment of the problem was given in a fixed mathematical form. This is due to the special nature of computers, with the memories, the circuit logic, and electronic switching elements having easy adaptation to mathematical problems and to a tremendous bulk of knowledge in the form of mathematical formalism.

There are now on the one side machines with more or less special features for the solution of particular problems, and on the other the problems, given sometimes in a self-contained formulation, sometimes in only a vague and inexact form, and ranging over the whole spectrum of life, science, and society. The medium to combine both is known as programming. This function consists of mapping a solution given to the problems on the machine, but now better defined as dividing the problems into elementary task-components and translating them into terms of the machine.

In this paper, the interface between the problems and the machines will be discussed with emphasis on the tools for the solutions—the programming languages and processors.

### *Statement of Problem*

The application of computers for solving problems in technical, scientific, and commercial fields

has been very successful. But progress is hampered by the fact that the machines accept primarily only their own special language, on digital computers composed of number sequences. These sequences are mostly long chains of zeros and ones—which is rather unintelligible to humans and quite different from the languages in which the tasks are and can be easily described.

*Possible Solutions*

There are two possibilities for solving the difficulties made by the gap between the languages of machines and the languages of problems. One solution would be to adapt the languages of the machines by developing machine languages more general and closer to the problems encountered, the high-level language computers, the other one would be to adapt the problems to the machines. This is done presently with intermediate languages, between machines and problems, which are easier for people to use than contemporary absolute machine languages.

*High-Level Language Computers.* This would mean to develop a machine which could communicate in a higher language. Suggested rather early, and attempted to implement to some extent (for example, in the SEAC machine,[1] this idea could give an elegant solution to the problem. Therefore perhaps it is revived in newer designs,[2,3] and it is even suggested to use a language of the ALGOL-type[4] as machine language.* In addition to the drawbacks due to the insufficiencies of contemporary programming languages (and these are the only candidates at present for high-level machine languages) there are several factors opposed to such a development.

The arguments of high cost for circuitry and restrictions to the applications area are mainly based on the economic feasibility of such designs. But with an advent of very cheap components and assembly methods, these restrictions could change in the future.

The arguments of altering bases must be taken more seriously. The development is neither fixed on the problem side nor on the machine side.

*Development on the Problem Side.* To illustrate this point a simple example might be taken. In applications to commercial problems a basic function is certainly sorting, which is used over and over again. So it would seem natural to include a machine operation for sorting in the repertoire of such high-level commercial machines. But what technique of sorting[5] should be implemented? The best technique to be selected depends on the data formats and on the machine configurations so that selecting only one technique is not very feasible. But inclusion of several different techniques is highly unlikely. This example will show the difficulties for only one task function. The overall requirements complicate the situation so much that no reasonable solution is in sight.

*Development on Machine Side.* Many opinions state the view that the development on the machine side is now fixed.[6] But this belief seems prejudiced and premature. For example, in the near future memory hierarchies (let's say a memory of 128-word diode storage with 50 nanoseconds and 2048 words thin film or thin wire with 200 nanoseconds and back-up storage of 32,768 words at 600 nanosecond cycle time. Behind these might be bulk core storage, drums, disks and tapes) could give a user more computer power (according to the principle of limited resources) than the more conventional recent design; or mastery of parallel execution features, etc. Although this argument affects mainly the internal design of a possible high-level language machine, it complicates the picture and eliminates many suggestions for solutions. The potentialities for a standard machine (or assembly) language are impeded too by this aspect.

*Solutions by Intermediate Languages.* The solution by intermediate steps between problem and machine languages via programming was at least in the past the most successful one. It can easily be seen that the closer to the problem the steps are taken, the more powerful and quickened the solution will be. So the region between problems and machines contains software levels of differing machine and problem independence.

*Efficiency of Machine Use.* Whenever a programming language is different from actual low-level machine language, questions concerning the efficient use of the hardware are apt to arise. These seem to be of greatest importance on slow and expensive machines. Linearly extrapolated, the emphasis on these questions is decreased to 2 percent when relating a machine with 0.5-miscrosecond cy-

*A similar step in this direction is sometimes attempted in microprogrammed machines with some higher-level language implemented in memory in a semifixed manner.

cle time in 1965 to one with 25-miscrosecond cycle time in 1951 at the same price. Interestingly, the highest requirements for run-time optimization with compilers are imposed on hardware which is inadquate for the intended problem solutions (e.g., optimization in FORTRAN II on the 704 and in ALPHA on the M20[8] for the solution of difficult partial differential equations). With the need for faster computers[9] and a decline in prices for hardware, as in the past decade, these efficiency questions are bound to diminish and perhaps to disappear altogether.

*Hierarchy of Programming Languages.* Different hierarchies of programming languages are already proposed,[10] where the criterion is the machine configuration concerned. Of course, many other characteristics could be chosen for classification of programming languages, but the one here presented in respect to machine independence seems to be most interesting. A good measure for the level is the degee of declarative freedom for the user. Therefore on the lowest level would be the absolute machine languages and with more declarative possibilities gradually increasing up to the problem level of declarative languages as follows:

| | |
|---|---|
| Absolute machine languages (machine level) | No declarative freedom |
| Assembly languages | No specification of names and locations necessary |
| Procedural languages | No detailed machine commands necessary |
| Problem oriented languages | Language elements from problem but command structure procedural |
| Specification languages, augmented by semantics | Description of relations in the problem, freedom of procedure |
| Declarative languages (problem level) | Description of the problem, freedom of procedure and solution |

The levels from absolute machine language to procedural languages are very well known from the literature of recent years. (Sometimes in the past, procedural languages like FORTRAN, ALGOL and JOVIAL were incorrectly denoted as problem-oriented languages.) Examples for problem-oriented languages are found in APT, COGO, Simscript, etc.[11] The block-notation languages[12] for analog-hybrid simulation on digital computers are examples of augmented specification languages. Semantic content is there defined by the fixed meaning of the block names (in MIDAS[13] they are the operations and the operands by means of the successor specification). Recently an example for another use of a specification language in an applications program was published[14] where Backus-Naur-Form specification was adopted. As can be expected, the experience reported stresses the improved facilities (compared with conventional programming languages) in programming, check-out, and incorporating changes into the program over conventional programming languages. Perhaps the first example in declarative languages, although not on the level designed by the term today, was the dynamic analysis language DYANA.[15] Some other approaches are described in a recent paper.[16]

*Translation among Programming Languages.* All programming languages above the actual machine language impose the necessity for translation to that language. This task is done by a translator, compiler or assembler, hereafter called a processor.

Two different aspects have to be distinguished concerning the translation of programs:

1. Translations horizontally on one level
2. Translations vertically to other levels

Obviously, all translations can be regarded as composed of these two possibilities to various degrees. The requirements for the practicability of translation are:

- The rules for the connections of elements in the languages (the grammars or syntaxes).
- The elements of the languages (the dictionaries).
- Their respective relations in both languages as well.

1. Translations Horizontally. Horizontal translations of programs among different programming languages of the same level are in general not possible. The reason is, that the results of one operation (in extended sense) in a program in source language (the language of the input) may determine the kind of operation to be used next in the program, and that often the target equivalent of a source language item is not available. The criterion for translatability is that all operations in the source language can be translated separately into the target language (the language of the output) in respect to power and extend. Translatability from

one source language A to a target language B gives, however, no indications for translatability from B to A. Whenever some operations are not translatable, they may be simulated, e.g., interpreted at run time. Because of the huge number of redundant operation parts involved, interpreted programs run normally orders of magnitudes slower than comparable translated ones on the same machine.

2. Translation Vertically. Vertical translation of programs is divided in (a) upward and (b) downward translation.

(a) Upward translations impose generally the same requirements as those detailed for horizontal translations. A special case governs the upward translation or previously downward translated programs. Contrary to some opinion,[17] no relevant information for the execution of a program is lost in the translation process, only the redundant. Therefore, if the translation algorithm is given, all necessary information can be retrieved from programs, that had been translated before, to build a fully equivalent program on the former level.

(b) Downward translations are normally not difficult, because the languages on the higher levels are so designed as to give a specific and determined target equivalent on the lower level for each source language element.

Now, by the mechanical transformation of the program (a description of a problem or its solution) into representations of other levels with or without intermediate levels (e.g., DYANA → FORTRAN II, FORTRAN II → SAP704, SAP704 → 704) not more solutions of a problem are obtained, but only different representations of the program. Therefore, with regard to problem considerations, all different representations of one program (e.g., diagrams augmented by text, DYANA, FORTRAN II, SAP and 704), and all programs giving the same results for the same sets of data, are said to be equivalent. A similar relation is given among specification languages or notations.[18] Continuing this thought, most efficiency questions, grammar and syntax peculiarities and details, though interesting and necessary for the development of the transformation processors, are definitely unimportant and sometimes even undesirable for the solution of a task in applications programming.

*Experience with High-Level Programming Languages.* The aspects of the historical development of high-level programming languages (with regard to machine independence) are described in detail elsewhere.[11] It might be stressed that FORTRAN was not the first high-level language of algebraic type but had forerunners in Rutishauser's algebraic language on the Ermeth in Zurich and in Lanig and Zirler's language on Whirlwind in MIT. Even MATH-MATIC for the Univac I, a commercially available machine, was earlier. But the small UI (a decimal and alphanumeric machine with only 45 instructions) did not really necessitate and justify a high-level algebraic language; this was later required with the more complex machines of von Neumann-type, like the 704.

The advantages of high-level programming languages are more apparent the more the considered languages are independent from the machines. These advantages are:

1. Easier learning and use than lower-level languages, because they are less complicated,
2. Time savings in programming of solutions for problems,
3. Time savings in debugging and correcting possibilities for slightly different problems,
5. Higher machine independence for transition to other computers, and otherwise for compatibility with hardware,
6. Better documentation (compatibility among programs and different programmers,
7. More powerful structuring in terms of problem.

Points (1), (2), and (3) were stressed in the past and found most important.[19] Nowadays (4) and (5) receive more attention and in the future (5), (6), and (7) may become the dominant ones.

It is interesting to note that points (1) through (4) have been similarly known to engineers for decades for the solution of problems in formal instead of numerical notation.

Most astonishing is the large number of programs still written in a low-level language.[20] This can only be explained by a steep information gradient between the knowledge in the field and the application programmers, or better, their managers.

### *Development of New High-Level Programming Languages*

*Introduction.* The development of new high-level programming languages, at least in the past, has been more evolutionary than revolutionary. So the

step from FORTRAN to ALGOL brought with it these advantages in order of their estimated importance:

- Chained logical decision sequences
- Block structure of program parts
- Free notation format
- Lifting of various machine restrictions (i.e., number of subscripts in variables, modes of expressions, etc.)

Unfortunately, due perhaps to the ambiguities embedded in ALGOL and its definition, the gain from switching over to ALGOL programming from FORTRAN is considered marginal. Despite all the efforts in the past, less than 10 percent of all programs for scientific and engineering applications are coded in ALGOL[20] — which is not a striking triumph for a successor to FORTRAN.[21] Similarly, less than 5 percent of the programs in the same area are coded in FORTRAN IV — what can be cautiously described as failure of the new facilities incorporated in FORTRAN IV over FORTRAN II. The use of a programming language by applications programmers has to be the measure for its success. If one is not sufficiently used, a programming language is certainly as dead and obsolete as Mayan or Babylonian and perhaps of just academic interest.

*Requirements for a New High-Level Programming Language.* Several important design criteria — often violated even in recent designs — have to be stressed:

*Close Relationship to the Problems in the Desired Area.* This allows the user a concise and powerful description of the processes and concepts.

*Uniqueness.* Each item in a correct program has to have one unique and defined meaning. This is required by all compatibility reasons.

*Simplicity and Clearness of Notation.* The language has to be developed and designed with a programming notation for ease of learning, use and handling of the language in the intended problem area. (Of course, that does not exclude a formal and rigid definition of the language. But such a definition should hardly ever be imposed upon a user.) Requirements for readability and intelligibility are included here. This point of convenience has to be the main criterion for the standardization of programming languages. Admittedly, generally one proposed standard is better than another, if it is more convenient for the user.

*Completeness.* A good programming language should be able to handle all occurring tasks within its designed scope, without need for using means from outside. Good counter-examples are the missing string-handling facilities in FORTRAN and the input/output part in ALGOL 60.

*Segmentation.* For the practical application of programming languages to large problems, efficient segmentation features are desirable so that parts of problems can be handled independently.

*Compatibility with Existing Programming Languages.* In addition to compatibility in other respects, one is important in regard to the already accumulated knowledge of problem solutions (the program libraries). These libraries consist of two parts—one created by the present user working with the language and the other developed elsewhere or earlier with other languages. The first part requires elements in the language to build up and use older programs and program parts in new connotations; the second demands some means for translation or interpretation of old libraries.

*Development Possibilities.* There are three ways of developing a new programming language:

- Cleaning up and refining existing languages;
- Elaboration and combination of known useful features;
- Development from the basic requirements of a problem area.

All three methods were used in the past either separately or combined.

*Proliferation and Solutions.* The application of computers with high-level languages to different problem areas causes a proliferation of programming languages according to the vernaculars in the application fields. There are two different possibilities:

1. If single programming languages are to be developed close to the vernaculars, then some incompatibility will exist between these.
2. On the other hand, if an intermediate language somewhere in the middle between problems and machines will be accepted as the programming standard, then much

more effort has to be spent on defining the problems to the computers.

The historical development of progress in the computer field favors the first alternative, while computer manufacturers and operations managers of computer installations try to hold to the second one. Possible solutions to the dilemma might be found in:

(a) Inclusion of language elements of neighboring problem areas into programming languages presently in use or being developed, or opening the borders to that area; for an intermediate language with the scope of an UNCOL[22] but on a higher level or as a subset of a universal programming language.
(b) Development of universal programming languages.
(c) Development of universal processors.

Universality in this respect is meant to comprise at least the elements of two really different problem areas (not vertical combinations or notations).[22]

Several proposals for the first of these solutions (inclusion of language elements) are already reported. Of these, BEEF[24] and ALGOL-Genius[25] are both designed to combine a programming language for algorithmic with one for commercial procedures. More ambitious in this respect is the NPL-SHARE[26,27] language to combine in addition the elements of real-time and command languages.

It is most noticeable that software systems (languages and processors) developed upwards from the machines by combinatinn of existing elements do not tend to please many users. Despite the desirability of larger extended systems, there are always users who do not need the new scope and are unwilling to pay for the clumsiness and complication due to inadequate design.

Other development possibilities going from a fixed base are found in the features of open-ended systems. To some extent at present, the combining of languages of two areas results in at least a partial universal programming language.

## UNIVERSAL PROGRAMMING LANGUAGES

### *Definition*

A universal programming language can be defined as a complete set of elements to describe the problems and solutions in all problem areas. If such a language can be developed, the design requirements will be the same as for a single high-level programming language (see the requirements listed above), but much more valid.

### *Mathematical Definition and Development.*

It is easy to define mathematically the design and development of a universal programming language in general.

The complete set $S_i$ of all equivalent programs* $p_{ik1}$ for the solution of problem k in one area is given by

$$S_i = U_i p_{ik1}$$

Then the operation $\delta$ selects from this set a program, maximal in respect to power of problem description

$$D_k = \delta\, U_i p_{ik1}$$

Now all maximal programs of one problem area form a new complete set $S_k$:

$$S_k = U_k\, \delta\, U_i p_{ik1} \quad .$$

From this new set, operation $\gamma$ extracts the language elements and operations for the given area to form the language for the problem area $G_j$:

$$G_j = \gamma\, U_k\, \delta\, U_i p_{ik1}$$

For the generalized and universal programming language $\Lambda_u$, the complete set $S_1$, generated by $U_1$, of all languages $G_1$ has to be considered, combined and integrated by the operation $\lambda$ to give

$$\Lambda_u = \lambda\, U_1\, \gamma\, U_k\, \delta\, U_i p_{ik1} \quad .$$

As may be recognized, the operations, $\delta$, $\gamma$, and $\lambda$ are very complex and difficult, but the most serious drawback seems to be the large extent of the various sets required. But this is the only way for development, be it by methods of evolution via open-ended languages or by revolution via problem analysis and then language determination (as given by an example in reference 28.

### *Old Proposals*

The problem of proliferation of programming languages was recognized rather early especially in respect to the effects on processor production.[22,29] So UNCOL, a universal computer-oriented language was proposed as an interface between high-level programming languages and computers. Due to the open-endedness on both sides of problems and machines, such a scheme cannot easily be designed on the basis of a fixed language. On the other hand, 30 examples for this scheme are known as notations,

*See "Translation among Programming Languages" above.

e.g., the prefix notation.[30] But this design level seems to be inadequate for a satisfactory solution to the problem.

A similar restriction is imposed on the well-known flow-chart notation to be used as a universal programming language, or even as a programming language. (Recent rumors suggest flow charts to be used on oscilloscope terminals for on-line programming.)

*Design Possibilities*

As mentioned in the section on Solutions, there are two possibilities for the design of universal languages. One is a conventional approach with open-ended languages and processors so that the users will develop gradually the required high-level programming languages in the interesting problem areas. Then from time to time the now achieved status of a system should be re-evaluated and reshaped to avoid and eliminate inefficiencies and obsolescence. So gradually the best language for a problem area will mature. As soon as there are enough languages developed for different problem areas, then the design of a universal one can be envisaged.

The more direct method suggested by the mathematical definition is to investigate the nature of the problems, depict the elements for description and solution, and combine these into a high-level programming language. This method was used to develop the new programming language BEST for commercial applications.[28] The reported five years of labor seem relatively high, but the rewards justify the effort to eliminate all the inadequacies and inconsistencies which arrive at the fire-line with programming languages, designed by mutual conconsent at conference tables.

## UNIVERSAL PROCESSORS

*General Requirements and Notation*

*Definition and Feasibility.* A universal processor can be defined as a program for transformation of all programs from all required problem areas into all required target languages. The extent of such a processor is dependent on the definition of the requirements of the problems and of the machines.

Processors which accept programs in a number of different programming languages are well known.[31] But no successful experience (aside from the projects outlined below could be found for easy combining of different target languages. This is certainly no accident, as will be stressed later. The target language area poses heavier and more stringent requirements on processors than the source language area where it is possible to easily combine several compilers for different languages (but for the same machine) into one system and to invoke the one momentarily required by control cards (e.g., in IBSYS.[31] The difficulties for the target language arise mainly because of a third language parameter in a processor, its own language, i.e., the one in which the processor is written or the language of the machine on which the processor has to run.

*Design and Implementation.* At the source language side of a processor, besides the simple IBSYS concept, a higher degree of integration could be obtained by (1) accepting mixtures of statements of different languages, perhaps with indicators as to which languages they belong; and (2) accepting the elements of different languages intermixed. This requires that incompatibilities among the languages are removed. (For example, the definitions of identifiers in FORTRAN and COBOL are incompatible, with blanks having no meaning in FORTRAN, but used as separators in COBOL.) So it is proven that a fairly universal programming language cannot be developed by simply combining useful features from different other languages.

If only a restricted universal processor can be developed, then by feeding a copy of it to itself a desired less-restricted one could be produced automatically.

*General Notation.* A processor can be defined as a function ($f$) for transformation of a program given in one language into that of another. The parameters of this function $f$ are then:

$\alpha$) Source language of programs to the processor;
$\beta$) Target language of programs from the processor;
$\gamma$) Own language of the processor;
$\delta$) Variables for measuring the efficiencies;
$\epsilon$) A variable for the method used in the processor;
etc.

So the processor can be designated by $f(\alpha,\beta,\gamma,\delta,\epsilon, \ldots)$.

*Transformation of Programs by Processors.* A source program is, for example, a given set $V_{1i}$ of ($i$) statements ($s_i$) in source language $A$ for the solution of problem 1 and similarly a target program can be defined:

$P_1 = V_{1i}s_i(A)$ as source program
$P'_1 = V_{1k}s_k(B)$ as target program in language B

The application of the transformation function gives the relations:

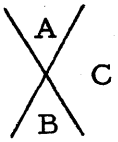
$V_{1k}s_k(B) = fV_{1i}s_i(A)$ for languages separate translatable only on the program level
$= V_{1i}gs_i(A)$ requires a different transformation algorithm
$= V_{1j}fV_{jm}s_m(A)$ for languages separate translatable on block level; (a block is defined as the set $V_ms_m(A)$
$= V_{1i}fs_i(A)$ for languages separate translatable on the statement level
$= V_{1i}s_i(fA)$ for languages separate translatable on the language level

*Simplified Notation.* The most interesting and important questions with processors are concerned with the function of changing the language representation of programs, (especially by translating them to actual machine language). Therefore, if no regard is given to other than the language parameters, the function is reduced to

$$f = f(A,B,C).$$

Of course, the other parameters cannot be completely ignored, but they depend on other variables. (Measured efficiency of a processor depends on the methods used, while efficiency requirements are functions of hardware speeds and costs again, etc., so other parameters are omitted here.)

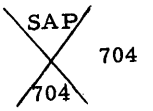
Now a new symbol for a processor is introduced:



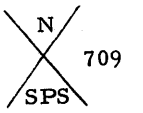

It designates a processor translating from source Language $A$ into target language $B$ and is itself written in (its own) language $C$. Sometimes a label as a name for a processor will be used and inserted into the empty space at the left side of the symbol. Where one language parameter in the following is not specified or pertinent, the space for it is left empty.

*Examples of the New Symbol*

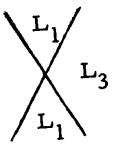

is a FORTRAN compiler written in SAP, translating from FORTRAN to SAP




is a SAP assembler given in 704 machine language and translating from SAP to 704 machine language



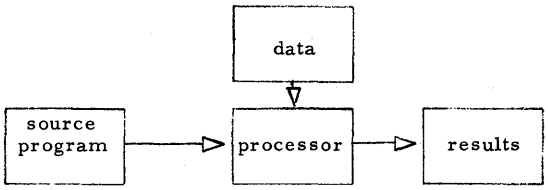

is a NELIAC compiler translating from NELIAC to 1401 SPS and running on the 709



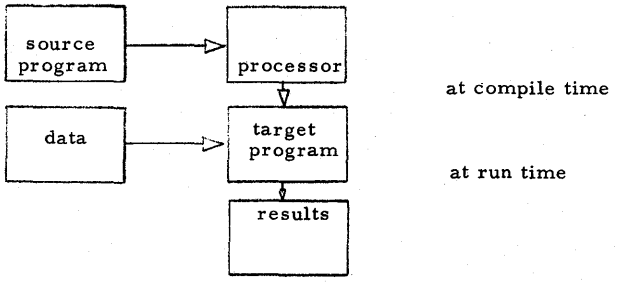

is a precompiler translating into the source language (e.g., for error checking in programs) and running on machine with language $L3$.

*Mode of Processor Use.* Basically two different modes of processor use can be distinguished: translative and interpretive.

1. Interpretive Mode. The interpretive mode of processor use is characterized by the handling of data and source statements at the same time, according to the diagram:



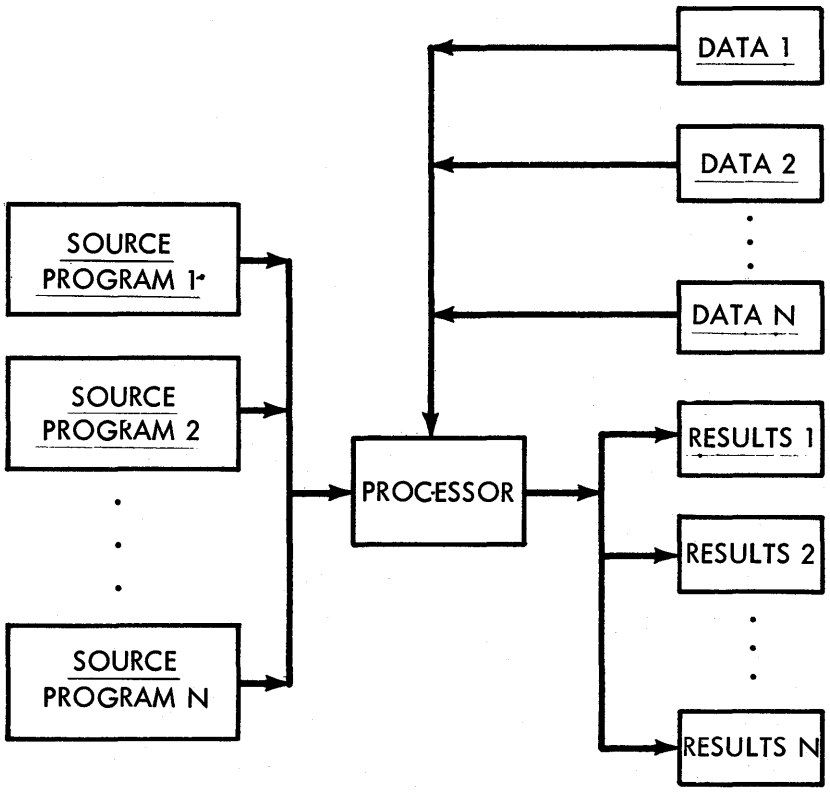

2. Translative Mode. The translative mode is characterized by the processing of source program and data at different times, at compile time and at run time, respectively:



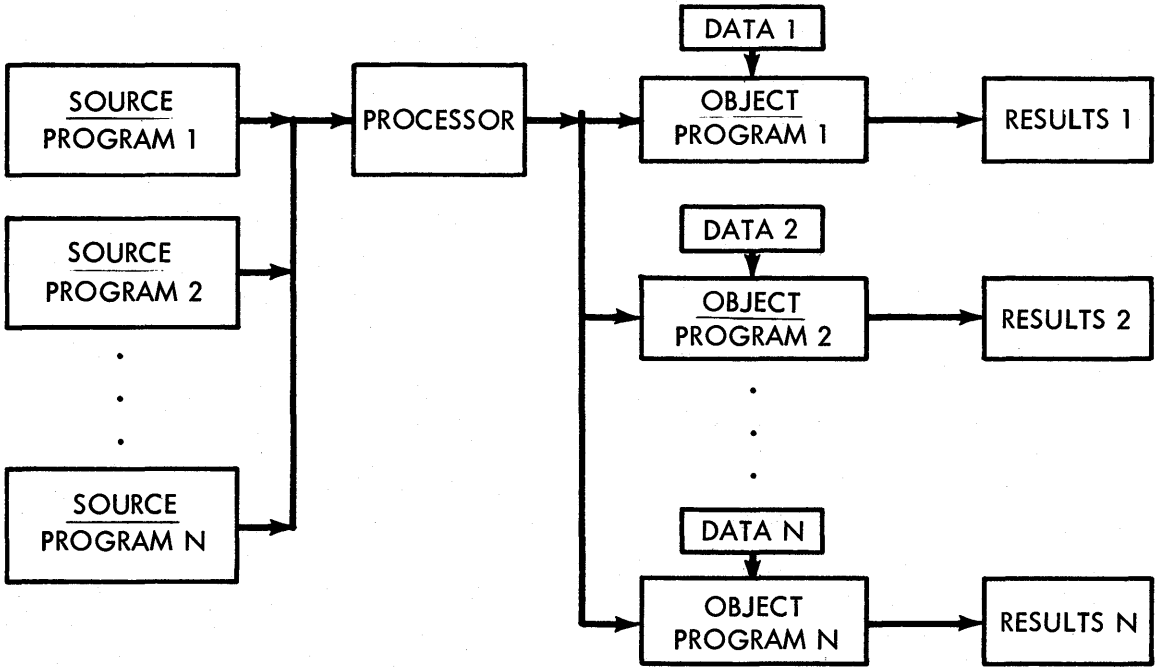
It must be understood that the execution of the target program at run time is itself considered again as interpretation.

In real-time concurrent processing, the schemes would look like

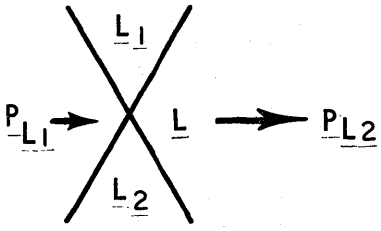
3. For Real-Time Interpretive:




4. For Real-Time Translative:

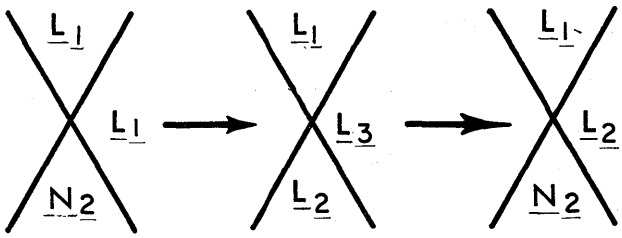


*General Use of Processors.* The general use of processors is given by feeding (designated by the simple arrow →) a program into the processor to receive (designated by the double arrow ➤) the program in another representation:



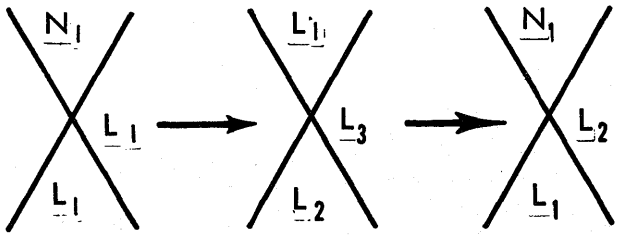

is the translation process of a program from source language $L_1$ to target language $L_2$ by a processor running on a machine with language $L_3$.

A more interesting case is that the program fed to the processor can itself be a processor. When it is written in its own-source language it is according to:




Here it is explained that a processor written in its source language can be translated to any other language for which a processor exists. From this prospect was derived the old requirement that each processor should be written in its source language. On the same process is based nowadays the production of assemblers for new machines. Details on that method will be explained later.

When the processor is written in its own-target language, this gives:




This is the ancient method of processor construction by writing it in its target language. So it is possible to build up on already available processors. An example of this is the old FORTRAN compiler written in SAP and translating to SAP, which is then translated by the SAP assembler into 704 machine language, but it needed the SAP assembler in the translating process from FORTRAN to 704 code.

*Restrictions on the Parameters.* The variables in the transformation function $f\ (A,B,C)$ of a processor are certainly not independent even among themselves. The following functional relations among the language parameters are interesting. Previous mention has been made of the relation between the target and the own language of a processor. Another, but not a very stringent one, governs the relation between source language elements and their target language equivalents.

It will now be assumed that the relations can be defined and the variables separated. Several cases are then distinguished:

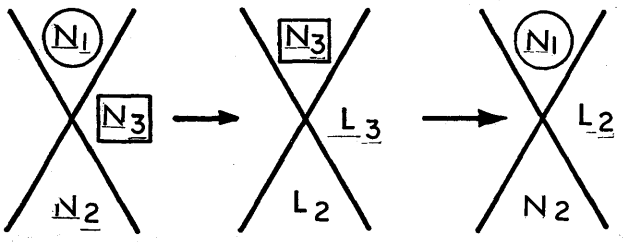
1. The source language parameter A is independent of the other ones, so that no functional relation is given there:
   $A \neq h_1(B)$; $A \neq h_2(C)$.
2. The target language variable $B$ depends not on the source or on the own language:
   $B \neq h_3(A)$; $B \neq h_4(C)$.
3. Both source and target language are not related to the own language (but might depend on each other):
   $A \neq h_2(C)$; $B \neq h_4(C)$.
4. All language parameters are independent among themselves.

The design of universal processors will now be investigated according to these restrictions.

*Universal Processors.* Universal Processors can be designed under the restrictions of the previous paragraph and will be treated in the same order.

1. A scheme for a universal processor limited by restriction (1) could be derived as follows. If the processor is not dependent with the source language either on target or on the own language, then the source language part could be made exchangeable. As soon as one processor with this characteristic would be available, processors for all different source languages could be constructed running and

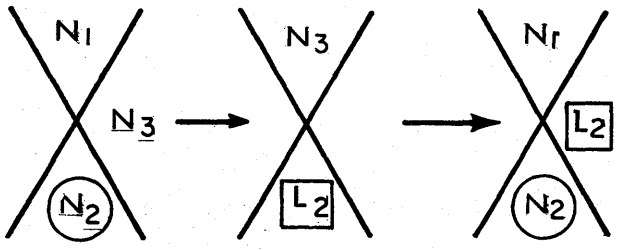
translating for the same machines. By transforming another processor with the same characteristic according to:




processors could be written in all languages for which exchangeable definitions exist, and then translated to the designated machines. The task of writing $2m \times n$ processors for $n$ languages and $m$ machines (there are only $m \times n$ processors if the possibility of translation of programs on one machine for running on another machine is excluded) is now reduced to the writing of $2m$ processors or $m$, respectively) for the $m$ machines and of $n$ language descriptions for the $n$ source languages.

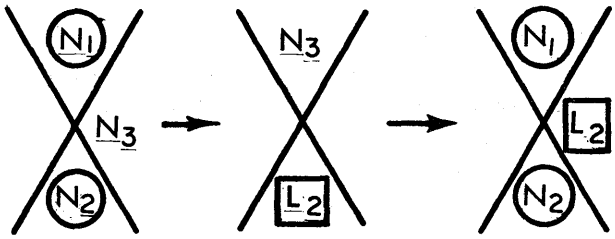
2. The case where the target language is considered independent of source and own language is even more interesting. Then target language descriptions for the machine could be developed and inserted into the processor to give a scheme for processors to translate for all machines.

Applying the same principle to the translation of processors could give a universal processor with any desired target and own language requirements:




The requirements for a universal processor system would now be to write $n$ processors for $n$ source languages and $m$ target language definitions for $m$ machines. These $n$ processors would be written preferably in a high-level language ($N_3$) for which a processor with the same characteristics for exchangeable target equivalents has been given already.

3. The case that source and target language are independent from the own processor language (although they may depend on each other, case 1) would give a very powerful and general system. By the application of the scheme to itself, any desired own language and so a rather general universal processor scheme could be obtained:




The implementation requirements would now be to develop: one processor with removable source and target language equivalent parts in two copies, and the definitions for each pair of source-target languages, giving $m$ x $n$ definitions if they are dependent on each other (case 1) or $m + n$ definitions if they are independent (case 2).

4. When all language parameters are independent, then we have the most general universal processor scheme. Of course, this brings not more solutions than could already be obtained in case 2. The requirements here would be to have one processor with the desired characteristics and $m + n$ descriptions of source, target, and own languages.

*Discussion.* The schemes for universal processors described in the preceding section are outlined on the assumption that the language parameters of processors are independent of other variables and among themselves, at least to a certain degree. Some relationships among source, target and own language are known. But up to now it was never proved or disproved that perhaps they could be separated, and if so, under what conditions. It can be seen, for example, that between source and target language only a simple connective relationship exists, but the requirements then imposed on the own language were not yet evaluated.

The area of source languages is now fairly well understood, although the techniques are still not in the best conceivable state; much work is left to be done; some is going on and progressing satisfactorily. But knowledge of the others is very insufficient and incomplete.

Many investigations in the past were dedicated to the theory of automata. However, most results from these investigations are too general or of too low a

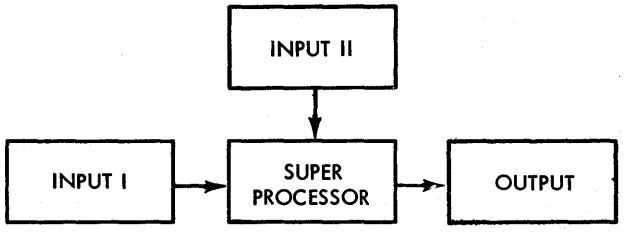
level to be of great value to present-day computers with their variety of special hardware features. Only in the recent past some work was performed on models of more contemporary machines.[29]

As long as actual computers are not well understood there will not be much hope for very successful development of useful universal processors.

The following section describes the various reported projects for automated processor production and compares these to the described scheme of universal processors.

*Projects for Universal Processors*

*General Scheme and Survey.* All literature uncovered in recent years regarding projects for proposals on universal processors fit into the same general scheme. There is always input I, consisting of a processor or its description, or the description of the source language. Input II is sometimes missing (in some cases of a processor description for I), or consists of specifications of the target language, and of a source program in interpretive cases in addition to that.




The different elements for input and the obtained output are summarized in Table 1.

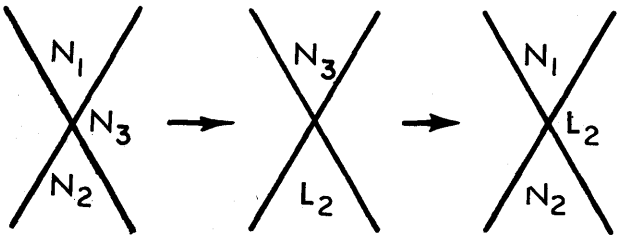
Table 1.

| Project | Input I | Input II | Resulting Processor | Special Features |
|---|---|---|---|---|
| High-level and special language use. | Processor written in high-level or special processor writing language | — | Processor in low-level language | High-level languages applied to processor construction |
| UNCOL | Processor in UNCOL to translate to UNCOL | — | Processor for UNCOL on designed machine | Reduction in number of processors required |
| CLIP-JOVIAL | Processor in high-level language | — | Processor in low-level for original language | "Boot-strapping" |
| NELIAC | Processor in high-level language | — | Same as above | Same as above |
| XTRAN | Processor in high-level language (with connectors?) | Target machine macros | Processor in low-level for designated language | Exchangeability of target language equivalent |
| SLANG | Processor in SLANG — POLMI | Target language description to generate the equivalents | Same as above | Generation of target equivalents from a description |
| TOOL | Processor in TOOL | Library of macros | Same as above | Translation for new machines |
| Syntax method | Language specification in terms of *M* | Source program in *L* | Target program in *M* | Interpretive processor accepting language *L* specification |
| TGS | 1. Language specification *L* | 1. Macros for *M* | Same as above | Interpretive processor with extensive descriptions and specifications |
| | 2. Generation statement tables for selection | 2. Source program | Processor in *M* | |
| Meta A | Description of language *L* in terms of *M* | — | | System written in specification languages |
| Meta B | Description of language *L* with connectors | List of target equivalents (macros) | Same as above | System in specification language separable for given source and target languages |
| Applicative Expressions | Description of *L* in Applicative Expressions | Machine definition in Applicative Expressions | Same as above | Same as above with Applicative Expressions as specification language |

Two different approaches can be distinguished, one starting with a processor or the description of a translation process and the other starting with definitions for the source language. The processor-based projects are generally the older ones, thus reflecting the progress in the field.

*Processor Based Projects*

1. *High-Level or Special High-Level Language Use.* To gain the advantages of programming using high-level languages (see Introduction "Experience with High-Level Programming Languages") in the construction of processors, projects based on this were tried rather early and often abandoned immediately. The main reasons were the inadequacies of high-level languages of those days (mainly FOR-

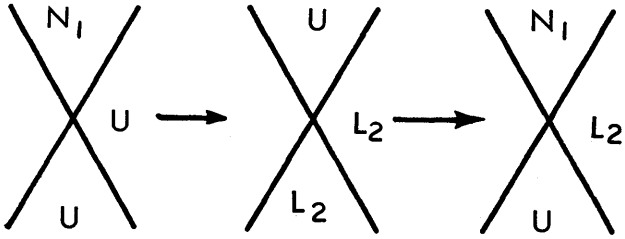



TRAN and ALGOL) for processor descriptions, and unfamiliarity with the new technique. To alleviate the difficulties special high-level languages were developed.[33,34] The scheme here is working like:

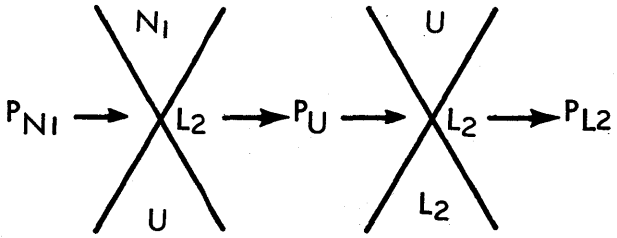
However, the gains by these projects for the construction of universal processors can be considered marginal, because the original number of processors required is not reduced and, in addition to that, one processor for the high-level description language is required for each machine. This scheme is reported only for the sake of completeness and because it is used heavily in other projects.

2. *UNCOL.* In this project the first suggestion for a system of some sort of a universal processor was given.[22,29] It calls for an intermediate language (see "Old Proposals" above) together with the appropriate processors. The requirements are here reduced to $m + n$ processors for $n$ languages and $m$ machines, instead of $m \times n$ (without translation of programs to run on other machines). For each source language a processor has to be written in UNCOL translating into UNCOL and then for each machine one translating into machine language.

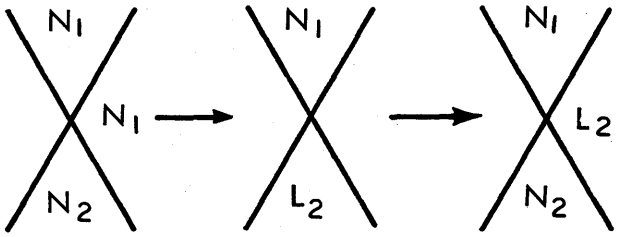
In the production process the processor (written in UNCOL) for the source language is translated by a processor from UNCOL to machine language:




All programs then written in source language $N_1$ are translated by this new processor, running on machine with language $L_2$, into programs in UNCOL. These programs are then finally translated to machine language $L_2$ by the translator from UNCOL to machine language $L_2$ (already required above):



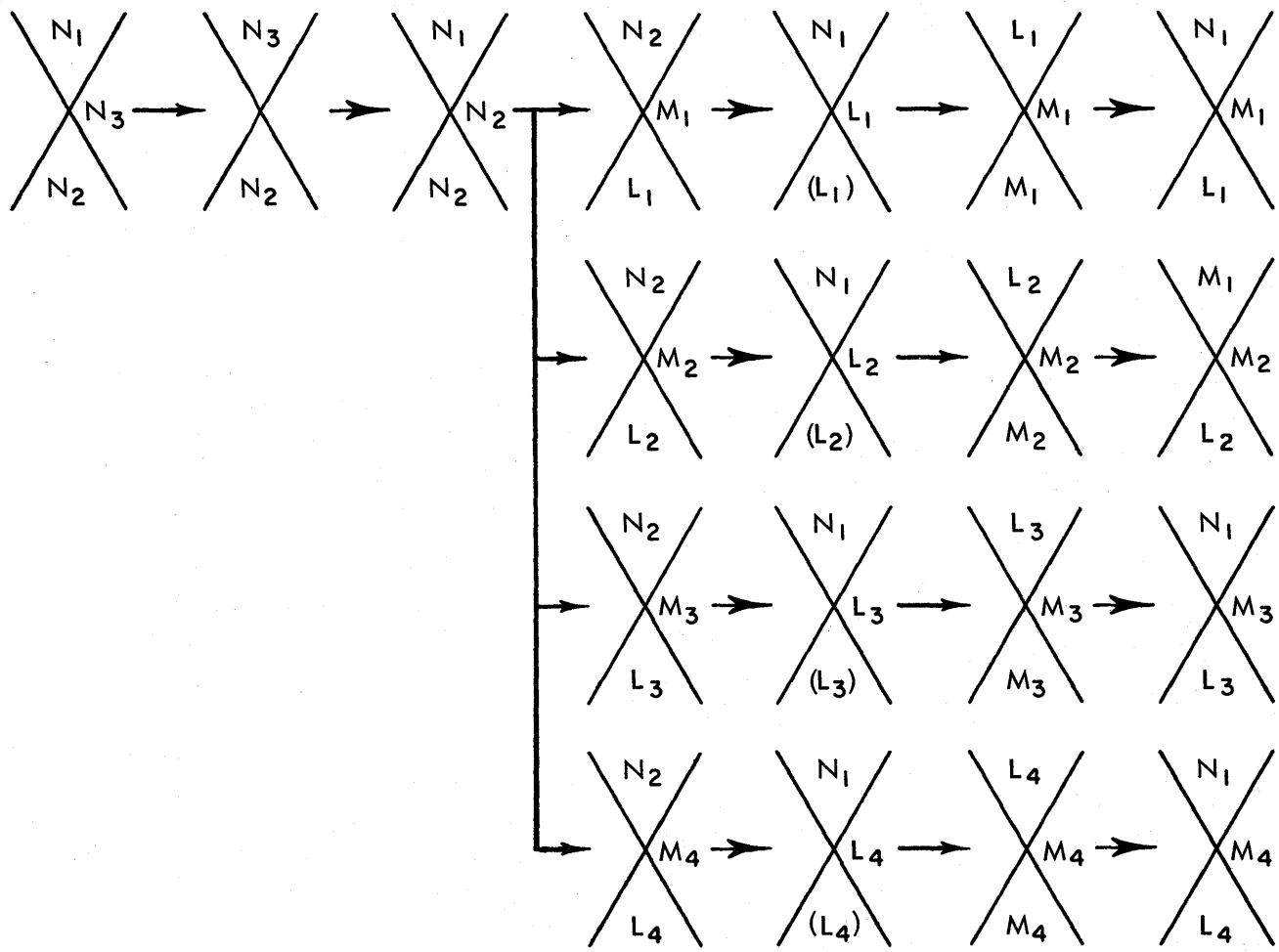

3. *CLIP-JOVIAL.* Very similar to both the UNCOL and high-level language project is basically the CLIP-JOVIAL approach. Several different versions are reported, one without intermediate language and another, more advanced, with it.[35] The diagram for the simpler version looks like:




Practically, the high-level language scheme where the source language is used for description with:

| | |
|---|---|
| $N_1$ | the CLIP language (a dialect of ALGOL 58 with additional features for table packing, string handling, storage overlapping, and local and global declarations) |
| $N_2$ | assembly language |
| $L_2$ | 709 machine language |

The more advanced version uses an interesting "bootstrapping" method for adapting the processors to different machines:




The parameters are given according to the table:

| $N_1$ | $N_2$ | $N_3$ | $L_1$ | $L_2$ | $L_3$ | $L_4$ | $M_1$ | $M_2$ |
|---|---|---|---|---|---|---|---|---|
| JOVIAL | INTERMEDIATE | CLIP | 709-A | 2000-A | ANFSQ-A | MILITARY-A | 709 | 2000 |

| $M_3$ | $M_4$ |
|---|---|
| ANFSQ | MILITARY |

with the indication of *-A* after a computer name standing for assembly language for that machine, and the computer name alone standing for its machine language. The parameters ($L_i$) enclosed in parentheses indicate the insertion of the appropriate target language equivalents for the intermediate language $N_2$ and the patching up for it.

A processor is written in CLIP to translate from source to intermediate language and is itself translated by the CLIP processor into intermediate language. For each machine, a processor is now written for translation from intermediate to assembly language of that machine. With these processors, the former processor is translated into assembly lan-

guage, and the target equivalent in intermediate language is exchanged for the one in assembly language. At last, the resulting processors are translated by the assemblers to the appropriate machines.

A universal processor scheme requires:

- One processor for each source language written in CLIP and translating to the intermediate language;
- One processor for each machine to translate from intermediate language to assembly language, the target equivalents for the intermediate language for patching up in the insertion;
- One CLIP processor for the intermediate language (and the assemblers for the different machines).

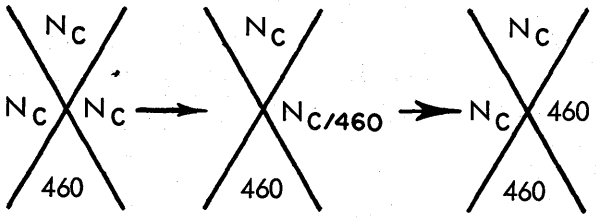
The main difficulty here is to design an intermediate language in a fixed form for many source languages (e.g., the UNCOL concept, see Section IIC).

4. *NELIAC.* In NELIAC likewise the high-level language is used for the programming of the processors.[36] The most interesting feature here is the bootstrapping scheme to obtain the processors for different machines.[37] In the original version on the U460 (Countess), about 20 percent of the processor was handwritten and in machine language inserted into the processor (indicated by $N_{C/460}$), after completion of writing the processor in its source language.

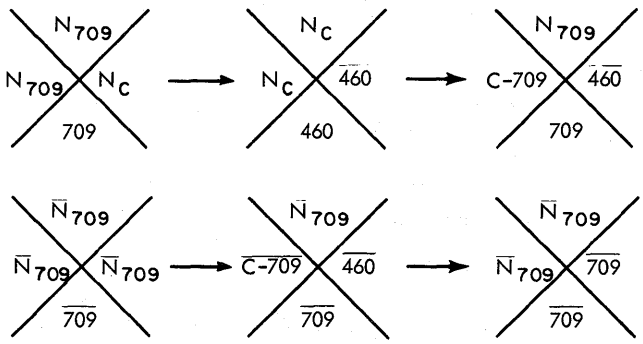
In the notation, symbols for the original names are retained as follows:

| $N_C$ | $N_{709}$ |
|---|---|
| NELIAC for the Countess | NELIAC for the 709 |

The NELIAC-Countess processor was produced with the patched-up processor:



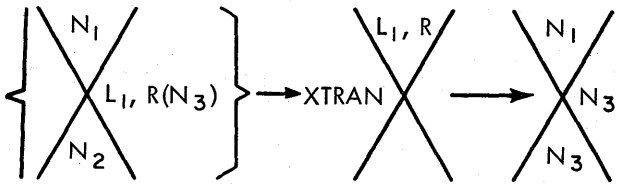

This version was used in the production of the processors for the B200, the CDC 1604, IBM 704 and the 709 machines according to the diagram for the 709:




For each machine, two different versions of the processors have to be written, one in the NELIAC language for the Countess and the other in the NELIAC language for the desired machine.

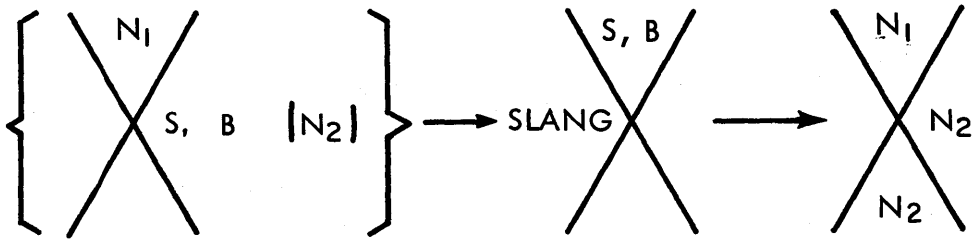
The procedure is to write a processor for a source language to a target language in a high-level language for which there is a compiler running on a machine. This processor is first translated to run on that machine. Then the processor is written in its source language and translated to its proper machine by the one already obtained. This process is the direct equivalent of the old assembler production method, which has been writing the assembler for a new machine first in an assembly language of a running machine and then translating it to run on this machine. The assembler was then again written, but in its source language and translated by the already obtained assembler to run on and translate for its proper machine.

5. *XTRAN.* To adapt the processors for different source languages, the XTRAN system[21] accepts those written in a simplified version of ALGOL '58 with string handling facilities (XTRAN language) and the set of the macros for the particular machine. Two different sets of macros are used, one is machine-independent of the three-address type (as an intermediate language) and the other consists of the macros of an actual machine in assembly or machine language. The XTRAN system translates the processor for a source language into the machine-independent macros, accepts the definition of these macros in terms of the machine-dependent ones, and then replaces the former with the latter:



The requirements for a scheme of universal processors are:

- One processor for each source language written in XTRAN ($L_1$)
- One set of machine-dependent macros for each machine
- The XTRAN system




Of course, the building up of the target equivalents from a machine description is usually a very difficult task, if in general possible at all. And the POLMI language might not be definable in a fixed set. Therefore, it is not surprising that no further experience with this system is yet reported in the literature.

The requirements for a universal processor scheme using this system are:

- One processor written in SLANG-POLMI for each source language.
- One description for each target machine.

In this case, the universal processor scheme would require:

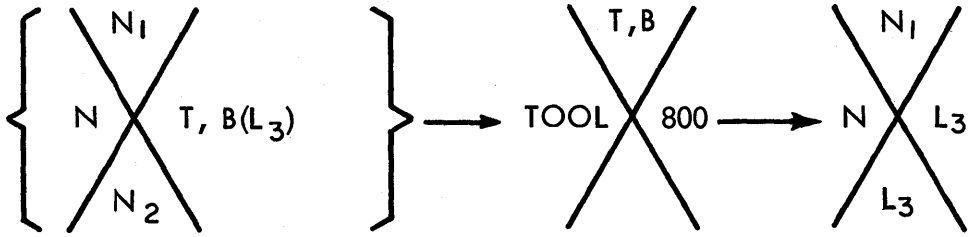
- One processor written in TOOL for each source language.
- One library of target equivalents for each machine (presumably with appropriate connectors).

As can be seen, this scheme is very similar to the

Unfortunately no experience is published yet for this system. This might be due to unsatisfactory performance for the macro setup.[38]

6. *SLANG.* The SLANG system[39] is very similar to XTRAN, but is more ambitious to generate by itself the macros from a definition of the target machine. The SLANG compiler accepts in addition to that a description of the processor in SLANG-POLMI, designated by $S$ :

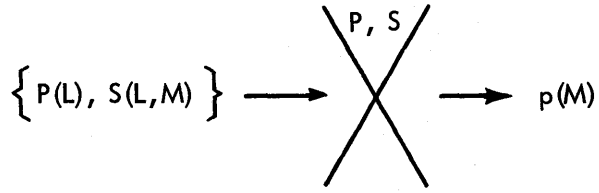
7. *TOOL.* A peculiar system was reported in TOOL.[40] It translates processors written in the TOOL language for other machines. The target equivalents for a new machine are extracted from a library file. So the automated translation of processors given in TOOL designated by $T$, to different machines is handled. Generally, processor notation gives:




method from SLANG. Without further details this scheme was reported to be working satisfactorily and to be running on the H800 and H400.

*Description Based Projects*

*The Syntax-Directed Method.* The reported projects use a syntactic description of the source language[41,42] and are compiling interpretively:



The reason for the requirement of interpretive mode here lies in the fact that the different language parameters of a processor are interwoven, but already to a much lower degree than in straightforward processors.

The requirements for a universal processor scheme here are that the syntax description be separable from the super-processor and that one description be developed for each source-target language pair.

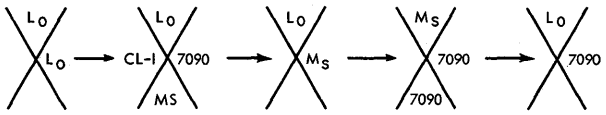
*TGS — The Translator-Generator-System.* An ensuing development to the syntax-directed method is given by the translator-generator-system TGS,[43,44] using macro concepts like the XTRAN project (see paragraph on XTRAN above).

This scheme accepts as input besides the program to be translated:

1. A sort of Backus-Naur-Form definition of the source language.
2. A table for macro description and code selection for the target language.
3. The generation strategy tables for the description of the linkage between source and target definitions.

The super compiler consists of five parts working subsequently on the source program. Most interesting among them are:

1. A syntactic analyzer for the source program to convert some piece of input string into an internal representation (a tree form is used);



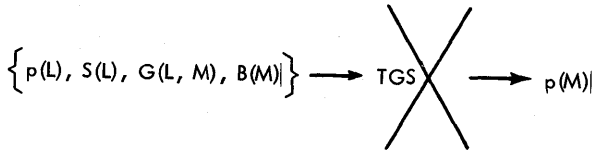

A universal processor scheme based on this project would require:

- One BNF definition for each source language
- One table for macro description and code selection for each target language
- One generation strategy table for each source-target language pair

*Metalanguage Compiler Direct.* Likewise derived from the syntax directed method[46] this project uses a metalanguage description of a source language in terms of the semantic target equivalents as basic elements of the language on the problem side.[47] This description is then compiled by the metalanguage compiler into a processor written in the target equivalents of the metalanguage compiler:

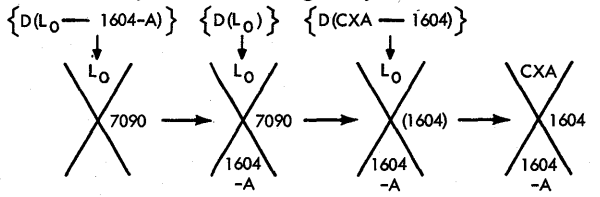
2. A generator phase translating the internal representation into an *n*-address instruction form, depending on syntactic context;
3. An optimizer phase for source and target program optimization, eliminating invariant computations out of loops and common subexpressions (thus being source-language-dependent to a certain degree) and assigning special registers (thus being machine-dependent to a certain degree).
4. A code-selector phase driven by the code-selector table to produce symbolic machine code.

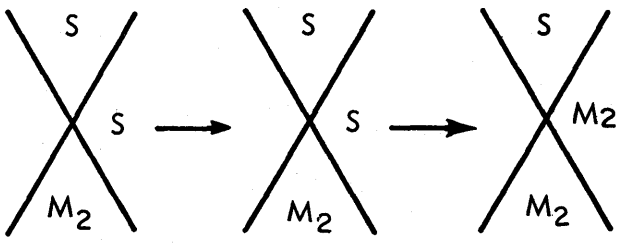
The translation process looks like this:



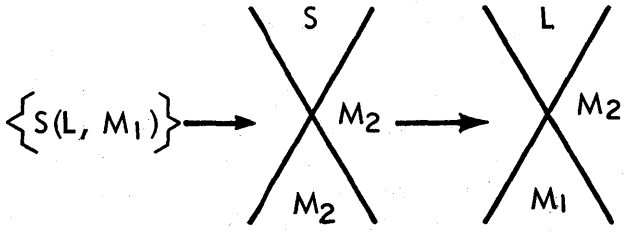

Most important is the endeavor to achieve an object code optimized to a rather high degree at the cost of great difficulty in the description of the code selection. In addition to that, the algorithms seem to be source- and target-language-dependent (in respect to algorithmic languages containing expressions and loops, and to machines possessing special registers, both in a given form).

For the production of the system, a bootstrap technique is used, starting from the algebraic language $L_0$ (the language of the *CL*-I system[45] in which the system was originally written:

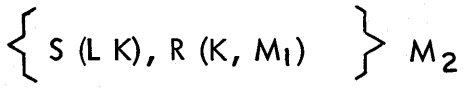


The metalanguage compiler was originally written in its augmented specification language and compiled by itself:



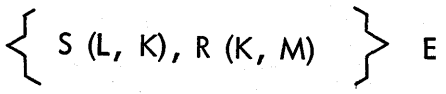

Since no machine was available to accept the metalanguage specification, this translation was done initially by hand. The target equivalent for $M_1$ and $M_2$ are normally rather far different from operations and elements found on actual computers. Therefore, they have to be interpreted in terms of those for execution.




A universal processor scheme requires here:

- One description for each source language
- One set of target equivalents for each machine plus the target equivalents of new basic semantic elements in a new source language
- The metalanguage compiler

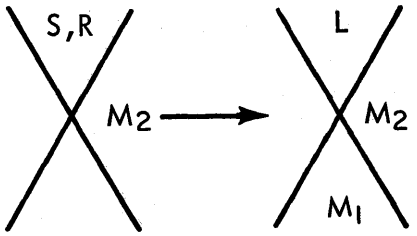
One special aspect of this method has to be stressed. By the design process of a source language in terms of the basic semantic elements, these elements can be separated in a form in which they are




Requirements for a universal processor scheme are here:

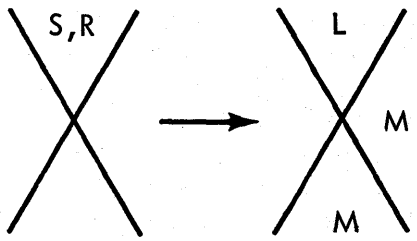
- One metalinguistic description for each source language
- One set of target equivalents for each machine and set of basic semantic elements of each source language
- The metalanguage compiler

*Metalanguage Compiler Indirect.* Based on the previous project a method more machine-independent can be proposed, using the macro principle. Here the basic semantic elements of the source language would be separated from the source language description and referred to by connectors. These are then inserted or executed to obtain translative or interpretive mode. The production of a processor would be accomplished according to:




required for the development of a universal programming language (see "Mathematical Definition and Development" and "Design Possibilities" above).

*Applicative Expressions.* Another proposal for a universal processor project could use Applicative Expressions[48] and would be very similar to the method described above. Both source and target language would be described in Applicative Expressions with connectors used for the correct interplay. Although this scheme might be more general, it seems to introduce many redundancies and to complicate the description, as the examples in reference 48 prove. The process would be:



The requirements for universal processors in this instance are similar to those in the preceding paragraph.

*Discussion*

All reported projects try to gain some power for the construction of processors in the direction of universal processors. There are basically three different starting points:

1. The own language for the processor
2. The source language of the processor
3. The target language of the processor

The first point is stressed by all methods to alleviate the specifications of the processor to various degrees from the use of a high-level language for explicit writing the processor to the syntax table specification in TGS.

The complete definition of a source language can specify at the same time a recognizer for programs written in that language. This characteristic is used in the description-based projects.

Techniques for the target language specification to use a processor on different computers were attempted rather early as they were important to the development of software in the variety of different computer designs. But, as far as can be seen, the obtained results are still very far from a satisfactory solution to the problem—if it is possible to find even a fairly general solution. Unfortunately no detailed experience with the XTRAN and TOOL projects is reported.

Several interesting methods are used for bootstrapping, i.e., the adaption of a processor to a special computer mechanically. They range from the old assembler construction method used in NELIAC to rather elaborate and sophisticated ones, as in TGS. Of course, the whole subject needs much more effort to develop the techniques for a fairly general universal processor scheme or to prove that the plan is not possible and to state the conditions and new insights into the problems will certainly bring much more progress than was achieved in the past. This paper is intended to serve as a basis for such a development.

## Appendix

*Explanation of Symbols*

| | |
|---|---|
| $A$ | for designation of the source language of a processor |
| $B$ | for designation of the target language of a processor |
| $C$ | for designation of the own langauge of a processor |
| $B(M)$ | description of a target machine |
| $D(L)$ | description of the language $L$ |
| $E$ | designates Applicative Expressions |
| $f(\alpha,\beta,\gamma,\delta,\epsilon,\ldots)$ | processor function |
| $f(A,B,C)$ | processor function with regard to the language parameters |
| $G(L,M)$ | connective relation |
| $K$ | connectors |
| $L_i$ | with $i$ as a number designating a language parameter |
| $M$ | macros |
| $N_i$ | similar to $L_i$ |
| $P(L)$ | program in $L$ |
| $R$ | a list |
| $S(L,M)$ | language specification of syntax type for source language $L$ and target language $M$ |
| $U$ | as language designator for UNCOL |
| $U_k,V_k$ | as a set operator in respect to $k$ |
| $X$ | a processor |
| A<br>D C<br>B | a processor with source language $A$, target language $B$, own language $C$ and name $D$ |
| { } | the braces are used to combine some input other than a single processor or program |
| $\rightarrow$ | simple arrow for designating the |
| ➧ | feed-in to a processor |
| | double arrow for designating the output from a processor |

REFERENCES

1. R. J. Slutz, "Engineering Experience with the SEAC," *Proc. EJCC 1951,* pp. 90-93.

2. A. C. D. Haley, "The KDF9 Computer System," *Proc. FJCC 1962,* pp. 108-120.

3. "Burroughs B5000," in *Data Processing Encyclopedia,* Detroit, 1961, pp. 50-55.

4. K. Samuelson, "Programming Languages and their Processors," *Proc. IFIP Congr. 1962,* Munich, pp. 487-492.

5. See for example M. H. Hall "A Method of Comparing the Time Requirements of Sorting Methods," *Comm. ACM,* vol. 5, pp. 259-263 (May 1963).

6. See for example F. P. Brooks Jr., "The Future of Computer Architecture," *Proc. IFIP Congr. 1965,* New York, pp. 87-91.

7. R. S. Barton, "A Critical Review of the State of the Programming Art," *Proc. SJCC 1963,* pp. 169-177.

8. A. P. Yershov, ALPHA—An Automatic Programming System of High Efficiency, *IFIP Congr. 1965,* New York.

9. J. A. Ward, "The Need for Faster Computers," *Proc. Pacif. Comp. Conf. 1963,* pp. 1-4.

10. R. F. Clippinger, "Programming Implications of Hardware Trends," *Proc. IFIP Congr. 1965,* New York, pp. 207-212.

11. S. Rosen, "Programming Systems and Languages," *Proc. SJCC 1964,* Washington, D.C. pp. 1-15.

12. R. D. Brennan and R. V. Linebarger, "A Survey of Digital-Analog Simulator Programs," *Simul.* vol. 3, pp. 22-36 (Dec. 1964).

13. H. E. Peterson et al, "MIDAS—How It Works and How It's Worked," *Proc. FJCC 1964,* pp. 313-324.

14. H. Schorr, "Analytic Differentiation Using a Syntax Directed Compiler," *Comp. J,* vol. 7, pp. 290-298 (Jan. 1965).

15. T. J. Theodoroff and J. T. Olsztyn, "DYANA, Dynamic Analyzer-Programmer I & II," *Proc. EJCC* 1958, pp. 144-151.

16. J. W. Young Jr., "Non-Procedural Languages," *7th Ann. Tech. Symp., Southern Calif. Chapter, ACM,* Mar. 1965.

17. A. Opler et al, "Automatic Translation of Programs from One Computer to Another," *Proc. IFIP Congr. Munich 1962,* pp. 245-247.

18. S. Gorn, "Specification Languages for Mechanical Languages and Their Processors, a Baker's Dozen," *Comm. ACM* vol. 4, pp. 532-542 (Dec. 1961).

19. J. W. Backus et al, "The FORTRAN Automatic Coding System," *Proc. WJCC* 1957, pp. 188-198.

20. H. Bromberg, "Surveys of Computer Language Use," *Data Proc. Mag.* Apr. 1965, p. 37.

21. R. W. Bemer, "Survey of Modern Programming Techniques," *Comp. Bull.,* Mar. 1961, pp. 127-135.

22. T. B. Steel, "A First Version of UNCOL," *Proc. WJCC,* 1961, p. 371.

23. See for example K. Iverson, "Recent Applications of a Universal Programming Language," *IFIP Congr.* 1965, New York, and *IBM Syst. Journ.,* vol. 2. pp. 117-128 (June 1963).

24. N. Moraff, "Business and Engineering Enriched FORTRAN (BEEF)," *Proc. 19th ACM Conf.* 1964, Phila. DI. 4.

25. B. Langefors, "ALGOL-Genius, A Programming Language for General Data Processing," *BIT,* vol. 4, no. 3, pp. 162-176 (1964).

26. *NPL Technical Report,* IBM Publications No. 320-0908, Poughkeepsie, New York, Dec. 1964.

27. *Computer Review* vol. 6, no. 2, ref. 7275, p. 108-112 (Mar.-Apr. 1965).

28. J. R. Ziegler, "Computer-Generated Coding (BEST)," *Datamat.,* Oct. 1964, pp. 59-61.

29. H. Bratman, "An Alternate Form of the UNCOL Diagram," *Comm. ACM,* vol. 4, p. 142 (Mar. 1961).

30. See for example C. L. Hamblin, "Translation to and from Polish Notation," *Comp. J.,* vol. 5, pp. 210-213 (Oct. 1962).

31. A. S. Noble and R. B. Talmadge, "Design of an Integrated Programming and Operating System, I & II," *IBM Syst. Journ.* vol. 2, pp. 152-181 (June 1963).

32. See for example C. C. Elgot and A. Robinson, "Random Access Stored Program Machines," *Comp. J.,* vol. 11, pp. 365-399 (Oct. 1964).

33. J. V. Garwick, "Gargoyle, a Language for Compiler Writing," *Comm. ACM,* vol. 7, pp. 16-20 (Jan. 1964).

34. C. A. R. Hoare, "A Programming Language for Processor Construction," *IFIP Congr.* 1965, New York.

35. D. Englund and E. Clark, "The CLIP-translator," *Comm. ACM,* vol. 4, pp. 19-22 (Jan. 1961).

36. J. B. Watt and W. H. Wattenburg, "A NELIAC-generated 7090-1401 Compiler," *Comm. ACM,* vol. 5, pp. 101-102 (Feb. 1962).

37. M. H. Halstead, *Machine Independent Computer Programming,* Spartan Books, Washington, D.C., 1962, p. 37 ff.

38. See for example G. Letellier, "A Dynamic Macro Generator for Optimum Use of Machine Facilities by a Translated Program," *IFIP Congr.* 1965, New York.

39. R. A. Sibley, "The SLANG-system," *Comm. ACM,* vol. 4, pp. 75-84 (Jan. 1961).

40. A. Opler, "TOOL, A Processor Construction Language," *Proc. IFIP Congr.* 1961, Munich, p. 513.

41. E. T. Irons, "A Syntax-Directed Compiler for ALGOL 60," *Comm. ACM,* vol. 4, pp. 51-55 (Jan. 1961).

42. T. E. Cheatham and K. Sattley, "Syntax Directed Compiling," *Proc. SJCC* 1964, Washington, D.C., pp. 31-57.

43. S. Warshall and R. M. Shapiro, "A General Table-Driven Compiler," *Proc. SJCC.* 1964, Washington, D.C. pp. 59-65.

44. T. E. Cheatham, "The TGS-II Translator-Generator System," *IFIP Congr.* 1965, New York.

45. T. E. Cheatham et al, "CL-I, an Environment for a Compiler," *Comm. ACM,* vol. 4, pp. 23-27 (Jan. 1961).

46. A Glennie, "On the Syntax Machine and the Construction of a Universal Compiler," Carnegie Tech. Rep. No. 2 (AD-240512), July 1960.

47. D. V. Schorre, "A Syntax Oriented Compiler Writing Language," *Proc. 19th ACM Conf.* 1964, Phila., D1. 3.

48. W. H. Burge, "The Evaluation, Classification and Interpretation of Expressions," *Proc. 19th ACM Conf.* 1964, Phila., A1.4.

# DIGITAL SIMULATION LANGUAGES: A CRITIQUE AND A GUIDE

John J. Clancy and Mark S. Fineberg
*McDonnell Aircraft Corporation*
*St. Louis, Missouri*

## FOREWORD

The field of digital simulation language, although barely ten years old, has shown a remarkable growth and vigor. The very number and diversity of languages suggests that the field suffers from a lack of perspective and direction.

While claiming no expertise in the writing of sophisticated compilers, the authors believe a relative unconcern with implementation details permits a wider objectivity in matters of format and structure. Competence to speak on these aspects is claimed on the basis of extensive analog, hybrid and simulation language experience.

In Locke's words, "everyone must not hope to be . . . the incomparable Mr. Newton . . ., it is ambition enough to be employed as an under-laborer in clearing the ground a little, and removing some of the rubbish that lies in the way to knowledge."[1]

## INTRODUCTION

The appellation "digital simulation language" unfortunately has been appropriated by two quite distinct fields: simulation of discrete, serial processes, as typified by the GPSS and SIMSCRIPT languages; and simulation of parallel, more or less continuous systems, as typified by the MIDAS or DYSAC languages. The scope of this paper is limited to the latter.

This field of what might be called parallel languages has enjoyed a vigorous growth since its inception ten years ago. New languages are appearing at frequent intervals, but it appears the effort is scattered, and perhaps needs to be channelized. The authors have applied themselves to providing a measure of needed direction and perspective.

## BRIEF SURVEY OF THE FIELD

### *History*

Since Selfridge's article appeared in 1955,[2] the field of analog like simulation languages for digital computers has grown at a rapid rate. Brennan and Linebarger[3,4] have provided an excellent review and analysis of the history of the field; their surveys are summarized and somewhat augmented below.

Lesh,[5] apparently inspired by Selfridge's work, produced DEPI (Differential Equation Psuedo Code Interpreter) in 1958. Hurley[6] modified this language for the IBM 704 (DEPI 4), and then in conjunction with Skiles[7] wrote DYSAC (Digitally Simulated Analog Computer). This line of development has continued at the Universities of Wisconsin and Colorado under Professors Skiles and Ride-

out, resulting in the BLOC languages (MADBLOC, HYBLOC, FORBLOC and COBLOC).[8,9]

Stein and Rose, in addition to generating ASTRAL,[10] (Analog Schematic Translator to Algebraic Language) in 1958, have provided the theoretical and practical background needed to write a sorting routine, i.e., an algorithm to deduce the proper order of problem statement processing.[11] This feature, although overlooked by many authors, is one of the keys to a useful language. This point is detailed below.

In 1963, Gaskill et al,[12] wrote an excellent simulation language, DAS (Digital Analog Simulator). The program unfortunately suffered from a rudimentary integration algorithm and the lack of sorting.* MIDAS[13] (Modified Integration DAS), written by Sansom et al at Wright-Patterson Air Force Base, supplied the necessary improvements and met with unprecedented success. (Approximately 100 copies of the program have been distributed.) The success of MIDAS is explainable by two facts: the integration routine and sorting feature make it extremely easy to use and MIDAS was written for the widely used IBM 7090-7094. The authors of MIDAS have now offered another entry, MIMIC,[14] which is implemented by a compiler program and provides symbolic labelling, logical control capability, and freedom from the block-oriented programming. MIMIC is not without faults, particularly in the areas of data entry, but seems destined for the same general acceptance as MIDAS—and deservedly so.

One of the most significant developments has been computer manufacturer interest in simulation languages. Scientific Data Systems has led the field in this respect, having proposed DES-1 (Differential Equation Solver) in 1962.[15] DES-1 has been modified extensively in the succeeding years, and now offers one of the best formats and one of the most variegated operator lists;[16] SDS has always promoted the language as part of a total computer system which provides analog type I/O and programming for a digital computer.

In late 1964, IBM entered the field in a small way with PACTOLUS, by R. D. Brennan.[17] Apparently, PACTOLUS was intended to have only modest computational capabilities and to contribute primarily as an experiment in man-machine communication. In this respect, the objectives of PACTOLUS are similar to those of DES-1, although the latter also attempts to be as powerful a computational tool as possible.

PACTOLUS was written for the IBM 1620, and brought simulation to a previously untapped audience of small installations. The popularity of PACTOLUS is rivaled only by that of MIDAS, if indeed it has a rival.

More recently, R. N. Linebarger of IBM has announced DSL/90 (Digital Simulation Language for the 7094 class computer).[18] This language is a significant advance over PACTOLUS as a computational tool and offers many format improvements.

The above are only a few of the languages; many others have appeared. Table 1 shows a list of simulation languages, along with some important characteristics of each. This table is the result of a rather diligent search, but certainly is not comprehensive. Much of the material has been taken from a survey given by Clymer.[19]

### *Trends*

The present trend in the field is towards extension of the power and utility of the programs. More efficient execution has been recognized as a goal, and the compiler approach to implementation has gained increased acceptance. The provision of more complex operators and more advanced algebraic capability represents an effort to increase the utility of the programs for less analog-oriented applications. These trends are in the right direction, but efforts have been scattered, and perhaps need to be channelized.

Another major step has been recognition of the importance of the man-machine interaction. As was mentioned, this has been the primary message carried by the PACTOLUS program, and has long been the concern of the DES-1 designers. The SCADS[20] program has been used on-line at Carnegie Institute of Technology and the Aerospace Corporation is using EASL[21] through a terminal to an IBM 7094. The authors agree wholeheartedly with this stress on man-machine interaction, and are aware of the real import this communication has for the future of digital simulation.

### *Areas of Use*

Simulation languages have been written for two more or less diverse reasons: to provide analog

*It should be noted that Gaskill considers neither point of particular significance.

Table 1. History of digital simulation languages.

| Name | Source of Name | Date | Author(s) | Author's Affiliation | Computer | Integration Routines | Ancestor | Sorting | Remarks |
|---|---|---|---|---|---|---|---|---|---|
| — | —— | 1955 | R. G. Selfridge | USNOTS Inkoyern | IBM 701 | Simpson's Rule | —— | No | The Adam of this genealogy |
| DEPI | Diff. Eq. Psuedo Code Interpreter | 1957 | F. Lesh | Jet Propulsion Lab | Burroughs 204 | 4th Order Runge-Kutta | Selfridge | No | Expanded and improved Selfridge's work |
| DIDAS | DIgital Differential Analyzer Simulator | 1957 | G. R. Slayton | Lockheed-Georgia | IBM 704 | Euler | —— | | Simulates a DDA |
| ASTRAL | Analog Schematic TRanslator to Algebraic Language | 1958 | Stein, Rose and Parker | Convair Astronautics | IBM 704 | Runge-Kutta | —— | Yes | Isaiah, the voice that crieth in the wilderness. A precursor of the modern languages. Sorting and compiler implementation were original with ASTRAL, and remained advanced features until very recently. |
| DEPI-4 | DEPI for the IBM 704 | 1959 | J. R. Hurley | Allis-Chalmers | IBM 704 | 4th order Runge-Kutta | DEPI | No | First language to use float point hardware, and thus eliminate scaling problems. |
| DYANA | DYnamics ANAlyzer | 1959 | T. J. Theodoroff | General Motors Research Lab | IBM 704 | Euler | —— | | Mechanical system dynamic analyzer |
| BLODI | BLOck DIagrammed Compiler | 1961 | Kelly, Lochbaum, and Vyssotsky | Bell Labs | IBM 704 and 7090 | None | —— | No | Block simulator for signal processing devices |
| DYSAC | Digitally Simulated Analog Computer | 1961 | J. J. Skiles and J. R. Hurley | Univ. of Wisconsin | CDC 1604 | 4th order Runge-Kutta | DEPI-4 | No | The prophet with honor *only* in his own country. Significant improvement of DEPI-4, particularly in format. The program was specific for a relatively little used computer, which probably caused its undeserved lack of wide acceptance and use outside the University of Wisconsin. |
| DYNASAR | DYNAmic Systems AnalyzeR | 1962 | Lucke, Robertson and Jones | General Electric-Evendale | IBM 704 and 7090 | Adams-Moulton 4 point predictor-corrector Variable integr. step-size | —— | Yes | Useful innovation was variable step size integration algorithm. |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| PARTNER | Proof of Analog Results Through Numerically Equivalent Routine | 1962 | R. F. Stover | Honeywell Aeorn. Div. | IBM 650 and H-800/1800 | Trapezoidal or Euler | —— | No but retains parallelism | Used extensively at Honeywell. Parallel nature retained by predicting variables around a feedback loop, rather than sorting. |
| DAS | Digital Analog Simulator | 1963 | R. A. Gaskill | Martin-Orlando | IBM 7090 | IBM 7090 | DYSAC | No | Major contributions to the format of block-oriented languages. Widely used in the Martin Company. |
| JANIS | ? | 1963 | R. G. Byrne | Bell Labs | IBM 7090 | Euler | —— | No | FORTRAN flavor |
| DES-1 | Differential Equation Solver | 1963 | M. L. Pavlevsky and L. Levine | Scientific Data Systems | SDS 9300 | Choice of five | —— | No | Excellent language. Part of a computer system that includes a special, analog type console. |
| DIAN | DIgital ANalog Simulator | 1963 | Farris and Buckhart | Iowa State Univ. | IBM 7074 | Euler | —— | No | Chemical Engineering Simulations |
| WIZ | ? | 1963 | J. E. Buchanan | U.S. Naval Avionics, Indianapolis | ? | 4th order Runge-Kutta-Gill | ASTRAL | Yes | ASTRAL's only known direct descendant. |
| COBLOC | COdap Language BLock Oriented Compiler | 1964 | Janoski and Skiles | Univ. of Wisconsin | CDC 1604 | Choice of three | DYSAC | Yes (Optionally No) | Logical building blocks (gates, flip-flops), etc. provided. |
| FORBLOC | FORTRAN compiled BLOck Oriented Simulation Language | 1964 | W. O. Vebber | Univ. of Wisconsin | Any machine FORTRAN compiler | Trapezoidal | DYSAC | No | Easily modified since FORTRAN used. This approach could lead to machine independence. |
| HYBLOC | HYbrid computer BLOck Oriented Compiler | 1964 | J. R. Hurley | Allis-Chalmers and Univ. of Wisconsin | IBM 709, 7090, 7094 | 4th order Runge-Kutta | DYSAC | No | Simulates hybrid computer. |
| MADBLOC | MAD Language BLOCk Oriented Compiler | 1964 | L. Tavernini | Univ. of Colorado | IBM 7090 | Trapezoidal | DYSAC | No | MAD (Michigan Algorithmic Decoder) statements. |
| MIDAS | Modified Integration DAS | 1964 | Harnett, Sansom and Warshawsky | Wright-Patterson AFB | IBM 7090-7094 | 5th order variable step predictor corrector | DAS | Yes | The Moses of the story, that led digital simulation to the verge of the Promised Land. Very widely distributed, modified and discussed. |
| SIMTRAN | —— | 1964 | W. J. Henry | Weapons Research Establishment — Australia | IBM 7090 | —— | DAS | Yes | Though not used on-line, the program's structured for such use in the future. |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| PACTOLUS | River in which King Midas washed off the golden touch. | 1964 | R. D. Brennan | IBM Research Lab | IBM 1620 | 2nd order Runge-Kutta | MIDAS | Yes | Mainly an experiment in man-machine communication. Widely used as a simulation tool. |
| ENLARGED MIDAS | —— | 1964 | G. E. Blechman | NAA-S&ID | IBM 7090 | Same as MIDAS | MIDAS | Yes | Enlarged component set of MIDAS and added plotting routines. |
| PLIANT | Procedural Language Implementing Analog Techniques | 1964 | R. L. Linebarger | IBM Develop. Lab. | IBM 7090 | Trapezoidal | JANIS | No | Build own FORTRAN blocks. |
| MIMIC | No meaning, Ex post facto, Sansom claims, MIMIC is MIDAS InCognito | 1965 | F. J. Sansom | Wright-Patterson AFB | IBM 7090-7094 | 4th order Runge-Kutta. Variable step size | MIDAS | Yes | Improvements over MIDAS include: compiler implementation, logical elements, improved algebraic capability, and logical control. |
| UNITRAC | UNIversal TRAjector Compiler | 1965 | W. C. Outten | Martin-Baltimore | IBM 7094 | ? | FOR-TRAN | Yes | Stylized differential equation input format. Free format. |
| DSL/90 | Digital Simulation Language for the IBM 7090 class computers | 1965 | Syn and Wyman | IBM Develop. Lab. | IBM 7090- | Choice of eight | PLIANT | Yes (Optionally No) | Powerful, flexible simulation tool. Advanced format ideas include free format capability. |
| SCADS | Simulation of Combined Analog Digital Systems | 1964 | J. C. Strauss and W. L. Gilbert | Carnegie Tech | CDC G-20 | An algorithm unique to SCADS. A four point method similar to Runge-Kutta. | PART-NER | No | Uses the same scheme for parallel operation as PARTNER, i.e., an extrapolation method. SCADS was used on-line at Carnegie Tech. |
| EASL | Engineering Analysis and Simulation Language | 1965 | L. Sashkin and S. Schlesinger | Aerospace Corp. | IBM 7094 | 4th order Runge-Kutta. Variable step size. | MIDAS | No | Used on-line through a terminal. FORTRAN statements are permitted in line. |
| SADSAC | Seiler Algol Digitally Simulated Analog Computer | 1965 | J. E. Funk | U.S. Air Force Academy | Burroughs B-5000 and B-5500 | 5th order variable step predictor-corrector | MIDAS | Yes | Essentially MIDAS, but written in ALGOL. Significant feature is handling of discontinuities by interrupting integration routines when switching occurs. |
| SLASH | Seiler Laboratory Algol Simulated Hybrid | 1965 | J. E. Funk | U.S. Air Force Academy | Burroughs B-5000 and B-5500 | 5th order variable step predictor-corrector | SAD-SACK | Yes | Gives an ALGOL program control of SADSACK for parametric studies, plotting optimization, etc. |

check cases, and to solve differential equations (in other words, replace an analog computer). MIDAS, ASTRAL and PARTNER[22] were written primarily for the first reason; DAS, the DEPI family, DES-1, and PACTOLUS, apparently for the second. Another, perhaps more important, use has not been stressed by any of the authors, i.e., providing the best digital computer language for a hybrid problem. Since an analog computer is a parallel device, and no amount of mental calisthenics can make it appear serial, it is clear that a parallel digital language is the only solution to the parallel-serial dichotomy in hybrid programming.

## FUNDAMENTAL NATURE OF SIMULATION LANGUAGES

### *Views Proposed*

As noted above, simulation languages have proliferated at an amazing rate in the last few years. Each new language comes equipped with its own format and structural idiosyncrasies, which generally reflect the creator's reading of the essence of simulation languages. These analyses might be classed as: analog computer simulator, block diagrammed system simulator, and differential equation solver. When the concepts are considered in detail, it is evident that all these views miss the point to some extent.

*Analog Computer Simulator:* Some simulation languages, as noted previously, were written specifically to simulate an analog computer. The purpose was to provide an independent check of the analog setup. Most of the authors state, however, that the resultant programs were used profitably for problem solving—in other words, the analog was bypassed.

Simulating the analog computer generally results in an operator (or statement) repertoire which reflects the fundamental physical limitations of analog computer elements. Examples of this phenomenon abound; one might mention the limitation on the number of summer inputs, different elements for variable and constant multiplication, and the lack of memory.

In short, the logical culmination of this concept is a system neither fish nor fowl, with many disadvantages of *both* analog and digital programming.

*Block Diagrammed System Simulator:* The overwhelming majority of authors state that their language (or program) was designed to simulate systems that can be represented by block diagrams. (Of course, an analog computer is such a system, so the opinion outlined in the previous subsection can be seen to be a sub-set of this view). By and large this opinion is justified, since many problems of interest are easily expressible in block form. However, if the problem is given simply as a set of ordinary differential equations, reducing the equations to a block diagram is a tedious, error-prone process. Even if a block-diagrammed system is considered, more often than not some of the blocks contain differential equations; an example is the airframe equations in a control loop.

*Differential Equation Solver:* An opinion sometimes expressed is that simulation languages should be designed to solve ordinary differential equations. From what has been said, this view has some merit. Unfortunately, two problems arise. First, in control system simulation, transfer functions and nonlinearities are not conveniently expressable in equation form. There is also a certain loss of familiarity with the system when blocks are completely eliminated. Second, the concept overlooks other important problem areas, e.g., sampled data systems, where the problem is stated in difference equation form.

### *A More Correct Approach*

It is seen, then, that all these views regarding the fundamental nature of simulation languages are too narrow and confining. Is there an "essence" (in the metaphysical sense) which is common to all, yet not so comprehensive as to be meaningless? *Parallelism,* the apparent parallel operation of a serial digital computer, may be an all-inclusive, rational statement of the essential nature.

All languages extant have taken their format and structural cues from analog programming and analog operators. Assuming the action was rational, and not an empty exercise in dialectical synthesis, this fact provides a clue to a valuable overall view of simulation languages. The analog computer is, of course, a parallel device.

As it happens, this is the way most of the world is structured. Representing physical phenomena with a serial digital computer is an artifice; useful, but nevertheless an artifice. The analog computer has achieved such success and generated such attachment largely because of the close analogy existing between the computer and the physical world.

Obviously, then, if physical systems must be represented with a serial digital computer, the machine should be made to appear parallel. This is in fact what has been done in simulation languages, and, of course, the success is manifest. Difficulties have arisen, though, because "pseudo-parallel" devices in the past been modeled too closely on the analog computer. If the notion of parallelism is correct, the *best* parallel device must be sought.

The importance of this concept cannot be overstated. It is not merely a convenient catch-all to include all previous efforts, but has real consequences for the future of simulation languages. A programmer if he is to "think parallel" must be freed from the chore of ordering problem statements. Such freedom is available if a sorting algorithm, as first proposed by Stein and Rose, is used. Alternatively, an extrapolation scheme, as used in PARTNER and SCADS, achieves the desired parallelism, but at a cost in storage and execution efficiency. The languages incorporating sorting or extrapolation are true *parallel* languages and provide the designer with a parallel device to represent his parallel physical system. It is always treacherous to be dogmatic, but on this point it seems clear that a language without sorting (or its equivalent) is simply another, perhaps slightly superior, method of programming a digital computer, and is in no way a parallel system simulator.

## FORMAT

### *Present Format Inadequacies*

Perhaps the most important consideration in designing a simulation language is the utility of the input format. A good, flexible, natural appearing format would encourage wide usage, facilitate training, and reduce errors. All existing formats are much too arbitrary and generally reflect both the artificialities of digital computer modes of thought, and the physically determined inadequacies of analog elements.

Under digitally derived restrictions, one might mentation the exaggerated importance given to column position in a statement, the need for commas and decimal points where unnecessary for clarity, and the requirement for a specific, arbitrary order of arguments within a statement.

Analog inadequacies have been mentioned; they appear as restrictions on the number of inputs to an element, poor logical and memory features, and rudimentary labelling capability. (This latter is a curious anachronism, since even the most primitive digital computer assemblers permit symbolic labelling). Some formats, notably ASTRAL, were based directly on a specific analog computer and may be expected to have certain deficiencies. In others, e.g., MIDAS and PACTOLUS, no real attempt was made to simulate an analog computer yet the implied hardware limitations are nonetheless present.

As a consequence of the poor format, operational difficulties are found to stem from trivial clerical errors, such as dropping commas or decimal points, or having input statements in the wrong order. These difficulties are increased by the multiplicity of primitive operators, and the consequent need for large complex networks to represent algebraic statements. The artificialities also tend to make the language more difficult to learn, or having been learned, to retain all the esoteric details. (The retention of these details might seem a small matter to the "professional" programmer, but is a real concern to the occasional user.) Modern computers with character handling ability and high execution speed can free the programmer from this sort of detail, with very little penalty in increased compilation time.

### *General Format Rules*

There is now a large fund of experience in the design and utilization of parallel languages, and some general, somewhat dogmatic, statements can now be made about format. On a very general level, these could be reduced to two rules: The format should be both "natural" and "non-arbitrary." These rules require some amplification.

*"Naturalness:"* The input problem statement should be as close as possible to the normal, accepted method of problem statement. The question in the designer's mind should be: if I were preparing a problem for my own future reference and explanation to others, how would I state it? The answer to such a query would naturally vary with the type of problem.

If a parallel control system is to be analyzed, the problem would most naturally be stated in a block diagram, wherein each block represented a more or less complex operator such as gain, limit, hysteresis, transfer function, etc.

If, however, a set of differential equations is to

be studied, it is a great deal of wasted effort to formulate the problem in block form. The process is conducive to error and really adds nothing to the understanding of the problem.

It is imperative that the input statement, the cards presented to the digital computer, should match as closely as possible the normal, natural statement of the problem. Of course, there are fundamental limitations, notably the fact that superscripts and subscripts are normally used while the card punch must work on one line. However, much can be done, and actually has been done to "naturalize" format, especially in the APACHE* and UNITRAC[23] programs.

To reiterate, the input statement should be flexible enough to match the natural form of diverse problems; block diagram and differential equation types have been mentioned. Many problems arise that are really combinations of these classes, e.g., an airframe in an autopilot loop. The format should natually be capable of stating each problem area in its normal form.

*Non-Arbitrariness:* Arbitrary formats, more than anything, have limited the utility of parallel languages. One commonly finds early enthusiasm for the idea of parallel languages, and then disenchantment when the "format gap" between the idea and its implementation is fully appreciated. In many facilities, where digital computer turn-around is measured in days, the trivial errors caused by the complex, arbitrary format extend a problem's check-out time to unacceptable periods.

However, this is not a fundamental problem; surely parallel languages can be written to eliminate most of these difficulties. It seems that few authors have given much thought to minimizing arbitrariness in their concern for other aspects of the language. This is seen clearly in the early work of Selfridge and Lesh, where the root idea of parallel languages was the main subject of study. Unfortunately, examples are still apparent: Many outstanding contributors, in their understandable enthusiasm for man-machine interaction and efficient implementation, have been satisfied with adopting earlier format ideas and have neglected the ramifications of a good, clear, non-arbitrary input statement.

*The APACHE[24] program is not really of the genre under consideration here. The program takes differential equations and generates an analog wiring diagram and static check. However, the format considerations are nearly identical to equation solving languages, and the APACHE authors have produced an input format worthy of study.

A few obvious requirements are discussed below:

(a) A "free format" statement capability, i.e., statements can appear anywhere on the card. This is available in the UNITRAC and DSL/90 programs, and should eliminate much of the frustration produced by coding or key punch slips.

(b) The ability to enter numerical data in any convenient form, e.g., 200.0 might be written 200, 200.0, 200., 2E2, 2.0E02, etc.

(c) The ability to use either literals in a statement (Y = 4X) or the option to symbolically label constants (Y = KX). In the latter case, the constant would be specified in the normal fashion. (K = 4 or K = 4., etc.)

(d) The ability to label quantities in a sensible, problem related fashion, and use this label to specify the variable without reference to an arbitrary block number. The latter labeling method could be retained for meaningless intermediate quantities. It should be noted that the need for problem related labeling was one of the first lessons learned by software designers and is now available with virtually all assemblers.

### *Complexity of Format*

An open question at this time is the allowable degree of complexity in the input format statement. The trend in new languages appears to be away from the simple, analog type blocks to statements reminiscent of FORTRAN.

Table 2 shows an "algebraic capability" scale, with some of the languages distributed along it. At the lower end, one finds a very primitive capability, which can represent any algebraic statement, albeit in an extremely awkward form. Fortunately, no one has been inspired to implement this sort of language. (This is not to say such codes are useless; a primitive language is generally the intermediate representation in a compiler program.) Moving up the table, the next stage is basic mathematical operators modeled by and large on analog computer components. DAS and MIDAS are good examples of this class. Here, there is some advance from a "minimum vocabulary" and a great deal of flexibility is available, particularly for block oriented sys-

Table 2. Algebraic Capability Scale.
(listed in order of decreasing capability)

| Description | Examples | Typical Statement |
|---|---|---|
| Statements in any form understandable by the engineer | — | $dy/dt = y \sin w\ t + x^2$<br>$y_o = 5,\ y_o = 0,\ w = 2^{II}$ |
| Nested sum of products and any functions or operators | — | Z = (XY + K1SIN WT)*ERF (L*N) – INT ((A + B) *C) K2 |
| Nested sum of products and operators | MIMIC | Y: ADD (X,MPY (B,Z,SIN)(U))) |
| Nested sum of products and certain functions (a la FORTRAN) | DSL/90, UNITRAC | Y = X * Y *(SIN(A + B))+K*M |
| Nested sum of products | DES-1 | Y = X *(C1 + C2 – C3*(C4–C5)) |
| Single level sum of products | — | Y = K1*X*X + K2*Z + K3*K4 – K5 |
| Coefficients on inputs to block operators | ASTRAL, DEPI, DYSAC | NO4 = PO1*NO1 + PO2*NO2 |
| Basic mathematical operator (a la analog) | DAS<br>MIDAS | MI: S1, I2<br>S1: K1, I2, M3, K5 |
| Primitive operators, the minimum necessary, with the minimum inputs | — | M1: S1, I2<br>S1: K1, I2<br>S2: S1, K3<br>I2: K1 |

tem representation. However, programming is awkward for algebraic type problems.

The next stage provides coefficient setting on all inputs to blocks of the type discussed above.* As on an analog computer, only constants can be multiplied by the input variable.

A natural extension of coefficient setting is, of course, variable multiplication at element inputs, the next step up the table. (Division is also assumed permissible here.) This stage has not been implemented, probably because the transition to the next stage is so evident.

In this stage, exemplified by DES-1, nesting of sums of products (and quotients) is allowed, i.e., any level of parentheses is permissible. A flexible statement is provided, although no functional relationship (sin, exp) can be imbedded in the sum of products.

This lack is provided at the next stage, now available in the DSL/90 and UNITRAC languages. Here, an input statement very similar to FORTRAN is available; a limited number of functions may be used in the statement.

It might seem odd that a MIMIC type language, which allows only operators in the statement, should be set above UNITRAC or DSL/90 which permit functions. However, since function generators can be considered as operators, this format is extendable downward, and further allows operations like integration and limiting to be imbedded in the input statement.

Moving now to the top, the next stage provides a synthesis of the two below, allowing both operators and functions to appear in the nested sum of products. The allowable function list would be open ended and at the user's discretion. No published work provides this capability. At the very top, and really hardly in sight, is the capacity to accept any reasonable problem statement understandable to the engineer.

There certainly are objections to considering upward movement on the table as evolutionary, with the implied value judgment concomitant to that view. The authors would agree that a powerful algebraic capability, although very useful for some tasks, must at the present state of the art carry with it increased complexity and arbitrariness. The tool may prove too powerful for many users and lead to confusion and errors. Further, many users prefer the block approach and with excellent reasons.

However, all fears can be allayed, since regardless of the available complexity, lower level statements are possible by simply limiting the size and complexity of the more powerful statement. A close examination of Table 2 will show that any of the formats are easily derived by restricting the extent of those above.

*Diagnostics*

In general, program error diagnostics should be extensive and specific. Each card containing the error, and only that card, should be printed, along with a specific comment on the trouble. Alternatively, diagnostics could be printed immediately adjacent to the erroneous statement, as the source language is being printed. Closed algebraic loops

*It is seen here that the evolutionary movement up the table is not in strict chronological order. ASTRAL and DEPI preceded DAS and MIDAS.

should naturally be printed separately from format errors.

It is expected that improved format will reduce this sort of error, but some are sure to appear and rapid checkout requires good diagnostics.

Diagnostics and corrections to the program must be permitted in the source language to free the programmer from the details of debugging at the machine code level. For hybrid work it is essential that such source level debugging and modification be permitted on-line through a console typewriter or similar unit. Extremely rapid compilation is of course necessary to make this economical.

On-line debugging and modification has been explored more fully in an earlier article by the authors.[25] The particular framework in point was a huge multiprogrammed digital computer with analog type terminals (one might say a large scale multiprogrammed PACTOLUS, or a time shared DES-1). With such a system and a parallel language, the advantages of analog computers (parallelism, intimate man-machine communication) and digital computers (accuracy, repeatibility) are both apparent. The language employed with this system must permit clear diagnostics and simple modification at the source level to retain the analog virtues of simple communication and rapid modification.

## STRUCTURE

### *Integration with Software System*

As is well known, all modern software systems contain many languages; one might mention FORTRAN, assembly language, ALGOL and COBOL. These languages are generally under the control of a monitor or executive program, which calls programs to compile (or assemble) various source programs into machine code. Usually subprograms can be written in any of the source languages, and a complete program linked at load time.

The parallel language should come under this organization, and be available along with FORTRAN and the rest, as the optimum source code for a particular class of problems. In this way, and using the subprogram feature, each problem area could be written in the best language for a particular task; say, parallel language for the differential equations, FORTRAN for the arithmetic, and machine language for chores such as masking, character handling, and Boolean operations.

### *Extension Capability*

In order to remain useful in the face of continuously expanding user requirements, any language must be able to grow with needs. When a parallel language is examined with the intent of increasing capability without organizational disruption, it is seen that expansion should take the route of adding operators or functions. Expansion must, of course, fit neatly into the total software system, i.e., the other languages and the executive program, as outlined above. Since a common subroutine format is already available with the other languages, any subroutine, written in any source code, could be used by the parallel language programmer and be called simply by using the subroutine name as an operator.

Augmenting the operator set must be made very simple and obvious so the average user, unfamiliar with normal digital techniques, can exploit the extension feature without recourse to a systems or maintenance programmer.

### *User-Oriented Organization*

The language should be designed to easily match the capabilities of diverse programmer levels. Basic subsets, such as primitive operators, algebraic statements, etc. should be made available to the less sophisticated programmers. These subsets should be capable of integration and mixed use by the more highly skilled user. Also, elements of a more complex nature (e.g., serial operators) should be available to the expert, but not a matter of concern for the novice. Thus, a structure is required that will permit the novice to learn a mimimum subset and then advance, if he wishes, to the use of an extremely complex and powerful simulation language. (Or looking at it yet another way, there should be open and closed shop versions; the various open versions upwardly compatible with the closed shop version.)

In sum, there seems to be no need to restrict the language's use to a particular programmer level, if the initial design is done in a systematic manner.

## IMPLEMENTATION

Regardless of format and structure, the language's effectiveness will depend entirely on the quality of

the implementation. This aspect has recently been a major interest area, and the concepts are becoming rather well developed.

In general, a program that produces machine code is a necessity for efficient execution. MIMIC and SCADS, compilers, directly achieve this, while DSL/90 and ASTRAL generate a FORTRAN deck which can then be compiled into an object program. This latter approach, (if a good FORTRAN compiler is used), can produce efficient code by exploiting the considerable efforts expended by FORTRAN designers. There are certain applications, particularly those with small machines, where an interpreter program makes more sense, but generally a compiler seems the best route. This is detailed more fully below. First, an examination of the trade-offs involved in writing a compiler program for a parallel language.

*Compiler for Different Applications*

As was mentioned, there are three major usage areas for parallel languages: analog check cases, differential equation solving, and the digital protion of a hybrid problem. The relative weights given to compiling and execution times vary with the particular application.

*Analog Check Cases:* This usage is generally on a single job basis, i.e., the program is compiled and run once and then discarded. Since the object program is never used again, only the sum total of compiling and execution time for one run need be minimized. In fact, this minimization is hardly a point to stress, since analog check cases would probably represent a small total of a digital facility's work load.

*All Digital Simulation:* If the language is to be used for this application on a "load-and-go" basis, minimization of the total time is of prime importance. On the other hand, if production programs are the expected rule, execution time is the quantity to be minimized.

*Hybrid:* Here, the requirement for an efficient object program is a vital consideration, and real sacrifices can and must be made in compiling efficiency.

As a general rule, compiling time should never be minimized at the expense of input format, and only as a last resort should format be sacrificed for decreased execution time. This latter seems a remote possibility, but it is easy to see compiling time increased in the interests of simpler programming.

*Different Computers*

If this language is to achieve the general usage typical of FORTRAN, some thought must be given to implementation for diverse computers. It is pointless to design a system workable only for a CDC 6600-6800 or the top of the IBM 360 line. Similarly, it is a waste of effort to aim at implementation solely for a PDP-8, DDP 116 or SDS 92. The large machines obviously should have the full language, i.e., all subsets, and be provided compiled versions. For the smaller machines, two approaches are possible. The basic subsets could be compiler versions, thus providing efficient programs although only for small problems and at a modest language level. Alternatively, the complete system could be run in an interpreter mode, sacrificing time, but permitting the use of a very powerful tool on a small machine.

## REQUIRED FEATURES

Along with implementation, the operational features of the language (the programmer's bag of tricks) have been a major concern of language designers. This section does not aim to be all-inclusive; probably some of the "required" features have not been invented yet. The requirements can be subdivided into two classes: structural or logical features, and the types of elements or operators. Sorting is not discussed, since it is assumed that all modern parallel languages will be so equipped.

*Structural Features*

1. *Logical Control:* Logical control over program structure and execution is of paramount importance. DES-1, being sequential, easily incorporates this feature by the use of "IF" statements similar to FORTRAN. MIMIC, a true parallel language, still provides decision capability with the "logical control variable."

   In substance, statements or operators are serviced or bypassed as a function of problem variables. So long as the by-passing is done in the true digital sense (non-execution), and not the analog sense (execution, but no use made of the outputs), a substantial time savings is realized.

Logical control is quite important. Without it a parallel language yields no more than a hyper-accurate and hyper-repeatable analog computer; some of the best features of the digital computer, decision capability and memory, are unused.

2. *Multiple Rates:* This important provision, available in DES-1, minimizes execution time by servicing slowly changing variables only as required. Generally speaking, multiple rates increase the effective bandwidth of the simulation program. This has real import for hybrid work.

   The multiple rate option is clearly a part of logical control capability. In this case, sections of the program are not executed at every pass, but unlike full logical control, the bypassing is not under control of a program variable.

3. *Macro Capability:* The macro capabilities of modern assemblers should be available to the parallel language programmer. Using this feature, prototypes of often used problem sections could be coded with unspecified parameters, and then subsequently used as integrated units. Macros would obviate repetitious coding of identical loops or problem areas, e.g., parallel sections of control systems that are identical in structure.

4. *Subprograms in Other Source Code:* In a normal digital computer operation, there is always a large library of routines available to the programmer. These programs should be easily incorporated within the parallel language. If expansion is implemented as suggested (see Extension Capability), not only would the entire library be available, but the programmer with digital training could use whatever language desired for particular problem areas. For example, logical or Boolean operations would be most easily handled in machine language.

5. *Repetitive Operation and Memory:* It should be possible to repeat runs, as on a repetitive analog computer, with new parameters calculated from previous runs. This implies two further requirements: function storage, and algebraic calculations between runs.

*Elements*

In this section, the normally found operators, e.g. summers, multipliers, etc., are taken for granted. No attempt has been made to be comprehensive; however, those discussed are considered important and/or relatively rare in present languages.

1. *Integrator:* An accurate, efficient integration method is the *sine qua non* of digital simulation languages. Apparently no firm conclusions have been reached as to the best algorithm; the number of schemes tried is almost as large as the number of languages (See Table 1). As an example of the dynamics of this situation, note that Sansom, having used an excellent method (4 point variable step predictor-corrector) in MIDAS, changed to another (modified Runge-Kutta) in MIMIC.

   DES-1, DSL/90 and COBLOC permit a number of integration options, ranging from simple Euler integration to complex Runge-Kutta and Adams-Bashford algorithms. This variety does allow the user to select a scheme which is adequate for the required execution time, but presupposes considerable knowledge of numerical techniques on the part of the programmer. This presupposition defeats in large part the basic idea, i.e., the simplicity and ease of use, even for relatively untrained people.

   In sum, it appears at this time that the debate is hot on integration methods, and more experience is still required. Parenthetically, it might be said that an objective, thorough comparison of the various options would be a real service to the field of digital simulation.

2. *One Frame Delay:* This element delays its input by one integration interval. It should be available for the sophisticated user to selectively "desort" the program list. COBLOC and DSL/90 presently have the option of sort/no sort, but if the no sort option is required in only one small area, much care must be taken in the other sections to insure proper operation. The one frame delay is also quite useful for representing sampled data systems or memory functions.

3. *Hysteresis or Backlash:* This element is not easily constructed from standard analog type elements, but represents a trivial task for the digital computer.
4. *Limited Integrator:* Again, no easy chore from the standard elements, and no real effort for the digital computer. MIMIC presently has a limited integrator element which is used in conjunction with a standard integrator.
5. *Transfer Functions:* These operators are used extensively in control system design and the like, and are simply constructed from analog type elements. However, the very frequency of their use suggests they be made available as integrated general-purpose units.

   In addition to the programming time savings, execution time can be saved, since the integration algorithm required for a closed loop transfer function is much simpler than a comparably accurate routine for open loop integration.

   Far greater savings are possible by using a difference equation algorithm. This method requires only *one* computational step of the same size and complexity of a single integration. Compare this with the n integrations required for a transfer function with nth order denominator, when programmed by normal analog methods.
6. *Print Operators:* Very often, in checkout and operation, it would be helpful to force a print (number or words) at an arbitrary point in the program. The operator would be similar to a "snapshot" print, but would be under the control of problem variables. As a trivial example, consider the printing of "OVERLOAD" when an analog check case variable exceeds 100.
7. *Parallel Logic:* The operators of interest here are the normal digital units found on most modern analog computers. (AND gates, flip-flops, counters, shift registers, etc.) These elements are presently available in the MADBLOC, COBLOC, and MIMIC languages. Their inclusion is essential to provide a digital check for a modern analog computer problem. When solving differential equations, such units are also useful for simple logic and storage. For the higher level logic of the type normally associated with general-purpose digital computers, machine language subprograms, as discussed above, are more useful.
8. *Linkage Elements:* For hybrid programming, elements or labels for analog to digital converter (ADC) and digital to analog converter (DAC) components must be provided. ADC's could be handled simply as another parameter input to the problem. These could be realistically labeled and then identified somewhere in the program listing, e.g., ALPHA = ADC1. DAC's could also be easily handled; for the digital program they are merely elements with one input and no output. Sorting these elements presents no difficulties: ADC's would be treated exactly like constants; DAC's would be sorted like any other element which has an input.
9. *Hybrid Integrators:* Since variables transferred to the analog computer are usually held for an entire frame, an effective half interval time lag results. Thus, integrators generating quantities destined for the analog must account for this lag phenomenon. Those integrators in a purely digital loop must, of course, neglect this extrapolation. Therefore, different integrator types, easily distinguishable and easily specified, must be provided for the two requirements. It is entirely possible that the sorting routine could automatically make the necessary distinctions by tracing back from a DAC to integrators not isolated by another integrator. An alternate procedure is the addition of an extrapolation calculation to each DAC element. However, this approach costs both storage and execution time.

## CONCLUSIONS

Digital simulation languages have made a real, and probably permanent, impact on the fields of both simulation and computer programming. As has been pointed out, there are more or less serious faults in all existing languages. The success of the approach is, however, evidenced by the undeniable acceptance, utilization and enthusiasm for simulation languages, regardless of the difficulties at the present phase of development.

It is the authors' hope that the conclusions and recommendations proposed herein will add significantly to the utility of simulation languages, and the field will enjoy even further growth and acceptance.

## REFERENCES

1. John Locke, *An Essay Concerning Human Understanding,* c.f., *The Age of Enlightenment,* editor—Isaiah Berlin, Mentor, New York, 1956 p. 33.

2. R. G. Selfridge, "Coding a General Purpose Digital Computer to Operate as a Differential Analyzer," *Proceedings 1955 Western Joint Computer Conference* (IRE), 1955.

3. R. D. Brennan and R. N. Linebarger, "A Survey of Digital Simulation: Digital Analog Simulator Programs," *Simulation,* Vol. 3, No. 6. (Dec. 1964).

4. R. D. Brennan and R. N. Linebarger, "An Evaluation of Digital Analog Simulator Languages," *I.F.I.P. 1965 Proceedings,* Vol. 2 (1965).

5. F. Lesh, "Methods of Simulating a Differential Analyzer on a Digital Computer," *ACM Journal,* Vol. 5, No. 3 (1958).

6. J. R. Hurley, "DEPI 4," internal memorandum, Allis Chalmers Mfg. Co. (Jan. 1960).

7. J. R. Hurley and J. J. Skiles, "DYSAC," *Spring 1963 Joint Computer Conference,* Vol. 23, Spartan Books, Inc., Washington, D.C., 1963.

8. V. C. Rideout and L. Tavernini, "MADBLOC," *Simulation,* Vol. 4, No. 1 (Jan. 1965).

9. J. J. Skiles, R. M. Janoski and R. L. Schaefer, "COBLOC," paper presented at Joint Meeting of Midwestern and Central States Simulation Councils (May 1965).

10. M. L. Stein, J. Rose and D. B. Parker, "A Compiler with an Analog Oriented Input Language (ASTRAL)." *Proc. 1959 Western Joint Computer Conference, 1959.*

11. M. L. Stein and J. Rose, "Changing from Analog to Digital Programming by Digital Techniques," *ACM Journal,* Vol. 7, No. 1 (Jan. 1960).

12. R. A. Gaskill, J. W. Harris and A. L. McKnight, "DAS," *Spring 1963 Joint Computer Conference,* Vol. 23, Spartan Books, Inc., Washington, D.C., 1963.

13. R. T. Harnett and F. J. Sansom, "MIDAS Programming Guide," Report No. SEG-TDR-64-1, Wright-Patterson AFB, Ohio, (Jan. 1964).

14. F. J. Sansom and H. E. Petersen, "MIMIC—Digital Simulator Program," SESCA Internal Memo 65-12, Wright-Patterson Air Force Base, Ohio (May 1965).

15. M. Palevsky and J. V. Howell, "DES-1," *Fall 1963 Joint Computer Conference,* Vol. 24, Spartan Books, Inc., Washington, D.C., 1963.

16. Anonymous, "SDS DES-1," Scientific Data Systems Descriptive Brochure, No. 64-42-01C (1964).

17. R. D. Brennan and H. Sano, "PACTOLUS," *Fall 1964 Joint Computer Conference,* Vol. 26, Spartan Books, Inc., Washington, D.C., 1964.

18. R. N. Linebarger, "DSL/90," paper presented at Joint Meeting Midwestern and Central States Simulation Councils, May 1965.

19. A. B. Clymer, "Report on Joint Midwestern-Eastern Simulation Council Meeting, June 1964," *Simulation,* Vol. 3, No. 4 (Oct. 1964).

20. J. C. Strauss and W. L. Gilbert, "SCADS," 2nd Edition, Carnegie Institute of Technology (March 1964).

21. L. Sashkin and S. Schlesinger, "A Simulation Language and its Use with Analyst-Oriented Consoles," Aerospace Corp. Report, ATR-65 (59990)-5, San Bernardino, Calif. (April 1965).

22. R. F. Stover and H. A. Knudston, "PARTNER," Doc. No. U-ED 15002, Aero Divn., Honeywell (1962).

23. W. C. Outten, "UNITRAC," paper presented at Joint Meeting Midwestern and Central States Simulation Councils (May 1965).

24. C. Green, H. D'Hoop, and A. Debroux, "APACHE," *IRE Transactions on Electronic Computers* (Oct. 1962).

25. J. J. Clancy and M. S. Fineberg, "Hybrid Computing—A User's View," *Simulation,* Vol. 5, No. 2 (Aug. 1965).

# AUTOMATIC SIMPLIFICATION IN FORMAC

R. G. Tobey, R. J. Bobrow and S. N. Zilles
*International Business Machines Corporation*
*Systems Development Division*
*Cambridge, Massachusetts*

## INTRODUCTION

Simplification is a central and basic operation in the manipulation of mathematical expressions. Indeed, much of the tedious algebra that plagues scientists and engineers involves the time-consuming application of simplifying transformations to unwieldly mathematical expressions. It seems obvious, conceptually, that some simplifying transformations can be applied "automatically" to arbitrary expressions. However, there are transformations that require special handling; they simplify some expressions and complicate others.

FORMAC, an acronym for FORmula MAnipulation Compiler, is an experimental programming system currently available as a Type II program from IBM. It is a tool for programming the IBM 7090/94 to perform tedious mathematical analysis on complicated mathematical expressions. The FORMAC language contains, as a subset, FORTRAN IV; hence, FORMAC provides the capacity for performing both nonnumeric and numeric calculations in the same program. The FORMAC language is described thoroughly in increasing amounts of detail in references 1, 2, and 3. The details of FORMAC implementation are presented in reference 4.

The remainder of this paper is divided into four sections: Historical Background, The Role of Simplification in the FORMAC System, Simplification Transformations, and The FORMAC Simplification Algorithm.

## HISTORICAL BACKGROUND

A "SIMPLIFY" routine was written as early as 1959 as part of the Dartmouth Mathematics Project. It is the most complex routine reported in reference 5. During the same academic year, Edwards[6] and Goldberg[7] explored the possibilities of automatic simplification in the context of electrical circuit analysis. A LISP coded simplification package was central to Goldberg's work. Within the next two years Maling[8] discovered that simplification was essential to the LISP differentiation effort and Hart[9] developed "SIMPLIFY," a LISP function for simplification. In 1963 Wooldridge[10] completed another LISP simplify program, which was written to be used on-line in a time-sharing environment. In April of 1964 the experimental FORMAC system completed systems test and became operational. It included a comprehensive simplification capability. In November 1964 the FORMAC system was released as a Type III program.

The FORMAC AUTomatic SIMplification routine (AUTSIM) presents several contrasts to previous efforts:

1. None of these efforts is part of a comprehensive mathematical-expression manipulation system, nor are they as ambitious as AUTSIM.
2. The LISP coded efforts are recursive. AUTSIM is essentially nonrecursive; although the basic scan is controlled by a push-down store, the simplification transformations do not employ recursion.
3. With respect to his own effort, Wooldridge (reference 10, page 31) observes, "There is no doubt that a large proportion of the time spent simplifying is devoted to repeating simplifications already done." AUTSIM is designed to avoid redundant simplifications. This is a fundamental aspect of the AUTSIM scan and the entire FORMAC object time system.

## THE ROLE OF SIMPLIFICATION IN THE FORMAC SYSTEM

The FORMAC programming system consists of three parts—a programming language, a preprocessor, and a set of object time routines. In this section we discuss the relationship of automatic simplification to the FORMAC programming system.

The FORMAC programming language is a proper extension of FORTRAN IV and contains 4 declarative statements and 15 executable statements in addition to the full FORTRAN IV language. FORMAC also introduces additional symbolic mathematical operators from which symbolic expressions can be composed. The language statements are summarized in Table 1, and the list of operators that may be used to compose symbolic expressions is displayed in Table 2. These additions to the FORTRAN IV language permit the user to construct and manipulate symbolic mathematical expressions at object time. The statements listed under 2b in Table 1 provide an interface with the FORTRAN program logic and with FORTRAN numeric capabilities. The form of generated symbolic expressions can control the logic of program execution. Symbolic expressions can be evaluated and the numeric results used in FORTRAN-coded, numeric calculations.

The FORMAC preprocessor scans through a FORMAC source program and converts the FORMAC language elements into FORTRAN IV statements. Among these are many calls to FORMAC object time routines. In addition, the preprocessor creates a prototype for each symbolic expression that occurs explicitly in a FORMAC command. This prototype is used by the object time routines

Table 1. Summary of FORMAC Language Extensions to FORTRAN IV.

1. Four Declarative Statements

| | |
|---|---|
| ATOMIC | declare basic variables, which name themselves. |
| DEPEND | declare implicit dependence relations. |
| PARAM | declare parametric pairs for SUBST and EVAL. |
| SYMARG | declare subroutine arguments as FORMAC variables; flag program beginning. |

2. Fifteen Executable Statements

(a) statements yielding FORMAC variables

| | |
|---|---|
| LET* | construct specified expressions. |
| SUBST* | replace variables with expressions. |
| EXPAND* | remove parentheses. |
| COEFF* | obtain coefficient of variable or of a variable raised to a power. |
| PART | partition expressions into terms, factors, exponents. |
| ORDER* | specify sequencing of variables within expressions. |

(b) statements yielding FORTRAN variables

| | |
|---|---|
| EVAL* | evaluate expression. |
| MATCH* | compare two expressions for equivalence or identity. |
| FIND* | determine dependence relations. |
| CENSUS | count words, terms, or factors |

(c) miscellaneous statements

| | |
|---|---|
| BCDCON | convert to BCD form from internal form (prepare symbolic expressions for output with FORTRAN "WRITE" statement). |
| ALGCON* | convert to internal form from BCD form (facilitates input of symbolic expressions with FORTRAN "READ" statement). |

*These commands called AUTSIM.

| | |
|---|---|
| AUTSIM | control arithmetic done during automatic simplification. |
| ERASE | eliminate expressions no longer needed. |
| FMCDMP | symbolic dump |

Table 2. FORMAC Operator Set.

| | | | |
|---|---|---|---|
| + | | EXP | |
| — | (unary internally) | LOG | (natural logarithm) |
| * | | SIN | |
| / | (external only) | COS | |
| ** | (power, ↑ ) | ATAN | (arctangent) |
| FAC | (factorial) | TANH | |
| DFAC | (double factorial) | ] | (delimiter) |
| COMB | (combinatorial) | DIF | (differentiation) |

to generate the required expression when the corresponding FORMAC command is executed. Consider the segment of a sample FORMAC program represented by statements 1 and 2, below.

```
1. LET U = (1 + Z)**N
2. LET X = (1 + Z)**M - U + Z**5
```

These two statements cause the prototype expressions 1p and 2p to be constructed by the preprocessor.

```
1p. (1 + Z)**N
2p. (1 + Z)**M - U + Z**5
```

When the call statement, generated by statement 1 during preprocessing, is executed at object time, the symbolic variable U is defined as the name of the newly generated expression (1 + Z**N. This is accomplished by scanning the prototype expression which the preprocessor constructed for (1 + Z)**N and replacing the variables Z and N by their current values. Let us suppose that Z is a symbolic variable, i.e., it is either the name of a symbolic expression or it may be an ATOMIC variable. In either case its value is symbolic. N, on the other hand, is a FORTRAN variable. Its value is numeric. Then the generated expression for 1p will be like 1p, only Z and N will have been replaced by their current values. If the value for Z is W-1 and for N is 2, then U names the generated expression 1g. Similarly, if the value for M is 3, then—after execution of statement 2—X names the expression 2g.

```
1g. (1 + W - 1)**2
2g. (1 + W - 1)**3 - (1 + W - 1)**2
        + (W - 1)**5
```

Note that both these expressions require simplification. The extent to which they may be simplified is unknown at compile time. Moreover, the degree to which the FORMAC user understands (when writing the program) just which simplifications will be applicable to these expressions depends on the complexity of the logic of the program in which they are embedded. For example,

```
(1 + Z)**M - U
```

will cancel if the current values of M and N are equal; however, the values of M and N may be determined by quite complex program logic. We see that in a mathematical formula manipulation system, organized as FORMAC is, an object time simplification capability is essential. (The expressions being manipulated may be far more complex than in the above example!) Moreover, that capability must be, to as great an extent as possible, automatic.

The FORMAC object time system is composed of many subroutines. The command level routines which correspond to the FORMAC executable statements are the basic object time routines. These in turn call a number of service routines. The automatic simplification routine, AUTSIM, is the most important service routine. Each command which is starred in Table 1 calls AUTSIM at least once.

The role played by AUTSIM in the FORMAC object time system is not unlike that envisaged by the Dartmouth Mathematics Project.[5] Their SIMPLIFY routine was designed to serve a threefold purpose:

(1) The answers produced by other programs may appear in a far more complicated form than necessary, and then it is desirable to simplify them.
(2) It may be desirable to simplify a given formula before applying one of the other routines.
(3) It is a program that attempts to reduce formulas to a canonical form. Such a form is particularly useful, for example, when we try to find out whether two formulas represent the same function or not.

Purpose (1) is relevant to FORMAC; the result of an EXPAND (see Table 1) may require the collection of like terms in a sum. There is an additional aspect which is worth noting. The design of the

algorithms for the basic FORMAC commands was simplified by the assumption that AUTSIM would clean up expressions after manipulations had been performed. It was not necessary to rule out a fast, simple algorithm for a particular manipulation simply because it produced a more complicated expression than necessary. Moreover, the inclusion of code in the algorithm to eliminate redundant expression elements and to clean up expressions, would have lead to a proliferation of redundant code throughout the FORMAC object time system.

Several FORMAC command algorithms are greatly simplified by the assumption that the expression they receive is already simplified and in p-canonical form (to be defined later). This is close to the intent of purposes (2) and (3) above. Note, however, that in purpose (2) the Dartmouth investigators were also concerned about the success of their algorithms on a given expression if it were unnecessarily complicated. The concern was not as significant in the design of FORMAC algorithms, but then the FORMAC design goals did not include symbolic mathematical manipulations for which complete algorithms do not exist.

The AUTSIM algorithm itself makes use of the applicability of purpose (3). Collection of like terms in a sum of like factors in a product is dependent upon the assumption that such terms or factors will be in a nearly identical form.

Since AUTSIM is at the iterative heart of the FORMAC object time system, it is important to consider just how fast the algorithm is.

An example is provided in reference 11. The FORMAC system was used to generate expressions of interest to the astronomer. The computer (IBM 7094) required 18.67 minutes to generate the first 27 iterates. It is estimated that to do this work with any reliability would require 60 years by hand. AUTSIM is called 2,975 times during execution of this program. Note, this does not mean that every time AUTSIM was called it changed the input expression. If the expression was already simplified no simplification was performed. This also holds for subexpressions which have been simplified previously (see Fig. 1). No doubt, many of the calls in the above example resulted in very little actual simplification. But this is a mark of a well-designed simplification algorithm; it must not perform redundant simplifications.

We have seen that an automatic simplification algorithm is essential to the successful operation of the FORMAC system. Moreover, once the decision has been made to include an automatic simplification algorithm in a formula manipulating system, the design of the other object time algorithms is simplified in two ways: one can make definite assumptions concerning the form of expressions which are to be manipulated; and, the form of the manipulated result need only be mathematically correct (it may contain redundant or unnecessary subexpressions which the automatic simplification routines will remove).

## SIMPLIFICATION TRANSFORMATIONS

As intuitively obvious as the need for it may be, simplification is a difficult class of expression transformations to define. Even to a human engaged in the manipulation of complicated or lengthy expressions, it is frequently not obvious which transformations constitute actual simplifications of an expression. The confusion is compounded by differences of opinion. Wooldridge[10] acknowledges this aspect of the problem by referring to his program as one "which performs 'obvious' (noncontroversial) simplifying transformations." Confusion may also arise from the failure to make a distinction between "simplified" form and "intelligible" form. Frequently, these are not equivalent. Consider the expression

$$C\sqrt{\frac{2A}{DE\ (2DF + C)}}$$

It is in some sense simplified, yet for the engineer it may be more intelligible in the form:

$$\frac{D}{C}\sqrt{\frac{A}{E\ (F + C/2D)}} = \phi\sqrt{\frac{A}{E\ (F + \phi/2)}}$$

with $\phi\ C/D$. An engineer may spend weeks or months massaging a simplified mathematical expression. His goal is to arrange the expression so that the relationship between the crucial variables becomes transparent or intelligible to the human observer. The adequacy of the result is highly dependent upon human perception of mathematical relationships. Yet the expression is already simplified. Like terms and factors have been collected in sums and products; evaluation of strictly numerical elements has been performed; various redundant or extraneous elements have been removed. Although

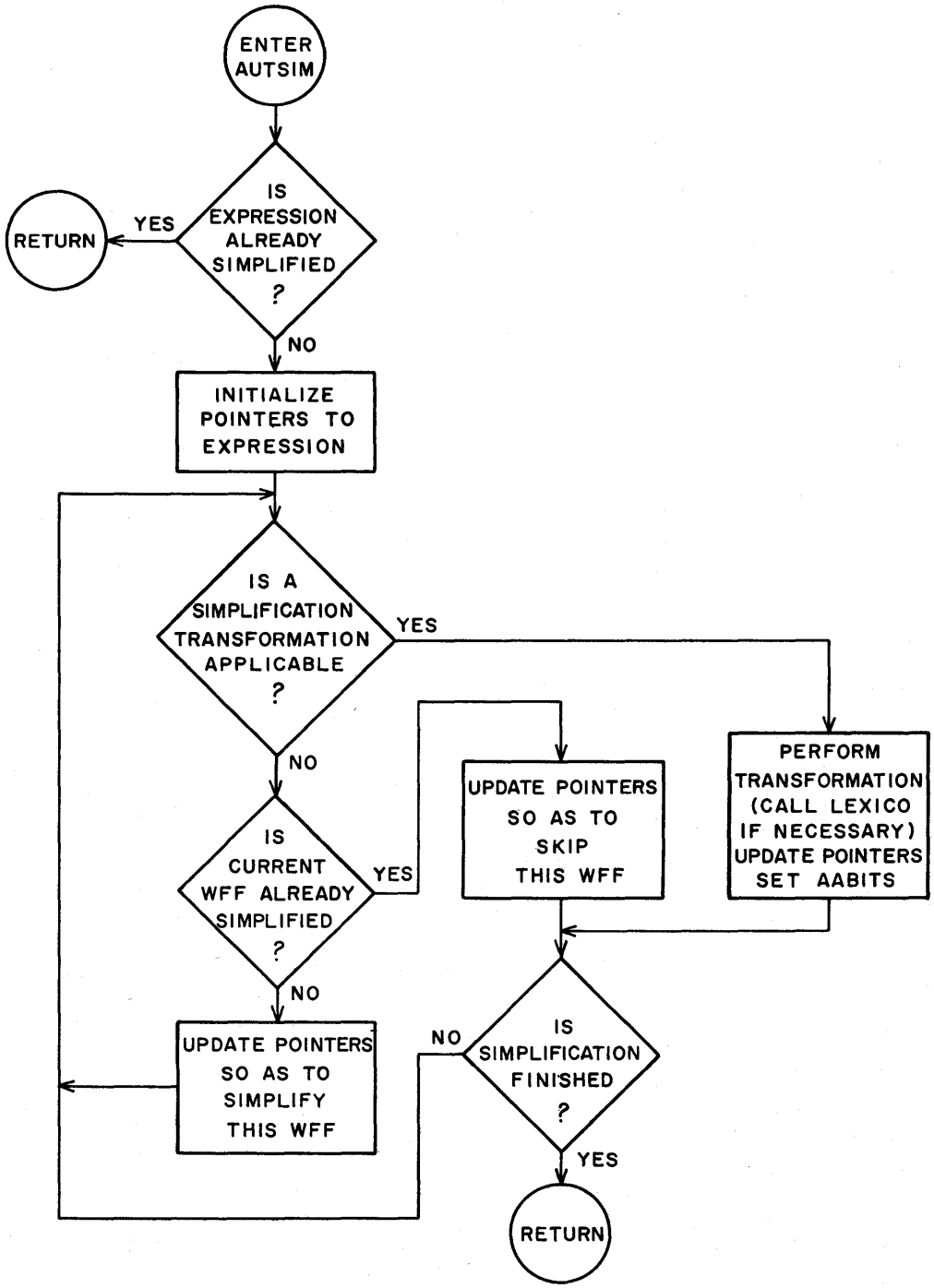



Figure 1. Flow diagram for AUTSIM subroutine.

we have no better definition to offer of intelligible form and simplified form, we maintain that the distinction is significant. It is well to note that even though the FORMAC simplification algorithm is central to the operation of the FORMAC system, code (written in the FORMAC language) designed to reduce expressions to a particular intelligible form is imbedded in many FORMAC user programs.

If the mathematical context within which simplification is to be performed is suitably uncomplicated, there exists a canonical form for all permissible

expressions. For example, there is a canonical form for polynomials in $n$ variables. The existence of such a form can greatly simplify the design of a simplification algorithm.[5] In polynomial manipulation systems,[12,13] simplification is simply the reduction of polynomials to canonical form. The problem of designing a simplification algorithm becomes that of reducing expressions to the canonical form. However, in FORMAC it is possible to generate expressions which when simplified are equivalent but not identical. The fact that, in FORMAC, expansion of expressions is performed only under user option indicates one way in which this can occur. FORMAC will not expand the expression $(a + b)^2$ to yield $a^2 + 2ab + b^2$, unless such expansion is specifically requested by the programmer. A second example of equivalent expressions is provided by the equation

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$

The automatic application of expansion to all expressions or the automatic replacement of tanh $(x)$ by its equivalent would make possible the reduction to a canonical form in these two cases. However, the intelligibility of the expressions being manipulated would be greatly impaired. Indeed, the automatic expansion of expressions would frequently produce the opposite result from that desired by the user. As a result, the FORMAC simplification transformations establish at best a pseudocanonical (p-canonical) form. As will become evident, this p-canonical form makes additional simplifications possible. It also establishes a structural context which can be assumed for all automatically simplified ("autsimmed") expressions; thus it reduces the complexity of the other FORMAC expression manipulation algorithms.

Expression transformations that are candidates for inclusion in a simplification algorithm can be categorized in many ways. There are transformations that contribute directly to the establishment of a p-canonical form. Several transformations embody basic mathematical laws such as the associative, commutative, and distributive laws. There are transformations which are "naturals" and transformations which should be placed under programmer option. Still others, employ basic arithmetic or functional identities. AUTSIM performs transformations that fall into each of these categories. It should be noted, however, that no attempt was made to incorporate trigonometric identities in the FORMAC simplification process. Since the categories indicated above are not mutually exclusive, there is no need to discuss them all. Our discussion of simplification transformations is partitioned as follows: natural transformations, transformations which apply the distributive law, transformations which embody associativity and commutativity, and mathematically undefinable expressions.

In the paragraphs which follow the mathematical transformations are defined in the FORTRAN notation. Letters from the beginning of the alphabet represent arbitrary mathematical expressions. As such, they are often referred to as "well-formed (sub) formulas," or "wffs." The transformations which AUTSIM performs are labelled with letters of the alphabet.

*Natural Transformations*

Consider the following transformations:

(a) $0**A \rightarrow 0 \qquad A \neq 0$

(b) $1**A \rightarrow 1$

(c) $A**0 \rightarrow 1 \qquad A = 0$

(d) $A**1 \rightarrow A$

(e) $(-A)**N \rightarrow \begin{cases} -A**N \text{ if N is an odd integer} \\ A**N \text{ if N is an even integer} \end{cases}$

(f) $-(-A) \rightarrow A$

(g) $EXP(LOG(A)) \rightarrow A.$

(h) $LOG(EXP(A)) \rightarrow A$

(i) $-(3*A*(-B)*C*(-D)) \rightarrow (-3)*A*B*C*D$

(j) $\sum_{j=1}^{n} A_j \rightarrow \sum_{\substack{j=1 \\ j \neq k}}^{n} A_j \qquad \text{Where } A_k = 0$

$$\prod_{j=1}^{m} B_j \rightarrow \prod_{\substack{j=1 \\ j \neq k}}^{m} B_j \qquad \text{where } B_k = 1$$

(k) $\prod_{j=1}^{m} B_j \rightarrow 0 \qquad$ where $k$ exists such that $B_k = 0$

These are "naturals." They are transformations which one usually performs automatically when manipulating a mathematical expression. FORMAC also performs these transformations automatically.

A less clear-cut but natural type of transformation is the evaluation of nonarithmetic operators with constant arguments. For example,

$$X + SIN\ (1.4) \rightarrow X + 0.98545$$

or

$$Y/FAC\ (5) \rightarrow Y/120$$

However, in some applications it is desirable to replace such expression elements with the proper numeric value; in others, it is not. The FORMAC solution to this quandry is to give the programmer control over the automatic evaluation of these expression elements. He has four options from which to choose: (1) all functions are automatically evaluated, (2) only the integer-valued functions (FAC, DFC, and COMB) are evaluated, (3) only the transcendental functions (**, EXP, LOG, SIN, COS, ATAN, TANH) are evaluated, or (4) no functions are evaluated. The first option is the default option.

*Transformations that Apply the Distributive Law*

Two types of "simplification" utilize the distributive law; these are expansion of a product of sums and primitive factoring. Some examples follow:

1. A*(B + C) → A*B + A*C
2. A*(B + C)*(B – C) + A*C**2 → A*B**2 – A*B*C + A*B*C – A*C**2 + A*C**2 → A*B**2
3. (B + C)**3 → B**3 + 3*B**2*C + 3*B*C**2 + C**3
4. A*X + B*X + C*E*X + D*E → (A + B + C*E)*X + D*E

(1) is a simple example of expansion. (2) is an expression which requires expansion as an intermediate step toward complete simplification. (3) illustrates multinominal expansion. (4) is an example of primitive factoring (the coefficient of a single variable, X, has been "factored" out of part of the expression).

Neither expansion or factoring should be applied automatically by a programming system. As in example (2), expansion may, for a given expression, provide the key for further simplification. However, there are expressions for which it inhibits simplification. Consider

$$(A + B + SIN(X))*(A - B) = A^2 - B^2 + A*SIN(X) - B*SIN(X).$$

If this expression is divided by A + B + SIN(X), only in the expanded form will cancellation occur automatically, since FORMAC cancels explicit factors but does not perform factorization to uncover them. Since only the global context of the expression manipulation will in general indicate if expansion or coefficient gathering will lead to desirable results, these transformations are not included among the transformations performed by AUTSIM. The FORMAC commands, EXPAND and COEFF, provide these transformations under programmer option.

*Transformations that Embody Associativity and Commutativity*

The associative and commutative laws for + and * contribute in a fundamental way to the behavior of mathematical expressions. A basic design goal for any simplification algorithm (for a mathematical structure for which these laws hold) must be to incorporate these laws as naturally as possible. As shall be obvious in a moment, these laws have implications for the internal representation of FORMAC expressions, the FORMAC mathematical operator set, and the p-canonical form for expressions —not to mention the simplification transformations.

The associative and commutative laws are assumed to hold for any expression which is generated or manipulated in FORMAC. There are three transformations included in the FORMAC system that establish associativity and prepare the way for the sorting of operands. Such sorting implements the FORMAC assumption that all expressions are commutative. These three transformations which affect the structure of the internal FORMAC p-canonical form, are listed below under (1) and (2) and in the next paragraph under (script 1). (Note: (1) and (2) are not performed by AUTSIM.)

1. A – B → A + (–B).
2. A/B → A*B**(–1).

Transformations (1) and (2) are analogous and accomplish analogous results. The binary inverse operators – and/ are replaced in one instance by a unary – and in the other by a binary**. Under transformation (1), the expression A + B – C + D – E becomes A + B + (–C) + D + (–E); under (2), A*B/C*D/E becomes A*B*C**(–1)*D*E**(–1). The net effect of both transformations is the same, however. The scope of the main operator (+ or *) is made explicit in the expressions and, hence, so is the assumption of associativity. This is an important characteristic of the FORMAC p-canonical form.

Moreover, commutativity of either operator can now be relaized simply by rearranging the well-formed formulas (operands) of the operator. These two transformations are performed in FORMAC by the expression translator, which takes expressions written in FORTRAN infix form and translates them to the internal FORMAC form.

$$(l) \qquad \overset{p}{\underset{j=1}{\Theta}} \rightarrow A_j \qquad \overset{q}{\underset{k=1}{\Theta}} \qquad B_j$$

where

$$A_i = \overset{s}{\underset{m=1}{\Theta}} \quad C_m,$$

$q = p + s - 1$, $B_1 = A_1, \ldots, B_{i-1} = A_{i-1}$, $B_i = C_1, \ldots, B_{i+s-1} = C_s$, $B_{i+s} = A_{i+1}$, $\ldots, B_q = A_p$, and $\Theta$ is either $\Sigma$ or $\Pi$.

The transformation $(l)$ must be included in the automatic simplification algorithm. It is not sufficient to include it only in the expression translator. The FORMAC substitution capability may substitute a sum for a single operand of a sum. Since both expressions are already in internal form, the transformation $(l)$ must be applied by AUTSIM in order to maintain the p-canonical form.

An important consequence of associativity and commutativity is the fact that like terms in a sum and like factors in a product may be combined. This is accomplished in FORMAC by imposing a specific linear order on the operands of these commutative operators.

The linear order is designed so that operands which can combine are equal. Hence, collection of like operands is accomplished by a twofold process: the operands are lexicographically ordered and those operands which will cancel are combined as they are sorted together.

That portion of AUTSIM which orders and combines operands is called LEXICO (lexicographic ordering). This routine is distinct from Autsim proper, because the scan required for sorting is noticeably different from that needed to perform the operator-operator transformations that make up the bulk of the AUTSIM routine. Lexicographic sorting accomplishes two things. In addition to the collection of like factors in a product and like terms in a sum, it contributes to the establishment and maintenance of the p-canonical form.

Examples of the type of combining which one might like to perform follow:

$A - A \rightarrow 0$
$5 * A + (-5) * A \rightarrow 0$
$5 * A + (-2) * A \rightarrow 3 * A$
$(A + B) * C + (-A + B) * C \rightarrow 2 * B * C$

Notice that all of these involve "factoring," the inverse of expansion. In general, they can be represented by $A_1 *B + A_2 *B + \ldots + A_n *B \rightarrow (A_1 + \ldots + A_n) *B$. While it would seem optional to apply such a "factoring" operation in all cases, the following difficulties arise if no restrictions are placed on the $A_i$:

1. In general, factoring of an arbitrary sum $(A_1 + \ldots + A_n)$ would require recursive use of AUTSIM and LEXICO, since it would create a new sum of arbitrary characteristics which would have to be simplified and sorted. This is not a major difficulty, but we hoped to avoid such recursion.

2. Finding B might be difficult if both B and $A_i$ are products. Consider how to factor something like:

$$A_1*B_1*B_2*A_4 + B_1*A_2*B_2*A_4 + B_1*A_2*A_3*B_2 \rightarrow (A_1*A_4 + A_2*A_4 + A_2*A_3)*B_1*B_2.$$

The coefficient of a term like $B_1*A_2*B_2*A_4$ is quite ambiguous when it stands alone. For example, does it represent $B_1$ occurrences of $A_2*B_2*A_4$, or $B_1*A_2*A_4$ occurrences of $B_2$? If one is told that the $A_i$ are parameters and the $B_i$ are variables, then one can say that the term represents $A_2*A_4$ occurrences of $B_1*B_2$. However, LEXICO does not have such information. It would have to determine what the symbolic coefficient of a term is by considering all the terms in the sum. This could be done by obtaining the greatest common divisor of all the terms. The symbolic coefficient of a term would then be taken to be the result obtained by dividing the term by the greatest common divisor. Thus, LEXICO could perform the transportation illustrated above. However, the resulting expression would often not be in the form desired by the FORMAC user.

3. Consider the expression

$$(A + B)*C + D) + (A - B)*(C + D) + (A + B)*(C - D),$$

in which combining of like terms could produce either $2*A*(C + D) + (A + B) * (C - D)$ or $(A + B) *2*C + (A - B) * (C + D)$.

Clearly, the major difficulty of cancellation in sums is the determination of which expressions are "coefficients." The difficulties which can arise in

the case when "coefficient" is a partially ambiguous concept are pointed out in (2), while (3) is an example of a case in which it is completely ambiguous as to which expressions are coefficients and which are not. In order to eliminate such ambiguities while still retaining a useful capability, cancellation was limited to combining numeric coefficients of like symbolic expressions.

In the case of cancellation in products, we wish to combine exponents. Notation removes nearly all of the ambiguity corresponding to the coefficient difficulty in sums, since base and exponent are readily distinguishable in most cases. However, problems can still arise; for example,

```
(A**B)**C*A**D*A**(B*E)*(A**C)**B
= (A**B)**C*A**(D + B*E)*(A**C)**B
= (A**B)**(C + E)*A**D*(A**C)**B
= (A**B)**(2*C + E) * A**D
```

This ambiguity is resolved by the presence of a transformation in AUTSIM which establishes the p-canonical form for nested exponents so that

$$(\ldots((A_1{**}A_2){**}A_3)\ldots{**}A_n) \rightarrow A_1{**}(A_2{*}A_3{*}\ldots A_n).$$

(m) $(A{**}B){**}C \rightarrow A{**}(B{*}C)$.

Because transformation (m) is performed, LEXICO can combine all factors with identical bases, giving a powerful cancellation ability. The expression above is reduced through a sequence of transformations to

$$A\ {**}\ (2{*}B{*}C + B{*}E + D).$$

While performing cancellation in products by combining exponents, LEXICO may create sums which had not previously existed, and which must be simplified. This could have required recursive use of AUSTIM and LEXICO. If the transformation ($LOGA_1 + LOGA_2 + \ldots + LOGA_n \rightarrow LOGA_1{*}\ldots{*}A_n$) were also performed by LEXICO, these two transformations could lead to an arbitrary depth of recursion. Hence, to avoid recursion, that LOG transformation was omitted from AUTSIM and steps were taken to make possible the simplification of the exponent sums by a simple, one-level use of LEXICO and no use of AUTSIM. Some of these steps will be described in the discussion of general flow and techniques of LEXICO. As is often the case, however, hindsight is much clearer than foresight. It appears that a recursive LEXICO would have resulted in both a faster and a more compact simplification package than the tricks used to avoid recursion in the experimental IBM 7090/94 FORMAC system.

In addition to transformation (m), there are several transformations included in the FORMAC simplification procedure for one reason: they contribute to the reduction of expressions to a p-canonical form which extends the simplification possible via combination.

There are three transformations that contribute to the cancellation of like terms in a sum.

(n) $-(A + B + C) \rightarrow -A\ -B\ -C$

(o) $A + LOG(B_1{*}\ldots{*}B_s) + C \rightarrow A + LOG(B_1) + \ldots + LOG(B_s) + C$

(p) $LOG(A{**}B) \rightarrow B{*}LOG(A)$

The last two transformations, (o) and (p), prepare for the collection of logarithmic terms in a sum. For example,

```
LOG(A) + LOG(B*A**(-2))
LOG(A) + LOG(B) + LOG(A**(-2))
LOG(A) + LOG(B) + (-2)*LOG(A)
```

Lexicographic ordering, accompanied by collapsing of like terms can then produce the simplified result, – LOG (A) + LOG (B). The transformation (n) is a trivial application of the distributive law. It makes possible the reduction $A + D - (A + B) \rightarrow (A + D - A - B \rightarrow (A - A) - B + D \rightarrow - B + D$. Two transformations perpare the way for the collection of like factors in a product, transformations (m) and (q).

(q) $(A_1{*}\ldots{*}A_n){**}C \rightarrow A_1{**}C{*}\ldots{*}A_n{**}C$.

The FORMAC p-canonical form is further reflected by these transformations. Factors of a product are maintained as distinct bases raised to powers; they are not gathered together as products (of those bases which have equal exponents) raised to that power. The form $(A_1{*}A_2){**}B_1 * (A_2{*}A_3{*}A_4){**}B_2 * (A_1{*}A_4){**}(-B_1)$ becomes $A_1{**}B_1 * A_2{**}B_1 * A_2{**}B_2 * A_3{**}B_2 * A_4{**}B_2 * A_1{**}(-B_1) * A_4{**}(-B_1)$. Then LEXICO can sort and collect like factors producing

$$A_2{**}(B_1 + B_2) * A_3{**}B_2 * A_4{**}(-B_1 + B_2).$$

*Mathematically Undefined Expressions*

There are three mathematically undefined expressions which may be introduced into FORMAC expressions. These are

1. 0**(–a) (a is a positive number),
2. LOG(0), and
3. 0**0.

In FORMAC, the first two expressions are evaluated as they would be by FORTRAN; i.e., each is replaced by 0 and a suitable message written on the output listing. The third expression is left intact so that the programmer may substitute a variable or expression of his own choice for it. These are neither consistent nor aesthetically pleasing solutions. Alternative approaches to this problem will be discussed in a later paper. It is the intent of the authors to publish a subsequent paper that will contain flow diagrams with sufficient detail to simulate the AUTSIM algorithm. In such a context, it will be possible to consider adequately the ramifications of various additions and changes to the AUTSIM algorithm.

## THE FORMAC SIMPLIFICATION ALGORITHM

This section is an introductory description of the FORMAC automatic simplification algorithm. There are three subsections: FORMAC Internal Expression Representatation, Details of the AUTSIM Scan, and The Organization of LEXICO.

### *Details of Expression Representation*

Any mathematical expression manipulation system must operate on expressions interpretively, since the form of an expression may change constantly as it is manipulated. An efficient internal coding for mathematical expressions is essential. There are two well-known notations for mathematical expressions: commonly used infix, and classical Polish notation. The unwieldiness of infix notation and the difficulties and inefficiencies it presents for algorithm design are well known. The use of prefix Polish notation overcomes many of these problems, and "Cambridge" Polish, introduced in reference 14, provides space and algorithm economies not provided by classical binary Polish.

"Delimiter Polish" is the form used to encode mathematical expressions in FORMAC. This can be thought of as classical Prefix notation, permitting unary, binary, and variary operators. Of the arithmetic operators, only "+" and "*" are variary. The scope of range of a variary operator is defined by a delimiter, "]". Hence, the Polish string + A B C D ] represents the expression A + B + C + D; the Polish string

+ A * BC] D 5]

represents the expression A + B * C + C + 5; and the Polish string

+ A * BCD5]]

represents the expression A + B * C * D * 5.

The introduction of the variary operators makes meaningful the application of transformation $l$ to a Polish string; hence, the implementation of both associativity and commutativity is simplified by this notation. Moreover, fewer symbols are required to represent a sum of product; less space is required for the internal representation of expressions.

The entire FORMAC operator set is displayed in Table 2. The differentiation operator requires a further word of explanation; it is also variary. In delimiter Polish,

DIF f x 1] represents $\frac{df}{dx}$

and DIF f x 2 y 1 z 3 ] represents $\frac{\partial^6}{\partial^2 x \, \partial y \, \partial^3 z}(f)$.

Note that, for the sake of clarity, the power operator ** will be represented by the symbol "↑" in delimited Polish expressions.

### *Details of the AUTSIM Scan*

*Governing Operators.* The key to the basic scan of the AUTSIM routine is the decision process that determines whether a "simplifying" transformation is applicable to that part of the expression currently being scanned. This decision process is based on the concept of governing operators. Every operator acts upon or governs its operands, and each operator (with the exception of the outermost) is governed by a higher level operator. In the example

* A + BC] D]

the outermost operator is the * which governs the + operator. Or in other words the + operator is governed by the *. The applicability of most simplification transformations can be determined from a simple transfer table based upon the juxtaposition of governing and governed operators.

*Contextual Checking.* For some of the transformations it is necessary to do extra contextual checking

to eliminate unnecessary or unwanted applications of the transformation. There are essentially three types of context which may be checked before applying a transformation:

1. The first type of checking tests for a specific pattern of operands or operators. For example, the simplification of ↑ -BK is only done if k is an integer, but the po ssible applicability of the transformation is recognized by the combination of the and the - sign.
2. A second type of context checking tests to see if the governed subexpression has already been simplified.
3. The final type of checking tests the system switches to determine if the transformation should be performed. The evaluation of nonarithmetic operators with constant arguments is an example of this type of checking, since it is done only if the proper switches are set. In some cases the contextual information is discovered before it is needed. This contextual information is associated with certain operators such as ↕, COMB, and * through flags which indicate the status of their operands.

Certain transformations are not applied when first recognized by the transfer table. These transformations are delayed until the governed subexpressions are simplified. This is done for two reasons. First the transformation may disappear when the subexpression is simplified. For example, the transformation (EXP(A))**B → EXP(A*B) is delayed because A might be the expression LOG C in which case (EXP (LOG C))**B → C**B. This reduction could not occur if the transformation were not delayed. The second reason is that an unnecessary intermediate growth in expression size results if certain size-increasing transformations are not delayed. An example of this type of transformation is the transformation $(A_1* \ldots *A_n)**C \rightarrow A_1**C \ldots A_n**C$. It is easy to see that the size of the right-hand expression is greater than that of the left-hand one since the exponent occurs n times. If the product is simplified before applying the transformation, then the number of replications of C is less than or equal to the number required for the transformation acting on an unsimplified product.

*The AA Bit.* The fact that an operator and its operands have been simplified is indicated by the AA bit (Already AUTSIMed bit). In the delimiter Polish notation the AA bit is indicated by a dot over the operator. However, constants and delimiters do not have an AA bit. For example, in the expression

$$+ \ A \ *\dot{B}\dot{C}5] \uparrow A - 1]$$

the subexpression *B̈C̈5] is simplified as indicated by the dot over the *. There are AA bits on B and C, the operands of the *, since in some instances the lead * can be removed by a transformation but the operands are still simplified. If the expression was

$$\text{LOG } *BC5]$$

then this can be transformed to

$$+ \ \text{LOG } \dot{B} \ \text{LOG } \dot{C} \ \text{LOG } 5]$$

and B and C would remain simplified and require no further simplification.

The AA bit has an addition function. It indicates that a subexpression has already been scanned for transformations, so only transformations which involved the lead operator of the expression are still applicable. The AA bit prevents the scan for transformations from oscillating indefinitely.

The FORMAC command subroutines do not reset the AA bit on expressions they manipulate, unless they actually alter the expression so that it is no longer simplified. Then, the AA bit is reset on the expression but not on those wffs of the expression which have not been changed. In this manner, the entire FORMAC object-time system operates to minimize redundant simplification.

*The Scan Pointers.* As the expression is scanned, the location of the governing and governed operators constantly changes. The current governing and governed operators are determined by the two scan pointers P and C. The pointer C points to the symbol which is currently being scanned. The pointer P points to the operator which governs the current symbol. Consider an example. Suppose that the scan is currently looking at the following expression fragment.

$$\begin{array}{c} \text{PC} \\ \downarrow \ \downarrow \\ \ldots + \dot{*}\dot{X}\dot{Y}] * \updownarrow Z2W] \ 5] \ldots \end{array}$$

The current symbol is the first * operator. The AUTSIM transfer table shows that no transformation is done for + * so the scan pointer must be moved. The AA bit on the first * indicates that no

transformations remain in that operand so that the scan moves to the next operand. The C pointer is updated to point to the second operand under the operator,

```
        P     C
        ↕     ↕
. . . + *X Y ] * ↕ Z2W] 5] . . .
```

Now the current symbol is the second * operator. As before, no transformation can be applied so the scan pointer is moved. However, in this case the product is not simplified so it must be scanned for possible transformations. Therefore, the scan pointer moves to the next symbol, the ↕ operator. But the ↕ is governed by the * operator so the pointer P must be moved to the new governing operator. After the scan pointer is moved the expression looks like this:

```
          P C
          ↕ ↕
. . . + *XY] * ↕ Z2W] 5] . . .
```

The scan pointers continue to move in this fashion. After a transformation is performed the pointers must be updated so that they will be properly positioned to continue the scan. At each step in the scan the current operand of the current governing operator is simplified before the next operand is scanned. This method of scanning means that the current symbol pointer, C, oscillates back and forth along the delimiter Polish string as the expression is simplified.

*The Push-Down* List. In the example above, the pointer P was moved from the + to the second * when the second product was being simplified. After that simplification is completed, the C pointer will have returned to the * and the AA bits will have been set.

```
            C
            ↕
. . . *XY] * ↕ Z2W] 5]
```

But the P pointer must be reset to the operator which governs the *, namely the + operator. Therefore, whenever the scan pointer is moved to scan an operand, it is necessary to save a pointer to the old governing operator and any flags associated with it. This information can then be used to reset P to the + operator. Since each operand is in itself a well-formed expression, a natural method for saving the current status of an operator is to make the AUTSIM routine recursive. However, for reasons of efficiency, the AUTSIM algorithm uses a push-down list (PDL) to save the simplification and scan status. The top entry on the PDL always indicates the current governing operator so that the P pointer is just another name for this entry.

*LEXICO and Expand.* Two major departures are made from the AUTSIM scan in order to perform large transformations. The first of these, LEXICO, is described in detail in the next section. The second major departure is under control of the EXPFLAG. If it is set on either a * or ↕ operator, then the expand algorithm is invoked to remove parentheses.

We have seen that the AUTSIM scan is essentially oscillatory in nature under the control of a push-down store, a current symbol pointer, and AA bits. A systems level flow diagram for AUTSIM is provided in Fig. 1.

### *The Organization of LEXICO*

The general flow of the LEXICO subroutine is displayed in Fig. 2. There are three major blocks in LEXICO: the COMPARE routine, the COMBINE routine, and the CLEANUP section. Although considerations of efficiency led to an intermixing of the code, the routines are most easily understood as separate entities. The remainder of this section will treat each of these blocks. The requirements placed on each unit, the motivation for imposing these requirements, and the manner in which these requirements are met are discussed.

*The COMPARE Routine.* Because the internal FORMAC representation is not designed specifically for sorting, a special sorting list (BWFF) is constructed from the input sum or product which is being sorted. Each subexpression under the + or * is handled separately and the sorting list is built up as each of the sub-wff's is sorted against the sub-wff's already on the sorting list (see Fig. 2). The heart of the sort is the routine which compares the sub-wff being sorted (AWFF) with the $i^{th}$ sub-wff on the sort list (BWFF(I)). We shall discuss the criteria for a sorting order and sketch the actual sorting order employed by LEXICO.

*(a) Criteria for a Sorting Order.* The description of the comparison routine would be difficult to understand without a description of the sorting or-

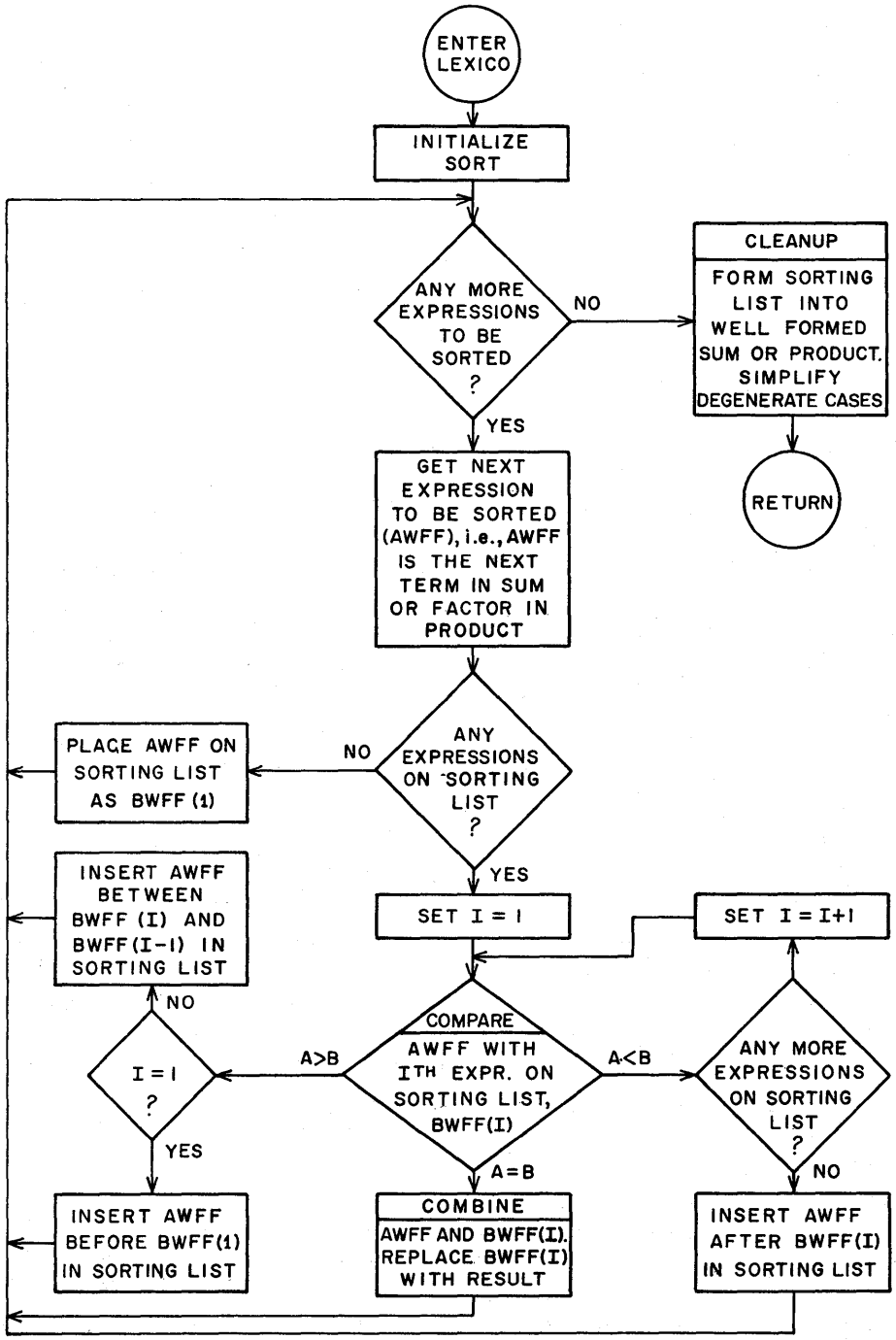



Figure 2. LEXICO flow diagram.

der which it is supposed to implement. The particular sorting order used in LEXICO was chosen to meet a wide range of requirements.

The first requirement of a method of sorting is that the sorted result be unique. That is, the result of the sort does not depend on the original order of the subexpressions it sorts. In this case, the sort yields a linear or total ordering. If the comparison of any two expressions gives a definite ordering for these two expressions and if this ordering is transitive, then the ordering is total. If this were the only requirement, expressions which are represented by linear strings of symbols, could be sorted by the left-to-right comparison used in dictionaries. All

that is needed for this technique is a linear ordering of the possible symbols, similar to that given to the letters of the alphabet ("alphabetical ordering").

However, LEXICO must "combine" similar expressions; e.e., it must perform cancellation. This imposes the requirement that expressions which can be combined by cancellation should have almost identical ordering properties, so that, during the sort, the routine will compare them with one another and realize that cancellation is possible.

Specifically, the following classes of expressions must have nearly dinetical sorting characteristics:

1. explicit products that differ only by a constant (numeric coefficient, e.g., * A B C $K_1$] and * A B C $K_2$ are numbers;
2. a nonproduct expression and a constant multiple of that expression, e.g., SI X and * SIN X $K_1$];
3. a negative expression and the product of that expression and –1, e.g., –SIN X and * SIN X –1] or * A B C –1] and – * A B C];
4. exponentiated expressions that differ only in their exponents, e.g., ↕ X Y and ↕ X + W Z] or ↕ + A B COS Y] 2 and ↕ + A B COS Y] Q or EXP + X Y – D] and EXP ATAN Q;
5. an expression and that expression raised to a power, e.g., SIN + X Y] and ↕ SIN + X Y] * A B C].

Note that since we have no unary operator denoting the multiplicative inverse, we avoid the problem analogous to (3) for exponents.

Combining depends upon recognizing these special cases. They involve differences in top-level structure; the only information required concerning the lower-level structures is whether or not they are identical. It would have been possible to design a routine that checked for these special cases on the top level, and used a simple dictionary ordering to compare subexpressions of the terms or factors it was sorting. For example, if the minus sign (-) came after all variables in the linear ordering of symbols, it would be possible to design a routine in which A and -A sorted almost identically as terms in a sum (and hence could be canceled) but in which EXP(A) and EXP(B) sorted closer together than EXP(A) and EXP(-A). However, it was decided that the sorting order would be based on levels and would be consistent on all levels. That is, identical functions (sums, products, and exponentiated wffs are considered functions in this sense) would be compared on the basis of their arguments, and the arguments would be compared in exactly the same way as arbitrary terms in a sum or factors in a product. Thus, to compare (COS(A) and COS (B), the routine would note that the functions were the same and then compare the arguments A and B exactly as it would have if they were terms or factors to be sorted.

The above sorting specifications are clear cut. In addition to these, however, there were two less well-defined requirements placed on the sorting order. Since automatic simplification is performed on all expressions in the system, and in particular on all expressions that are to be written out, a criterion of intelligibility was imposed. That is, within the limits of the other requirements we felt that the ordering induced by LEXICO should produce output that is in some sense"understandable" to the programmer. It should produce expressions which are similar in appearance to ones which the program user might write—given the limitations of the character set and printing techniques available. To achieve this goal without sacrificing the requirements of the internal p-canonical form, a double approach was taken. The sorting order itself was designed to be as "understandable" as possible, and a way of modifying it prior to output (the ORDER command) was included in the system.

The final requirement was that the ordering sould be as "simple" as possible. This served the double purpose of making it understandable to the user and as easy to implement as possible. AUTSIM is the most active major subroutine in the FORMAC system. Every expression created is sent through AUTSIM at least once, and often several times. AUTSIM may call LEXICO many times in the process of simplifying a single expression. Thus efficiency is a prime consideration in the design of LEXICO. Any increase in the speed of LEXICO produces a noticeable increase in the system's performance. In this context, the suggestions made by Martin[5] may prove quite significant.

*(b) The Sorting Order.* With these requirements in mind, we now consider the sorting order. Symbols other than +, *, ↕, and - are linearly ordered in the following manner. Atomic variables come before all other symbols, and are ordered among themselves by the numerical value of the core location of their symbol-table word. Since the symbol table for each routine is placed in core with the

variables in alphabetic order and with arrays in standard FORTRAN order, this results in variables defined in the same routine being sorted in straightforward alphabetical order by name, with array elements in their normal order. However, variables defined in different routines sort according to the placement of their defining routines in core, which means the actual order in which variables are sorted is dependent on the order in which subroutines are loaded.

Immediately following the atomic variables in the linear ordering of symbols are the operators in the order SIN, COS, ATAN, TANH, EXP, LOG, FAC, DFAC, COMB, DIF. This ordering is arbitrary and is determined by the bit patterns assigned to each operator in the internal expression coding. Following the operators are the constants. Constans are ordered among themselves by their numerical value. The algebraically smaller of two constants preceeds the larger one. Since all constants in expressions have been coverted to the same mode (rational or floating point) by AUTSIM, there is no possibility of comparing constants of differing mode.

As may be noted, the symbols +, *, ↕, and – are not compared with other symbols. Instead they cause the scan routine to compare subexpressions of the expressions AWFF and BWFF (I) and use that result as the final comparison. Only if all indicated comparisons result in identical matches, does the fact that a +, *, ↕, or – appeared directly influence the result of the sort. This is done in order to meet the constraint that items, which can combine with one another, sort together.

Once the sorting order has been specified and understood, the structure of the comparison routine becomes quite clear. The routine represented by the COMPARE diamond in Fig. 2 is basically two leveled. The lower level is a routine that performs a straightforward comparison of the two expressions it receives as arguments. As output, it indicates the relative ordering of the two expressions. The upper-level routine sends parts of expressions to the lower level, and uses the finromation returned to determine if expressions can be canceled or combined. It then passes control as indicated in the flow chart.

*The COMBINE Routine.* The COMBINE routine consists of a basic cancellation routine and an arithmetic routine. Cancellation is accomplished by several small routines that perform the various tasks needed for cancellation in sums and products. Under sums the routine adds the coefficients of the two terms and produces a new term with the sum as coefficient. This is made somewhat more complex by the problem of determining the true coefficients. The routine must act as if explicit coefficients of 1 and –1 occurred instead of implicit occurrences; e.g., as if * ABC] were *ABC1], –A were *A –1] and – * ABC] were *ABC –1].

Other difficulties can arise in finding the constant coefficients of an expression. The numerical coefficients of products appear as the final constant before the delimiter (since all products under sums have been put into canonical form). But, because the expressions are in Polish notation, it is possible that a constant appearing in that position may be the argument of a preceding function. This is difficult to check in the case of rational constants which may appear in the form . . . $K_1$ $K_2$ –1 . . . under a product. In this case, given a product ending in the string . . . $K_1$ $K_2$ –1], it is possible that either $K_1$ or both $K_1$ and ↕ $K_2$ –1 are arguments of preceding functions and not coefficients.

If the constants to be combined are rational-mode constants, arithmetic must be performed by special routines which add fractions and produce a reduced fraction. Finally, if the result is either 0, 1, or -1, LEXICO must duplicate the AUTSIM transformations involving such coefficients (e.g., deleting the entire produce in the case of 0, or deleting the constant and prefixing the expression with a minus sign in the case of -1). This was done in LEXICO for efficiency as LEXICO could easily check for such results at the time of the addition, while UTSIM would have to make an entire rescan. This does, however, result in duplication of code in the two routines as they are currently written. This is an example of the old problem of space efficiency versus time efficiency.

Combination in products requires the addition of exponents. Since exponents may be arbitrary expressions, the resulting sums must in turn be sorted and simplified. This is done by permitting the product cancellation routine to use the rest of LEXICO in a pseudorecursive fashion. Whenever it is necessary to cancel exponents, the expressions involved are modified so that the first expression (the one on the product sorting list) is no longer a well-formed formula in internal form. Instead, its expo-

nent is transformed into a sorting list of the type used for sorting sums. The comparison and sum cancellation routines are then used to sort the terms of the second exponent onto the sorting list, performing cancellations where possible. If the exponent is not a sum, then only a single use of the sort and combine routines is needed. However, if the exponent is a sum, then the sort and combine section is used once for each wff under the plus, and the plus is discarded because the sort list is already a sum. Of course, since one section of the routine is being called by another section, care must be taken to preserve and restore the status of all internal switches and registers that are modified. When all factors in a product have been sorted and all exponent cancellations have been completed, it is up to the cleanup section to take the modified expressions and restore them to normal form. This may involve further simplification, as the exponent which finally remains may be either 0 or 1.

The arithmetic routines are called upon to do almost all the arithmetic required in LEXICO. These include the routines that are needed for computing the constant term in a sum and the constant coefficient in a product and the routines for cancellation. Basically, these routines perform addition and multiplication of floating-point and rational-mode constants. The floating-point package is straightforward, with standard checks for overflow and underflow. The decision to implement rational mode was made after the system design had been frozen. In particular, it was impossible to add any new operators since many routines had been coded making explicit use of the originally defined operator set. Thus, it was necessary to represent fractions in the form *$K_1$ ↕ $K_2$ –1 rather than by a binary rational constant operator, RC $K_1$ $K_2$. This introduced several complications for the LEXICO scan.

The rational-mode arithmetic package is more complicated than the floating-point package. Multiplication requires two integer multiplications, and addition requires three multiplications and one addition. The major difficulty arises in controlling overflow. In the experimental 7090/94 FORMAC, both the numerator and denominator are limited to integers less than or equal in magnitude to $2^{36}$ -1. Thus, with a moderately large denominator, it is very easy to have a numerator overflow.

In particular, it is possible to create intermediate results that require double-precision arithmetic, although the final results are small enough to be represented by single-precision fractions. Therefore, the rational arithmetic routine contains a double-precision, fixed-point add routine, in addition to a routine that employs the Euclidean algorithm, for reducing fractions to lowest terms.

*The CLEANUP Routine.* The cleanup action of LEXICO converts the sort list form of the expression (BWFF) back to normal FORMAC internal form. Several transformations are performed in the process of reconstructing the expression. The order of the items on the BWFF list is inverted and a simplified sum or product is created. If the result is a product, the parity bit is tested to determine if the product has a negative sign. If parity is negative, the minus sign is included in the constant if it exists. The minus operator is inserted preceding the product, if there is no constant factor. If the sort list is empty and there is no constant, then a sum is replaced by 0 and a produce is replaced by + 1, or - 1. If the product or sum has only one argument, then the operator (* and +) and its delimiter are not appended and the single wff is returned.

## SUMMARY

The central role of simplification in the FORMAC programming system and the general approach pursued in implementing the FORMAC automatic simplification algorithm have been described. It has been shown how the universal application of associativity, commutativity, and properties of the additive and multiplicative identityelements (0 and 1) in conjunction with the establishment of a p-canonical form can produce "simplified" expressions. In addition, the need for placing application of the distributive law under programmer option has been indicated. The basic principles employed in the organization of the simplification algorithms (AUTSIM) have been presented in detail. In particular, the role of governing relationships between operators, the need for additional contextual information, the movement of scan pointers and the organization of the sorting routine (LEXICO) has been indicated. The importance of an already simplified flag is completely eliminating redundant simplification has been stressed.

This paper has introduced the FORMAC approach to the automatic simplification of mathematical expressions. A subsequent paper is planned in

which the simplification algorithm will be presented in complete detail.

## REFERENCES

1. J. E. Sammet and E. R. Bond, "Introduction to FORMAC," *IEEE Transactions on Electronic Computers,* vol. EC-13, no. 4, p. 386 (Aug. 1964)

2. E. Bond, et al, "FORMAC—An Experimental FORmula MAnipulation Compiler," *Proceedings of the ACM National Conference,* August 1964.

3. "FORMAC," *SHARE General Program Library,* 7090 R2 IBM 0016, IBM Program Information Department, White Plains, N. Y.

4. E. Bond, et al, "Implementation of FORMAC," IBM Technical Report 00.1260 (Mar. 16, 1965).

5. "Symbolic Work on High Speed Computers," Dartmouth Mathematics Project, Project Report No. 4 (June 1959).

6. D. J. Edwards, "Symbolic Circuit Analysis with the 704 Electronic Computer," MIT B.S. Thesis, 1959.

7. S. H. Goldberg, "Solution of an Electrical Network Using a Digital Computer," MIT Master's Thesis, 1959.

8. K. Maling, "The LISP Differentiation Demonstration Program," MIT Artificial Intelligence Project, Memo 10.

9. T. Hart, "Simplify," MIT Artificial Intelligence Project, Memo 27 (1961).

10. D. Wooldridge, "An Algebraic Simplify Program in LISP," Stanford Artificial Intelligence Project, Memo 11 (Dec. 1963).

11. P. Sconzo, A. R. LeSchack and R. Tobey, "Symbolic Computation of $f$ and $g$ Series by Computer," *Astronomical Journal,* vol. 70 (May 1965).

12. W. S. Brown, "The ALPAK System for Nonnumerical Algebra on a Digital Computer," *Bell System Technical Journal,* vol. XLII, no. 5 (Sept. 1963).

13. A. R. M. Rom, "Manipulation of Algebraic Expressions," *Communications of the ACM,* Sept. 4, 1961.

14. J. McCarthy et al, "LISP Programmer's Manual," MIT Press, Mar. 1960.

15. W. A. Martin, "Hash-Coding Functions of a Complex Variable," MIT Artificial Intelligence Project, Memo 70 (June 1964).

# THE NEW BLOCK DIAGRAM COMPILER FOR SIMULATION OF SAMPLED-DATA SYSTEMS

B. J. Karafin
*Bell Telephone Laboratories, Incorporated*
*Murray, Hill, New Jersey*

## INTRODUCTION

The block diagram compiler known as BLØDI was put into use at Bell Laboratories in 1959 and reported in the Bell System Technical Journal in 1961.[1] The compiler has been completely rewritten to provide substantially increased flexibility. The new program is called BLØDIB.

## BLØDI

BLØDI was written to aid in the simulation of sampled-data systems. It accepts as input a description of a sampled-data s ystem block diagram written in the BLØDI language. It produces, as output, a machine language program which will simulate the described system. The major asset of BLØDI is that the language corresponds very closely to an engineer's block diagram. It is easily learned and used even by people with the most superficial knowledge of computing techniques.

The systems with which the compiler can deal are combinations of sampled-data circuits, i.e., of blocks that accept pulses as input and yield pulses as output. These pulses vary in height, but they all occur at multiples of a fixed clock time. Thus the systems the compiler accepts are digital in the strictest sense of the word. The blocks comprising the system perform the only functions a digital computer can perform, namely, accept a number as an input, operate on it, and produce a number as an output.

Continuous systems that are sufficiently band-limited may also be simulated. However, a sampled-data system must be designed whose output pulses would correspond to the sample values of the desired system. This is easily done if there is a highest system frequency, and if it is practical to have a sampling rate higher than twice this highest frequency. For systems that do not meet these requirements, approximations must be made if the system is to be simulated using BLØDI. Techniques and computer programs[2,3] are available at Bell Laboratories which produce efficient sampled-data approximations to a wide class of continuous transfer functions. These programs are even capable of producing punched cards suitable for BLØDI input. The important point here is that BLØDI makes no pretense of being a continuous system simulator. The transformations and possibly approximations necessary in simulating a continuous system with a discrete system are left to the engineer.

It should be clear that BLØDI is not a digital imitation of an analog computer. In its most normal

mode of operation, it permits the user to go directly from a block diagram of a sampled-data system involving transfer functions, etc., to the object simulation program without explicitly considering the underlying differential equations. A myriad of programs exist that do try to make the digital computer behave like an analog computer.[4,5] These block diagram compilers are built around an integrator block; the diagrams they accept are not so much system block diagrams as they are block representations of analog computer programs. BLØDI is system oriented. In a paper describing BLØDIB applications,[6] simulation of a vocoder is discussed. In that problem the interest is in system performance and optimization. The system is so large and so complicated that asking for the differential equiations is an unrealistic question. The system is simulated using digitized speech as input data. The output is also digitized speech which is listened to and subjectively judged.

The BLØDI language is used to describe the system block diagram. Each block of the block diagram is represented by one list, i.e., by one punched card. (If the description of the block is lengthy, it may spill over onto more than one card.) The coder chooses block types from a dictionary of about 40 types. This dictionary includes such blocks as delay lines, amplifiers, transversal filters, rectifiers, cosine generators, function generators, etc. Table 1 shows the type dictionary. The list for each block specifies its type, a name assigned by the coder, parameters associated with it, and the names of the blocks to which the signal from the present block is connected. An example of a block-specifying list is

RAISE AMP 13.7, QUANT, SUM/3

The block named RAISE is an amplifier with a gain of 13.7. It feeds a block called QUANT and the third terminal of a multi-input-terminal block called SUM. (The various inputs to multi-input-terminal blocks are specified by a slash followed by the terminal number.) It should be noted that the user need specify only one half the connection matrix, namely the outputs. The compiler internally fills out the matrix and reports unusual circumstances by way of diagnostics.

One of the most pleasant features of the language is the fact that the order of appearance of the various lists in a circuit description is immaterial. The blocks may be described in any order; the compiler analyzes the connection matrix and internally orders the blocks for processing.

Figure 1 is a skeleton form of a PCM television transmission scheme[7] involving quantization with error feedback. Consider the BLØDI description of this system which is given below.

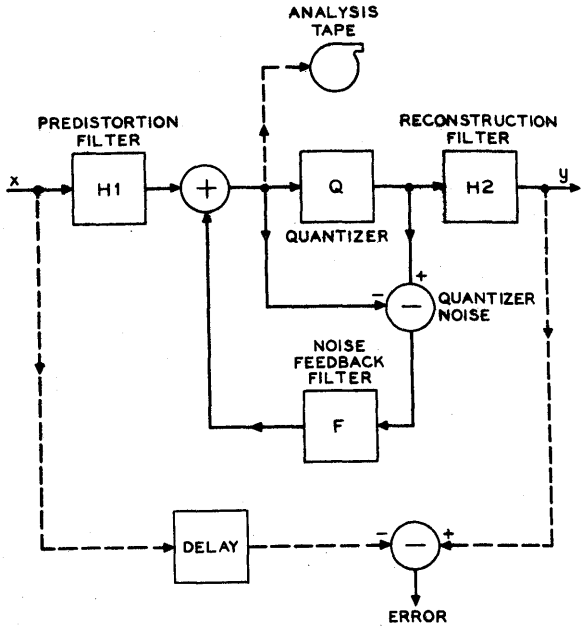



Figure 1. Transmission scheme involving quantization with error feedback.

```
INPUT    INP    5,,501,1,,,12,H1
H1       FLT    2,1,–0.502,1.0, SUMM/1
SUMM     ADR    QUANT, NØISE/2
QUANT    QNT    8,120,200,280, (etc.),
                H2, NØISE/1
NØISE    SUB    FBDEL
FBDEL    DEL    1, F
F        AMP    –0.96,,SUMM/2

H2       ACC    0.502,,ØUTPT
ØUTPT    ØUT    5,,501,1,,,12
         END
```

Each line of the above represents one block of the system diagram of Fig. 1. For each block the left hand column contains the name assigned the block, the center column specifies the type of the block, and the right column specifies the parameters and output connections. Strictly speaking, INP and OUT are not functional blocks. They specify points in the system were samples are read from and written onto designated tape units respectively.

END is also not a type; it specifies the end of a circuit description.

In the example we again point out that the process is one of a system simulation. Both the input and output are tapes of digital television picture data.

## BLØDIB

Using the BLØDI language as a foundation, a new compiler, BLØDIB, has been written. The new compiler provides a language that is more flexible and a programming system that is more complete. Whereas the old compiler offered a fixed source language and produced object programs of one rigid structure (main programs all of whose arguments were explicitly numeric), BLØDIB offers the engineer some of the generalities available to users of general purpose programming systems.

With the new compiler, simulation programs (the result of BLØDIB compilations) are more like the programs produced by other more general compilers. For example, the user may now choose to have the simulation run in either the integer or floating point mode. (The old program ran only in the integer mode.) Furthermore, simulation programs may now have symbolic parameters, the values of which can be supplied at run time and varied to facilitate optimization routines and collections of families of data. Most important too is that simulation routines may now communicate with other programs in the computational environment. These features open the way for many new applications, some of which are discussed below.

The source language also has new flexibility and offers new potentialitis. The changes impart roughly the same flexibility to BLØDI as MACRØ FAP does to FAP. Along the providing features to make the coding of simulation problems more convenient and less tedious, the new program gives the user the ability to write higher-level, specialized, simulation languages using BLØDI statements as atomic blocks.

The compiler itself has a new structure. BLØDIB is actually a preprocessor coupled with a library of high-level macro definitions. This structure provides general flexibility, lends itself more readily to changes and additions, and is somewhat less dependent on particular monitor systems and machines.

For purposes of discussion, BLØDIB can be thought of as providing the capabilities for system simulation coding that is

- evolutionary,
- compatible, and
- modular.

Each of these categories will be discussed separately.

### *Evolutionary Programming ( SUPERs )*

It may happen that several interconnected BLØDI blocks are required to realize a single functional block that will be used many times, perhaps with several different values for some group of parameters. What is called the SUPER facility of BLØDIB essentially permits the programmer to draw a line around such a group of blocks, to name it, and to use it thereafter as if it were a basic block. Figure 2 shows the block diagram of what can be thought of as a rectangular integrator. To define a SUPER to realize this function, coding of the following form is given to the compiler as part of the source description.

```
INT      MACRØ   INCØN
INPUT    MIP     1, INTRV
INTRV    AMP     1.0 E–5,,SLØW
SLØW     DEL     1,SUMM/1
VALUE    BAT     INCØN, DELAY, SUB/1
DELAY    DEL     1, SUB/2
SUB      SUB     SUMM/2
SUMM     ADR     RECT
RECT     ACC     1.0,,ØUTPT
ØUTPT    MØT
         END
```

After defining INT in this way, it can be used as if it were a basic block type with one parameter (the initial value). (In the example the integration interval, which corresponds to the sampling period, is $10^{-5}$.)

We digress to point out that although BLØDIB is not primarily an analog computer simulator, the preceding definition of a rectangular integrator shows that the compiler has at least as much power to tackle analog computer problems as some compilers that are so oriented. An example of such a compiler is DAS[4], which uses rectangular integration and a fixed time base.

Since a SUPER may be used as a basic block after it has been defined, it can be used in the description of a more complicated SUPER. A trival example is shown where the previously defined INT block is used in the definition of a new super that

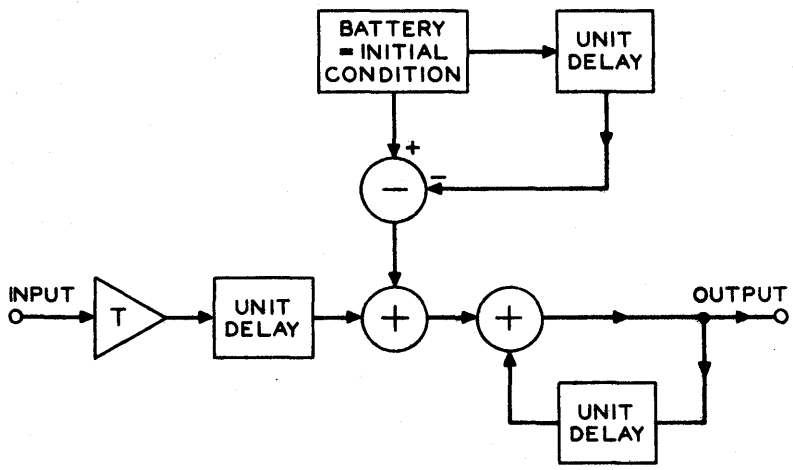



Figure 2. A BLØDIB rectangular integrator.

rectangularly integrates the weighted sum of four inputs.

```
SIN      MACRØ    W1, W2, W3, W4, INCØN
IN1      MIP      1, WGHT1
IN2      MIP      2, WGHT2
IN3      MIP      3, WGHT3
IN4      MIP      4, WGHT4
WGHT1    AMP      W1,,SUM/1
WGHT2    AMP      W2,,SUM/2
WGHT3    AMP      W3,,SUM/3
WGHT4    AMP      W4,,SUM/4
SUM      ADR      INTEG
INTEG    INT      1,INCØN,ØUT
ØUT      MØT
         END
```

In BLØDIB, SUPERs may be nested to an almost arbitrary number of levels. Since each SUPER is very nearly an independent circuit description, a BLØDIB program can be thought of as an ordered sequence of circuit descriptions. Each circuit possibly contains one or more of the previously defined circuits one or more times. The last circuit is of course the one to be simulated. Coding can thus be thought of as an evolutionary process. The programmer builds up a "super-BLØDI" language featuring blocks that are commonly used in his application.

*Compatible Programming (Subroutines)*

BLØDIB offers the user the facility to write simulation programs that can coexist, communicate, and interact with other programs in the computational environment. The user is provided with a staement that closely resembles the FØRTRAN SUBRØUTINE statement. With this facility the programmer has the ability to write a BLØDI program, i.e., a circuit description, with some parameters, such as amplifier gains or quantizer decision levels expressed as variables, specifying them as symbolic names rather than as numbers. At the beginning of the program, the programmer *tells* the compiler that what follows is to be a subroutine with the name he specifies and that the variables in the list he writes must be supplied when the subroutine is called. A typical opening statement of a BLØDIB subroutine source program might appear as

```
TEST     SUBR     (GAIN1,GAIN2,SHIFT),
```

where test is to be the name of the subroutine and GAIN1, GAIN2, and SHIFT are variables whose values shall be specified by the calling program, and which are used somewhere in the circuit description or standard BLØDI program that follows. Statements involving the variables might appear as

```
AMP22    AMP      GAIN1,SHIFT,ØUTP3
CØUNT    ACC      GAIN2,2,ØUTP4
```

The program that is produced is a subroutine structurally identical to subroutines produced by other compilers such as FØRTRAN. Its parameters may be varied. It can be loaded along with other programs. It can receive parameter values, and it can transmit and receive data.

Futhermore, the ability to use floating point arithmetic is simulation programs enhances the compatibility of those programs with numerical analysis

routines, almost all of which are written in the floating point mode. (It hardly seems necessary to state that the use of floating point arithmetic almost completely eliminates the scaling difficulties inherent in many simulation techniques.)

The simplest application of the subroutine feature is the case in which one wants to simulate a system repeatedly for a range of values of a parameter(s). For this case one codes the simulation program as a subroutine with a variable parameter(s). The subroutine is then used in conjunction with a main program, written in some general purpose language, that calls the simulation subroutine with various values for the parameter(s) of interest. The main program can obtain the parameter values from some internal array, by reading cards, or perhaps receiving them from a remote console at which the engineer is stationed.

Iterative system design schemes can be implemented using the subroutine facility. Situations arise where one has to optimize a group of parameters associated with a complicated system. The solution of such a problem might involve the following steps:

1. Make an initial guess of values of the variables.
2. Simulate the system using those values.
3. Use the results of the simulation to calculate new values for the variables.
4. Return to step (2) unless the new values are within a given neighborhood of the last values.

To implement such a procedure one writes a simulation subroutine and loads it along with the necessary analysis and design programs. A main program will probably also be necessary to handle control.

Unfortunately, many problems of parameter optimization require human intervention commonly known as *eyeballing* and *knob-twiddling*. This points up another reason why it is so important for the simulation program to have the ability to communicate with other routines. The era is almost upon us in which men at remote stations will have the capability to interact with the computer. Simulation seems supremely suited for man-machine interaction.

Another use of the subroutine feature is to allow simulation programs to function as *analysis* programs. At Bell Telephone Laboratories, a program used to design sampled—data filters presents, as part of its output, a plot of the impulse response of the designed filter. Instead of using residue calculations based on the continuous analog of the filter, the response was obtained from a BLØDIB-produced subroutine which simulates the actual sampled-data filter. The *circuit* is excited by a single pulse and the resultes of the simulation are plotted and presented as part of the output.

To sum up this discussion, simulation programs can now be written to be compatible with other kinds of programs, opening the way for a great many diverse functions.

### *Modular Programming*

This feature allows the user to write simulation subroutines in the BLØDI language that structurally resemble simulation blocks. This facility allows what can be thought of as modular programming. A basic simulation program can be thought of as a piece of hardware with sockets. Various subsystems are coded, and the main system is run with a selected group of subsystems *plugged into the sockets*.

Among other things, modular programming allows the user to effect structural changes. Suppose there is a very complicated system to be simulated, and suppose there is a block whose function one would like to change for each of three simulations. Perhaps one would have this block a transversal filter for one simulation, a quantizer for a second, and perhaps nothing, a wire, for a third. In the main circuit description the block in question might have a type called MØD. Three modules would then be coded, each of type MØD, one a filter, one a quantizer and one a wire. One then runs three different simulations, loading a different module each time.

An important point here is that the main system need be compiled only once. This main *piece of software* is never changed. New elements are merely *plugged into the sockets* to test various design schemes.

Modular programming makes it possible for one to build a library of simulation modules in much the same way as a library of numerical functions subroutines is built. The feature aids in producing complete simulation systems for major projects such as, for example, vocoder research.

## CONCLUSIONS

BLØDI has proven to be an efficient and easy to

use tool for the simulation of a class of systems that:

1. involve a relatively smooth flow of data (signal) into and out of the system, i.e., the system does not process samples in a complicated order;
2. can be represented in terms of the pulse circuitry that has been described;
3. and that, once depicted as digital system block diagrams, contain blocks that usually change state at each sampling instant.

For this class of problems the compiler produces machine language programs that are as efficient as those produced by professional programmers. (For example, in the course of the algorithm for ordering the blocks for processing, the compiler searches the connection matrix, attempting to save store and fetch operations.)

A new program, BLØDIB, has been written and described, which is based on the BLØDI language. BLØDIB is coded for the IBM 7094 II under the control of the Bell System VII monitor and I/Ø system. To use the system under another monitor would involve only changes in the I/Ø and interaction with the macro assembler. Changes in the I/Ø are identical to those that were necessary for the original BLØDI, which were carried out successfully at many computer (7090) installations. This new program offers many new features but suffers no degradation in object program efficiency or ease of usage. The new features change the nature of the compiler from a language translator to a full programming system. The new program is particularly suited for

1. building higher-level BLØDI languages by defining block types from combinations of blocks,
2. interative simulation procedures,
3. coding of large, flexible, modular, simulation systems, and
4. the use of simulation programs as subroutines for more complex procedures.

## ACKNOWLEDGMENT

We wish to thank R. H. Roth and D. E. Eastwood, with whom many valuable discussions concerning BLØDIB were held, G. J. Spivak, who helped with the time-consuming task of writing the compiler, and M. Karnaugh who suggested and encouraged the entire project.

## REFERENCES

1. J. L. Kelly, Jr., C. Lochbaum and V. A. Vyssotsky, "A Block Diagram Compiler," *B.S.T.J.*, vol. 40, no. 3, pp. 669-76 (May 1961).

2. J. F. Kaiser, "Design Methods for Sampled-Data Filters," *Proc. Frist Allerton Conference on Circuit and System Theory,* Nov. 1963, Monticello, Ill.

3. R. M. Golden and J. F. Kaiser, "Design of Wideband Sampled-Data Filters," *B.S.T.J.*, vol. 43, no. 4, part 2, pp. 1533-46 (July 1964).

4. R. A. Gaskill, "A Versatile Program-Oriented Language for Engineers," *IEEE Transactions on Electronic Computers,* vol. EC-13, no. 4, pp. 415-21 (Aug. 1964).

5. R. D. Brennan and R. N. Linebarger, "A Survey of Digital Simulation: Digital Analog Simulator Programs," *Simulation,* vol. 3, no. 6, pp. 23-36 (Dec. 1964).

6. R. M. Golden, "Digital Computer Simulation of Communication Systems Using the Block Diagram Computer: BLØDIB," *Proc. Third Annual Allerton Conference on Circuit and System Theory,* Oct. 1965, Monticello, Ill.

7. C. C. Cutler, "Transmission Systems Employing Quantization," U.S. Patent No. 2,927,962, Mar. 8, 1960 (filed Apr. 26, 1954).

Table 1. BLØD1B Type Dictionary.

| Block Type | Function | Output |
|---|---|---|
| DEL | Fixed Delay | $y_k = x_{k-1}$ |
| VLD | Variable Delay | $y_k = x^1{}_{k-x^2}$ |
| ACC | Accumulation | $y_k = x_k + p_1 y_{k-1}$ ( $p_1$–2) |
| FLT | Transversal Filter | $y_k = \sum_{l=0}^{p_1} p_{l-l^x k-lp_2}$ |
| AMP | Amplifier | $y = p_1 x^4$ |
| ADR | Adder | $y = \sum_{i=1}^{4} x^i$ |
| SUB | Subtractor | $y = x^1 - x^2$ |
| MAX | Maximum | $y = \max_{i=1}^{4} x^i$ |
| MIN | Minimum | $y = \min_{i=1}^{4} x^i$ |
| MPR | Multiplier | $y = x^1 x^2$ |
| DIV | Divider | $y = x^1/x^2$ |

| | | |
|---|---|---|
| CLP | Positive Clipper | $y = \begin{Bmatrix} x; x < p_1 \\ p_1; x \geq p_1 \end{Bmatrix}$ |
| CLN | Negative Clipper | $y = \begin{Bmatrix} x; x > p_1 \\ p_1; x \leq p_1 \end{Bmatrix}$ |
| SCL | Symmetric Clipper | $y = \begin{cases} x_1, \lvert x_1 \rvert < p_1 \\ p_1, \lvert x_1 \rvert \geq p_1 \end{cases}$ |
| FWR | Full Wave Rectifier | $y = \lvert x \rvert$ |
| BAT | Battery | $y = x + p_1$ |
| CØS | Cosine Generator | $y_k = p_3 \cos\left(\frac{2\pi k}{p_1} + p_2\right)$ |
| GEN | Function Generator | $y = p_i$; $i = 2, \ldots, p_1$ the sequence of parameters repeated cyclically from second to last. |
| WNG | Noise Generator | Pseudo-vondon noise from Gaussian distribution with standard deviation $p_1$. |
| QNT | Quantizer | $p_1$ in the number of representative levels. The representative levels are $p_2, p_4, \ldots, p_{2p_1}$. The decision levels are $\ldots, p_3, p_5, \ldots, p_{2p_1-1}$. |
| LQT | Linear Quantizer | $y = \left(\frac{x}{p_1}\right)_{\text{rounded up.}} p_1$. |
| SQT | Square Root | $y = \sqrt{x}$ |
| SMP | Sampler | $y = \begin{cases} x \text{ one sample out of } p_2 \\ p_3 \text{ otherwise} \end{cases}$ $p_1$ specifies an initial phase. |
| HLD | Sample and Hold | $y_k = x_l^1$ where $x_l^2$ was the last control sample to exceed the threshold, $p_1$. |
| CNT | Counter | Output is active level every $n$th time input exceeds a threshold and a passive level the remainder of the time. |
| DTS | Double Throw Switch | $y = \begin{cases} x^1; x^3 \geq p_1 \\ x^2; \text{otherwise.} \end{cases}$ |
| FLF | Flip-flop | Output has a low state output until the input exceeds an upper threshold. An upper state output is then maintained until the input is below a lower threshold, etc. |
| PLS | Pulser | If the input exceeds trigger level, the output is the pulse level for the next $d$ samples, where $d$ is a parameter for pulse length. The output is a quiescent level if none of the last $(d-1)$ samples exceeded the trigger level. |
| PRT | Printer | Lists up to three input signal under control of a fourth signal and a threshold. |
| INP | Input | Reads data from tape for input. |
| ØUT | Output | Writes data on tape for output. |
| BØF | Output Buffer | Writes output samples into a buffer area in the computer memory. |
| FIP | Integer to floating point conversion | |
| FØP | Floating point to kuteger conversion | |
| MIC | Microfilm plotter | Plots up to three signals on a linear plot under control of a fourth input and a threshold. |
| MIP | Input to SUPER | |
| MØT | Output from SUPER | |

In the above $y_k = y(kT)$ where $T$ is sampling period
$x^i{}_k = x^i(kT)$ where $x^i(t)$ is the $i$th input to the block

The superscript is missing for blocks with only one input, while the subscripts are neglected for blocks with no dependence on the past. $p_j$ is the $j$th parameter of the block.

# TWO-DIMENSIONAL PROGRAMMING*

Melvin Klerer and Jack May
*Columbia University*
*Hudson Laboratories*
*Dobbs Ferry, New York*

A new user-oriented programming system for the purpose of facilitating the programming and analysis of well-formulated problems has been designed and implemented at Columbia University, Hudson Laboratories. This system consists of a standard Flexowriter modified to construct two-dimensional mathematical expressions and a new programming language.

The typing and language rules are quite flexible, unrestrictive, and easy to learn. Typing errors are easily corrected by backspacing and overtyping or by pressing a special "erase" key. Subscripted and superscripted arithmetic expressions can be typed conveniently. Arbitrary-sized summation, product, integral symbols, and other mathematical symbols can be constructed from elementary strokes or formed automatically by selecting the desired symbol from an accessory console keyboard.

We attempted to meet the following criteria:

1. There should be less human effort: by this we mean fewer instructions (therefore fewer errors), less total time spent in coding, less debugging, and less high-level thinking necessary to solve the problem;
2. The system should be easy to learn (and therefore subject to universal use);
3. It should be adaptable to a wide range of problems and applications; and
4. It should produce a final product that is better than the "old-fashioned" product. In other words, not only should the object program be cheaper to produce but it should run faster than programs obtained using present compilers.

Figure 1 is an example of what one research scientist brought to us for computation and illustrates what we mean by a well-formulated problem. $A$ is a function of all the other variables and, except for $x$ and $y$, all are input parameters. When this is coded—regardless of whether we use FORTRAN, ALGOL, or any other system—the programmer must be careful that the argument of the square root does not become negative, and that the denominator of the function to be integrated does not become so small as to cause overflow.†

Figure 2 shows the corresponding source language statement as typed on our input device in our language.

*This work was supported by the Office of Naval Research and the Advanced Research Projects Agency.

†A system which will *automatically* make such analyses and take appropriate actions is under development by the authors.

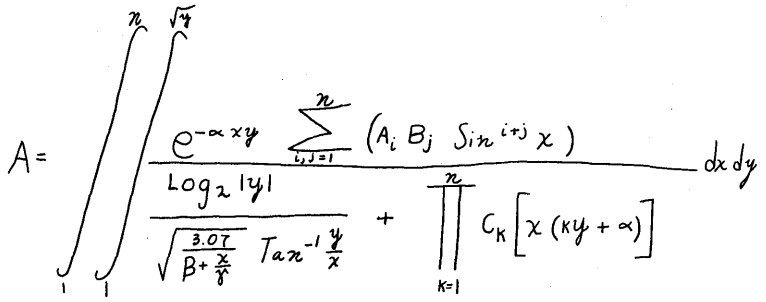


Figure 1.

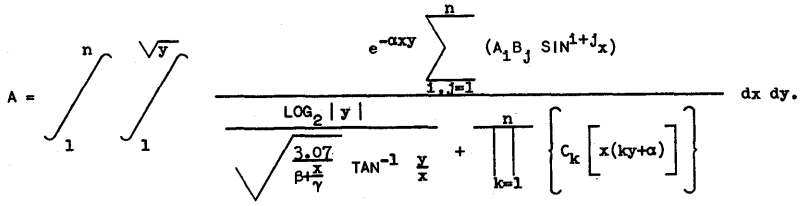


Figure 2.

Our Flexowriter has been modified so that the platen may be revolved by keyboard control. One presses the subscript key for the subscript position and the paper moves up half a line; similarly, the paper moves down half a line by pressing the superscript key. A typewriter device with this particular facility is not particularly new. As indicated in the references, announcement of the intention to construct such a machine was made as far back as July 1958 by two independent groups, one working at Los Alamos,[11] the other at Lincoln Laboratories,[14] and pioneer work in this field has been done by Mark Wells at Los Alamos.[12,13]

Our system permits the construction of symbols of *arbitrary* size besides allowing the use of other conventional mathematical forms such as implied multiplication, subscripting without the use of artificial conventions, subscript notation to denote a logarithmic base, and superscripting as in $\cos^{-1}x$ and $\cos^{2}x$. Arbitrary-sized integrals, summation symbols, or parentheses may be typed by combining basic strokes—horizontal and vertical bars, diagonals in both directions, and upper and lower semicircles as shown in Fig. 19. These basic strokes have been designed to interlock with each other. Symbols need not be symmetric nor well composed. Figure 3 illustrates some of the poorly formed symbols that are recognized correctly by our system. The strokes may be typed in any order. For instance, one may type part of a summation sign, then type part of the argument and go back to type more of the sum or part of the limits. Restrictions are of a minor nature; for instance, there must be enough room above and below the summation symbol to type the upper and lower limits.

Figure 4 is a photograph of a page from Hildebrand's book[7] on numerical analysis, illustrating his prescription for solving linear equations. We chose this as a good example of a well-formulated problem for computation. We would not call this an algorithm because it contains an inherent ambiguity. Note that the last equation has to be computed for $i$ taking on the value of $n$ first and then $n-1, n-2$, etc., down to $i = 1$. When $j = 1$ in the first equation, the sum becomes the null set.

Figure 5 is the corresponding program for the solution of a set of $n$ linear equations in $n$ unknowns as written in our programming language. The maximum value of $n$ has been arbitrarily set at 20. It can be noted that the body of the typescript is a fairly reasonable transcription of the text indicating double subscripts, etc. In the last formula our loop is programmed backwards, but we did not have to worry about the fact that the summation's upper limit can be less than the lower limit—the system automatically takes care of this consideration.

Figure 6 shows a truncated continued fraction as typed for computer input.

One question usually asked after we demonstrate our system to a visitor is: The system seems fine for the novice but will it be a useful tool for the experienced senior programmer? Our answer is yes. The language allows the expert to write long complex statements that are not possible with most presently operational compilers. Figure 7 illustrates a program written by one of our senior program-

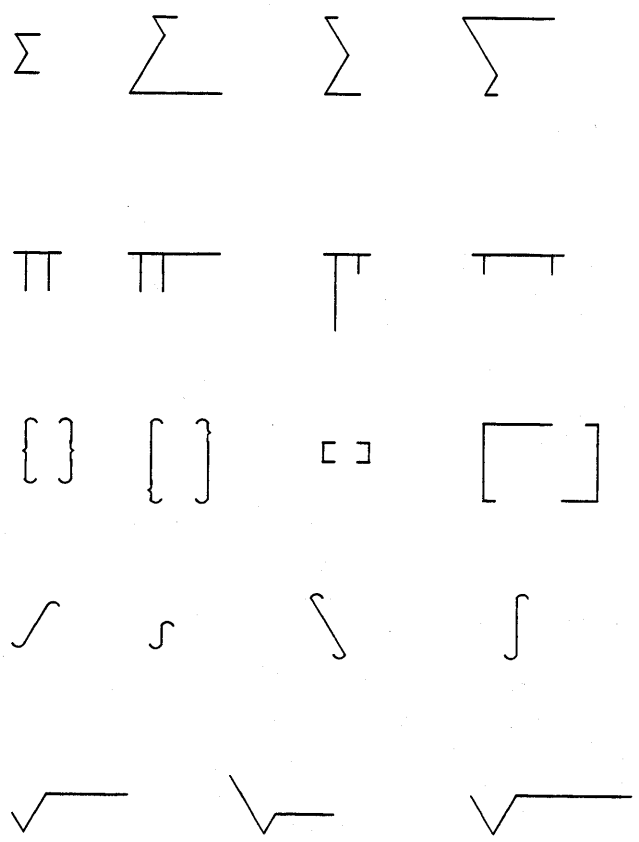


Figure 3.



tical with $c'_n$. Each succeeding element above it is obtained as the result of subtracting from the corresponding element of the **c**′ column the inner product of its row in **A**′ and the **x** column, with all uncalculated elements of the **x** column imagined to be zeros.

The preceding instructions are summarized by the equations

$$a'_{ij} = a_{ij} - \sum_{k=1}^{j-1} a'_{ik}a'_{kj} \qquad (i \geqq j), \qquad (10.4.4)$$

$$a'_{ij} = \frac{1}{a'_{ii}}\left[a_{ij} - \sum_{k=1}^{i-1} a'_{ik}a'_{kj}\right] \qquad (i < j), \qquad (10.4.5)$$

$$c'_i = \frac{1}{a'_{ii}}\left[c_i - \sum_{k=1}^{i-1} a'_{ik}c'_k\right]. \qquad (10.4.6)$$

and

$$x_i = c'_i - \sum_{k=i+1}^{n} a'_{ik}x_k, \qquad (10.4.7)$$

where $i$ and $j$ range from 1 to $n$ when not otherwise ·tricted.† It is seen that ·he process d·· ·l by (10.4.7) is iden·· ·h the "· solution" ·the Ga· ·on, w·· ·terr (10.3 ·

Figure 4.

mers. Except for the initialization, it is a one-statement algorithm for computing prime numbers. Note that two variables are stepped; one variable is incremented by 2 with no explicit upper limit while the other variable is stepped by 1 (an increment of 1 is assumed if a BY clause is absent) until a terminating condition is satisfied. In this case the condition involves the use of the incremented variable, but, in general, the condition does not have to depend upon the incremented variable and may be a

MAXIMUM n=20.

READ n.

READ $A_{ij}$ FROM j=1 TO n AND i=1 TO n.

READ $C_i$ FROM i=1 TO n.

FROM j=1 TO n AND i=1 TO n IF i>j THEN $\alpha_{ij} = A_{ij} - \sum_{k=1}^{j-1} \alpha_{ik}\alpha_{kj}$ OTHERWISE $\alpha_{ij} = \dfrac{A_{ij} - \sum_{k=1}^{i-1} \alpha_{ik}\alpha_{kj}}{\alpha_{ii}}$.

FROM i=1 TO n COMPUTE $\gamma_i = \dfrac{C_i - \sum_{k=1}^{i-1} \alpha_{ik}\gamma_k}{\alpha_{ii}}$.

FROM i=n BY -1 UNTIL i<1 COMPUTE $X_i = \gamma_i - \sum_{k=i+1}^{n} \alpha_{ik}X_k$.

PRINT i{2}, $X_i$ FOR i=1, 2, ..., n. FINISH.

Figure 5.

READ z.

$$x = 1 + \cfrac{z}{1 - \cfrac{z}{2 + \cfrac{z}{3 - \cfrac{z}{2 + \cfrac{z}{5 - \cfrac{z}{2 + \cfrac{z}{7 - \cfrac{z}{2 + \cfrac{z}{9}}}}}}}}}$$

PRINT z, x. FINISH.

Figure 6.

function of parameters unknown at compile time. Another feature of our "implied loop" is that the index *i* is tampered with inside the loop. Also, FROM clauses may be located anywhere in a statement as long as they make sense. The first FROM clause is performed most often; only when its UNTIL condition is satisfied does the next FROM variable become incremented. Apart from computer memory size there is no restriction to the number of FROM or FOR clauses allowed.

The use of IF conditional clauses and PRINT FORMAT statements is shown in Fig. 8. This program computes moving averages of every 10 data points and prints out a cumulative average every 20 points. We may see that multiple replacement operators can be used within a statement. The UNTIL condition on the FROM loop is satisfied by the reading of a particular data point from a punched card.

```
DIMENSION P=1000.   j=1.  PRINT j, P_j=3.

FROM i=1 UNTIL FRACTIONAL PART P/P_1 =0 IF P_1^2>P THEN i=j, j=j+1 AND PRINT j, P_j=P FROM P=5 BY 2 TO INFINITY.

FINISH.
```

Figure 7.

```
A=B=i=j=k=0.
FROM γ=1 UNTIL X=99999 READ X, COMPUTE A=A+X, B=B+X, i=i+1, j=j+1,
(IF k=0 PRINT γ{3}, X{4.2} AND k=1 OTHERWISE k=0 AND PRINT γ{3}, , X{4.2}),
IF i=10 PRINT FORMAT 1, γ, B/10 AND COMPUTE i=B=0, IF j=20 PRINT FORMAT 2, A/γ AND COMPUTE j=0.
FORMAT 1 AFTER A TOTAL OF xxx POINTS THE MEAN OF THE LAST TEN POINTS IS xxxx.xx.
FORMAT 2 AND THE AVERAGE OF THE WHOLE SET IS xxxx.xx.   FINISH.
```

Figure 8.

Punched input card format is free field with blanks separating each datum. As many data points as desired are allowed on each card and the number may vary from card to card without the need for any defining information. The range of IF statements may be delimited by parentheses. OTHERWISE or ELSE clauses may be absent. Additional IF clauses may be nested within other IF clauses and may be put after THEN or ELSE clauses. The parentheses around the IF $k = 0$. . . clause cause the program to go to the IF $i = 10$. . . clause in either case; if the parentheses were not present then the program would test IF $i = 10$ only if $k \neq 0$.

There are several ways to print answers on the highspeed printer. A standard PRINT A statement will cause the value of $A$ to be printed in floating point style. $A$ may be an expression and may contain a replacement operator. The PRINT $\gamma$ {3}, X {4.2} will cause $\gamma$ to be printed as a 3-digit integer and $X$ to be printed with 4 places to the left of the decimal point and 2 places to the right. Finally, one may mix numerical results with literal messages by using a PRINT FORMAT statement and a FORMAT image. The PRINT FORMAT statement lists the expressions whose values are to be printed, while the FORMAT image contains the text with the position and size of the results denoted by groups of lower case $x$'s. When the magnitude of a number is unknown one lower case $y$ will cause the result to be printed in floating point style. Figure 9 shows the output of this program.

Whenever it is easily possible to determine the size of an array by inspecting a program there should be no need to specify its dimension explicitly. Inspection of Fig. 10 indicates that the $X$ array will require 500 locations and so the program automatically assigns 500 locations to $X$ at compile time. Currently there is no analysis of space requirements at run time, so these decisions are made by the compiler. Figure 10 illustrates a program to compute the mean and standard deviation of a group of numbers. This also shows how comments are inserted—by putting them between braces.

Figure 11 shows semiautomatic dimensioning. In this correlation program, the number of points will be decided at run time but it is known that there will never be more than 500. It is certainly easier to specify a "maximum value" for one variable and let the system do the clerical work than to specify the same dimension for a number of arrays ($X$, $Y$, and $A$). Finally, one may also dimension arrays directly using a DIMENSION statement as shown in Fig. 12. Here the size of the arrays is not obvious so a DIMENSION statement is needed.

For systems of this type there is always the inherent possibility of ambiguity. For example, since we allow both double subscripts and implied multiplication, $A_{ij}$ in Fig. 13 may mean either $A$ as a function

```
 1          2145,32
 2                          9012,45
 3           128,73
 4                          2373,38
 5          6308,75
 6                          9523,54
 7          7520,75
 8                          4510,83
 9          6280,40
10                          6479,01
AFTER A TOTAL OF  10 POINTS THE MEAN OF THE LAST TEN POINTS IS 5428,32
11          7063,57
12                          6312,47
13           245,12
14                          1860,23
15          6105,85
16                          3054,12
17          1754,95
18                          8578,44
19          3328,49
20                          6650,17
AFTER A TOTAL OF  20 POINTS THE MEAN OF THE LAST TEN POINTS IS 4495,34
AND THE AVERAGE OF THE WHOLE SET IS 4961,83
21          2130,11
22                          8365,77
23          2238,01
24                          7543,25
25          3658,91
26                           375,75
27            92,14
28                          9825,16
29          5247,36
30                          6327,01
AFTER A TOTAL OF  30 POINTS THE MEAN OF THE LAST TEN POINTS IS 4580,35
31          2238,47
32                          9510,42
33          5514,03
34                          2584,45
35          3275,28
36                          3514,75
37           258,14
38                          6325,02
39          3514,75
40                           258,14
AFTER A TOTAL OF  40 POINTS THE MEAN OF THE LAST TEN POINTS IS 3699,34
AND THE AVERAGE OF THE WHOLE SET IS 4550,84
41          6325,02
42                          6854,14
43           267,14
44                          4874,14
45          9235,41
46                          6732,14
47          5602,14
48                          2478,46
49          3579,96
50                          9520,41
AFTER A TOTAL OF  50 POINTS THE MEAN OF THE LAST TEN POINTS IS 5546,90
51          8762,28
52                           357,79
53          6854,42
54                          2398,75
```

Figure 9.

of two independent indices, $i$ and $j$, or $A$ as a function of one index, the expression, $i$ times $j$. Also, since the argument of a function does not have to be delimited by parentheses, the interpretation of "SIN $A$ COS $B$" is ambiguous. Is the argumen tof the "SIN" "$A$ COS $B$" or just "$A$"? If the expression is interpreted as (SIN $A$)$\times$(COS $B$)—which seems reasonable—does this mean that if no parentheses are present to delimit an argument, then the appearance of another function name will delimit the argument? If so, then "SEC TAN$^{-1}$ ($A/2B$)" would be "SEC of nothing times TAN$^{-1}$ ($A/2B$)."

READ S { IDENTIFICATION } .

READ $X_1$ FROM i=1 TO 500.

$$Y = \frac{\sum_{i=1}^{500} X_i}{500} .$$

$$\sigma = \sqrt{\frac{\sum_{i=1}^{500} (X_i - Y)^2}{500}} .$$

PRINT FORMAT 1, S, Y, σ.

FORMAT 1 THE MEAN OF GROUP xx IS xxxx.xx WITH A STANDARD DEVIATION OF y.

FINISH.

Figure 10.

MAXIMUM n=500.

READ n.

READ $X_i$, $Y_i$ FROM i=1 TO n.

$$A_\tau = \sum_{i=1}^{n-\tau} X_i Y_{i+\tau} \text{ FOR } \tau=0, 1, \ldots, n-1.$$

WRITE TAPE A, 2, 2, n.

FINISH.

Figure 11.

DIMENSION A=100, B=100.

READ K, ϵ.

FROM i=1 TO $\frac{\text{LOG } K}{\epsilon^2}$ COMPUTE $A_i = B_i^{\,K}$ AND PRINT i, $A_i$, $B_i$.

FINISH.

Figure 12.

$$X=A_{ij}.$$

$$Y=SIN\ A\ COS\ B.$$

$$Z=SEC\ TAN^{-1}\ \frac{A}{2B}.$$

$$A=CSCH^{-1}\ A\ -\ L.$$

$$B=\sum_{i=1}^{100} A_i B_i\ +\ M.$$

$$C=A/2B.$$

$$D=COS^{3n-2}t\ -\ 3.6\ e^{x}.$$

$$E=LOG_{2k+1}4P^{2}\ +\ \frac{\pi}{2}\ .$$

Figure 13.

For the next statement we might ask whether the argument of $CSCH^{-1}$ is $A$ or $A - L$. For the following statement we would want to know whether or not $M$ is within the summation. The system's interpretation of the last three statements is also of interest. ($\pi$ and $e$ are interpreted as 3.14159. . . and 2.71828. . .) Immediately after the source program is read, the system's interpretation is listed on the high-speed printer in a linear, FORTRAN-like, intermediate language. The output resulting from Fig. 13 is shown in Fig. 14. This shows that $A_{ij}$ is interpreted as a two-dimensional array *but another context may yield a more appropriate interpretation.* $Y$ is interpreted as the product of SIN $A$ and COS $B$. The argument of the secant is interpreted as $TAN^{-1}$ $(A/2B)$. The variables $L$ and $M$ are interpreted to be outside the argument of the $CSCH^{-1}$ and the sum, respectively. The next statement is interpreted in the FORTRAN manner and is not $(A/2B)$ but $(A/2)B$. The argument of the $TAN^{-1}$ in statement 3 is interpreted as $A/(2B)$, because it was originally typed using displayed division rather than a slash. Statement 7 shows that the exponent, $3n - 2$ has been moved from its location adjacent to the COS to its proper place for computation. Statement 8 shows how the base of a log is treated. After the "LOG to the base 10" of the argument is taken, that quantity is divided by the "LOG to the base 10" of the base. We emphasize that each particular interpretation is a function of the local context in which it is actually used. In our opinion, an *immediate* response to the user returning the system interpretation is of great utility in resolving many ambiguous forms.

The single variables on our keyboard include the entire uppercase alphabet, 12 lowercase letters and 16 Greek letters for a total of 54 variables. This may be doubled because a letter typed in red is considered different from the same letter typed in black. If 108 variables are not enough one may define other var-

```
THIS IS THE WAY WE INTERPRET YOUR STATEMENTS.  IF ANY ARE INCORRECT PLEASE RETYPE THE STATEMENT CORRECTLY.

A00001      X=A SUB (I,J)

A00002      Y=SIN(A)*COS(B)

A00003      Z=SEC(ARCTAN((A)/(2*B)))

A00004      A=ARCCSCH(A)-L

A00005      B=SUM WITHIN (100,I=1) OF (A SUB (I)*B SUB (I))+M

A00006      C=A/2*B

A00007      D=(COS(T)) RAISED TO (3*N-2)-3.6*E RAISED TO (X)

A00008      E=LOG(4*P RAISED TO (2))/LOG(2*K+1)+((PI)/(2))

FINISH.
```

Figure 14.

iables by a SPECIAL VARIABLES statement as shown in Fig. 15. However, if it is desired, extra variables may be constructed without the need for predefinition by appending a red superscript to a variable, as in $A^{MAX}$. This figure also illustrates the trick used to show functional relationship while saving memory space. Comment braces are put around the subscript of $F$ so that the sums are not stored in an array. The system's interpretation is shown in Fig. 16.

```
SPECIAL VARIABLES TEMPERATURE, PRESSURE.

READ A^MAX, TEMPERATURE, PRESSURE.

V=(C × TEMPERATURE + K × PRESSURE) A^MAX.

V=C + TxExMxPxExRxAxTxUxRxE - K + A^MAX.

F {i} = Σ (j=1 to 10) X_ij FOR i=1, 2, 3, ..., 50.

PRINT F.
```

Figure 15.

Another example of the use of comments is indicated in Fig. 17. This is a segment of an actual production program to calculate the power spectrum of a filtered signal. The part shown in the figure com-

```
THIS IS THE WAY WE INTERPRET YOUR STATEMENTS.  IF ANY ARE INCORRECT PLEASE RETYPE THE STATEMENT CORRECTLY.

A00001      DIMENSION TEMPERATURE,PRESSURE

A00002      READ AMAX,TEMPERATURE,PRESSURE

A00003      V=(C*TEMPERATURE+K*PRESSURE)*AMAX

A00004      V=C+T*E*M*P*E*R*A*T*U*R*E-K*A RAISED TO (M*A*X)

A00005      F*SUM WITHIN (10,J=1) OF (X SUB (I,J)) FOR I=1,2,3,...,50

A00006      PRINT F

FINISH.
```

Figure 16.

FROM 1=0 TO M COMPUTE $C_1=C_1+XX_1$. FROM 1=0 TO M-1 COMPUTE $X_1=X_{1+1}$.

READ TAPE $X_M$,2,2,1 AND COMPUTE $X_M=X_M-S$. STATEMENT 2. FROM j=N-M TO N LOOP TO STATEMENT 3.

FROM 1=0 TO M COMPUTE $C_1=C_1+XX_1$. FROM 1=0 TO M-1 COMPUTE $X_1=X_{1+1}$ . M=M-1.

STATEMENT 3. FROM 1=0 TO M COMPUTE $C_1=\frac{G^2}{N-1}\,C_1$.

{FINITE COSINE SERIES TRANSFORMATION OF $C_1$ FOR 1=0 TO M $V_1=\Delta t\left[C_0+2\sum_{j=1}^{M-1} C_j \cos\frac{j1\pi}{M}+C_M\cos 1\pi\right]$}

k=2M. $D_\rho=\cos\frac{\rho\pi}{M}$ FROM ρ=0 TO M. $D_\rho=D_{k-\rho}$ FROM ρ=M+1 TO k. LOOP TO STATEMENT 4. FROM r=0 TO M. ρ=W=0.

FROM 1=1 TO M-1 COMPUTE ρ=ρ+r, (IF ρ>k THEN ρ=ρ-k) AND W=W+$C_1D_\rho$. $V_r=\Delta t(C_0+2W+C_M\cos r\pi)$. STATEMENT 4.

{SMOOTHING THE SPECTRUM OPTION A HANNING OR OPTION B HAMMING}

{OPTION A} IF OPTIONAB=0 THEN E=.5 AND F=.25 OTHERWISE {OPTION B} E=.54 AND F=.23.

$U_0=E(V_0+V_1)$. FROM 1=1 TO M-1 COMPUTE $U_1=EV_1+F(V_{1-1}+V_{1+1})$. $U_M=E(V_{M-1}+V_M)$.

Figure 17.

putes a correlation function and its cosine transformation. For clarity the actual "book" formula is printed in red (the color is immaterial) and put between the comment braces. An experienced programmer would realize it would be highly inefficient to compute COS $ij\pi M$ a total of $M^2$ times. Since $M$, $i$ and $j$ are integers and cosine is a periodic function, the same numbers would constantly repeat themselves. Thus, the program, which is listed below the comment, gives the correct answers, but the computation is done in a much more efficient manner, and the comment serves as documentation.

With regard to machine code efficiency, it is always difficult to pick unbiased examples. However, our experience has led us to believe that in general our object programs are very efficient. Our symbol recognizer, translator and compiler make for a complex software system and there is no escaping the fact that it would be a substantial task to code our system for another machine. Because we were

interested in machine-efficient object programs we made no attempt to make our coding techniques machine-independent. However, it is possible to simplify the task of recoding for another system. One may just recode the symbol recognizer and translator parts, i.e., up to the point where the FORTRAN-like intermediate language is produced. Thus these parts can be considered to be a preprocessor for an existing FORTRAN-like compiler.

We have become convinced that the voluminous programming instruction and operating manuals usually encountered are rarely necessary. Thus we are trying to explore how concise one can make a system reference manual without impairing its practical utility. Presently we are using a manual consisting of one sheet of stiff 8½ x 11-inch paper printed on both sides, as shown in Figs. 18 and 19. As yet, we do not have enough experience with it to know whether we want to increase the size of this one-sheet manual; yet it is hard to envisage its ever growing to a size larger than two or three double-faced sheets.

## ACKNOWLEDGMENTS

We acknowledge with thanks the programming assistance of David Levine and Fred Grossman, the engineering assistance of Charles Amann and Saverio Conforti, and the encouragement of Alan Berman, Robert A. Frosch, and Ivan E. Sutherland. Fig. 4 is reproduced by permission of the McGraw-Hill Book Company.

## REFERENCES

1. K. G. Balke and G. Carter, "The COLASL Automatic Coding System," *Dig. Tech. Papers, ACM Natl. Conf.,* 1962, pp. 44-45.

2. A. J. T. Colin, "Note on Coding Reverse Polish Expressions for Single-Address Computers with one Accumulator," *Comput. J.,* vol. 6, pp. 67-68 (1963).

3. H. J. Gawlik, "MIRFAC: A Compiler Based on Standard Notation and Plain English," *Comm. ACM,* vol. 6, no. 9, pp. 545-547 (1963).

4. A. A. Grau, "The Structure of an ALGOL Translator," Oak Ridge Nat. Lab. Rep. 3054, Feb. 1961.

5. M. Grems and M. O. Post, "A Symbol Coder for Automatic Documenting," *Comput. News,* vol. 147, pp. 9-18; and vol. 148, pp. 15-19 (1959).

6. C. L. Hamblin, "Translation to and from Polish Notation," *Comput. J.,* vol. 5, pp. 210-213 (1962).

7. F. B. Hildebrand, *Introduction to Numerical Analysis,* McGraw-Hill Book Co., New York, 1956.

8. M. Klerer and J. May, "Algorithms for Analysis and Translation of a Special Set of Computable Mathematical Forms," Tech. Rep. 113, Columbia U., Hudson Labs. (Oct. 1963).

9. M. Klerer and J. May, "An Experiment in a User-Oriented Computer System," *Comm. ACM,* vol. 7, no. 5, pp. 290-294 (1964).

10. M. Klerer and J. May, "A User-Oriented Programming Language," *Comput. J.,* vol. 8 (July 1965).

11. Los Alamos Scientific Laboratory, "MANIAC II," *Comm. ACM,* vol. 1, no. 7, p. 26 (1958).

12. Mark B. Wells, "MADCAP: A Scientific Compiler for a Displayed Formula Textbook Lan-

13. Mark B. Wells, "Recent Improvements in guage," *Comm. ACM,* vol. 4, pp. 31-36 (1961). MADCAP," *Comm. ACM,* vol. 6, pp. 674-678 (1963).

14. A. Vanderburgh, "The Lincoln Keyboard—A Typewriter Keyboard Designed For Computers Input Flexibility," *Comm. ACM,* vol. 1, no. 7, p. 4 (1958).

# REFERENCE MANUAL

## Vocabulary List

| | | | | | |
|---|---|---|---|---|---|
| ABS | CARD | ELSE | LN | READ | TAN |
| ABSOLUTE | CARDS | END | LOG | RETURN | TANGENT |
| AND | COMPUTE | EOF | LOOP | REWIND | TANH |
| ARCCOS | CONTINUE | EQUALS | MAXIMUM | ROUND | TAPE |
| ARCCOSH | COS | EXP | MESSAGE | SEC | THE |
| ARCCOT | COSECANT | FILE | MINUS | SECANT | THEN |
| ARCCOTH | COSH | FINISH | OF | SECH | TIMES |
| ARCCSC | COSINE | FOR | OR | SIN | TO |
| ARCCSCH | COT | FORMULA | OTHERWISE | SINE | TOP |
| ARCSEC | COTANGENT | FROM | PAUSE | SINH | TRUNCATE |
| ARCSECH | COTH | GO | PERFORM | SLEW | TYPE |
| ARCSIN | CSC | HEADING | PLOT | SPECIAL | UNTIL |
| ARCSINH | CSCH | IF | PLUS | SQRT | UPPER |
| ARCTAN | CYCLE | INFINITY | PRINT | STATEMENT | VARIABLE |
| ARCTANH | DIMENSION | LABEL | PROCEDURE | STOP | VARIABLES |
| BY | DIVIDED | LINE | PROGRAM | SUBROUTINE | WITHIN |
| CALL | DO | LINES | PUNCH | SWITCH | WRITE |

A period denotes the end of a statement or the end of an implied loop.
Corrections can be made by overtyping or by pressing the control key ERASE when positioned over the error.
The initial value of all variables (including subscripts is assumed to be 0 unless defined.
Each program must be terminated by the statement END OF PROGRAM. or FINISH.
More than one statement per typing line is acceptable.
To continue a statement beyond the maximum typing length for one line, press the TAB button and at least one carriage return.
Names of variables with more than one character should be defined by a SPECIAL VARIABLES statement before use.
A comma or the word AND may be used to separate computable statements.

FROM i=1 TO 10 COMPUTE $A_i=B_i+C_{i+1}$, $C_i=A_{i+1}$X AND D =SIN $\theta_i$.

## Examples of Acceptable Forms

The letters E, F, G denote an arithmetic expression, e.g., E may denote the expression A + 2B + i, otherwise a single variable is meant. Braces { } denote a choice of forms. Square Brackets [ ] denote those forms that are optional.

$\sum_{i=E}^{F}$ $\sum^{F}_{i,j=E}$

Note: The horizontal extension of the lower limit equation and upper limit expression should not exceed the corresponding arms of the sum symbol. The operand of the sum should be outside the symbol.

$A_1\ A_E\ A_{ij}\ A_{E,F}$ $\prod_{i,j=E}$ $\sqrt{\ E\ }$

DIMENSION A=(N, M).
This indicates that A is an (N+1) by (M+1) array
DIMENSION B=40, Z=30, Q=(10, 50).
SPECIAL VARIABLE[S]≡DIMENSION
SPECIAL VARIABLES TEMPERATURE, HUMIDITY, PRESSURE, COUNT, LBJ=(14, 200), αγ=(10, 15).
UPPER is used in the same manner as DIMENSION AND SPECIAL VARIABLES except that the indicated variables are stored in upper memory.
UPPER C, WEIGHT=56, K=(20, 30).

Subscripted variables need not be dimensioned when used in forms such as:
(1) $A_{ij}=B_iQ_{i,j}$ FOR j=0(2)20 AND i=1 TO 5
or
(2) MAXIMUM n=10, K=15
. . . . . .
$A_{ij}=B_iQ_{i,j}$ FROM j=4,5, ..., K WITHIN i=0 BY 3
UNTIL n.
or
(3) A= $\sum_{i=1}^{40} B_i$, P= $\prod_{i,j=0}^{30} C_{ij}$

FROM i=E [BY F] {$\overset{\text{TO}}{\underset{\text{UNTIL}}{}}$} [$j \overset{=}{<}$ etc.] G

FROM i=E TO G (Unit steps assumed)
FROM i=N BY 2.34 UNTIL A+B
FROM A=B+5 BY 2 UNTIL Q>20
FROM i=E TO INFINITY

FOR i=1, 2,...,5
FOR j=5(10)55
FOR i=0,.5,...,7.5

Note: Any number of dots permissable but no extra spaces before terminating comma. The difference between the first two numbers specifies the increment in the first FOR form.

FROM or FOR forms can be used either to begin or end a statement.
$A_i=iB_i$ FROM i=1 TO 10.

FROM i=1 TO 10 COMPUTE $A_i=iB_i$.

DO [$\overset{\text{TO}}{\underset{\text{UNTIL}}{}}$] ≡ LOOP [ ] ≡ CYCLE [ ]
DO STATEMENT 5 FROM J=1 TO 10.
This indicates that all statements up to but not including 5 will be executed. (No two LOOP statements should terminate at the same statement number. Otherwise, any number of LOOP procedures within or external to other LOOP procedures is permitted.)

FROM≡WITHIN≡AND
FOR φ=0,5,...,90 WITHIN r=1 TO 10 AND σ=1 TO 5 LOOP TO FORMULA 6.
The loop to be performed most often is the first one; the least often is the last.

READ ≡ READ CARD ≡ READ CARDS
READ $A_i$ FROM i=1 TO $A_i$>15.
Card Format is free field; number of data points may vary from card to card and may be in either fixed or floating point form.
READ X. (only one card is read, one datum per card)
READ $A_i$, $B_{i+1}$ FROM i=E UNTIL $A_i$=93.643. (Only one set $A_i$, $B_{i+1}$ per card.)
READ $A_i$ FOR i=0(1)105. (Any number of $A_i$'s per card.)
Data may be punched into cards in the following forms:
2 -2 1.596 +3.213 -4.60 2.78T2[≡$2.78\times10^{2}$]
2.78T-2[≡$2.78\times10^{-2}$] 2.78E-3[≡$2.78\times10^{-3}$]
Each datum should be separated by at least one blank space and the value should be within $\pm10^{\pm76}$ and not exceed nine significant digits.

## Example

C=-1. D=15. E=3. F=4. G=2. M=1.
FROM n=1 UNTIL 4 COMPUTE $B_n$=5-n.
PERFORM $H_r$=r FOR r=1 UNTIL r=5.

$$A=\sum_{i=1}^{4}\left[1\ \frac{\sum_{j=1}^{4}(B_j\sum_{k=1}^{4}H_k)}{C+\frac{D}{E+\frac{F}{G}}}+M\right]$$

PRINT A.
FINISH.

## Three Alternate Formulations Of The Same Problem

MAXIMUM n=20.
READ n.
READ $A_1$, $B_i$ FROM i=0 TO n.

$$X=\sum_{i=0}^{n}\left\{A_i\prod_{j=1}^{n}B_jA_j\right\}.$$

PRINT X. FINISH.

DIMENSION x=20, y=20.
α=0, READ ω.
FORMULA 1. READ $x_\alpha$, $y_\alpha$.
α=α+1. IF α≤ω GO TO FORMULA 1.
S=α=0. STATEMENT 1. β=α, P=1.
STATEMENT 2. $P=Px_\beta y_\beta$, β=β+1.
IF β≤ω THEN GO TO STATEMENT 2.
$S=S+Px_\alpha$ AND α=α+1.
IF ω≥α GO TO STATEMENT 1.
PRINT S. END OF PROGRAM.

MAXIMUM W=20.
READ W. ρ=0.
FROM X=0 TO W READ $u_X$, $v_X$.
DO STATEMENT 3 FROM X=0 TO W.
σ=1.
FROM Y=X TO W COMPUTE $\sigma=\sigma u_Y v_Y$.
$\rho=\rho+u_X\sigma$.
FORMULA 3. PRINT ρ.
END OF PROGRAM.

Figure 18.

```
PRINT X,Y,Z.  PRINT Yi FOR i=1,2,...,N.
FROM i=1 TO N PRINT Ai.
PRINT Yi (A.B).
```

A and B are integers between 0 and 9. $Y_i$ will be printed in fixed point output, A significant figures to left of decimal point, B significant figures to right of decimal point. PRINT $Y_i$(3.2), $Y_i$(4), $Y_i$(4.), $Y_i$(0.2)

```
PRINT Yi (A.B.C).
```

(Printed as above except that the number is first divided by $10^C$ to change its range.)

```
PRINT A, B(4.2), C(0.1.1), D.  (Maximum of 8 variables)
PRINT LABEL ≡ LABEL ≡ HEADING ≡ PRINT HEADING
```

Only symbols available on the printer are to be used, maximum of 15 characters per label, maximum of 8 labels separated by commas.

```
LABEL A, COUNT, 3X, Z1A, , SIGMA(X), TEMP..
```

Messages on the printer or typewriter are printed using the following forms:

```
PRINT MESSAGE ... or TYPE NEGATIVE SQUARE ROOT.
SLEW N (Printer paper spaced N lines)
SLEW [TO] TOP (Paper will advance to top of page)

IF F = G THEN GO TO STATEMENT 1.
IF F = G GO TO STATEMENT 1.
IF F = G THEN B = C + E.
IF F = G THEN READ... .

IF F = G THEN . . . {ELSE / OTHERWISE} { GO TO E / COMPUTE... E }
IF F = G THEN [CONTINUE] {OTHERWISE} { GO TO E / COMPUTE... }
```

Examples of multiple conditions:

```
IF τ=5 OR G<H OR SIN θi>β² THEN { COMPUTE ...
                                  READ α
                                  C=D                OTHERWISE
                                  GO TO FORMULA 3
                                  CONTINUE
IF P=G AND H>ε/2 AND ...
IF U=0 OR (G=r SIN θ AND H≤Cω) ...

GO ≡ GO TO
GO TO STATEMENT 20
```

Comments (non-computable statements) are entered between { } symbols.

```
FROM i=1 TO 10 READ Xi {READ VALUES}.
Y{i,j}=i+12j.
```

Use of the next forms eliminates the necessity of using "DO" or "LOOP" statements. Computable sub-statements within an implied loop are separated by a comma or AND.

```
FOR i=1(1)50 AND j=0 BY 2 UNTIL Y>2000 READ Xij,
COMPUTE Y=2Xi,j AND PRINT Y.
FROM i=1 TO INFINITY READ Xi, IF Xi≠0 COMPUTE Y=Y+Xi,
n=n+2 OTHERWISE GO TO STATEMENT 1.
```

Superscripts that are red are used to form new characters rather than being interpreted as exponents. The following is a short program to determine the maximum absolute value of a set of positive numbers $X_i$. The memory cell used by $X^{MAX}$ is set to 0 if not previously defined.

```
FROM i=1 TO 100 IF |Xi|>X^MAX THEN X^MAX=|Xi|.   (red)
```

In the following tape commands L is the number of elements in the array V, T is the tape number and P is the controller (plug) number.

```
READ TAPE V, T, P, L.  The first L elements of the tape
  record is read into locations V0 to VL-1.
WRITE TAPE V2, T, P, 5.  (Locations V2-V6 are written on
  tape)
REWIND T, P.  RWD T, P.
WRITE END OF FILE T, P.  EOF T, P.
IF END OF FILE P THEN ...    IF EOF P GO TO ...
```

In the following example Y is the variable to be plotted, x is the "independent index" (i.e. Y=f(X), A=the minimum value of X and B=the maximum value of X.

```
PLOT Y, X, A, B.  PLOT Zi, i, 0, 1  FROM i=1 TO 565π.
```

## EXAMPLES

READ $A_i$, COMPUTE $Y=\frac{A_i}{A^{MAX}}$ AND PLOT Y, i, -1, 1

FROM i=1 UNTIL Y>1.

IF α>k COMPUTE $x=\sqrt{(\alpha-k)\Delta}$ , $Y=B_{ij}x+C_0T$
AND PRINT Y, α, T, k OTHERWISE COMPUTE x=2αk, $Y=B_{ij}x+C_0T\Delta$ AND PRINT Y, α, T, k FROM α=1 TO n WITHIN T=2 BY .01 UNTIL 3 AND FOR k=0(5)90.

FROM i=1 TO 10 AND j=1 TO 10 READ $A_{ij}$,
COMPUTE $B_{ij}=A_{ij}+X_i+Y_j$ AND PRINT $A_{ij}$, $B_{ij}$, $X_i$, $Y_j$, i, j.

FOR r=1, 2, ..., 10 AND FOR θ=-π(.01)π

COMPUTE $S_r=r\text{SIN}^2\theta$, $C_r=\sqrt{r\ \text{COS}^{-1}\theta}$ , $A=T_r=\sum_{\phi=1}^{30}\text{TAN}(.1\pi\phi\theta)$,

$V_r=\prod_{\phi=1}^{25}\frac{\text{LOG}_2\phi}{A+\frac{r}{\theta C+\frac{DEF}{G}}}$ AND PRINT r, θ, $V_r$, A.

IF ($X\leq Y$ AND $Y<0$) OR $|42-Y/\varepsilon| > (X-Y)^2$ THEN
COMPUTE $T_{XY}=Y(\varepsilon=\frac{X}{Y})^2$ AND $W=(YT_{XY})^{Y\varepsilon}$ AND PRINT W, $T_{XY}$, X, Y
FROM γ=2k+3 BY .01τ UNTIL W>5800 AND FROM X=1 TO 100 OTHERWISE GO TO STATEMENT 2

---

To define a procedure within a program:

```
. {SUBROUTINE / PROCEDURE} (Name).  .....................
.....................................................
.....................RETURN.........................
.................................RETURN..............
.............  . [END[(Name)][SUBROUTINE / PROCEDURE] ].
```

The name of a subroutine can be an alphanumeric string of any length but must begin with an alphabetic character and cannot be identical to any item in the vocabulary list. As many RETURN's as desired may be inserted to branch out of the subroutine back to the main program. The END statement is optional. It is preferable that all procedures be typed at the end of the program. If this is done precede the subroutine by the statement: STOP. However, if it is desired to define a procedure inside the main program then in some manner the program should "jump over" the procedure.

To call a procedure:

```
... CALL (Name) [SUBROUTINE / PROCEDURE] ...
```

Relative Positions of Special Characters

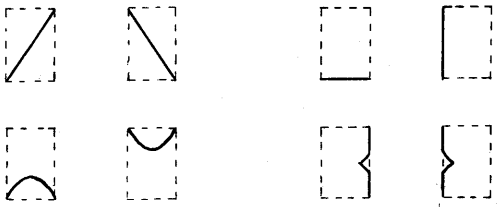


Figure 19.

# MICROPROGRAM CONTROL FOR THE EXPERIMENTAL SCIENCES

W. C. McGee and H. E. Petersen
*IBM Systems Research and Development Center*
*Palo Alto, California*

## INTRODUCTION

In many areas of the experimental sciences, increasing use is being made of general-purpose computers to control experimental apparatus and to record data from experiments. In most such applications the problem exists of connecting the apparatus to the computer so that data and control information may flow between the two. The problem is usually solved by placing a *controller* between the computer and the external equipment (Fig. 1). In this position the controller serves two functions:

(a) It provides a suitable electrical and logical interface between the computer and the external equipment; and

(b) It provides detailed control of the external equipment, thus leaving the computer free for other work.

The design of a controller for a particular set of external equipment and a particular computer presents no serious obstacles. Traditionally, controllers are implemented from flip-flop registers, logic elements (AND, OR, NOT, etc.), and occasionally small memories for data buffering. Timing diagrams are drawn showing the levels and pulses required at the controller's terminals, and the logic

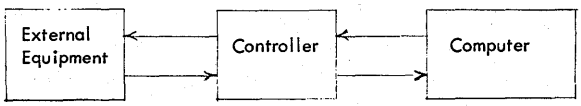



Figure 1. General control system.

elements are then interconnected to give the controller its proper terminal characteristics. The design process is essentially no different from that conventionally used in designing computers, except of course that a controller is usually not as complicated as a computer.

Although the conventional design process is straightforward, it has the inherent disadvantage that it must be repeated for each new configuration of computer and external equipment. In the experimental sciences, the number of such configurations is increasing rapidly, and it is quite possible that progress in this area will be limited by the time and cost to develop the requisite controllers by traditional methods. The situation would be materially improved if there were a *single design schema*

which was sufficiently general to accommodate a wide variety of computers and external equipment, and which could be quickly and easily particularized to meet the requirements for specific controllers. One design schema which appears to approach this goal is *microprogram control.*

In microprogram control, the functions of the controller are vested in a microprogram which is stored in a control memory. The microprogram is made up of microinstructions which are fetched in sequence from the memory and executed. The microinstructions control a very general type of hardware configuration, so that merely by changing the microprogram, the functions available in the controller can be made to range between wide limits. In addition, instead of the multiple, parallel logic elements found in conventional controllers, the microprogrammed controller requires only a single, central element to perform all logic and arithmetic. The microprogrammed controller thus has a potential cost advantage over the conventional controller.

The microprogrammed controller concept has been used to implement the IBM 2841 Storage Control Unit, by means of which random access storage devices may be connected to a System/360 central processor. Because of its microprogram implementation, the 2841 can accommodate an unusually wide variety of devices, including two kinds of disk storage drive, a data cell drive, and a drum. The 2841 thus provides an instance of the effectiveness of the microprogrammed controller concept in minimizing the effort that must go into controller design.

In this paper we will attempt to extend and generalize the microprogrammed controller concept, as embodied in the 2841 Storage Control Unit, to yield a more general controller design schema which would be suitable for use in the experimental sciences. We will first describe the basic concepts of the microprogrammed controller, and then describe how such a controller might be applied to a control problem typical of the experimental sciences, namely, the control of a CRT for scanning bubble chamber firm.

## THE MICROPROGRAMMED CONTROLLER CONCEPT

The functions of a microprogrammed controller are expressed in a *microprogram* which is stored in a control memory. The microprogram is composed of *microinstructions* which are read out of the control memory, one at a time, decoded, and executed. The microprogrammed controller is thus primarily a *sequential* device, in contrast to the conventional controller in which different operations usually proceed in parallel.

The microinstructions of a microprogrammed controller control a simple yet general hardware configuration. This hardware must be capable of storing small amounts of data, performing simple arithmetic and logic operations on these data, and accepting and transmitting data and control signals to the attached equipment. The general structure of one possible configuration meeting these requirements is shown in Fig. 2. The controller is organ-

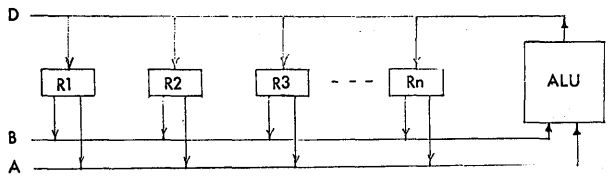



Figure 2. General structure of microprogrammed controller.

ized around a set of three data buses, A, B, and D; an arithmetic and logic unit (ALU); and a set of registers, R1, R2, . . . , Rn. Two of the data buses (A and B) provide input data to the ALU, and the third (D) receives the output of the ALU. The ALU is capable of performing simple arithmetic and logic operations on the input data, such as add, subtract, AND, OR, etc. The registers provide the sources of data to be operated on and also serve as destinations for the results. In general, any two registers specified by the microinstruction may provide the input data, one of them being connected to the A bus and the other to the B bus. The result of the ALU operation may then be returned, via the D bus, to any specified register (including one of the source registers, if desired). The registers, buses, and ALU are all the same width, which may be chosen to match the requirements of the control application. For example, a bus width of 8 bits (+ parity) appears to be a good choice for a wide class of applications.

Microinstructions are divided into fields, each of which has a specific function. To control the data flow in the configuration of Fig. 2, four fields

would be required: CA, CB, COP, and CD. Field CA determines which register will be connected to the A bus; field CB determines which register will be connected to the B bus; field COP determines what operation the ALU will perform on the A-bus and B-bus data; and field CD determines which register the ALU results will be sent to. Each of these fields can be made large enough to handle the maximum number of variations required. For example, by using 4 bits for the CA field, one can specify up to 16 different sources for the A bus.

| R1 | R2 | ADD | R3 |
|---|---|---|---|
| CA | CB | COP | CD |

To illustrate, suppose it were desired to add the quantity in register R1 to the quantity in R2 and place the sum in register R3. This could be accomplished with the following microinstruction:*

The structure of Fig. 2 provides for the manipulation of data already in the system, but provides no way of introducing data into the system. In general there are two ways of accomplishing this. One is by providing external connections to some of the registers, as will be described below; the other is by providing input directly from the microinstruction itself. In particular, one of the B-bus sources can be defined to be the "constant" field CK of the microinstruction. Whenever this source is specified in the CB field, the contents of the CK field in the same microinstruction will be gated onto the B bus. This technique is especially useful for introducing increments to counts (e.g., + 1) or certain bit patterns to mask off portions of data bytes. For example, to increment the quantity in register R1 by 4, the following microinstruction could be used:

| R1 | CK | ADD | R1 | 4 |
|---|---|---|---|---|
| CA | CB | COP | CD | CK |

The registers of a microprogram controller fall naturally into three groups: a *control* group, in *input* group, and an *output* group. To illustrate this grouping, a slightly more detailed schematic of a microprogrammed controller is shown in Figure 3.

The *control* registers are those registers required for general controller operation, i.e., without regard to the particular device or devices being controlled. For example, registers must in general be provided to hold intermediate results of ALU processing. These registers are designated TMP in Fig. 3. Another set of registers (CHI and CHO in Fig. 3) is provided for data to be transmitted to and from a general-purpose computer. These registers could, strictly speaking, be placed in the input and output register groups, so that the computer would assume the same status as any other device connected to the controller. Communication with a general-purpose computer is sufficiently stereotyped, however, that the registers required to effect this communication can be properly viewed as part of the control group.

A third type of register in the control group is the status register, designated ST in Fig. 3. Each bit of a status register indicates the (binary) status of some portion of the controller or device being controlled. A status bit may in general be set or reset from an external source (e.g., the computer channel, to signify that the CHI register is ready to be sampled); or from the microprogram itself. For the latter purpose, a CS field is provided in the microinstruction whose decoded value designates a particular status register bit and the value to which the bit is to be set. This provides the controller with the ability to "staticize" certain conditions existing at one time so they may be used to condition operations at a later time. An important example is the condition "D=O," i.e., whether or not the output of the ALU is zero. A certain value in the CS field will cause a 1 or a 0 to be set into a certain status register bit, according as D=0 or D≠0 on that microinstruction step. In addition to conditioning later operations of the controller itself, the status registers may of course be used to condition the operation of external equipment, and as such provide one of the sources for external equipment control.

To illustrate the function of the CS field, suppose register R1 contains a count of the data bytes received from the external equipment. Each time a byte is received, the count is to be decreased by one

*The fields of the microinstruction are shown symbolically; in practice they would contain equivalent binary codes.

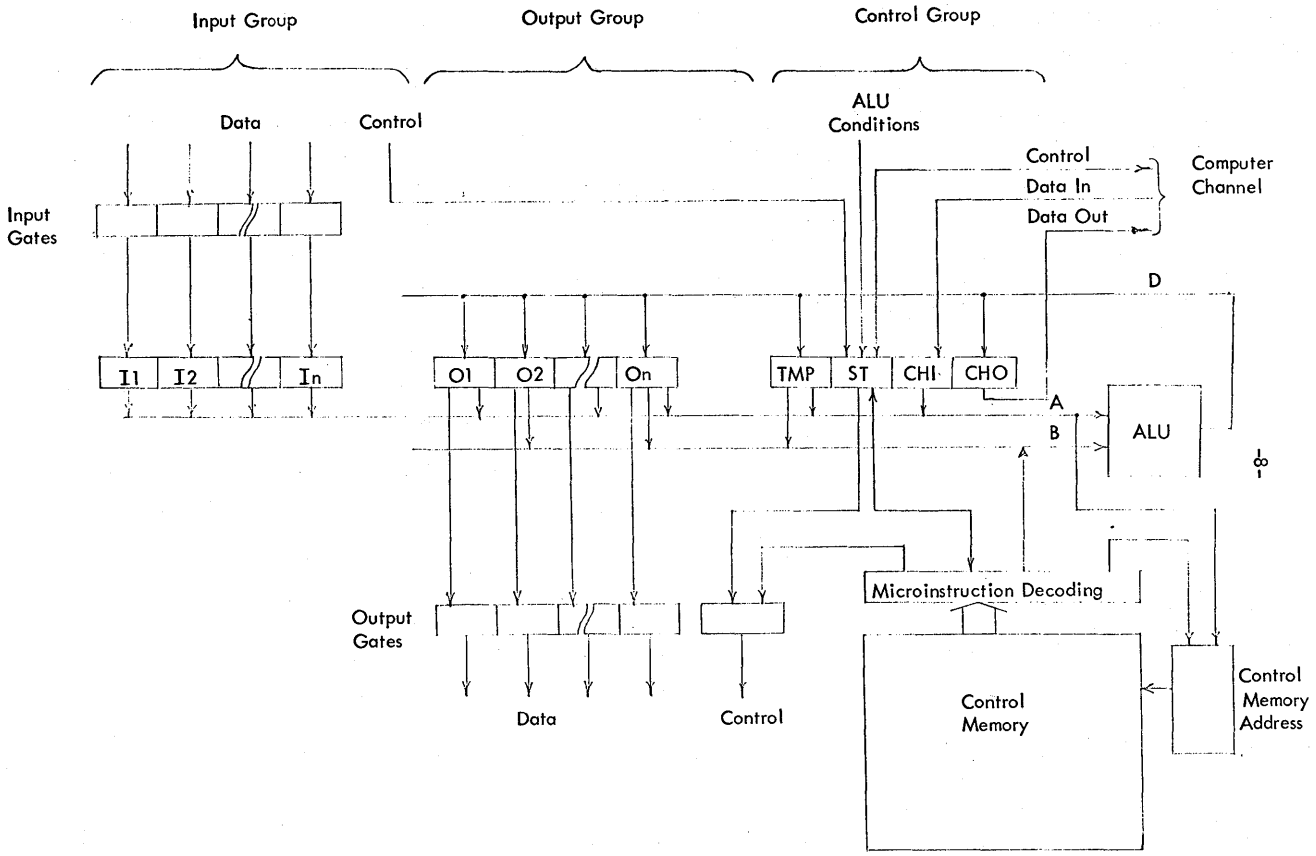
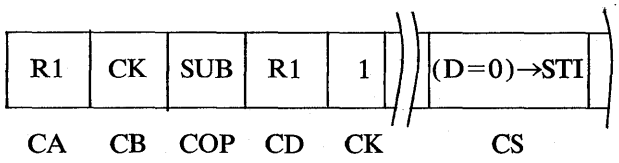



Figure 3. Basic microprogrammed controller.

and the resulting value tested for zero. When the value goes to zero, bit 1 of the status register is to be set to 1. This can all be accomplished with the following microinstruction:

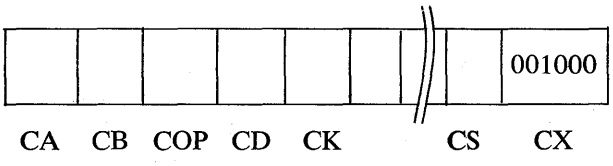



The second group of controller registers, the *input group,* provides a place to hold data coming from the equipment being controlled. In many cases these data will take the form of binary voltage levels which will be held by the external equipment until sampled by the controller and perhaps later caused to change. In such cases the registers need not be flip-flop registers but simply terminals which can be connected to the A or B bus. In other cases, it may be necessary, because of timing or other considerations, to buffer the input into flip-flop registers. In these cases, gating control must generally be provided by the controller and/or external equipment.

The third and final group, the *output group,* consists of flip-flop registers where the controller may deposit data to be used by the external equipment. The output of these resigters may be taken directly to the external equipment as binary levels, or may be transferred, through suitable gating, to external registers. Normally, the output registers need be connected only to the D bus. Under certain conditions, however, it may be convenient to bring the output register data back into the system, and so A-bus connections are in general provided for the output registers.

The setting of status register bits under microprogram control provides the basic mode of controlling the external equipment. In some cases it is convenient to augment this facility with control bits taken directly from the microinstruction. This is the principal purpose of the CX field. When a mi-

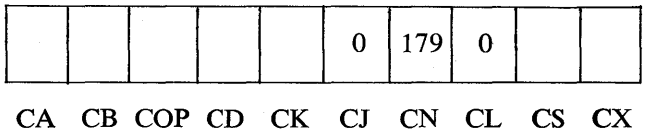
croinstruction is read out of control memory, the bits of the CX field are not decoded as are the other fields, but are instead used directly in the external equipment. For example, the microinstruction might, by virtue of the 1 in the CX field, cause the gating of a quantity into an input register.



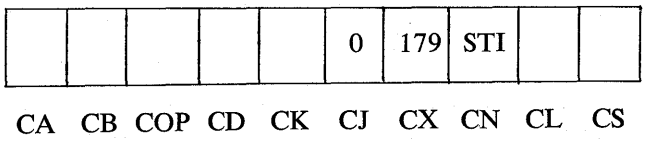

The CX field may also be used to extend the facility provided by the CK field for introducing arbitrary constants into the external equipment. Unlike constants from the CK field, any constants in the CX field would not enter the bus structure, but instead would go directly to the external equipment.

Since the microprogrammed controller is a sequential device, a key characteristic is the method used to get from one microinstruction to the next. One could, for example, have a conventional "program counter" which contains the memory address of the instruction currently being executed, and which is either incremented by one or is respecified to an arbitrary value (in case of a branch) to obtain the address of the next instruction. For microprograms, a more efficient procedure is to specify the address of the next instruction in *every* instruction, whether branching may occur or not. By this means, branching does not take a separate step, but may be performed on the same step as some other operation. Further, successive instructions may be located anywhere in control memory relative to one another, providing greater flexibility in the sharing of common sequences of microinstructions among different control functions.

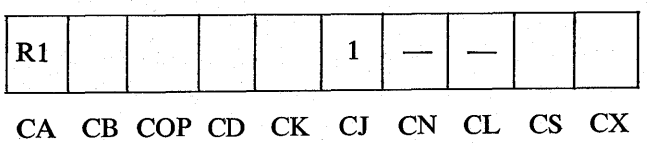
In the microprogram controller we are describing, the address of the next instruction is normally determined by the CN and CL fields. The CN field contains the high-order n-1 bits of the n-bit address of the next microinstruction; and the CL field determines which of a number of sources will be used to supply the low-order bit of the address. Two such sources, of course, are simply a "zero" and a "one," so that the location of the next microinstruction may be arbitrarily specified. Thus, the microinstruction




specifies that the next microinstruction is to come from location $179 \times 2 + 0 = 358$. Other sources which can be specified in the CL field include single bits of the status register. The microinstruction specifies that the next microinstruction is to come from location $179 \times 2 + 0 = 358$ if ST1 = 0, or from location $179 \times 2 + 1 = 359$ if ST1 = 1, thus effecting a two-way branch on the value of bit 1 of the status register.




By providing additional fields in the microinstruction to designate sources for other bits of the next-instruction-address, a capability of performing four-way, eight-way, etc., branching may be obtained. For simplicity, only two-way branching is assumed. However, we do provide a more general facility for using any input or calculated quantity as the address of the next instruction. Specifically, if the CJ field of the microinstruction is 1, the next address will not be obtained from the CN and CL fields as described in the preceding paragraph, but instead will be taken from the A bus. Thus, whatever register is gated onto the A bus by the coding in the CA field, that register's contents will be taken as the address of the next instruction. This facility provides a versatile many-way branch which is useful in command decoding and function generation. For example, suppose a code has been operated on arithmetically, and the result, which represents the starting address of the microprogram corresponding to this code, has been left in register R1. The coding will perform the desired branching.



In the examples considered above, the microinstructions are shown performing only a single function. In actual use, a single microinstruction will in general perform a number of functions, within the constraints imposed by the microinstruction format. Thus, a single microinstruction will in general perform an arithmetic or logic operation (fields CA, CB, COP, CD, and possibly CK); set a status register bit (CS field); set external lines (CX field); and select its successor (CN and CL fields).

The performance of the system described above is obviously very dependent on the speed of the control memory in providing microinstruction sequences, as well as the speed of the circuits being used. In general we can assume that the circuits are fast enough that the memory will be the limiting factor. Thus, for high performance, a very high-speed memory is required. However, it is not necessary that the control memory be an ordinary read/write memory. What is desired is that the memory be capable of being read very rapidly but that writing may take place fairly slowly.

In many applications even manually changing the contents of the memory might be suitable, since it is possible to rapidly change from one element of a program to another by branching without shifting in large blocks of a new program. This means, then, than any fast-read, slow-write memory that is economical might be used. For several years, the literature in computer technology[1-5] has described many such memories in which data can be changed in a period of minutes or at most hours, whereas the data reading times may be as short as a few tenths of a microsecond. In general, such "read-only" memories have shown savings in cost over their read/write counterparts in the same performance range. Should this cost picture change, read/write memory can be used though adequate consideration must be given to the system and operational advantages that apply to each.

By using the control memory as described, we very much limit the data storage capability of the system as described thus far. At least two simple alternatives are available to provide such a capability. One is to attach a conventional random-access memory directly to the controller bus structure, using one or more of the controller's registers as the "memory address register" (i.e., to hold the address of the location to be read or written); and one or more of the controller's registers as the "memory data register" (i.e., to hold the words read from or written to memory). Reading and writing operations would then be effected by appropriate microprogram sequences, much as any other external device is controlled.

The second alternative to providing storage capability is simply to use the memory of a general-purpose computer attached to the controller through the computer interface which is provided in the controller design. This method is especially appealing if the computer has input-output "channels" which can operate independently of the main processor, since in this case the controller can communicate with computer memory without interfering with the main computer program. Given such a facility, the controller we have described can perform many functions normally considered to be in the computer's domain, such as limit testing, function generation, data assembly, etc.

## APPLICATION TO PEPR

To illustrate the basic concepts of microprogramming, we would like in this section to describe briefly a typical control application and the manner in which a microprogrammed controller might be configured and programmed to handle the job. The application we have chosen for this purpose is the PEPR[6] film scanning application. PEPR is a computer-controlled CRT scanner used to automatically measure bubble chamber tracks which have been recorded on film. The PEPR cathode ray tube defocuses the electron beam into a short lnie segment whose angular orientation and location can be independently controlled by the system. Thus, a short line of light is controlled in angle and position on the face of the CRT and swept for a short distance under system control. When this line of light falls on one of the film tracks, a photomultiplier tube responds and the position of the beam is recorded as the time of arrival of the photomultiplier response. This is accomplished by starting a counter at the same time the line starts to move and remembering the count value when response occurs. These count values and the associated angle are sent to memory where subsequent processing will occur. The angle of the line is changed and scan repeated until the entire range of angles specified has been examined. A similar scan in another small area of the film is then initiated until, after approximately

500 such cells have been examined, the entire picture has been scanned.

In controlling this scan, the system must specify the coordinates of the cell center, the range of angles to be covered, and a few other factors such as sweep speed, line length, etc. The generation of the line by the CRT requires a special focusing system and currents that are non-linear functions of the angle $\Phi$ of the line. These nonlinear focusing currents functions $M(\Phi)$ and $N(\Phi)$ must also be supplied by the system.

A possible configuration for the PEPR system is illustrated in Fig. 4. The principal elements in this configuration are:

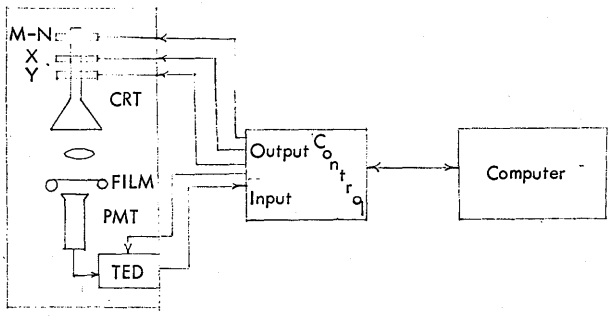



Figure 4. PEPR system schematic.

(a) *Scan Table.* The scan table contains the cathode ray tube and associated beam control circuits; the film transport equipment; the optics equipment, including the photomultiplier tube and circuits; and the data acquisition registers.

(b) *Computer.* The computer provides overall direction for the scanning, and performs the logic and arithmetic necessary to correlate isolated "hits" from the scanner into "tracks." Although not shown, the computer configuration would include certain standard peripheral items, including a magnetic tape unit for recording system output.

(c) *Controller.* The controller provides detailed control of the scan table, in response to commands from the computer.

Under program control, the computer may issue a command to the controller to initiate a scan of a designated cell on a bubble chamber photograph. The controller responds by issuing the proper signals to the scan table, receiving signals from the scan table, converting these signals into meaningful data, and relaying these data back to the computer. Command parameters which are available to the computer include

(a) The coordinates of the scan cell center.

(b) The length of the flying line.

(c) The effective excursion of the flying line on a single sweep.

(d) The range of angles to which the flying line will be oriented on successive sweeps.

Data returned to the computer includes

(a) The angle at which one or more hits were detected.

(b) The interpolation count representing the location of each hit.

(c) The identification of which track element detector (TED) detected the hits.

For the sake of illustration, we will assume that the controller responds to three different commands from the computer, as follows:

1. *Accept Parameters.* When the computer issues this command, it follows the command immediately with a single set of scan cell parameters, i.e., coordinates of scan cell center, line length, etc. For simplicity, we assume that the entire set is transmitted each time the command is given, and always in a fixed order. These parameters will then be in effect until the next set is transmitted.

2. *Start Sweep.* Using the parameters previously supplied, the controller initiates a sequence of sweeps, each sweep being made at an angle one degree greater than the previous sweep. This continues until the final angle is reached, or until a hit is detected at some angle. In either case, the controller then sends an appropriate interrupt signal back to the computer.

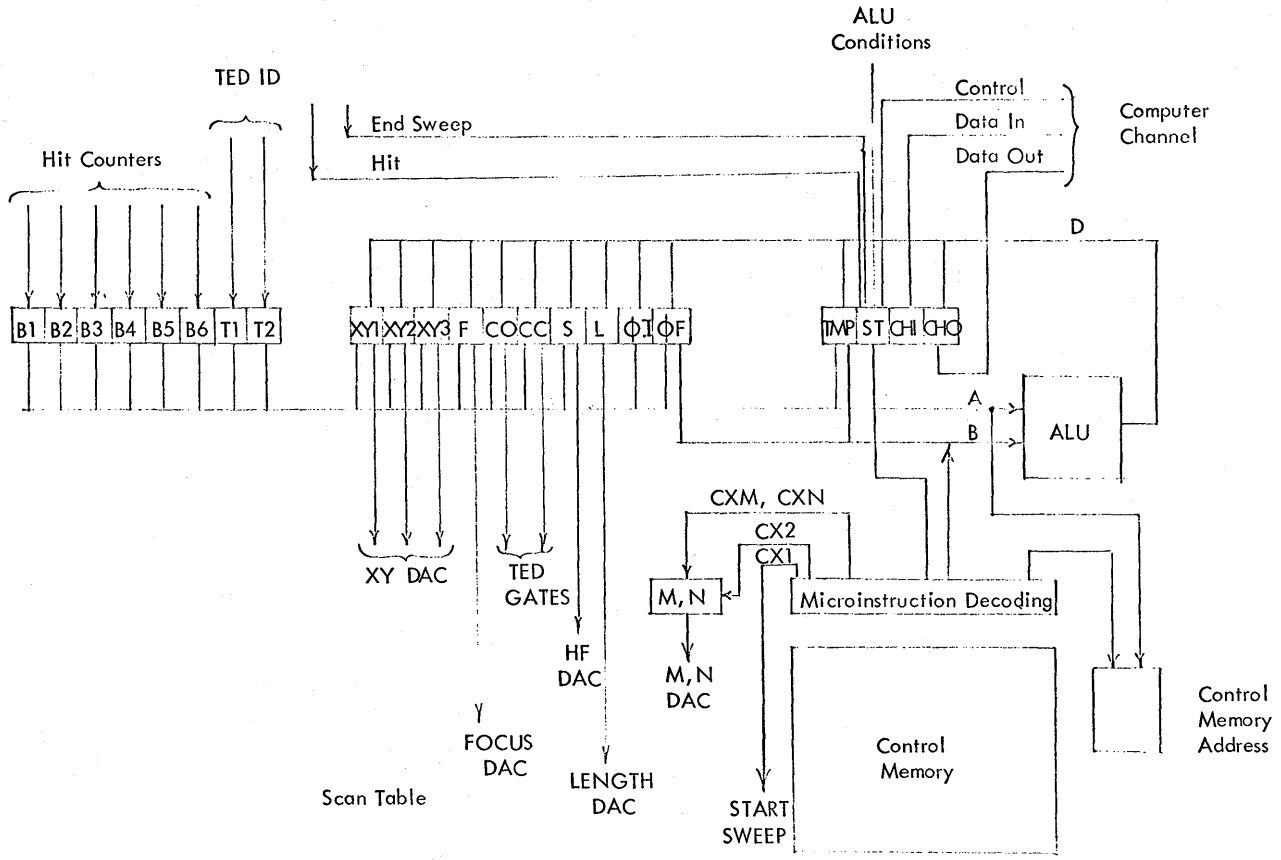



Figure 5. Microprogrammed controller for PEPR scanner.

3. *Send Data.* This command causes the controller to transmit to the computer the results of the last-detected hit, i.e., the hit angle, interpolation counts, etc. Again for simplicity, we assume the entire set of data is transmitted each time the command is given and always in the same order.

The computer uses these commands to control the scanning operation. When data is received from the controller via the "send data" command, the computer must make any ncecssary coordinate conversion, consolidate redundant data and correlate track data from different cells. The result of this processing and control are track coordinates which are then recorded on magnetic tape. This data tape is later processed by a general-purpose computer to do the physics calculations.

A microprogrammed controller to perform the above-described functions is shown in Fig. 5. Since the majority of parameters and data values are . . . bits or less, we assume a register-ALU-bus width of . . . bits. The functions performed by the ALU include addition, subtraction and "no-operation," the latter being used when it is desired to just move a data byte from one register to another.

The control registers include a single "temporary" register; two channel registers for communication with the computer; and a single status register with bit assignments as follows:

| | |
|---|---|
| ST0 | Not used |
| ST1 | End of sweep; set by scan table |
| ST2 | Hits obtained on sweep; set by scan table |
| ST3 | Controller request; set by controller |
| ST4 | Channel request; set by computer |
| ST5 | Final angle reached; set by controller |
| ST6 | Channel acknowledge; set by computer |
| ST7 | Controller acknowledge; set by controller |

Bits ST 2 and ST5 are also used directly to initiate interrupts in the computer.

The input group consists of 8 sets of input terminals of 8 bits each. Six of these (B1 through B6) are connected to interpolation counters, while the remaining two (T1 and T2) are connected to the TED indicator logic. The input terminals are connected through control gates to the A bus.

The output group contains 10 registers of 8 bits each. Eight of these registers hold parameters for a given scan cell and are directly connected to the appropriate external device:

| *Register* | *Parameter* |
|---|---|
| XY1, XY2, XY3 | Cell center coordinates X, Y (12 bits each) |
| F | Focus correction |
| CO | Interpolation Counter Open Gate |
| CC | Interpolation Counter Close Gate |
| S | Sweep Speed (Amplitude) |
| L | Line Length |

The remaining two registers, ΦI and ΦF, hold the initial and final angle, respectively. These registers are not connected to external equipment, but are included in the output group for convenience. All registers in the output group may be connected through control gates to the A bus, the B bus where specified, and to the D bus.

The control memory for this application requires a capacity of at least 256 words and a word length of at least 54 bits. The format of the microinstruction word, together with the various values which may be assumed by its fields, is shown in Fig. 6. Except for the CX field, the meaning of the various fields is exactly as described in the preceding section. For the present application, the CX field would be divided into four subfields:

(a) CX1 provides a one-bit signal to the scan table to start a sweep.

(b) CX2 provides a one-bit signal to gate the CXM and CXN fields (see below) of this microinstruction into a pair of external registers, which in turn drive DAC's in the scan table.

(c) CXM and CXN are each 9 bit fields, containing the values of M and N, respectively, corresponding to a given angle.

The intent of placing M and N into the CX field is to provide a very rapid means of supplying new M and N values on each sweep repetition, as descirbed next.

The operation of the controller is depicted in the flow charts of Fig. 7. The corresponding coding is given in Fig. 8. In its quiescent state, the controller sits in an endless loop waiting for the computer to issue a command. The controller expects one of three commands, as follows:

| *Command* | *Code* |
|---|---|
| Accept Parameters | 00000001 |
| Start Sweep | 00000010 |
| Send Data | 00000011 |

When the presence of a command is detected, the controller adds a constant to the command to obtain the address of the next microinstruction. This microinstruction in turn branches the controller to the sequence corresponding to a given command.

In the "accept parameters" sequence, the controller simply waits for the computer channel to transmit successive 8-bit bytes. As each byte is transmitted, the controller deposits it in the appropriate register in the output group.

The "send data" sequence is similar, execept that the controller places the contents of successive input-group registers on the channel and each time waits for the computer channel to acknowledge.

In the "start sweep" sequence, the controller execute a microprogram loop, with each traversal of the loop corresponding to a different sweep angle. On each traversal the controller performs the following steps:

(a) Places the current angle (ΦI) on the A bus and branches to the corresponding location. In locations 0 through 179 are stored 180 microinstructions which are identical except for the values in the CXM and CXN fields. These values correspond

## MICROINSTRUCTION CODING FOR PEPR CONTROLLER

| Decoded Field Value | Field CA (5) | CB (2) | COP (2) | CD (4) | CK (8) | CJ (1) | CN (7) | CL (3) | CS (2) | CX1 (1) | CX2 (1) | CXM (9) | CXN (9) |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | NOP | 0 | 0 | UseN | 0 | 0 | NOP | | | DIRECT | |
| 1 | XY1 | ϕF | ADD | XY1 | 1 | UseA | 1 | ST1 | 1→ST3 | | | OUTPUT | |
| 2 | XY2 | TMP | SUB | XY2 | 2 | | 2 | ST2 | (D=0)→ST5 | | | TO | |
| 3 | XY3 | CK | — | XY3 | 3 | | 3 | — | 1→ST7 | | | INTERFACE | |
| 4 | F | | | F | 4 | | 4 | ST4 | | | | | |
| 5 | CO | | | CO | 5 | | 5 | ST5 | | | | | |
| 6 | CC | | | CC | 6 | | 6 | ST6 | | | | | |
| 7 | S | | | S | 7 | | 7 | 1 | | | | | |
| 8 | L | | | L | . | | . | | | | | | |
| 9 | ϕI | | | ϕI | . | | . | | | | | | |
| 10 | ϕF | | | ϕF | . | | . | | | | | | |
| 11 | TMP | | | TMP | (to | | (to | | | | | | |
| 12 | CHI | | | — | 225) | | 127) | | | | | | |
| 13 | —(1) | | | CHO | | | | | | | | | |
| 14 | — | | | — | | | | | | | | | |
| 15 | — | | | — | | | | | | | | | |
| 16 | — | | | | | | | | | | | | |
| 17 | B1 | | | | | | | | | | | | |
| 18 | B2 | | | | | | | | | | | | |
| 19 | B3 | | | | | | | | | | | | |
| 20 | B4 | | | | | | | | | | | | |
| 21 | B5 | | | | | | | | | | | | |
| 22 | B6 | | | | | | | | | | | | |
| 23 | T1 | | | | | | | | | | | | |
| 24 | T2 | | | | | | | | | | | | |
| 25 | — | | | | | | | | | | | | |
| ⋮ | ⋮ | | | | | | | | | | | | |
| 31 | — | | | | | | | | | | | | |

Figure 6. Microinstruction coding for PEPF controller.
(1) Indicates not defined

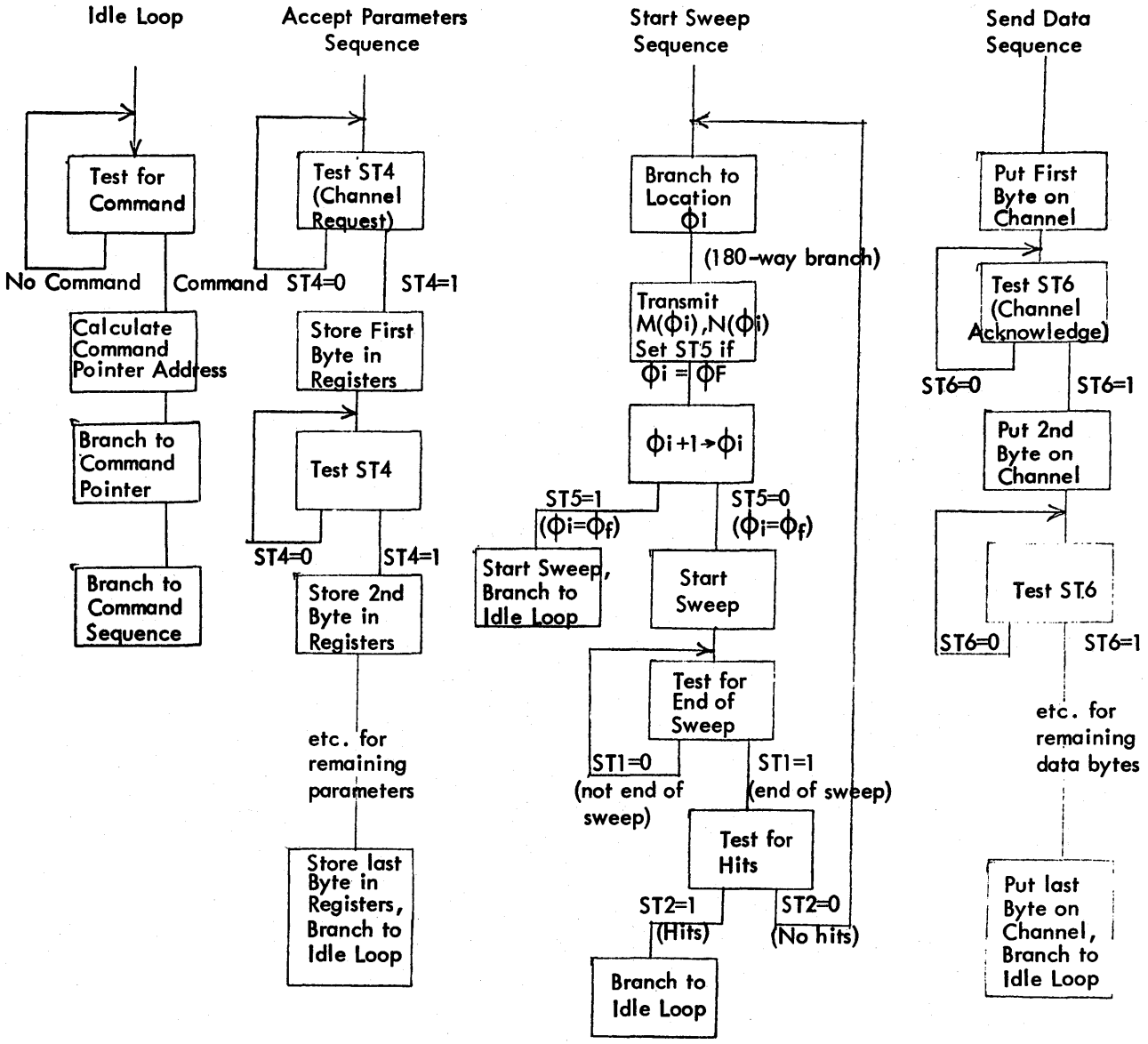



Figure 7. Flow chart of microprogram for PEPR controller.

to the associated angle, e.g., location 80 contains M(80) and N(80).

(b) Transmits the appropriate M and N to external registers; subtracts ΦF from ΦI and sets ST5 to 1 if the result is zero, i.e., if ΦI = ΦF.

(c) Increments the current angle by one degree; tests ST5 and branches accordingly.

(d) Assuming ST5 = 0 (i.e., ΦI = ΦF), transmits a start sweep pulse to the scan table.

(e) "Waits" for end of sweep.

(f) Tests for hits on the sweep and branches accordingly.

Assuming no hits were detected, step (f) would branch back to step (a), and the loop would be repeated. If hits were detected, the controller would

MICROPROGRAM FOR PEPR CONTROLLER

| Location | CA | CB | COP | CD | CK | CJ | CN (2) | CL | CS | CX1 | CX2 | CXM | CXN | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **IDLE LOOP** | | | | | | | | | | | | | | |
| 200 | —(1) | — | — | 0 | — | 0 | 200 | ST4 | 0 | 0 | 0 | — | — | |
| 201 | CHI | CK | ADD | TMP | 202 | 0 | 202 | 0 | 0 | 0 | 0 | — | — | |
| 202 | TMP | — | — | 0 | — | 1 | — | — | 0 | 0 | 0 | — | — | |
| 203 | — | — | — | 0 | — | 0 | 208 | 0 | 0 | 0 | 0 | — | — | Branch to "Accept" |
| 204 | — | — | — | 0 | — | 0 | 230 | 0 | 0 | 0 | 0 | — | — | Branch to "Start" |
| 205 | — | — | — | 0 | — | 0 | 238 | 1 | 0 | 0 | 0 | — | — | Branch to "Send" |
| **ACCEPT PARAMETERS** | | | | | | | | | | | | | | |
| 208 | — | — | — | 0 | — | 0 | 208 | ST4 | 0 | 0 | 0 | — | — | |
| 209 | CHI | 0 | NOP | XY1 | — | 0 | 210 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 210 | — | — | — | 0 | — | 0 | 210 | ST4 | 0 | 0 | 0 | — | — | |
| 211 | CHI | 0 | NOP | XY2 | — | 0 | 212 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 212 | — | — | — | 0 | — | 0 | 212 | ST4 | 0 | 0 | 0 | — | — | |
| 213 | CHI | 0 | NOP | XY3 | — | 0 | 214 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 214 | — | — | — | 0 | — | 0 | 214 | ST4 | 0 | 0 | 0 | — | — | |
| 215 | CHI | 0 | NOP | F | — | 0 | 216 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 216 | — | — | — | 0 | — | 0 | 216 | ST4 | 0 | 0 | 0 | — | — | |
| 217 | CHI | 0 | NOP | CO | — | 0 | 218 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 218 | | — | — | 0 | — | 0 | 218 | ST4 | 0 | 0 | 0 | — | — | |
| 219 | CHI | 0 | NOP | CC | — | 0 | 220 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 220 | — | — | — | 0 | — | 0 | 220 | ST4 | 0 | 0 | 0 | — | — | |
| 221 | CHI | 0 | NOP | S | — | 0 | 222 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 222 | — | — | — | 0 | — | 0 | 222 | ST4 | 0 | 0 | 0 | — | — | |
| 223 | CHI | 0 | NOP | L | — | 0 | 224 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 224 | — | — | — | 0 | — | 0 | 224 | ST4 | 0 | 0 | 0 | — | — | |
| 225 | CHI | 0 | NOP | ϕI | — | 0 | 226 | 0 | 1→ST7 | 0 | 0 | — | — | |
| 226 | — | — | — | 0 | — | 0 | 226 | ST4 | 0 | 0 | 0 | — | — | |
| 227 | CHI | 0 | NOP | ϕF | — | 0 | 200 | 0 | 1→ST7 | 0 | 0 | — | — | |

(1) Means field value is immaterial,
(2) Value shown is 2 times actual value, so that next address may be obtained by adding CN and CL.

Figure 8. Microprogram for PEPR controller.

simply branch back to the idle loop. The fact that ST2 was set to 1 would indicate to the computer that hits were present. Note that the current angle is one greater than the angle at which hits were detected, and to compensate for this the angle is decremented by one as it is transmitted back to the computer in response to a "Send data" command.

If, on step (c), ST5 indicated that the final angle had been reached, the controller would transmit a start sweep pulse to the scan table (to cause the final angle to be swept) and then return to the idle loop. The condition of ST5 would indicate to the computer that the final angle had been reached, while that of ST2 would indicate the presence or absence of hits at the final angle.

Once hits have been detected or the final angle

(continued)

| Location | CA | CB | COP | CD | CK | CJ | CN | CL | CS | CX1 | CX2 | CXM | CXN | Remarks |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| START SWEEP | | | | | | | | | | | | | | |
| 230 | $\phi$I | — | — | 0 | — | 1 | — | — | 0 | 0 | 0 | — | — | Branch to MN Table |
| 231 | — | — | — | 0 | — | 0 | 200 | 0 | 0 | 0 | 0 | — | — | Hits |
| 232 | $\phi$I | CK | ADD | $\phi$I | 1 | 0 | 234 | ST5 | 0 | 0 | 0 | — | — | |
| 233 | (not used) | | | | | | | | | | | | | |
| 234 | — | — | — | 0 | — | 0 | 236 | 0 | 0 | 1 | 0 | — | — | $\phi$I $\neq$ $\phi$F |
| 235 | — | — | — | 0 | — | 0 | 200 | 0 | 0 | 1 | 0 | — | — | $\phi$I = $\phi$F |
| 236 | — | — | — | 0 | — | 0 | 236 | ST1 | 0 | 0 | 0 | — | — | |
| 237 | — | — | — | 0 | — | 0 | 230 | ST2 | 0 | 0 | 0 | — | — | |
| SEND DATA | | | | | | | | | | | | | | |
| 239 | B1 | 0 | NOP | CHO | | 0 | 240 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 240 | — | — | — | 0 | — | 0 | 240 | ST6 | 0 | 0 | 0 | — | — | |
| 241 | B2 | 0 | NOP | CHO | — | 0 | 242 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 242 | — | — | — | 0 | — | 0 | 242 | ST6 | 0 | 0 | 0 | — | — | |
| 243 | B3 | 0 | NOP | CHO | — | 0 | 244 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 244 | — | — | — | 0 | — | 0 | 244 | ST6 | 0 | 0 | 0 | — | — | |
| 245 | B4 | 0 | NOP | CHO | — | 0 | 246 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 246 | — | — | — | 0 | — | 0 | 246 | ST6 | 0 | 0 | 0 | — | — | |
| 247 | B5 | 0 | NOP | CHO | — | 0 | 248 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 248 | — | — | — | 0 | — | 0 | 248 | ST6 | 0 | 0 | 0 | — | — | |
| 249 | B6 | 0 | NOP | CHO | — | 0 | 250 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 250 | — | — | — | 0 | — | 0 | 250 | ST6 | 0 | 0 | 0 | — | — | |
| 251 | T1 | 0 | NOP | CHO | — | 0 | 252 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 252 | — | — | — | 0 | — | 0 | 252 | ST6 | 0 | 0 | 0 | — | — | |
| 253 | T2 | 0 | NOP | CHO | — | 0 | 254 | 0 | 1→ST3 | 0 | 0 | — | — | |
| 254 | — | — | — | 0 | — | 0 | 254 | ST6 | 0 | 0 | 0 | — | — | |
| 255 | $\phi$I | CK | SUB | CHO | 1 | 0 | 200 | 0 | 1→ST3 | 0 | 0 | — | — | |
| MN TABLE | | | | | | | | | | | | | | |
| 0 | $\phi$I | $\phi$F | SUB | 0 | — | 0 | 232 | 0 | (D=0) →ST5 | 0 | 1 | M(0) | N(0) | |
| 1 | | | | | | | | | | | | M(1) | N(1) | |
| 2 | | | | | | | | | | | | M(2) | N(2) | |
| ⋮ | | | | | | | | | | | | | | |
| 179 | | | | | | | | | | | | M(179) | N(179) | |

has been reached, the controller returns to its quiescent state to await further commands. The computer may at this point request data; or it may transmit new parameters; or, it may cause scanning to resume at the angle one greater than that at which hits were last obtained, simply by issuing a "start sweep" command.

In the PEPR application, the cycle time of the control memory would ideally be selected so that the loop time of the controller exactly matched the sweep time of the scan table, i.e., so that neither device waited for the other. According to Fig. 7, the controller executes six steps in the loop. Thus, a sweep time of 10 microseconds would require a con-

trol memory cycle of no greater than 10/6 = 1.67 microseconds. Such cycle times are readily available with current technology.

## REMARKS

In this paper we have described a different approach to controller design, whose salient aspect is the use of microprogram sequence in place of conventional wired logic. The implications of microprogrammed control can perhaps be better understood by considering briefly the similarities and differences between two kinds of controllers for the PEPR application: the microprogrammed controller we have just described, and a controller implemented in the conventional manner.

From the standpoint of the hardware (registers, gates, etc.) required to hold the data coming from the scanner and computer, the two kinds of controller appear to be roughly equivalent. For exampl, the conventional controller would have to have two registers for holding the current and final angles, a counter for incrementing the current angle, and a comparator for comparing the current angle against the final angle. In the microprogrammed controller, the corresponding hardware is to be found in two registers for holding the current and final angle; and an ALU for decrementing and testing the angle. The specific manner in which this "common" hardware is controlled, of course, is much different in the two kinds of controllers.

Another significant difference is in the manner in which the M and N functions are generated. In a conventional controller this would be accomplished with a nonlinear function generator, whereas in the microprogrammed controller, the function is stored directly in control memory. The latter is obviously an advantage if it is ever necessary to modify the function.

Another difference is found in the generality of the ALU in the microprogrammed controller. A conventional controller would not be built with more arithmetic capability than absolutely required by the application, whereas the microprogrammed controller, through its very simple ALU and data flow characteristics, has theoretically unlimited arithmetic capability. The possibility therefore exists of extending the controller's functions (e.g., adding or subtracting constants to data as they are transferred to or from the computer) without loss of time or additional cost.

We have provided a detailed discussion of one application, but several others are immediately apparent. Control of any CRT scanner would be similar to the PEPR scanner. In other data collection systems, counters, pulse height analyzers, telemetry converters, etc., may all serve as sources of input data. The controller output registers may be used for a variety of external control purposes as well as data sources for display and printing devices. A wide and dynamic range of control can be provided by a single hardware complex by means of the "personality" provided by the control program. The simultaneous control of different experiments, for example, would be possible simply by incorporating separate programs within the control memory. While communication line switching and exchange and multiple data channel control are potential applications, preliminary analysis has indicated that expansion of some of the baisc concepts presented may be desirable.

The microprogrammed controller appears to offer significant advantages in design simplicity and flexibility, with respect to both the functions performed (as determined by the microprogram) and the particular equipment to be controlled (as determined by the input/output register configuration). For this reason, we feel that the approach is well suited to the laboratory environment, where changing requirements must be accommodated with a minimum of confusion and cost.

## ACKNOWLEDGMENT

The authors would like to acknowledge the initial suggestion made by Dr. Horace P. Flatt that we explore the use of microprogram control for scanning systems and his continued encouragement of this effort. We also appreciate the courtesy and help given by Drs. Irwin Pless, Horace Taft and Arthur Rosenfeld toward understanding the PEPR system.

REFERENCES

1. D. H. Looney, "A Twistor Matrix Memory for Semi-Permanent Information," *Proceedings of the Western Joint Computer Conference* (1959).

2. J. H. DeBuske, J. Janik and B. H. Simons, "A Card Changeable Non-Destructive Readout Twistor Shore," ibid.

3. H. R. Foglia, W. L. McDermid and H. E. Petersen, "Card Capacitor—A Semi-Permanent, Read Only Memory," *IBM Journal* (Jan. 1961).

4. T. Ishidate, S. Yoshizawa and K. Nagamori, "Eddycard Memory—A Semi-Permanent Storage," *Proceedings of the Eastern Joint Computer Conference* (Dec. 1961).

5. J. M. Donnelly, "Card-Changeable Memories," *Computer Design* (June 1964).

6. I. Pless et al, "A Precision Encoding and Pattern Recognition System (PEPR)," 1964 International Conference on High Energy Physics.

# PICOPROGRAMMING: A NEW APPROACH TO INTERNAL COMPUTER CONTROL

B. E. Briley
*Automatic Electric Research Laboratories*
*Northlake, Illinois*

## INTRODUCTION

The central processors of conventional computers may be roughly divided into two sections, an arithmetic section, which performs operations analogous to arithmetic upon representations of numbers, and a control section, which produces essentially a sequential group of gating pulses to accomplish the desired manipulation in the arithmetic section.

The arithmetic section lends itself admirably to modularization because of its repetitive structure. It is relatively easy to design and diagnose. The control section, however, has stoutly resisted similar treatment because it conventionally consists of an ensemble of special logic arrangements which differ, and, therefore, do not lend themselves to modularization on a logic level. This section is more difficult to design, and if the control malfunctions, an attempt at self-diagnosis by a typical machine may be roughly compared with asking an insane psychiatrist to analyze himself.

Described herein is a new approach to the design and realization of a control section which is modular by nature, simple to design and diagnose, and flexible to a unique degree.

## CONVENTIONAL CONTROL SECTIONS

### *Wired In*

A conventional control section decodes an instruction to ascertain which of a prewired set of control pulses must be dispatched to the arithmetic unit. In addition, it performs certain housekeeping duties, such as incrementing the instruction location counter.

Most of the housekeeping is performed for all instructions, and the housekeeping hardware is shared by them for economic reasons. Similarly, like portions of different instructions are often realized with the same piece of equipment. This design technique is very desirable from the point of view of economics, but it makes the machine more prone to total failure if a single element malfunctions.

The conventional control is totally wired in, rendering it quite inflexible. Any afterthought alteration of the instruction set requires a "soldering iron" approach.

### *Microprogrammed*

The microprogramming approach is a definite step forward in increasing the flexibility of a machine. A microprogrammed control section utilizes a macroinstruction to address the first word of a series of microinstructions contained in an internal, comparatively fast, control memory. These microinstructions are then decoded much as normal instructions are in wired-in control machines, to initiate production of (in general) a sequential series of pulses to control the arithmetic section.[1]

The microinstructions are generally relatively weak, but the macroinstructions, calling upon sub-

routines of microinstructions, can be quite powerful.

Since these subroutines are alterable, the nature of macroinstructions is very flexible, limited only by the microinstruction capabilities which are, however, fixed by wiring.

Unfortunately, the portion of the control which handles the microinstructions suffers the same lack of modularizability as the totally wired-in system.

## PICOPROGRAMMED

### *Philosophy*

Consider the control wires passing between the control section and the arithmetic section; these might number 100 in a machine of moderate size. If the signals which appear on these lines are examined during the execution of a typical instruction, it will be found that relatively few are activated. Further, for most instructions, the number of pulses which are produced on any one line is quite small (there is, of course, a small class of cyclic instructions typified by SHIFT N, which require long trains of pulses; these will be discussed separately).

Understanding the picoprogramming technique requires the recognition of a correspondence between the pulse-programming requirements of a control section and the capabilities of a memory element known as MYRA.[2] A MYRA memory element is a MYRiAperture ferrite disk which, when accessed, produces sequential trains of logic-level pulses upon 64 or more otherwise isolated wires. The relative width and timing positions of the pulses on the various wires are trivially adjustable, not only with respect to other pulses on the same wire, but with respect to pulses on the other wires associated with the same disk.

Thus, each memory element is capable of directly providing the gating pulses necessary to execute an instruction. A picoprogrammed system, then, consists essentially of an arithmetic section and a modified MYRA memory. A macroinstruction merely addresses an element in the MYRA memory; when the element is accessed, it produces the gating signals which cause the arithmetic unit to perform the desired functions. In addition, it provides gating pulses which fetch the operand (if needed), increment the control counter, and fetch the next instruction. Thus, the housekeeping is distributed upon the disks, so that each instruction is essentially independent of the others.

No clock is needed because each disk, as it completes its switching, causes the next instruction to be obeyed (i.e., the next (or the same) element to be addressed). Thus, the machine is not synchronous. On the other hand, it does not have the generally accepted earmarks of an asynchronous machine. Therefore, a new term has been coined to categorize this species of system: Autochronous (or self-timed).

As a consequence of autochronous operation, if the driving mechanism for a particular disk should fail, the machine will halt upon attempting to obey the corresponding instruction, rather than continuing to perform incorrect calculations as might a clocked machine when an instruction malfunctions.

It will be seen that the picoprogramming scheme may be viewed as a logical extension of a microprogrammed system, but on a more basic level. The required instantaneous levels on all the gate leads may be considered as bit values of a picoinstruction which has a word length equal to the number of gate leads. Picoinstructions are stored at constant radii upon a MYRA disk, in the proper order to perform the desired task. The advantages of the MYRA element are that the picoinstructions are automatically accessed in sequence (without the necessity of a picoinstruction counter), and successive one's or zero's in the same bit position are automatically "slurred" together, so that, for example, two successive one's produce a pulse of duration twice that of a single one preceded and succeeded by zero's. Thus, race conditions and static hazard difficulties are easily avoided.

### *Advantages*

The advantages of a picoprogrammed system are as follows:

1. *Tailorability:* Since the instructions are in effect memory elements they can be plugged in anywhere, and only their address (order code) changes; a wide range of instructions (greater than the number which the system can accommodate) can therefore be offered. Thus, for example, a customer could choose any 64 of perhaps 200 available instructions. This produces about $1.6 \times 10^{53}$ combinations (in practice, since some software, e.g., an assembler, is usually desired, a standard nucleus of instructions might be provided, and the free-

dom of choice would be somewhat reduced; even so, it should be possible to offer a range of machines extending from highly bit-manipulative to highly computational with the same mainframe).

2. *Post-Alterability:* As a corollary to item 1, a machine in the field can be altered easily if the customer decides later that his original choice of instruction set is no longer optimum.
3. *Graceful Failure:* Because each instruction is independent, the failure of one will not affect others. Thus, catastrophic (nothing works) failure should become a rarity.
4. *Diagnosable:* Because of item 3, it should nearly always be possible to successfully use some diagnostic routine. In addition, the unique modularity of the control section makes localization of a control failure easy.
5. *Easily Designed:* Disk wiring follows directly from the required timing diagram.
6. *Housekeeping Processes:* May take place anytime during the instruction execution, allowing optimum sequencing.

## THE MYRA ELEMENT

### *General*

A brief description of the operation of the MYRA element is in order. Already applied as a semiconventional memory element,[2] where its natural propensity to produce sequential trains of pulses is, to some degree, circumvented, this ability is instead capitalized upon in the picoprogramming approach.

When a step of voltage is applied across a drive winding passing through the central aperture of a square loop apertured ferrite disk in the proper remanent flux state, a current ramp will result, and a propagating flux change wave will nucleate at the central aperture, and propagate outward at a uniform velocity. As this wave traverses a portion of material singly looped by a conductor, it produces across the conductor an emf which, in a properly proportioned and constituted disk, is large enough to drive logic circuits directly. This voltage pulse is proportional in temporal length to the physical length of the looped portions of the disk. Its position in time relative to other looped portions of the disk is likewise related directly to the relative physical positions of the looped portions as shown in Fig. 1.[2]

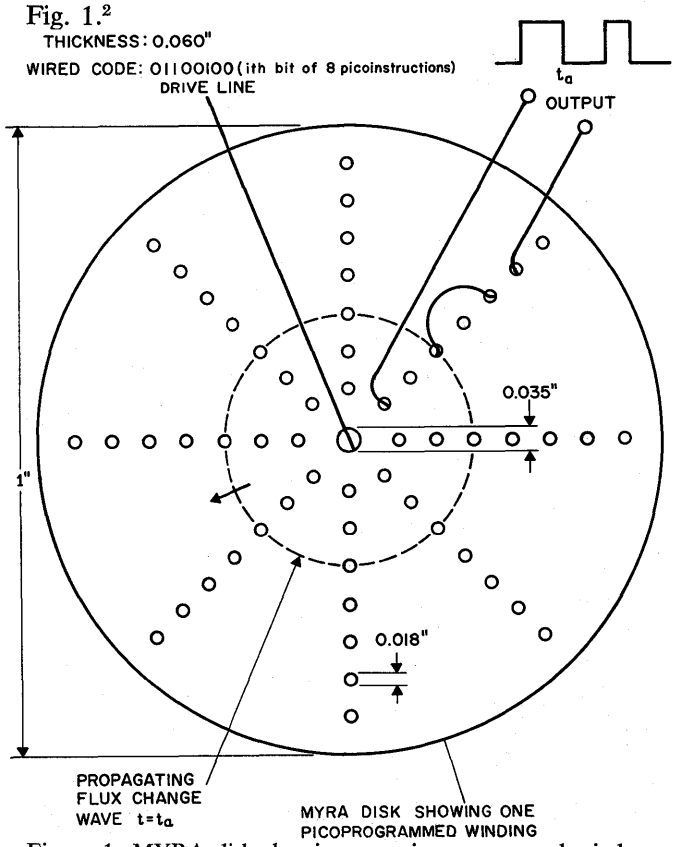



Figure 1. MYRA disk showing one picoprogrammed winding.

When the disk is reset by a drive and current in the opposite sense, a flux change wave in the opposite sense nucleates at the center (as before) and propagates outward, producing an emf in the opposite sense at the outputs of each looping conductor. If, as is usually the case, the logic elements driven are sensitive to only one polarity of input voltage, they will react only to the train of pulses produced during say, set, and not to their mirror image about ground produced during reset. However, other windings can produce the proper polarity during reset and be ignored during set. Thus, the reset time is not wasted, for useful pulses differing from (or identical with) those during set can be produced. This is somewhat analogous to playing both sides of a phonograph record.

Pulse widths as brief as 250 ns can be produced directly, and 125 ns pulses are obtainable from a disk with phased radii. The output impedance of the disks is less than 10 ohms, so that many gates may be driven without buffering, and even coaxial cable can be driven directly.

*System Considerations*

Many instructions require a waiting period in the midst of their execution (e.g., for an operand fetch from memory). It is easily possible to accommodate such delays between the set and reset of a disk.

Instructions also differ in their total time of execution. While it is possible to force all instructions to occupy the same time (dictated by the lengthiest), it is more efficient to allow differing execution times. This is effected by making use of disks which either physically or operationally differ in dimensions; the operationally small disks are of the same physical dimensions, but are only partially switched.

## WORKING PROTOTYPE

*General Description*

A feasibility model prototype dubbed PUPP (Prototype Utilizing Picoprogramming) was constructed. Its instruction repertoire includes:

1. ADD — add contents of addressed memory location to contents of accumulator.
2. STR — store contents of accumulator in addressed memory location
3. SFT
   - (a) Right — shift contents of accumulator one binary place to the right
   - (b) Left — shift contents of accumulator one binary place to the left
4. SKP — skip next instruction on non-zero accumulator
5. PCH — punch (on paper tape) contents of accumulator
6. NOP — perform no operation, proceed to next instruction
7. HLT — cease activity

Only a four instruction repertoire can be accommodated at once, one of them necessarily HLT. Each instruction is implemented in a single pluggable circuit card as shown in Fig. 2 and the instruction cards are completely interchangeable; the same

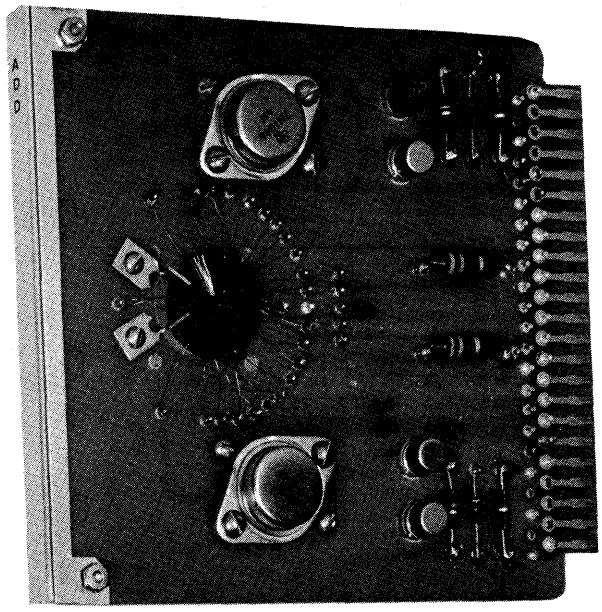


Figure 2. ADD instruction card.

instruction in a different location performs in exactly the same fashion, the only difference being that its order code (i.e., its address) changes.

HLT is a special case: the address corresponding to its order code is an empty location. When an attempt is made to access an empty location, all activity ceases, and the machine halts. The cessation of activity is quite literal because of the absence of a clock.

All flip-flops in the system are of the set-reset variety. Thus, the double rank instruction location counter requires four timed pulses for address incrementation; similarly, the accumulator is double rank. Thus, the system would be considered four-phase if a clock were used.

Memory is provided by flip-flop registers. The word length is a modest four bits, but the control signals are essentially identical with those necessary for a full size machine.

*Instruction Implementation*

The implementation of a typical instruction, ADD, will be discussed.

The pulses which this and all instructions must provide are those necessary for housekeeping, that is, incrementing the instruction location counter, fetching the next instruction, and providing a pulse which causes the next instruction to be obeyed (addressed). (See Fig. 3.)

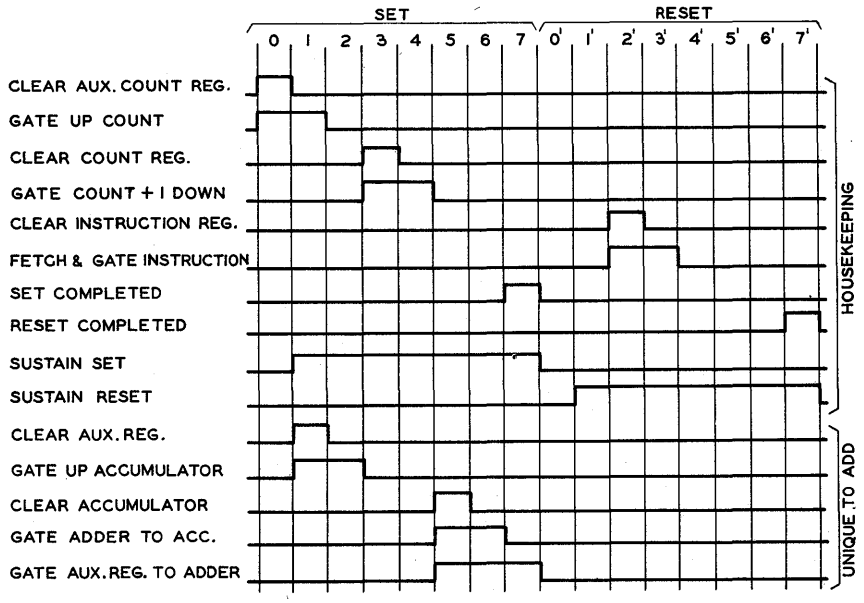



Figure 3. Pulse trains produced by one picoprogrammed MYRA disk to execute ADD (250 ns/division).

In addition to the above, there are some housekeeping pulses which are unique to the picoprogrammed implementation; among these are the sustainers, which are logically combined with a test pulse to assure the health of the driving circuits independent of the remanent state of the accessed disk. Normally, an accessed disk will always be in the proper remanent state to provide a relatively high impedance to the driving circuit. If, however, a malfunction (e.g., a power failure) should upset this arrangement, a driving circuit might attempt to switch an already-switched disk, and endanger itself; to prevent this, a narrow test pulse is first applied, then, if the device is in the proper state, the sustainer pulse sustains the drive; otherwise the drive ceases with no damage done. A SET COMPLETED and RESET COMPLETED pulse is also provided to inform the circuitry that the disk has finished setting or resetting, respectively.

Each instruction disk must, of course, provide pulses (in addition to housekeeping) which are unique to it; these are shown in Fig. 3 for ADD.

This timing diagram, which in a conventional sequential circuit control design would mark the beginning of the design problem, marks instead the completion of the picoprogramming design problem. This is true because the timing diagram is essentially identical to a cross-section of a wired disk, with a correspondence between time and radial position, and voltage and axial position of the wire.

It is, of course, understood that a control wire which performs some function such as Clear Accumulator will loop some portion (or portions) of all those disks which require clearing of the accumulator. Since only one disk can ever be in the act of switching, oring is performed by this common wire; this wire plays the same role as a sense wire in a conventional core memory, except that it fails to loop all memory locations (in general).

In Fig. 2, the windings for the ADD instruction can be seen. The two multiple turn windings are the drive windings, one used for setting, the other for resetting. These are the only multiple turn windings necessary. Note how few of the radial sets of apertures are populated with wires.

The instruction card is constructed with the heavy current drivers upon it to minimize the areas of high-current loops. This has the added advantage of making each instruction autonomous with respect to failure because any component on the card can fail without disturbing the remainder of the machine.

The disks could be driven on a matrix basis (though, of course, not by coincident current), but the economic saving would not be significant, the noise problem could become severe, and the localization of failure advantage would be lost.

*Results*

PUPP has logged over 3,000 hours of successful operation running a Markov Chain program at speeds not less than 200,000 instructions per second, and up to 300,000 instructions per second.

Complete interchangeability of instructions is demonstrably realized and electrically as well as physically smaller disks are successfully employed.

Self-induced noise is a non-existent problem; instruction cards reside comfortably beside logic cards.

Forced air cooling is used in the instruction card area.

A later instruction card design is shown in Fig. 4.

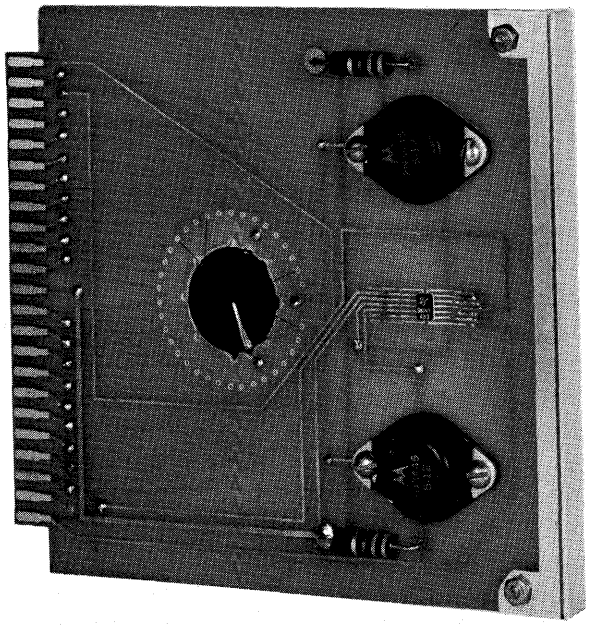


Figure 4. Improved instruction card.

*Cyclic Instructions*

There is a relatively small class of instructions such as Multiply, Divide, Shift N, etc., which are cyclic in the sense that the same sets of pulses must be made available repetitively. These instructions can be handled most easily by a set of three disks per instruction. The first (Set Up) disk performs the set-up functions, indexing, fetching the operand, placing it in the proper register, and setting certain count flip-flops; it then addresses the second disk. The second disk performs one cycle of the operation, decrements an operation counter and tests for completion of the operation; this is the Cycler Disk. This disk readdresses itself if the test indicates that the operation is incomplete; when the disk has cycled a sufficient number of times to complete the operation, it addresses the third disk. The third, or Clean Up, disk performs the remaining housekeeping operations: fetching the next instruction, etc. In addition, it performs any operations necessary for, and unique to, the completion of the instruction.

One disk could suffice for realization of a cyclic instruction, but it would require extensive inhibition logic to prevent the start-up and clean-up pulses from occurring during the cyclic portion of the operation, etc.

## FUTURES

Picoprogramming should be applicable to a wide range of computer sizes because of its mix of advantages. It should make a stored program approach feasible for very small systems because of its economic advantages; it should substantially tilt the rule-of-thumb balance between the costs of a processing unit and its control in such systems. Its flexibility and diagnostic advantages should make it attractive in rather large systems as well, particularly in multiprocessor arrangements.

## ACKNOWLEDGMENTS

I wish to acknowledge the encouragement of this study by J. E. Fulenwider and E. L. Scheuerman, the cooperation of Dr. M. E. Dempsey and R. J. Nin, and the very able services of J. R. Holden.

## REFERENCES

1. For example, D. Fagg et al, "IBM System/360 Engineering," *Fall 1964 Joint Computer Conference,* Vol. 26, Part I, Spartan Books, Inc., Washington, D.C., 1964.

2. B. E. Briley, "MYRA: A New Memory Element and System," *Proc. 1965 Intermag. Conference.*

# PRECESSION PATTERNS IN A DELAY LINE MEMORY

Stanley P. Frankel
*Los Angeles, California*
and
Jorge Hernandez
*SCM Corporation*
*Oakland, California*

## INTRODUCTION

The SCM COGITO-240 is an electronic desk calculator which makes use of one sonic (magneto-restrictive) delay line as its primary memory element. Some 480 bits of information are held in the delay line circulation pattern. These are represented in a Pulse-No Pulse code; the insertion of a pulse into, or its emergence from, the delay line at a particular moment indicates the value one for the corresponding information bit. The absence of that pulse represents the value zero. For a memory unit of this type it is convenient to recirculate information at a rate which is of the order of $10^6$ bits per second. Thus a convenient value for the delay time of the line, and for the time of one complete recirculation of the stored information, is about one-half millisecond.

In the development of the COGITO design it proved preferable to make use of a rate of *handling* of information, as for example in the performance of an arithmetic operation, which is substantially smaller than the 10 bits per second recirculation rate. One reason for this preference is as follows: The circulated information bits compose 120 "characters" of 4 bits each. (Most of these are decimal digits.) These form three "visible" registers called K (for "keyboard"), Q ("quotient"), and P (the double-length "product") register. Associated with each of these is a storage register of equal capacity, which is not displayed. Many of the operations of the calculator involve the transfer of the content of one register to another; from one visible register to another or from a visible to a storage register or conversely. These operations are facilitated by increasing the time of handling of each bit, hence the time in which it is conveniently available for such exchange processes, to cover the period in which all possible exchange partners pass through the delay line circuitry. The use of precession in the COGITO memory was, in part, motivated by this facilitation of the transfer operations.

It proved possible to provide the COGITO memory with a precession pattern which divorces the rate of information handling in arithmetic processes from the bit transmission rate of the delay line, and thereby to permit choosing each of these rates to fit the convenience of its associated circuitry. It is the

purpose of this report to describe that precession pattern and also several simpler patterns which provide some, but not all, of the desired properties. Many details of the system chosen were motivated by unusual aspects of the COGITO design and are not likely to be of widespread interest. They are not discussed here.

## TIMING CHAIN SYNCHRONIZATION

A series of flip-flops, the timing chain, serves to count the successive one-microsecond-long time intervals in which successive information bits are delivered to and received from the delay line. The timing chain is driven by a free-running oscillator, the "clock."

If the delay line were to be used merely to recirculate without change the 480 bits of stored information, then the task of the timing chain would be merely that of subdividing one "memory cycle"; that is, the period of time required for one recirculation of the stored information. That is, in fact, the task of an early part of the timing chain which serves to distinguish one from another of approximately 480 "clock periods" into which a memory cycle is divided. By reason of the precession system a longer period of time, called a "machine cycle," becomes significant. The later part of the COGITO timing chain serves to count the 60 memory cycles in each COGITO machine cycle.

In a delay line memory of this kind there arises a problem of synchronization; that is, of ensuring that information-bearing pulses emerge from the delay line in an accurately controlled phase relationship with the clock oscillation. A straightforward way of ensuring synchronization is to impose rigid control on the frequency of the oscillator and on the delay time of the line, and to adjust one or the other of these parameters so as to bring about the desired phase relationship. As a measure of the necessary rigidity of control it may be noted that in a machine like COGITO a long-term drift in either parameter of 0.1 percent would be intolerable.

Hindall[1] has described a method of synchronization which obviates the need for rigid long-term stability of these parameters. He uses a delay time, and therefore also a memory cycle length, which is substantially longer than the time required for the insertion of the entire body of stored information into the delay line. In each memory cycle a "marker pulse" which is distinguishable (for example, by greater magnitude) from the information pulses is set into the line before the insertion of the stored information. After the entire block of stored information has been received from, and reinserted into, the delay line there occurs a "silent period" in which no further information is received from the line. During the silent period the clock oscillator is disabled; that is, its oscillation is suppressed. The emergence of the marker pulse from the delay line, somewhat later, marks the end of the silent period and brings about the release from inhibition of the clock oscillator. The timing of the succeeding activities is controlled by the now-enabled clock. These succeeding activities are: the insertion into the line of a new marker pulse, the receipt from and the reinsertion into the line of the pulses representing stored information, the disabling of the oscillator for the following silent period, etc.

COGITO makes use of a method of synchronization which is distinguished from that of Hindall in that the marker pulse does not differ from the information pulses in magnitude or the like. Rather it is recognized as the marker pulse by reason of its emergence from the delay line during a silent period. That is, the first pulse to emerge after the clock has been disabled is accepted as the marker pulse and terminates the period of inhibition of the clock oscillator.

The silent period provides a convenient reference point for the description of the memory cycle. In the following the term "memory cycle" will be used to refer to a period of time which begins in one, and ends in the next succeeding, silent period.

With either the Hindall or the COGITO method of synchronization the oscillator frequency and the delay time of the line may, without harm, drift gradually; the duration of the silent period will change continuously to accommodate these drifts. (It must not, of course, be allowed to shrink to zero.) The possibility of *continuous* change in the length of the silent period, consistent with the desired synchronization, arises from the suppression of oscillation of the clock. During the silent period all phase relations from the previous memory cycle, in which a marker pulse and the block of information pulses were inserted into the line, are forgotten. After the silent period the phase of oscillation of the clock is determined by the time of emergence of the marker pulse and is thus consonant with the times of emergence of the information pulses. Although these two methods of synchronization provide tolerance of

gradual changes in the two parameters discussed, a sudden change in either, that is, a substantial change occurring within one memory cycle, would still lead to malfunction. Fortunately, such sudden changes are much more easily prevented than are long-term drifts.

## PRECESSION PATTERNS

In a simple recirculating memory using the COGITO synchronization the duration of the memory cycle is equal to the delay time of the line, together with its associated circuitry, since each pulse is reinserted into the line simultaneously with its emergence. The word "simultaneously" must not be interpreted very literally, since the time of traversal of the associated circuitry is substantial. More precisely: the recognition of an emerging pulse permits the introduction into the line of a pulse which is of well-standardized magnitude, duration, and phase with respect to the clock oscillator. Figure 1 shows the simple recirculation without precession of a marker pulse and a group of information pulses, with the emergence and reinsertion shown as "simultaneous" in this conventionalized sense.

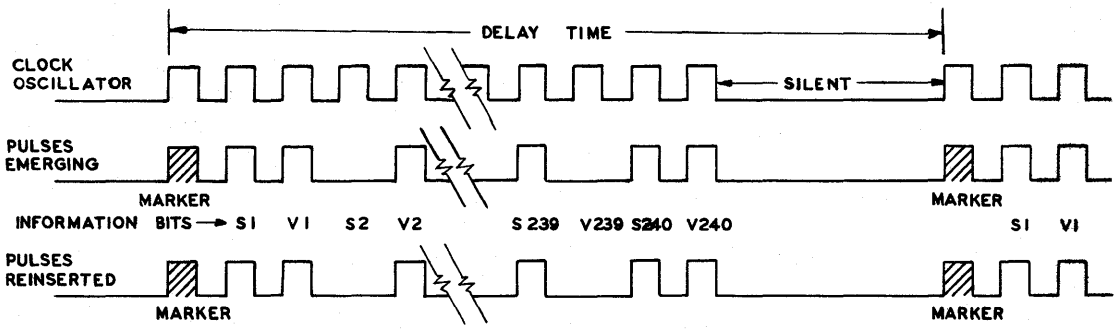



Figure 1. Recirculation without precession.

One-half of the 480 bits of memory held in COGITO, called "V-bits," represent the numbers held in the 3 visible registers while the remaining 240 bits, called "S-bits," form the storage registers. One S-bit and the corresponding V-bit form a "bit-pair." It proves convenient to handle the two bits of a pair together, for the most part, and to make them available for manipulation over periods of time considerably longer than a few microseconds. A simple way in which that can be done is illustrated in Fig. 2. The information bits which emerge from the delay line in the first memory cycle are named

s1, v1, s2, v2, . . ., s239, v239, s240, v240

in the order of their appearance following the marker pulse. The first two bits, s1 and v1, form one bit-pair; the following two another pair, etc. As the first two bits emerge they are captured in two flip-flops, S and V respectively, and are not simultaneously reinserted into the delay line. The following 478 bits are reinserted immediately upon emergence. Following the insertion of bit v240, the bit (s1) held in S is inserted, and after it the bit (v1) is inserted into the line from flip-flop V. Only then is the clock oscillator inhibited in order to begin a silent period. The marker pulse which emerged immediately before s1 was not reinserted immediately but was inserted only at the time of emergence of the second information bit, v1. In that way the introduction of a gap between the marker pulse and the first information bit is avoided. In the second memory cycle the information bits emerge in the sequence

s2, v2, s3, . . ., v239, s240, v240, s1, v1

and the first two bits, s2 and v2, are captured in S and V and are held for later reinsertion in the same way as the first pair was earlier, etc. It will be seen that in each memory cycle the sequence of 240 bit-pairs is cyclically permuted and that after 240 memory cycles the original sequence has been restored.

Figure 2 has been drawn so as to emphasize another feature of this simple precession pattern: the duration of the memory cycle is greater by two clock periods than that shown in Fig. 1. It is also to be noted that the "early part" of the timing chain

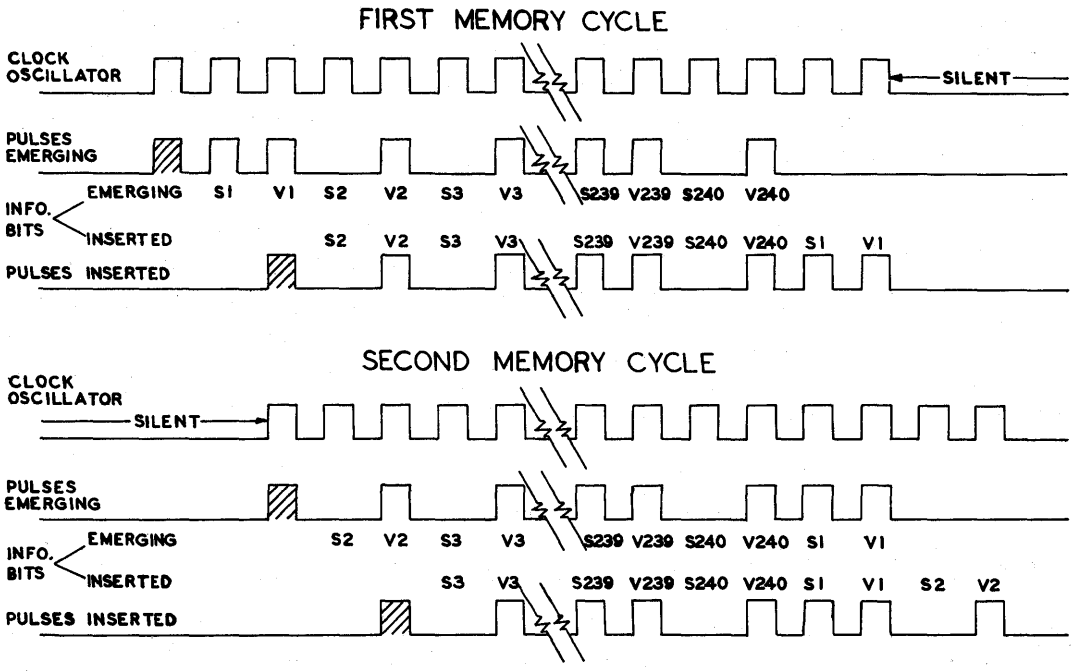



Figure 2. A one-group precession pattern.

must now distinguish 482 rather than only 480 clock periods following the appearance of the marker pulse. Atypical activities occur in the first two and in the last two of these.

In the system of Fig. 2 each V-bit is held in flip-flop V throughout one memory cycle (perhaps excepting the silent period) and is available there for leisurely manipulation, and one S-bit is similarly held in S. After one machine cycle, consisting of 240 memory cycles, all information bits have thus been held and the original bit-configuration has been restored. It did not prove convenient to use in COGITO a machine cycle quite so long as 240 memory cycles (about one-eighth of a second) and therefore a slightly more complex precession pattern was considered.

A further classification of the information-bits held in the COGITO memory must now be described. Most of these bits represent the decimal digits which constitute numbers held in the various registers. Each decimal digit is represented by four bits, called t1, t2, t3, t4 in order of increasing significance. (A simple 1,2,4,8, BCD representation is used.) It therefore proves convenient to organize all other information held—decimal point positions, plus or minus signs, etc.—in similar 4-bit characters. Thus the entire body of stored information may be divided into 4 equal parts; a group of 120 bits (60 bit-pairs) which are t1-bits, 120 t2-bits, etc. The transfers of numbers from one register to another respects this separation into four groups; in such a transfer a t1-bit always remains a t1-bit, etc. Thus it proves convenient to separate the body of stored information into four parts, and to introduce a precession within each part separately. Such a precession pattern is shown in Fig. 3.

To reflect the separation into four groups the 480 bits held in memory are renamed as follows. The 120 t1-bits are called

$$s^1 1,\ v^1 1,\ s^1 2,\ v^1 2,\ \ldots, s^1 60,\ v^1 60.$$

Similarly the group of t2-bits carries the superscript $^2$, etc. In the first memory cycle (of a machine cycle) these 480 bits emerge from the delay line in the order named: the 120 t1-bits follow immediately after the marker pulse, then the t2-bits, etc. The first information bit, $s^1 1$, is copied into flip-flop S and is not immediately reinserted. Then the second bit, $v^1 1$, is copied into flip-flop V and a marker pulse is inserted into the line at this time. (It is the first pulse inserted since the silent period.) In the following 118 clock periods the remaining bits of the t1-group are reinserted immediately upon emergence. Then, however, when the first bit of the second group, namely $s^2 1$, emerges from the line it is exchanged with the content of flip-flop S. That is, the emerging bit is set into flip-flop S while the prior content of S is returned to

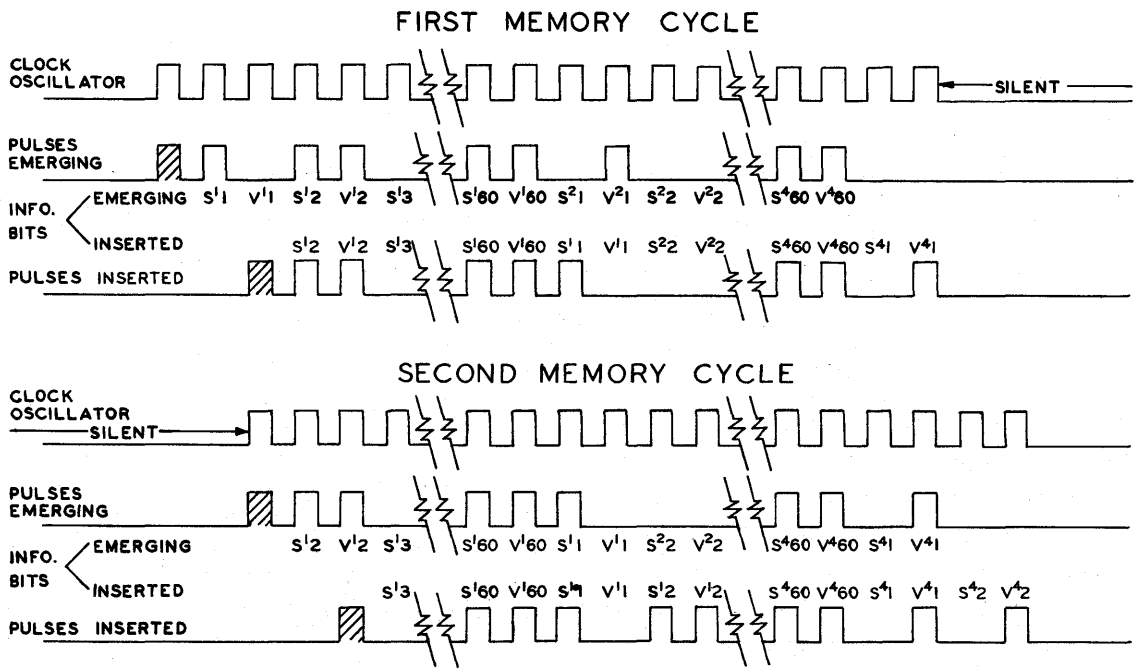



Figure 3. A four-group precession pattern.

the delay line. In the next clock period the content of flip-flop V, namely $v^1 1$, is set into the line and the bit $v^2 1$ is placed in flip-flop V. The remaining bits of the second group are then reinserted as they emerge. Similarly, the first two bits of the third group are exchanged with the contents of flip-flops S and V and the rest reinserted, and similarly during the emergence of the fourth group. After the emergence and reinsertion of the last bit of the fourth group ($v^4 60$), the contents of flip-flop S and V are inserted into the delay line in two further clock periods in the same way as has been described for the simpler precession pattern of Fig. 2. The bits thus returned to the line are $s^4 1$ and $v^4 1$ respectively.

As can be seen in Fig. 3, the operations just described result in a cyclic permutation of the 60 bit-pairs of each group separately—together with a rightward displacement of the entire pattern in the same way as in Fig. 2. After one machine cycle, consisting of 60 memory cycles, the original configuration has been restored. During that machine cycle each V-bit has been held in flip-flop V, and each S-bit in S, for one-fourth of one memory cycle (with the neglect of the silent period).

In the discussion above attention has been directed to the circulation and precession of the bits of information held in storage in a delay line. The possibility that the *value* of an information-bit may have changed by reason of an inter-register, or an arithmetic operation, etc., has not been mentioned. Each of the symbols used, such as $v^1 1$, should, however, be understood to represent merely the *name* of a variable which may change its *value* from time to time by reason of activities not described.

The precession pattern illustrated by Fig. 3 fails to provide one essential feature of COGITO. Each V-bit (that is, each bit of a "working register") upon being picked up into flip-flop V must be provided with opportunity for leisurely interaction with the fourth bit to precede or succeed it in occupancy of flip-flop V (that is, the corresponding bit in another register, with which it may be involved in arithmetic manipulation.) For this reason a bit which has been held in V for a quarter of a memory cycle (called a "bit period") is not, in fact, returned to the precession pattern as has just been described. Instead, it is set into a 4-flip-flop shift register in which it remains easily accessible for 4 additional bit periods, that is, for an additional one memory cycle. A bit which is held in storage in flip-flops in this way may be changed in value in any one of these 5 bit periods. After this holding period the (possibly modified) V-bit is returned to circulation as illustrated in Fig. 4. By reason of the general precession the time for rein-

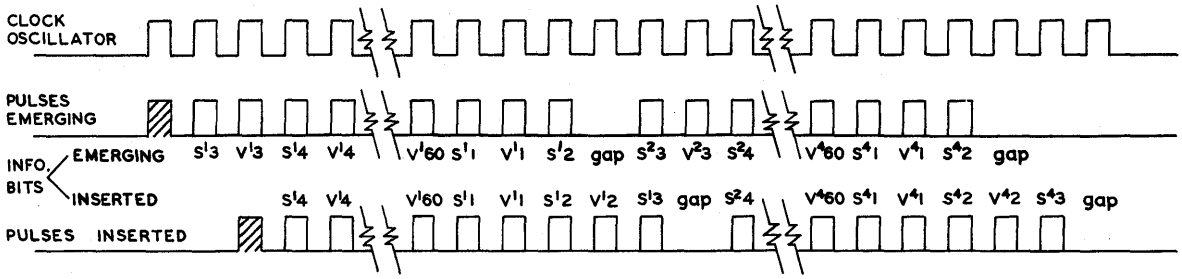



Figure 4. COGITO precession pattern (shown for third memory cycle of a machine cycle). The information bits are shown as ambiguous, pulse present or absent.

sertion of the bit which has been held out of circulation for an additional memory cycle is immediately before, rather than after, the clock period in which the bit held in S is reinserted. In the clock period following the reinsertion of the S-bit no pulse is set into the line, thus the bit pattern shown in Fig. 4 has one-clock-period-long gaps there. The "bit value" shown with these gaps is zero. The omission of four bits from the pattern shown in Fig. 4, which is otherwise like that shown in Fig. 3, can be understood as arising from the fact that at each moment four bits are held out of circulation in the flip-flop shift register.

## CONCLUSION

These examples illustrate the considerable flexibility of delay line storage systems provided with simple precession patterns. It seems likely that similar techniques will prove helpful in many situations in which the desired rate of handling of data is smaller than that convenient for a delay line memory.

## REFERENCES

1. L. D. Hindall, "Self Synchronous Delay Line," United States Patent No. 2,783,455, issued Feb. 26, 1957.

# AN ASSOCIATIVE PARALLEL PROCESSOR WITH APPLICATION TO PICTURE PROCESSING*

R. H. Fuller and R. M. Bird
*General Precision Inc.*
*Librascope Group*
*Glendale, California*

## INTRODUCTION

In recent years, a number of hardware associative memories had been designed and experimentally verified.[1,2] These memories allow simultaneous comparison of all stored data to external data. Data may be read from, or written into, comparing words. These memories, acting as peripheral devices to conventional computers, have been studied for application to various tasks described in references 1 and 2. The concept of "associative processing," i.e., simultaneous transformation of many stored data by associative means, has been described previously.[3,4,5] This processing mode showed promise in a variety of tasks, but was not efficient when peripherally controlled by a conventional machine. Novel machine organizations were required to fully exploit the potential of these techniques for solving poorly structured nonnumeric problems, at which present-day machines are not efficient.

This paper describes a novel Associative Parallel Processor (APP), having an associative memory as an integral part of the machine. Arithmetic algorithms are described which allow it to perform adaptive pattern recognition by evaluating threshold logic functions. Novel algorithms allow simultaneous processing of many operands in bit-parallel fashion. The processor is a stored program device with a powerful command set, and thus has general utility in problems which allow a single set of commands to be executed independently, and thus simultaneously over many data sets. These conditions frequently arise in nonnumeric data processing tasks such as pattern recognition.

Parallel processing is accomplished within the associative array of APP by the powerful technique of "sequential-state-transformation," previously described by one of the authors.[3] The parallel search function of associative memories requires that comparison logic be provided at each memory word cell. The APP, by moderate additions to this logic, allows the contents of many cells, selected on the basis of their initial content, to be modified simultaneously through a "multiwrite" operation. Content search and multiwrite are the primitive operations necessary to parallel processing by sequential-state-transformation.

*The work repored here was supported by AF Rome Air Development Center, Griffiss Air Force Base, N.Y., under Contract AF 33 (602)-3371.

To illustrate the concept of sequential-state-transformation, consider an associative memory which stores two operands, $A_i$ and $B_i$, in each word of memory. We desire to add operand $A_i$ to operand $B_i$ simultaneously in some subset of these words. Processing is serial by bit, and parallel by word, starting at the least significant bit of each field. Each word has an auxiliary storage bit, $C_i$ stored within the memory array. Bits within operand field $A_i$ are designated $A_{ij}$ ($j$=1,2,. . ., $N$), where N is the field length. Bits in field $B_i$ are similarly designated. The truth table defining the addition is as follows:

| State Number | Present State | | | Next State | |
|---|---|---|---|---|---|
| | $A_{ij}$ | $B_{ij}$ | $C_i$ | $B_{ij}$ | $C_i$ |
| 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 | 0 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 0 | 0 | 1 | 0 |
| 6 | 1 | 0 | 1 | 0 | 1 |
| 7 | 1 | 1 | 0 | 0 | 1 |
| 8 | 1 | 1 | 1 | 1 | 1 |

Note that variables $B_{ij}$ and $C_i$ differ in their present and next states only in states numbered 2, 4, 5, 7. The search and multiwrite operations may be used to perform the addition of all number pairs, starting at the least significant bit, as follows:

1. Search words having $A_{ij} = 1$, $B_{ij} = 1$ and $C_i = 0$.
   For these words multiwrite $B_{ij} = 0$, $C_i = 1$.
2. Search words having $A_{ij} = 0$, $B_{ij} = 0$ and $C_i = 1$.
   For these words multiwrite $B_{ij} = 1$ and $C_i = 0$.
3. Search words having $A_{ij} = 0$, $B_{ij} = 1$ and $C_i = 1$.
4. Search words have $A_{ij} = 1$, $B_{ij} = 0$ and $C_i = 0$.
   For these words multiwrite $B_{ij} = 1$.

Steps (1) through (4) are repeated at each bit of the operands. Within each bit time, processing is sequential by state over present states which differ from the next state in one or more variables. All words in a given present state are transformed simultaneously to the desired next state.

Sequential-state-transformations used to perform the above word-parallel, bit-serial addition, is evidently a very general mode of associative processing. It allows transformation of memory contents according to any Boolean function of stored and external variables. It makes full use of comparison logic, implemented at the bit level within an associative array, and thereby simplifies logic required at the word level. It compares favorably with other associative processing methods in both speed and processor complexity.

In the next section we describe the organization and command set for a processor using the sequential-state-transformation mode of associative processing. In the following section command routines are given for word-parallel, bit-serial processing described above; and also for a novel mode of word-parallel, bit-parallel processing which yields significant speed improvement over bit-serial mobes. The pattern-processing applications is discussed last.

## ORGANIZATION AND COMMAND SET

The addition operation presented in the preceding section is a typical example of the associative processing technique. From it, several conclusions concerning the desired structure for an associative processor can be formulated.

1. The single primitive step in associative processing is identification of all words storing some configuration of binary state variables, followed by binary complementation of some state variables within identified words. This primitive makes the basis of an associative "micro instruction" which, repeatedly executed with varying parameters, can transform memory contents according to any Boolean function of stored and external binary variables.
2. Since processing is simultaneous over all stored data, no explicit word address is provided within an associative instruction. Many words may be transformed in response to a given instruction. These words are identified by search criteria contained within the instruction.
3. Data is processed column-serially to minimize the number of state variables and thus memory states which must be identified and transformed sequentially. Associative micro instructions thus address a small number of bit columns in memory. Since many consecutive bit columns are sequen-

tially transformed, efficient means for column indexing are required.

4. Several temporary storage or "tag" bits within each word are useful to identify words as members of various sets currently undergoing transformation. The carry storage bit, defined for the addition task of the previous subsection, is a tag bit. The location of tag columns are unchanged as successive data columns are processed.
5. Each word cell must have electronics, external to the memory array, which temporarily store the match status of the word, relative to the most recent search criteria, and allow writing of selected bits in matching words to either the one or zero state. Writing is simultaneous over all matching words.
6. For generality, stored program control of the associative processor is desired. Instructions are accessed sequentially from control memory, with possible branching as in conventional machines. Since no benefit derives from storing these instructions in associative memory, the control memory is a less costly location-addressed random-access memory. Having separate instruction and data memories, the access times for each may be overlapped.

Elements of the Librascope processor are shown in Fig. 1 This realization contains an associative array, partitioned into data and tag columns (fields), together with requisite word and digit electronics. Instructions are read from a random-access control memory, and are interpreted by control logic to sequence operations within the processor. The associative array may be loaded word-serially through the data register and unloaded column-serially through word electronics. The data register also stores search keys. The random-access memory is written or read through the data register. Contents of word flip-flops (i.e., match status of words) may be transferred to the data register for subsequent use as a search key or to be outputed. Column indexing, over two distinct fields ($A$ and $B$), is provided by the $A$ and $B$ counters. $A$ and $B$ limit registers store either the upper or the lower limit column for their respective fields.

The electronics repeated at each word cell (Fig. 2) contains a match storage flip-flop and a word driver capable of writing ones or zeros into selected bits of matching words. The state of each match flip-flop is fed to a "match status gate" whose output denotes the presence or absence of matching words. The group of match flip-flops, termed a detector plane (DP), may be set collectively to one or zero by signals $E_{s1}$ or $E_{s0}$ common to all match flip-flops. Prior to writing into lower neighbors, contents of all match flip-flops are shifted down one word position.

Operation of the processor required the following steps:

1. Load instructions to control memory.
2. Load data to associative array.
3. Process data according to state transforming instructions read from the control memory.
4. Output results.

Instructions are fed to the data register then written into the control memory by action of the central control. Data are likewise fed to the data register then written into the associative array by data and word drivers. The associative array is content-addressed rather than location-addressed. Prior to writing word zero, the match flip-flop for this word is set to one. The first data element is written into this "matching" word. Subsequent words are written after shifting this one into consecutive match flip-flops.

Data within the associative array are processed by use of the associative command described below. Following associative processing the output data may be a single quantity (e.g., a picture "name") or may be the entire transformed contents of the associative array (e.g., field potentials at all nodes in a discretized solution space. Either class of data may be efficiently outputed through the match flip-flops. The contents of a column within the associative array are read to match flip-flops by a search on that column then transferred to the data register for output. The alternative of reading words sequentially from the associative array requires the availability of bit sense amplifiers and word-read drivers together with means for addressing word-read drivers. The latter readout scheme would be more costly in hardware with no gain in efficiency.

The format for associative commands is shown in Fig. 3. Each associative command effects a primitive transformation of state variables as discussed in the preceding section. The left-most bit identifies

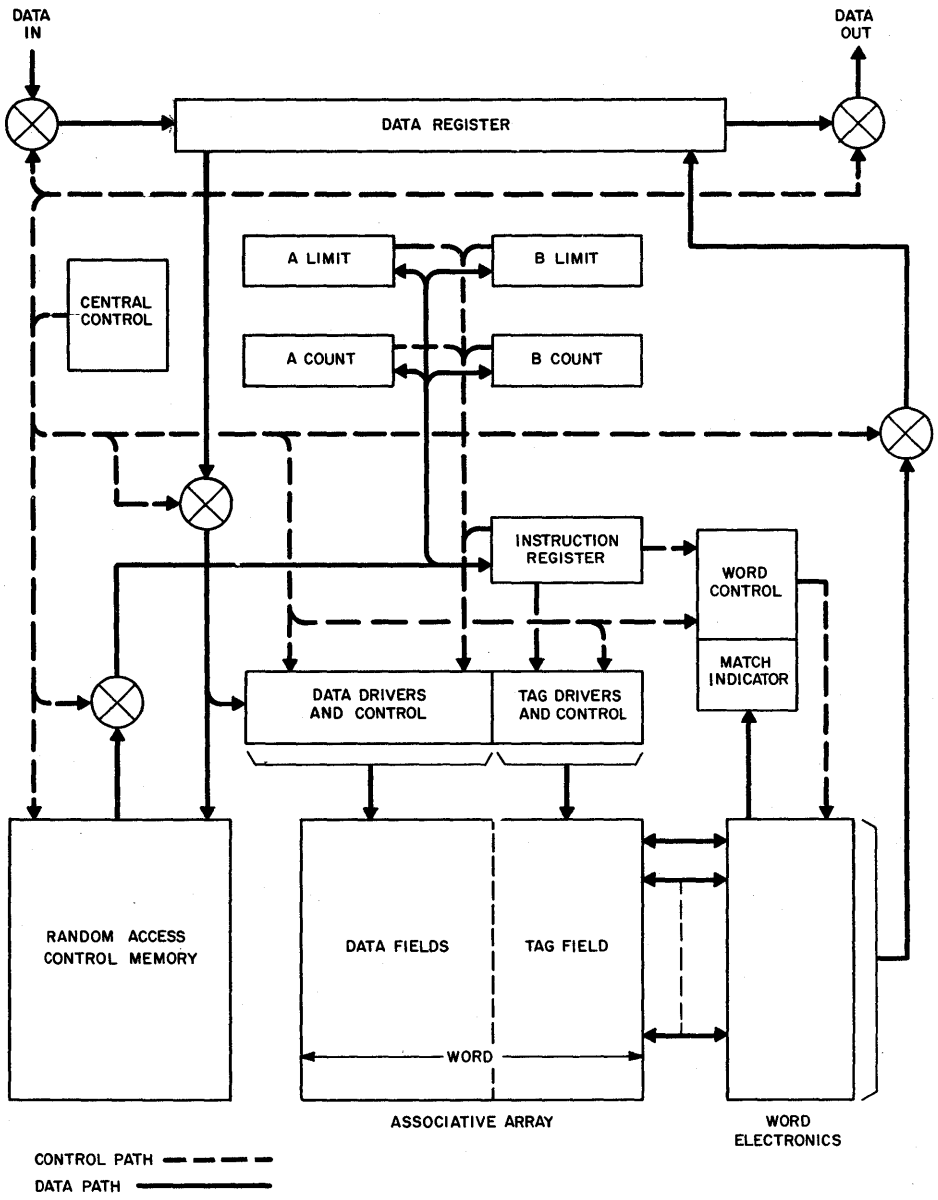



Figure 1. Structure of the associative parallel processors.

the command as associative. The two adjacent bits define the initial state of match flip-flops in word logic units (i.e., the detector plane). Other bits define search and rewrite criteria for the *A* field, the *B* field, and for each of four tag bits. The rightmost bit controls rewrite into matching words or their next lower neighbors. Functions of these bits are described in Fig. 3.

To illustrate the utility of this command, consider the task of searching the associative memory for words matching the data register over a field having it upper limit stored in the *A* limit register and its lower limit stored in the *A* counter. Matching words are to be tagged in tag bit 1.

The following command accomplishes the desired tasks:

1 $E_{si}$ S L D W I S – – W – S O W

A Control B Control Tag 1

The following routine loads data into each word in the associative array. The word field to be writ-

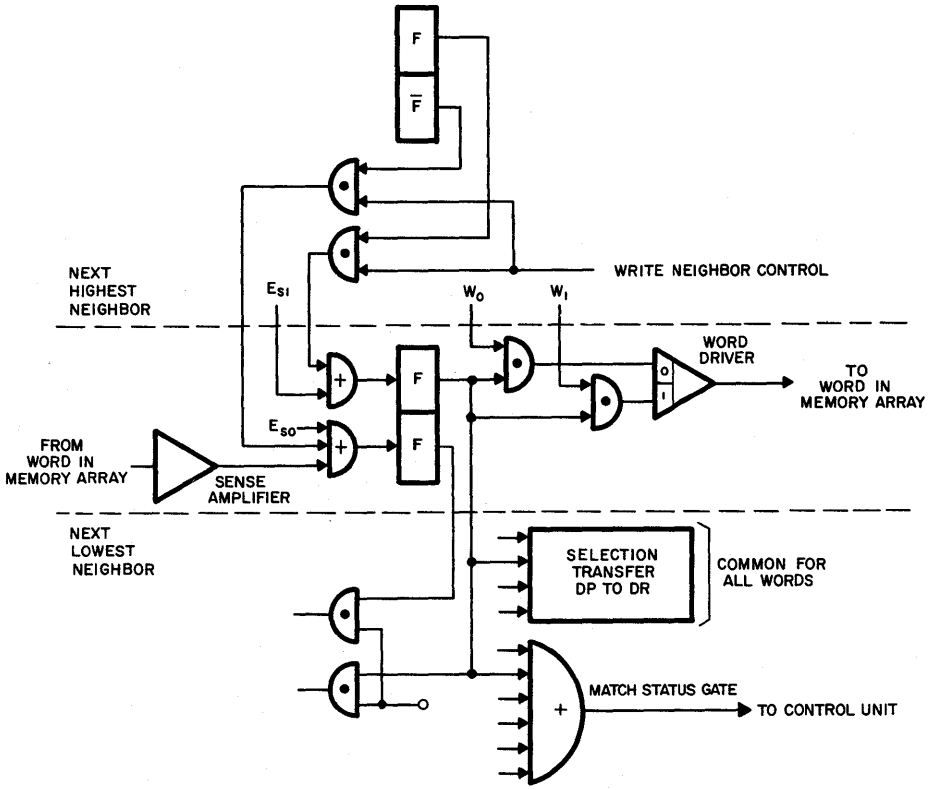



Figure 2. Word electronics.

ten is again defined by contents of the $A$ counter and the $A$ limit register:

1. Set the match flip flop for word 0 to "1."
2. 1 N S L D W S – – – – S – W N
3. 1 N S L D W S – – – – S – W L
   A Control B Control Tag 1
4. If not match, exit: otherwise go to (3).

Instruction (2) writes into word 0; instruction (3) writes sequentially into each remaining associative word.

Nonassociative commands are provided to load the $A$ and $B$ counters and limit registers, to branch from linear instruction sequencing either unconditionally or when specified conditions are met, and to input or output data. Nonassociative commands are specifically defined in the illustrative programs presented in the next section.

## ARITHMETIC ALGORITHMS

In previous parallel processors[4,5,6] arithmetic algorithms were typically executed over many operands simultaneously, but in bit-serial fashion. All bits of an operand are stored within a single word cell as shown in Fig. 4. For operations requiring two operands, operands may be paired by storing them in the same cell or by restricted communication between word cells (e.g., communication between "nearest neighbors" only). This word-per-cell ($W/C$) organization is efficient when all or most operands in memory are processed by each associative command.

An alternate data organization stores bits of an operand in separate contiguous word cells as shown in Fig. 5. A similar organization, independently derived, was recently described. The bit-per-cell ($B/C$) organization allows many operands to be processed simultaneously in bit-parallel fashion. Command execution is thus appreciably faster than for the $W/C$ organization, but each command is typically executed over fewer operand pairs. Any operands stored in the same set of contiguous word cells may be simply paired. The $B/C$ organization is thus efficient for problems which do not allow simultaneous processing of all operands by a single command, and for problems in which each

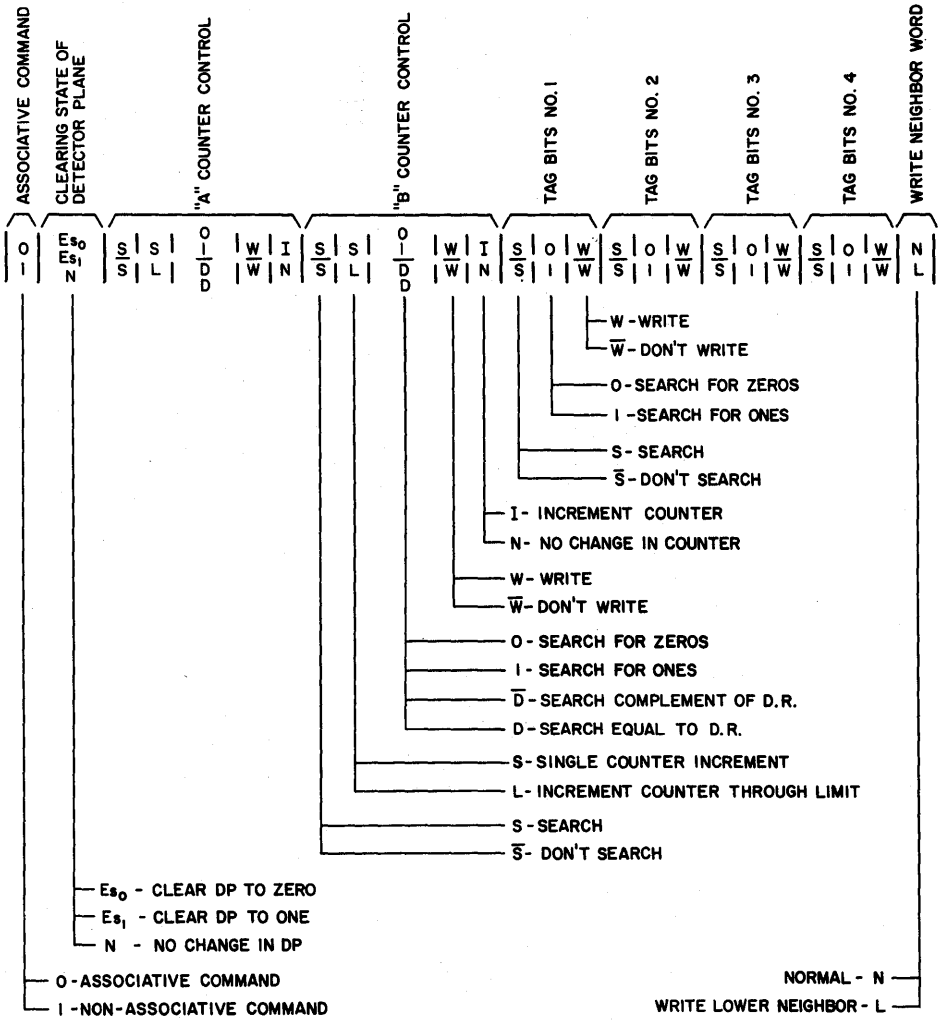



Figure 3. Format for associative command.

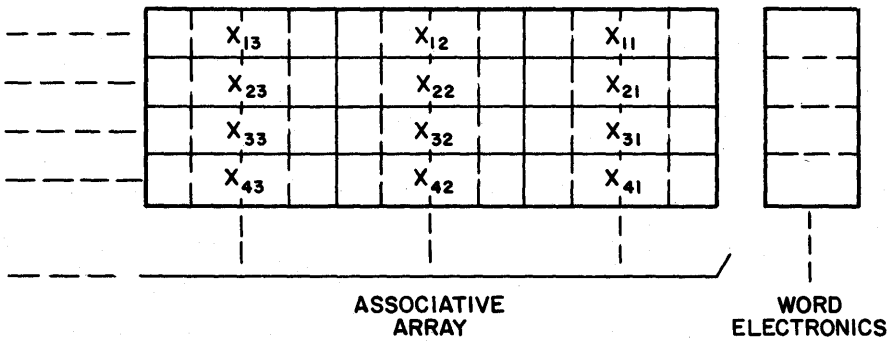



Figure 4. Word per cell data organization.

operand must be paired with many others at various computational steps.

The processor organization and command set described in the preceding section is equally applicable to *W/C* and *B/C* data organizations. Arithmetic algorithms, appropriate to each data organization, are presented below.

Consider first an algorithm for the *W/C* data organization (Fig. 4), which adds contents of all *A* fields to contents of respective *B* fields, leaving the resulting sum in the *B* fields. Tag 1 is used for carry storage and is assumed initially cleared. The routine is as follows:

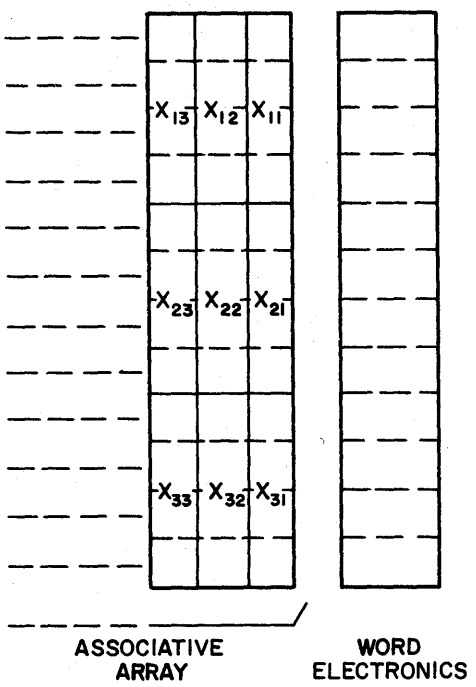



Figure 5. Bit per cell data organization.

| Cell | Contents | | | |
|---|---|---|---|---|
| | *A Field* | *B* Field | Tag 1 | |
| 0 | 1 $E_{s1}$ | *S S* 1 *W N* | *S S* 1 *W N* | *S* 0 *W* |
| 1 | 1 $E_{s1}$ | *S S* 0 *W N* | *S S* 0 *W N* | *S* 1 *W* |
| 2 | 1 $E_{s1}$ | *S S* 0 *W N* | *S S* 1 *W N* | *S* 1 *W* |
| 3 | 1 $E_{s1}$ | *S S* 1 *W I* | *S S* 0 *W I* | *S* 0 *W* |
| 4 | If *A* Count > *A* Limit, continue; otherwise jump to (0). | | | |

The routine first addresses least significant bits of *A* and *B* fields and tag 1. Of eight possible states for these v ariables, four must be transformed to new states (see Introduction). Commands 0-3 accomplish the four required state transformations. Command 3 increments *A* and *B* column addresses. The routine is repeated at each column in the *A* (and thus *B* ) field.

A routine which performs the equivalent operation for operands stored in the *B/C* organization (Fig. 5) is shown as follows:

| Instruction No. | | | *A* Field | *B* Field | Tag 1 | Lowest Neighbor Control |
|---|---|---|---|---|---|---|
| 0 | 1 | $E_{s1}$ | *S* – 1 *W N* | *S* – 0 *W N* | – – – | *N* |
| 1 | 1 | $E_{s1}$ | *S* – 1 *W N* | *S* – 1 *W N* | – – – | *N* |
| 2 | 1 | *N* | *S* – – *W* – | *S* – – *W* – | *S* 0 *W* | *L* |
| 3 | 1 | $E_{s1}$ | *S* – – *W* – | *S* – 0 *W N* | *S* 1 *W* | *N* |
| 4 | 1 | $E_{s1}$ | *S* – – *W* – | *S* – 1 *W N* | *S* 1 *W* | *N* |
| 5 | 1 | *N* | *S* – – *W* – | *S* – – *W* – | *S* 0 *W* | *L* |
| 6 | 0 | *N* | If DP $\neq$ 0 jump to (3). | | | |

Instructions 0, 1, and 2 form the partial sum. Instructions 3, 4, and 5 ripple all carries to completion, as is detected by instruction 6. For a worst-case carry, $N$ -1 iterations of steps 3, 4, and 5 are required.

However, even when the number of parallel data words is very large, the longest expected carry string is significantly less than $N$ -1 bits. Typically, the foregoing algorithm is two to three times faster than for the algorithm presented for the $W/C$ configuration.

## ASSOCIATIVE PATTERN PROCESSING

A number of linear threshold-pattern-recognition devices have been built using analog techniques.[8,9,10,11] Such devices are relatively fast and inexpensive when applied to simple pattern recognition tasks. They are limited in the allowed number of input variables and in the dynamic range of weights assigned to these variables. Wiring complexity increases rapidly with the number of threshold units used. Modification of the weights assigned input variables or of thresholds is expensive and time-consuming. Higher-order restructuring is even more difficult. Analog units are thus not suited for classification of complex problems for which many properties are measured, and where suitable properties may not be known a priori.

The parallel processing capability of an associative processor is well suited to the tasks of abstracting pattern properties and of pattern classification by linear threshold techniques. Threshold pattern recognition devices execute a given operation independently over many data sets, and thus allow the parallelism necessary for efficient associative processing. Associative processing affords the accuracy of digital number representation, and is thus unlimited in fan-in and dynamic range of weights. Weights are simply altered by changing memory contents. Wiring and components are regular and are thus amenable to low-cost, batch-fabrication techniques. The set of measured pattern properties is changeaable by changing memory contents, rather than by rewiring as for analog units. Adaptation is thus possible in measured properties as well as in classification.

### *The Pattern-Processing Model*

In this subsection, the pattern-processing model will be briefly described. Figure 6 represents the model of the pattern recognition system. "$N$" binary valued sensor units are summed, with weights $\pm 1$, into some or all of "$K$" thresholding logic units. A threshold level, $t_k$, is established for each logic unit. If the sum of weighted inputs exceeds the threshold, the unit becomes active and the output, $b_k$, is one; otherwise the output is zero. Each logic unit has a weighted connection to some or all of $N_r$ response units. Weights of active logic units are summed and thresholded at each response unit. A pattern is classified according to the set of activated response units.

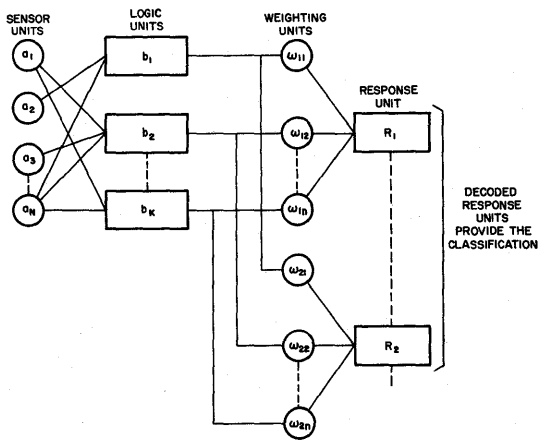



Figure 6. Analog model of pattern recognition system.

## Associative Realizations

The associative memory is organized into three sections containing, respectively, the connectivity vectors $C_k$, $1 \leq k \leq K$; the system of weights $w_{kn}$, $1 \leq k \leq K$, $1 \leq n \leq N_r$; and the target vectors $\tau^m$, $1 \leq m \leq M$. The general organization of the associative memory is shown in Fig. 7, which is interpreted as follows: In Phase (1), the set of logic units activated by the $i$th pattern is determined, using the input vector $A^i$ and the stored connectivity vectors $C_k$. Logic unit outputs which yield the property vector $B^i$ are formed in the detector plane. In Phase (2), the inputs to the response units are calculated, using the vector $B^i$ and the weights stored in the appropriate portion of the associative memory. This yields the response vector $R^i$ in the detector plane. In Phase (3), the response vector $R^i$ is compared with the target vectors $\tau^m$ stored in the associative memory, and the classification of the pattern associated with $A^i$ is determined. The three processing phases are further described as follows:

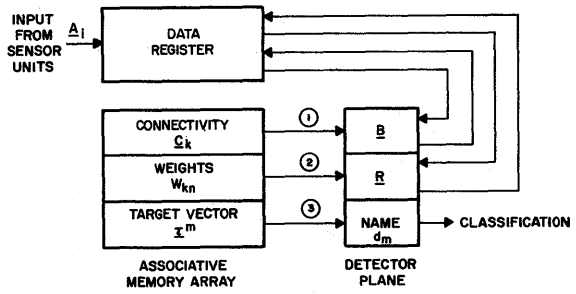



Figure 7. Associative parallel processor realization of pattern recognition system.

*Phase (1).* The set of sensor-logic unit connections with weights 1 and $-1$ may be represented by a matrix $[C]$ which is stored in the connectivity sector of an associative memory in the format of Fig. 8. The matrix $[C]$ may be written

$$[C] = [C^+] + [C^-]$$

where $[C^{+(-)}]$ is a matrix with binary values whose entries represent those connections which are positive (negative). That is, the element $c^+_{jk}$ is $+1$ if there is a positive connection between sensor $j$ and logic unit $k$, and is 0 otherwise. Similarly $c^-_{jk}$ is 1 if there is a negative connection between sensor $j$ and logic unit $k$. Thus, all connections from sensors to the $k$th logic unit are represented by the row vector $C_k = C^+_k + C^-_k$. Matrix rows $C^+_k$ and $C^-_k$ are stored in adjacent fields of as associative memory word.

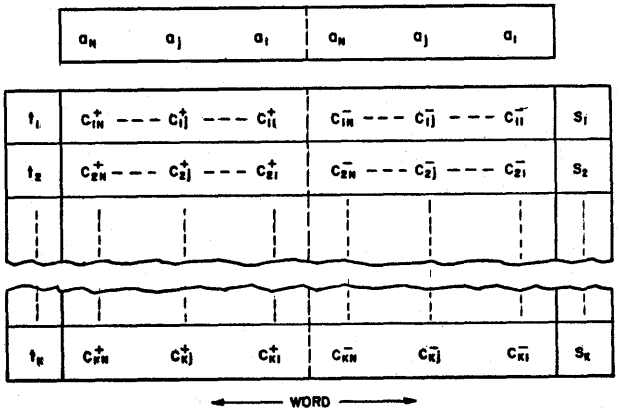



Figure 8. Connectivity sector of associative parallel processor.

The input vector $A^i$ is used as a key to interrogate matrix [$C$] in the associative memory in order to determine the set of logic units activated by the input vector $A^i$. Bit positions having $a^i_j = 1$ are interrogated for $c^+_{jk}$ or $c^-_{jk} = 1$. A binary count of the number of bits in $C^-_k$ which satisfy this condition is made in the accumulation fields $S_c$ respectively of each word in Fig. 8. The procedure is again repeated for $C^+_k$. Thresholds $t_k$ are prestored in fields as indicated. After counting, the contents of the $T$ fields are subtracted from those of fields $S_c$. Elements of $S_c$ remaining positive under these operations correspond to activated logic units.

Following these Phase (1) operations, a single search for positive counts $S_c$ sets the detector plane to the match state at each word corresponding to an activated logic unit. Contents of this segment of the detector plane are transferred to the data register for use as search criteria in Phase (2).

*Phase (2).* This phase generates the output vector $r^i, \ldots, r^i_{Nr}$ corresponding to the $i$th (input) pattern. The components

$$r_n = Sgn \left[ \sum_{k=0}^{K} w_{kn}\ b^i_k \right]$$

are to be generated within the associative processor, using $b^i_k$ as an input from Phase (1) and the weights $w_{kn}$ stored in the weight section of the associative memory, Fig. 7. Arithmetic operations, similar to those discussed in the second section of this paper, are then used to yield the response vector $(r^i_1, \ldots, r^i_{N_r})$.

*Phase (3).* The target vectors stored in the associative memory (Fig. 7) are vectors with binary valued components $(r^i_1, \ldots, r^i_{N_r}) = \tau^m$, where $\tau^m$ represents the known classification of the $i$th pattern. It is assumed that the $K \times N_r$ matrix of weight [$W$] stored in the associative memory have been predetermined such that each of the $M$ patterns of interest is correctly classified, i.e., the output of Phase (2) matches the appropriate target vector, component for component.

Components $(r^i_1, \ldots r^i_{N_r})$ of target vectors $\tau^m$ are stored in a portion of the associative array at locations derived from "names" $d_m (i \leq m \leq M)$ of patterns associated with the target vectors (Fig. 7). A response vector generated during Phase (2) is used as a search key to interrogate the target vector portion of the associative array. The location of any responding word denotes the name of the pattern, as in the following table, showing the target vector portion of associative memory.

| Target | Vector | Name | Word Address |
|---|---|---|---|
| $r^1_1$ | $r^1_{N_r}$ | $d_1$ | $f(d_1)$ |
| $r^2_1$ | $r^2_{N_r}$ | $d_2$ | $f(d_2)$ |
| $r^M_1$ | $r^M_{N_r}$ | $d_M$ | $f(d_M)$ |

*Processing Times*

A program was written for the described pattern-recognition model using the instruction set presented in the second section of this paper.

The pattern recognition program has an adaptive or learning mode, requiring 120 instructions, in which weights are adjusted to properly classify a set of input patterns. The program for each mode is invarient to pattern parameters used for classification. Since 82 instructions are common to the two modes, only 131 instructions need be stored in program memory.

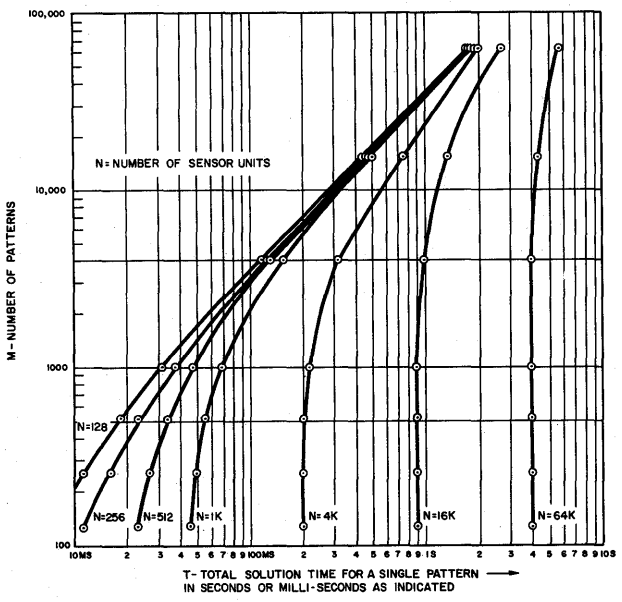



Figure 9. Time for associative classification of a single pattern as a function of the number of patterns and sensors for the $W/C$ data organization.

*This cycle time is based on a magnetic film realization of the associative array described in reference 12.

Based on the aforementioned recognition program, a word-per-cell data organization, and an associative command cycle of 0.8 microseconds, the graph of timing efficiency shown in Fig. 9 was constructed. Note that "$N$" represents the number of sensor units at the input and "$M$" the number of patterns distinguishable by the processor. The "MARK I Perceptron" used 400 sensors in a $20 \times 20$ array and 512 logic units.

It can be seen that the APP could solve this problem in approximately 3 milliseconds using some 2000 words of associative storage. The APP realization offers significantly greater ease of alteration and somewhat lower cost at a moderate increase in processing speed relative to the Mark I.

## CONCLUSION

The associative parallel processor, described in this paper, achieves considerable generality with simple word and bit logic through the use of the sequential-state-transformation mode of associative processing. Its range of applicability is increased by novel arithmetic algorithms allowing simultaneous processing of many operands in bit parallel fashion. These algorithms allow efficient use of the processor for problems in which only a fraction of the stored operands are processed by a given command. Earlier processors[4,5,6] were efficient only when nearly nearly all operands were processed by each command.

An important feature of the parallel processor, when used as a pattern recognition device, is the ability to modify its functional structure, through alteration of memory contents, without change in its periodic physical structure. This adaptive feature has importance in applications where patterns change with time, or where the processor is used as a prototype of subsequent machines having fixed recognition capabilities. Further research is required to fully exploit this adaptive capability.

Linear threshold pattern classifiers of the type here presented are beginning to find many applications. To date, these types of pattern classifiers have been studied and/or implemented for character recognition, photointerpretation, weather forecasting by cloud pattern recognition, speech recognition, adaptive control systems and more recently, for medical diagnosis from cardiographic data. Other possible applications include terminal guidance for missiues and space vehicles and bomb damage assessment.

Currently the processor is being studied for application to the tasks of job shop scheduling, optimum commodity routing and processing electromagnetic intelligence (ELINT) data. In each instance signfiicant speed gains have been shown possible over conventional sequential digital computers. It is interesting to note that the processor described in the second section of this paper may be applied to this variety of tasks without significant changes in organization or command structure.

## ACKNOWLEDGMENTS

The authors take pleasure in acknowledging contributions to this effort by Mr. J. Medick and Dr. J. C. Tu. We are most appreciative of support given this work by Messrs M. Knapp and A. Barnum of the Rowe Air Development Center.

## REFERENCES

1. P. M. Davies, "A Superconductive Associative Memory," *Proc. of the Spring Joint Computer Conference,* May 1962.

2. J. E. McAteer, J. A. Capobianco and R. L. Koppel, "Associative Memory System Implementation and Characteristics," ibid., Nov. 1964.

3. G. Estrin and R. H. Fuller, "Algorithms for Content-Addressable Memory Organizations," *Proc. Pacific Computer Conference,* Mar. 1963, pp. 118-130.

4. ——— and ———, "Some Applications for Content Addressable Memories," *Proc. Fall Joint Computer Conference,* Nov. 1963, pp. 495-508.

5. P. M. Davies, "Design of an Associative Computer," *Proc. Pacific Computer Conference,* Mar. 1963, pp. 109-117.

6. R. G. Ewing and P. M. Davies, "An Associative Processor," *Proc. Fall Joint Computer Conference,* 1964, pp. 147-158.

7. B. A. Hane and J. A. Githens, "Bulk Processing in Disturbed Logic Memory," *IEEE Trans. on Elect. Comp.,* vol. EC-14, no. 2, pp. 186-195 (Apr. 1965).

8. R. Widrow et al, "Practical Applications for Adaptive Data Processing Systems," 1963 WESCON (Aug. 1963).

9. J. C. Hay, F. C. Martin and C. W. Wight-

man, "The MARK I Perceptron-Design and Performance," *IRE International Convention Record,* 1960 (Part 2).

10. C. H. May, "Adaptive Threshold Logic," Rept. SEL 63-027 (TR 1557-1) Stanford Electronics Lab, Stanford, Calif. (Apr. 1963).

11. G. Nagy, "System and Circuit Designs for the Tobermory Perceptron," OTS, AD 604 459, Sept. 1963.

12. R. H. Fuller, J. C. Tu and R. M. Bird, "A Plated Wire Associative Memory," NAECON Conference, Dayton, Ohio, May 10-12, 1965.

# COMPUTER ORGANIZATION FOR ARRAY PROCESSING*

D. N. Senzig and R. V. Smith

*IBM Watson Research Center*
*Yorktown Heights, New York*

## INTRODUCTION

In spite of recent advances in computer speeds, there are still problems which make even greater demands on computer capabilities. One such problem is that of global weather prediction. Here a three-dimensional grid covering the entire world must be stepped along through relatively short periods of simulated time to produce a forecast in a reasonable amount of real time. This type of problem with its demand for increased speed in processing large arrays of data illustrates the applicability of a computer designed specifically for array processing.

When arrays of data are being handled, it is usual to have to do the same calculations on each piece of data. This kind of problem is suited to a machine with multiple arithmetic units (AU's) since each can be carrying on the same task on different parts of the array. We are fast approaching the physical limit in speed for computer AU's. On the other hand, a number of AU's can operate simultaneously to increase the amount of work done per unit time. The speed and number of these units can be selected to suit the economics of the case and the logical characteristics of the problem. When multiple AU's are all doing the same task, a single control unit suffices. For example, one load instruction can cause all AU's to load their separate accumulators each from a different part of the array. Facility must be provided to inhibit some of the AU's when exceptional conditions are being handled by the others, or when the number of pieces of data to be processed is smaller than the total number of AU's available. A suitable paralleling of separate memory units must also be provided to yield data at the rate required by the AU's.

The cost and speed of an array processing computer depends on the speed of the memories and the circuitry used, and also on the number of AU's provided. Speed can be characterized by the maximum rate at which bits can be brought from the memories and processed. Studies to date have indicated that higher bit rates at proportionately lower costs are possible with given types of hardware by using the array processing approach rather than the conventional types of organization.

## VAMP ORGANIZATIONAL CONCEPTS

The VAMP (Vector Arithmetic Multi-Processor) computer will be described independent of

*This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

technology in order to stress organizational concepts apart from arithmetic execution times. However, the ideas described below were tested in a simulator where details such as instruction format, word length and number of AU's are fixed.

The VAMP computer consists of three major units: the Mill, the Memory and the Control.

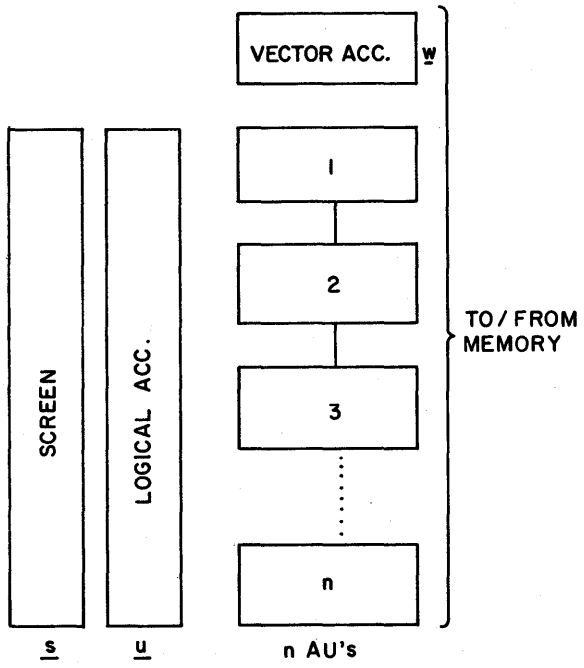



Figure 1. Vamp mill.

*The Mill*

As shown in Fig. 1, functionally the Mill consists of a linear array of $n$ arithmetic units (AU's) and the registers w, u, and s. (A typical value for $n$ is 16, but this could vary to suit requirements.)

Within each AU, processing is done in parallel on all bits in the registers. The number of bits per word, $k$, is immaterial to this discussion, but a typical number is 36. The registers of the $i$th AU are illustrated by the $X^i$ and $Z^i$ registers in Fig. 2. (We will use the symbol $X^i$ to represent the $i$th register of the register array X. $X^i_j$ will represent the $i$th bit position of the register $X^i$.) The $i$th accumulator, $X^i$, $2k$-bits long, receives the result of most arithmetic operations. The $k$-bit storage register, $Z^i$, is not program addressable and serves mainly as a buffer between memory and $X^i$. The $Z^1$ register also serves as a buffer between the u, s, and w registers and memory. An instruction for loading X normally fetches $n$ words simultaneously from the memory into the Z registers. The words are then transferred into the X register array.

In arithmetic instructions using operands from memory, the instruction results in $n$ operands being simultaneously fetched into Z. For the floating point add instruction, the $n$ numbers in Z and X are then added and the $n$ double length sums are placed in X. For multiply, the numbers placed in Z are the multiplicands, the $n$ multipliers are in the most significant half of X as a result of a previous operation. Double length products are placed in X.

The vector instructions (those in which multiple operations are specified) include the conventional Boolean, fixed point, and floating point operations performed $n$ at a time.

In addition to the vector instructions, the conventional register-register and memory-register Boolean, fixed point, and floating point instructions that one would find in a conventional multiregister computer are included in VAMP.

The registers s (screen) and u (logical accumulator) contain $n$ bits each. Looking again at the registers $X^i$ and $Z^i$ of $AU^i$, one bit of s and one bit of u correspond to each AU. In particular, $s_i$ and $u_i$ go with $X^i$ and $Z^i$.

The screen bit $s_i$ serves to inhibit the operation of its corresponding AU. The Mill is designed to, and generally will, simultaneously execute a common instruction in all AU's. However, the screen control is provided to give each AU the capability of not executing a given instruction. For example, if the given instruction specifies addition, those AU's whose screen bit is one perform the addition; those whose scbreen bit is zero do not.

The logical accumulator, u, serves to hold the results of logical operations and acts as the control vector in certain vector operations on X. Tests on the relationship ($=, \neq, <, \leq, >, \geq$) of members of X to zero set their logical result into u where individual elements can be set or tested. An interchange of the contents of s and u allows such results to become the screen.

The effective use of structured operands (vectors and matrices), depends upon the ability to extract and reinsert certain elements or groups of elements. This restructuring depends on selection of elements, which is a binary operation (i.e., to select or not select), and is conveniently specified by the bit vector u. Two operations for restructuring data arrays are included—compress and expand. The following discussion of the operands is based on Iversons's description.[1]

The compress operation defined on an arbitrary vector a and a compatible (of equal dimension) bit vector u is denoted by $c \leftarrow u/a$ and is defined as follows: the vector c is obtained from a by suppressing from a each component $a_i$ for which $u_i = 0$. Clearly, the dimension of c is equal to the sum of the ones in u. For example, if u = (1, 0, 0, 0, 1, 1,) and a = (1, 2, 3, 4, 5, 6), then u/a = (1, 5, 6). For the VAMP instruction, the elements of a are the numbers stored in the array X. Hence the dimensions of the operand vectors is assumed to be $n$. The result vector, c, is stored in X. Denoting the sum of components (ones) in u by $i$, then the first $i$, registers in X contain the vector c, the remaining $n - i$ are set to zero.

The expand operation is expressed as $c \leftarrow u/a$. The vectors c and u are of a common dimension. The expand operation is equivalent to choosing successive components of $c_{ij}$ from $a_i$ or 0 according as the successive components $u_i$ are 1 or 0. For example, if u = (1, 0, 0, 1, 1,) and X contains the numbers (1, 2, 3, 4, 5, 6), then the result is X containing the numbers (1, 0, 0, 0, 2, 3). Denoting the sum of components of u by $i$, the first $i$ elements originally in X will be preserved, and $n$–$i$ elements of the result are necessarily zero.

The use of compress and expand can be illustrated in a weather problem. The radiation calculation depends principally on the type of cloud and amount of cloud cover. For efficient handling, separate arrays for each of the various cloud characteristics should be created by compression. Then all arithmetic units can be put to work on one such array after the other, all doing the same operation at any one time. The separate arrays can then be reassembled into one by the expand and load under screen control operations for use in the next procedure. Essentially these instructions partially recover the advantage of making use of the average that a conventional computer enjoys.

The vector accumulator w is a $2k$-bit register. All members of X (subject to s) can be added or multiplied together and the sum or product placed in w ($w \leftarrow w + \Sigma_j X^i$; $w \leftarrow w \times \pi_i X^i$). Search instructions to find the maximum (or minimum) value in w and the registers of X place this maximum (minimum) value into w.

*Memory*

The design of VAMP requires simultaneous access to $n$ words in memory—one word for each AU (Fig. 3. Several memory organizations are possible and these are based on the physical arrangements chosen. The result will be a functional arrangement, that is, an arrangement as seen by the program.

One type of organization results when one physical word is used to feed more than one AU. One reason for doing this is that the physical word brought out at each memory access is large enough to hold several functional words and therefore it is more efficient to use all of them. If one physical word does not hold enough functional words to supply all AU's, then several boxes can be accessed simultaneously. The simplest approach is to group boxes together and bring out the same physical word from each in the group to feed all AU's. The SOLOMON[2] case is functionally similar to the one above but, being serial, one bit of each physical word in the group is sent to each AU and a succession of groups supply the succession of bits which make up the functional word. In this case, the data received by the AU's have a fixed spatial relationship to each other. Functionally it is as if each SOLOMON AU had its own memory and accessed the same word from it at any one time as all the others did from theirs. To ease this restriction, provision may be made for units to access words in their neighbor's memories, proceeding in step as before. This results in the concept of a matrix of processing elements (arithmetic units, each with a memory) as in SOLOMON.

For an $n$ arithmetic unit VAMP, $2^j$ separate memory boxes are required ($2^j \geq n$). Assuming a total of $2^m$ words of memory, the addressing is set up so that the least significant $j$ bits of the $m$ bit address select the memory box. The most significant $m - j$ bits of the address then specify the particular word in the box.

VAMP addresses memory in two modes. In the vector direct mode, the instruction specifies an initial

address $a_0$ and an increment $d$. The $n$ addresses $a_0$, $a_0 + d$, $a_0 + 2d, \ldots, a_0 + (n-1)\ d$ are generated. Since the number of memory boxes is a power of 2 and at least as great as $n$, an odd value of $d$ will result in all addresses being in different boxes. The most usual cases arise from proceeding along a vector or a column of matrix so that $d = 1$, or along a row of matrix so that $d$ equals the length of a column. If the columns are of even length, an additional dummy members can be added to achieve an odd number. Should the program require the use of an even value of $d$, it will still be accepted, but with consequent loss of time. A test is made to determine if $d = 0$. If so, the same word is transferred to all AU's without attempting to access the memory more than once.

Addressing in the vector indirect mode uses a set of $n$ addresses in memory. These addresses are fetched into Z, but they are then fed to the memory address registers to bring out $n$ words. Any set of addresses may be used so that any desired set of words, possibly including repeats, will be fetched or stored. The vector indirect mode will generally lead to multiple requests from the same memory box. This will result in less than maximum speed; however, it will be faster statistically than executing $n$ separate instructions, one for each word to be fetched or stored.

The SOLOMON scheme has certain physical advantages. It does not require as many memory boxes to avoid memory access conflicts (but to obtain a suitable memory size, it may need just as many). The number of bits to access the AU memory is less than it would be if the entire physical memory were adressable.

The VAMP scheme requires hardware and time to produce the address vector. Functionally, the bus between memory and AU's is an electronic cross bar switch to gate the $n$ addresses to the memory units, and an electronic cross bar switch to gate the $n$ data words between the memories and the $n$ AU's.

Simulation showed that the memory bus hardware for VAMP can be significantly reduced by time-sharing a small number of transfer lines (typically 2 or 4) in combination with a simple anticipatory control. This would allow decoding and address generation to be overlapped with the execution of the previous instruction. Thus, when considering the entire hardware inventory of a machine of this type, the increase due to the incorporation of the flexible memory of VAMP should be small. Also, this is a one-time cost which will be offset by savings in programming which is continuing activity.

The VAMP memory organization results in a common functional memory with no relation between a given AU and a section of memory. The SOLOMON organization yields individual functional memories and fixed addressing. The former has greater functional advantage than the latter since AU's have access to all of memory, not just to the part of it which is their own and their neighbors'.

The flexibility of its common functional memory means that VAMP will adapt easily to automatic programming. Computation can proceed in essentially the same way as has been conventionally done, but $n$ items will be handled at a time. With SOLOMON-type addressing there will be severe problems brought about by segmentation of the memory into individual AU memories of limited size and the consequent segmentation of the problem data. Boundary problems between the segments and storage allocation, particularly dynamic storage allocation, will be prominent difficulties.

Table look-up (where each AU may require a different value from the table) is rather easily done with VAMP through use of the vector indirect mode. No scheme for doing this seems to be available with SOLOMON-type addressing and indeed, indirect addressing appears to be incompatible with this addressing scheme. At Livermore a problem known as Coronet IV uses approximately 50 percent of the total running time of their STRETCH computer, and 36 percent of the Coronet IV time is used for table look-up and interpolation of table values.

### *Control Unit*

The control unit in VAMP includes the instruction fetch controls, instruction decoding index unit and address unit. As in conventional computers, instructions are stored in the same memory as (and are indistinguishable from) data. Instruction decoding and execution differs from the 7090, for example, only in that for the vector instructions, micro operations are issued to $n$ AU's rather than to 1, and on a vector memory access an index register is specified as containing an increment from which the address unit has to generate $n$ addresses.

## FURTHER DISCUSSION OF THE VAMP AU's

One of the reasons for building faster computers is to solve bigger problems. Bigger problems usually mean longer chains of computation and, for a given word length, more round-off error and scaling difficulties. The uncertainty of magnitudes in long calculations has led to the almost universal use of floating point arithmetic in modern large-scale machines designed for scientific problems. The classical arguments for floating point in a conventional machine would seem to hold equally in array processors.

If the Mill were designed as it functionally appears, that is, as a collection of "classical" AU's of the type shown in Fig. 2, we would have a require-

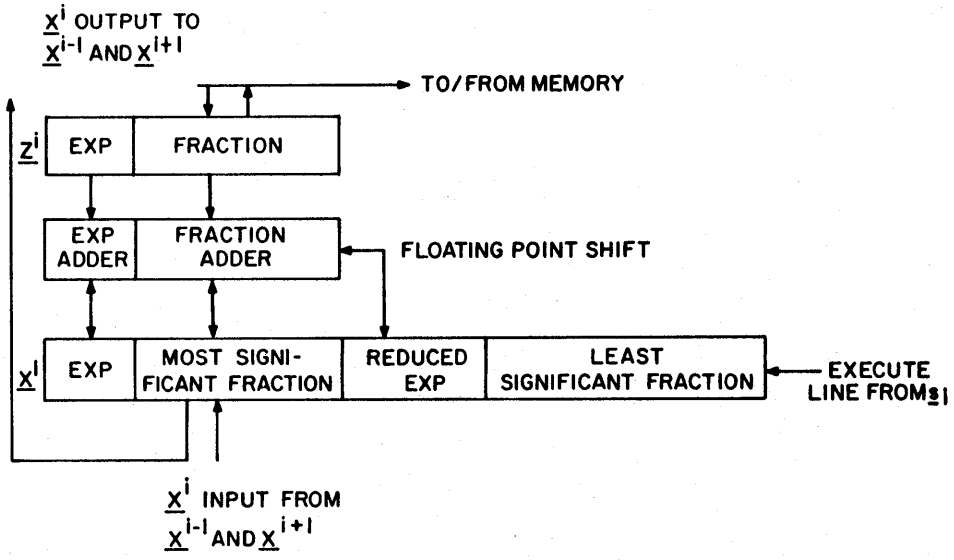



Figure 2. Vamp arithmetic unit functional data flow.

ment that does not exist in conventional computers. Namely, all arithmetic operations should run in step and the time required should be data independent. It is permissible, for example, for the execution of a floating point add on the IBM type 7044 to vary between 4 and 23 cycles with an average execution time of 5.5 cycles. In VAMP this is not suitable. Statistically, with many AU's the execution time for all the reach completion will be close to the maximum and a data dependent execution time would require extensive local control on each AU that we prefer to do away with. Floating addition (and floating subtraction), multiplication, and division are the operations that normally have data dependent execution times.

In the case of multiplication and division, algorithms that give a data independent execution time are well known. Multiplication by two bits per iteration and division by the nonrestoring algorithm reduce the local control for multiply and divide to merely picking the correct multiple (-1, 0,1,2) of the multiplicand or divisor. The other control hardware, shift counter, etc., is common to all units.

In the case of floating point addition, a fast algorithm with execution time independent of data can be developed by providing multiple shifts in the alignment and normalization phases. In particular, digit shifts of powers of two (1, 2, 4, 8 digits, etc., are provided. For alignment, assume the exponent difference has been obtained and is represented in binary positional notation, the bits having weight 1, 2, 4, 8, etc. Then by having central control issue the micro operation "shift," the local control need merely be sufficiently sophisticated to gate the

fraction through the shifter or to bypass it, depending on whether the corresponding bit of the exponent difference is one or zero.

For normalization, each AU must be provided with control logic to determine if a one would be shifted out of the most significant end of the accumulator by the shift micro operation about to be given. Again the local control need only be sufficiently sophisticated to execute or ignore the common shift micro operation, based on the value of the most significant bit.

Unfortunately, the schemes for floating point add get to be very expensive when applied to each of $n$ AU's. Some compromise between multiple shift gates and completion controls would undoubtedly be made. For instance, the central control could re-

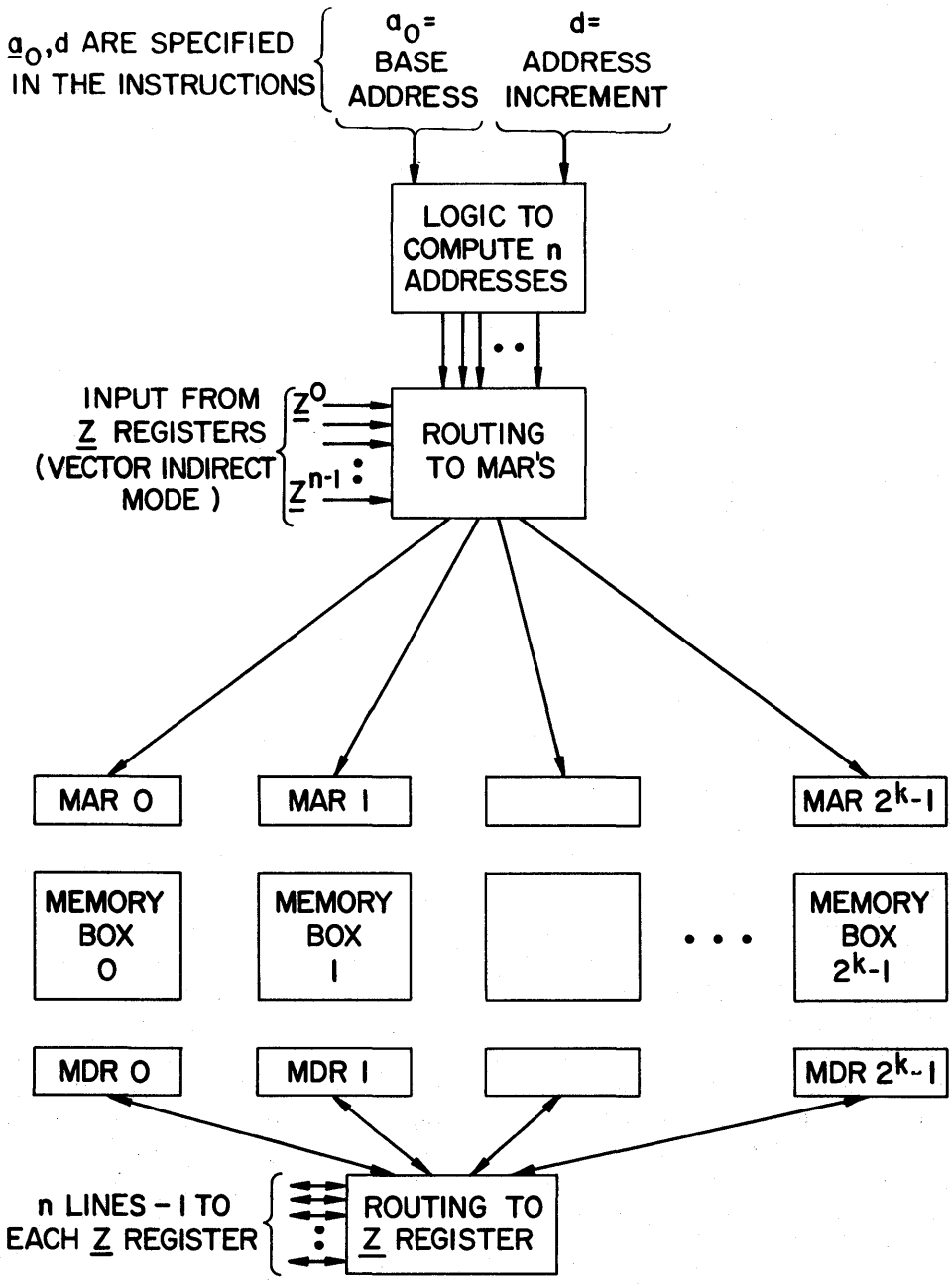



Figure 3. Vamp memory organization.

peatedly issue micro operations for a single shift, stopping when all AU's signal completion. Another reasonable alternative is to use a larger radix. Quaternary, octal or hexidecimal systems are obvious candidates.

Assuming the engineering problems are solved one is still left with the situation where a scalar (one pair of operands) operation requires as much time as a vector operation ( $n$ pairs of operands). Thus when a problem is heavily dominated by scalar operations there is little gain in speed.

Rather than attempt to work with multiple AU's, a register array and a much smaller number of very high-speed Execution Units may be used. The basic procedure is to load the register array and stream operands and results to and from high-speed Execution Units. Figure 4 shows the resultant organization of the Mill.

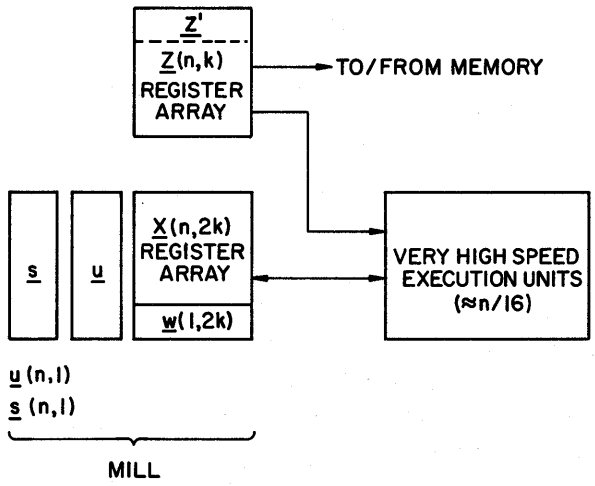



Figure 4. Vamp mill ($n$ = number of functional AU's;$k$ = number of bits per word).

The algorithms we use to obtain high-speed arithmetic operation are almost completely combinatoral circuits. Floating add (and its variants) use the alignment and normalization combinatorial shifter described above.

The multiplier is based on the well-known carry save multiplier as recently extended by Wallace[3] to process all partial products simultaneously. Wallace's proposal is to interpret the contents of the multiplier register as radix 4 digits and recode these digits into the digits - 2, - 1, 0, 1, 2. The now easy to generate partial products are grouped by threes and gated into a tree of carry save adders. The outputs of each level of carry save adders are again grouped by threes and gated into the next level. Each level of carry save adders reduces the number of summands by about a factor of 1.5. The two outputs of the tree are added to produce the double length product.

The divide algorithm is an iterative method based on that used in the Harvard Marck IV and discussed by Richards.[4] The technique essentially involves generating the divisor inverse by a series of truncated multiplications.

During a vector floating point multiply we could have four sets of operands in motion at once. One set is being accessed from the register array. One set is being multiplied. A product is being normalized. A normalized product is being stored in the register array.

In an effort to keep all operations times in the stream to about the same duration the multiply tree can be split such that some fraction of the multiplier bits are processed simultaneously in each section. While this may slightly increase the time to complete a single multiply, the cost will be slightly decreased and overall speed will be significantly increased.

As with multiply, a snapshot of Vector Floating Add would reveal many sets of operands in motion at once. Here we have the following possibilities: Fetch operands from registers, determine the exponent difference, alignment shift, addition or subtraction, normalization shift, and store result in register.

To a good first approximation, for floating point fraction lengths of about 32 bits these techniques give approximately 16 times the speed and cost 16

times as much as the classical "parallel" algorithms that use a parallel adder but do serial shifting and process multiplier and quotient digits serially. Hence, for given speed and cost to implement the vector operations we obtain improved speed for operations involving less than $n$ operands at a time. The discussion of $n$ arithmetic units will continue but it should be kept in mind that we propose to use $n$ registers and a much smaller number of Execution Units. The exact number and form of the Execution Units would be determined by word length and the particular circuit characteristics.

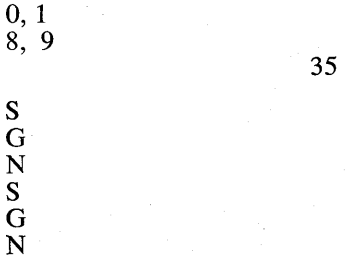
## VAMP COMPUTER

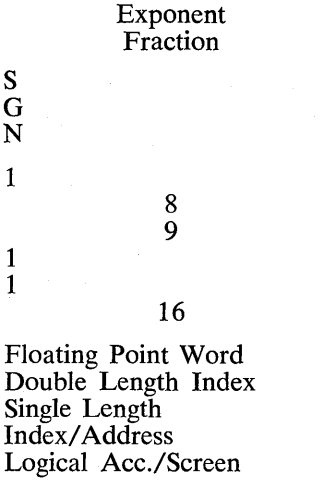
This section describes VAMP with word length, number of arithmetic units, number of memory units, and instruction set fixed. A simulator to investigate the organizational concepts described above was written to run on the 7094 for the particular organization described in this section. Since our study was to investigate concepts, not circuit and memory speeds, we will not supply such things as multiply times and, hence, performance improvement factors over convenient targets.

The simulator was not complete in the sense that interrupts were not programmed and I/O must be done outside the simulator program. The simulator will accept a program written in VAMP symbolic assembly language, assemble, and execute the program.

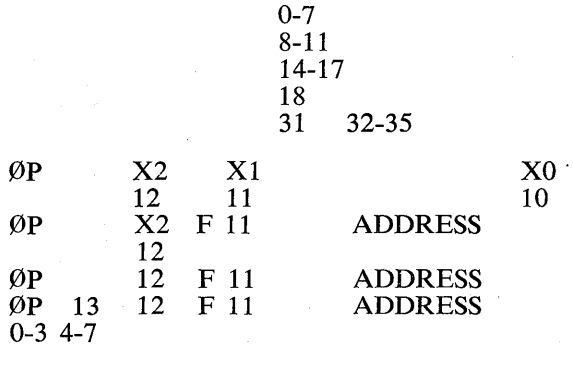
The simulated VAMP assumes 16 functional arithmetic units and 16 memory boxes of 16,384 words each. The word length is 36 bits. The floating poit number representation is the same as the IBM 7094: the floating point fraction is binary, sign and magnitude, the exponent is excess 128. (-1.0, 0, 1.0 are represented in octal by 601 400 000 000, 000 000 000 000, and 201 400 000 000 respectively). The data and instruction formats are given below.

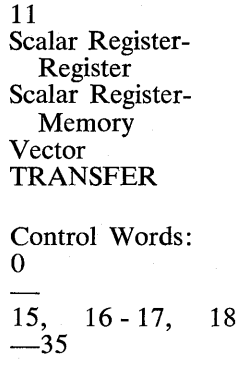
Data Format:



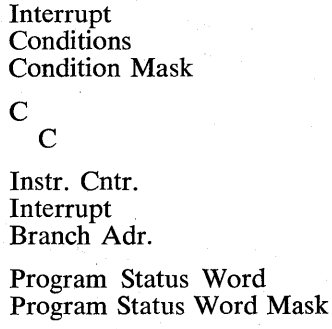

Instruction Formats:




Control Words:




(C C - condition code)

VAMP has 15 index registers. The I0, I1, I2 and I3 fields of instruction formats refer to one of these registers or, if the field is 0000, to an implicit register that contains an unmodifiable zero.

To keep all address calculations out of the Mill the index units contains a complete set of arithmetic (add, subtract, multiply, and divide) and Boolean (AND, OR, equivalence) operations. The number representation in the index unit is 2's complement. The word length is normally 18 bits with multiply producing a double length product and divide producing an 18 bit quotient and 18 bit remainder.

Memory Addresses are calculated from information specified in the F, I1, and Address fields. The bit combination in the field I1 selects the index register to be used in modifying the Address field. The instruction is then executed as if its address field contained the stated address *plus* the contents of the index register.

Address modification is extended to include base address indirect addressing. Base address indirect is specified by a one in bit 13 of the instruction (right right-most bit of the flag, F, field). An address is computed by adding the contents of the index register specified by I1 to the address part of the instruction to form a memory address. Bits 13–35 at this base indirect address replace bits 13–35 of the instruction register. The process then repeats—a new memory address is computed for I1 and the address field. Bit I3 is examined for another level of base indirect addressing. The address that comes out at the end of the chain of indirect addresses is called the effective base address.

Vector instructions, i.e., those that do 16 operations simultaneously, use the effective base address as the address of the first operand. The address of the second operand is determined by adding the contents of the index registers specified by field I2 to the effective base address. Letting $a_0$ represent the effective base address and $i_2$ the contents of the index register addressed by I2, the address vector, a, is of dimension 16 and the components are $(a_0, a_0 + i_2, \ldots, a_0 + 15\ i_2)$. All values $0 \leq i_3 < 2^{18}$ are valid.

There is another form of indirect addressing known as vector indirect addressing. In this mode the address vector is used, not to address the operands directly, but to address an address vector. This mode is indicated by a 1 in bit 12 of the vector instruction format. Vector indirect addressing does not proceed beyond 1 level; i.e., the address vector fetched from memory is used as the operand address vector without further modification. (When modification of the address vector is required it can be fetched into the accumulator X and treated as data.)

In scalar floating point or scalar boolean operations the 4-bit fields X0, X1, and X2 refer to one of the 16 double length accumulators. The contents of the registers X0 and X1 serves as operands and X2 specified the register to receive the result. For scalar register-memory instructions the contents of the memory location specified by the effective base address and the contents of the X register specified by the X2 field serve as operands. The result is placed in the X2 register.

To facilitate programming of loops, where one is processing 16 elements at a time, three loop closing instructions, VTILE (vector transfer on index low or equal), VTIH (vector transfer on index high), and VCTR (vector transfer on counter) are provided. These instructions combine stapping an index, testing the index, and conditional branching. They are made more powerful by having them set to the "do not execute" state the screen bit of arithmetic units which will not participate on the last iteration when there are less than 16 items to process.

Looking at the instruction VTILE in detail, the first step is to add 16 times the contents of index I2 to index I3 and put the sum in index I3. (The multiply and add is done by simply shifting index I2 four bits left and adding Mod $2^{18}$.) Then, the new value of index I3 is compared against the contents of index (I3 + 1) Mod 16.

If the contents of index (I3 + 1) Mod 16 are greater than the contents of I3 a branch is not taken —the instruction counter is to be advanced by 1. If the content of index I3 + 1 is less than or equal ot the content of index I3 the branch is taken, that is, the effective address is placed in the instruction counter. The results of the test ($>$ or $\leq$) are also noted for use in the following part of the VTILE instruction.

The contents of index I2 are now added to I3 and the sum Mod $2^{18}$ is compared against the contents of index (I3 + 1) Mod 16. If the result of the comparison is the same as that on which the branch decision was made (i.e., both greater than or both less than or equal) the screen bit 2 is not modified. If

the results of the two comparisons are different, the screen bit 2 is set to zero (do not execute state).

The contents of index I2 are now added to the above sum. If this sum Mod $2^{18}$ bears the same relation to index (I3 + 1) Mod 16 as did the addition on which the branch decision was made, screen bit 3 is not modified. If the relation changes the screen bit is set to zero. This process repeats until 15 additions have been done.

Note that screen bit 1 is never modified, screen bit 2 may be modified at the end of the first addition, screen bit 15 may be modified at the end of the 15th addition.

The VTIH instruction differs from VTILE only in that the branch decision is reversed. If, after the first add shift operation, the contents of index (I3 + 1) Mod 16 are greater than index I3 we branch, i.e., put the effective address in the instruction counter; if the contents of index (I3 + 1) Mod 16 are less than or equal to index I3 the instruction counter is advanced by 1. All other operations are the same in VTILE and VTIH. The instruction VTCR uses an implicit -1 as the increment and the contents of the register I2 as the initial value. The comparison is against an implicit zero. Other than this the instruction is identical to VTIH.

The 15 additions required by VTILE, VTIH and VTCR are performed by the same unit and in the same way as addresses for a vector instruction are generated.

The instruction set for VAMP has been designed for the processing of vectors in memory, including rows and columns of matrices. These will normally have considerably more components than the number of AU's. Many operations such as compress, search for largest, and sum and product reduction (sum or product of all compnents) must operate

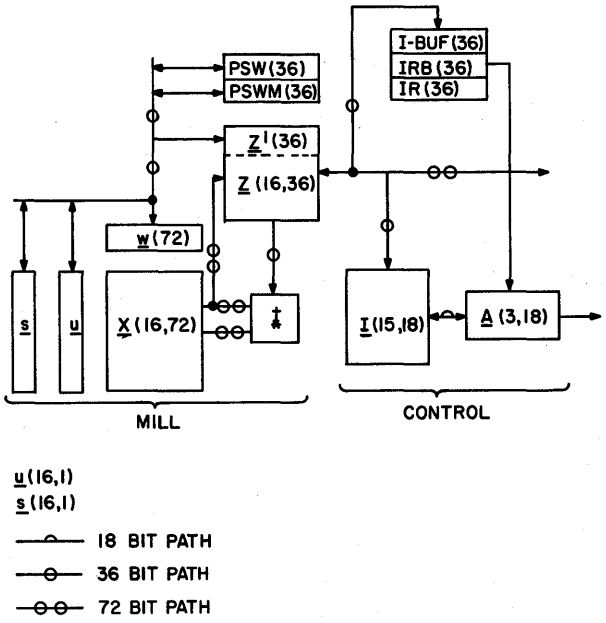



Figure 5. The simulated vamp CPU.

over the entire vector even though only 16 are handled at any one time. The instruction set is designed around this concept.

The simulated CPU is shown in Fig. 5. The w, s, u, X and Z arrays perform the functions described in the second section of this paper. The address unit A contains three 18-bit registers and two 18-bit adders. Like Z, A is not seen by the programmer. It is used by the control for index and address arithmetic.

The right-most 18 bits of the program status word, PSW, contain the instruction counter. Bits 16 and 17 of the PSW word contain the condition code. The results of all index operations as well as scaler operations in the Mill are used to set the condition code to indicate whether the result of the last operation was zero, less than zero, greater than zero, or overflowed. An instruction to test the condition code and branch accordingly is provided.

Bits 0-15 of the PSW and PSWM were not de-

fined in the simulation. They are reserved for interrupt indicators, interrupt masks, and the interrupt branch address (the location where a new PSW and PSWM are to be picked up from and where the current ones are to be placed).

The registers I-BUF and IRB in Fig. 5 are used by a relatively simple anticipatory control. Instructions are executed in three levels. At level 1 the instructor is fetched from memory and placed in register I-BUF (instruction buffer). In level 2 the instruction op code is scanned to determine if it is a vector arithmetic insturction or one of the vector transfer instructions: VTILE, VTIH or VTCR. If it is one of the vector instructions the necessary addresses are generated. If it is not a vector instruction, no operation is performed. In step 3 the instruction is executed. For most instructions steps 1 and 2 can easily be overlapped with the execution of the previous instruction. Note however that in no case are the contents of registers visible to the programmer modified until the previous instruction has been completed. Thus if an interrupt occurs at the end of the current instruction some unnecessary work has been done but no procedure for recovery of previous register contents need be included.

## PROGRAMMING EXAMPLE

The following small FORTRAN problem is coded in the VAMP symbolic assembly language:

```
    DO 1 I = 2, 59
    L = A2(I)
1   A1(I) = C*A2(I) + A2(I - 1) + A2(I + 1)
      + B(L)
```

A1 and A2 are one-dimensional arrays of 60 elements. B is a one dimensional array of 40 elements which enter by indirect addressing. C is a constant.

The following instructions are used (definitions given above).

LDA A, I1, F, 12 Load Address
: Places the effective address into the index register addressed by the field I2.

GPS A, I1, F, 12 Generate Prefix in s.
: Set the first a bits of the screen to one where a is the effective address.

VLDX A, I1, F, 12 Vector Load X
: Load the contents of the memory locations specified by the address vector into the most significant half of X (subject to s)

VAND A, I1, F, 12 Vector AND
: AND each bit of X with the corresponding bit in the memory locations specified by the address vector (subject to s).

VLXI Vector Load X Indirect
: Replace the most significant half of X by the contents of the memory location specified by bits 18-35 of X. Note that in this version of indirect addressing the address vector is assummed to be in X as the result of a previous operation.

VSTX A, I1, F, 12 Vector Store X
: Store the contents of the 16 accumulators (subject to s) in the memory locations specified by the address vector.

VFAD A, I1, F, 12 Vector Floating Add
: Algebraically add the floating point numbers specified by the address vector to the floating point numbers contained in X (subject to s). The sums are normalized.

VUFA A, I1, F, 12 Vector Unnormalized Floating Add
: Same as VFAD except the sums are not normalized.

VFMP A, I1, F, 12 Vector Floating Multiply
: Multiply the floating point numbers at the memory locations specified by the address vector by the floating point numbers stored in the most significant half of X register (subject to s). The double length product appears in X.

VTILE A, I1, F, 12, 13 Vector on Index or Low or Equal
: See description in section VAMP Computer.

BSS A Block Started by Symbol
: BSS is a psuedo-operation that reserves a block of A consecutive storage locations.

OCT Octal Data
: OCT is a psuedo-operation that introduces binary data expressed in octal form into the program.

The VAMP assembly language program is shown in Fig. 6.

## CONCLUSION

The concept of an array processing computer is due to SOLOMON. By taking advantage of the features inherent in interleaved memories, very high-speed arithmetic units, multiple register CPU's and by adding a number of special instructions one ob-

tains a machine that has the functional capabilities of SOLOMON but which fits within the framework of a more conventional computer organization. Further, the ideas presented here should result in a machine which is applicable to a much wider range of problems than SOLOMON.

There certainly exists a large class of problems for which neither VAMP nor SOLOMON would show any appreciable advantage over a more conventional organization. Compiling is probably the best example of these.

## REFERENCES

1. K. E. Iverson, *A Programming Language,* Wiley, 1962.

2. J. Gregory and R. McReynolds, "The Solomon Computer," *PGEC,* vol. EC-12, no. 5, pp. 774-781 (Dec. 1963).

3. C. S. Wallace, "A Suggestion for a Fast Multiplier," *PGEC,* vol. EC-13, no. 1, pp. 14-17 (Feb. 1964).

4. R. K. Richards *Arithmetic Operations in Digital Computers,* Van Nostrand, 1955, pp. 279-282.

```
LOCATION    OP      ADDRESS, I1, F, I2, I3

            LDA      1,,,1          SET IR 1 =  1
            LDA      1,,,2          SET IR 2 =  1
            LDA     58,,,3          SET IR 3 = 58

            GPS     16              SET SCREEN TO
BEGIN       VLDX    A2,2,,1         LOAD A2(I), A2(I+
            VUFA    FXZR            PLACE BINARY P
            VAND    MASK            REMOVE EXPONEN
*  A2 CONVERTED TO TRUNCATED INTEGER
            VUFA    LOCB            ADD LOC OF B(1) TO
            VLXI                    LOAD B(A2(I), B(A2(I
            VFAD    A2+1,2,,1       ADD A2(I+1), A2(I+2), ...
            VFAD    A2-1,2,,1       ADD A2(I-1), A2(I), ...,
            VSTX    TEMP,,,1        STORE TEMP. RESULT
            VLDX    A2,2,,1         LOAD A2(I), A2(I+1), ..., A
            VFMP    C               MPY BY C
            VFAD    TEMP,,,1        ADD TEMP. VECTOR
            VSTX    A1,2,,1         STORE A1(I), ..., A1(I+15)
            VTILE   BEGIN,,,1,2     STEP INDEX CNTR 16
*  END TEST PROGRAM
*  DATA STORAGE
C           BSS      1
A1          BSS     60
A2          BSS     60
B           BSS     40
TEMP        BSS     16
LOCB        VFMP    B
FXZR        OCT     233000000000
MASK        OCT     400777777777

            END
```

Figure 6. A VAMP assembly language program.

# MANAGEMENT PROBLEMS OF AN AEROSPACE COMPUTER CENTER

G. A. Garrett
*Lockheed Missiles and Space Company*
*Sunnyvale, California*

At this session it seems to me that you might be interested in several of the more-or-less technical facets of the direction of a large aerospace computer installation. Consequently I will avoid competing with our environment by discussing the ubiquitous problems of recruiting, of personnel motivation, of obtaining cooperation among the members of the various computing groups, or even the basic problems inherent in convincing our computer folks that the whole computer center does not exist for them at all, but rather as a service for the other parts of our company.

Instead, I want to tell you today about a few of the figures we have on the actual costs of "Change"; then go into a few aspects of the "Turn-Around-Problem" from the management point of view; and finish with a few remarks on what a computer center such as ours may reasonably expect in the future.

While there are many fields in which constant change is the order of the day, the operation of virtually any modern computer center is faced with adjustment to changing computers, changing computer languages, and changing software systems with a frequency which is quite notable. In a recent attempt to analyze the economic effects of such changes, several interesting relationships have been noted.

In the past, the speed with which a newly installed computer has been "loaded" has often been of interest, but few figures have appeared which treat as a dynamic quantity the relationship between the program checkout load and the production load. Such relationships must be known, however, before one can evaluate the effects of change, since the purely static "before and after" pictures tend to conceal many of the significant points.

In analyzing the dynamics of the situation, some simple relationships have been postulated, and their predictions compared with those historical data which were obtainable at the LMSC computation center.

First, it was assumed that the loading on a computer could be divided between check-out and production, and that the ratio of these two would vary with time from installation. Since the proportion of computer programs which run for production without previous check-out is vanishingly small, it would seem reasonable that the load on a newly installed computer initially must consist solely of program development work. From such a starting point it follows that the ratio of production to development will show a continuous increase until it either levels off with time, or the pattern becomes confused by changes in language, operating system, or type of work processed by the center.

Both the ratio of production to development at steady state and the rate at which this ratio approaches steady state must be determined in order to understand and to evaluate the economics involved.

Historical data obtained subsequent to the installation of two types of computers, the IBM 709's which replaced Univac 1103's and the IBM 1410's which were installed to handle administrative systems, shows the patterns given in Figs. 1 and 2 respectively.

It can be seen that the data in both cases have a basic similarity, and that the experience in general seems to follow the same type of curve. The smoothed curves themselves were obtained by assuming that the development load would be reduced half-way to the steady state load during each four month period.

Data on the introduction of new computer languages have been somewhat more difficult to obtain, and tend to be less definitive. Hardware changes are necessarily abrupt. Software changes need not be. However, records have been found which show the ratio of production to development following a fairly general shift from Fortran I to Fortran II on the IBM 7090's which started in June of 1963. Since the scatter of these data is considerably greater than it was for the introduction of a computer itself, and since figures are not available for rework as a separate item, smooth curves were not derived from the data. Instead, the curves from Fig. 1, the introduction of the 709's, were plotted

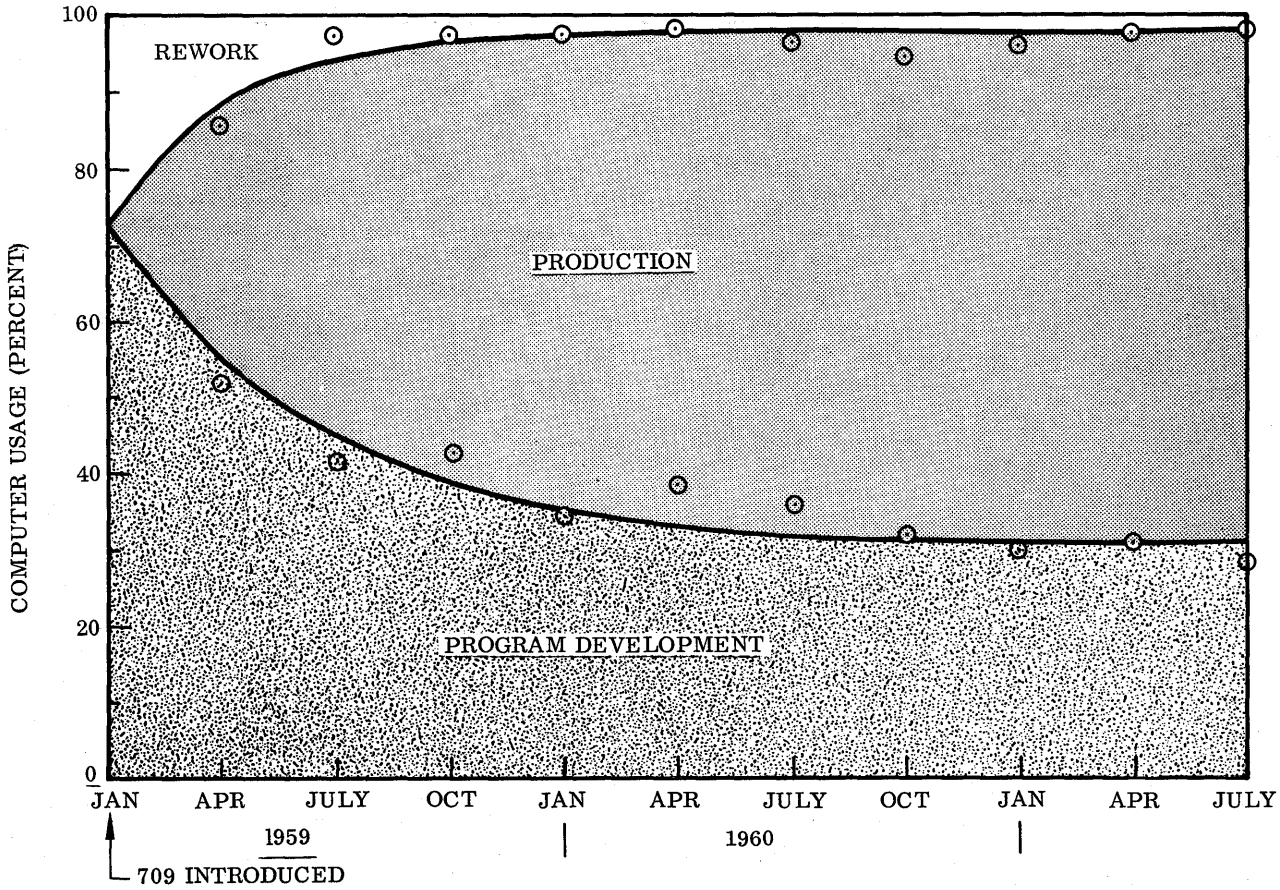



Figure 1. Load components versus time from installation of 709.

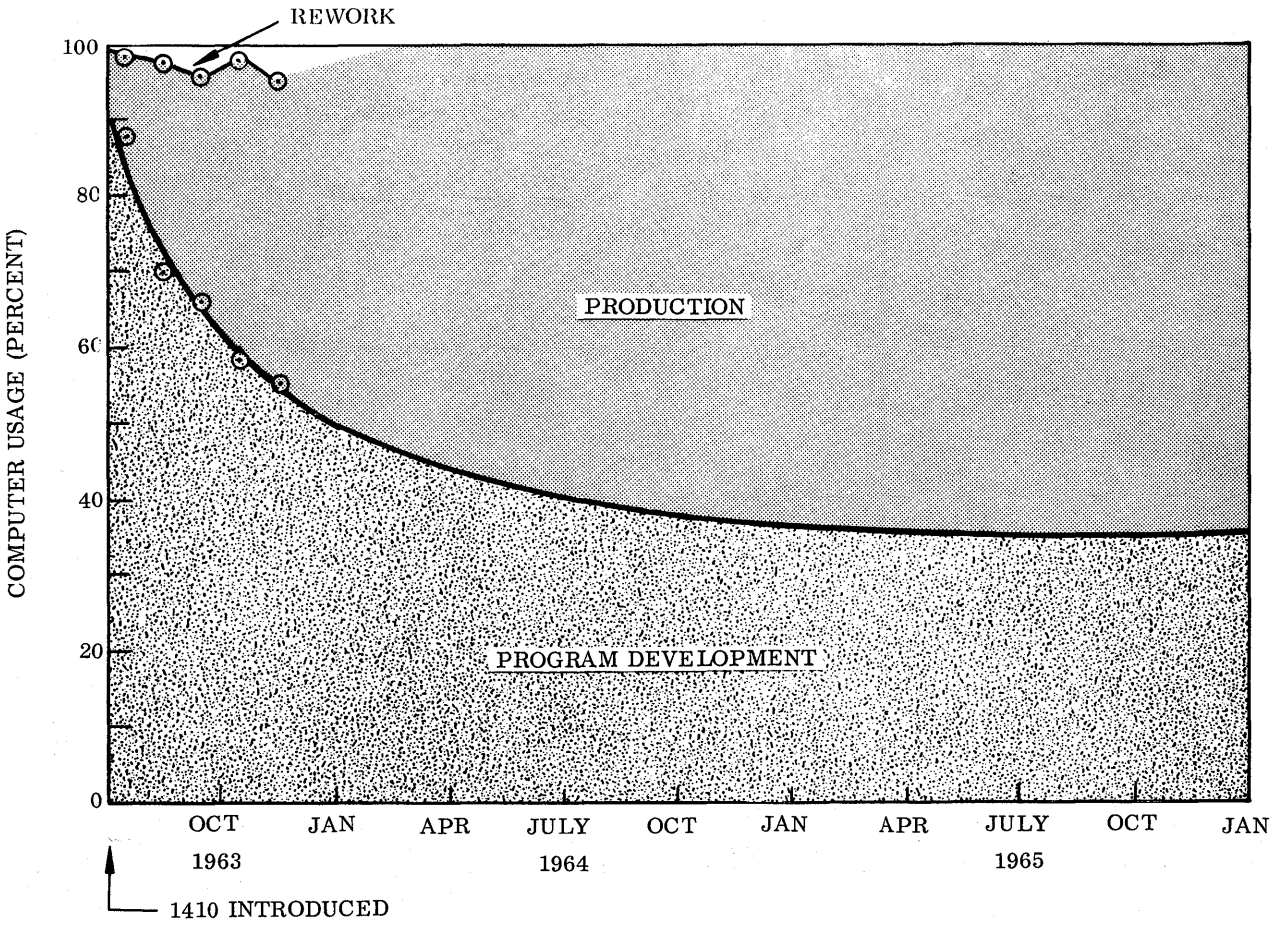



Figure 2. Load components versus time from installation of 1410.

directly in Fig. 3. It would seem that the data at least do not disagree with this pattern.

During the ten-year period from mid-1955 to July 1965 LMSC has employed six distinctly different computers to handle important parts of its work (IBM 650, 709/7090/7094, 1410; Univac 1103, 1107/1108; and RCA 301), and six additional computers either as peripheral units or to serve special purposes (IBM 1401, RPC 4000, CDC 160A, 924, 3200 and GE 415).

During the same ten-year span, in addition to the machine languages of these twelve computers, more than twelve distinct versions of compiler languages or general operating systems have been put into operation.

Each innovation brought improvement. It also brought a new set of problems. With each improvement some portion of the total number of computer users at LMSC has been required to familiarize itself with either a new computer, new computer language, or new system some 24 times in these 10 years.

Since some of these changes affected more than two-thirds of the work load, while others touched only a few percent, it would appear reasonable to assume that on the average each change affected one-sixth of the work load. From this figure it follows, again on an average basis, that there must have been the equivalent of a complete change some four times in ten years. That is not to say that large disruptions can be noted at that frequency, but only that they are actually present in the average figures to that extent. The fact that truly disruptive changes do not show in the records results from several fac-

tors. First, in an operation as large as the one being studied, there are usually several large computers of the same type, which can be phased into or out of the operation over a long enough period of time to give a nearly smooth operation in the aggregate. Second, the introduction of a new language can also be phased in rather smoothly by involving only a few programs at a time. As you all probably realize, phasing a language out is an even more gradual process than introducing it. It may not be completely impossible, but it is almost impossibly difficult.

Nevertheless, the net effect of changes must have the same cumulative effect as would a complete change every two and one-half years. The following analysis is based upon that frequency.

Figures 1, 2, and 3, together with the mean

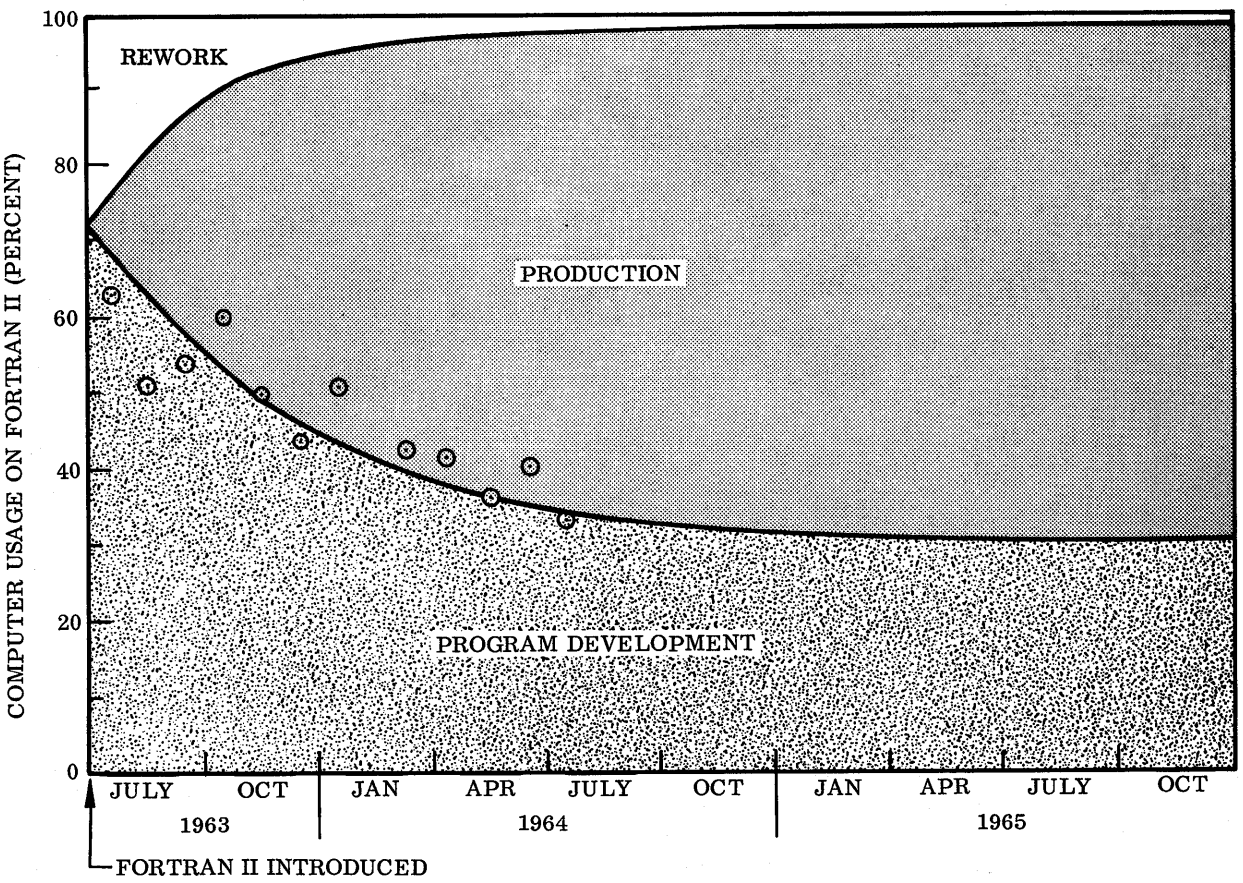



Figure 3. Load components versus time from introduction of Fortran II.

change rate of 0.4 changes per year, provide the data required to calculate the average annual cost of change.

Subtracting the asymptotic values of the curves on these figures gives a value of (98 – 30) = 68 percent for the steady state, which must be contrasted with the integrated value for production, which has an average value of 59.7 percent. Since the production time on a computer is the only time which is of actual value to the user, we may compare these figures directly with the result showing that the steady state condition would provide (68 – 56.7)/56.7 or 20 percent more useful time than that obtained with the change frequency prevailing.

In addition to the information on machine time which these curves give, they also show us something about the way in which our programming manhours are spent. Specifically, they show that the

amount of development which must be directly attributed to this change frequency is the difference between the integrated mean value, 38.8 percent, and the asymptotic value, 30 percent. These data provide the estimate that the gross programming cost of the change frequency is (38.8 - 30)/30 or 29.3 percent of the total programming costs.

At this point, then, we can state that a proposed change must promise a gross cost reduction equalling the total of 20 percent of current machine cost plus 30 percent of current programming costs in order to break even.

Now let us take a moment to look at the "Turn-Around-Time."

Some of you will remember the queues that formed and the average times that prevailed when a simple diagram was used to show the path of an individual job. It was something like the following:

| | DISPATCH DESK | Q | SET-UP TABLE | Q | CTT | Q | CPU | Q | TTC & TTP | Q | DISPATCH DESK |
|---|---|---|---|---|---|---|---|---|---|---|---|
| JOB→ | | | | | | | | | | | |
| HOURS | 0.05 | 1.00 | 0.10 | 1.00 | 0.10 | 1.00 | 0.10 | 1.00 | 0.10 | 1.00 | 0.05 |

This diagram purports to show that while CPU time for the average job may be 6 minutes, and the peripheral equipment time another 12 minutes, the five queues which form may add 5 hours. In the past, we have occasionally been guilty of jumping to the conclusion that by combining operations and eliminating 4 of the 5 queues, we could cut the queue length from 5 hours to one. As many of you realize, in practice it only works that way as long as there is an excess of machine capacity over work load. As in most shops, the various items of equipment are already fairly well-balanced to each other before the figures above could be obtained, rather like a series of five synchronized traffic signals.

Actually, of course, the basic problem lies not in balancing the equipment (even though that must be done) nor in eliminating all but one queuing point (the one remaining could be nearly as long as the sum of those replaced), but in balancing the equipment to the job load.

Fortunately, or perhaps unfortunately, I'm not sure, we have one rather automatic balancer inherent in our system, to the effect that whenever the turn-around-time goes up, the rate at which we receive jobs goes down. The mechanism behind this effect is obvious when one considers that a programmer can't remove a bug and resubmit a developing program if he never gets it back.

The balancing of equipment work load is thus the real problem, which management must solve on the basis of the most economical operation in the broadest sense.

While no one has yet come up with an accurate estimate of the cost of delay in turn-around, we can start with a highly over-simplified picture, making use of Erlang's basic queuing equations:

$$D = \frac{LP}{(c-a)}$$

$$\text{and } P = \frac{a^c}{(c-1)\,!\,(c-a)}$$

$$\left[1 + \frac{a}{1!} + \frac{a^2}{2!} + \cdots \frac{a^c}{(c-1)!\,(c-a)}\right]^{-1}$$

to obtain the average delay $D$ as a function of mean job length $L$, the mean number of jobs received $a$ in the time interval $L$, and the number of computers $c$.

From this equation, and assuming that computer time costs $35,000 per week and that the cost of an hour's delay is 4 percent of the cost of an hour's chine time, Fig. 4 can be obtained.

There are several features worth noting in this chart. If one looks at the cost of doing a job when only one computer is involved, it is apparent that the cost of turn-around delay is not very significant with respect to the economic advantage to be gained by loading the single machine as heavily as is feasible. Even when we allow a large economic factor for the hourly cost of delay, as we have here, examination of the first cycle of Fig. 4 shows why the programmer is seldom satisfied with the turn-around-time in an efficiently operated one machine shop.

On the other hand, when the job load has gone

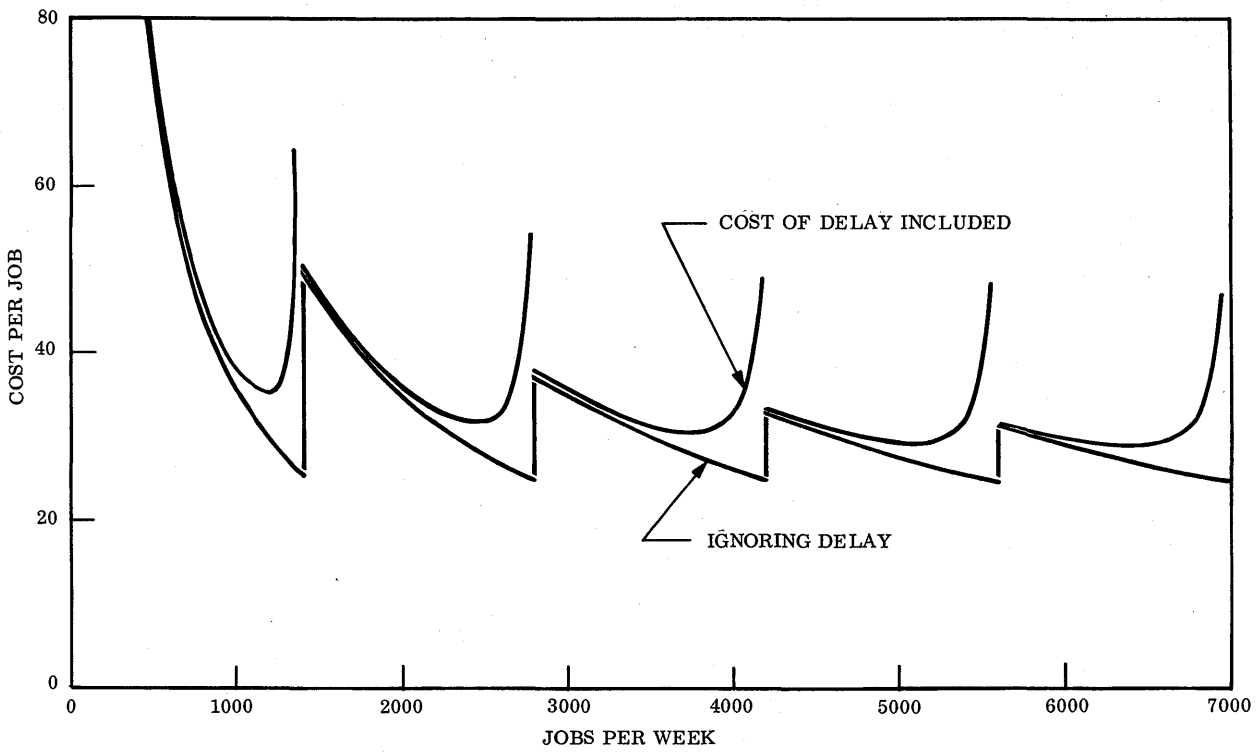



Figure 4. Cost versus loading versus number of computers.

up to the point where four or five machines are required, the relative importance has considerably shifted.

Examining the fourth or fifth cycles, we note that the generalizations based on the first cycle no longer apply. In fact, the greatest danger of excessive cost at this point lies in hitting an over-load condition, rather than an under-load. Further, the difference between the cost per job with optimum loading of four machines, and with the lowest loading of five machines, is less than 7 percent.

While Fig. 4 illustrates the point I wish to make, the simplifications involved are much too gross for efficient operation, so that we have found that a rather elaborate computer program is really required. Such a program, called LOMUSS (Lockheed Multiprocessor Simulation System) has been developed, and will be reported upon separately.

Now I would like to take a few minutes more to make some observations on the evolutionary course that we might expect the next few years to follow.

A few minutes ago we noted that over the last decade there had been a large number of changes in our computers, our computer languages, and our operating systems. One's first reaction might be that this is inherently bad (and of course it is) and that to avoid a continuous repetition we should standardize upon a single hardware configuration supported by a single language and operating system. However, if we plot the pertinent data from Dr. Knight's intensive study of the first 225 digital computers,[2] we obtain Fig. 5, from which it is obvious that the computer field has not yet reached the point at which we can afford to standardize on hardware.

On the software side the situation does not appear to be as obvious as Fig. 5 showed it to be for hardware. I assume that you will all agree that, despite the advances made, from machine language through assemblers, Macro-assemblers, procedure-oriented compilers and problem-oriented languages, the economic advances which have been made are still unimpressive when compared with hardware improvements shown in Fig. 5.

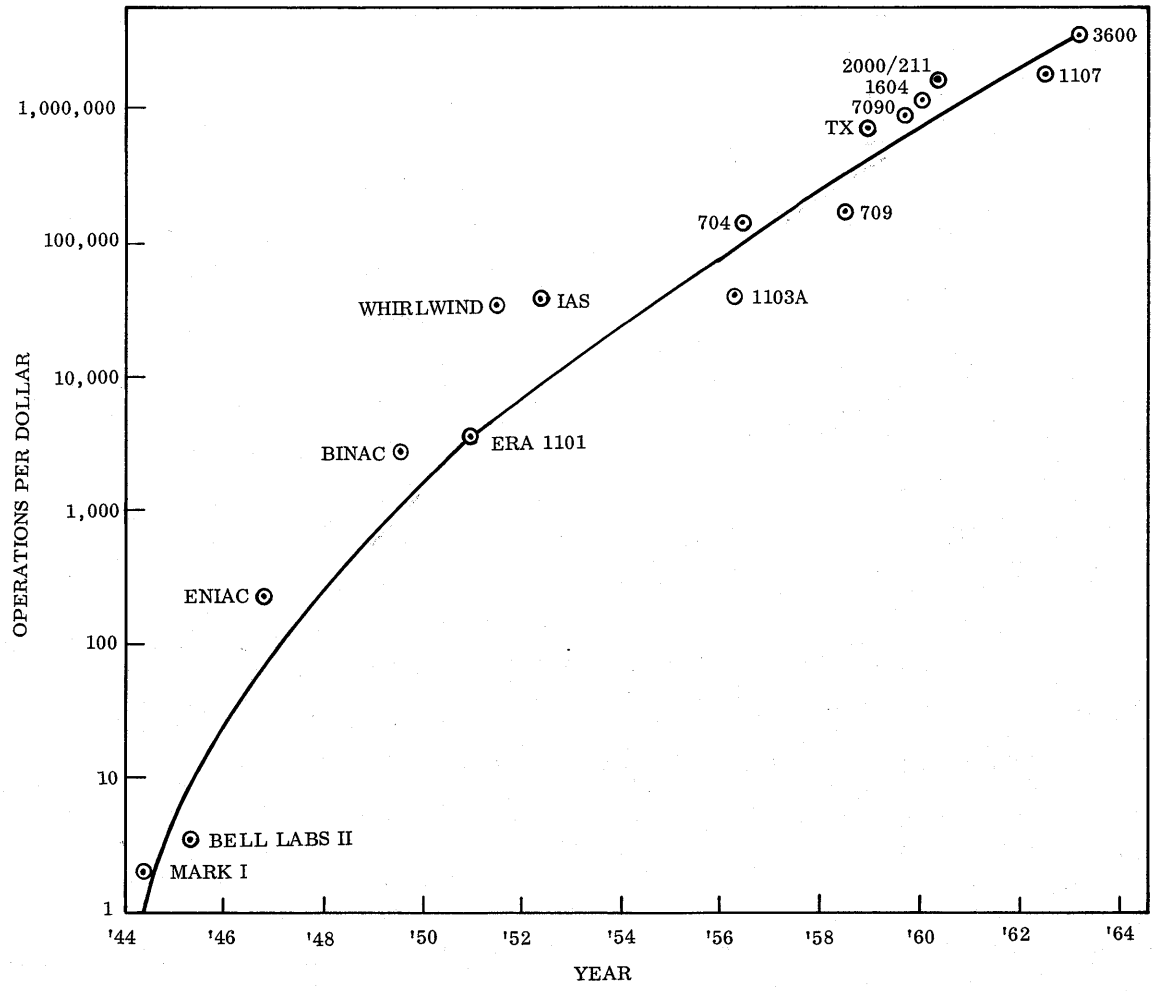



Figure 5. "Computer Operations" per dollar versus time.

Consequently, I have the feeling, just as many of you have, that despite the advances yet to be realized in hardware, it is in the software area that the greater improvements should be sought.

Rather than expecting revolutionary advances, such as software which merely accepts a few sets of test data and automatically generates the algorithm required, it would seem more reasonable to expect that progress in software will be evolutionary.

If we look at the way that man communicates we see that, while he uses a native language as his basic tool, most of his working time is spent using some special jargon imbedded in the basic language. This is true whether he is a physician, a chemist or a truck driver. In order to explain something to another physician, chemist or truck driver, he uses the jargon of his trade in order to save the

large amounts of time and effort which would be required to communicate in the basic language.

This point is easily illustrated by the story of the gangsters who had taken over a legitimate theater. They didn't realize that when a stagehand yelled "Take the silks off the broads," he meant to remove the light diffusers from a certain type of stage light.

While this idea of a jargon imbedded in a basic language is neither revolutionary nor glamorous, it does seem to be the way that software might progress. Individual jargons could gradually be developed, in the form of special-purpose libraries of functions, subroutines, procedures or what you will, to serve each of the definable areas of need. At LMSC we might expect a dozen or so, in areas such as thermodynamics, trajectory work, real-time control, accounting, medical support, etc.

Before we can develop these various jargons,

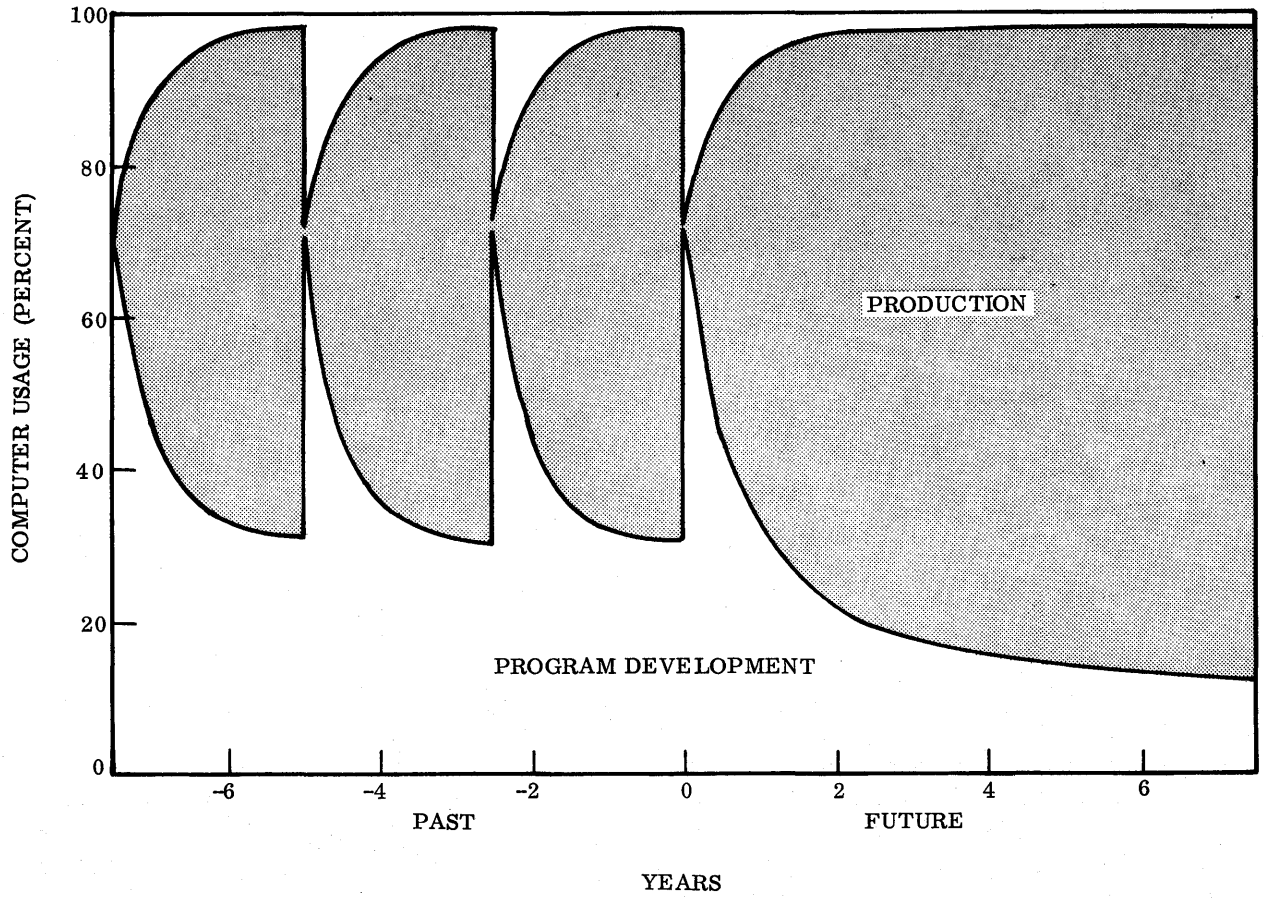



Figure 6. Past and potential percent production versus time.

however, we must have a basic language in which to imbed them; for remember, these jargons are not to be distinct languages, each requiring a new compiler and operating system.

Further, such a basic language must itself have certain characteristics, such as ease of use, efficiency, open-endedness, complete machine independence, perhaps even manufacturer independence, and ease of implementation for new computers.

Given such a basic language, and the development of the needed jargons, then we might expect that the composite of Fig. 1, 2, and 3 would more nearly approach Fig. 6, where the economy in hardware costs shown by the increased ratio of production to development load is also a reflection of increased programming efficiency. Further, since there would be no reprogramming, all of the programming time would be devoted to new programs.

## REFERENCES

1. R. H. Bowman and R. B. Fetter, *Analyses of Industrial Relations,* Richard D. Irwin Inc., Homewood, Illinois, 1959, pp. 300, 301.

2. K. E. Knight, "A Study of Technological Innovation-The Evolution of Digital Computers," Ph.-D. Thesis, Carnegie Tech. (1963).

# THE MULTIDISCIPLINE APPROACH: A MARKETING APPLICATION

Burton G. Mendelson and Ralph V. Monaghan
*Motorola Instrumentation & Control Inc.*

No approach is more widely heralded, yet more effectively ignored in actual practice, than the multidiscipline technique in systems design.

It has been long apparent that for effective systems design practitioners must:

1. understand the dynamics of the specific business,
2. know the difference between essential and nonessential business activities,
3. be aware of—and understand—existing technology and emerging developments,
4. appreciate the economics of developing specialized equipment versus using standard equipment, and
5. be sensitive to the external environment in which the company must operate.

These criteria are difficult to meet; the American business scene is littered with systems which aptly demonstrate insensitivity to the real systems needs of businesses.

Once management makes a decision for a traditional capital investment they proceed to allocate the neessary ingredients to make that capital asset a reality. If it's a new factory, they provide funds, time and the necessary personnel to complete the project. No ingredient is left out, for that elimination could jeopardize the project. Assistance is secured from lawyers, bankers, real estate men, engineers—everyone that is required. The latest technology is used; the most efficient layout is determined. The life of a plant is long and initial mistakes can prove crucial, even fatal, to that life.

## SYSTEMS DESIGN

But take the same management and give them the problem of providing an internal system which may be critical to corporate health and watch out! The finely wrought sense of proportion disappears. The problem is considered to require a departmental solution for, after all, the company is organized to provide departmental-type solutions on a daily basis. The personnel involved generally include those who have an accounting background plus the usual array of programmers. It is no coincidence that the first computer business applications for most companies encompass the financial areas.

Somehow the early history of bookkeeping machines and accounting machines lingers in the background as the proper ancestors for the "new system." The boundaries are departmental, personality and supplier. The lingo is new, the technology is exciting, but the application is archaic.

The application of the multidisciplinary approach has been, when used, in defense activities or in civilian manufacturing projects. Rarely has it been tried in the area which is the stepchild of the

data processing revolution—marketing. And yet, the potential benefits here are staggering. The sheer size of the field, the cash flow involved, the multitude of problems all add up to the need for intensive, structured problem-solving.

## RETAIL GASOLINE MARKETING

In this area, one of the most complex marketing operations today is the retail gasoline business. Everyone is an expert in the gasoline market! The sheer size, the pricing, the distribution, the vast amounts of capital involved make this a prime candidate for automation consideration. And this incredible complexity makes it a primary candidate for the multidisciplinary approach. Not only is the traditional systems experience and the obvious electronics design talent needed, but a measure of legal, marketing, and accounting background is required.

Much work has been done in the automation of distribution facilities in the petroleum business. Today a driver using a credit card can secure access to a bulk terminal and draw the products he is authorized, while the system records who he is, where he is taking the product, what the product is, its temperature, the time, product totals, and tank levels. This data is then often transmitted to remote central processing locations, usually by teletype. Invoices or bills of lading are prepared in many cases and returned to the driver before he leaves on his delivery.

Fine, that solves one part of the marketing system. But why not use the same general idea and apply it to the over 220,000 service stations in the United States? The techniques are known, the idea is here, so all we need is a black box!

Unfortunately, such simplification rarely produces valuable results.

Since Motorola has had considerable experience in the petroleum field it was decided to take the multidiscipline approach to this problem. A group was formed embodying the disciplines which were considered essential: electronic design, communications, internal systems, marketing.

The actual key to the ultimate solution was in defining the basic data need of the operation. In this case information content was in each gallon of gasoline as it was pumped into the motorist's automobile. The resulting statistical structure depends upon that one element. (Actually petroleum marketing data has been based historically upon the quantity of gasoline delivered to a station. Actual sales to the motorist have not been available on a practical basis.) Capture that data at the source. Simple. But do it rapidly, accurately, periodically and on-call, and, above all, inexpensively. There was the design problem.

## EQUIPMENT AND PROCEDURES

The resulting hardware and procedures was an outgrowth of automated terminal techniques married to Bell System DATA-PHONE techniques. The system, as designed, applies not only to service stations, but to any outlet where a discrete amount can be measured. It does not matter whether the measure is in gallons, barrels, pounds or tons. The increment is recorded and transmitted on request to the central location in machine-acceptable form.

The remote station equipment consists of impulse transmitters installed on each product pump or point of flow. One pulse for each gallon of throughput is stored in magnetic memory. The totals of all like products are added together and held in storage for interrogation by the central. The remote also contains a 3-digit station identification code along with the necessary equipment for tying into the Bell System DATA-PHONE installation.

The central equipment consists of a Motorola console for inquiry of the remotes and for recording the captured data. Included are a typewriter, paper-tape punches, a manual entry unit for adding local information, together with automatic telephone dialing equipment of the Bell System and the receiving DATA-PHONE equipment.

Dialing from the central is accomplished automatically at set times by feeding paper tape containing the telephone numbers into the dialer, by using the different types of card and magnetic tape dialers, or by manually dialing the remote number.

The data returned from the remote station includes throughput figures for each product, the identification of the station, and status codes pertaining to the condition of the pumps.

The need for a one-way DATA-PHONE was solved when the Bell System developed their low-cost 401-H for this application. The call is made always from the central to the remote, consequently, the normal two-way dataset was not required.

An even newer dataset permits the central control station to turn off and on the remote product dispensers.

## BENEFITS TO MANAGEMENT

The benefits to management from having such source sales information immediately available on demand are difficult to calculate. The data can be useless or it can be invaluable. The critical feedback is now within reach of all levels of management. How does one calculate the pay-out of individual marketing decisions except in the overall health of the marketing effort?

For petroleum marketing management, this system, for example, provides immediate response by station to the sales campaigns, new marketing concepts, special events, credit card promotions, cooperative advertising, radio and television programs so important to a successful marketing effort. It provides rapid response to the sales effect of weather conditions, traffic flow, road impediments and other local occurrences on a specific time basis—information unique to highway-type locations. Further, it eliminates statistical distortion of "sale by delivery," and above all, provides the basic data for market studies in depth.

No one point in the petroleum distribution picture is more critical today than the economic health of the service station dealer. With an annual operator turnover in the neighborhood of 25 percent, this group has been caught in a spiraling profit squeeze. The objective was to achieve the following effects:

1. Eliminate the basis for dealer complaints and fears associated with the effect of price changes on inventories.
2. Eliminate manipulation of inventory during periods of price fluctuations.
3. Eliminate paper work.
4. Reduce possibilities of product substitution.
5. Allow tighter inventory control.
6. Eliminate danger of "running dry."
7. Eliminate any necessity for trucks to deliver only during open hours.

Happily, the resulting system design met these qualifications.

It is now possible to suggest a model financial management system to a dealer having a substantial chance of being successful. By eliminating the periodic large withdrawals from the working capital of a dealer for delivery payments, a systematic payment schedule to the supplier can be established. This, together with the elimination of the large gasoline inventory investment usually required, provides much of the groundwork for a financially stable dealership. With an operation that survives on the slimmest of margins, the potential saving is substantial.

The invoicing of service station deliveries has long been a cumbersome error-filled procedure. Even with computers available for the processing of sales data, the reliability of the high-speed output is still tied to the accuracy of the ofttimes manually prepared input data.

An extensive system of paper work has been developed over the years to provide the driver with pricing and tax information to prepare invoices at the time of delivery. To back up the driver, clerical staffs have been formed at various levels to correct his mathematical and coding errors.

By installing this data acquisition system, drivers are allowed to do what they do best: drive. The dual functions of pricing and billing are transferred to computers where they can be most economically handled. The tape output from the central control is fed directly into the data processing flow. Individual invoices are prepared or, more economically, statement-type invoicing is instituted. Under the latter method, the dealer receives a combination statement/invoice which records product used at whatever prices existed during the specific periods, together with remittances received by the company.

The severe competition of major commercial accounts has put a premium on services which can be offered profitably to these important customers. The theory of the supplier carrying the financial burden of a large product inventory applies as well to this type of account as it does to a service station. By installing the system at the location of a large consumer account, the customer can be converted from a "charged as delivered" to a "charged as used" basis. Whether the customer operates a truck fleet, a manufacturing facility, a marina, or an airport installation, is of little importance. This elimination of his investment in stored bulk product is a tangible dollars-and-cents advantage to him.

An important by-product of the Motorola data acquisition system is the new tool for credit administration that it generates. By checking product usage with remittances received on a basis suited to the particular customer, credit management assumes a new aspect.

Rather than waiting the many days it usually takes for a credit man to find out that a delivery

was made, the office charged with credit control knows, on call, the product actually used by the customer. This information, correlated with remittances received, gives an accurate up-to-the-minute picture of the credit standing of the customer.

Cumbersome credit control through the dispatcher, the plant cashier and the driver disappears and the techniques of credit management are allowed free play. Instead of adapting credit terms to the peculiarities of the distribution pattern, terms can be tailored to the requirements of the individual customer.

Nowhere is the effect of the system more dramatic than in the operation of the bulk plant/terminal. By using the full advantages of the system, the terminal becomes strictly a physical installation for the storage and dispensing of product rather than the major clerical organization it has been.

By removing the now extraneous activities from the bulk terminal, the real economics of product distribution can be investigated. No longer are dealers' hours of operation and credit positions a restriction on the use of the truck fleet. No longer must dispatching be done by the "seat of the pants."

Managerial abilities become free to determine the size of truck fleets, to schedule personnel hours, and to plan distribution patterns without outside restraints. By removing much of the terminal clerical overhead costs and reducing delivery expenses, the economics of distribution undergo a major improvement.

Having automatic equipment guarantees no competitive advantage to a company. However, the imaginative application of that same equipment can bestow a substantial competitive advantage. The very nature of this automated system—capturing data at the sources—allows many of its advantages to accrue in existing internal operations. However, the full application of the system changes much of the marketing distribution, accounting, and data processing structure as it is operated today. Obviously, the major payout rests in full integration. These changes are compatible with the "total systems" trend so evident now:

1. One-time automatic collection of data at the source,
2. transmitted automatically,
3. to high-speed processing computers
4. for distribution to all levels required, when required, and in the form required.

The automated terminal equipment used in conjunction with the data acquisition system completes the automation picture for bulk product movement.

## THE CYCLE IS COMPLETE

What previously was considered a desirable but strictly "blue sky" operation is today a practical, existing situation. Let's describe it:

> Each evening the console operator in the area office brings in throughput figures for all stations, large commercial accounts, fuel oil jobbers, and leased storage in a major geographical area on the flat-fee WATS telephone line, which has been used during the day for normal service. The output punched tape is fed to a small computer which prepares a dispatch schedule for the bulk terminals delivering in this area. The schedule is transmitted by teletype to each terminal during the night.
>
> A by-product of this process is a listing of station sales on a salesman and district breakdown basis. This information is transmitted by teletype to each district sales office during the evening. Another by-product is the updating of accounts receivable, gallonage rents, and trend sales statistics for area and higher management. *All information is available at the opening of the new business day.*
>
> During the early evening at each terminal, the by-product tape from the loading of all trucks and the measurement of the tank levels is transmitted to the area office. This information supplies the raw data for the bulk stock position. No calculations are made at the bulk terminal; rather, the facilities of the area computer are used. Should complete results of the stock picture show discrepancies, the information is transmitted by teletype to the bulk plant or terminal manager *in time for corrective action.*

Variations of the above, based on organizational structure and geographic dispersion, are infinite. But the principles of rapid collection of data in machine-sensible form plus immediate distribution of the refined data to management are inherent in the system in any pattern.

Data processing, distribution, and management information are integrated into a single economic flow. By eliminating—not changing, but eliminating—many of the steps in distribution and accounting, the tangible payout is substantial. The intangible benefit can be incalculable: a strong competitive position.

## A VALID APPROACH

What I have outlined is an integrated system for a particular industry. It presumes an integrated environment for maximum operational benefits, an environmental situation which does not exist today. The development of this system is a result of the multidiscipline approach to problem-solving. I think it has been particularly successful in its relationship to the dynamics of petroleum marketing. Will the results of the multidisciplinary approach ever really be satisfactory in today's corporate climate? Can departmental judgments ever be brought to bear adequately on the results of such multidiscipline solutions? The approach is valid, the techniques are proven. But now the real challenge: can system integration be effective in a nonintegrated enviroment?

The ripple effect of the multidiscipline approach is so widespread, the adaptability of the modern corporation so enormous, the economic pressures for profitability so great, the obvious trend must be toward an environmental change.

What we are really reviewing is the process of dynamic change with a glance at the near future, a future shaped by the creative combination of talent and technology.

# ORGANIZATIONAL PHILOSOPHY AND THE COMPUTER CENTER

Malcolm H. Gotterer
*The Pennsylvania State University*
*University Park, Pennsylvania*
and
Ashford W. Stalnaker
*Georgia Institute of Technology*
*Atlanta, Georgia*

The computer has been so widely accepted as a tool of management that there is no longer any debate about its ability to serve the needs of management. But, unfortunately, the introduction of a computer does not in itself guarantee increased effectiveness, efficiencies or profits for the firm. Rather, the computer must be viewed as still another management resource from which certain benefits can be extracted but these advantages do not automatically result from installation of the device. Even in view of this fact, the number of computers being used in business continues to increase at a rapid rate, how rapid being dependent upon which set of projections one chooses to believe. Today, the question is not should a computer be used, but rather how management can best use the computer to serve its needs.

In its survey McKinsey and Company[1] found that while some companies were able to make profitable use of computers others were not (profitable being used here in the sense of both direct and indirect benefits). Their survey indicated that the realization of the computer's potential has its basis in recognition of managerial rather than technical requirements. The companies showing the greatest profits were typically those which recognized the potential of the computer for problem solution throughout the organization. On the other hand, the less successful users were using computers for routine unimaginative activities. Perhaps the most important conclusion of the report is

> In the successful companies, top management demonstrated a balanced point of view on computers' potentials and demands. In the average company, by contrast, corporate executives typically underrate their potential or else overrate the importance of the technical requirements.[2]

In another study made by the National Industrial Conference Board, the failure of managerial personnel to involve themselves with the computer's potential was indicated by the conclusion,

For the purpose of this study,

> Most managements want to know the reasons (supported by facts) why their companies should approve the use of electronic computers. But they are seldom concerned with the mechanics or operating details, execept as they affect the policies or general organization structure of the company.[3]

This paper deals with certain aspects of the need for a balanced point of view by management rather than the indifference so frequently encountered. Intensive studies of eight computer centers suggest that management philosophy toward the computer's potential and the role of the computer center in the organization is one of the crucial variables in achieving profitable utilization.

For the purposes of this study,

> A computer center consists centrally of a computer, of the stored program type, surrounded by the supporting personnel and equipment to use it. Jobs or demands for services of the center originate both within the center and from organizational units outside of it.

The computer center is unique in that it focuses upon a machine and will therefore be an organizational entity regardless of where it may be assigned within the firm. The attitudes and perceptions of the managerial personnel of the firm will dictate the legitimate activities the center may undertake and will thereby influence the orientation of the personnel assigned to the computer center. Central to this discussion is the concept of classification of the center since it is in this way that management tends to bound its perception of the computer's capabilities. Therefore, we will first consider classifications of centers and then the relationship of such classifications to personnel practices and performance.

## CLASSIFICATION OF COMPUTER CENTERS

Traditionally, computer centers have been defined as either business or scientific. "Business" generally meant that the center processed essentially accounting information such as payrolls, inventory transactions, cost data, etc. "Scientific" implied that the center was engaged in solution of unstructured and often loosely defined problems. In the early days of the application of computers to business these classifications may have been useful. For the business center a good accountant could be trained to translate manual or unit record procedures to a program for a "business" type computer. On the other hand, engineers, scientists and mathematicians were required to provide comparable skills for the more expensive and more complex "scientific" computer. Whatever usefulness the two categories may have had has disappeared and in most cases their retention tends to so limit management's perception as to inhibit the development of effective computer operations.

Rather than labeling the computer center in terms which tend to describe the nature of its predominant activity, we suggest management's perception of the center and its role in the firm should be in terms of the degree of structuring of the center's tasks, a concept originally advanced at the 1963 meeting of the Computer Personnel Research Group.[4] By degree of structuring is meant the amount of independence the computer center has in dealing with an assigned task. For example, a payroll would usually be a highly structured task. Similarly, processing of checks in a bank would be highly rigid. In both examples the design of the computational procedure is dictated by the data requirements of the firm. Further, these structured tasks tend to become rigid due to volume and legal aspects inherent to the nature of such processes. At the other extreme would be the development of a computer-based simulation of the firm where experimentation and analysis of alternative policies are the objectives. This would be classified as an unstructured task. Thus in the general category of business applications we find both structured and unstructured jobs are among the jobs which may be presented to the computer center. In similar fashion, scientific applications may possess the same range of structuring. The routine reduction of orbital data is, for example, highly structured, while the development of new computer techniques to analyze such data is conversely less structured. It is in terms of a continuum with these two end points that a classification scheme should be developed for a computer center rather than in the descriptive designations of business or scientific.

The center whose mission is *unstructured* will receive assignments from various organizational units and will be required to use a great deal of ingenuity and flexibility in defining and solving the problems presented; a *structured* problem is one which would require the computer center to routinely process a set of data in a prescribed manner. In discussing the design of scientific computing facilities, Wagner and Granholm[5] have used a somewhat similar conceptual framework for classifying scientific type computational requirements.

Seldom will one find a computer center which is solely devoted to unstructured type of activities, but it is relatively simple to identify many which oper-

ate only on highly structured jobs. Computer centers are required to adapt to the needs of the organization they serve and to process jobs which are both structured and unstructured. Typically a center within a business organization will be processing routine payrolls and other accounting information and will also be engaged in solving management science problems which may involve computer simulation, linear or non-linear programming, queueing theory or other such methods. Non-management science tasks frequently are presented by engineering or research groups. Among these tasks might be, for example, the solution of a problem requiring the inversion of a matrix, generally a highly structured job. On the other hand, the engineer may ask the computer center to develop methods for appraising and analyzing potential new methods of dealing with old problems, an unstructured task.

Subsequent to the original research on which the paper is based another element which must be taken into consideration is that of the changes in available equipment. In particular is the availability of equipment which will permit the simultaneous processing of both structured and unstructured problems, regardless of their organizational origin. Also to be considered is the availability of time-sharing systems with remote consoles which make the computer available to all users without regard to the nature of their problem. In this regard, Fernbach recently pointed out that, "Ten years ago, one was careful not to discuss business and scientific computers in the same tone of voice. Today one thinks of systems that will do everything."[6]

Faced with the problem of simultaneously undertaking tasks occupying different points on the structured-unstructured continuum the computer center must adjust itself to satisfactorily meet both types of demand. Our research has indicated that in this condition two parallel organizations tend to develop within the computer center—one to handle structured problems and another to handle unstructured tasks. It does not appear efficient, or effective, to have a single group handling both types and, as will be discussed later, in some cases it may be undesirable to place differing demands on a single group of employees.

This partitioning of the organization is illustrated by the experience of a large manufacturing firm which makes extensive use of computers. Within its computing center is one group called the Data Processing Department which handles all routine processing of data regardless of the origin of the work, i.e., commercial, scientific, engineering, research, etc. There is also a group referred to as the Scientific Programming Department which deals with non-structured tasks eminating from the same work sources. Each group has its own director who reports to the head of the computer center. This formalized duality of organization represent management's recognition of an informal system which embodied the same concepts but tended to prohibit economical use of available computers. The inefficiencies resulted from identification of specific machines with the two groups and thus restricted the use of a specific computer to either the scientific or data processing group. Thus the newest machine became known as the "scientific computer" even though it was identical to many others being used for highly structured tasks. This did not mean that the machine was idle most of the time; in fact, it was operated three shifts per day. The loss in effectiveness resulted from the fact that many of the scientific-type problems possessed considerably lower priorities than those assigned to many highly structured, routine jobs. Formalization of the dual organizational structure permitted central scheduling for all computers and in this way management's philosophy adjusted to a solution which, although contrary to traditional concepts, permitted more effective operation.

## MANAGEMENT PHILOSOPHY

The relationship of management philosophy to the continuum between structured and unstructured tasks is demonstrated in the ability of the computer center to respond to the needs of the organization. The essential factor is that management face up to two questions:

- Why does it have a computer center?
- What does it need from the center?

In answering these questions management must not only reconsider the initial planning for development of its computer center but must also review the results of their plans. Further, if management expects to obtain anything near optimum use of their computer they must be prepared to devote to this activity an amount of time proportional to its cost and potential.

In too many firms we found these questions are answered by default. The computer is brought in as a highly structured, task-oriented machine. Later, when other demands are placed on the center it typically is not able to perform the service. This results from a failure to recognize the need to prepare for such change. This may, in part, account for the expansion and proliferation of firms specializing in consulting services to computer users. But more important than the added cost of this failure to prepare for changing demands are the organizational pressures and waste which can easily result. A typical and unfortunately too frequent example of such waste is the case of the Linear Company* which we studied in depth. This firm history clearly demonstrates the disadvantages which result from narrow and unrealistic definitions of the role of the computer center within the firm.

The Linear Company installed a large computing system to perform a highly structured task. Relative to this task the computer center was doing an excellent job; in fact, management had nothing but praise for the success of the computer application. However, management's perception of the role of the computer center was "to process orders and effectively maintain inventory." The director of the computer center understood he had under his control a large machine with substantial idle time and a staff which was capable of dealing with other problems. The research department of Linear Company had been assigned the task of developing a computer simulation of the firm which would be used to test management decisions and changes. The director of research and his staff laid out the general nature of the project, engaged a consulting firm to have the simulator programmed, and then contracted with a service bureau for the required computations. Whenever management sought advice on a question which was applicable to the simulator they merely had the service bureau enter the current information and make a run. Then the research department would analyze the output. Naturally the service bureau charged a high, but reasonable, rate for their services. But with idle time in the company's computer center, we might question the necessity of the research department utilizing facilities of other organizations.

We find the answer to this question in analysis of the attitudes of the personnel involved and their preconceptions of the role of the firm's computer center. Management's view was, "the computer center's job is to maintain inventory and associated records." The director of the computer center felt, "we are a computer center and are capable of satisfying all of the company's computational needs." The director of research obviously "heard" management's view and felt, "the computer center can not handle our type of problem." Since their project differed from the accepted mission of the computer center, they sought outside computer services. The consequence was delays in obtaining results and organization frictions and frustrations. The director of the computer center was unable to understand why, "the director of research was so unfriendly." On the other hand, management saw nothing wrong with the arrangement. Management's perception of the role and mission of its computer indicated to the Director of Research what his proper behavior should be. This is an unfortunate example of misuse of available resources.

## THE INTERACTION OF THE CLASSIFICATION SYSTEM AND PERSONNEL

Not only will management's philosophy determine the legitimate scope of activities of the computer center but it will also influence the way in which the center is staffed. This influence will demonstrate itself in that staffing will be in such a manner that it can most effectively respond to management's demands and management's perception of the proper role of the center.

People who are able to successfully communicate and relate with computers may be broken into at least two major groupings, those with predominately professional orientations and those primarily organizationally oriented. By professional orientation we mean people who relate to the profession of computation rather than to a specific firm or to a specific goal of the firm. Typically such an individual will be a member of at least one professional computer group. He is more likely to have been attracted to his present employer from the general computer labor market and will tend to look back there in addition to his present employer for opportunities for advancement. He is the person most likely to be attracted by the professional recruiter at a computer conference or by an advertisement in a computer journal. He thus tends to possess a potential for relatively high mobility, and

*Fictitious.

recognizing this potential he tends to try to increase his abilities so as to develop professionally thereby opening new opportunities for himself not only with his present employer but also in the general labor market. This person will tend to be extremely competent and will usually be best suited for dealing with relatively unstructured assignments. Structured assignments will fail to present a challenge to him with resulting lowered performance and morale. Continued structured assignments may cause him to seek a new employer where assignments are commensurate with his abilities and interests.

The counterpart of the professionally oriented individual is one whose major loyalties and goals are to the firm. He is typically selected for his computer position from some other organizational unit within the firm and then trained for his present position. This person does not usually view the computer field as his profession but rather as another assignment in his progress through the hierarchy of the company. Thus he tends to force the computer into the mold of his area of major interest rather than restating his personal values in terms of the computer. Self-development is likely to be limited to those aspects of computer technology which directly relate to his next position within the organization and to his specific functional profession, if he possesses one. His objectives will tend to emphasize proficiency and greater understanding of his employer's objectives, and on this basis he will tend to be most compatable in dealing with the more highly structured tasks within his field or primary interest. Typical of such individuals are accountants who are assigned to a computer center to implement data processing applications in the accounting area or engineers who are assigned to provide the technical ability required to systematize and program a technical application. Both look to their primary interest fields for advancement and usually do not consider the computer labor market when they decide to change jobs.

The importance of recruiting computer personnel whose personal orientation is compatable with the role assigned the computer center within the organization is further emphasized in a study by Deutsch & Shea, Inc.[7] In this study of 549 programmers it was found that 21.1 percent of the respondents indicated that internal climate, management politics, and working conditions were a factor in causing them to leave their previous job. In addition, 18.4 percent replied they changed jobs because of lack of job interest.[8] This is especially interesting because the study states, "we feel that the sample involved is not representative of the total population in the sense that it is skewed, by virtue of its source, towards programmers with experience in the scientific aspects of programming."[9] The source of the sample was the membership file of the Association for Computing Machinery.

It is possible to distinguish between those programmers with different orientations. In a recent RAND study[10] a number of tests were given to programmers which were then correlated with supervisory ratings. It was found that:

> Correlation with supervisor's rankings for the Programming Aptitude Test and the Test of Sequential Instructions were higher for the scientific sample than for the business sample. For the scientific sample there is good predictive information available from high and low group scores. For the business sample the study shows that the best predictive value is in the second highest test score group and in the lowest group.[11]

In an examination of the interests of both groups, the RAND study reported:

> Overall, both samples expressed an overwhelming preference for mathematics subjects. This would certainly be expected of the scientific sample which was composed largely of people with math backgrounds (63%), but it included the business programmers, of whom only 18 percent had backgrounds in math.[12]

Thus if management has defined its needs it is possible to staff the center with the type of personnel who can make the greatest contribution. If the center's activities are restricted primarily to a set of highly structured tasks, carefully selected intelligent individuals from within the organization can be trained to adequately perform such tasks. If new personnel are needed they might be hired from other businesses with similar activities. If, on the other hand, management is required to go to the general labor market to acquire its computational skills, it could easily and inadvertently disrupt a highly structured center.

## CONCLUSIONS

Management's philosophy as demonstrated in the staffing of the center tends to establish parameters

on the profit potential of the computer center. It appears that in the short run highly structured centers are most efficient. That is, they implement desired procedures and complete assignments in the shortest possible time and at relatively low cost. But this apparent efficiency is short lived, for in the long run the lack of questioning attitudes, the failure of the center to formulate alternative solutions to policy problems if not to completely fail to comprehend the total problem rather than the specific segment under consideration, and the lack of experimentation will lead to higher costs in terms of both production and lost opportunities. Possibly even more serious is the inability of such center to deal with new challenges without outside assistance. This condition is well displayed in the dependence of such organizations on consultants or the technical staffs of computer manufacturers.

Optimizing on short-run considerations is obviously not the best way of introducing the computer into a firm, but this philosophy will probably not change for some time. It has been suggested that the optimal return from the computer results from applications somewhere between the highly structured and highly unstructured endpoints. This hypothesis is in accordance with the findings of the McKinsey study which indicated that those companies which had used computers for only routine, unimaginative classes of activities such as payroll, billing, inventory, etc. failed to show a profit. Those indicating optimal usage were the firm which resorted to what we might call higher ordered applications of computers to problems, problems which emanated from relatively unstructured tasks. But if the center has been staffed solely for fully structured tasks, it will be unable to cope with a change in management's need.

## IMPLICATIONS

From our intensive study of eight computer centers it can be concluded that, while computers may be installed to meet a specific need, a broader frame of reference is required in management's approach to the assignment of tasks and the staffing of the department. A department established to meet rigidly defined or implied managerial needs may never have the imagination or perspection needed to break out of this tight mould and engage in new and more profitable uses of the computer.

On the other hand, the computer center cannot be allowed to run rampant, acting as an end unto itself. As an organizational unit it must show a profit and assume its place as a responsible member of the organization. The NICB study pointed out that "In organizing an electronic data-processing activity, the authority of the manager and the internal and external relationships of the various work units require careful consideration to ensure efficient and harmonious operation."[13] This view is reinforced and expanded by Brabb and Hutchins who state "The primary administrative responsibility of the manager of data processing can be defined as the organization of the EDP department to more effectively contribute to the over-all objectives of the company."[14]

In no way does this portend a new order of things in which management tends to be subservient to the computer center. Rather, attaining the profit potential of the computer requires clear understanding of its capabilities and the adjustment of managerial philosophy over a period of time. Unlike most of its other equipment investments, management should not perceive the computer as a static, single purpose machine facility or as merely another means of increasing productivity; instead the need is for a viable attitude and a constant quest for new applications and uses for the computer.

## REFERENCES

1. J. T. Garrity, "Top Management and Computer Profits," *Harvard Business Review,* vol. 41, no. 4, July-Aug. 1963, pp. 4 ff.

2. *Ibid.,* p. 174.

3. C. G. Baumes, *Administration of Electronic Data Processing,* New York, National Industrial Conference Board, Inc., 1961, p. 45.

4. M. H. Gotterer and A. W. Stalnaker, "The Orientation of Computer Centers and Some of Its Implications," Computer Personnel Research Group *Communique,* Vol. 1, No. 1, Aug. 1963 pp. 7-8.

5. F. V. Wagner and J. Granholm, "Design of a General Purpose Scientific Computing Facility," W. A. Kalenich, editor, *Information Processing* 1965, *Proceedings of IFIP CONGRESS* 65, *Volume 1,* Spartan Books, Inc., Washington, D.C., 1965, p. 283.

6. S. Fernbach, "Computers in the U.S.A. Today and Tomorrow," in W. A. Kalenich, editor, *Information Processing* 1965, *Proceedings of IFIP*

*Congress* 65, *Volume 1*, Spartan Books, Inc., Washington, D.C., 1965, p. 77.

7. "A Profile of the Programmer," *New York Industrial Relation News, Inc.,* 1963.

8. *Ibid.,* p. 44.

9. *Ibid.,* p. 6.

10. R. N. Reinstedt, et al., "Computer Personnel Research Group Programmer Performance Prediction Study," Report No. RM-4033-PR, The RAND Corporation, March 1964.

11. *Ibid.,* p. 19.

12. *Ibid.,* p. 16.

13. *Op. Cit.,* p. 50.

14. G. J. Brabb and E. B. Hutchins, "Electronic Computers and Management Organization," *California Management Review,* Fall 1963, p. 35.

# PLANNING FOR GENERALIZED BUSINESS SYSTEMS

Robert V. Head
*Touche, Ross, Bailey & Smart*
*Los Angeles, California*

## INTRODUCTION

The systems and programming staffs which have grown up within many large organizations may be described with considerable appropriateness as programming "factories." This designation could be used to characterize the programming groups employed by the computer manufacturers and those of the large companies engaged in producing proprietary programs. It is especially descriptive of the technical staff of a large commercial bank.

The idea of regarding the systems and programming resources of a bank as instruments of production has become especially appropriate within the recent past as the emphasis in banking automation has shifted from development of internal systems to systems designed to aid the bank's customers.

This change—and it is a fundamental one—is partly the result of the remarkable progress made to date in automating the voluminous internal accounting and data handling functions of the bank. In commercial banking the "Big Three" applications are demand deposit accounting, savings accounting, and installment loan accounting. In the course of computerizing these applications a great deal of data processing expertism has been developed within the banks, and tremendous equipment capability acquired to perform the work. The banks are now finding themselves in a position to focus their attention away from their own data processing needs and outward to those of their customers.

But this phenomenon has arisen not alone because of the availability of programming and equipment resources to do external tasks. Perhaps of equal importance, it is the result of customer pressure and demand. For, after all, the bank's programming staff could be reduced (though this would be virtually unparalleled in the annals of modern business) once the internal automation job is largely accomplished. This has not happened though, because the bank's customers, having become aware of the bank's technical facilities and of its crucial role in practically all financial transactions, have pressed for development of computer-based services to be made available either on a fee basis or in return for compensating balances.

## COMPUTER-BASED CUSTOMER SERVICES

Among the computer based customer services that the banks now offer may be found:

- Account Reconcilement
- Billing
- Classroom Scheduling

- Credit Union Accounting
- Golf Handicapping
- Insurance Agency Accounting
- Linear Programming
- Payroll

And this rather heterogeneous listing of systems is by no means complete.

These systems are of the "retail" type, i.e., oriented towards businesses rather than individual depositors. There are other services which may be described as consumer-oriented. An example of this is a system for automatic bill payment, widely practiced by banks in Europe and elsewhere outside the United States and now beginning to receive attention here. In this service, an individual may accumulate all of his bills to be forwarded to the bank along with a single check, upon receipt of which the bank makes the distribution to the various payees. Alternatively, the bank might receive the bills directly from the merchants, utilities, etc., and automatically pay them by debiting the consumer's account. Thus it may be seen that when one thinks of banking services as being products of a programming factory, they may be further regarded as analagous to both industrial products and consumer goods.

This transformation of the bank's systems and data processing ability from internal to external applications has created a classical business planning situation, one in which there are more opportunities available than there are resources to exploit them. In this instance there are more potential services for which demand exists than there are technical personnel available to develop them.

There is also a highly competitive environment in which the share of market for the bank's more traditional services is affected by the bank's ability to produce and market these new and complex products. The strategic importance of computer-based services raises the fundamental question: How much business will a bank lose or gain by developing or not developing one set of services in preference to another? If a bank elects to offer a payroll service before developing a billing service, it is vulnerable to its competitor who possesses the billing system. And, strategically, this can mean more than merely forfeiting the profit from this service; it may mean the loss of valuable customer relationships which presently exist. For example, a company desirous of having a bank handle its billing may give to the competitive bank not only the profit from doing the billing work, but may also transfer its demand balances and load relationships, its corporate trust work and other business. It is evident that the staks are high and that the consequence of misjudgment in determining which products to fabricate and sell are at least as catastrophic here as in the more conventional manufacturing operations.

## SYSTEM PLANNING AND PRODUCTION

Let us now consider the major functions involved in product development in any manufacturing company:

- Product planning
- Engineering
- Manufacturing
- Marketing

It should prove instructive to examine the nature of these functions when the product to be manufactured is a system, once again taking the services development work of a large bank as an illustration.

In this context, product planning is concerned with identifying what is potentially both realizable and profitable. It involves, first of all, application analysis to determine the *feasibility* of producing and operating a generalized system, say one which will perform credit union accounting for a large number of potential customers among the credit unions in the bank's service area. Here there are many alternatives. Perhaps it would be better to design a system around the needs of one or a few potential customers rather than to attempt a somewhat theoretical generalization. By taking this more specific, and more limited approach, the service might be proven in, with the resultant experience later applied to produce a more generally applicable package. To do this would be roughly comparable to producing a pilot or prototype model of any manufactured product.

Product planning for systems is also concerned with the available technology. What tools are needed to produce the service? Should programs be written for an IBM 1410, for example, or should the bank plan to upgrade its equipment by acquiring a newer computer? What will be the impact of data communications on possible services?

Finally, product planning for computer based services must be concerned with the *market,* and here

the efforts of a professionally trained market research staff may be required. Market research plays a more important role in consumer type services than in the business oriented systems, primarily because the acceptability of a product to the business segment affected can usually be more readily determined. One bank, for example, felt the need to conduct extensive market research before introducing a new revolving credit service in which transfers of funds were made automatically from the loan account to the checking account whenever the checking account became overdrawn. This research extensively probed consumer attitudes toward overdrafts and the use of credit. Other banks have performed market research in conjunction with plans for an automatic bill payment service.

Product planning for new banking services, supported by market research, should result in the establishment of target costs, target prices, and a break-even point based upon a target market. To do this kind of planning, a great deal or rather astute estimating is necessary, supported when appropriate by market research. First of all, the cost of developing the system must be estimated and amortized over the estimated useful life of the product, an extremely difficult task in the volatile field of computer technology. Costs of marketing the system, converting customers to the system, and operating the system must also be detailed. Here the concept of product line profitability accounting can be usefully applied, with estimates pertaining to a given product accumulated and, as the product is developed and marketed, actual costs and revenues accumulated and compared with the estimates.

If there is to be a realistic cost estimating and accounting approach, there will have to be standard costs for machine usage, keypunch operators, messengers, and all other clerical and control personnel involved in operating the service. Preferably, there should be standard rates which are capable of being translated into cost per item processed. For example, it may be necessary to determine the cost per check issued when considering various kinds of payroll services.

One facet of the planning problem and indeed of the entire product development cycle should be noted. System production is the kind of business in which the lead time between product planning and product use is considerable. Even though there is a continuing buildup in technical skills and experience, it still frequently requires months and sometimes years to produce a complex generalized package like an accounts receivable system. This means that a high quality of technical ability combined with business judgment is necessary to do an effective job of product planning in this new industry. It means also that the level of confidence placed upon the product planning estimates cannot be too high. There are too many technological and competitive variables present during the developmental period which militate against firm estimates.

The engineering function, in the sense in which it is used here, can be seen to be systems engineering. Assuming that a computer based service has left the planning stage, the detailed specifications for the service must now be produced. The end result of the engineering activity should be a programmable specification. But before the engineering process is begun, some sort of formal approval for allocation of resources to the project must be obtained. In the organizational environment of a bank, this authorization frequently comes from a committee of senior management. In any case, the funding of a project should usually be viewed as the start of the engineering function.

The engineering function lends itself well to project organization, as in the case with most product engineering. As in most manufacturing businesses, products may be extensively modified while in the engineering state. It has been mentioned that there may be a pilot or prototype model which will yield experience applicable to the design and engineering of the production models. And, of course, some products may have to be abandoned during the engineering phase, although the rejection of systems ideas should more typically occur earlier, as an immediate result of product planning.

The manufacturing function in services development is, basically, programming. In a well run programming factory, there should not be extensive modification of the product once the specifications prepared by systems engineering have been frozen. One difficult question in organizing for production is whether to carry forward the project organization used to accomplish the engineering work or to keep engineering and manufacturing organizationally separate. What is really involved here is the decision confronting any large user of computers: whether he will combine the tasks of systems design and programming within a single group or whether

he will compartmentalize the work between separate groups specializing in system design and in programming. In either case, there must be a manufacturing function which emphasizes program *production*. This implies the need for tight labor standards and launches the organization into the troubled waters of programmer productivity. Labor standards for software manufacturing are much more nebulous and less controllable than in other kinds of manufacturing operations. Although programming may not be an art, as some assert, it certainly is not at the present time a well disciplined profession. This characteristic has implications for the product planning as well as the manufacturing function, of course, since the ability to produce a marketable product depends ultimately upon the ability of manufacturing to adhere to initial estimates and projections.

The marketing function encompasses a variety of responsibilities, including the firm pricing of the new product, advertising and sales promotion, the selling job itself, and finally the installation of the product, that is, the conversion of the customer to the banks computer processing.

## AN ORGANIZATIONAL VIEW

Let us now take an organizational perspective of the functions just described. Fig. 1 identifies the services activity as well as the internal systems activity of a bank. The systems planning section performs the product planning function as well as the related tasks of analyzing future equipment needs and planning for the internal systems to be used within the bank. The systems development section shown here reflects a functional split between systems engineering and programming, rather than the alternative division which could be made along project lines. The marketing section includes not only sales but market research as well. It may be noted that the marketing section becomes involved in services development at two different stages: at the beginning when the service is still in the planning section and market research is required, and again at the end of the cycle after the system has been programmed and is ready to be marketed. The operations section differs from other computer operations groups in that there must be much more emphasis in terms of organization and personnel on source document control. After all, if the bank undertakes the responsibility of calculating the customer's payroll, there must be every assurance that the documents submitted to the bank by the company are complete, accurate, and amenable to correct machine processing.

## A CONCEPTUAL APPROACH

A third way of looking at systems planning and development is somewhat more conceptual in nature, in that it seeks to identify the major *phases* of planning and development. In Fig. 2, the planning and development of a service is shown in three phases, each demarcated by a vertical line.

1. *Preliminary review*. This initial study seeks to determine whether, from either a policy, marketing, or operational standpoint, the bank wished to invest further effort in investigating a proposed system. This phase is characterized by very brief analyses of many system proposals. Each arrow shown in the diagram represents one proposed system. At the end of this phase, some proposals are aborted and others are carried forward into the next phase.
2. *Detailed investigation*. For those proposed systems which appear to have good market potential, further investigation is performed. This results in a final go/no go decision.
3. *Implementation*. The few systems approved for complete development pass through this final, more protracted phase, the end result of which is an operational system. Unlike phases 1 and 2, phase 3 is typified by many complex subphases and the investment of substantial developmental resources.

The spacing of the vertical lines in the diagram suggests roughly the proportion of calendar time and dollar investment usually involved in the various phases of system planning and development. Actually, the vertical lines should not be straight and parallel as shown in Fig. 2, as the investment of resources may vary considerably in the preliminary review, detailed investigation, or implementation of any one system. Figure 3 suggests a more realistic way of looking at the process. Here the duration of a given systems project varies markedly, especially in the detailed analysis and implementation phases. In terms of organizational responsibility, the services planning section would be responsi-

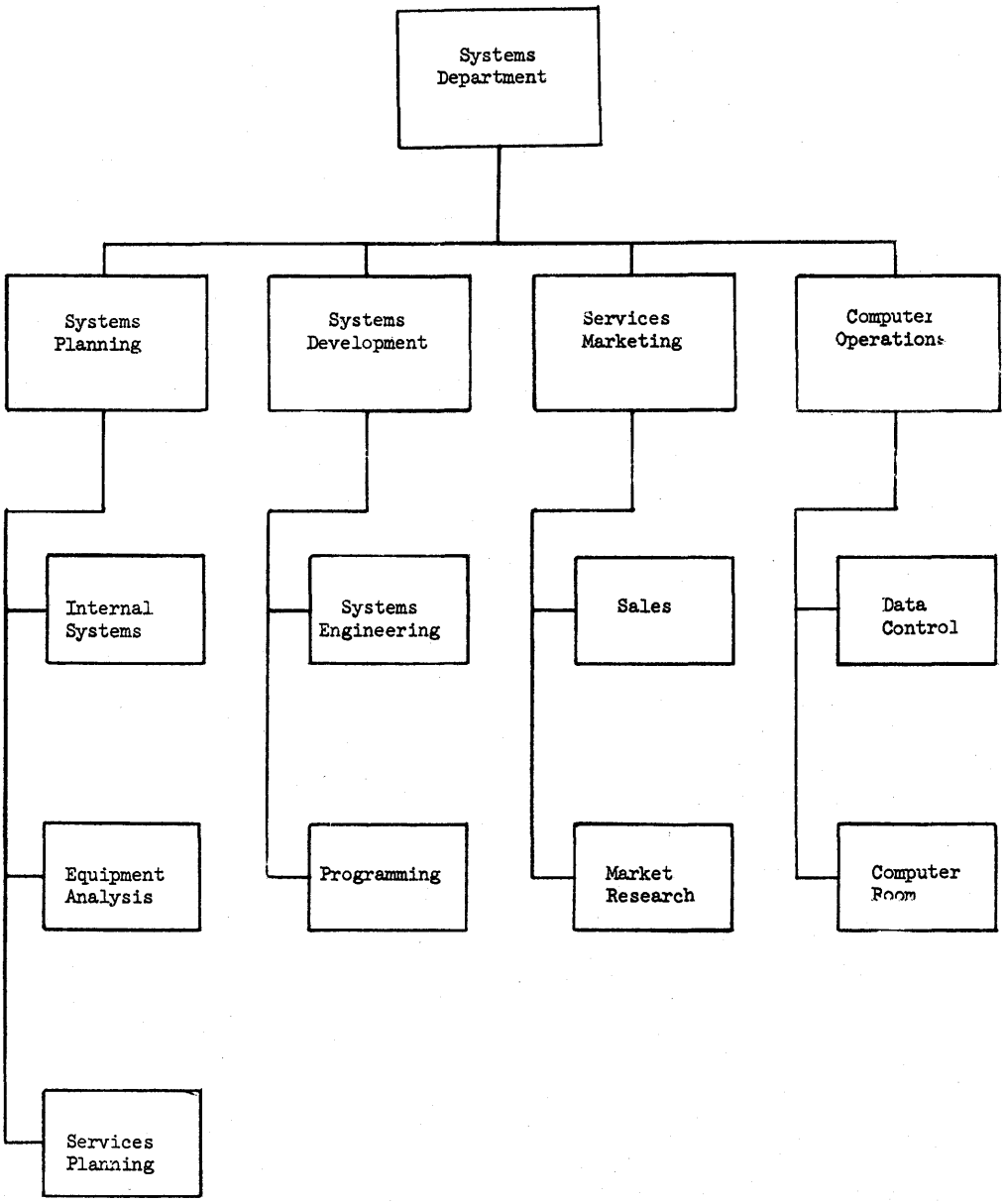
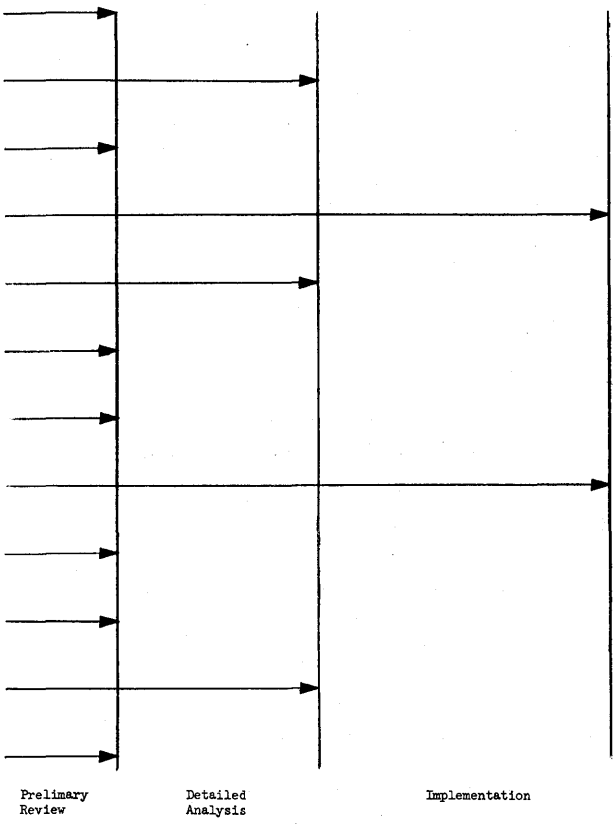



Figure 1. Systems department.




Figue 2. System development schematic — I.

ble for preliminary review and much of the detailed analysis with the other appropriate organizational components accomplishing the implementation.

## CRITICAL PATH SCHEDULING

Figure 3 also shows that the two projects which finally pass through the implementation phase have been planned and controlled by critical path scheduling. Since critical path techniques have been gaining in acceptability for more conventional programming projects, the question of their appropriateness in services planning and development should be considered. If a network could be developed for each potential service during the preliminary review phase, the information produced by such an effort could aid the product planners materially in arriving at better informed go/no go decisions because all facets of the proposed project could be identified and costed. But other factors such as policy and competitive considerations may exercise a more decisive influence on developmental priorities and the allocation of resources than will the detailed projections resulting from critical path scheduling. Furthermore, any expected benefits must be weighed against a number of serious objections to performing critical path analysis at an early stage.

Perhaps the most serious of these has to do with the amount of effort required to apply critical path methods early in a project's history. As Fig. 2 and 3 show, out of 12 projects selected for study in phase 1, only two eventually come to fruition. Therefore, if a critical path analysis has been performed for each project at the beginning, only about 16 percent of the effort required to perform this analysis would turn out to be ultimately useful. Similarly, only 40 percent of the critical path analysis would prove useful if it were undertaken for projects which had progressed to phase 2.

A second objection centers around the feasibility of performing a critical path appraisal prior to the completion of the detailed investigation phase. Generally speaking, one frequently cannot anticipate the scope of the implementation phase until the detailed investigation phase has been accomplished. Consider, for example, the needs of a customer who wishes to have the bank maintain his accounts receivable. Investigation of this customer's requirements may reveal that (*a*) an existing billing system can, with only a little expansion, suffice, or (*b*) an entirely new system will have to be devised. The implementation steps for course of action (*a*) would be trivial but for (*b*) would be tremendous. The point is that the scope of the implementation effort often cannot be analyzed or charted until the detailed investigation phase is completed. If thus appears that the logical place—indeed the only justifiable place—for the efficient exploitation of critical path techniques is during the implementation phase.

This mention of critical path techniques is intended merely to suggest that the tools for project management and control may, for the manufacture of these externally marketed systems, differ from those used previously to control the development of internal systems. Typically, in the latter case, there are considerably fewer choices of projects to undertake, and the characteristics of the systems to be devised are better known at the outset. Thus it is more realistic to draw up a critical path network for an internal system at the beginning because one can be reasonably confident that the effort required to analyze and diagram the project will be effort well spent and that the project will not be aborted.

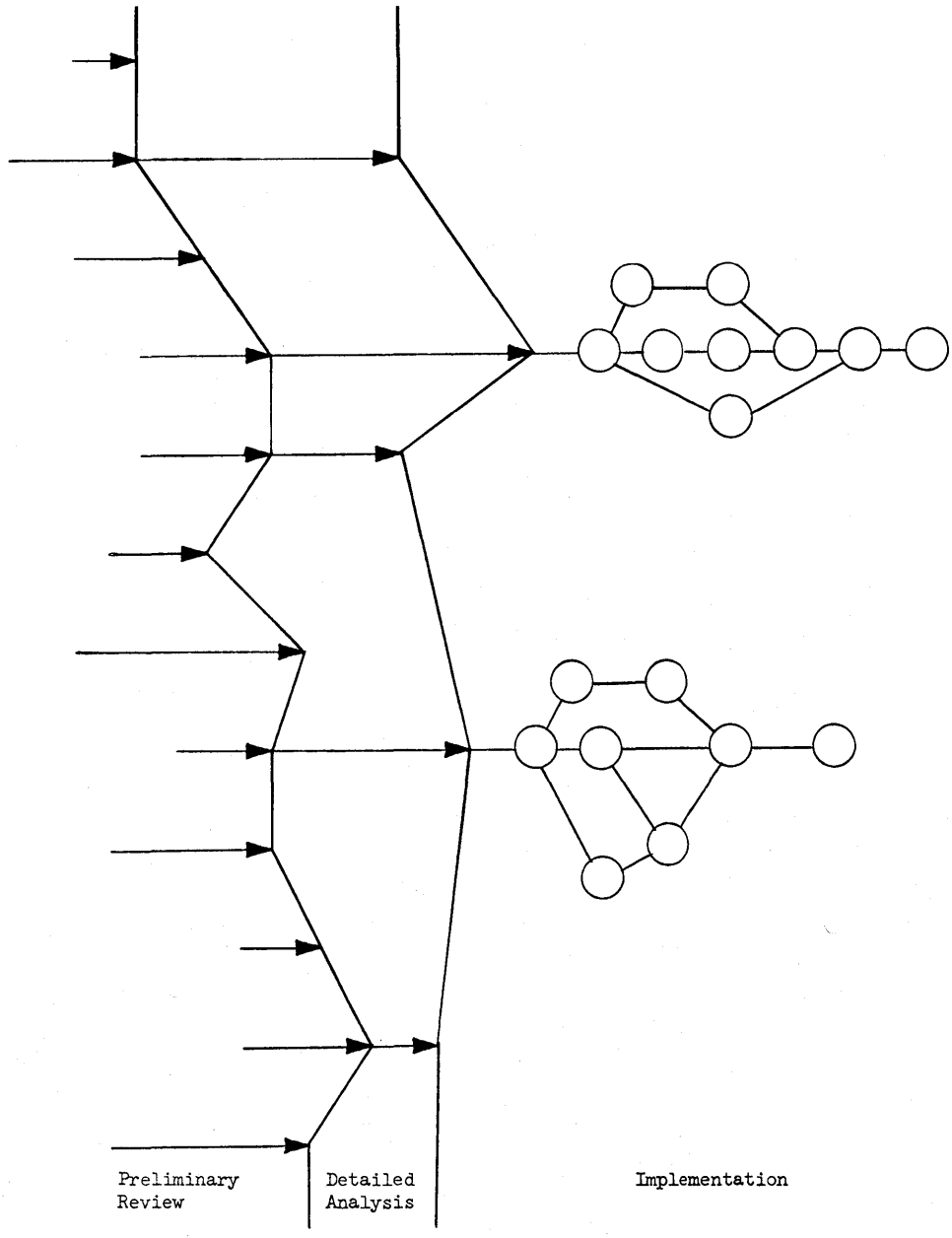



Figure 3. Systems development schematic — II.

## CONCLUSION

Although these comments about the design and manufacturing of a system have been presented in terms of customer services being developed by commercial banks, they have applicability to any organiation which has a variety of programming projects for which the assignment of priorities is of vital importance. One such activity would certainly be the development of programs by a computer manufacturer where there must be planning, scheduling, and control not only of the production of compilers, sorts, operating systems, and the like, but also of application oriented programs for customer use. Should the New Programming Language, COBOL, and FORTRAN compilers all be supported by the manufacturer? Or can the customers be served and satisfied with only one or two of these programs? Accurate answers to such questions as these may, for any company engaged in producing computer based systems for its customers, spell the difference between expnsion or decline in share of market.

# COMPUTER SYSTEMS DESIGN AND ANALYSIS THROUGH SIMULATION

George K. Hutchinson and John Norris Maguire
*Lockheed Missiles & Space Company*
*Lockheed Aircraft Corporation, Sunnyvale, California*

## INTRODUCTION

In March 1965, the Lockheed Missiles & Space Company's Computation Center installed a UNIVAC 1107 with two FH880 drums and three UNIVAC 1004 print, card punch and read systems, two of which are remote. This configuration is illustrated in Fig. 1. The Central Processing Unit was scheduled to be upgraded to an 1108 in October 1965. The UNIVAC EXEC II monitor was used for the operating control of this equipment. Under this monitor, the typical job enters the system through a 1004 card reader, queues at a section of one drum designated the Input/Output buffer, is processed by the main frame using the drums for scratch storage and systems routines, occupies I/O drum buffer area for storage of output, and is printed on the originating 1004 system. The monitor "steals" cycles from the main frame to control the flow of data between the 1004's and the drum I/O buffer. The insert in Fig. 1 shows the original storage allocation of the two FH880 drums which allocated 262,000 36-bit words to the I/O buffer.

If the I/O buffer is full when the program residing in the CPU wishes to place output in the buffer, the main frame waits for the monitor to relieve the condition by sending output to the 1004's. During this period, the fast, expensive CPU logs "green lite" time while waiting for the peripheral data transmission, which takes place at a relatively slow rate. On the other hand, if an adequate input queue is not maintained on the I/O buffer the CPU must wait in a similar manner for a 1004 which is reading cards. The determination of the proper I/O buffer size and peripheral capacity immediately became a major concern to the systems group. Other 1107 users were contacted in an attempt to utilize their experience in actual usage or in the method of solution. The information gathered in this manner was useful in respect to preventing or solving many operating problems, but was not directly applicable to the buffer size/peripheral capacity decisions because of the difference in workload characteristics. The configuration illustrated in Fig. 1 was chosen as an interim one on the basis of the known requirements, estimated requirements, and a forecast of the total workload.

More accurate information on the workload characteristics and requirements became available after the initial system was installed and a more specific plan for the conversion of programs from other computers was laid out. At this point, the system designers turned to simulation as the tool to aid in the decisions for the 1108 system configuration scheduled for October 1965. Coupled with more accurate workload information was the recent

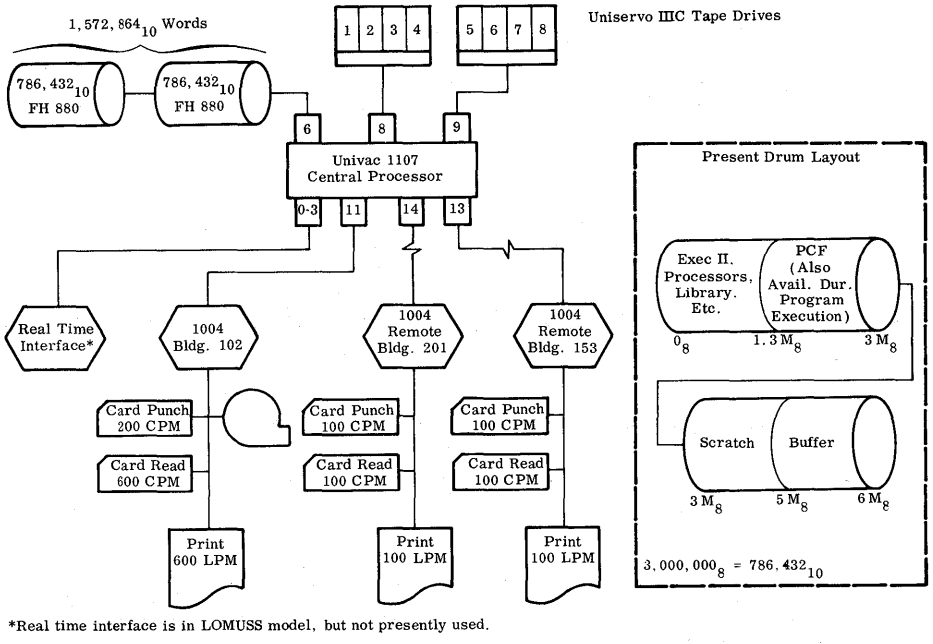



Figure 1. UNIVAC 1107 system configuration.

development of a generalized data processing model developed by Lockheed called the Lockheed Multiprocessor Simulation System (LOMUSS I). This system provided the vehicle for conducting dynamic simulations of proposed 1108 configurations.

In addition, it was felt that conducting simulations of the existing 1107 system would be useful for validation purposes, determining the system capacity, aiding in evaluating different data transmission systems for the remote stations, and scheduling second and third shift operations.

## STUDY OBJECTIVES

The primary objectives of the study were to determine which combination of I/O drum buffer size and peripheral devices and their locations would provide good service at a reasonable cost. The tradeoff between good service and system capacity is a management judgment, but simulations would indicate the relative changes in system performance measures (turnaround time, maximum queues, system cost, etc.) as a result of experimenting with different configurations. This problem was complicated by an increasing workload as programs were generated for and converted to the 1107. At the start of this simulation study, the 1107 was running less than 16 hours per day and, as of the middle of May, the 1107 was on a 24-hour-per-day schedule.

The possibility of the main frame having to wait because of I/O buffer storage saturation could be reduced to any desired level by the addition of sufficient equipment, i.e., increased drum capacity, additional peripherals (1004's), or a faster data transmission capability. Thus, the basic problem was to balance the expected costs of the main frame waiting for space on the I/O drum buffer against the costs of additional drums and/or peripheral capacity. The large number of variables, their complex relationships, and the changing characteristics of the workload made simulation a logical choice for problem analysis.

## DATA COLLECTION

The data collection phase of the study was undertaken with the study objectives as guidelines. Two catagories of data were collected, the data necessary

for the construction of the model of the 1107 configuration, and data to be used in establishing the validity of the model.

The 1107 monitor provided a major source of data as it summarized for each job processed the input volumes, main frame time, cards output, and the pages printed. The arrival patterns of the jobs into the system, together with the originating source was obtained from the log-in desk records. The routings that the jobs followed as they moved from processing step to processing step were identified and the equipment requirements at each step were determined. A typical routing is given below.

Frequently, the time at the various processing functions is collected and used as the basis for determining simulated processing times. The Lockheed Multiprocessor Simulation System (LOMUSS I) uses the volume of work to be processed and the processing rate of the specific equipment involved as the basis for processing time calculations during a simulation. Thus, if available, the number of cards input and the number of lines and cards output can be used rather than the time required to perform the function.

Summary data were collected for establishing the validity of the model on such system statistics as main frame utilization, average turnaround time (as a function of workload), average run time of jobs, system operating time, and daily beginning backlog of jobs. The use of these summary statistics in establishing the validity of the model will be discussed in a later section.

## MODEL FORMULATION

The Lockheed Multiprocessor Simulation System (LOMUS I) was used in developing the 1107 model. LOMUSS I is a 7,500 SIMSCRIPT instruction, dynamic, generalized simulation system developed for the evaluation of data processing systems. It views a data processing system as an augmented job shop. Jobs are described by the functions to be performed and the volume of work to be done. The job routings may vary from a linear list to a complex PERT-like network of functions. A variable set of requirements may be associated with each function of each routing and the requirements may each be described with a variable level of detail. All of the above information is described for LOMUSS I through data cards. Thus the construction of the 1107 model was actually the translation of the information gathered in the data collection into LOMUSS I input formats.

The remaining information that was required for the 1107 model was the specification of the model output reports. LOMUSS I provides the analyst, at his option, with a wide variety of reports on such items as "snapshots" of queue lengths and machine status, job processing reports, critical paths of each job, turnaround time reports and machine utilization reports. Snapshot reports are provided at an increment set by the analyst. Summary reports are available at any time interval specified by the analyst. With this reporting power available, care must be exercised that (1) one is not inundated with reports and (2) that the essential information is provided. In the 1107 model, interest centered on the utilization of the main frame and the I/O buffer area, so report options were exercised to provide this information.

The development effort required was one man for approximately one month to achieve a working LOMUSS model of the basic 1107 system. Simulation of one day's operation of the 1107 system required about 10 minutes on an IBM 7094 computer.

## MODEL VALIDATION

The following tests were applied to establish the validity of the model: a detailed trace of jobs through the system, a stability analysis, a comparison of summary statistics from the model with the same variables as collected in the data study and, finally, a test of the ability of the model to predict.

Initially, a small number of jobs were submitted to the model in order that a complete trace of their progress through the system could be printed out for detailed analysis. The following consistency checks were made (1) the job's path through the system with its routing, (2) the elapsed time at each function with the volume-rate specified time, (3) assignment of functions to equipment, (4) job processing order when queues developed, and (5) the matching of equipment capabilities with the function requirements.

When the above validity checks were completed, the 1107 model was loaded with the number of jobs (160) that the actual system was running as established by the data collection. The model indicated that this level load could be handled on a 2-shift (16-hour) basis, which corresponded with the actual operation. The load was then increased in

steps, observing the effects on the system at each step. The plots in Fig. 2 were generated at this time. Illustrated in Fig. 2 is the arrival rate pattern of jobs into the system and number of jobs in the I/O drums queue plotted as a function of (1) the time of day and (2) the aggregate workload rate. A detailed analysis of the simulation output provided information on the "mix" (input, waiting to be punched or printed) of the I/O queue which was manually translated to "words of drum storage required." The model predicted that the quarter million words of I/O drum storage would be saturated

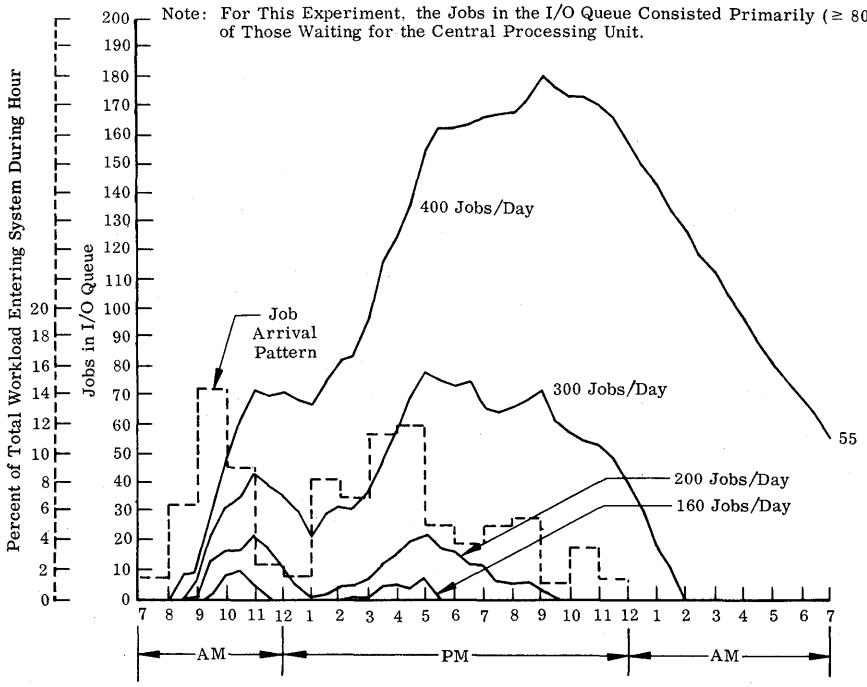



Figure 2. Analysis of UNIVAC 1107 workload rates and drum input/output queues.

in the late morning and late afternoon when the workload rate exceeded 200 jobs per day. This created surprise among some systems and operations personnel who had felt that the I/O buffer was adequate for the 1107 configuration but confirmed the intuitive judgment of others. During the month of April the workload gradually built up until April 8th when a record 215 jobs were processed. On that day, the I/O drum buffer reached a saturation point for the first time. From a scientific point of view, this was only one of many factors in the validation procedure; but from a practical point of view this accurate drum saturation prediction proved to be very significant for purposes of implementing model results in the actual system, which was the responsibility of operations and systems personnel. More system status data (words being utilized on the I/O drum queue, jobs queued at card readers, etc.) were sampled frequently during the following two weeks. This information further validated the model and lent credence to results obtained with higher workloads rates fed into the model through parameter variation.

During the initial model runs with the actual 1107 workload, the 1107 utilization reported by the model was 54 percent. The actual operating data collected showed 56 percent utilization. The average turnaround time of jobs in the model as a function of workload level was compared with turnaround times obtained from the actual operating system. These data are illustrated in Fig. 3. The differences between the actual and model turnaround times are relatively constant and represent the manual handling times exterior to the hardware of the actual system and the domain of the model. The time between clocking in a job and actually inserting it into the card reader could have been accounted for in the model, but the actual times were found to be about constant and not significant to the problems for which the model was being utilized.

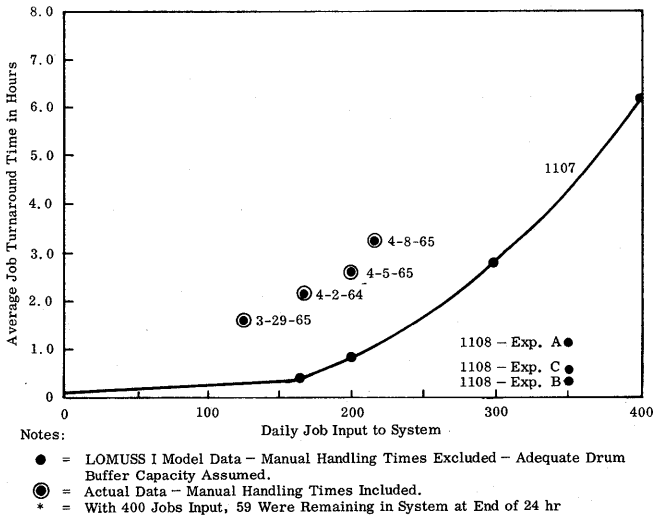



Figure 3. Analysis of UNIVAC 1107/1108 turnaround time as function of workload (manual handling times exterior to hardware system excluded for LOMUSS I simulation).

## SYSTEMS ANALYSIS

The simulations were conducted primarily for determining an 1108 system configuration for October 1965, but information obtained from the 1107 simulations in April was useful to the then operating system.

With the workload characteristics remaining approximately constant, the model showed that a 3-shift operation would be required when the workload rate reached approximately 250 jobs per day. This can be seen in Fig. 2 when a linear interpolation is made between the I/O queue plots of 200 and 300 jobs per day. That is, a queue of jobs will still be on the drum after midnight, which is the end of the second shift, if the workload rate exceeds 250 jobs per day.

There were some manual procedures that were immediately implemented to minimize the drum saturation problem, namely, when the operator detected a saturation condition he could stop the input flow by stopping one or more card readers and/or he could dump some of the drum output queue onto tape for later processing during a low workload period (which would increase the turnaround time). The model made a contribution by pointing up the importance of the console operator to system efficiency and by so doing, helped accelerate work on improved manual procedures and automatic system status indicators (for example, modification of the executive routine to give an on-line console printer message when the I/O drum buffer became saturated).

Another pertinent piece of information was the 350-job-per-day 1107 system capacity predicted by the model as illustrated in Fig. 2. This provided a useful guideline to establishing system programmer support levels and program conversion (from other systems to the UNIVAC system) schedules prior to installation of the faster 1108 CPU system.

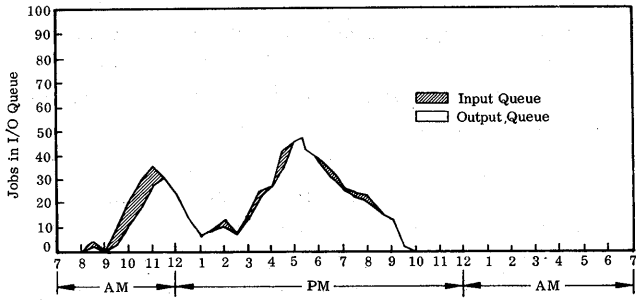
The next step in designing and conducting the simulation experiments was to incorporate some tentative system hardware plans into the model. These plans are summarized as follows:

- Installation of a high-speed data link between the central system and a high-use remote station.
- Installation of an additional on-site UNIVAC 1004 print, card read and punch system.
- Replacing the 1107 CPU with an 1108 CPU.

Three experiments were conducted which recognized the above system changes combined with a forecasted change in the characteristics and level of the 1108 workload. The output from these experiments was the primary information used for determining the 1108 system configuration for October 1965.

Some output from experiment A is illustrated in Fig. 4, where the original 1107 model configuration was run with (1) a 350-job-per-day input which was the maximum forecasted level of work through 1965 and (2) the faster 1108 CPU in place of the 1107. Three system performance measures were significantly affected in experiment A:

- The average job turnaround time for 350 jobs dropped from over 4 hours to about 1 hour.



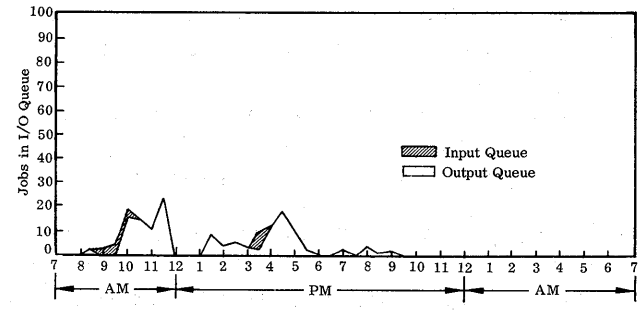

Notes:

1. Average Turnaround Time = 1.02 Hours (Manual Handling Times Exterior to Hardware System Excluded.)
2. 1108 CPU Utilization = 49% (Two Shift Operation).
3. Maximum Requirement for I/O Drum Buffer Storage Occurs at 5:10 PM and Equals 1,860,000 Words.

Figure 4. Experiment A — present 1107 system with 1108 CPU and a 350-job-per-day workload.

- System throughput increased substantially as the 350 jobs were processed in only 2 shifts instead of 3.
- The mix of the I/O queue changed from primarily input to primarily output and the peak I/O drum buffer requirement rose to almost 2 million words.

Similar output from experiment B is illustrated in Fig. 5 where experiment A was repeated except that a faster data transmission facility for a high-




Notes:

1. Average Turnaround Time = 0.36 Hours (Manual Handling Times Exterior to Hardware System Excluded.)
2. Maximum Requirement for I/O Drum Buffer Storage Occurs at 11:30 AM and Equals 900,000 Words.

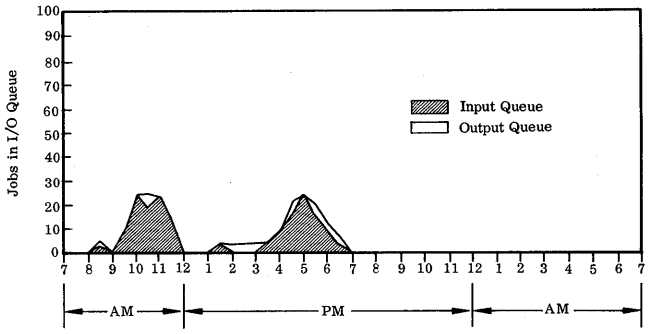
Figure 5. Experiment B — 1108 CPU with building-153 COAX and additional 1004 system in building 102.

use remote station was added to the model along with an additional on-line, on-site UNIVAC 1004 Read, Punch, and Print System.

Two system performance measures were significantly affected in experiment B:

- The average job turnaround time dropped from 1 hour to 0.36 hour.
- The peak I/O drum buffer requirement dropped from almost 2 million to about 900,000 words.

The output from experiment C is illustrated in Fig. 6 where experiment B was repeated except that the mean run time of the jobs was raised to 6 min-




Notes:

1. Average Turnaround Time = 0.45 Hours (Manual Handling Times Exterior to Hardware System Excluded.)
2. Maximum Requirement for I/O Drum Buffer Storage Occurs at 10:30 AM and Equals 354,000 Words.

Figure 6. Experiment C — same as Experiment B but with an average CPU time of 6 minutes instead of 2.54 minutes (as measured on 1107).

utes to more than account for an expected increase in the average run time characteristic of the workload.

The significant result from experiment C was:

- The peak I/O drum buffer requirement was reduced from 900,000 to 354,000 words.

## CONCLUSION

The quantitative information provided by experiments A, B, and C was used to revise the tentative 1108 system hardware plan. The significant difference between the old and new plan for the 1108 configuration was the change from all rapid access, expensive UNIVAC 432 drums to a reduced 432 drum capacity and an increased (from ¼ to ¾ million words) I/O drum buffer area using the slower and less expensive 880 drums. The simulations showed that the forecasted 1108 workload would initially be primarily constrained by peripheral data transfer rates and therefore the faster 432 drums would not alleviate this problem. In addition, the peak I/O queues indicated that a larger I/O drum buffer area would provide improved system performance (e.g., reduced turnaround time) and minimize manual intervention.

The LOMUSS model of the Lockheed UNIVAC

on-line, time-sharing, remote terminal system simulated two critical periods (spring and fall of 1965) and provided information upon which the design of the 108 system configuration was based. An effort is continuing which will monitor the level and characteristics of the workload, equipment utilization, turnaround time, etc., for further model validation. The information derived from this continuing effort, coupled with up to date workload forecasts, will determine whether further model experimentation will be conducted for future system decisions.

Lockheed has been constructing general and specific models of computer systems for five years using a variety of simulation languages. As illustrated by the implementation of the above simulation study, the development of LOMUSS has significantly assisted in establishing simulation as a design and analysis tool of the Lockheed Missiles & Space Company.

# BASIC CONCEPTS FOR PLANNING AN ELECTRONIC DATA PROCESSING SYSTEM

A. F. Moravec
*General Dynamics*
*Fort Worth, Texas*

## INTRODUCTION

The achievement of a reliable management information feedback system is dependent upon meaningful and well-defined electronic data processing objectives and basic concepts. The purpose of this paper is to attempt to define those basic concepts which can be used as a foundation for planning advanced electronic data processing (EDP) systems.

Developments in digital transmission, the availability of faster bulk storage devices and the use of man/machine interface devices, such as displays and interrogation consoles, have stimulated a new kind of data processing. In this processing, information is entered into the system as it is generated. Outputs are requested as they are required. These inputs and outputs are occasioned by external stimuli—man or machine—to which the computer responds.

For the future then a basic data processing philosophy is required to match these recent developments.

Two basic divergent data processing concepts for the future are being discussed in much of the electronic data processing literature today. The remainder of this paper presents each concept.

## TOTAL SYSTEMS APPROACH VS SINGLE INFORMATION FLOW PHILOSOPHY

The total Systems Approach and the Single Information Flow Philosophy are the two fundamental concepts which analysts have been alluding to in many of their discussions, but to date have not defined these concepts in a manner which should and can facilitate the designing of an advanced electronic data processing system.

Figure 1 asks the first basic question, "Which *is* the driving gear?" That is, is the system to be designed based upon the total systems concept or the single information flow concept. And Fig. 1 also asks the question, are we user-oriented or fundamental information-oriented as we visualize our EDP systems?

Before we attempt to answer these questions, let's first define the two concepts.

### *Definitions*

*The Total Systems Approach.* This approach represents the final goal of computer installations of many companies today. It encompasses present data processing operations and thinking, and its most

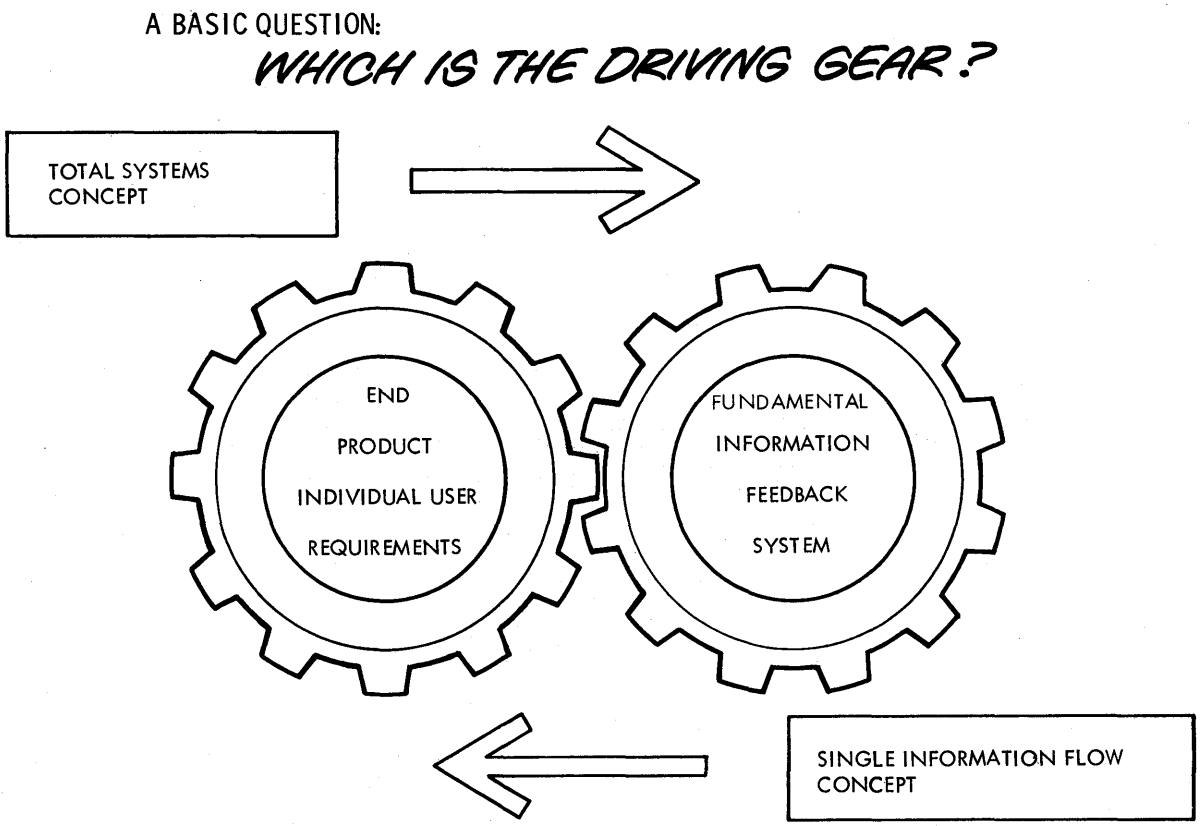



Figure 1 FW/65/374/019-223A

sophisticated ultimate development represents a future attainment of the complete "total systems concept."

The Total Systems approach has evolved from techniques such as "Batch Systems" and "Integrated Systems."

In this approach, major functions (operations), e.g., inventory control, procurement, payroll, reports, etc., are usually considered separate subsystems. This total systems approach recommends treating these subsystems on an integrated (compatible) basis—for example, providing the ability of the payroll subsystem to run with the labor distribution subsystem or perhaps the inventory control subsystem running with the purchase order subsystem.

Ideally, through evolutionary reprogramming and redesigning, where required, there evolves a single executive control subsystem which monitors subsystem integration, produces desired reports, controls run sequence and operations and, to some degree, will automatically change programs as required.

Figure 2 (the upper portion) reflects this kind of subsystems monitor.

Information processing requires that these subsystems be processed in an ordered sequence regardless of actvity and that information be retrieved after each subsystem is run. This is indicated in Fig. 3 (upper portion).

*Single Information Flow Philosophy.*[1] The new computer data processing of the future will be concerned with this philosophy.

In this approach it is recognized that all "essential" information is completely *interdependent.* The attempt in this concept is to enter only once in its history, a single piece of information to be processed, and from that time on it is available to serve all data processing requirements until its usefullness has been exhausted.

This approach has sometimes been called the "single transaction processing" or complete "single record" concept and in some literature it is known as "Total Information System." Regardless of name, the key to making this concept workable is under-

DEFINITIONS

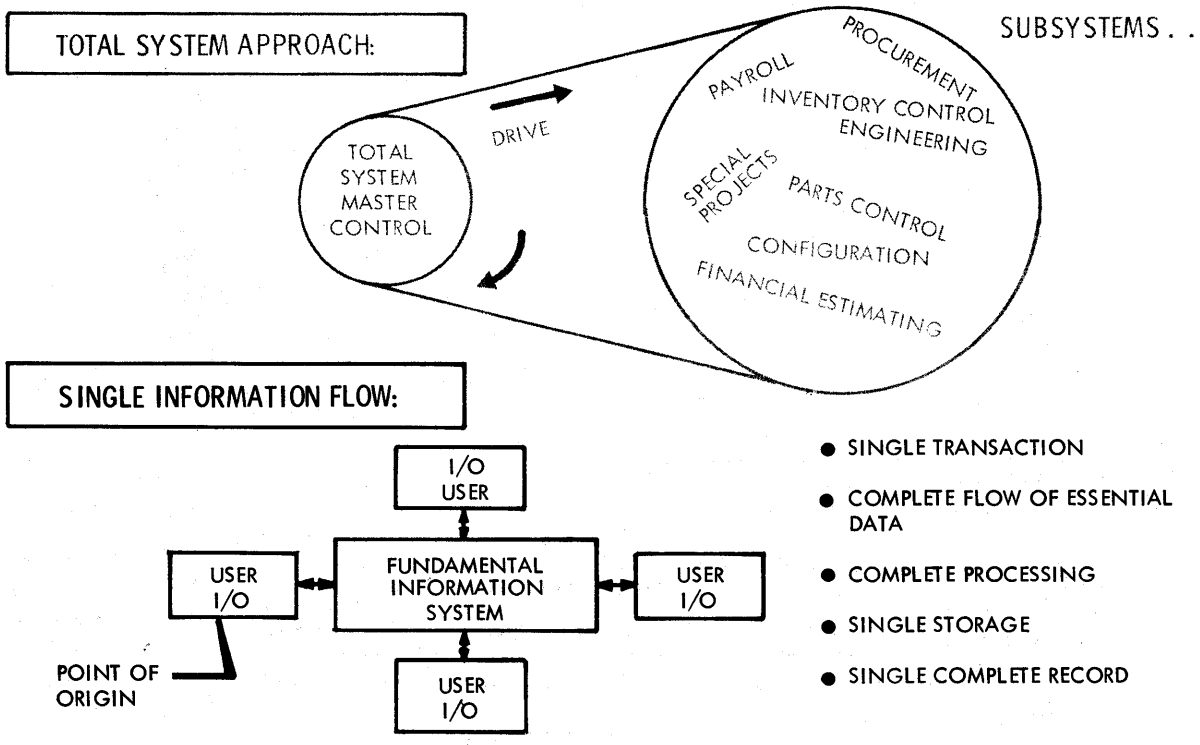



Figure 2 FW/65/374/019-224

standing the following rules: (a) "Information must be essential to the conduct of business," and (b) "It must be part of a single flow of information essential to the operation of the business."

Statement (b) implies that much information being processed in present-day computer operations is not "essential." These nonessential types of information include "protective" type reports (e.g., auditable fax-cards, special audit runs), multitudinous repetitions and overlapping of the same basic data records (e.g., identical requirements data being held on the requirements file, then on the Inventory Status file, and also on the Purchase Order Status file, etc.), which are maintained on an unrelated basis to meet needs long lost through the evolution of time, and special requests the need of which has long ago disappeared.

This concept is likened to the efficient one-man storekeeper who came quite close to ultimate real-time random information handling. The cans on the shelf and a few pencil marks gave him both inventory and purchasing information; the book next to the cash drawer provided accounts receivable, credit and customer information; the bank book plus cash drawer gave him his cash balance; while accounts payable were visible on the nail on which he spindled the bills. In the drive for seeming efficiency, computer organizations began to specialize and to batch-process information, which, of course, runs counter to this one-man type operation.

Figure 2 (lower portion) outlines the basic ingredients and the fundamental information system.

Ideally, under the Single Information Flow Philosophy, a piece of information is retained in only one place and is available for all necessary uses. For example, at the time Engineering releases a part with the needed material requirements, inventory status and "on order" conditions (including procurement) are immediately updated through a complete information flow and processing of transactions resulting in the proper action (e.g., buy, issue, manufacture, etc.) taking place as needed. All necessary status reports both units and dollars are then taken from this single and common source data. It

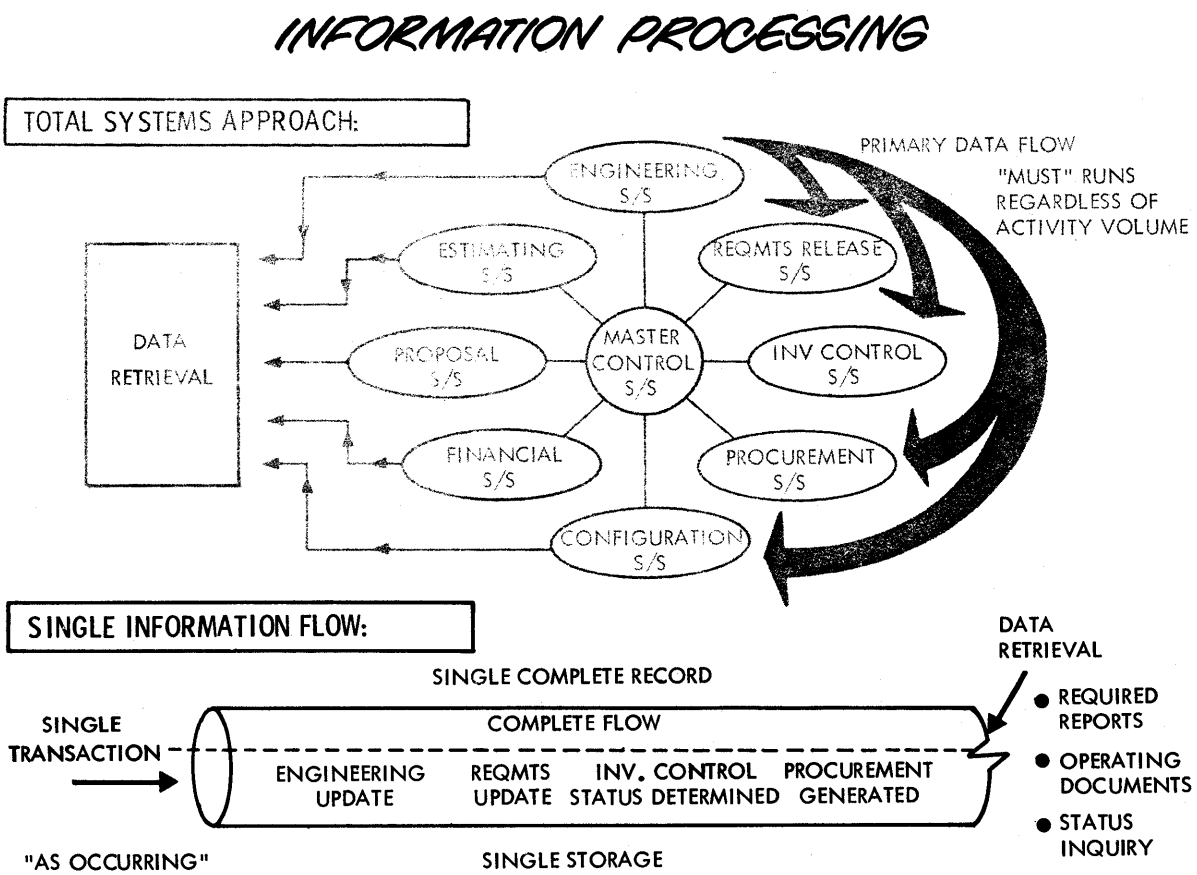



Figure 3

is like taking a picture of a condition without double exposure or varying time. The lower portion of Fig. 3 indicates the single transaction and complete processing technique.

Organizationally, the systems design technique varies according to the basic approach used. As Fig. 4 indicates, there is normally a specific systems engineer and programmer assigned to design and maintain each subsystem in the *Total Systems Approach* whereas the *Single Information Flow* concept provides for preparing the basic specfications of all categories of "essential" information before the system is designed. It is after this point that the specific systems engineer, programmer and communication specialist get together and start designing the system.

## BASIC ENVIRONMENT

The two approaches differ in their basic environments. The total systems concept is output-oriented.[2] Files and data processing procedures are established to provide end products that meet specific user requirements. Information orientation is by particular functions or departments. Applications are specialized to meet particular needs. (See Fig. 5 upper portion.) Processing is predominantly of the batch type. Data are collected over a period of time for processing during a particular machine run. The same information is read and reread into the computer following various sorts and merges with other data. Files are run sequentially regardless of the amount of activity.

Information is oriented differently between the two approaches. In the upper section of Fig. 6 it can be seen that subsystems are but individual files of a total file and retrieval of information takes place on an individual file basis.

As the number of subsystems going "on-the-air" increases (Fig. 7), it will generate additional need for large computers and attendent peripheral equipments. Because much of the data processing operation is conducted off-line, there is high use of peri-

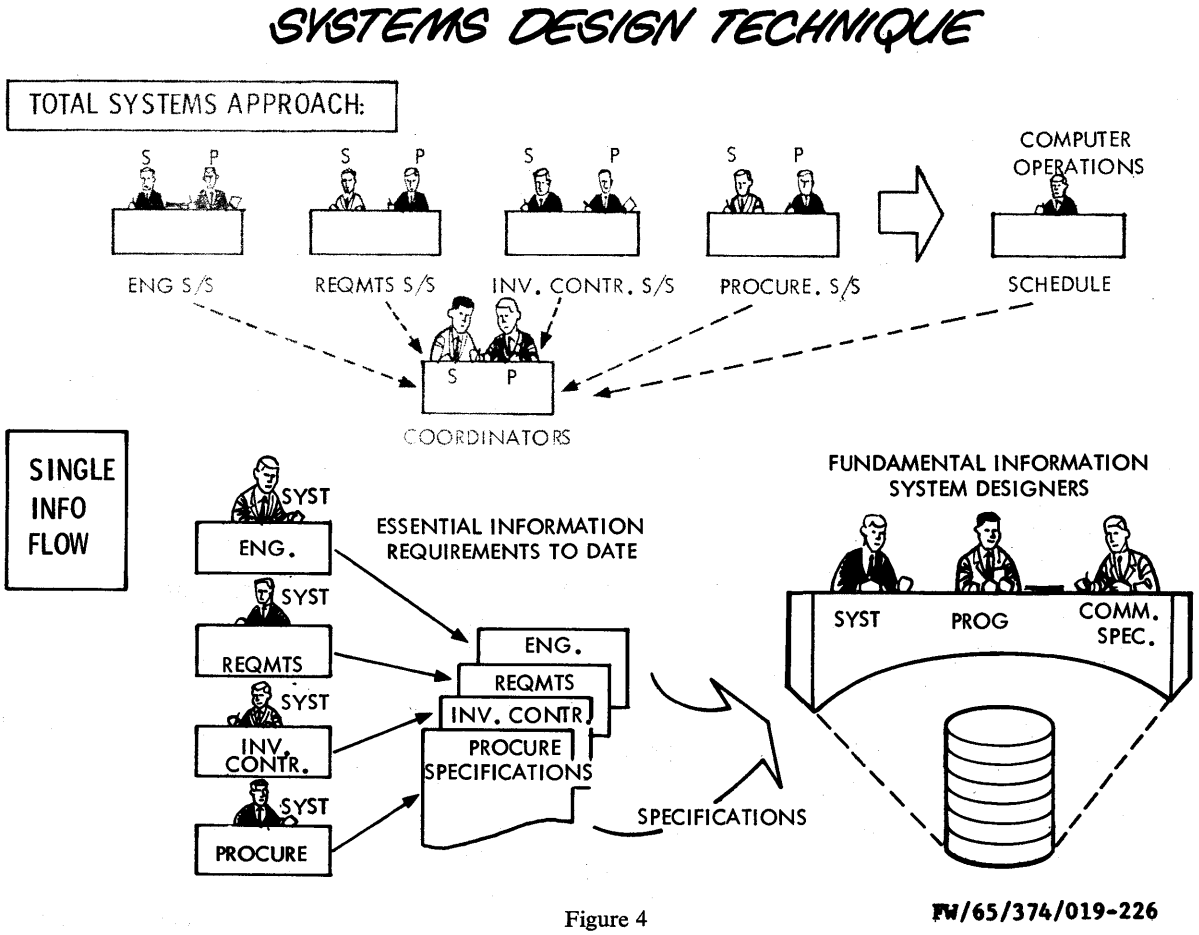



Figure 4

pheral equipment. Control and audit of data also take place off-line; manual calculations and various audit comparisons may be involved.

The future trend of the total systems approach may well be toward multicomputer operations. It may, indeed, foster a decentralized data processing environment, in which the user processes his own data on less sophisticated peripheral computing equipment while complex data processing remains with the centralized main frame computer. (See Fig. 8.) As the number of computers and the number of users demanding to process their own data increase, there will be pressure from the users for current data under their own control.

The single information flow concept, on the other hand, is input-oriented.[2] The system is organized so that essential data are inserted into a common reservoir through point-of-origin input/output devices. User requirements are then satisfied from this reservoir of fundamental data about transactions.

Thus, the single information flow concept is characterized by random entry of data, direct access to data in the system, and complete real-time processing. (As soon as a transaction occurs, all the necessary and related records are updated and posted.) This method of single-transaction processing provides fast response, a high degree of reliability, and an easily expansible system.

Information orientation, instead of being toward individual users, fits overall company requirements. It is likely to cut across departmental and functional lines.

Planning objectives or operational targets are associated with "fundamental" record information. Exceptions are noted at the time of processing.

This approach will easily facilitate the use of "time-sharing" by a number of users and the use of "implicit programming" techniques (direct decision making). (The term "time-sharing" means that user groups can share time in common on the company's centralized business computer.) In addition to intradivisional user-group time-sharing on the central computer, interdivisional time-sharing operations

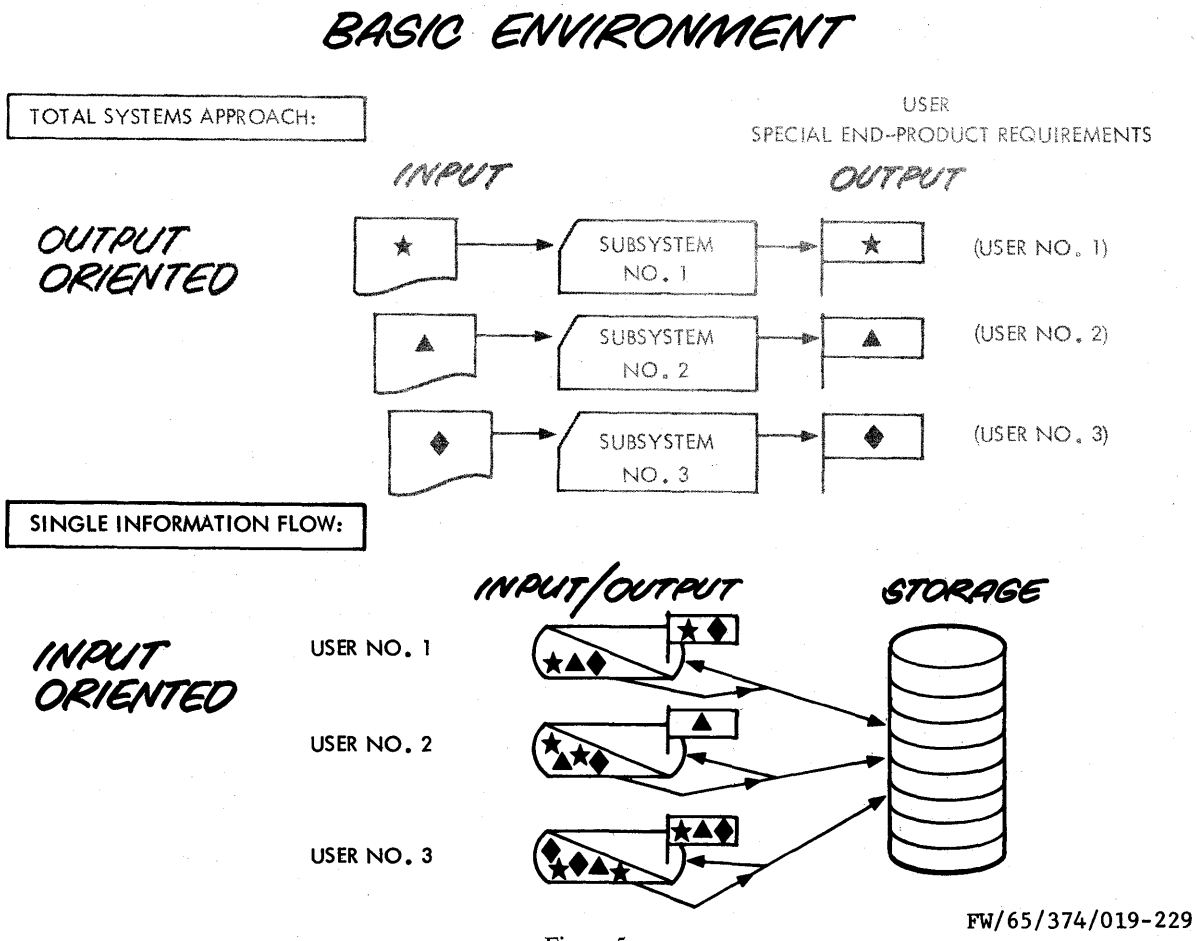



Figure 5

can be established on the same basis. (See Fig. 8.)

Time-sharing operations will probably result in a trend toward centralized computing facilities and decentralized input/output equipment for insertion and retrieval of information. This will permit development of man/machine simulation techniques, which will enhance managers' systems understanding, broaden their training, and eventually facilitate direct decision making.

The two basic data processing concepts also involve widely differing equipment concepts. The choice between them will have a major impact on the choice of equipment throughout the data processing system.

## EQUIPMENT CONCEPT

Adoption of the total systems concept imposes a need for high speed of operation to compensate for redundance of data and for long subsystem computer runs. A large amount of high-speed storage will be required. Sophisticated peripheral equipment—almost with the capability of small computers—will be needed to reduce the load on the central main frame computers and solve the "input/output constraint" problem.

Each individual user's file will have to be stored separately—on disks or drum—and accessed by name only through a file director. If time-sharing techniques are to be used under this concept, great care must be taken to protect the user programs from one another in order to preserve their integrity and independence. "Crosstalk" between users will be tightly restricted. (See Fig. 9.)

Equipment for use under the single information flow concept, on the other hand, will need to possess on-line real-time capabilities. Storage will also have to be of large capacity, but it need not be high-speed. Much of the equipment emphasis will be on communication systems to connect users with the central processor. Instead of satellite computers,

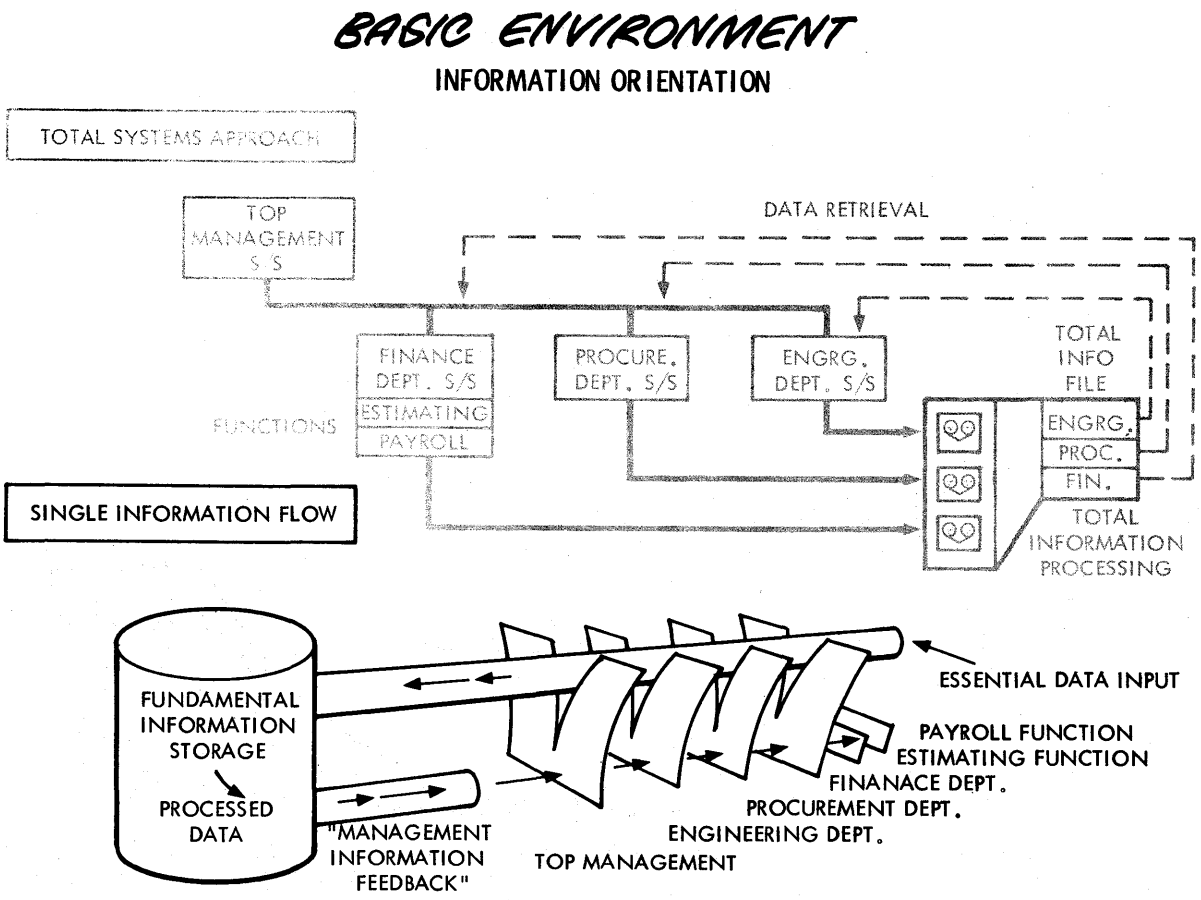



Figure 6 FW/65/374/019-232

users will want point-of-origin input/output devices.[3]

With the use of communication equipment appropriate for time-sharing, communication among users will be encouraged. Crosstalk will be the rule rather than the exception.

Individual user files will not be maintained. Instead, there will be a single record file accessible to all users. Nonessential data and data not needed to update records will be processed directly through crosstalk between user point-of-origin devices. Such data could be documented if need be through an off-line printer.

Appropriate software techniques will have to be designed for information insertion and retrieval. Implicit (man/machine response) programming will be developed.

## ADVANTAGES

Each of these concepts, of course, has both advantages and disadvantages. The chief advantage of the total systems concept is that it offers a relatively simple transition from existing systems. Mechanization can be accomplished piecemeal. Subsystems can be developed independently as they are required or as systems workloads and resources permit. Interdependence among subsystems is limited almost entirely to the need for agreeing upon and coordinating standard interface formats.

Thus, the total systems concept permits step-by-step phased achievement of automation. As each subsystem is mechanized, valuable experience is gained that can be applied to the next one (see Fig. 10.)

The total systems concept has the additional advantage of lending itself to "productionization," meaning that set times can be set aside for and assigned to each subsystem. (See Fig. 11.) Much processing of data can be accomplished off-line or on peripheral equipment, thereby leaving the main frame computer free for other uses.

The single information flow concept; however, offers a number of control advantages. Engineering,

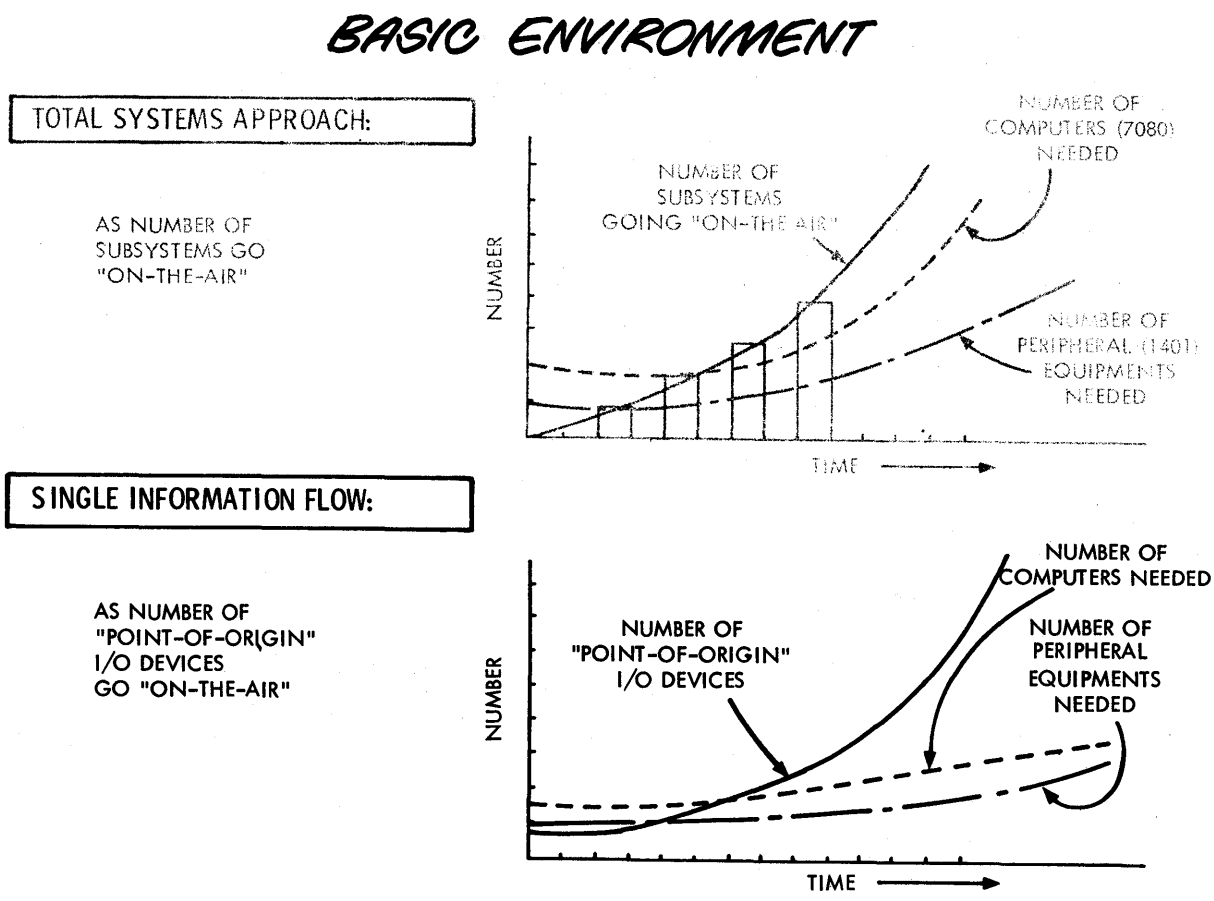



Figure 7 FW/65/374/019-230

manufacturing, accounting, purchasing, material, and other departments all use the same data rather than different iterations of the same data. Since data are transported only once, they need only a single edit. Thus, it becomes economical for employees to exercise greater care in entering information into the system.

Real-time processing permits current comparisons with planned objectives and exception reporting of out-of-tolerance situations. The centralization of operation characteristics of the single information flow concept makes control easier—and also make it easier to determine data processing costs. Systems and programming revisions can be handled more rapidly by substitution of a computer program at a central location than at multiple locations with the inherent transmission distortions.

The single information flow concept also has the advantage of facilitating adaptive systems design. A system designed to make internally generated adjustments from source input is likely to be more responsive to additional requirements placed on it and less likely to require a complete overhaul from time to time.

## DISADVANTAGES

The total information systems concept presents problems of equipment efficiency and timeliness of data. Data handling by separate groups, often handling like data, fosters redundant data processing. Duplicate data storage causes inefficiencies. As subsystems feed data to each other, long computer runs result. Data are only as current as the frequency and length of running cycles permit. (See Fig. 12.)

Not only is there duplication of data, but it is difficult to reconcile records since files are altered, updated, and organized at different times in different subsystems. Since the same kind of data is stored in several subsystems, management reports will reflect the status of the data in the subsystem from which it was taken. Because data and transactions are intertwined among various subsystems,

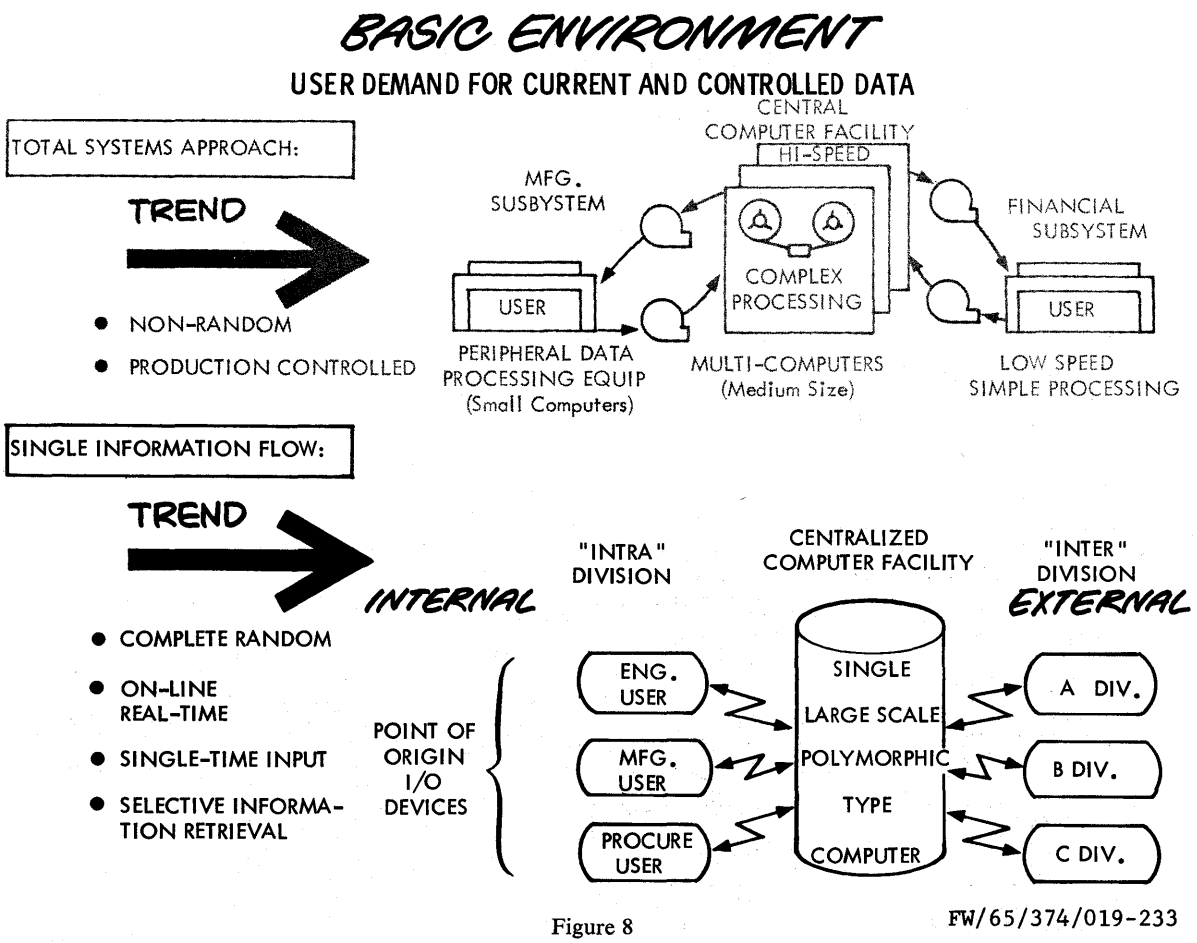



Figure 8 FW/65/374/019-233

costs of data handling and processing are difficult to track down.

The total systems approach may fail to allow adequately for systems and data interdependency and the ripple effect of data. For example, the inventory control subsystem needs to have the on-order status data from the purchase order subsystem. The purchase order status subsystem needs to have total requirements data from the inventory control subsystem, which in turn should have current total requirements from the requirement subsystem.

Since the subsystems are, for the most part, designed separately by different individuals, different methods and principles are applied. This problem is aggravated, of course, by different user requirement of the same data.

As the number of systems increases, efficient scheduling of computer and supporting tabulating equipment become difficult. In some cases a second or a larger computer may be ordered in order to avoid redesigning the system.

The disadvantages of the single information flow concept, on the other hand, lie more in the demands it makes upon systems and data processing personnel than in its inherent deficiencies. Both systems designers and programmers will require training to assimilate new concepts. Systems designers will need communications knowledge and experience in addition to EDP knowledge. Programmers will need training in the technical applications of random and direct access operations.

Reorientation of operations will require complex advance planning. User needs, equipment requirements, and programming needs will have to be analyzed. A fundamental information system for the entire company will have to be designed before this concept can be installed. Each step of the conversion will have to be planned and scheduled.

## IMPACT

If the total systems concept is adopted as the cornerstone of planning, the following action is necessary:

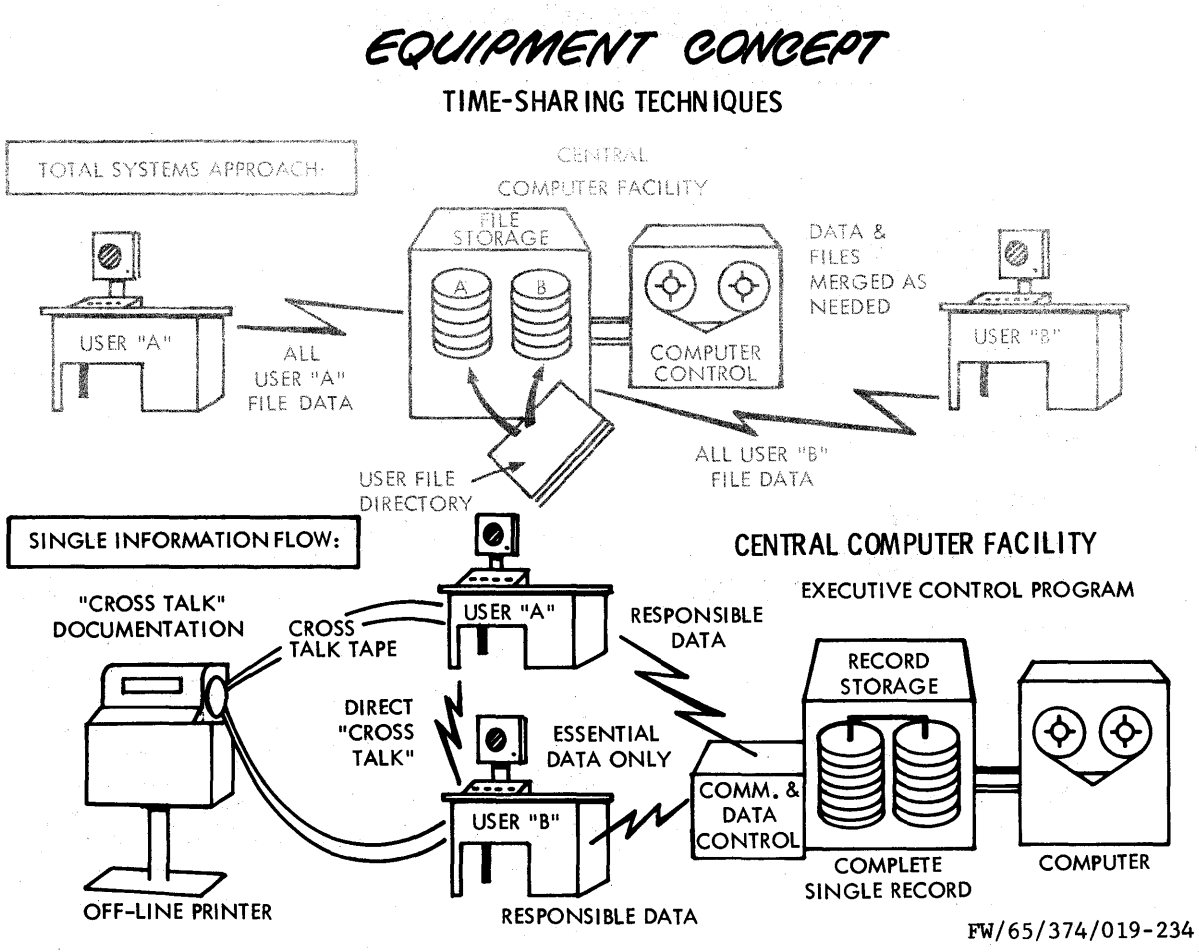



Figure 9

1. Although this concept represents the ultimate sophistication of present-day data processing methods rather than a totally new approach, there remains the problem of integrating the various subsystems into a total information system. This requires proper data definition so that the system will be responsible to the needs of various levels of management. (See Fig. 13.)
2. The shortcomings of present operations must be analyzed in the light of the total systems objective.
3. An estimate of the total anticipated scope of operations must be made in order to establish realistic boundaries for resource planning.

If, instead, the single information flow concept is selected as the basic information systems concept, each of the following steps will be necessary:

1. The conversion from the old to the new information system must be planned. A step-by-step time-phased action schedule should be prepare.
2. If the transition is to be smooth, reorientation and training programs must be given for management, user groups, system designers, and programmers.
3. Both management and operating personnel will have to make extra efforts to make sure they understand the communication aspects of the new concept.

## SYSTEMS ENGINEERING

As is probably obvious from the foregoing, I favor the single information flow concept. It seems to me that this is the best approach if a company really wants an information system that will enable management realistically to weigh the effects of all

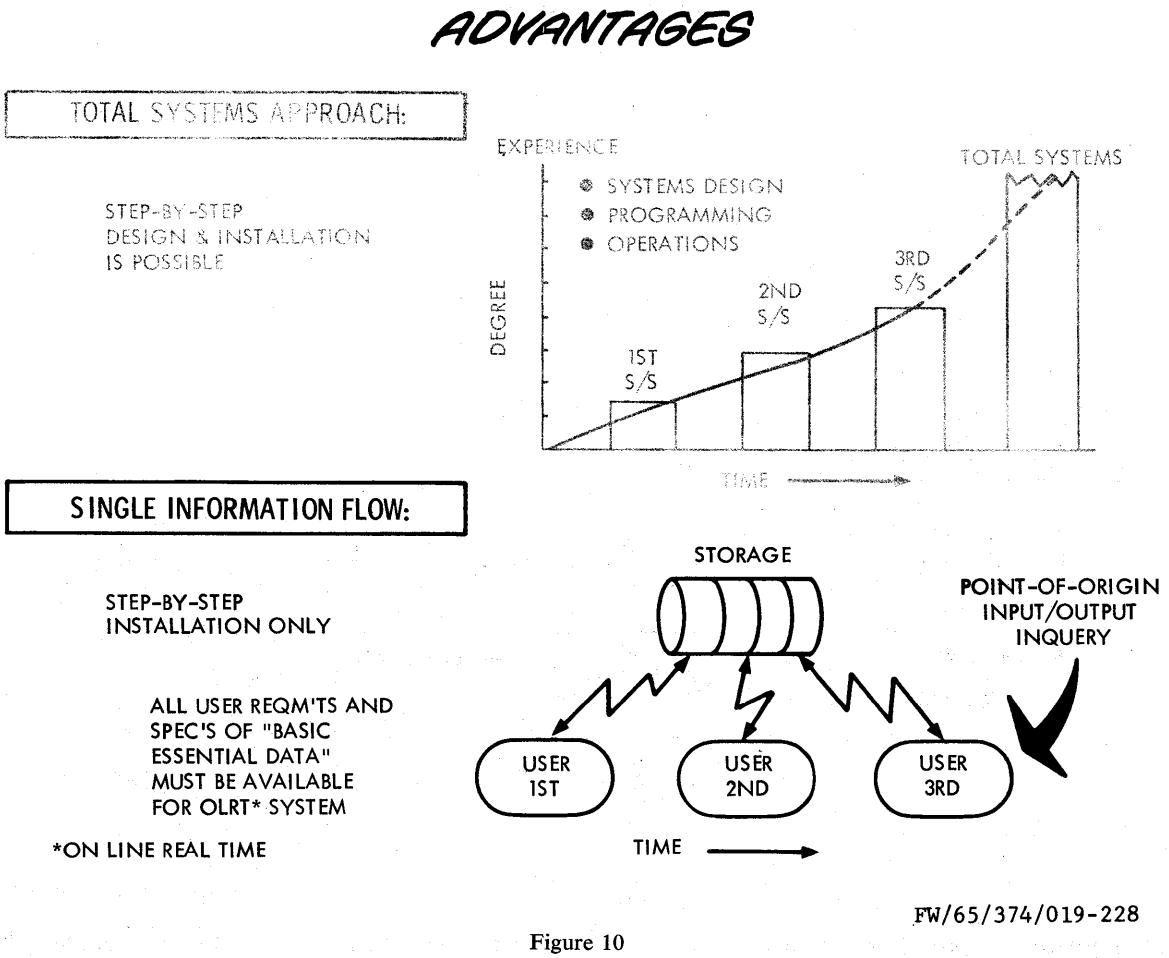



Figure 10

business parameters on current and future operations and thus to optimize decisions. (See Fig. 14.) With such a system not only can corporate activities be analyzed and synthesized for management's review and tactical appraisal today, but ultimately simulation techniques can be used as predicators of the effects of long-range planning. This will allow management to determine the tactical decisions that should be made now to accomplish the strategic planning so necessary for success tomorrow.

The scientific concept by which the fundamental information system is best designed and implemented is known as business systems engineering. Business systems engineering may be defined as a formal awareness of the interactions among the various parts of a business complex. Until recently much of management education and practice dealt only with functional components of business—accounting, production, marketing, finance, engineering, and the like—that were taught and practiced as if they were unrelated subjects.

Now attitudes have changed, and there is growing awareness that interactions and interdependencies among components of the system are more important than the components themselves. This awareness is the keystone of fundamental information systems design and of the single information flow concept of data processing.

## LIMITATIONS OF PRESENT CONCEPT

The present concept of business systems engineering has evolved over a number of years. In the early years of computer technology the components (subsystem) approach prevailed. At that time an integrated business information system was thought to exist if a business transaction element was introduced into the system and perpetuated in the system with a minimum of manual intervention. The assumption was that mechanizing data and providing it to operating groups would, per se, result in benefit to the company.

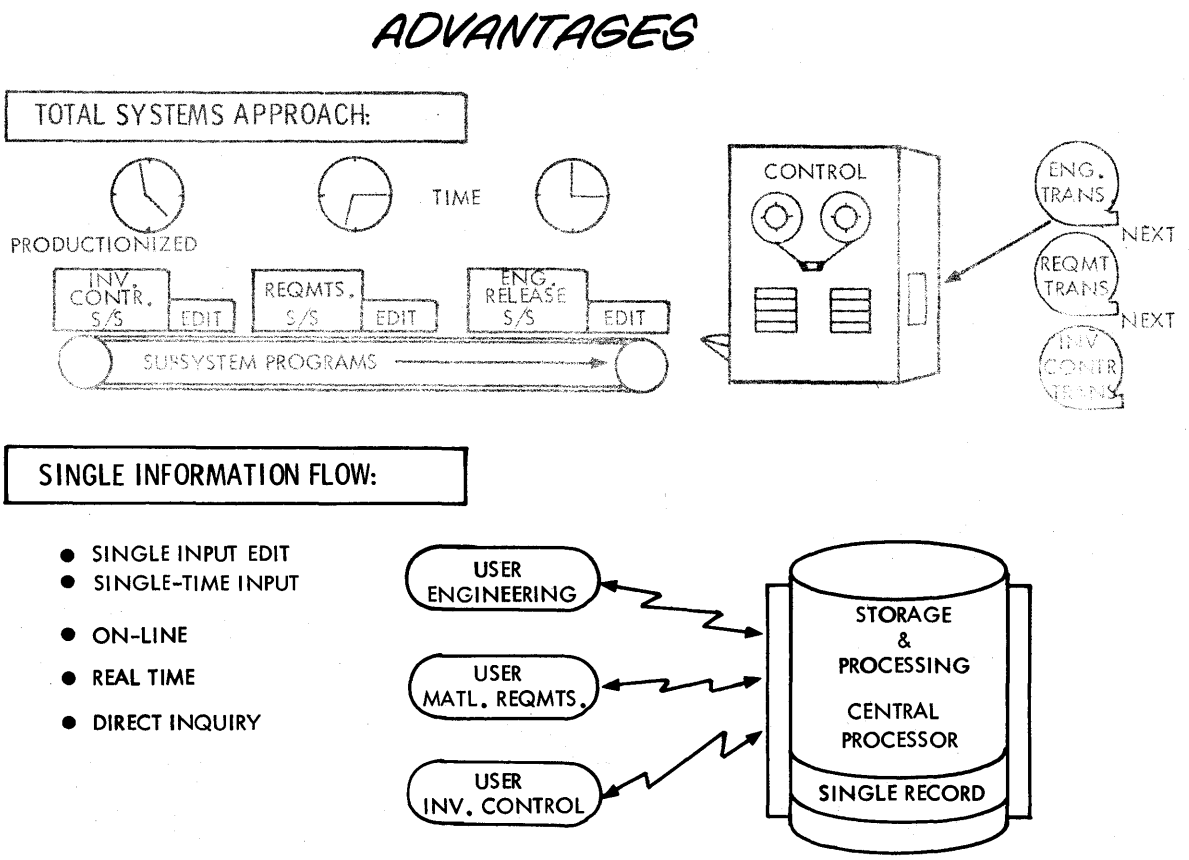



Figure 11 FW/65/374/019-227

An alternative approach envisioned good business systems design as the mechanization of data for specific random jobs as dictated by the needs of operating groups, with reliance on the assumed economies involved in mechanized data production. Both these alternatives, of course, represent piecemeal static systems because they inherently lack the flexibility of systems design necessary to coordinate the overall business process.

## CHANGE IN APPROACH NEEDED

The need for a change in approach has become obvious. The interdependence approach owes some of its impetus to the growing emphasis on long-range planning. In the development of multidimensional master plans there has been a tendency to ignore traditional departmental lines in favor of broad company functions and processes, analyzed in terms of problems and informational content. Long-range planning has also evoked interest in constructing organization models and examining them through simulation in an effort to predict the effects of proposed changes.

In terms of systems planning, the result has been a demand for analyzing company processes in a way that will permit mechanization of data elements at their source. The reason has been not only a desire to perpetuate the data in their original form but also the need for integrating the overall process and developing a truly realistic fundamental information system.

## SYSTEMS DESIGN

Whichever data processing system philosophy is selected—whether the total systems approach or the single information flow concept—management must make the choice and then stick to it. Once the choice is made, then each of the following steps can be taken:

1. Management can begin to define its corporate objectives precisely.

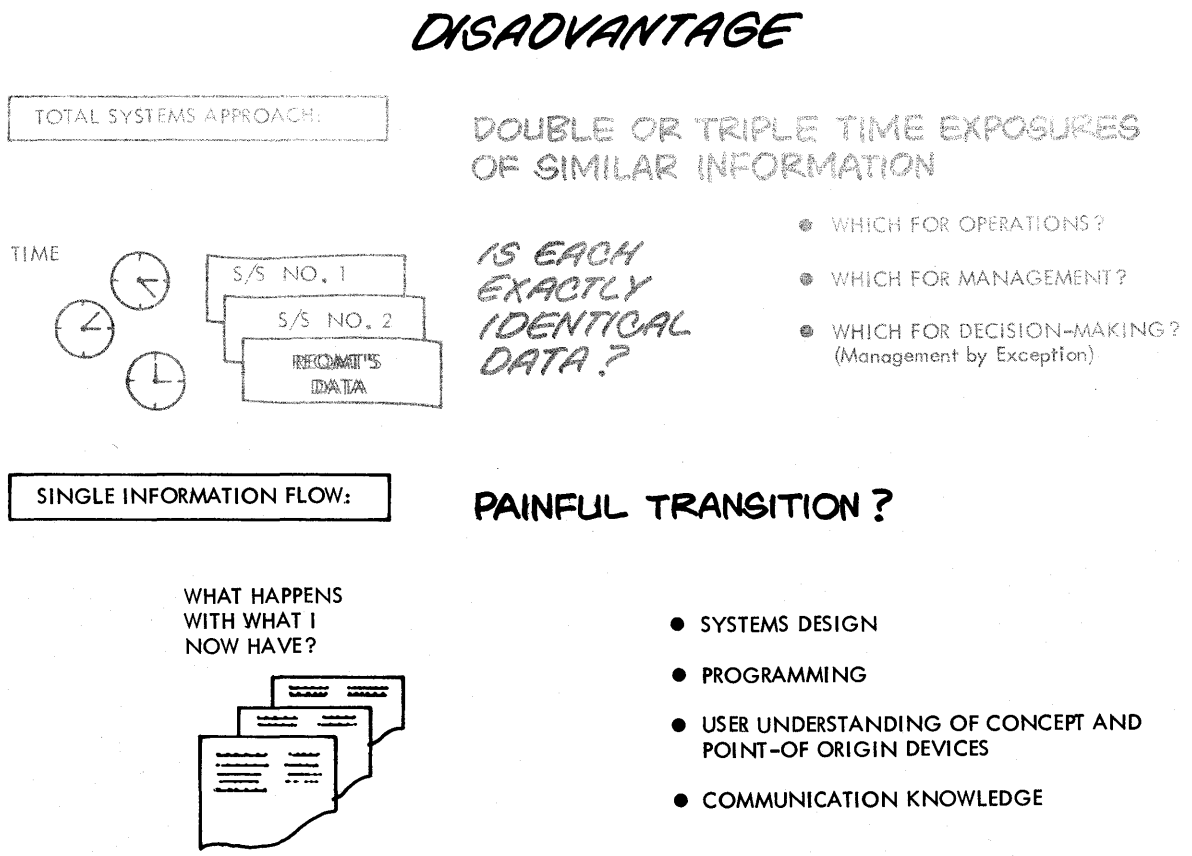



Figure 12

2. All systems plans can become oriented to these objectives.
3. Each resource can be analyzed to determine its contribution to the objectives and its interdependency with other resources.
4. Standard information flow procedures can be adopted and software developed.
5. Management's information needs can be converted into specific output formats.
6. Input formats and controls can be designed.
7. Editing and processing subroutines can be written.
8. The files can be converted and the system installed.

Systems design must be oriented toward corporate management's responsibility for directing the various activities of the enterprise. Management's success depends upon its ability to establish well-defined and measurable events within its area of responsibility. Competently designed information sytems will reduce the efforts managers must exert in making routine decisions, enabling them to obtain short-run results with minimum difficulty, and thus allow them to devote their energies to the major decisions of business strategy and long-range planning. To achieve this goal, decisions must be harnessed under policy and controlled through integrated data processing systems.

A basic plan for designing the information system in a typical company might be outlined as follows:

1. Determine management's needs to monitor the enterprise as a whole.
2. Design the fundamental information flow, indicating the interrelationships of the major functions and data, such as engineering, manufacturing, marketing, and finance.
3. Develop in detail the "essential" information that each function requires to operate efficiently.

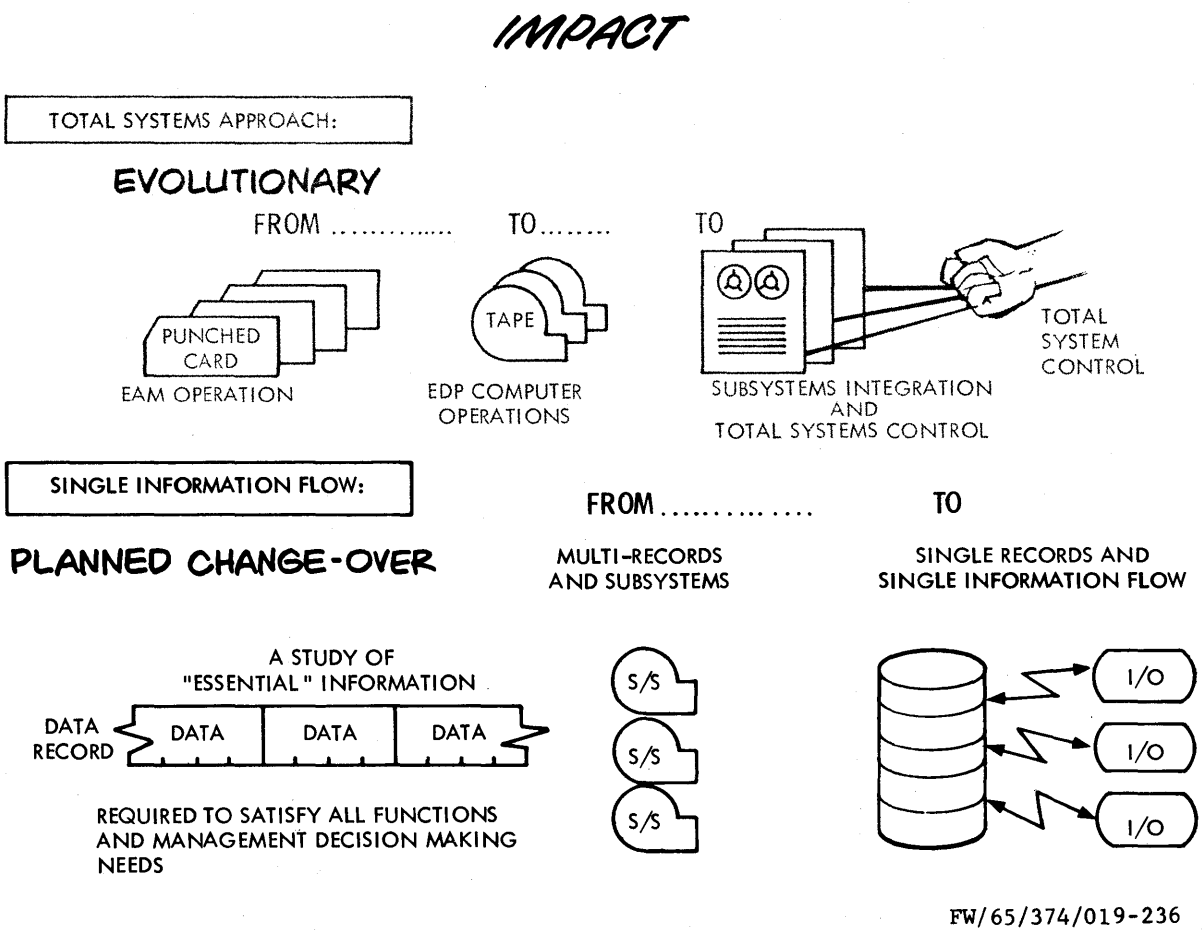



Figure 13

4. Determine each function's data and action requirements and their dependence upon other functions' actions and/or information.

After these steps have been completed, decision criteria responsive to management's needs can be formulated. In addition, measurable critical "information points" can be selected and a control network developed for economically retrieving and consolidating the information. Thus, management can be made aware of potential problems and their impact far enough in advance to take corrective action.

## IMPLEMENTATION NEEDS

After a satisfactory data processing approach and plan have been developed, they still have to be put into effect. The volume and everchanging complexity of business data make it difficult to satisfy even the current needs of management, much less its need for longer-range planning. The problem is complicated by the need for interpreting the data and perpetuating the information involved in the decision making processes. Furthermore, the information has to be manipulated rapidly to make it meaningful now—for judgments to be made and decisions to be arrived at in time to arrest potential problems.

The answer to these problems, in my opinion, lies in (1) high-speed data processing and communication equipment, (2) adoption of the single information flow data processing approach, and (3) a competent business systems engineering staff (see Fig. 15) capable of translating these fundamental requirements into the necessary data collection, processing, control, and selective information retrieval programs necessary to maintain a current picture of business activity within the company for all levels of management.

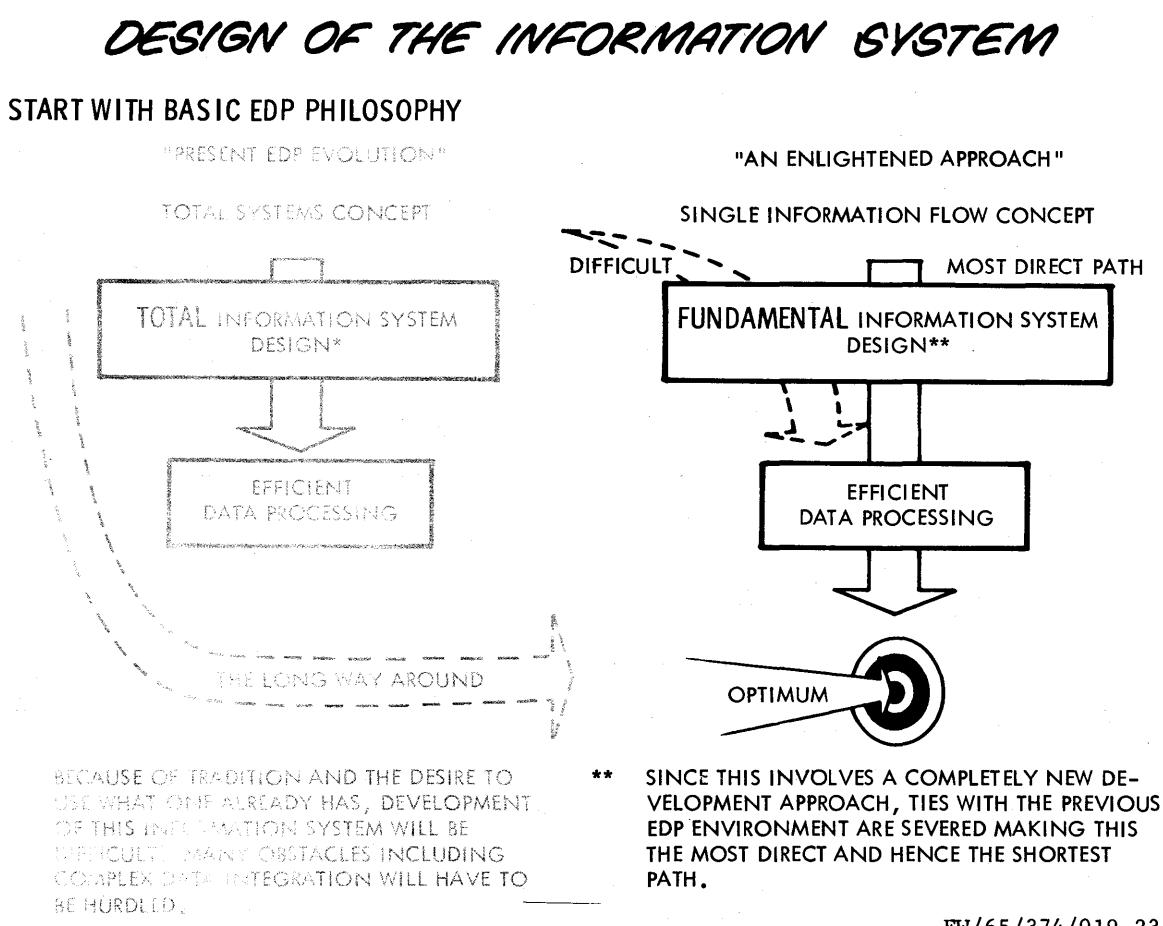



Figure 14

## REFERENCES

1. A. L. Baumann, Jr., "Single Information Flow Philosophy," *Data Processing Year Book,* American Data Processing, Inc., Detroit, Mich., 1963.

2. Gregory and Van Horn, *Automatic Data Processing Systems—Principles and Procedures,* 2nd ed., Wadsworth Publ. Co., Belmont, Calif., 1963.

3. R. E. Sprague, *Electronic Business Systems,* Ronald Press, New York, 1962.

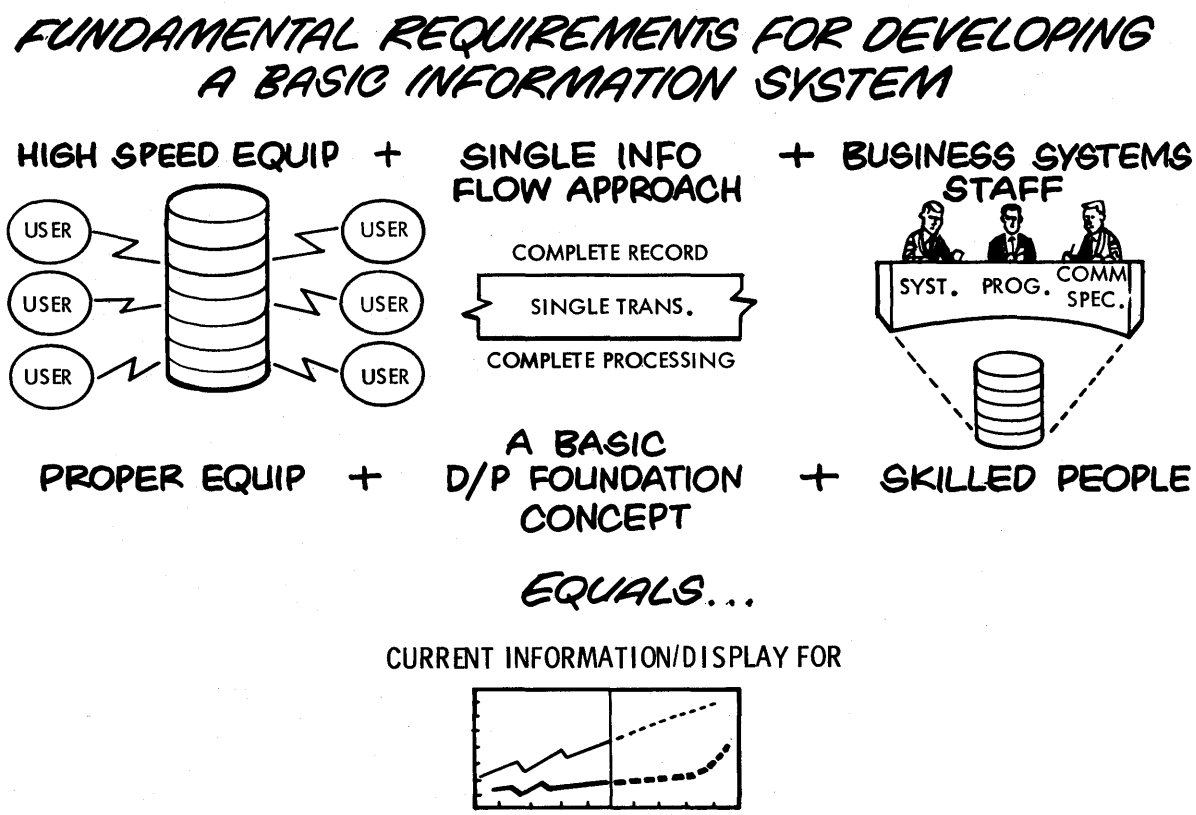

FUNDAMENTAL REQUIREMENTS FOR DEVELOPING A BASIC INFORMATION SYSTEM
HIGH SPEED EQUIP +
SINGLE INFO FLOW APPROACH
+ BUSINESS SYSTEMS STAFF
USER
USER
USER
USER
USER
USER
COMPLETE RECORD
SINGLE TRANS.
COMPLETE PROCESSING
SYST.
PROG.
COMM SPEC.
PROPER EQUIP +
A BASIC D/P FOUNDATION CONCEPT
+ SKILLED PEOPLE
EQUALS...
CURRENT INFORMATION/DISPLAY FOR
MANAGEMENT DECISION-MAKING AT ALL LEVELS
FW/65/374/019-240

Figure 15

# INTRODUCTION AND OVERVIEW OF THE MULTICS SYSTEM

F. J. Corbató
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*
and
V. A. Vyssotsky
*Bell Telephone Laboratories, Inc.*
*Murray Hill, New Jersey*

Multics (*Multi*plexed *I*nformation and *C*omputing *S*ervice) is a comprehensive, general-purpose programming system which is being developed as a research project. The initial Multics system will be implemented on the GE 645 computer. One of the overall design goals is to create a computing system which is capable of meeting almost all of the present and near-future requirements of a large computer utility. Such systems must run continuously and reliably 7 days a week, 24 hours a day in a way similar to telephone or power systems, and must be capable of meeting wide service demands: from multiple man-machine interaction to the sequential processing of absentee-user jobs; from the use of the system with dedicated languages and subsystems to the programming of the system itself; and from centralized bulk card, tape, and printer facilities to remotely located terminals. Such information processing and communication systems are believed to be essential for the future growth of computer use in business, in industry, in government and in scientific laboratories as well as stimulating applications which would be otherwise undone.

Because the system must ultimately be comprehensive and able to adapt to unknown future requirements, its framework must be general, and capable of evolving with time. As brought out in the companion papers,[1-5] this need for an evolutionary framework influences and contributes to much of the system design and is a major reason why most of the programming of the system will be done in the PL/I language.[6] Because the PL/I language is largely machine-independent (e.g. data descriptions refer to logical items, not physical words), the system should also be. Specifically, it is hoped that future hardware improvements will not make system and user programs obsolete and that implementation of the entire system on other suitable computers will require only a moderate amount of additional programming.

The present paper attempts to give a detailed dis-

*Work reported herein 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).

cussion of the design objectives as they relate to the major areas of the system. Some of the highlights of the subsequent papers are: a virtual memory system for each user involving two-dimensional addressing with segmentation and paging; the dynamic linking of program segment cross-references at execution time to minimize system overhead; the routine use of sharable, recursive, pure procedure programming within the system as the normal mode of operation; the pooled use of multiple processors, memory modules, and input-output controllers; and multiprogramming of all resources and of multiple users. Automatic management of the complex of secondary storage media along with backup, retrieval, and maintenance procedures for the stored information will be provided by a file system. Further, it is expected that most of the software of the system will be almost identical in form to user programs. The system will incorporate automatic page-turning for both user and system programs alike.

## INTRODUCTION

As computers have matured during the last two decades from curiosities to calculating machines to information processors, access to them by users has not improved and in the case of most large machines has retrogressed. Principally for economic reasons, batch processing of computer jobs has been developed and is currently practiced by most large computer installations, and the concomitant isolation of the user from elementary cause-and-effect relationships has been either reluctantly endured or rationalized. For several years a solution has been proposed to the access problem.[7-9] This solution, usually called time-sharing, is basically the rapid time-division multiplexing of a central processor unit among the jobs of several users, each of which is on-line at a typewriter-like console. The rapid switching of the processor unit among user programs is, of course, nothing but a particular form of multiprogramming.

It is now abundantly clear that it is possible to create a general-purpose time-shared multiaccess system on many contemporary computers (especially after minor but basic modifications are made). Already two major and extensive systems have been created, one on the IBM 7094[10,11] and one on the Q-32 computer.[12] In addition, there have been numerous smaller scale systems, the most notable being on the DEC PDP-1,[13,14] the IBM 7094,[15] the GE-235,[16] the DEC PDP-6,[17] and the SDS 930,[18] as well as somewhat more limited versions of time-sharing on the RW-400,[19,20] and the CDC G21,[21] the Johnniac,[22] and the IBM 7040.[23] As time goes on, surveys of implemented systems are being made[23,24] and "score cards" are being kept.[25]

The impetus for time-sharing first arose from professional programmers because of their constant frustration in debugging programs at batch processing installations. Thus, the original goal was to time-share computers to allow simultaneous access by several persons while giving to each of them the illusion of having the whole machine at his disposal. However, at Project MAC it has turned out that simultaneous access to the machine, while obviously necessary to the objective, has not been the major ensuing benefit.[26] Rather, it is the availability at one's fingertips of facilities for editing, compiling, debugging, and running in one continuous interactive session that has had the greatest effect on programming. Professional programmers are encouraged to be more imaginative in their work and to investigate new programming techniques and new problem approaches because of the much smaller penalty for failure. But, the most significant effect that the MAC system has had on the MIT community is seen in the achievements of persons for whom computers are tools for other objectives. The availability of the MAC system has not only changed the way problems are attacked, but also important research has been done that would not have been undertaken otherwise. As a consequence the objective of the current and future development of time-sharing should extend way beyond the improvement of computational facilities with respect to traditional computer applications. Rather, it is the on-line use of computers for new purposes and in new fields which should provide the challenge and the motivation to the system designer. In other words, the major goal is to provide suitable tools for what is currently being called machine-aided cognition.

More specifically, the importance of a multiple-access system operated as a computer utility is that it allows a vast enlargement of the scope of computer-based activities, which should in turn stimulate a corresponding enrichment of many areas of our society. Over two years of experience indicates that continuous operation in a utility-like manner,

with flexible remote access, encourages users to view the system as a thinking tool in their daily intellectual work. Mechanistically, the qualitative change from the past results from the drastic improvement in access time and convenience. Subjectively, the change lies in the user's ability to control and affect interactively the course of a process whether it involves numerical computation or manipulation of symbols. Thus, parameter studies are more intelligently guided; new problem-oriented languages and subsystems are developed to exploit the interactive capability; many complex analytical problems, as in magnetohydrodynamics, which have been too cumbersome to be tackled in the past are now being successfully pursued; even more, new, imaginative approaches to basic research have been developed as in the decoding of protein structures. These are examples taken from an academic environment; the effect of a multiple-access system on business and industrial organizations can be expected to be equally dramatic but experience in this area is still very limited. It is with such new applications in mind that the Multics system has been developed. Not that the traditional uses of computers are being disregarded. Rather, these needs are viewed as a subset of the broader more demanding requirements of the former.

To meet the above objectives, issues such as response time, convenience of manipulating data and program files, ease of controlling processes during execution and above all, protection of private files and isolation of independent processes become of critical importance. These issues demand departures from traditional computer systems. While these departures are deemed to be desirable with respect to traditional computer applications, they are essential for rapid man-machine interaction.

## SYSTEM REQUIREMENTS

In the early days of computer design, there was the concept of a single program on which a single processor computed for long periods of time with almost no interaction with the outside world. Today such a view is considered incomplete; for the effective boundaries of an information processing system extend beyond the processor, beyond the card reader and printer and even beyond the typing of input and the reading of output. In fact they encompass as well what several hundred persons are trying to accomplish. To better understand the effect of this broadened design scope, it is helpful to examine several phenomena characteristic of large service-oriented computer installations.

First, there are incentives for any organization to have the biggest possible computer system that it can afford. It is usually only on the biggest computers that there are the elaborate programming systems, compilers and features which make a computer "powerful." This comes about partly because it is more difficult to prepare system programs for smaller computers when limited by speed or memory size and partly because the larger systems involve more persons as manufacturers, managers, and users and hence permit more attention to be given to the system programs. Moreover, by combining resources in a single computer system, rather than in several, bulk economies and therefore lower computing costs can be achieved. Finally, as a practical matter, considerations of floor space, management efficiency and operating personnel provide a strong incentive for centralizing computer facilities in a single large installation.

Second, the capacity of a contemporary computer installation, regardless of the sector of applications it serves, must be capable of growing to meet a continuously increasing demand. A doubling of demand every two years is not uncommon.[27] Multiple-access computers promise to accelerate this growth further since they allow a man-machine interaction rate which is faster by at least two orders of magnitude. Present indications are that multiple-access systems for only a few hundred simultaneous users can generate a demand for computation exceeding the capacity of the fastest existing single-processor system. Since the speed of light, the physical sizes of computer components, and the speeds of memories are intrinsic limitations on the speed of any single processor, it is clear that systems with multiple processors and multiple memory units are needed to provide greater capacity. This is not to say that fast processor units are undesirable, but that extreme system complexity to enhance this single parameter among many appears neither wise nor economic.

Third, computers are no longer a luxury used when and if available, but primary working tools in business, government, and research laboratories. The more reliable computers become, the more their availability is depended upon. A system structure including pools of functionally identical units

(processors, memory modules, input/output controllers, etc.) can provide continuous service without significant interruption for equipment maintenance, as well as provide growth capability through the addition of appropriate units.

Fourth, user programs, especially in a time-sharing system, interact frequently with secondary storage devices and terminals. This communication traffic produces a need for multiprogramming to avoid wasting main processor time while an input/output request is being completed. It is important to note that an individual user is ordinarily incapable of doing an adequate job of multiprogramming since his program lacks proper balance, and he probably lacks the necessary dynamic information, ingenuity or patience.

Finally, as noted earlier, the value of a time-sharing system lies not only in providing, in effect, a private computer to a number of people simultaneously, but, above all, in the services that the system places at the fingertips of the users. Moreover, the effectiveness of a system increases as user-developed facilities are shared by other users. This increased effectiveness because of sharing is due not only to the reduced demands for core and secondary memory but also to the cross-fertilization of user ideas. Thus a major goal of the present effort is to provide multiple access to a growing and potentially vast structure of shared data and shared program procedures. In fact, the achievement of multiple access to the computer processors should be viewed as but a necessary subgoal of this broader objective. Thus the primary and secondary memories where programs reside play a central role in the hardware organization and the presence of independent communication paths between memories, processors and terminals is of critical importance.

From the above it can be seen that the system requirements of a computer installation are not for a single program on a single computer, but rather for a large system of many components serving a community of users. Moreover, each user of the system asynchronously initiates jobs of arbitrary and indeterminate duration which subdivide into sequences of processor and input/output tasks. It is out of this seemingly chaotic, random environment that one arrives at a utility-like view. For instead of chaos, one can average over the different user requests to achieve high utilization of all resources. The task of multiprogramming required to do this need only be organized once in a central supervisor program. Each user thus enjoys the benefit of efficiency without having to average the demands of his own particular program.

With the above view of computer use, where tasks start and stop every few milliseconds and where the memory requirements of tasks grow and shrink, it is apparent that one of the major jobs of the supervisor program (i.e., "monitor," "executive," etc.) is the allocation and scheduling of computer resources. The general strategy is clear. Each user's job is subdivided into tasks, usually as the job proceeds, each of which is placed in an appropriate queue (i.e., for a processor or an input/output controller). Processors or input/output controllers are in turn assigned new tasks as they either complete or are removed from old tasks. All processors are treated equivalently in an anonymous pool and are assigned to tasks as needed; in particular, the supervisor does not have a special processor. Further, processors can be added or deleted without significant change in either the user or system programs. Similarly, input/output controllers are directed from queues independently of any particular processor. Again, as with the processors, one can add or delete input/output capacity according to system load without significant reprogramming required.

## THE MULTICS SYSTEM

The overall design goal of the Multics system is to create a computing system which is capable of comprehensively meeting almost all of the present and near-future requirements of a large computer service installation. It is not expected that the initial system, although useful, will reach the objective; rather the system will evolve with time in a general framework which permits continual growth to meet unknown future requirements. The use of the PL/I language will allow major system software changes to be developed on a schedule separate from that of hardware changes. Since most organizations can no longer afford to overlap old and new equipment during changes, and since software development is at best difficult to schedule, this relative machine-independence should be a major asset.

It is expected that the Multics system will be published when it is operating substantially and will therefore be available for implementation on any

equipment with suitable characteristics. Such publication is desirable for two reasons: First, the system should withstand public scrutiny and criticism volunteered by interested readers; second, in an age of increasing complexity, it is an obligation to present and future system designers to make the inner operating system as lucid as possible so as to reveal the basic system issues.

The accompanying papers describe in some detail how the Multics system will meet its objectives. However, it is useful, in establishing an overview, to touch on the highlights and especially on the design motivation.

## DESIGN FEATURES OF THE HARDWARE

The Multics system objectives required equipment features that were not present in any existing computer. Consequently it was necessary to develop for the Multics system the GE 645 computer. The GE 635 computer was selected for modification to the GE 645 inasmuch as it already satisfied many of the crucial requirements. In particular, it was designed to have multiprocessors, multiple memory modules, and multiple input/output controllers. Thus, the requirements of modular construction for reliability and for ease of growth were amply met. The communication pattern is particularly straightforward since there are no physical paths between the processors and the input/output equipment; rather all communication is done by means of "mailboxes" in the memory modules and by corresponding interrupts. Furthermore, major modules of the system communicate on an asynchronous basis; thus, any single module can be upgraded without any changes to the other modules. This latter property is useful in that one of the ways in which system capacity (and cost) may be regulated is by changing either the speed or number of memory modules. Of course further adjustment of system capacity is possible by varying the number of processor units or the configuration of drum and disk equipment. In any case, one obtains the important simplification that a single supervisor program can operate without substantial change on any configuration of equipment.

Figure 1 illustrates the equipment configuration of a typical Multics system. All central processors (CPU) and Generalized Input/Output Controllers (GIOC) have communication paths with each of the memory modules. When necessary for maintenance or test purposes, the system can be partitioned into two independent systems (although each of the drum, disk and tapes must belong to one of the two systems). The remote terminals can dial either of the two GIOC through the private branch exchange, which is not shown in the figure.

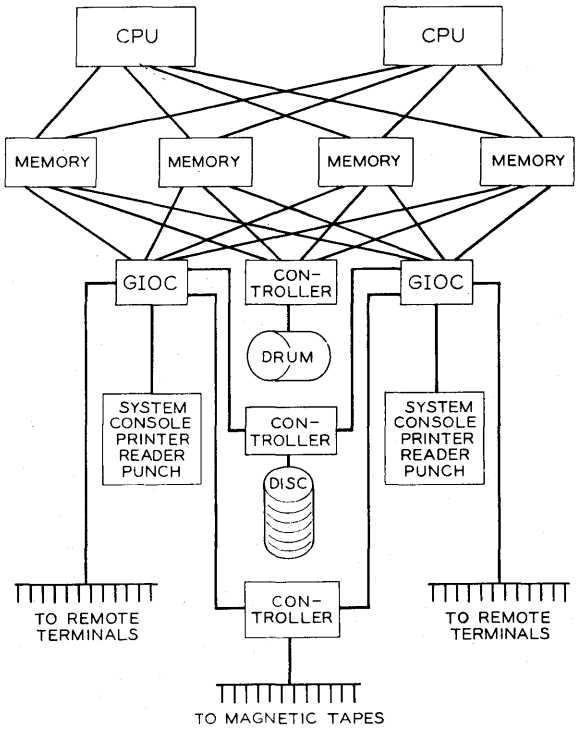



Figure 1. Example of Multics system configuration.

The most novel feature in the GE 645 is in the instruction addressing. A two-dimensional addressing system has been incorporated which allows each user to write programs as though there is a virtual memory system of large size. This system is organized into program segments (i.e., regions) each of which contains an ordered sequence of words with a conventional linear address. These segments, which can vary in length during execution, are paged at the discretion of the supervisor program with either 64- or 1,024-word pages. This dual page size allows the supervisor program to use more effective strategies in the handling of multiple users. Paging, first introduced on the Atlas computer,[28] allows flexible dynamic memory allocation techniques as well as the sensible implementation of a one-level store system. To the user in the Multics system, page addressing is invisible; rather, it is the segments which are explicitly known to him and to which he is able to refer symbolically in his programs. These notions were first suggested by Holt,[29] further developed by Den-

nis,[30,31] Dennis and Glaser,[32] Forgie,[33] and others.[34,35] The value of segmentation and paging has since been widely discussed during the past year and has gained broader acceptance.[36-39] The explicit hardware implementation details of segmentation and paging for the Multics system are discussed in the companion paper by Glaser, Couleur and Oliver.[1]

Because two-dimensional addressing is rather new, it is useful to clarify the reasons for it.

The major reasons for segments are:

1. The user is able to program in a two-dimensional virtual memory system. Thus, any single segment can grow (or shrink) during execution (e.g., in the GE 645, each user may have up to a quarter million segments, each including up to a quarter million words).
2. The user can, by merely specifying a starting point in a segment, operate a program implicitly without prior planning of the segments needed or of the storage requirements. For example, if an error diagnostic segment is unexpectedly called for, it is brought in automatically by the supervisor; it is never brought in unless needed. Similarly, elaborate computations which branch into many different segments in a data-dependent way use segments only as needed.
3. The largest amount of code which must be bound together as a solid block is a single segment. Since binding pieces of code together (sometimes called "loading") is a process similar to assembling or compiling, the advantage of being able to prepare an arbitrarily large program as a series of limited-overhead segment bindings is significant. The saving in overhead is comparable to that in FORTRAN when one uses multiple subprograms instead of a single large combined block of statements. If the combined block is used, not only does the compilation process become particularly cumbersome but the eradication of programming errors in all the different sections requires more compilations.
4. Program segments appear to be the only reasonable way to permit pure procedures and data bases to be shared among several users simultaneously. Pure procedure programs, by definition, do not modify themselves. Therefore a supervisor program can minimize the core memory requirements of a collection of user programs by supplying only one copy of a jointly used pure procedure. Nearly all of the Multics system as well as most of the user programs will be written in this form. One consequence is that there will be no clearcut demarcation between user programs and system programs; instead the demarcation will depend largely on the responsibility for maintenance.

Pages are a separate feature from segments and have further and distinct advantages.

1. The use of paged memory allows flexible techniques for dynamic storage management without the overhead of moving programs back and forth in the primary memory. This reduced overhead is important in responsive time-shared systems where there is heavy traffic between primary and secondary memories.
2. The mechanism of paging, when properly implemented, allows the operation of incompletely loaded programs; the supervisor need only retain in main memory the more active pages, thus making more effective use of high-speed storage. Whenever a reference to a missing page occurs, the supervisor must interrupt the program, fetch the missing page, and reinitiate the program without loss of information.

A critical feature in the segment and paging hardware is the descriptor bit mechanism which controls the access of processors to the memory. These bits essentially allow hardware "fire-walls" to be established within the programming system which assist the isolation of hardware or software difficulties. Besides controlling the usual properties such as read-only, data-only, etc., one descriptor bit allows a segment to be declared "execute-only." The presence of this bit allows procedures to be transferred to and executed but never read by user programs. This feature will be of interest to commercial service bureaus, and in application areas where privacy of program procedure is essential (e.g., a class-room grading program). Another property of the descriptors is that they allow most of the supervisor modules to be written with

the same descriptors as user programs; most system programs thereby do not have access to privileged instructions, the inadvertent use of which can cause drastic machine misbehavior. This feature is especially pertinent when it is recognized that time-sharing systems are real-time systems with behavior which it is difficult to duplicate or repeat. Consequently, all possible compartments and protection mechanisms that one can have are of value.

For effective operation of the Multics system, a drum with a high transfer rate is needed. The drum provided with the GE 645 meets the requirement and allows convenient and efficient management of a high rate of input/output requests. In particular, requests are organized by the supervisor program into queues in core memory and are fetched from these queues by the drum controller asynchronously of the processors. Because of the queues and because drum record sizes are commensurate with core memory page sizes, it is straightforward to program for continuous input/output transmission without latency delays.

Disk input/output requests are also organized into queues and are fetched from core memory by the generalized input/output controller. This controller is discussed in more detail in the paper by Ossanna et al.[4] Again, because the supervisor is contending with a statistical mix of user and supervisor requests for information to and from disk, it is expected that latency delays between requests will be negligible. Because the transmission capacity to the disk is large, system performance is expected to be unhampered by input/output bottlenecks.

Since the Multics system will be used as an information processor in a wide range of applications, it is important that a readable character set be used. The standard character set will be the recently proposed ASCII code which has 128 codes and includes upper and lower case letters.[40] This set, which contains 95 printing graphics, can be reasonably represented on contemporary input/output consoles. Line printers capable of printing the 95 graphics will be standard equipment.

## DESIGN FEATURES OF THE SOFTWARE

An important aspect of the software is the subroutine and linkage conventions which are associated with the use of the segment and paging hardware. The following features are incorporated.

1. Any segment has to know another segment only by symbolic name. Intersegment binding occurs dynamically as needed during program execution. Intersegment binding is automatic (i.e., not explicitly programmed by the user) and the mechanism operates at high efficiency after the first binding occurs.
2. Similarly, a segment is able to reference symbolically a location within another segment. This reference binds dynamically and automatically; after binding occurs the first time, program execution is at full speed.
3. It is straightforward for procedures to be pure procedures, capable of being shared by several users.
4. Similarly, it is straightforward to write recursive procedures (i.e., subroutines capable of calling on themselves either directly or indirectly by a circular chain of calls).
5. The general conventions are such that the call, save, and return macros used to link one independently compiled procedure to another do not depend on whether or not the two procedures are in the same segment.
6. Each user is provided with a private software "stack" for temporary storage within each subroutine. Of course, any user can choose to ignore this storage mechanism, but it is available and does not have to be added as an afterthought by a subsystem designer.

In addition, there is basically only one kind of calling sequence, thus avoiding much confusion. System programming is done with the same facilities, tools, etc., available to the ordinary user, and system programs do not have to be written with special forethought. It is anticipated that the system will be open-ended and will be largely created by the users themselves; many of the useful languages and subsystems will undoubtedly be contributed without solicitation. For this reason supervisor and user programs are constructed with similar form, and processes such as paging do not distinguish between user and supervisor programs. (Of course, a few key pieces of the supervisor are locked in core memory.) Thus there is no intrinsic limit on the size of the supervisor program nor on the complexity or the features which it may have. The avoidance

of a size limitation will be of major value as the system services grow.

It is important to recognize that the average user of the system will see no part of the segmentation and paging complexity described in the paper by Glaser et al. Instead he will see a virtual machine with many system characteristics which are convenient to him for writing either single programs or whole subsystems. As a subsystem writer he must be able to make the computer appear to have any particular form ranging from an airline reservations system, to an inventory control system, from a management gaming machine, to even a "FORTRAN machine" if so desired. There are no particular restrictions on the kinds of new systems or languages which can be imbedded.

Further features which should ultimately appear in the system are:

1. the ability to have one process spawn other processes which run asynchronously on several processors (thus improving the real-time response of the overall process);
2. the ability for data bases to be shared among simultaneously operating programs.

In addition the system will include all the major features of the present Project MAC system such as interconsole messages and macro-commands. The latter allow users to concatenate sequences of console-issued commands as short programs thereby forming more elaborate commands which can be used with a single name and parameter call.

Another feature of the system is that it will include batch processing facilities as a subset. In particular, users will start processes which may have $n$ terminals attached, with $n=1$ for individual man-machine interaction, and $n=0$ for running an absentee-user program, the latter case corresponding to batch processing. A user will be able to transform conveniently a process back and forth between the zero and one terminal states. In addition, for the purposes of teaching machines and gaming experiments, it will be possible to attach to a process an arbitrary number of additional terminals.

The supervisor will, of course, do scheduling and charging for the use of resources. Scheduling policies will be similar but more general than those currently in the MAC system; for batch processing, jobs should be scheduled so that a user will be able to obtain a quotation of maximum completion time. The time accounting done by the system will be accurate to a few microseconds. In particular, the system will "fight back" by charging for exactly what equipment is used (or others are prevented from using). In this way, orderly system expansion will be possible since the particular equipment charges which are collected will always allow further acquisition of equipment. In addition the system will incorporate hierarchal control of resource allocations and accounting authorizations. A project manager will be able to give computing budgets to group leaders who in turn will be able to delegate flexibly and straightforwardly sub-budgets to team leaders, etc. An important aspect of this resource allocation and budgeting is the ability of any member of the hierarchy to reallocate flexibly those resources over which he has control. With control of the resource allocation and administrative accounting decentralized, the operation of systems which serve hundreds of persons becomes manageable.

In a similar way, system programming is decentralized. For example, the maintenance of the system might not be entirely under the control of a single group; instead particular translators might be delegated to independent subgroups of system programmers. This isolation and distribution of responsibility is considered mandatory for the growth of large, effective systems. Hierarchal and decentralized accounting and system programming is made possible by a highly organized file system which controls the access rights to the secondary memory of the system and thus to the file copies of the vital procedures and data of the system.

## DESIGN CONSIDERATIONS IN THE FILE SYSTEM

The file system is a key part of a time-sharing or multiplexed system. It is a memory system which gives the users and the supervisor alike the illusion of maintaining a private set of segments or files of information for an indefinite period of time. This retention is handled by automatic mechanisms operated by the supervisor and is independent of the complex of secondary storage devices of different capacity and access. A scheme, such as is described in the paper by Daley and Neumann,[3] where all files of information are referred to by symbolic name and not by address, allows changes in the secondary storage complex for reasons either of reliability or capacity. In particular, the user is never responsible

for having to organize the movement of information within the secondary storage complex. Instead the file system has a strategy for arranging for high-speed access to recently used material.

Of considerable concern is the issue of privacy. Experience has shown that privacy and security are sensitive issues in a multi-user system where terminals are anonymously remote. For this reason, each user's files can be arranged to be completely private to him. In addition, a user may arrange to allow others to access his files selectively on a linking basis. The linking mechanism permits control over the degree of access one allows (e.g., a user may wish a file to be read but not written). The file system allows files to be simultaneously read but automatically interlocks file writing.

The file system is designed with the presumption that there will be mishaps, so that an automatic file backup mechanism is provided. The backup procedures must be prepared for contingencies ranging from a dropped bit on a magnetic tape to a fire in the computer room.

Specifically, the following contingencies are provided for:

1. A user may discover that he has accidentally deleted a recent file and may wish to recover it.
2. There may be a specific system mishap which causes a particular file to be no longer readable for some "inexplicable" reason.
3. There may be a total mishap. For example, the disk-memory read heads may irreversibly score the magnetic surfaces so that all disk-stored information is destroyed.

The general backup mechanism is provided by the system rather than the individual user, for the more reliable the system becomes, the more the user is unable to justify the overhead (or bother) of trying to arrange for the unlikely contingency of a mishap. Thus an individual user needs insurance, and, in fact, this is what is provided.

## DESIGN CONSIDERATIONS IN THE COMMUNICATION AND INPUT/OUTPUT EQUIPMENT

A design feature of the system is that users can view most input/output devices uniformly. Thus a program can read from either a terminal or a disk file, or output can be sent either to a file or to a punch, a typewriter, or a printer. In particular, the user of the system does not have to rewrite his program to change these assignments from day to day or from use to use. The symmetric use of equipment is, of course, highly desirable and makes for greater simplicity and flexibility.

A typical configuration of the Multics system will contain batch processing input/output devices such as card readers, punches and printers and these normally will be centrally located at the main computing installation. For remote users there will be terminals such as the Model 37 Teletype which uses the revised ASCII code with upper and lower case letters. The Model 37 Teletype also can operate on the TWX network of the Bell System. It will therefore be possible for many of the 60,000 TWX subscribers to be, if authorized, users of a Multics installation. An additional standard terminal for the Multics system will be a modified version of the IBM 1052 console. This unit (and all other terminal devices which do not have the ASCII character set) will have software escape conventions, defined to allow unambiguous input or output of the complete ASCII character set. The escape conventions are general and allow even primitive devices (in a graphic sense) to communicate with the system. The IBM 1052 terminals, which basically use the Selectric typewriter mechanism, are operated with a special typeball, prepared for Project MAC as a compromise subset of the ASCII graphics.

For those users who wish to have remotely located satellite substations capable of punching and reading cards and line printing, there are a variety of options available. Because the design of the General Input/Output Controller is relatively flexible, it is possible to use the GE 115, the Univac 1004, or virtually any other similar subcomputer as a terminal, provided one is prepared to implement the necessary interface program modules within a Multics system. At present none of these terminals are completely satisfactory since the full 128-code revised ASCII character set is not standard and excessive use of the software escape mechanism is required for printing.

In general, the area of remote terminal equipment is considered to be in an early state of development. Equipment innovations are expected, as it becomes evident that systems are capable of supporting their use. Terminals with graphical in-

put/output are highly desirable although at present costly. The initial approach of the Multics system will be such that there will be no standard graphical input/output terminal although several special projects are being attempted. The system viewpoint initially will be that all graphical input/output will be with small, dedicated computers capable of handling the immediate interrupts. These small computers may multiplex a few terminals and in turn appear to be not too demanding to the main system. Thus the main system interrupt load will not become excessive. In a similar way the need for real-time instrumentation such as in monitoring experimental apparatus is expected to be handled initially on a nonstandard basis. The philosophy is the same as with graphical input/output, namely, to employ small, dedicated computers for handling the real-time interrupts so as to draw upon the main system for major processing of information in a more leisurely way.

## GENERAL CONSIDERATIONS

It is expected that the ultimate limitation on the exploitation of the Multics system will be the knowledge which the user has of it. As a consequence, documentation of what the system contains is considered to be one of the most important aspects of the system. For this purpose a technique has been developed wherein the main system reference manual is to be maintained on-line in a fashion similar to what is currently being done at Project MAC. This allows any user of the system to obtain a current table of contents with changes listed in reverse chronological order. Thereby he can keep abreast of all system changes. Because the manual text is on-line, one is able to obtain immediate access to the latest changes at any hour or at any terminal. The on-line storage of the text also lets the system documentation group, by using appropriate editing programs, make global revisions whenever necessary. Of course, the distribution of manual revisions will still be handled in the ordinary way in that revised manual sections will be available at document rooms. Furthermore, it should be clear that there is no substitute for a good editor maintaining discipline over the documentation and for intelligent selectivity in the reference material. A documentation technique such as the one given here is believed to be an absolute necessity when users of the system no longer visit a computation center in the course of their daily activities. The user who is 200 miles away from the computer installation should have nearly the same knowledge about the system as the one who is 20 feet away.

Another area of consideration is that of compatibility with batch processing. In the Multics system for the GE 645, it will be possible to use simultaneously, but independently, the GECOS batch-processing system; user jobs operating under GECOS should behave exactly as they do on the GE 625 or GE 635 computers. Effort will be made to allow the GECOS user to change conveniently to the Multics frame of operation but there will be no particular attempt made for compatibility between the two systems of basically different design. A user of the GECOS system may continue to use the GECOS system until he is prepared to make a change to the Multics system at his own place, time, and choosing. This, of course, relieves a manager installing a Multics system of the transient effect of several hundred persons changing their computing habits in one day and thus allows distribution of the normal dissatisfaction that arises under such circumstances.

One of the inevitable questions asked of a multiple-access system is what capacity it will have for simultaneous on-line users. The answer, of course, is highly dependent upon what the users are doing. Clearly, if they are requesting virtually nothing, one can have a nearly infinite number of terminals. Conversely, if one person wishes, for a single problem, system resources which equal the entire computing system, it is conceivable, if the scheduling policy allows it, that there can be only one terminal attached to the system. If one assumes that the service requirements are similar to those which have been experienced at Project MAC, then on the basis of simple scaling of processor and memory speed it is expected that the system will be able to serve simultaneously a few hundred users. But it is hazardous to predict any firm numbers; rather the pertinent parameters in a system of this type will always be the cost-performance figures. Performance, of course, is somewhat subjective, but the issues are not those of memory speed, processor speed or input/output speed. Instead the user should judge a system by the quality and variety of services, the response times, the reliability, the overall ease of understanding the system, and the

performance with respect to the interface of the system which he uses. For example, pertinent questions for a PL/I user to ask are how costly, on the average, the translator is per statement, how easy it is to debug the language, and how efficiently the object code produced by the translator runs. Here, the object code referred to is that for an entire problem and not just for isolated "kernels"; the efficiency refers to the total resource drain required to execute the problem and thereby includes the input/output demands as well.

## CONCLUSIONS

The present plans for the Multics system are not unattainable. However, it is presumptuous to think that the initial system can successfully meet all the requirements that have been set. The system will evolve under the influence of the users and their activities for a long time and in directions which are hard to predict at this time. Experience indicates that the availability of on-line terminals drastically changes user habits and these changes in turn suggest changes and additions to the system itself.

It is expected that most of the system additions will come from the users themselves and the system will eventually become the repository of the procedure and data knowledge of the community. The Multics system will undoubtedly also open up large classes of new uses not only in science and engineering but also in other areas such as business and education. Just as introduction of higher-level programming languages, such as FORTRAN, increased by an order of magnitude the number of persons using computers, multiple-access systems operated as a utility will substantially extend the exploitation of information processing systems to the point of having significant social consequences. Such social issues are explored in a companion paper by David and Fano.[5]

## REFERENCES

1. E. L. Glaser, J. F. Couleur and G. A. Oliver, "System Design for a Computer for Time-Sharing Application," this volume.

2. V. A. Vyssotsky, F. J. Corbató and R. M. Graham, "Structure of the Multics Supervisor," this volume.

3. R. C. Daley and P. G. Neumann, "A General Purpose File System for Secondary Storage," this volume.

4. J. F. Ossanna, L. E. Mikus and S. D. Dunten, "Communications and Input/Output Switching in a Multiplex Computing System," this volume.

5. E. E. David, Jr., and R. M. Fano, "Some Thoughts About the Social Implications of Accessible Computing," this volume.

6. "IBM Operating System/360, PL/I: Language Specifications," File No. S360-29, Form C28-6571-1, I.B.M. Corp.

7. C. Strachey, "Time Sharing in Large Fast Computers," *Proceedings of the International Conference on Information Processing, UNESCO,* June 1959, paper B. 2. 19.

8. J. C. R. Licklider, "Man-Computer Symbiosis," *IRE Transactions on Human Factors in Electronics,* vol. HFE-1, no. 1, pp. 4-11 (Mar. 1960).

9. J. McCarthy, "Time-Sharing Computer Systems," *Management and the Computer of the Future* (M. Greenberger, ed.), M.I.T. Press, Cambridge, Mass, 1962, pp. 221-236.

10. F. J. Corbató, M. M. Daggett and R. C. Daley, "An Experimental Time-Sharing System," *Proceedings of the Spring Joint Computer Conference, 21,* Spartan Books, Baltimore, 1962, pp. 335-344.

11. F. J. Corbató et al, *The Compatible Time-Sharing System: A Programmer's Guide,* 1st ed., M.I.T. Press, Cambridge, Mass., 1963.

12. J. Schwartz, A General Purpose Time-Sharing System, *Proceedings of the Spring Joint Computer Conference, 25,* Spartan Books, Washington, D.C., 1964, pp. 397-411.

13. J. B. Dennis, "A Multiuser Computation Facility for Education and Research," *Comm. ACM,* vol. 7, pp. 521-529 (Sept. 1964).

14. S. Boilen et al, "A Time-Sharing Debugging System for a Small Computer," *Proceedings of the Spring Joint Computer Conference, 23,* Spartan Books, Baltimore, 1963, pp. 51-58.

15. H. A. Kinslow, "The Time-Sharing Monitor System," *Proceedings of the Fall Joint Computer Conference, 26,* Spartan Books, Washington, D.C., 1964, pp. 443-454.

16. "The Dartmouth Time-Sharing System," Computation Center, Dartmouth College, Oct. 19, 1964.

17. "PDP-6 Time-Sharing Software," Form F-61B, Digital Equipment Corp., Maynard, Mass.

18. W. W. Lichtenberger and M. W. Pirtle, "A Facility for Experimentation in Man-Machine Interaction," this volume.

19. G. J. Culler and R. W. Huff, "Solution of Nonlinear Integral Equations Using On-line Computer Control," *Proceedings of the Spring Joint Computer Conference, 21,* Spartan Books, Baltimore, 1962, pp. 129-138.

20. G. J. Culler and B. D. Fried, "The TRW Two-Station, On-Line Scientific Computer: General Description," *Computer Augmentation of Human Reasoning, Washington, D. C., June 1964,* Spartan Books, Washington, D.C., 1965.

21. "Carnegie Institute of Technology Computation Center User's Manual."

22. J. C. Shaw, "JOSS: A Designer's View of an Experimental On-Line Computing System," *Proceedings of the Fall Joint Computer Conference, 26,* Spartan Books, Washington, D.C., 1964, pp. 455-464.

23. T. M. Dunn and J. H. Morrissey, "Remote Computing—An Experimental System," Part 1; J. M. Keller, E. C. Strum and G. H. Yang, Part 2, *Proceedings of the Spring Joint Computer Conference, 25,* Spartan Books, Washington, D.C., 1964, pp. 413-443.

24. J. I. Schwartz, "Observations on Time-Shared Systems," *ACM Proceedings of the 20th National Conference,* p. 525 (1965).

25. "Time-Sharing System Scorecard, No. 1 (Spring 1965)," Computer Research Corp., 747 Pleasant St., Belmont, Mass.

26. R. M. Fano, "The MAC System: The Computer Utility Approach," *IEEE Spectrum,* vol. 2, pp. 56-64 (Jan. 1965).

27. P. M. Morse, "Computers and Electronic Data Processing," *Industrial Research,* vol. 6, no. 6, p. 62 (June 1964).

28. T. Kilburn, "One-Level Storage System," *IRE Transactions on Electronic Computers,* vol. EC-11, no. 2 (Apr. 1962).

29. A. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Comm. ACM,* vol. 4, pp. 422-431 (Oct. 1961).

30. J. B. Dennis, "Program Structure in a Multi-Access Computer," Tech. Rep. No. MAC-TR-11, Project MAC, M.I.T., Cambridge, Mass. (1964).

31. J. B. Dennis, "Segmentation and the Design of Multiprogrammed Computer Systems," *IEEE International Convention Record,* Institute of Electrical and Electronic Engineers, New York, 1965, Part 3, pp. 214-225.

32. J. B. Dennis and E. L. Glaser, "The Structure of On-Line Information Processing Systems," *Information Systems Sciences: Proceedings of the Second Congress,* Spartan Books, Washington, D.C., 1965, pp. 1-11.

33. J. W. Forgie, "A Time- and Memory-Sharing Executive Program for Quick-Response On-Line Applications," this volume.

34. M. N. Greenfield, "Fact Segmentation," *Proceedings of the Spring Joint Computer Conference, 21,* Spartan Books, Baltimore, 1962, pp. 307-315.

35. "The Descriptor," Burroughs Corp., 1961.

36. "Univ. of Mich. Orders IBM Sharing System," *EDP Weekly,* vol. 6, no. 5, p. 9 (May 24, 1965).

37. B. W. Arden et al, "Program and Addressing Structure in a Time-Sharing Environment" (submitted for publication).

38. *Computing Report for the Scientist and Engineer,* Data Processing Division, I.B.M. Corp., vol. 1, no. 1, p. 8 (May 1965).

39. W. T. Comfort, "A Computing System Design for User Service," this volume.

40. "Proposed Revised American Standard Code for Information Interchange," *Comm. ACM,* vol. 8, no. 4, pp. 207-214 (Apr. 1965).

41. P. A. Crisman, ed., *The Compatible Time-Sharing System:* A Programmer's Guide, 2nd ed., M.I.T. Press, Cambridge, Mass., 1965.

# SYSTEM DESIGN OF A COMPUTER FOR TIME SHARING APPLICATIONS*

E. L. Glaser
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*
and
J. F. Couleur and G. A. Oliver
*General Electric Computer Division*
*Phoenix, Arizona*

## INTRODUCTION

In the late spring and early summer of 1964 it became obvious that greater facility in the computing system was required if time-sharing techniques were to move from the state of an interesting pilot experiment into that of a useful prototype for remote access computer systems. Investigation proved computers that were immediately available could not be adapted readily to meet the difficult set of requirements time-sharing places on any machine. However, there was one system that appeared to be extendible into what was desired. This machine was the General Electric 635. The 635 is a single address stored program computer with a word length of 36 bits. It possessed many of the characteristics that were deemed necessary for the application of a computer to time-sharing. The three most important characteristics are:

1. A clean and comprehensive order code,
2. a multiprocessor capability, and
3. nonsynchronous design.

The first of these requirements stems from the quantity of software to be written for the machine. The size of the operating system demands it be written in some higher level language. An orderly instruction set is essential to permit the use of good code selection algorithms in the compiler. The multiprocessor characteristic was desired to permit a feasible fail-soft characteristic in the system and to allow system growth without major increments equivalent to entire system duplication. The third characteristic, non-synchronous communication between major components, was deemed desirable because of the flexibility afforded in the significant modification needed to achieve the time-sharing system that we had in mind. The nonsynchronous characteristic allows a system to become large without suffering measurable degradation. These modifications in a fully synchronous system could result in degradation of performance. (Degradation of

*Work reported herein 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).

course will take place in any system if the delay time of a signal through a cable is a significant part of a basic operating time. However the nonsynchronous approach permits this time to be minimized.)

## THE SCOPE OF EXTENSION

The design of the extensions to the 635 began in early May 1964 and the end result is what is now known as the GE 645. The changes are in several areas. First, a totally new I/O control unit has been designed to integrate the control of standard peripheral devices and various types of communications lines. The latter are necessary in the time-sharing environment. A large movable head disc, the DS 25, was available as a standard 635 peripheral. This unit appeared suited for the type of use we envisioned. A high-speed drum (DS 300) was also available, but its performance was not sufficient for the purposes of the highest speed secondary store in the projected time-sharing system. Therefore, a new high-speed drum system was designed for this function (MS 32). The introduction of a new form of addressing logic incorporating segments and pages is a significant change to the system. This, with its concomitant changes in interrupt logic and related portions of the machine affected by it, was by far the major change to the system.

## THE SEGMENTS AND PAGES

The concept of the paged memory has appeared in the literature for the past several years and has been implemented on at least one machine[1,2]. The purpose of paging is to make the allocation of physical memory easier. One can think of paging as the intermediate ground between a fully associative memory, having each word addressed by means of some part of its contents, and a normal memory, having each memory location addressed by a specific integer forever fixed to that physical location. In paging, blocks of memory are assigned differing base addresses. Addressing within a block is relative to the beginning of the block. Thus if association and relative addressing are handled with a break occuring within a normal break of the word (viz. in a binary machine block size is a power of 2), then a number of noncontiguous blocks of memory can be made to look contiguous through proper association. The association between a block and a specific base address can be dynamically changed by program during the execution of appropriate parts of the executive routine.

Segments on the other hand are used not for the allocation of physical memory, but for the allocation of address space. The concept of segments has received little attention in the literature until recently.[3,4,5,6] A segment defines some object such as a data area, a procedure (program) or the like. In a sense, each segment corresponds to a virtual memory whose size is whatever size, up to a maximum limit, that is required. In theory, as many such segments can be available to a programmer as necessary. In the case of the 645, the practical limit is $2^{18}$ segments, each one of which can obtain up to $2^{18}$ words. Observe that although segments and pages are two distinctly different entities, they work together to facilitate the allocation of physical memory and virtual memory. Although a large number of segments may be defined, each one having a large number of words, only the currently referenced pages of pertinent segments need to be in memory at any time. A very limited concept of segments has been used in computers previously.[7] Recently other system designs have employed a similar segment and paging technique.[8]

## DESCRIPTORS

A descriptor is a word that is used to define and locate in physical memory either a page or a segment. Hence there are two kinds of descriptors: page descriptors and segment descriptors. The difference between them is the table in which they are found. Each has slightly different functions as will become obvious.

A segment descriptor contains among other things the location of either the segment itself or, if the segment is paged, the location of the table in which its pages are defined. Each page descriptor corresponds to one of the pages of the segment. A page descriptor contains the location of the base of the block of memory in which this page is to be found. All of the page descriptors for a given segment must lie in contiguous locations of the page table for that segment.

Both segment and page descriptors also contain certain access control information known as the descriptor control field. These fields define the nature of access permitted to a particular piece of information. An example of such a control might be the Write Permit Bit. This bit determines whether

this segment or this particular page of a segment can be written into or only read. Alternatively, we may think of the segment descriptor as defining a certain set of restrictions on accessing the entire segment. Specific page descriptors may add additional restrictions; however, they may not take any away. For example, the segment is defined as being a data segment with writing permitted. The result is that control can not be transferred to this segment, but the words of the segment can be used as data with either reading or writing possible. If for some reason there is one specific page of data that we wish to protect, that page can be marked as "Read Only" (the write permit bit is set to zero). As a consequence, this page is defined not only as data but it is now "Read Only" data. The page descriptor has applied an additional constraint above and beyond those constraints contained in the segment descriptor. In addition to the control and address information the segment descriptor contains a bounds field. This bounds field defines the number of pages that the segment contains. In the case of unpaged segments this bounds field defines the total number of words in the segment.

## THE DESCRIPTOR SEGMENT AND BASE REGISTERS

The segment descriptors associated with a given process are all contained in a single segment known as the descriptor segment. The descriptor segment has a distinguished role in the operation of the system in that the processor uses this segment as the sole means of relating program references to memory location. The descriptor segment may be paged in the same manner as any other segment. Its location in memory is defined by a special processor register known as the descriptor base register. This register defines either the base of an unpaged descriptor segment or the base of the segment page table of a paged descriptor segment. This register can only be loaded or stored by privileged instructions not available to slave mode programs. Note that all user programs and most of the executive system are written in slave mode.

A segment can now be identified by an ordinal number which locates its descriptor relative to the base of the descriptor segment. This number is known as the segment number. The address of a location in memory is specified in terms of a segment number and a location within that segment. In the 645 both quantities are expressed as 18 bit numbers. During all addressing in the 645 both parts are necessary except under very unusual circumstances. Both parts are usually supplied explicitly. In some cases they are implied by certain conditions of the machine.

There are several provisions for forming these two-part addresses. The first is by means of the instruction word. An address may refer to a location within the current procedure segment or alternatively to some other segment. A control field in the instruction specifies the choice. If the reference is within the current procedure segment the segment number is found in the procedure base register. This is an internal processor register and not directly accessible to the user. If a reference is to some other segment the segment number is located in one of eight address base registers. The three most significant bits of the instruction address field are used to specify this selection.

A set of commands is available to load, store and modify the contents of the address base registers. Additional flexibility is provided by allowing these base registers to operate as index registers (internal bases). When employed in this manner the indexing base register is coupled with a base register that holds a segment number. This base pair in effect defines a base location internal to the segment. The association of bases, together with marking bases as internal, and "locking" certain bases so that they cannot be changed except in a privileged "master" mode, are all contained in a control register for the bases.

For high-speed storing and reestablishing of status, it is possible to store or load the eight base registers with a single command. Any bases which are locked, on a load bases command are passed over and remain unchanged. The use of these various forms of addressing would be difficult to discuss at this time and the reader is referred to one of the subsequent papers in this session describing the software for the new Multics system.[9]

A second form of segment addressing is made available by means of the indirection facilities in the 645. Two variants are proveded. The first of these is known as INDIRECT TO SEGMENT (ITS). This form of modifier requires two words, the first of which gives the segment number together with the tag that indicates it is an ITS word. The second word appears as a normal indirect word in either the 635 or 645. It contains an address to-

gether with an address tag indicating whether further indirection is to take place and if so what type, and additionally if indexing is to take place before using this address. The second variant is by means of the indirect word pair which is called ITB, that is, INDIRECT THROUGH BASE. This form of indirection is identical to the ITS form with the exception that the first word of the pair contains the ITB modifier and a number from 0 through 7 which indicates which of the 8 address base registers contains the segment number of this address. When either form is encountered during indirection, the segment name then in effect is canceled and replaced with that given by the pair. The indirection will continue as prescribed by the pair.

## THE THREE MODES OF PROCEDURE EXECUTION

In the 635 there were two modes for program execution: namely, master and slave. In slave mode only a restricted set of processor instructions are executable. Certain instructions such as I/O connect, those instructions dealing with the loading of the elapsed-time register and instructions affecting the relocation register were trapped. In master mode these privileged instructions could be executed and the relocation feature disabled.

In the 645 three distinct modes of execution are defined. These are absolute, master and slave. Slave mode is considered to be the normal mode of instruction execution. In this mode no privileged instructions may be executed. Further the relocation logic for segments and pages is fully operative. Master mode uses the segment and paging hardware identically to slave mode with the exception that privileged instructions may be executed and certain constraints on the access to segments are removed. Absolute mode is superior to the other two modes of operation. In the absolute mode, the segmentation and paging hardware is disabled and all instructions in the machine may be executed. Additionally, none of the segment access restrictions apply. Absolute mode is entered only by the occurrence of an interrupt. The machine enters temporarily into absolute mode to record system status but can be caused to remain in this mode if desired. The segmentation hardware can be temporarily enabled on any instruction merely by the use of the instruction word control bit used to indicate base register selection. Further, encountering either an ITS or ITB modifier will cause the segmentation hardware to be turned on for this instruction execution. When in absolute, the mode of the program can be turned back to either master or slave by the execution of an appropriate instruction. This instruction is a branch instruction that defines a segment number which indicates what form of procedure shall be executed, that is, master or slave.

Because it was felt desirable to make it possible to branch easily between various programs including between slave and master programs, a certain degree of insurance has to be built into the hardware to guarantee that spurious branches would not take place into the middle of master mode programs from slave programs. As a consequence, a master mode procedure when viewed from a slave mode procedure appears to be a segment which can neither be written nor read. Further, the only method of addressing this segment that is permitted is a branch to the $0^{th}$ location. Any attempt to get at other locations by branch, execute, return or any other instructions will result in an improper procedure fault causing an appropriate interrupt. A special form of procedure called EXECUTE ONLY has also been defined which is similar to MASTER PROCEDURE in terms of entry restrictions imposed on slave mode. Once entered, this procedure has all of the execution characteristics of slave mode.

## THE ASSOCIATIVE MEMORY

The addressing system as now defined would be very unsatisfactory if employed for each instruction or operand reference. If both the descriptor segment and the data segment are paged and if the procedure being executed is paged, a large number of memory cycles might be required to develop a memory address. To overcome this an associative memory is incorporated in the processor. This memory "captures" a compounded descriptor, derived from the segment and page descriptors. The resultant working descriptor represents a particular page of a particular segment. This can be either a data segment or the procedure segment. As a consequence, if a particular page of a segment is being used quite heavily, its descriptor will always be in this associative memory and no additional references to main memory are required to develop the memory address. The associative memory is "invisible" to the user. Its only effect is to greatly speed

up the execution of the programs. Whenever a new working descriptor is created, it is placed in the associative memory. If the associative memory is already full, a wired algorithm selects a memory position to be used for the new descriptor and causes an older descriptor to be discarded. A set of commands permit storing selected words from the associative memory and clearing the associative memory. All of these instructions are privileged.

## ADDITIONAL AIDS TO MEMORY ALLOCATION

The environment in which this system is to work places a high premium on efficient management of memory resources. Paging of itself simplifies the allocation process. A further gain is possible if one appreciates the effect of frequency and duration of usage. Two distinct mechanisms are employed to supply this information. The first of these involves the page descriptors. A record is made in the descriptor if the page is accessed for any reason. Once this bit has been set to a one, it is unaffected by subsequent page references. A supervisor program will periodically reset these "use bits" to zero and at the same time determine which pages have been accessed since the last entry into this procedure. A second bit in each of the page descriptors is set to one if the contents of the page is altered in any way.

The second mechanism involves the associative memory. The privileged instructions that store the contents of the entire associative memory or store the contents of the cell whose contents are "the oldest" descriptor provide the supervisory program with a measure of the extent and frequency of page usage.

## INTERRUPT CONSIDERATIONS

In the 645 interrupts are generated by external stimuli while faults are generated by processor conditions. The occurence of either an interrupt or fault causes the execution of two commands located at specific "wired" addresses. On most computers, interrupt can only take place between command executions. In the case of the 645, certain aspects of segmentation made desirable the interruption of the computer at many points during execution of a specific command. After the interrupt is serviced, execution is resumed. Resumption at the precise point of interruption, rather than restart, is mandatory because of the nature of the indirection in the 645.

The features of the segmentation system which first made it mandatory to add this more generalized interrupt capability were associated with the various control checks implied by the descriptors. Examples of these control checks are the bounds check, attempting to write in a Read Only segment, etc. It is advantageous in a segmented environment to cause an interrupt as a result of accessing an appropriately coded segment or page descriptor. This type of interrupt or fault is called a Directed Fault. Its name is derived from the fact that this descriptor directs control to a specific function based on which one of 8-bit configurations is found in the segment or page descriptor. The encoding and placement of this type of descriptor is done by the supervisor. The use of these descriptors for marking missing pages or missing segments will be discussed in a subsequent paper.

The 645 interrupt handling mechanism has been called the "snapshot" register. This is a set of flip-flops which, although used by other functions of the machine, are primarily available for storing the machine state. A trap, be it either an interrupt or a fault, causes the state of the machine to be stored in this snapshot register. The contents of essential registers and a history of control states comprise the snapshot. The first instruction in the interrupt handling routine normally will be a store control unit instruction. This privileged instruction stores the contents of the snapshot register into six memory locations. The subsequent execution of a restore control unit instruction takes the contents of the six words and reestablishes the control unit.

## CONFIGURATION CONTROLS

Because of the highly on-line nature of this system it is necessary to reconfigure the system relatively easily. As a consequence, those switches required for reconfiguration control are remotely located from the units they control to allow rapid setting from a central point. At first glance it might seem desirable to make program configuration possible; however, if the system is malfunctioning it would be necessary in any event for the program to notify and probably obtain permission from the floor supervisor. As a consequence, it was decided initially to make reconfiguration controllable by manual switches and to

allow the computer to instruct the operator during reconfiguration.

Reconfiguration is used for two prime purposes: to remove a unit from the system for service or because of malfunction, or to reconfigure the system either because of the malfunction of one of the units or to "partition" the system so as to have two or more independent systems. In this last case, partitioning would be used either to debug a new system supervisor or perhaps to aid in the diagnostic analysis of a hardware malfunction where more than a single system component were needed.

The effectiveness of rapid reconfiguration is difficult to determine in a paper simulation and efficiency of the system chosen will only be proved or disproved after a number of months of practical use in the one-line environment.

## CONCLUSIONS

We have attempted in this paper to describe some of the considerations that led to the unique design of the GE 645 as a processor for a multi-access, remote user, information processing system. We have already learned much in the process of designing this machine and feel that within the next two to three years much more will be learned. It is difficult at this time to make any statements about what the future of such processors should be beyond a few tentative conclusions.

First, additional speed both in arithmetic capability and in the memory hierarchy is desirable still; but even more, increased channel capacity of the main store of the machine is required. At present it appears that the principal limitation and expansion of the system will be the channel capacity of core memory. Obviously, this will be alleviated to a great extent by the advent of higher speed central memories for computers. However, this is but one answer. We feel that superior facilities can be gained by closer attention to the system functions that we have emphasized with the 645; namely efficient interrupt handling capability, and comprehensive addressing logic to improve the allocation and protection of physical and logical memory.

Finally, it is felt that the designer of future time-sharing systems must remember that a main part of the system is not in the computing center. Rather it is composed of the communications lines, the terminals and the various users, be they human beings, experiments or the like at these terminals. Therefore, the designer must keep in mind that he is engaged in a communications activity as well as an information processing activity and that proper attention must be paid to both aspects. We have done this to the best of our ability in the present system, although we are sure that some several years from now we will be able to return with the description of a machine that will be as great a step over the 645 as the 645 is over previous designs when applied to this new emerging field of time-shared computation.

## REFERENCES

1. F. H. Sumner, "The Central Control Unit of the Atlas Computer," *Proc. IFIP Congress,* 1962, pp. 291-296.

2. J. Fotheringham, "Dynamic Storage Allocation in the Atlas Computer," *Comm. ACM,* vol. 4, no. 10, Oct. 1961, pp. 435-436.

3. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Comm. ACM,* vol. 4, no. 10, Oct. 1961, pp. 422-431.

4. M. N. Greenfield, "Fact Segmentation," *Proc. SJCC,* vol. 21, May 1962, pp. 307-315.

5. J. B. Dennis and E. B. Van Horn, "Nesting and Recursion of Procedures in a Segmented Memory," Project MAC Memo, M-187, M.I.T., Oct. 1964.

6. J. B. Dennis and E. L. Glaser, "The Structure of On-Line Information Processing System," *Proc. of the Second Congress on Information System Sciences,* Spartan Books, Inc., Washington, D.C., 1965.

7. "The Descriptor," Burroughs Corp., 1961.

8. "IBM 360, Model 67, Computing Report for the Scientist and Engineer," *1,* 1 (May 1965) p. 8, Data Processing Division, I.B.M. Corporation.

9. V. A. Vyssotsky, F. J. Corbato, R. M. Graham, "Structure of the Multics Supervisor," this volume.

# STRUCTURE OF THE MULTICS SUPERVISOR

V. A. Vyssotsky
*Bell Telephone Laboratories, Incorporated*
*Murray Hill, New Jersey*
and
F. J. Corbató, R. M. Graham
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*

## INTRODUCTION

This paper is a preliminary report on a system which has not yet been implemented. Of necessity, it therefore reports on status and objectives rather than on performance. We are impelled to produce such a prospectus by two considerations. First, time-sharing and multiprogramming are currently of great interest to many groups in the computing fraternity; a number of time-sharing systems are now being developed. Discussion of the issues and presentation of goals and techniques is valuable only if it is timely, and the appropriate time is now. Second, every large project undergoes a subtle alteration of goals as it proceeds, extending its aims in some areas, retracting them in others. We believe it will prove valuable to us and others to have on record our intentions of 1965, so that in 1966 and 1967 an unambiguous evaluation of our successes and failures can be made.

The scope of this paper is an operating system in the strict sense. It is only slightly concerned with the hardware of the GE 645, for which the system is now being implemented. It is equally little concerned with the translators and utility programs which make the system useful for computing. Furthermore, this paper pays little attention to the file system, which is the largest single component of the operating system, including well over half of the total code. A separate paper is devoted to the file system.

Much of the content of this paper is statements of mechanisms or techniques for achieving particular goals. In very few cases do we discuss proposed alternative methods, or our reasons for choosing particular methods. Such discussion would require an extended treatise; such a treatise might be useful, but it does not exist, and is not likely to. We hope to produce fragments of it in the future. In every case, our choice of method is based on one or more of four criteria. First, some of the mechanisms were adopted from previous systems because they proved satisfactory there. Second, alternative solutions to some of the problems were tried on previous systems and found unsatisfactory. Third, in some cases the merits and defects of alternative

methods have been vigorously debated and subjected to gedanken experiments; the chosen method was that which appeared most satisfactory (or least unsatisfactory). Finally, many approaches were chosen because they are evidently workable and are well aligned with the overall approach advocated by our firmly opinionated planning group. The strongest opinion of our planning group is that consistency is a virtue, and that general solutions are better than particular ones.

## VIEWPOINTS AND OBJECTIVES

We view an operating system as an evolving entity. Every operating system with which we have been associated has been greatly modified during its useful life. Therefore, we view the initial version of Multics not as a finished product to be cast in concrete, but as a prototype to be extended in the future. In two ways this is an unhappy conclusion. Users (except those users who benefit substantially from a particular change) tend to resent bitterly any fluidity in the tools with which they must work. System programmers become satiated with reworking programs which they would like to forget. However, the one thing which most users resent more than a fluid system is a frozen system inadequate to the users' expanding needs. So the system must evolve.

Therefore, one of the primary objectives of Multics is that it shall include any features that we can clearly discern to be useful in allowing future changes or extensions to be made with minimum effort and minimum disruption of existing applications. The initial cost of including such features is substantial. We believe from past experience that the initial cost will be more than repaid in reduced future cost of reworking both the operating system and the application programs that use the system.

We view the operating system as having an ill-defined boundary. The software field is replete with examples of user installations or individual application programmers using a cutting torch and jack hammer to break into a neatly defined software package. The effort involved in many such cases is so large as to constitute prima facie evidence that the job was not done for frivolous reasons.

Therefore, Multics is designed to be a single-level system. Most modules of the operating system itself are indistinguishable from user programs, except that they are guarded against unintended or ill-advised changes by protective locks administered by the user installation. Changes to the operating system can therefore be made by the same techniques as are used to change user programs. A programmer who wishes to change a module of the operating system must be authorized to do so. He does not, however, need a large "system edit" program, since the format and conventions of operating system modules are the same as those of user programs.

We view a large open-shop computer facility as a utility like a power company or water company. This view is independent of the existence or non-existence of remote consoles. The implications of such a view are several. A utility must be dependable, more dependable than existing hardware of commercial general-purpose computers. A utility, by its nature, must provide service on demand, without advance notice in most cases. A utility must provide small amounts of service to small users, large amounts to large users, within very wide limits. A utility must not meddle in its customers' business, except by their request. A utility charges for its services on some basis closely related to amount of service rendered. A utility must provide its product to customers more cheaply or more conveniently than they could supply it for themselves. Most important of all, a utility must provide service to customers who neither know nor wish to know the detailed technology employed by the utility in providing the service.

All of these considerations save played a role in the design of Multics. The file system contains elaborate automatic backup and restart facilities to make the dependability of information storage within the system greater than the dependability of the media on which the information is recorded. The operating system is designed to be dynamically adjustable to compensate for temporary loss of one or more hardware modules. Multics is designed to provide service without batching or prescheduling, although prescheduling facilities will be provided for runs whose size and urgency dictates such treatment. Multics employs allocation and scheduling algorithms intended to allow small and large jobs to flow through the machine together, without differentiation, with any special priorities supplied by human beings on the basis of urgency of jobs (or categories of jobs), rather than built-in priorities based on size or type of job. An explicit criterion of Multics is that computation center personnel shall not be required to take cognizance of, or perform any action whatsoever for, a routine job which

does not demand unusual facilities. Multics is intended to accommodate within it standard (but replacable) charging and accounting routines. Multics will accommodate a variety of input-output terminals, ranging from Teletypes to line printers to laboratory measuring equipment for the convenience of its users. The scheduling and allocation algorithms are intended to run the installation with low housekeeping overhead, especially when the load is heavy.

The most important consideration is the one which Multics seems least likely to meet to the satisfaction of its designers. Most of the ultimate users of a large-scale computer have no interest whatsoever in computers or computer programming, let alone the details of particular machines, programming language and operating systems. They have problems to which they wish answers, or data they wish transformed or summarized in some particular way. No computer shop can be considered to function satisfactorily as a utility unless the users can get results without having to formulate the problems in an alien notation. In other words, the system should be sympathetic to its users. Multics provides no direct assistance toward this goal, and little indirect assistance. Neither can any amount of evolution of algebraic languages offer much assistance, since they are still programming languages closely reflecting the structure of a digital computer, and most users are not interested in programming computers in the first place. Progress in this area will require extensive effort in analysis of particular application fields, and development of specialized program packages relevant to the specialized needs of the application fields. The only assistance Multics provides is a framework within which a user can conveniently interact with such a specialized package if it exists, and a measure of isolation from detailed hardware eccentricities which should very substantially ease the life of programmers developing such packages.

We consider privacy of user information to be vitally important. In many applications it is essential that all authorized personnel, and no unauthorized personnel, should have easy access to programs and data. Multics provides, in its hierarchial file structure and its protection mechanisms, very substantial aids to privacy. These aids, when intelligently used, should provide virtual certainty that unintentional privacy violations will not occur, and should provide excellent protection against intentional, ill-advised, but unmalicious attempts to access or modify private information without permission. Multics does not safeguard against sustained and intelligently conceived espionage, and it is not intended to.

## ADMISSIBLE HARDWARE CONFIGURATIONS

The minimum hardware configuration with which 645 Multics can run is one 645 CPU, 64K of core memory, one high-speed drum or one disc unit, four tape units, and eight typewriter consoles. However, Multics will not run efficiently on this minimum configuration, and would normally be operated thus only when a substantial part of a larger configuration was unavailable for some reason.

A small but useful hardware complements would be 2 CPU units, 128K of core, 4 million words of high speed drum, 16 million words of disc, 8 tapes, 2 card readers, 2 line printers, 1 card punch and 30 consoles.

The initial implementation of 645 Multics software is designed to support a maximum configuration of up to 8 CPU's, up to 16 million words of core, up to 2 high speed drums, up to 300 million words of disc and disc-like devices, up to 32 tapes, up to 8 card readers, 8 punches, 16 printers, and up to 1000 or more typewriter consoles. It will not, of course operate efficiently (or in some cases at all) with an arbitrary and unbalanced mixture of these. For instance, 645 Multics would not run well with 6 CPU's and 128K words of core.

## TECHNICAL POLICY FOR WRITING SOFTWARE

As stated earlier, Multics is intended to be a single level system, and an evolving system. In spite of evolutionary tendencies, 645 Multics must be a useful product and it is to be in operational use in 1966. These factors combine to motivate a small but crucial body of technical policy for system programming. This technical policy differs from standards of good practice in that technical policy is mandatory and enforced upon system programmers working on 645 Multics, and requests for exceptions are skeptically reviewed by project supervision.

Absolute mode (execution without relocation of addresses) is used only

a) for the first two instructions of each trap-answering routine
b) for startup of a cold machine
c) for the initial stages of catastrophe recovery (e.g., recovery from a trouble fault), and
d) for appropriate product service routines (hardware test and diagnostic routines).

Master mode (execution with unrestricted access to privileged hardware features) is used only

a) for absolute mode execution
b) to exercise privileged hardware features
c) where temporary disabling of all interrupts is required, and
d) for appropriate product service routines. Code which is written in master mode because its purpose is to exercise privileged hardware features will be written as standard subroutines. Each such subroutine may perform only one function (e.g., issue an I/O select). Each such subroutine will check the validity of the call.

All operating system data layouts for the initial implementation of 645 Multics will be compatible with data layouts used by PL/I, except where hardware constraints dictate otherwise. All modules of the initial implementation of the operating system will be written in PL/I, except where hardware constraints make it impossible to express the function of the module in the PL/I language.

All procedures and data will be usable paged to 64 words, paged to 1024 words, or unpaged, except for vectors and data blocks which are inherently unpaged because of direct hardware access to them.

Since the PL/I translator which will be used until mid-1966 generates inefficient object code, it is clear that 645 Multics in its first few months of existence will be inefficient. This penalty is being paid deliberately. After mid-1966, two courses of action will be available: upgrade the compiler to compile more efficient code, or recode selected modules by hand in machine language. We expect that both strategies will be employed, but we expect to place preponderant emphasis on upgrading the PL/I compiler; indeed, one subsequent version of PL/I is already being implemented, and a second is being designed.

## PROCESSES

In Multics the activities of the system are divided into *processes*. The notion of process is intuitive, and therefore slightly imprecise. To convey the notion we shall talk around it a bit, and then give a reasonably exact definition.

When a signal from the external world (e.g., a timer runout signal) arrives, and a CPU interrupt occurs, what is being interrupted? Presumably a "run." Observe that if a program is defined in the usual way as a procedure plus data, there is no meaning to the phrase "interrupt a program," if it is taken literally. What is interrupted is the execution of a program. In a time-sharing system this distinction becomes so important, and ignoring the distinction is so pernicious, that we shall use the word "process" to denote the execution of a program, and reserve the word "program" to denote the pattern of bits (or characters) which the hardware decodes.

In most cases a process corresponds to a job, or run; it is a sequence of actions. Consider for example the sequence of action: build a source program, compile it, execute it and the programs it requires, produce output files including postmortem information and accounting data. This sequence of actions would typically be a single process in Multics.

If the notion of process is to be useful, it must be possible, given some action, to determine to which process it pertains; that is, it must be possible to distinguish unambiguously between processes. In 645 Multics we base our distinction on descriptor segments. At any given moment a 645 CPU is using one and only one segment as the descriptor segment. At different times the CPU may use various different descriptor segments. We define a process to be all those actions performed by a CPU with some given segment as descriptor segment, from the first time that segment becomes the descriptor segment until the last time the segment ceases to be the descriptor segment. Thus a process has a very definite beginning; if it ends, it has an equally definite end.

For each process there is in addition to the descriptor segment a stack segment, for the user's programs and most supervisory routines, and a concealed stack segment, used by some supervisory routines to hold information such as charging data, which must be safeguarded against garden variety user program errors. There are also any other seg-

ments (including supervisory segments) which are required by the process. For each process there will typically be many segments, containing the user and supervisor programs and data, but most of the segments will be attached to the process only as they are dynamically required.

Since we have already observed that almost no procedures will run in absolute mode, and since the operational definition of process places all master mode and slave mode execution firmly in some process, it follows that almost all CPU activity occurs as part of some process. Most processes will be initiated by customers and charged to customers. Some processes will be initiated by the installation and charged to overhead. An example is a process which purges a disc unit.

## STATUS OF A PROCESS

Any process that exists in 645 Multics is either running, ready, or blocked. A process is *running* if its descriptor segment is currently being used as the descriptor segment for some CPU. A process is *ready* if it is not running but is not held up awaiting any event in the external world or in another process. A process is *blocked* if it is awaiting an event in the external world or in another process (e.g., arrival of input data, or completion of output, or 3 PM, or retrieval of a page from drum, or release of a data file by another process).

## SEGMENTATION, PAGING AND ADDRESSABLE STORAGE

A general principle in Multics is that programs are written to reference locations in addressable storage, rather than locations in core. An address consists of a segment number and word number. The address of an item is clearly important to the program, and possibly to the programmer. Therefore, in Multics the division of programs and data into segments, and the sizes, names and types of the segments, are controlled (explicitly or implicitly) by customers and customer processes.

Paging, on the other hand, is considered in Multics to be the responsibility of the operating system. The view of the designers of Multics is that provided the customer gets his answers when he wants at the price he expects to pay and agrees to pay, it is none of his business where in core his programs and data resided—nor, indeed, whether they were in core at all. The 645 hardware was designed with this philosophy, and the software is built to implement this approach.

However, in some real-time applications it is demonstrable that the application cannot be correctly implemented unless certain programs and data are in core when external signals arrive. In some other applications reasonable efficiency may be attainable only if the user program can specify explicitly what should be in core at which stages of execution. Therefore, calls to the paging routines are provided for specifying:

a) that certain procedures and data must be "bolted to core" in order for the application to run,
b) that certain material is going to be accessed soon, and should be brought into core if possible,
c) that certain material will not be accessed again, and may be removed from core.

It is expected that few application programs will need to make use of such calls.

The paging routines will normally operate with only three sources of input information.

The pager will know when a page must be brought into core by the fact that a page-not-in-core fault occurs. It will know which pages are candidates to be removed from core by a usage measure it derives from the "used" bit of each page table entry, and by a specification in the core map of whether the page is accessed other than through a page table (e.g., a page which is itself a page table, and therefore is referenced directly by CPU hardware). The pager will also know from specifications in the core map which pages may not be removed from core at all (e.g., because they are currently attached to peripheral devices).

A program such as the linker will deal with addressable storage, and will not consider the place of physical residence of any procedure or data block in establishing a linkage. If the linker happens to access information which is not in core, the pager will be invoked by a page-not-in-core fault, the process in which the linker was working will be blocked until the page arrives, and will then be ready to resume.

## SEGMENTS AND FILES

In 645 Multics, every segment is a file, and every file is a segment. A reference to one of these ob-

jects, however, may be made in two distinct ways: by segment referencing and by file referencing. Segment referencing is, by definition, referencing by means of a 2-component numerical address, each component consisting of 18 bits, of which the first component specifies a word number in the descriptor segment and the second specifies a word number in the referenced segment. File referencing is anything else. Every file is a segment to some procedure in some process at some time. Any file reference which results in retrieval or modification of any part of the contents of a file (except retrieval, replacement or deletion of the entire file) is a call to a procedure which references the file by segment referencing. Thus, the question of whether a data object is a segment or a file is a question about the viewpoint from which some particular procedure sees the file.

Segments (files) come in two varieties: bounded segments and unbounded segments. A bounded segment is a segment which is guaranteed to consist of $2^{18}$ words or less. An unbounded segment may have any number of words (e.g., 27), but is not guaranteed to have no more than $2^{18}$. You have to look at it to find out. Segment referencing using the appending hardware can only be done directly for bounded segments. To each unbounded segment there may be associated a bounded segment called a "window"; the origin of the window segment may be set, by a supervisor call, to any 1024 word boundary in the unbounded segment. More than one window segment may be attached to a single unbounded segment, if desired, and the windows may be adjusted independently. In principle, the size of an unbounded segment could be arbitrarily large. However, the software of 645 Multics will limit the size of unbounded segments to $2^{28}$ words, and in some installations storage limitations will hold the maximum segment size even below $2^{28}$ words.

## PERIPHERAL DEVICES AND FILES

In 645 Multics, one of the kinds of file given special recognition will be the serial file. In 645 Multics, unit record equipment and typewriter-like consoles will be treated as serial files of restricted capabilities. User programs will be able to know that such hardware units are not serial files, but it will not normally be advantageous to make use of that fact, and to use such knowledge may severely restrict the applicability of a program. If a program handling a peripheral device as a serial file attempts to perform an illegal primitive (e.g., rewind a card reader), then either

a) the effect on all ensuing processing will be as if the primitive had been performed successfully (e.g., the input file copied from the card reader will be rewound) or
b) a diagnostic wil occur (e.g., skip to the end of file on typewriter input).

The effect of treating peripheral devices as serial files is to make it possible for many programs to run either with a typewriter console as a peripheral device or with the console replaced by files on secondary storage.

## SCHEDULING

In Multics the system is regarded as having a pool of anonymous CPU's; scheduling and dispatching procedures are executed by each CPU when it must determine what to do next. The only result with any operational meaning that can ensue from scheduling and dispatching in Multics is that CPU number $n$ resumes process $p$ at time $t$ . Furthermore that process must have been in ready status.

We shall state here some fundamental assumptions concerning scheduling which appear evident to us, but some of which are not universally accepted. The goal of scheduling in an open-shop general purpose computer system is to give good service to customes at reasonable cost. When the offered load is greater than system capacity, it is impossible to give good service to all those who desire it. Therefore, on an overloaded system, scheduling should be done so as to minimize overhead and to complete the most urgent work first. Two basic techniques for minimizing overhead are to employ service denial rather than service degradation, and to minimize the number of times control is switched from one process to another. That is, it is more efficient to serve a few users at a time and do it well than it is to serve all users poorly at once. A job is urgent, in the last analysis, because it is costing someone time and/or money not to have the results. The urgency of a job is only slightly correlated, if at all, with the extent of its demands on such system resources as CPU time, core storage, secondary storage, and peripheral facilities. Hence, urgency of work must be determined by human beings, not by the computer.

If offered load is less than system capacity, it is possible in principle to give good service to all who desire it. It may not be possible, however, to achieve satisfactory service for all and still keep the percentage of overhead low. A moderate increase in overhead on a lightly loaded system is acceptable if the increase permits improved service.

Switching between processes is mandatory when a given process becomes blocked. Switching is done at other times to meet explicit or implied service guarantees. For example, placing a typewriter in a customer's office implies a guarantee that response times to simple requests will usually be short. Therefore, frequent switching between processes makes excellent sense when offered load is light, although not when offered load is heavy.

Offered load will rarely be well-matched to system capacity. Any general-purpose open shop computing installation where offered load is at the same approximate level at 3 a.m. Sunday and 3 p.m. Wednesday is either employing load flattening measures outside the computing shop (e.g., by human prescheduling) or is so heavily overloaded that offered load is almost always above system capacity, and service denial is the rule of the shop.

We believe that a general-purpose open-shop computing facility which is never (or almost never) overloaded is spending too much money for computing hardware. It is cheaper to accept occasional overloads. Further, we believe that any scheduling technique for a time-shared multiprogrammed computer system which behaves satisfactorily during overload will require at most a very slight modification to behave well under light load.

Hence, we contemplate an environment in which offered load is almost always either substantially above or substantially below system capacity. We believe that scheduling algorithms should be designed with good performance during overload as the primary objective, and good performance when load is light as a criterion to be met within the framework imposed by the overload design. The scheduler should get information concerning urgency of jobs from human beings, and should not have any built-in assumptions that console jobs are either more or less urgent than absentee jobs, or that short runs are either more or less urgent than long runs.

Unfortunately, in a multiprogrammed time-sharing system with dynamic storage allocation neither the machine nor human beings can determine directly how large the offered load is. How, for example, could one tell how many people at typewriter consoles would type messages if you unlocked their keyboards? Similarly, it is not possible in most cases to predict with any accuracy what demands a given process will make upon system resources during its next few seconds of running. Therefore, the scheduling algorithm must base its action on measurable quantities related to the unmeasurable offered load.

Several such measurable quantities are conveniently available. The most important of these appears to be a running measure of the rate of progress toward completion of processes, compared with a "satisfactory" rate of progress determined by information supplied by human beings about types of procsses or individual processes. For example, if there are exactly six processes to be considered each requiring 20 seconds of CPU time and no I/O, all with desired completion time 3 minutes away, and if in one second each process has received 100 milliseconds of CPU time, then each process at its current rate will require 3 minutes 20 seconds to complete. Presumably the system is overloaded, and one or more of the processes should be postponed. This is a fairly typical example; overloads in a system with dynamic storage allocation tend to become manifest by excessive overhead rather than by excessive visible demand. The scheduling algorithms for Multics will rely heavily on this fact.

The choice of which processes to postpone depends on several factors. If some processes have higher priority than others, the lower priority processes will be postponed. If, in the lowest priority class which will continue to run, some processes have been prescheduled for given completion times or computing rates, the prescheduled processes will be given preference. Finally, to make a choice among processes otherwise equal, the scheduler will prefer a process currently using expensive facilities (e.g., core) over one occupying inexpensive facilities (e.g., drum); the former is in some sense using more system resources than the latter, so it is desirable to move it toward completion.

## DYNAMIC LINKING

In Multics linking of one procedure segment to another, or of a data segment to procedures, is by and large done dynamically. That is, if a translator compiles symbolic intersegment references, these will not normally be replaced by numerical interseg-

ment references until the first time the reference actually occurs during execution of the compiled program.

The standard form of programs in Multics will be common shared procedure. Code run as common shared procedure may not be modified for execution of any one process. Hence, for each compiled segment of code there will be an accompanying linkage section, which will be maintained on a per-process basis, and all modifications required to link two segments together will be made in the linkage sections rather than in procedure segments. A linkage section contains, among other things:

(a) the symbolic (character string) name of each externally known symbol within the segment to which the linkage section belongs.
(b) for each symbolic reference from this segment to some other segment, the symbolic name of the foreign segment and the symbolic name of the referent within the foreign segment, plus an indirect word which is compiled with a tag that will cause a trap to occur when an indirection through it is attempted.

When a procedure is attached ot a process, the linkage segment of the procedure is copied into a data segment of the process. If the procedure during execution attempts to access a foreign segment by indirection through the linkage section, a trap ("linkage fault") will occur. At this time the linker will substitute the correct numerical value into the indirect word. The reference will then be completed; subsequent references, of course, will be completed without occurrence of a trap.

The original symbolic information is retained in the linkage section even after linking. Hence, it is possible to break such a link after it has been established, and detach a segment from a process. This will be done only upon explicit call to the unlinker, and is expected to be infrequent.

## TRAP HANDLING

The hardware traps on the 645 can be divided into two categories. In one category are process traps (e.g., overflow) which normally occur as a consequence of action in the running process. Handling of these traps will be done as part of the running process, by supervisory routines attached to the process. In the other category are system traps, some of which are relevant to some process but probably not one which is running (e.g., I/O termination), and others of which indicate hardware or software error (e.g., parity error in core).

Some of the process traps, such as the illegal procedure fault, will cause the process to be removed from running state after a bit of initial flailing around. The division between traps and system traps is not based on whether the running process will continue to run, but on whether the running process is known to be responsible for the trap.

What happens when a trap occurs? It varies somewhat, but generally speaking the status of the running process is stored in its concealed stack segment. Then, for system traps only, control switches to a special trap process. Then the concealed stack of the process (trap process for system traps, process which is still running for process traps) is pushed one level, and the appropriate trap-handling procedure is called. The supervisory routines have a standard trap-handling procedure for each trap, which discovers what caused the trap and takes appropriate action. However, for every trap there is at least one point in the trap-handling procedure where control will pass to some other routine in the process if the process is administratively entitled to provide alternative treatment for the trap. The extent to which customer processes can provide nonstandard trap handling is, of course, controlled by the installation, but it will by and large vary from complete freedom (for handling overflow) to very strict control (for handling page-not-in-core faults from the appending hardware).

Many traps will have several intercept points, corresponding to different causes of the trap. It should thus be possible for authorized processes to selectively modify the handling of every process trap. Only a restricted group of people will normally be able to modify handling of system traps, since these affect operation of the entire system. The technique for making the modifications, however, is the same as for process traps.

The work of the system trap process is to discover which processes are responsible for system traps. It must, for example, decode words in the status storage channels of the general I/O controller to find out what device caused an I/O interrupt, and then check status tables to discover which process issued the select that resulted in the interrupt. The

trap process can then bring the process responsible for the trap into ready status for further treatment of the particular interrupt; the trap process is then finished with that particular interrupt.

The process responsible for the interrupt may be a customer process; if not, it is a housekeeping process that behaves like a customer process. This process, when it enters running state, will resume in an interrupt routine exactly analogous to a process trap routine, complete with intercept points.

To a very high degree of approximation, all I/O for a process is handled within the process. This does not imply that I/O for each process is handled independently of I/O for other processes. The programs and tables involved in input and output are for the most part common to all processes requiring a given type of I/O activity, such as input from magnetic tape. These programs and tables, however, are attached to each of the processes which requires them, so that they can be called by normal subroutine calls.

This makes it possible to insert special I/O routines (e.g., for controlling a data line to a special-purpose device) in a particular customer process by taking only two actions: get administrative authorization to call relevant master mode routines and to intercept interrupts in the process, and then link to the I/O routines by calling with a standard call. However, this technique places stringent restrictions on timing-dependent I/O, and virtually eliminates the possibility of certain data-dependent I/O techniques. These restrictions appear to be reasonable in a system like Multics; we see no way to permit complete control of I/O by one user program without danger to other user programs.

## CREATION, BLOCKING AND TERMINATION OF PROCESSES

Every process begins by being spawned from some other process. In particular, certain system processes exist for no end except to recognize customers' identification and spawn new processes for the customers. However, any process may spawn others by an appropriate call to the operating system. The call specifies what segments the new process is to share with its parent, what segments it should receive copies of, what segments the new process should not know, and at what point the new process should resume.

A process may go into blocked state for many reasons, such as waiting for 3 p.m., or waiting for a page to arrive in core, or waiting for another process to release a file. In all of these cases, the process will indicate a particular flag which must be reset before the process can resume, and the presumption is that some other process (alarm clock routine in the scheduler, or system trap process, or process holding the file) will be cooperative enough to reset the flag. There is, however, no guarantee whatsoever that the flag will ever be reset.

It would be poor strategy to allow the blocked process to remain in limbo forever. Therefore, each process will have attached to it a maximum time for which it may remain continuously blocked. Multics will provide a default value of this time, but a customer may specify a value other than the default value for any particular process. A procedure in the scheduling process will occasionally scan the task list for processes which have been blocked for more than the allowable time. If one is found, a diagnostic message will be generated and shipped off to the error message file for the blocked process, if that can be found, and also to a standard system file. The blocked process will then be completely removed from the task list and, although its procedures and data are still intact, it will not resume if the condition on which it was waiting becomes satisfied. Human intervention is now required to retrieve it, either to attempt to resume it or to obtain diagnostic information. If such human intervention does not occur, the data segments of the process will eventually be purged from the system.

This is also the chain of events which occurs when a process violates some restriction. If, for example, a process attempts to execute a privileged instruction in slave mode, the standard trap procedure will generate a diagnostic message and then call a standard program to force out any relevant output. The process will then go into blocked state to allow a human attempt for further diagnostics or a fixup. If the attempt is not made, the process will then be removed from the task list, and eventually purged.

Termination of a process may occur in two ways. It may call a procedure in the operating system and say "I am through," or some other process may point at it and say "Get rid of him." The second method is used by the scheduler in disposing of processes which have been blocked for too long a

time. This second method may also be used by customer processes, subject to some restrictions.

Both methods may be employed with two degrees of severity. The process may merely be removed from the task list, or it may be marked as completely dead and subject to immediate purging from the system. In general, modules of the operating system will only remove a process from the task list if troubles occur, so that the customer may have a reasonable chance to come and rummage around in the procedures and data of the process to find out what happened.

## PROTECTION AGAINST MACHINE ERRORS

Like all other systems, 645 Multics will suffer from hardware and software failures. The goal of dependable operation can be achieved only if the effect of these failures can be limited. A companion paper discusses methods for safeguarding of data in the file system. Equally important and equally difficult is the problem of keeping the system on the air, or getting it back in a hurry, when a hardware failure occurs. This breaks down into two parts: how to run the system on a crippled machine, and how to share the machine with product service routines. We have no solutions to either problem, but some fragments of solutions are developing.

First, the policy of running the CPU's symmetrically is expressly intended to allow any CPU to be pulled at any time without stopping the system (although pulling a CPU at an arbitrary moment will undoubtedly wreck a particular process and some data files).

Second, the policy of minimizing absolute mode operation is designed to allow the system to resume execution with core banks missing with somewhat less agony than would otherwise be the case, and to allow the system to abandon a core bank with very little effort. I/O calls and fabrication of I/O data control words will be concentrated in a few procedures, with the explicit intent of allowing easy abandonment of a general I/O controller. For installations which can afford the luxury of using less than full core interlace, 645 Multics will provide the ability to pick up the pieces more or less automatically after loss of any one core bank, but this feature will probably not be included in the first version of 645 Multics.

We do not know in general how to make the software cope with a berserk CPU, drum controller or general I/O controller. In 645 Multics such a trouble will undoubtedly require a restart, the magnitude of which will vary greatly depending on exactly what the sick hardware unit did before it was caught.

The problem of coexisting with product service routines will be partly solved by subordinating some product service routines to Multics, and partly by the fact that Multics can easily abandon half the hardware of a large enough system on request, so that product service routines can test the other half. It appears likely, however, that integration of product service routines into Multics will be the most difficult aspect of the project, and the last to be satisfactorily completed.

We have no very useful techniques for protecting the system from software bugs. We are reduced to the old-fashioned method of trying to keep the bugs from getting into the software in the first place. This is a primary reason for programming the system in PL/I, and for insisting that modules of the operating system should conform to conventions for user programs. The 645 lends itself exceptionally well to being driven with repeatable sequences of events, and this will help to find timing-dependent software bugs. But some software bugs will survive; they always do.

## ACKNOWLEDGMENTS

It would be nearly impossible to name all those who have participated in formulating the material presented in this paper. All of the authors of the other papers in this group have contributed substantially, as have many others of our colleagues and friends. We are particularly grateful to Dr. E. Wolman of Bell Laboratories for allowing us to paraphrase some of his concise observations about the problem of scheduling.

# A GENERAL-PURPOSE FILE SYSTEM FOR SECONDARY STORAGE*

R. C. Daley
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*
and
P. G. Neumann
*Bell Telephone Laboratories, Inc.*
*Murray Hill, New Jersey*

## 1. INTRODUCTION

The need for a versatile on-line secondary storage complex in a multiprogramming environment is immense. During on-line interaction, user-owned off-line detachable storage media such as cards and tape become highly undesirable. On the other hand, if all users are to be able to retain as much information as they wish in machine-accessible secondary storage, various needs become crucial: Little-used information must percolate to devices with longer access times, to allow ample space on faster devices for more frequently used files. Furthermore, information must be easy to access when required, it must be safe from accidents and maliciousness, and it should be accessible to other users on an easily controllable basis when desired. Finally, any consideration which is not basic to a user's ability to manipulate this information should be invisible to him unless he specifies otherwise.

The basic formulation of a file system designed to meet these needs is presented here. This formulation provides the user with a simple means of addressing an essentially infinite amount of secondary storage in a machine-independent and device-independent fashion. The basic structure of the file system is independent of machine considerations. Within a hierarchy of files, the user is aware only of symbolic addresses. All physical addressing of a multilevel complex of secondary storage devices is done by the file system, and is not seen by the user.

Section 2 of the paper presents the hierarchical structure of files, which permits flexible use of the system. This structure contains sufficient capabilities to assure versatility. A set of representative control features is presented. Typical commands to the file system are also indicated, but are not elaborated upon; although the existence of these commands is crucial, the actual details of their specific implementations may vary without affecting the design of the basic file structure and of the access control.

*Work reported herein 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).

Section 3 discusses the file backup system, which makes secondary storage appear to the user as a single essentially infinite storage medium. The back-up system also provides for salvage and catastrophe reload procedures in the event of machine or system failure. Finally, Section 4 presents a summary of the file system program modules and their interrelationship with one another. The modularity of design enables modules affecting secondary storage device usage to be altered without changing other modules. Similarly, the files are formatless at the level of the file system, so that any changes in format do not affect the basic structure of the file system. Machine independence is attempted wherever it is meaningful.

Sections 2 and 3 are essentially self-contained, and may be read independently of the companion papers (see references 1-5). Section 4 requires a knowledge of the first three papers.

## 2. THE FILE STRUCTURE AND ACCESS CONTROL

In this section of the paper, the logical organization of the file structure is presented. The file structure consists of a basic tree hierarchy of files, across which links may be added to facilitate simple access to files elsewhere in the hierarchy. Each file has an independent means for controlling the way in which it may be used.

If files are to be shared among various users in a way which can be flexibly controlled, various forms of safeguards are desirable. These include:

S1. Safety from someone masquerading as someone else;

S2. Safety from accidents or maliciousness by someone specifically permitted controlled access;

S3. Safety from accidents or maliciousness by someone specifically denied access;

S4. Safety from accidents self-inflicted;

S5. Total privacy, if needed, with access only by one user or a set of users;

S6. Safety from hardware or system software failures;

S7. Security of system safeguards themselves from tampering by nonauthorized users;

S8. Safeguard against overzealous application of other safeguards.

These safeguards recur in the subsequent discussion. The various features of the file system presented below are summarized in Section 2.4, along with the way in which these features help to provide the above safeguards.

### *2.1 Basic Concepts*

In the context of this paper, the word "user" refers to a person, or to a process, or possibly to a class of persons and/or processes. The concept of the user is rigorously defined in terms of a fixed number of *components,* such as an accounting number, a project number, and a name. (Classes of users may be defined by leaving certain components unspecified.) For present purposes, the only users considered are those who employ the file system by means of its normal calls.

A *file* is simply an ordered sequence of *elements,* where an element could be a machine word, a character, or a bit, depending upon the implementation. A user may create, modify or delete files only through the use of the file system. At the level of the file system, a file is formatless. All formatting is done by higher-level modules or by user-supplied programs, if desired. As far as a particular user is concerned, a file has one name, and that name is symbolic. (Symbolic names may be arbitrarily long, and may have syntax of their own. For example, they may consist of several parts, some of which are relevant to the nature of the file, e.g., ALPHA FAP DEBUG.) The user may reference an element in the file by specifying the symbolic file name and the linear index of the element within th file. By using higher-level modules, a user may also be able to reference suitably defined sequences of elements directly by context.

A *directory* is a special file which is maintained by the file system, and which contains a list of *entries.* To a user, an entry appears to be a file and is accessed in terms of its symbolic entry name, which is the user's file name. An *entry name* need be unique only within the directory in which it occurs. In reality, each entry is a pointer of one of two kinds. The entry may point directly to a file (which may itself be a directory) which is stored in secondary storage, or else it may point to another entry in the same or another directory. An entry

which points directly to a file is called a *branch,* while an entry which points to another directory entry is called a *link*. Except for a pathological case mentioned below, a link always eventually points to a branch (although possibly via a chain of links to the branch), and thence to a file. Thus the link and the branch both *effectively point to* the file. (In general, a user will usually not need to know whether a given entry is a branch or a link, but he easily may find out.)

Each branch contains a description of the way in which it may be used and of the way in which it is being used. This description includes information such as the actual physical address of the file, the time this file was created or last modified, the time the file was last referred to, and access control information for the branch (see below). The description also includes the current state of the file (open for reading by N users, open for reading and writing by one user, open for data sharing by N users, or inactive), discussed in Section 4. Some of this information is unavailable to the user.

The only information associated with a link is the pointer to the entry to which it links. This pointer is specified in terms of a symbolic name which uniquely identifies the linked entry within the hierarchy. A link derives its access control information from the branch to which it ultimately points.

### 2.2 The Hierarchy of the File Structure

The hierarchical file structure is discussed here. The discussion of access control features for selected privacy and controlled sharing are deferred until Section 2.3. For ease of understanding, the file structure may be thought of as a tree of files, some of which are directories. That is, with one exception, each file (e.g., each directory) finds itself directly pointed to by exactly one branch in exactly one directory. The exception is the root directory, or *root,* at the root of the tree. Although it is not explicitly pointed to from any directory, the root is implicitly pointed to by a fictitious branch which is known to the file system.

A file directly pointed to in some directory is *immediately inferior* to that directory (and the directory is *immediately superior* to the file). A file which is immediately inferior to a directory which is itself immediately inferior to a second directory is *inferior* to the second directory (and similarly the second directory is *superior* to the file). The root has level zero, and files immediately inferior to it have level one. By extension, inferiority (or superiority) is defined for any number of levels of separation via a chain of immediately inferior (superior) files. (The reader who is disturbed by the level numbers increasing with inferiority may pretend that level numbers have negative signs.) Links are then considered to be superimposed upon, but independent of, the tree structure. Note that the notions of inferiority and superiority are not concerned with links, but only with branches.

In a tree hierarchy of this kind, it seems desirable that a user be able to work in one or a few directories, rather than having to move about continually. It is thus natural for the hierarchy to be so arranged that users with similar interests can share common files and yet have private files when desired. At any one time, a user is considered to be operating in some one directory, called his *working directory*. He may access a file effectively pointed to by an entry in his working directory simply by specifying the entry name. More than one user may have the same working directory at one time.

An example of a simple tree hierarchy without links is shown in Fig. 1. Nonterminal nodes, which are shown as circles, indicate files which are directories, while the lines downward from each such node indicate the entries (i.e., branches) in the directory corresponding to that node. The terminal nodes, which are shown as squares, indicate files other than directories. Letters indicate entry names, while numbers are used for descriptive purposes only, to identify directories in the figure. For example, the letter "J" is the entry name of various entries in different directories in the figure, while the number "0" refers to the root.

An entry name is meaningful only with respect to the directory in which it occurs, and may or may not be unique outside of that directory. For various reasons, it is desirable to have a symbolic name which does uniquely define an entry in the hierarchy as a whole. Such a name is obtained relative to the root, and is called the *tree name*. It consists of the chain of entry names required to reach the entry via a chain of branches from the root. For example, the tree name of the directory corresponding to the node marked 1 in Fig. 1 is A:B:C, where a colon is used to separate entry names. (The two files with entry names D and E shown in this directory have tree names A:B:C:D and A:B:C:E, respectively.)

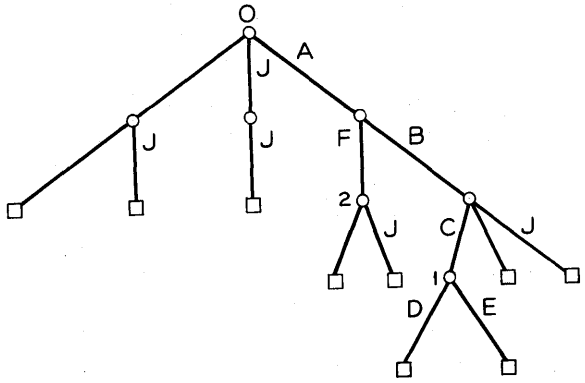



Figure 1. An example of a hierarchy without links.

In most cases, the user will not need to know the tree name of an entry.

Unless specifically stated otherwise, the tree name of a file is defined relative to the root. However, a file may also be named uniquely relative to an arbitrary directory, as follows. If a file X is inferior to a directory Y, the tree name of X relative to Y is the chain of entry names required to reach X from Y. If X is superior to Y, the tree name of X relative to Y consists of a chain of asterisks, one for each level of immediate superiority. (Note that since only the tree structure is being considered, each file other than the root has exactly one immediately superior file.) If the file is neither inferior nor superior to the directory, first find the directory Z with the maximum level which is superior to both X and Y. Then the tree name of X relative to Y consists of the tree name of Z relative to X (a chain of asterisks) followed by the tree name of Y relative to Z (a chain of entry names). For the example of Fig. 1, consider the two directories marked 1 and 2. The tree name of 1 relative to 2 is :*:B:C, while the tree name of 2 relative to 1 is :*:*:F. An initial colon is used to indicate a name which is relative to the working directory.

A link with an arbitrary name (LINKNAME) may be established to an entry in another directory by means of a command

LINK LINKNAME, PATHNAME.

(A command is merely a subroutine call.) The name of the entry to be linked to (PATHNAME) may be specified as a tree name relative to the working directory or to the root, or more generally as a path name (defined below). Note that a file may thus have different names to different users, depending on how it is accessed. A link serves as a shortcut to a branch somewhere else in the hierarchy, and gives the user the illusion that the link is actually a branch pointing directly to the desired file. Although the links add no basic capabilities to those already present within the tree structure of branches, they greatly facilitate the ease with which the file system may be used. Links also help to eliminate the need for duplicate copies of sharable files. The superimposing of links upon the tree structure of Fig. 1 is illustrated in Fig. 2. The dashed lines downward from a node show entries which are links to other entries. When the links are added to the tree structure, the result is a directed graph. (The direction is of course downward from each node.)

In the example of Fig. 2, the entry named G in directory 2 is a link to the branch named C in directory 3. The entry named C in directory 4 (recall that entry names need not be unique except within a directory) is a link to the entry G in directory 2, and thus acts as a link to C in directory 3. Both of these links effectively point to the directory 1.

It is desirable to have a name analogous to the tree name which includes links. Such a name is the *path name,* and is assumed to be relative to the root unless specifically stated otherwise. The path name of a file (relative to the root) is the chain of entry names used to name the file, relative to the root. (For example, the directory 1 in Fig. 2 may have path name A:B:C, A:F:G or H:C, depending on its usage.) The working directory is always established in terms of a path name. A user may change his working directory by means of a command such as

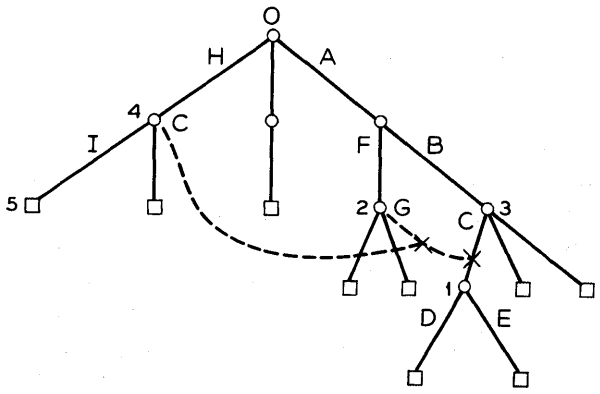



Figure 2. The example of Fig. 1 with links added.

CHANGEDIRECTORY PATHNAME,

where the path name may be relative to the (old) working directory or to the root. The definition of a path name relative to a directory other than the root is similar to the definition of a tree name, with the following exceptions: the concept of a file immediately inferior to a directory is replaced by the concept of a file effectively pointed to by the entry. The concept of a directory immediately superior to a file is replaced by a concept which is well defined only as the inverse of the above effective pointer, that is, dependent on what entry in which directory was previously used to reach the file.

In general, any file may be specified by a path name (which may in fact be a tree name, or an entry name) relative to the current working directory. A file may also be specified by a path name relative to the root. In the former case, the path name begins with a colon, in the latter case it does not.

To illustrate these somewhat elusive concepts, consider the example of Fig. 2. Suppose that the working directory has the path name H (i.e., directory 4). The command

CHANGEDIRECTORY :C

results in the working directory with path name H:C (i.e., directory 1). Subsequent reference to a file with path name :*:I (relative to the working directory with path name H:C) refers to the file 5 in the figure. The command

CHANGEDIRECTORY:*

results in restoring the original working directory with path name H. (With this interpretation of *, the user believes he is working in a tree. Note that the design could be modified so that a path other than the one used on the way down could be used on the way back up toward the root, but not without adding considerable complexity to the design.)

The pathological case referred to above with respect to a link effectively pointing to a file arises as follows. Consider again Fig. 2. Suppose that the branch C in directory 3 is deleted from this directory; suppose also that in the same directory a link with name C is then established to the entry C in directory 4, e.g., by means of the command

LINK C,H:C.

Access to entry C in directory 3 (or to entry G in 2, or C in 4, for that matter) then results in a loop in which no branch is ever found. This and similar loops in which no branch is found may be broken in various ways, for example, by observing whether an entry is used twice on the same access. Note that much more devious loops may arise, as for example that resulting from the establishment of a link (named K) from directory 1 to entry H in the root. Then the path name :C:K relative to directory 4 refers to directory 4 itself. This and similar loops which involve chains of directories are inherent in the use of links, and may in fact be used constructively.

### *2.3 Access Control*

An initial sign-on procedure is normally desirable in order to establish the identity of the user for accounting purposes. It may also be necessary to control the way in which the user may use the sys-

tem. There are two basic approaches to using the hierarchy of files described here. First, the file structure may be *essentially open,* with initial access unrestricted and with subsequent access permitted to all other directories unless specifically denied. On the other hand, the file structure may be *essentially closed,* with initial access restricted for any user to a particular initial directory (assuming his ability to give a password, for example) and with subsequent access to other directories denied unless specifically permitted. There are in fact arguments for each extreme. The essentially open scheme implies that locks need be placed only where they are essential (and most effective). The essentially closed scheme provides well-defined working areas, frees the user from worrying about other users, and helps prevent the user's files from being accidentally altered. It may be observed that the scope of capabilities of the file structure described here does not depend on whether the structure is essentially open or closed. In practice, a position somewhere in between the two extremes is likely to result.

In attempting to access a file, a user may or may not be successful, depending upon what he is trying to do. The basic framework within which permissions are granted is now considered. This framework is independent of the file structure described above. Although the exact set of permissions may therefore vary from system to system, a flexible set adequate for normal usage is given here as an illustration. All permissions are logically on the branches which point to files. (In actual implementation, however, there may in some cases be permissions associated with a directory rather than repeated for each entry in that directory.)

The set of permissions with which a given user may access a particular branch is called the *mode* of the branch for that user. Associated with each branch is an *access control list,* which contains the list of users (or sets of users) along with the corresponding mode associated with each user. The permissions for any users on the list may be overridden (assuming permission to do so—see below) by adding subsequent users and modes to the list. The list is scanned in order of recency, and thus the addition acts as an override. (Each time the access control list is changed, a garbage collection is performed in order to keep the list nonredundant.) All access control information required for the use of a given file is contained in the list on the branch pointing to that file, and is thus independent of the way in which the file was accessed.

The mode consists of five *attributes,* named TRAP, READ, EXECUTE, WRITE and APPEND, each of which is either ON or OFF. In performing access control, the TRAP attribute is examined first. It is by itself powerful enough to accomplish the roles of the other four attributes, which are called *usage attributes*. However, the four usage attributes are included here for ease of description, as well as for ease of use of the system. The four usage attributes indicate permission to perform the given activity on the branch by the particular user only if the corresponding attribute is ON. The function of each attribute is now defined.

*TRAP.* When a branch has the TRAP attribute ON for a given user, a trap occurs on any reference by that user which affects the contents of the file to which that branch points. In this case, the access control module calls a procedure whose name is given as the first entry of a *trap list.* A trap list may be associated with each user in the access control list. Additional parameters may be defined in the trap list, and are passed as constants to the called procedure. Furthermore, all pertinent information regarding the branch as well as the calling sequence which caused the trap are passed to the called procedure. The traps are processed in the order specified by the trap list. The return to the access control module specifies the effective values of the four usage attributes which are to govern the access. The returned value may override the initial values of these attributes.

The user of a branch may inhibit the trap process. In this case, all references to an entry with the TRAP attribute ON cause an error return to the calling procedure. The TRAP attribute is extremely useful for monitoring of file usage, for placing additional restrictions on access (e.g., user-applied locks), for obtaining subroutines only if and when they are actually referenced, etc. A pair of commands such as LOCK and UNLOCK provide the user with a standard way of applying locks on an entry.

```
LOCK   FILENAME,KEY
UNLOCK FILENAME
```

(FILENAME is the name of a branch given as a

path name.) The command LOCK inserts a trap which on each attempted access may request the user to supply the designated key, and permit access only if the key is correctly supplied. UNLOCK removes the lock. (A timelock command might also be desirable, for example, to make a given branch available to a particular user only between certain times on certain days.) These commands are available to a user only if the branch pointing to the directory which contains the entry FILENAME has the WRITE attribute ON for that user (see below).

*The Usage Attributes.* The READ, EXECUTE, WRITE and APPEND attributes govern permission to perform operations upon files with certain intents, with an intent corresponding to each attribute. Every operation on a given branch implies one of the four intents, namely *read, execute, write* or *append.* The interpretation of the intent depends upon whether the accessed branch points to a directory (a *directory branch*) or to a nondirectory (a *nondirectory branch*), as seen below.

If a branch is a nondirectory branch, the meaning of each intent is quite simple. The read intent is the desire to read the contents of the file. The execute intent is the desire to execute the contents of the file as a procedure. The write intent is the desire to alter the contents of the file without adding to the end of it. The append intent is the desire to add to the end of the file without altering its original contents. The attribute on a nondirectory branch which corresponds to the particular intent of an operation on that branch indicates permission to carry out that operation only if that attribute is ON.

If a branch is a directory branch, the meaning of each intent is different. The read intent is the desire to read those contents of the directory which may be available to the user, i.e., to obtain an itemization of the directory entries. The execute intent is the desire to search the directory. The write intent is the desire to alter existing entries in the directory without adding new ones. This includes renaming entries, deleting entries, and changing the access control list for branches in that directory. The last of these includes adding traps to the trap list and changing the usage attributes. The append intent is the desire to add new entries without altering the original entries. The attribute on a directory branch which corresponds to the particular intent of an operation on that branch indicates permission to carry out that operation only if that attribute is ON.

Several additional examples of system commands are now given. Assuming the necessary WRITE attributes are ON for the appropriate directory branches, a user may by use of suitable commands change the access control list of entries or delete entries in various ways. For example, he may change (wherever permitted by the WRITE attribute of an inferior branch) the list for all inferior directory branches, or for all inferior nondirectory branches, or for all inferior nondirectory branches whose names include the parts FAP DEBUG, or for all directory branches not more than some number of levels inferior. Similarly, an elaborate delete command may be constructed. (The possibility of no one having the WRITE attribute ON for a given directory branch can be combatted in various ways. One way is not to permit a change in the list to occur which brings about this circumstance, another way is to make this condition imply no restriction.)

Assuming that the necessary READ attributes are ON for the appropriate directory branches, a user may obtain an itemization of desired portions of desired inferior directories, possibly obtaining a graphical picture of the hierarchy.

### *2.4 Summary of File System Features*

At this point, it is desirable to summarize the various features of the file system, and to state which of these contribute to which of the safeguards S1 through S8 mentioned above. The basic features of the file system may be stated as follows:

F1. The inherent hierarchical structure of the file system itself;

F2. The access control which may be associated with a directory branch;

F3. The backup procedures (discussed in the next section).

In addition, certain aspects of the hardware and of the central software also contribute to providing these safeguards:

F4. The hardware, and central software system.[2,3]

The ways in which the safeguards S1-S8 interact with the features F1-F4 are summarized in Table 1.

Table 1. The Features of the File System and Which Safeguards They Assist in Providing.

| Safeguards / Features | MASQ S1 | PERM S2 | DENY S3 | SELF S4 | PRIV S5 | BUGS S6 | TAMP S7 | ZEAL S8 |
|---|---|---|---|---|---|---|---|---|
| F1. Hierarchy | Y | | Y | Y | Y | Y | | |
| F2. Attributes | Y | Y | Y | Y | Y | Y | Y | Y |
| F3. Back-up | Y | Y | | Y | | Y | | |
| F4. System | | | Y | | Y | Y | Y | |

Note: Y = YES, the feature does assist.
Blank = NO, it does not.

## 3. SECONDARY STORAGE BACKUP AND RETRIEVAL

One important aspect of the file system is that the user is given the illusion that the capacity of file storage is infinite. This concept is felt to be extremely important, as it gives all responsibility for remembering files to the system rather than to the individual user. Many computer installations already find themselves in the business of providing tape and card-file storage for their users. It is intended that most of this need will be replaced by the file system in a more general and orderly manner.

That portion of the file system storage complex which is immediately accessible to the file system, i.e. disks and drums, is called the *on-line storage system*. Devices which are removable from the storage complex, such as tapes, data cells and disk packs which are used by the file system as an extension of the on-line facilities, are called the *file backup storage system*. To the user, all files appear to be on line, although access to some files may be somewhat delayed. For the purpose of discussion, a backup system consisting only of magnetic tape is considered. However, the system presented here is readily adaptable to other devices.

### *Incremental Dumping of New Files*

Whenever a user signs off, additional copies of all files created or modified by that user are made in duplicate on a pair of magnetic tapes. At the end of every N hour period, any newly created or modified files which have not previously been dumped are also copied to these tapes. When this is done, the tapes are removed from the machine and replaced by a fresh set of tapes for the next N hour period. Typically N would be a period of between 2 and 4 hours. This procedure has the advantage that the effects of the most catastrophic machine or system failure can be confined to the N hour dumping period.

### *Weekly Dumping of Frequently Used Files*

In the event of a catastrophe, the on-line storage system could be reloaded from these incremental dump tapes. However, since many valuable files, including system programs, may not have been modified for a year or over, this method of reloading is far too impractical. In order to minimize the time necessary to recover after a catastrophe, a weekly dump is prepared of all files which have been used within the last M weeks. This dump is also made on duplicate tapes for reliability.

Actually this weekly dump is taken in two parts. The first part consists of all files which must be present in order to start and run the basic system. The second part consists of all other files which have been used within the last M week period. Typically M would be a period of about three to five weeks. The weekly dump tapes may be released for other use after a period of about two or three months. The incremental dump tapes must be kept indefinitely. However, it may be advantageous to consolidate these tapes periodically by deleting obsolete files.

### *Catastrophe Reload Procedure*

Should a catastrophe occur in which the entire contents of the on-line storage system is lost, the following reload procedure can be used. First reload a copy of the system files from the most recent weekly dump tapes. When this has been done the system may be started, with the rest of the reloading process continuing under the control of the system. Note that this does not necessarily represent the most recent copy of the system. If an important system change has been made since the weekly dump was taken, it may be necessary to reload the incremental tapes before starting the system.

After the system files are reloaded, the incremental dump tapes are reloaded in reverse chronological order, starting with the most recent set of incremental tapes. This process is continued until the time of the last weekly dump is reached. At this time, the second part of the weekly dump tapes is reloaded.

During this process all redundant or obsolete files are ignored. The date and time a file is created or last modified is used to insure that only the most recent copy of a file remains in the on-line storage system. Since directories are dumped and reloaded in the same manner as ordinary files, the contents of the on-line storage system can be accurately restored.

It is possible to continue to load the older weekly tapes until the on-line system is totally reloaded. However, the amount of new information picked up from these tapes becomes increasingly small as one goes further back in time. In view of this, files which do not appear on the most recent set of weekly dump tapes, due to inactivity, are not reloaded at this time. Instead, a trap is added to the appropriate directory branch so that a retrieval procedure is called when the file is first referenced. This allows these files to be reloaded as needed by the retrieval mechanism which is discussed later in this paper.

*On-Line Storage Salvage Procedure*

Although the catastrophe reload procedure can accurately reconstruct the contents of the on-line storage system, it is normally used only as a backstop against the most catastrophic of machine or system failures. When the milder and more common failures occur, it is often possible to salvage the contents of secondary storage without having to resort to the reload procedure. If this can be done, many files which have been created or modified since the end of the last incremental dump period can be saved. In addition, much of the time necessary to run the reload procedure can also be saved.

The usual result of a machine or system failure is that the contents of secondary storage is left in a state which is inconsistent. For example, two completely unrelated directory entries may end up pointing to the same physical location in secondary storage, while the storage assignment tables indicate that this area of storage is unused. If the system were restarted at this time, the situation might never be resolved. The usual effect is that any information subsequently assigned to that area of secondary storage is likely to be overwritten.

This situation arises when the system goes down before the file system has updated its assignment tables and directories on secondary storage. What has probably happened is that some user has deleted a file and another user created a new file which was assigned to the area of storage just vacated by the previous file. When the system goes down, the changes have not been recorded in secondary storage. This is only one example of the type of trouble which occurs when the system fails unexpectedly.

The salvage procedure is designed to read through all the directories in the hierarchy and correct inconsistent information wherever possible. The remaining erroneous files and directory entries are deleted or truncated at the point at which the error was found. Storage assignment tables are corrected so that only one branch points to the same area of secondary storage. Since it is necessary to read only the directories and the storage assignment tables, the salvage procedure can be run in a small percentage of the time necessary to run a complete reload procedure.

The salvage procedure also serves as a useful diagnostic tool, since it provides a printout of every error found and the action taken. This program can also be run in a mode in which it only detects errors but does not try to correct them.

*Retrieval of Files from Backup Storage*

Unless a file has been explicitly deleted by a user, the directory entry for that file remains in the file system indefinitely. If, for some reason, the file associated with this entry does not currently reside on an on-line storage device, the corresponding branch for that file contains a trap to a file retrieval procedure. When a user references a file which is in this condition, his process traps to the retrieval procedure. At this time the user may elect to wait until the file is retrieved from the backup system, to request that the file be retrieved while he works on something else, to abort the process that requested the file, or to delete the directory entry.

If the user elects to retrieve the file, the date and time the file was created or last modified (which are available from the directory entry) are used to select the correct set of incremental dump tapes. The retrieval procedure requests the tape operator to find and mount these tapes. These tapes are then searched until the precise copy of the requested file is found and reloaded. At this time the original access control list of the branch is restored, and the file is now ready to be used by the user.

If a user deletes a file, both the file and the corresponding directory entry are deleted. However, if

a copy of this file appears on a set of incremental dump tapes, this copy is not deleted at this time. This file can still be retrieved if the user specifies the approximate date and time when the file was created or last modified. To help the user in this situation, the incremental dump procedure provides a listing for the operations staff of the contents of each set of incremental tapes. These listings are kept in a log book which may be consulted by the operators in situations such as the above. Selected portions of this listing may be made available to the user.

The user is able to declare that he wishes a certain file to be removed from the on-line storage system without deleting the corresponding directory entry. This may be accomplished by using a system procedure which places the file in a state where it can be retrieved by the normal retrieval procedure.

*General Reliability*

Since the file system is designed to provide the principal information storage facility for all users of the system, the full responsibility for all considerations of reliability rests with the file system. For this reason all dumping, retrieval and reloading procedures use duplicate sets of tapes. These tapes are formatted in such a manner as to minimize the possibility of unrecoverable error conditions. When reading from these tapes during a reload or retrieval process, multiple errors on both sets of tapes can be corrected as long as the errors do not occur in the same physical record of both tapes. If an error occurs which cannot be corrected, only the information which was in error is lost. If the error is a simple parity error, the information is accepted as if no error occurred. When a user first attempts to use a file in which a parity or other error was found, he is notified of this condition through a system procedure using the trap mechanism.

*Secondary Storage Allotments*

The file system assigns all secondary storage dynamically as needed. In general, no areas of the on-line storage system are permanently assigned to a user. A user may keep an essentially infinite amount of information within the file system. However, it is necessary to control the amount of information which can be kept in the on-line storage system at a time.

When a user first signs on, the file system is given an account name or number. All files subsequently created by this user are labelled with his account name. When the user wishes to increase his usage of secondary storage, the file system calls upon a secondary storage accounting procedure giving the user's account name and the amount and class of storage requested.

The accounting procedure maintains records of all secondary storage usage and allotments. A storage allotment is defined as the amount of information which a particular account is allowed to keep in the on-line storage system at one time. Normally the accounting procedure allows a process to exceed the allotment after informing the file system that the account is overdrawn. However, the accounting procedure may decide to interrupt the user's process if the amount of on-line storage already used seems unreasonable.

*Multilevel Nature of Secondary Storage*

In most cases a user does not need to know how or where a file is stored by the file system. A user's primary concern is that the file be readily available to him when he needs it. In general, only the file system knows on which device a file resides.

The file system is designed to accommodate any configuration of secondary storage devices. These devices may cover a wide range of speeds and capacities. All considerations of speed and efficiency of storage devices are left to the file system. Thus all user programs and all other system programs are independent of the particular configuration of secondary storage.

All permanent secondary storage devices are assigned a level number according to the relative speed of the device. The devices which have the highest transmission and access rates are assigned the highest level numbers. As files become active, they are automatically moved to the highest-level storage device available. This process is tempered by considerations such as the size of the file and the frequency of use.

As more space is needed on a particular storage device, the least active files are moved to a lower-level storage device. Files which belong to overdrawn accounts are moved first. Files continue to be moved to lower-level storage until the desired amount of higher-level storage is freed. If a file must be moved from the lowest-level on-line

storage device, the file is removed and the branch for this file is set to trap to the retrieval procedure.

## 4. FILE SYSTEM PROGRAM STRUCTURE

This section describes the basic program structure of the file system presented in the preceding sections, as implemented in the Multics system.[1] (It is assumed here that the reader is familiar with the papers referred to in references 1, 2 and 3.)

A user may reference data elements in a file explicitly through read and write statements, or implicitly by means of segment addressing. It should be noted here that the word "file" is not being used in the traditional sense (i.e., to specify any input or output device). In the Multics system a file is a linear array of data which is referenced by means of a symbolic name or segment number and a linear index. In general, a user will not know how or on what device a file is stored.

A Multics file is a segment, and all segments are files.[1,3] Although a file may sometimes be referenced as an input or an output device, only a file can be referenced through segment addressing. For example, a tape or a teletype cannot be referenced as a segment, and therefore cannot be regarded as a file by this definition.

Input or output requests which are directed to I/O devices other than files (i.e. tapes, teletypes, printers, card readers, etc.) will be processed directly by a Device Interface Module (see reference 4) which is designed to handle I/O requests for that device. However, I/O requests which are directed to a file will be processed by a special procedure known as the File System Interface Module (see reference 4). This module acts as a device interface module for files within the file system. Unlike other device interface modules, this procedure does not explicitly issue I/O requests. Instead, the file system interface module accomplishes its I/O implicitly by means of segment addressing and by issuing declarative calls to the basic file system indicating how certain areas of a segment are to be overlayed.

### *4.1 The Basic File System*

Whether a user refers to a file through the use of read and write statements or by means of segment addressing, ultimately a segment must be made available to his process. The basic file system may now be defined as that part of the central software which manages segments. In general this package performs the following basic functions.

1. Maintain directories of existing segments (files).
2. Make segments available to a process upon request.
3. Create new segments.
4. Delete existing segments.

Figure 3 is a rough block diagram of the modules which make up the basic file system. This diagram is by no means complete but is used here to give the reader an overall view of the basic flow. The directional lines indicate the flow of control through the use of formal calling sequences, with formal return implied. Lines with double arrowheads are used to indicate possible flow of control in either direction. The circles in the diagram indicate some of the data bases which are common to the modules indicated. The modules and data bases drawn below the dotted line must at least partially reside in core memory at all times since they will be invoked during a missing-page fault (see reference 3).

#### *Segment Management Module*

The segment management module maintains records of all segments which are known to the current process. A segment is *known to a process* once a segment number has been assigned to that segment for this process. A segment which is known to a process is *active* if the page table for that segment is currently in core. If the page table is not currently in core, that segment is *inactive*.

If a segment is known to a process, an entry will exist for that segment in the Segment Name Table (SNT). This entry contains the call name, the tree name and the segment number of the segment (file) along with other information pertinent to the segment as used by this process. The *call name* is a symbolic name used by the user to reference a segment. This name normally corresponds to an entry in the user's directory hierarchy which ultimately points to the desired file. It should be noted that a different copy of the segment name table exists for each individual process.

If a segment is active, an entry for that segment exists in the Segment Status Table (SST). This ta-

ble is common to all processes and contains an entry for all active segments. If a segment is inactive (no page table is in core), no entry exists for that segment in this table. Each entry in the segment status table contains information such as the number of processes to which this segment is known and a pointer which may be used to reference the file or files which are to receive all I/O resulting from paging this segment in and out of core.[3]

When a user references a segment for the first time, a directed fault will occur. At this time control is passed to a procedure known as the linker.[3] This procedure picks up the symbolic segment call name from a pointer contained in the machine word causing the fault. The linker must now establish a segment number from this symbolic name. An entry to the segment management module is provided for precisely this purpose.

When a call is made to the segment management module to establish a segment number from a call name, the segment name table is searched for that call name. If the call name is found in the segment name table, the segment number from this table is returned immediately to the calling procedure. However, if this is not the case, the segment management module must take the following steps.

1. Locate the segment (file) in the user's directory hierarchy via a call to the search module.
2. Assign a segment number for this segment.
3. Update the segment name table indicating that this segment is now known to this process.
4. Open the file or files which are to receive I/O resulting from paging.
5. Create or update the appropriate entry in the segment status table.
6. Establish a page table and segment descriptor for this segment if the segment was not already active for some other process.
7. Return the segment number to the calling procedure.

If a segment is known to a process but is not currently active, the descriptor for that segment will indicate a fault condition. If and when this fault occurs, the segment can be reactivated by locating the appropriate entry in the segment name table and repeating steps 4 through 7. Note that the segment does not have to be located again in the directory hierarchy since the tree name is retained in the segment name table.

If a segment is to be modified during its use in a process, the user may elect to modify a copy of that segment rather than the original. When this is the case, the copying of this segment is done dynamically as a by-product of paging. However, if the copying is not complete at the time the segment becomes inactive, the copying must be completed at this time.

If a segment is to be copied, there are actually two open files involved, the original file and the copy or *execution file*. When a page table is initially constructed by the segment management module, each entry in that page table will contain a fault indication and a flag indicating what action should be taken if and when that fault occurs. This flag may indicate one of the following actions:

1. Assign a blank page.
2. Retrieve the missing page from the original file.
3. Retrieve the missing page from the execution file.

Once a page has been paged out (written) into the execution file, it must be retrieved from that file.

An entry to the segment management module is provided by which a user may declare a synonym or list of synonyms for a segment name. For example, a user may have a certain procedure which references a segment called "Gamma" and another procedure which references a segment called "Alpha." If the user wishes to operate both procedures as part of the same process using a segment called "Data" he may do so by declaring Alpha and Gamma to be synonyms for Data. This association is kept by the segment management module in a Synonym Table (SYNT). Whenever the segment management module is presented with a call name which has been defined as a synonym, the appropriate name is substituted before any further processing takes place.

In addition to the functions described above, the segment management module provides entries through which the user may ask questions or make declarations involving the use of segments known to his process. Some of these functions are listed below.

1. Declare that a segment or some specific locations within a segment are no longer needed at this time.
2. Declare that a segment or some specific locations within a segment are to be reassigned rather than paged in as needed. (The user is about to overwrite these locations).
3. Ask if a segment or some specific locations within a segment are currently in core.
4. Declare that a certain segment is to be created when first referenced.
5. Terminate a segment, indicating that this segment is no longer to be considered as known to this process.

*Search Module*

The search module is called by the segment management module to find a particular segment (file) in the user's directory hierarchy. The search module searches individual directories in the user's hierarchy in a predetermined pattern until the requested branch is found or the algorithm is exhausted. This module calls the file coordinator to search particular directories and to move to other directories in the hierarchy. The user is able to override this search procedure by providing his own search procedure at the initiation or during the execution of his process.

*The File Coordinator*

The file coordinator provides all the basic tools for manipulating entries within the user's current working directory. The functions provided by this module perform only the most primitive operations and are usually augmented by more elaborate system library procedures. The following is a list of some of these operations.

1. Create a new directory entry.
2. Delete an existing entry.
3. Rename an entry.
4. Return status information concerning a particular entry.
5. Change the access control list for a particular branch.
6. Change working directory.

Whenever a user wishes to perform any operation through the use of the file coordinator, the access control module is consulted to determine if the operation is to be permitted.

Since most calls to the file coordinator refer to entries contained in the user's working directory, the file coordinator must maintain a pointer to this directory. This is done by keeping the tree name of the working directory in a Working Directory Table (WDT) for this process.

*Directory Management Module*

When the file coordinator wishes to search the user's working directory, the actual search is accomplished by use of the directory management module. This module searches a single directory specified by a tree name for a particular entry or group of entries. The actual directory search is confined to this module to isolate the recursion process which may be required to search a given directory.

The directory management module issues calls to the segment management module to obtain a segment number for the directory for which it has only a tree name. When the directory management module obtains this segment number and references the directory by means of segment addressing, a descriptor fault may occur indicating that this segment is no longer active. If this happens, the segment management module will try to reactivate this segment by attempting to find this directory in the next superior directory by means of the tree name in the segment name table. To do this the segment management module issues a direct call to the directory management module to search the next superior directory for the missing directory. After obtaining a segment number for the superior directory, the directory management module may cause another descriptor fault to occur when attempting to search this directory. This process may continue until a directory is found to be an active segment or until the root of the directory hierarchy is reached. Since the root is always known to the directory management module, the depth of recursion is finite.

*File Control Module*

The file control module is provided to open and close files for the segment management module. A

file is said to be open, or *active,* if it has a corresponding entry in the Active File Table (AFT). If a file is active, the corresponding entry in the active file table provides sufficient information to control subsequent I/O requests for that file.

If the file is inactive, the open procedure needs only to open the file to the requested state and make the corresponding entry in the active file table. If the file is active, it may have N users reading, or 1 user reading and writing, or N users data sharing (using file as a common data base). If the requested state is incompatible with the current state of the file, the current process must be blocked.[3] For example, if the current user wishes to read a stable copy of the file and there is currently a user writing into that file, the requested state (reading) and the current state (reading and writing) are said to be incompatible.

If the requested state and the current state of the file are found to be compatible, the number of users using the file in that state is increased by one. When a file has been successfully opened by the file control module (with the permission of the access control module), the pointer to the corresponding entry in the active file table is returned to the calling procedure. This pointer is used to direct requests for subsequent input or output to the correct file.

*Access Control Module*

The access control module is called to evaluate the access control information for a particular branch, as defined in Section 2. This module is given a pointer to the directory entry for the branch in question and a code indicating the type of operation which is being attempted. The access control module returns a single effective mode to the calling procedure. The effective mode is the mode which governs the use of a file with respect to the current user or process. The calling procedure uses this mode to determine if the requested operation is to be permitted.

If the access control information indicates that a trap is to be effected, the procedure to which the trap is directed is passed the entry for the branch in question and the operation code. The procedure which processes the trap must return to the access control module, specifying the effective mode to be returned by the access control module to its calling procedure. The procedure which processes the trap may choose to strengthen, weaken or leave unchanged the usage attributes which define the effective mode for the branch.

*Page Marker Module*

The page marker periodically interrupts the current process and takes note of page usage, and resets the page use bits[2] of all pages involved in the current process. Pages which fall below a dynamically set activity threshold are listed in the Page Out Table (POT) as likely candidates for removal when space becomes needed.

*Page Management Module*

Control passes to the page management module by means of a missing-page fault in a page table in use by the current process. This fault may indicate that a new page should be assigned from free storage or that an existing page should be retrieved from an active file. In either case a free page must be assigned before anything else can happen. If no pages are currently available, the first page listed in the page out table is paged out. If no pages are listed in the page out table, a random page of appropriate size is removed.

If a new page is to be read in, the page table entry for the missing page contains a pointer to the appropriate entry in the segment status table and a flag indicating whether this page is to be read from the original file or the execution file. In either case a pointer to the appropriate active file may be obtained from the segment status table. This pointer is passed as a parameter to the I/O queue management module with a read request to restore the correct page to core memory.

*I/O Queue Management Module*

The I/O queue management module processes input and output requests for a particular active file. The calling procedure specifies a read or a write request and a pointer to an entry in the active file table which corresponds to the desired file. This request is placed on the appropriate queue for the particular device interface module which will process the request. The queue management module then calls that device interface module indicating that a new request has been placed on its queue. When this is done, the queue management module

returns to the calling procedure which must decide whether or not to block itself until the I/O request or requests are completed.

*Device Interface Modules*

For each type of secondary storage device used by the basic file system, a device interface module will be provided. A device interface module has the sole responsibility for the strategy to be used in dealing with the particular device for which it was written. Any special considerations pertaining to a particular storage device are invisible to all modules except the interface module for that device.

A device interface module is also responsible for assigning physical storage areas, as needed, on the device for which it was written. To accomplish this function, the interface module must maintain records of all storage already assigned on that device. These records are kept in *storage assignment tables* which reside on the device to which they refer.

*4.2 Other File System Modules*

The modules described below are not considered part of the basic file system and are not indicated in Fig. 3. However, these modules are considered to be a necessary and integral part of the file system as a whole.

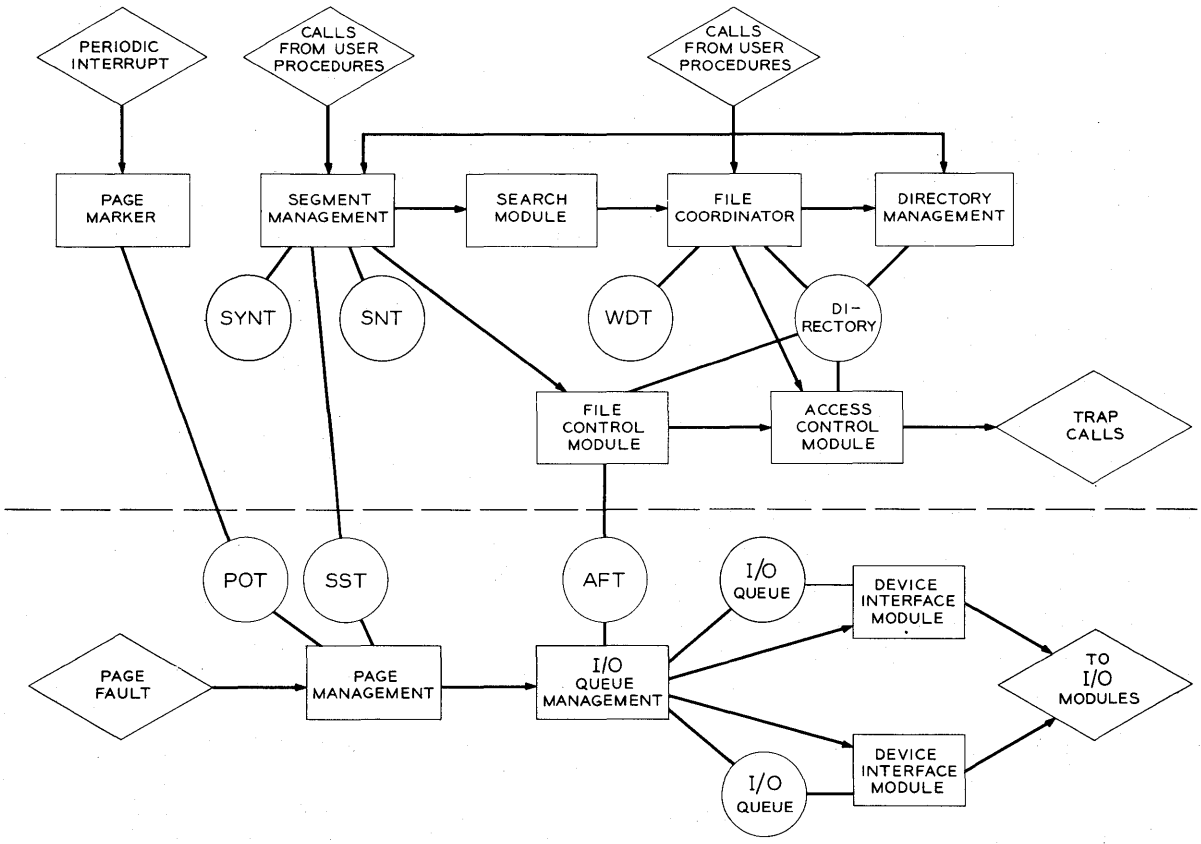



Figure 3. The basic file system.

*Multilevel Storage Management Module*

The multilevel storage management module operates as an independent process within the Multics system. This module collects information concerning the frequency of use of files currently active in the system. In addition, this module collects information concerning overdrawn accounts from the secondary storage accounting module.

The storage management module insures that an adequate amount of secondary storage is available to the basic file system at all times. This is accomplished by moving infrequently used files downward in the multilevel storage complex. This module also moves the most frequently used files to the highest-level secondary storage device available.

*Storage Backup System Modules*

The storage backup system consists of five modules which operate as independent processes. These modules perform the functions described in Section 3.

1. Incremental Dump Module—The sole responsibility of this module is to prepare incremental dump tapes of all new or recently modified files.
2. Weekly Dump Module—This module is run once a week to prepare the weekly dump tapes.
3. Retrieval Module—This module retrieves files which have been removed from the on-line storage system.
4. Salvage Module—This module is run after a machine or system failure to correct any inconsistencies which may have resulted in the on-line storage system. Since the Multics system cannot safely be run until these inconsistencies are corrected, the salvage module must be capable of running on a raw machine.
5. Catastrophe Reload Module—This module is used to reload the contents of the on-line storage system from the incremental and weekly dump tapes after a machine or system failure. Normally, this module is run only when all attempts to salvage the contents of the on-line storage system have failed. This module must be capable of running on a raw machine or under the control of the Multics system.

*Utility and Service Modules*

A large library of utility modules is provided as part of the file system. These modules provide all the necessary functions for manipulating links, and branches using the more primitive functions provided by the file coordinator.

A special group of utility modules is provided to copy information currently stored as a file to other input or output media, and vice versa. The following functions are provided as a bare minimum:

1. File to printer
2. File to cards
3. Cards to file
4. Tape to file
5. File to tape

Actually these modules merely place the user's request on a queue for subsequent processing by the appropriate service module. The service module executes the requests in its queue as an independent process. As soon as the user's request has been placed on an appropriate queue, control is returned to the calling procedure although the request has not yet been executed.

## 5. CONCLUSIONS

In this paper, a versatile secondary storage file system is presented. Various goals which such a system should attain have been set, and the system designed in such a way as to achieve these goals. Such a system is felt to be an essential part of an effective on-line interactive computing system.

## 6. ACKNOWLEDGMENT

The file system presented here is the result of a series of contributions by numerous people, beginning with the MIT Computation Center, continuing with Project MAC, and culminating in the present effort.

## REFERENCES

1. F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System," this volume.

2. E. L. Glaser, J. F. Couleur and G. A. Oliver, "System Design of the GE 645 Computer for the Time-Sharing Application," this volume.

3. V. A. Vyssotsky, F. J. Corbató and R. M. Graham, "Structure of the Multics Monitor for the GE 635," this volume.

4. J. F. Ossanna, L. E. Mikus and S. D. Dunten, "Communications and Input-Output Switching in a Multiplex Computing System," this volume.

5. E. E. David, Jr., and R. M. Fano, "Some Thoughts About the Social Implications of Accessible Computing," this volume.

Additional References

C. W. Bachman and S. B. Williams, "A General Purpose Programming System for Random Access Memories," this volume.

J. B. Dennis and E. C. Van Horn, "Programming Semantics for Multiprogrammed Computations," *ACM Conference on Programming Languages,* San Dimas, Calif., Aug. 1965.

A. W. Holt, "Program Organization and Record Keeping for Dynamic Storage Allocation," *Comm. ACM* 4, pp. 422-431, Oct. 1961.

T. H. Nelson, "A File Structure for the Complex, the Changing and the Indeterminate," *ACM National Conference,* Aug. 1965.

M. V. Wilkes, "A Programmer's Utility Filing System," *Computer Journal* 7, pp. 180-184, Oct.-1964.

# COMMUNICATIONS AND INPUT/OUTPUT SWITCHING IN A MULTIPLEX COMPUTING SYSTEM

J. F. Ossanna
*Bell Telephone Laboratories*
*Murray Hill, New Jersey*

L. E. Mikus
*General Electric Company*
*Phoenix, Arizona*

S. D. Dunten
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*

## INTRODUCTION

This paper discusses the general communications and input/output switching problems in a large-scale multiplexed computing system. A basic goal of such a computing system is to serve simultaneously and continuously a wide range and large number of users. By rapidly time-multiplexing the use of computer system facilities on behalf of these users, the system attempts to satisfy the completion time and response time desires of both the on-line interactive user and the absentee user.

Problems arise in such systems because of the large number and variety of on-line input/output devices, the dynamically changing hardware and software environment, and the need to efficiently use devices such as line printers.

In this paper a new general purpose input/output controller is described which is capable of simultaneously operating a large number of devices of almost arbitrary variety and speed. An input/output software system philosophy is presented which is tailored to the environment of a multiplex computer system. It includes a message coordinator which connects a user program's input and output streams to various input/output devices and to the secondary storage file system. Execution-time redirection and multiple-direction of these connections is a software system feature.

This paper represents the philosophy and direction of the development of the communications and input/output portions of the Multics system (*Multi*plexed *I*nformation and *C*omputing *S*ystem).[1-5]

## GENERAL PROBLEMS

The on-line input/output devices in a computer system may be classified as local or remote. The local peripherals such as drums, discs, tapes, line printers, card readers and punches are typically con-

nected to the computer by short, many-conductor cables. The computer system usually has considerable status information available about local devices and has operators to assist in their care and feeding.

By comparison, remote devices such as typewriters, line printers, card readers and punches are typically connected to the computer system via private or switched telephone company transmission facilities. The computer system can directly obtain only some status of the transmission facilities. The remote operator, if any, is likely to be reachable only via his remote terminal. The number of remote terminals will generally be large compared to the number of local peripherals, and will vary dynamically with the number of remote users.

The actual and potential variety of input/output devices is a challenge to a large-scale multiplex computer system. Likely additions to the devices mentioned above are removable discs, some form of inexpensive mass storage such as the data cell, a variety of both local and remote graphical display terminals, a microfilm processor, remote data collectors and various dependent peripheral analog and digital computers. Further, devices like remote typewriters and line printers will typically abound in several models. In some cases the computer itself will need to originate calls to remote terminals.

Certain remote terminals can impose stringent real-time response obligations on a multiplex computer system. Examples are remote process control or experiment control and certain types of discontinuous remote high-speed data collection.

Supervisory program modules which attend to bulk input/output devices such as line printers and card readers must be scheduled for execution frequently enough to guarantee the efficient use of these devices. Because the supervisor is multiprogramming (processing in parallel) the programs of perhaps hundreds of users, the problem of allocating such facilities as tapes and removable discs to these user programs is nontrivial.

## INPUT/OUTPUT HARDWARE SYSTEM PHILOSOPHY

A basic goal of the Multics system is continuous service. The principal method in achieving this goal is to include more than one copy of each hardware module and to configure the connecting paths between modules such that no single module is essential for continued system operation.

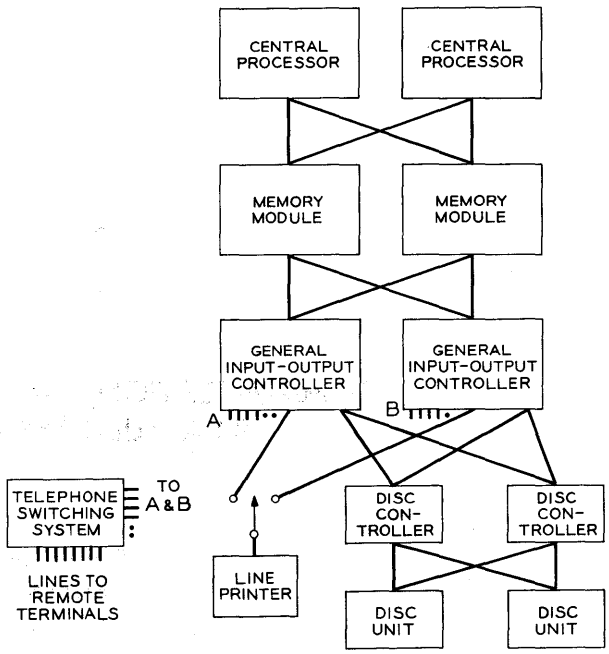



Figure 1. Skeletal hardware system configuration.

Figure 1 is a skeletal system configuration, and shows just enough modules to illustrate this principle. In this example, there are two central processors (CPU), two memory modules, two general input/output controllers (GIOC), two disc controllers, and two disc units. In each case the number two is illustrative and can be higher. Each CPU and GIOC can access every memory module. The disc controller is twin-tailed and accessible by either GIOC; each disc is reachable from either disc controller. Thus there are multiple paths for data flow between disc and main memory.

Single-tailed input/output devices (connectible to only one controller at a time), illustrated by the line printer in Fig. 1, must be manually switched when necessary by the local computer operator.

Remote terminals such as typewriters may access the system via some switching system such as a regular telephone central office or private branch exchange. A number of lines connect each GIOC to the switching system providing multiple-path availability. GIOC ports intended for different types of remote terminals operating at different transmission rates can be assigned different telephone numbers. The switching of remote terminals is discussed below (Connecting and Switching Remote Terminals).

## A GENERAL INPUT/OUTPUT CONTROLLER

The need to handle simultaneously input/output devices having a wide range of speeds, and the need to impose automatically priorities dictated not only by this speed range but also by real-time requirements and by the actual relative importance of different terminals, have motivated the design of a generalized input/output controller (GIOC). The GIOC, conceived for the real-time environment, achieves several functions not highly developed in current large-scale computer input/output subsystems.

A tradeoff between hardware complexity and memory usage is available for all speed ranges of terminal devices, whether they be 10-character-per-second teletypewriters or 400,000-character-per second mass storage peripherals. A modular organization allows functional building blocks to be assembled and tailored specifically for the terminal complement of any rational system.

The input/output capacity of the system is increased over present computer systems by a hardware priority scheme which considers individually the allowable latency of every event requiring the use of main memory. No event requiring the use of main memory is given any higher priority than it requires for error-free operation.

Corresponding to the hardware priority scheme for memory usage is a hierarchy of program interrupt priorities that can ensure rapid response times for real-time events at the expense of slower response times for nonreal-time events.

To facilitate real-time responses to terminal devices, channel commands may be executed without program intervention. By placing commands in a list of channel control words, the commands can be conditionally or unconditionally triggered by the data stream.

All input/output operations are under the direct control of an input/output software system, and hardware memory protection for input/output is not required. Uniform programming is facilitated by the implementation of identical formats and procedures for the control of all terminal devices.

### *Functional Division*

The functional division of the GIOC hardware is illustrated in Fig. 2. The modular functional building blocks are the common control, adapter control, and adapter channels.

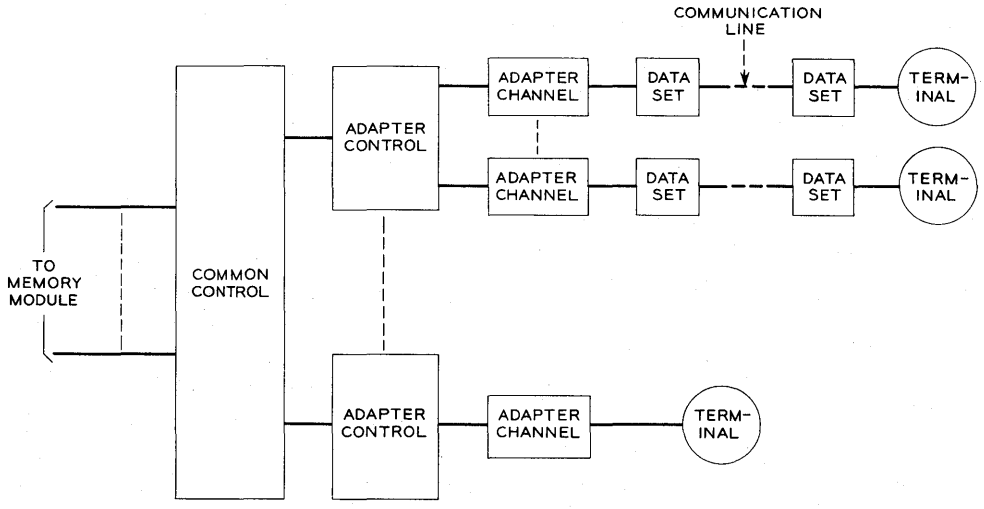



Figure 2. Functional organization of the input/output controller.

Two general types of adapters are used—direct and indirect. Direct adapters, for which the active control word for data transfer resides in the adapter hardware rather than in main memory, are employed with high-data-rate devices. Data are transferred directly to or from the designated locations in main memory using the control word resident in the adapter hardware. By transferring up to 12 characters in a single main memory cycle, direct adapters efficiently utilize memory cycles at the expense of adapter complexity.

Indirect adapters, which contain minimum control within themselves, are employed with low- and medium-data-rate devices. Control words reside in main memory and must be accessed and updated every time a data transfer occurs. Up to 12 charac-

ters of data can be transferred in three main memory cycles.

Adapter channels supply the proper termination to lines connecting the terminal devices to the GIOC. Adapter channels may contain data buffering or simply supply the line interface.

The common control section does not contain any data buffering, but serves mainly to order and control all data transfers. For direct adapters, the common control provides the functions of:

1. Adapter priority ordering and allocation.
2. Communicating program command information.
3. Interfacing to the adapters.
4. Sequencing and subsystem control.
5. Diagnostics and error control.

In addition to the above, for indirect adapters the common control also provides the functions of:

6. Word assembly and disassembly.
7. Parity checking and generation.
8. Control word updating.
9. Dynamic control function detection.
10. Temporary adapter status buffering.

For control of data transfers, the mailbox technique is used. Each adapter channel has a dedicated area, or mailbox, in protected main memory for the residence of control words. Data transfer control words are independent for every adapter channel but command, diagnostic, and status control words are shared by all adapter channels.

Each adapter channel mailbox contains two types of control words: Data Control Words (DCWs) and List Pointer Words (LPWs). DCWs control the data transfer between main memory and adapter channels. Each adapter channel has associated with it an LPW which is a pointer for locating the next DCW to be used for data transfer.

Except for command initiation and termination of unit record peripherals, all information necessary to carry out an input/output operation is uniquely contained within the control words of each adapter channel. In this manner, no adapter channel is dependent upon any other adapter channel in the GIOC for control.

*Hardware Priority Scheme*

A hardware priority scheme minimizes the effects of low-data-rate devices upon the latency of high-data-rate devices by the establishment of priority for all events which require the use of main memory. Six general classes of events exist:

1. Status (communication from the hardware to the program).
2. Commands (communication from the program to the hardware).
3. Direct adapter data services.
4. Indirect adapter data services.
5. List pointer services.
6. Diagnostic functions.

Each class of events itself contains several levels of hardware priority. The priority levels for one class of events can be intermixed with those of another. For example, seven levels of normal status priority are allowed. These seven levels may be intermixed with the priority levels of the other classes.

Commands are treated in the same way as all other events which require the use of main memory. They must wait until the common control grants priority for their issuance.

Two levels of hardware priority for commands are allowed. To issue commands, the program loads one of the two allocated mailboxes with a pointer to the command list. The program then issues an interrupt to the common control. One command is executed each time this event receives priority, until no further commands remain in the command list.

Indirect data service events are those required to transfer data between main memory and indirect adapter channels. Each data transfer takes three main memory cycles.

Direct data service events are those required to transfer data between main memory and direct adapter channels. Each data transfer takes one main memory cycle.

Whenever a DCW exhausts, a list pointer service is required to obtain another DCW and place the new DCW in the proper mailbox. An LPW contains the address of the next DCW to be used in scatter-gather operations.

Instead of accessing the LPW immediately upon a DCW exhaust, the list pointer service event is given a priority just higher than the data service priority for that adapter channel. To initiate the list pointer service, priority must first be granted for that list pointer service event. In this way, the new DCW is guaranteed to be in the proper mailbox before the next data service for that adapter channel occurs. However, the list pointer service time does

not add to the latency of higher priority events. This feature allows low-speed terminal devices to operate using sophisticated scatter-gather and control techniques without adding additional latency to higher priority events.

All work to be done on the GIOC is partitioned into events in such a way that no single event requires more than four accesses to main memory. With the exception of direct adapter data services, priority is allocated to a new event at the completion of every current event.

To further reduce latency for high-transfer-rate terminal devices, any direct adapter data service can temporarily preempt the priority of any lower priority event and thus gain access for the next main memory cycle. Thus, high priority direct adapter data services do not wait for other events using multiple memory cycles (such as indirect adapter data service) to complete their sequence before gaining access to main memory.

Each adapter uses one or more levels of hardware priority. All adapter channels within a single adapter are assigned subpriorities among themselves by the adapter control.

A complete look at all levels of hardware priority is taken after the completion of every event, and the event which will receive the next memory access is then determined. This priority determination occurs concurrently with other common control functions. In effect, after every event the priorities of all events requiring further memory accesses are reconsidered, and the allocating of the next event to receive a memory access is granted.

Except for direct channel data services, which can temporarily preempt the priority of other events, any event which requires the services of main memory is guaranteed to be recognized in its allocated hardware priority sequence within four main-memory access times used by the common control.

*Program Interrupt Priorities*

Program interrupt priorities can be dynamically assigned by the supervisor on a per-channel basis. Program interrupts result from status being stored in main memory after an event occurs which requires program action.

In addition to the seven priority levels for memory access provided by the hardware, status has seven levels of program interrupt priority. For any given event, when a level of hardware status priority is changed by the software, its corresponding level of program interrupt priority is also automatically changed, guaranteeing the associated change in real-time response for that event.

Four subclasses of status events exist:

1. Exhaust
2. Terminate
3. External signal
4. Internal signal

Exhaust status indicates the current active control word for an adapter channel cannot be used for further control. This event implies that a new control word must be obtained to continue data transfer.

Terminate status indicates that the current active control word for an adapter channel cannot be used for further control and that, in addition, no further data transfer for that adapter channel is allowed.

The external signal status is the vehicle by which events outside the GIOC can gain recognition by the program. Such events as operator actions on peripherals fall into this subclass.

The internal signal status is the vehicle by which special control events within the GIOC can gain recognition by the program. Such events as the dynamic detection of incoming communication control characters fall into this subclass.

Each adapter channel, through its control words, can independently activate any one of the status levels when one of the four subclasses of status events occurs. For any given adapter channel, the status levels associated with the subclasses of events may be the same or different. At any given time, each adapter channel thus has access to four levels of status response corresponding to the four subclasses of status events.

Under program control, the same status event occurring on different adapter channels can be assigned different levels of hardware priority and correspondingly different levels of program interrupt priority. This allows optimization of the real-time effect of any event upon any other queued events.

*Channel Commands*

Channel commands can be accepted by an adapter channel at any time, but not all allowable commands have rational meaning when executed without program intervention. Command execution

without program intervention can occur only as a result of a data transfer.

Commands such as "change from transmit to receive mode" can be preplanned in the data sequence and executed without the need for program cognizance at the time of execution. Other commands, such as "change from the inactive to active mode" only have meaning when initiated by the program.

## INPUT/OUTPUT SOFTWARE PHILOSOPHY

The input/output software philosophy must simplify wherever possible the design of a large-scale multiplex computer system and must adequately cope with the general communication and input/output problems discussed earlier.

### *Modularity*

In order to accommodate a dynamically changing device environment and to permit the introduction of new input/output devices without major effort, the input/output software should be highly modular. The device-dependent software for each device should be isolated in a separate, replaceable module. Figure 3 shows a general overall block diagram of the input/output software and its relation-

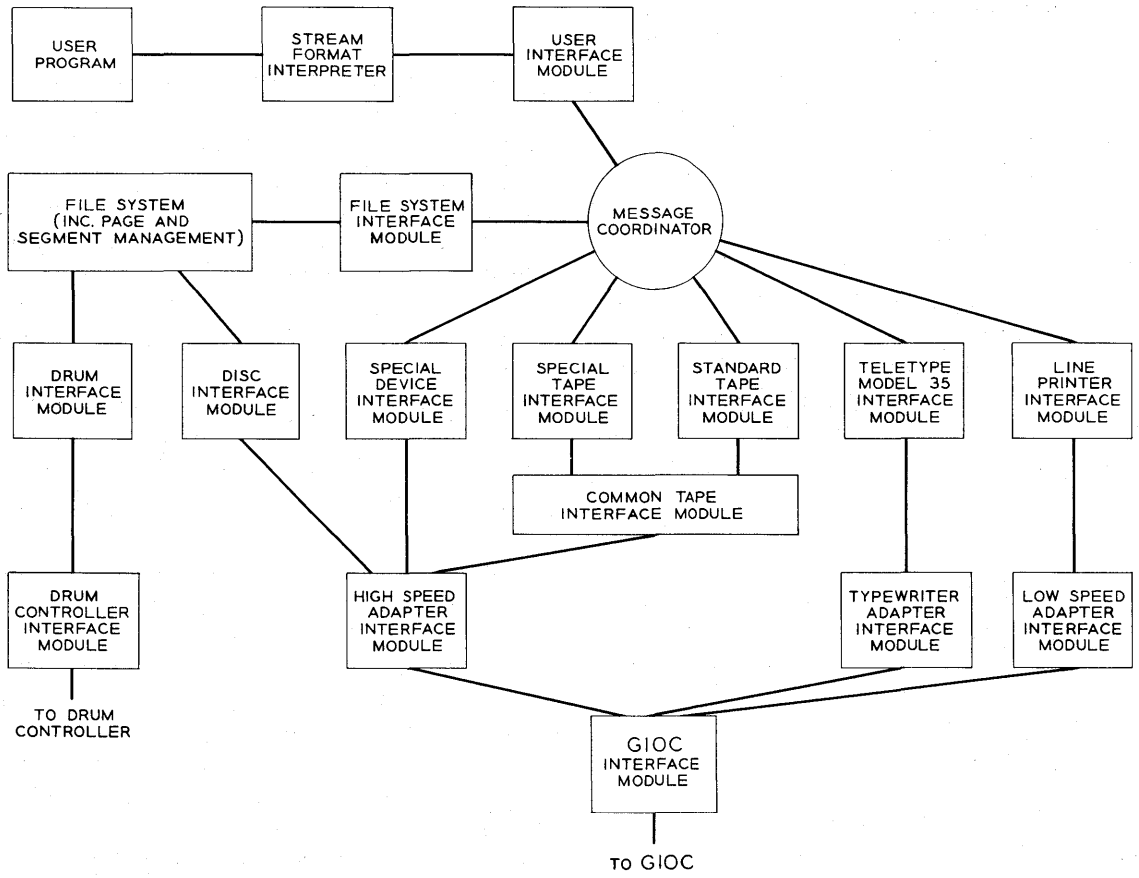



Figure 3. Block diagram of input/output software system.

ship to other software. The message coordinator (MC) and other blocks are discussed below. The MC essentially switches a user's input/output streams to various device interface modules (DIM). A standard DIM will exist for each type of device; for example, there will be a DIM to operate all currently attached Teletype Model 35 typewriters using a standard strategy. This DIM as well as others must be replaceable during system operation. In addition, it must be possible to add a second DIM to operate one or more Model 35's with some special strategy, and to easily associate it with the proper user's streams and the proper communication lines (i.e., the proper Model 35's). This indicates the necessity

for a well-defined, standard software interface for DIMs. The same standard interface must apply to the user interface modules (UIM) and the file system interface module (FSIM).

Because the GIOC is a device, all software which knows about standard GIOC behavior is isolated in the controller interface module (CIM) for the GIOC (the GIOC CIM is hereafter called the GIM). Inasmuch as the GIOC can accommodate various adapter control modules, any special behavior inherent in one kind of adapter control should also be isolated. The typewriter adapter control and adapter channels, for example, may have an echoplex feature (retransmit what is being received), which has no connection with the basic GIOC philosophy or with any particular typewriter. Thus a need exists for adapter interface modules (AIM).

*Flexible Input/Output Direction*

It should not be necessary for a user to decide at the time he writes a program what actual sources and destinations are to be associated with his program input/output streams. The term "stream" is used here to include all input/output transactions, whether they be sequential access or random access in nature. For example, the "title" of a "file" in a PL/I program is the "stream" name to the input/output system.

Prior to execution, a declaration to the UIM establishes a stream-device connection. The absence of such a declaration implies a default connection; a program invoked from a remote typewriter would have that typewriter connected to all input/output streams in the default case. It must be possible to alter this connection during execution by a call to the UIM. Further, it must be possible to multiple-connect streams and devices. During debugging, a particular program might direct both bulk-production printout and commentary printout to a remote typewriter. Later the bulk printed output might be directed to a computation center high-speed line printer or a file in the secondary storage file system. At another time, some printout may need to be directed simultaneously to two line printers in two different locations and to a tape file.

It should be understood that the "user program" may be some supervisory system module. Printer-destined streams are normally diverted to a file in the file system for later scheduled printing, unless the user really meant to attach a printer to his program. Such diversion is also useful for output on tape, removable disc, etc., to facilitate allocation of such devices.

*Description*

A user program initiates input/output by calling a standard or special UIM. The call arguments include the stream name. The redirection and multiple-direction of streams and devices require a module which acts as a switchboard; the message coordinator (MC) in Fig. 3 has this purpose. The UIM uses the MC to implement the stream-device connections for a call. If a stream is associated with a file in the File System (FS), a connection is made to the FSIM.

The File System contains (or knows how to retrieve) all retained files; it is described in detail in a companion paper.[4] Files in the File System are essentially formatless. The FSIM can impose a standard file format. The FSIM makes only declarative calls to the File System; it accomplishes the input/output of files implicitly by means of segment addressing.[3,4]

Figure 3 shows a few representative DIMs. The DIMs basically invoke the strategy for handling particular devices. For example, they may convert between the system's standard character set and the device's particular character set. This conversion includes handling of escape conventions necessary to represent characters absent on the device. The Model 35 DIM knows what messages are needed to operate various terminal features. Two standard tape DIMs are shown, because of the need for two distinct tape strategies. One tape DIM handles tapes in standard system format. The second tape DIM permits handling of nonstandard tape formats in a standard way. Both of these tape DIMs call a subsidiary tape DIM which handles common tape problems.

A single level of format interpretation is available from the DIMs and the FSIM. File System files and device data can be interpreted as being formatless or as having arbitrarily long logical records. In the latter case the files and data must contain format information. Any additional format interpretation such as that required by PL/I can be done by the user or by some standard stream format interpreter (see Fig. 3).

Certain DIMs, such as the disk DIM used exclusively by the File System, do not use the Message

Coordinator. Some may not funnel through the GIM; this is illustrated by the drum DIM and separate drum controller CIM in Fig. 3. The File System may also directly call certain DIMs that are normally reached via the message coordinator. For example, disc pacs can be used for extending the File System storage.

*An input/output software interface language* which is independent of the computer, of the input/output controllers, and of the input/output devices themselves should be used for communicating between software modules. The ease of adding, substituting, and replacing modules implies the need for every module to check the validity of each call to it. For example, the GIM must determine whether a request for service from a DIM or AIM is valid—perhaps whether or not the requester has the right to initiate activity on the referenced channel. The interface language must facilitate this validity checking. All address references in the language are relative. An inner module in the GIM will translate to absolute addresses when actual DCWs are formed.

*Interrupt Handling.* All basic trap or interrupt handling is begun in a supervisor module outside the input/output system.[3] This module determines which module of which process is to be informed about the interrupt. Interrupts originating from the GIOC, for example, are passed for handling to the GIM which knows how to disentangle the associated status information from the GIOC. In turn the GIM passes back to the Tape DIM interrupt and status information relevant to tape handling. Certain interrupts might ultimately be reflected back to a user process.

*Random and sequential* input/output calls are permitted to be mixed and used for all of a user's input/output streams. Sequential calls include calls for the next record, message, character, etc., and calls for spacing and backspacing. A call for "record fourteen" is a random call. All DIMs and the FSIM shall take some action for every type of call. A call to backspace the card reader may result in an error return or no-operation depending on circumstances. Backspacing a typewriter with reverse line feed might be valid. Random calls to a tape file are permitted, because of the inclusion of logical record numbers within the logical record on the tape file.

There is no intended direct correlation between the type of call and efficient device utilization. The user of files in the file system will not usually know on what physical device the file exists. Even if the user did, the file may be scattered on the device in an unknown way. The multiplex character of the monitor system will overlap rewinds, seeks, etc.

*Synchronous and asynchronous* input/output are the two basic operating modes for any particular input/output stream. In the asynchronous mode, the physical input/output transactions are not necessarily synchronized or interlocked with the execution of a program's input/output statements. For example, a user at a typewriter would be allowed to type messages into the system prior to the execution of the read statement which would use them; every execution of a read statement merely plucks the next waiting message out of an input buffer. This example of asynchronous input is analogous to buffered read-ahead schemes which have been used with discs, tapes, etc. An example of asynchronous output is the collecting of output in a core buffer until some physical record size is reached.

In the synchronous mode, the physical transaction associated with a program's input/output statement is carried out during the statement's execution; i.e., control is not returned to the program until the actual transaction is completed. For example, a typewriter user would not be allowed to type (the keyboard might be locked) until the read statement was encountered.

For a particular stream, the input and output modes are independent; for example, the input might be interlocked and the output not. The modes are declarable both prior to and during execution by calls to the UIM. Appropriate interpretation of these modes appears possible for multiple-connected streams and devices. Establishment of a mode amounts to determining which system module in the chain initiates the return to the user program.

Under most circumstances asynchronous input/output is the most efficient. The synchronous or interlocked input/output is useful when operator or user attention is required, and most important when a user is interacting with an undebugged, strange, or many-branched program. The synchronous mode should be imposed on a remote terminal whenever a stream is not associated with the terminal, i.e., when there is no program to which to give messages.

*Statistics.* Sufficient statistical collecting ability must be included in the input/output software de-

sign to accommodate almost any conceivable charging and facilities-allocation schemes. Modes of operation for taking extensive data relevant to system performance should be possible.

## REMOTE TERMINAL CHARACTERISTICS

Remote terminals may be classified as independent or controlled, insofar as the computer system is concerned. A remote small computer which interrupts occasionally for a fast calculation is largely independent. A typical remote typewriter is completely controlled when connected to the system. The following discussion pertains to controlled terminals generally and is illustrated by reference to remote typewriters. The discussion is not intended to provide a list of all of the desirable remote typewriter characteristics.

### *Status*

It is important that the system always have as complete a knowledge of terminal status as possible. Therefore, all pertinent terminal functions must be accompanied by transmission to the system of appropriate information. For example, line feed, carriage return, ribbon color shift, etc., on a typewriter all must transmit characters to the system. Of course, these same functions must be performable by the system by transmission of suitable codes to the terminal. The Proposed Revised ASCII character set provides 32 control characters.

### *Terminal Lock*

To implement the synchronous input mode, the terminal must be lockable by the system. When a read statement is executed, the typewriter keyboard can be unlocked by the system. Even in the asynchronous input mode the keyboard should not be unlocked until the input/output software and hardware are ready to buffer a message. The inability to lock a terminal is an invitation to unexpected and/or unwanted input. The terminal is typically locked during computer output. Of course, a printed, audible, or preferably visual proceed indication is needed to alert the user that input is possible.

An alternative to a terminal lock on terminals producing their own local copy of the input is to operate them full-duplex and to have the computer system echo or retransmit the input back for display. The lack of local copy becomes an indication that input is not wanted. This scheme is workable provided the proceed indication is available. Long transmission delays due to long distances or due to intervening store-and-forward systems would however render this approach awkward or unusable.

The error-detecting possibilities of the echo-back scheme suggest its use even when terminal lock is used. Provision to switch to half-duplex in cases of excessive echo delay is then necessary.

### *Interrupt*

An absolutely essential feature of remote terminal operation is the "interrupt" ability. There must be a key or button whose depression causes instant detachment of the terminal from the current program stream. This interrupt must work even when the terminal is locked. The resulting status of the previously attached program is not discussed here. Normally the terminal is attached to some supervisor command module and is readied for command level input.

Reasons for needing interrupt ability include: (1) the need to stop the attached program which may for example be looping or producing meaningless printout; (2) the desire to attach the typewriter to another stream, possibly belonging to some other program.

Implementation of this interrupt feature requires either full-duplex operation of both terminal and computer, or half-duplex operation with some sort of an auxiliary, possibly narrowband independent channel. The latter is effectively provided by the teletype line-break technique of putting a "space" on the line whose duration is long enough for unique interpretation. The terminal lock must not lock the interrupt button.

### *Identification*

All terminals must be able to identify themselves uniquely to the system. The teletype automatic answer-back scheme is a good example of this ability, because the answer-back message can be long enough not only to provide unique identification but also to independently indicate possible special teminal features.

Although user identification rather than terminal identification should normally be used to control access to the computer and to files in the File

System,[4] positive terminal identification permits default user identification and can indicate that the terminal is in fact a type known to the system.

## CONNECTING AND SWITCHING REMOTE TERMINALS

As suggested in Fig. 1, remote terminals can access a computer system via a telephone central office or private branch exchange (PBX). The basic reasons why such automatic switching is advisable in a large-scale multiplex computer system involve its general flexibility and lower cost. This is especially true if continuous system availability is important. The removal from service for repair or preventative maintenance of one of a system's GIOCs, for instance, requires expensive duplication of computer ports to guarantee access to private lines.

Complete automatic switching provides:

1. User-controlled access to more than one computer. The Bell Telephone Laboratories, for instance, will have four geographically separated, large multiplex computer systems by 1967.
2. User-controlled or switching-system-controlled avoidance of unusable computer ports.
3. Static and dynamic load sharing of remote users with multiple computers.
4. Greater flexibility in planning.
5. Concentration of low-usage terminals.
6. Automatic Direct Distance Dialing access and possible use of existing tie lines between PBXs.
7. Easier terminal maintenance, because of the availability of test centers via the switched network.
8. Terminal-to-terminal communication.
9. Ability to speedily assign, connect, move, reassign, etc., terminals.

An interesting problem can arise while switching remote computer terminals through a telephone switching system. Existing telephone switching plant is engineered to handle the traffic of talkers. A crucial parameter of talker traffic statistics is the product of the average circuit holding time and the average calling rate during the "busy" hour; this quantity is typically in the range of 3-6 call-minutes. Thus an adequate number of talking paths through the switching system may be from 5-10 percent of the number of subscribers. A study made of the holding times of Project MAC users revealed an average holding time of about 1 hour; 20 percent held less than 5 minutes, 50 percent less than 30 minutes, and 80 percent less than 100 minutes.

It may therefore be difficult to add any sizable number of remote typewriters to an existing switching system without impairing telephone service, unless the terminals are to be used for only short holding time inquiries. It is possible to modify existing switching facilities or engineer new facilities at reasonable cost. This, however, can be time-consuming, and planners of remote computing systems should alert telephone companies as soon as possible.

### *Transmission Status*

Each communication-oriented adapter channel on the GIOC can, in addition to receiving and transmitting data, sense a number of local external conditions. These sense lines will be used typically to read status information from standard telephone data sets. The system can then be fully aware of when the ringing signal is present, when data set carrier is present, when data can be sent, etc. These adapter channels also provide control outputs which can operate data set functions, such as causing a hangup.

## CONCLUSION

The communication and input/output problems inherent in a large-scale multiplex computer system have been discussed. Hardware and software philosophies and a description of a general input/output controller intended to cope with these problems have been presented.

It is felt that the modular and dynamic structure of the input/output software and its flexible stream switching ability are essential to the success of a multiplex computer system. Similarly, the hardware flexibility and the uniform software approach permitted by the new controller greatly simplify the design of such computer systems.

## ACKNOWLEDGMENT

The material presented here represents the thoughts and efforts of many people at Bell Tele-

phone Laboratories, Inc., General Electric Company, and Project MAC at the Massachusetts Institute of Technology.

## REFERENCES

1. F. J. Corbató and V. A. Vyssotsky, "Introduction and Overview of the Multics System," *FJCC,* 1965.

2. E. L. Glaser, J. F. Couleur and G. A. Oliver, "System Design of the GE 645 Computer for the Time-Sharing Application," *FJCC,* 1965.

3. V. A. Vyssotsky, F. J. Corbató and R. M. Graham, "Structure of the Multics Monitor for the GE 645," *FJCC,* 1965.

4. R. C. Daley and P. G. Neumann, "A General-Purpose File System for Secondary Storage," *FJCC,* 1965.

5. E. E. David, Jr., and R. M. Fano, "Some Thoughts About the Social Implications of Accessible Computing," *FJCC,* 1965.

# SOME THOUGHTS ABOUT THE SOCIAL IMPLICATIONS OF ACCESSIBLE COMPUTING*

E. E. David, Jr.
*Bell Telephone Laboratories, Inc.*
*Murray Hill, New Jersey*
and
R. M. Fano
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*

Prominent among the products of technology that have shaped our society are automobiles, electric power, and telephones. They provide us with personal transportation, with aids in our physical labor, and with convenient communication. They have radically altered the pattern of our business and private lives. Nobody will deny that these products of technology have substantially increased our mobility, have eliminated a great deal of tedious physical labor, and have contributed vital threads to the fabric of society and commerce.

Yet, they have also brought to our society ills, frustrations, and problems, few of which seem on the wane. The flight to suburbia in search of more elbow room and greenery has left a disproportionate fraction of economically and culturally underprivileged families in the cities. The same technology which has given us new dimensions in communication has been used to implement eavesdropping equipment. The same power tools and machines that are at the foundation of our industrial society caused great grief to people whose obsolete skills were their only source of livelihood and pride as working members of society. Finally, automobiles and power tools are causing us to lose our physical stamina, thereby making us easier prey for disease.

The full influence of these products of technology was felt only some years after the underlying technical advances had come to pass; namely, at about the time each of them became accessible to a large segment of the population. We are now at that stage with computers. Technical means are now available for bringing computing and information service within easy reach of every individual in a community. What will be the effect on our society?

Such service will provide to the individual "thinking tools," somewhat analogous to power tools, to aid him in his daily intellectual labor. These thinking tools will increase the power, skill, and precision of his mind, just as power tools today increase the power, precision, and skill of his muscles. As a matter of fact, there is some question whether our increasingly complex society can survive much longer without falling apart from its own weight, unless individual thinking aids become

*Work reported herein was 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).

available. At the same time, the benefits they may bring to society will unquestionably be mixed with a dose of new problems and frustrations.

The following remarks cannot help being superficial because of the great complexity of the issues involved. Not one but several papers would be required to analyze these issues to any depth. Thus, this paper is being presented primarily to stimulate discussion and further thought.

## A HANDLE ON COMPLEXITY

The increasing sphere of influence of all events and human decisions is a characteristic of our society. Any change or perturbation in the status quo has reverberations which reach often into unexpected quarters. The increasing complexity of provisions embodied in our laws, regulations, and business operating procedures means that the individual has to contend more and more often with situations that he cannot personally master. Frustration and loss of time are among the least painful results.

The tax laws are a good example, as well as one of considerable importance to all of us. As a matter of fact, the tax situation of any one particular individual or business is usually rather straightforward. The difficulty lies in reducing general laws and regulations to one's own specific case. The laws and regulations must apply to a great variety of situations and their complexity is probably unavoidable. Examples, which are intended to illustrate application to common situations, are seldom useful, because they differ in some minor detail, not obviously unimportant, from the case of interest. The crux of the matter is that the number of special situations differing in some material details is so great, it would be impossible to explain for each of them the implications of the applicable laws and regulations. Even if it were possible to do so, the individual would still have the problem of finding the one applicable to his case among all possible special situations.

On the other hand, it would be perfectly feasible to write a computer program that would ask pertinent questions, in sequence, and provide necessary instructions and warnings on the basis of the answers supplied by the individual. In its simplest form, such a program would operate as a mechanized income tax form, with the important difference that it would not ask questions clearly inappropriate in view of preceding answers. Of course, computations would be made automatically on the basis of the data supplied, but this would be the least important and least helpful aspect of the program. Such a program would not have to store a dictionary of specific situations, but could work out the logical consequences of the laws and regulations in each particular instance. Where choices were available, an individual could investigate their implications in his own special case and follow the course of action most advantageous to him. One can conceive also of having the program approved by the Internal Revenue Service so that no question would exist about its correct interpretation of the law. Even further, we can envision the income tax laws and regulations being originally prepared in the form of computer programs so that legislators and Internal Revenue officials could explore more accurately and efficiently their consequences. Speculating about such matters is merely an amusing exercise, and at this time we are bound to invent merely the equivalent of a horseless carriage, rather than the modern automobile.

One can think of many other instances in our society where accessible computing service, with the appropriate software, could help individuals to contend more successfully and with less frustration with the complexities of the modern world: from paying bills and balancing one's bank account to planning a will; from budgeting the family income to selecting investments and making plans for retirement. It may seem strange at this time to envision the average man and housewife using a computer. Yet, to some people years ago it must have seemed equally inconceivable and perhaps sacrilegious to allow the average housewife to turn on powerful motors and operate such complex machines as today's automatic washing machines and driers. Not many years ago we would have winced at the thought of allowing teen-agers to spend hours monopolizing such a priceless creation of human inventiveness and technology as the telephone.

## A HANDLE ON INFORMATION

Information is alarmingly plentiful these days. We are dutybound to acquire, record, search, and use it. While a great deal of effort is being spent in acquiring and recording information, our effectiveness in searching and using it still leaves much to be desired. Information has the unfortunate habit of most often being outdated, hard to locate, and re-

corded in a form poorly suited to one's needs. One reason information is often outdated is that it takes so long to collect and process it. Perhaps nothing short of a widespread information and computing service could provide an effective handle on information.

If such a service were in widespread use, information could be acquired and digested in near real-time and automatically recorded in the mass memory of the computer system. Thereby inventories, abstracts, bank balances, and on and on could then be available on a topical basis. The cost of storing information in the mass memory of a computer is still high, but not inordinately so. A page of single-spaced text stored in the disk file of the current MAC computer system costs approximately 10 cents per month. We see no reason why recording in the mass memory of a computer system should not become competitive with other recording media. With all significant actions being taken with the aid of a computer system, the contents of the system's mass memory would provide a complete, up-to-date representation of the state of the community that it serves. Technical means are not lacking for protecting private information from unauthorized access, while at the same time making it available for statistical surveys and other legitimate purposes.

Once the necessary raw data are automatically available in a computer system, we envision the development of programs to answer any well-defined queries; even those not specifically envisioned by the developers of the programs. We do not intend to imply that we or anybody else knows how to prepare such programs yet, but we do not see any major roadblock to progress in this direction. We are optimistic about technological progress, and can envision computer systems that permit communication (voice and other) interspersed with data processing. On a "conference telephone call," the third party would be a computer. Such a system would enhance, by orders of magnitude, the ability of people to interact and cooperate with one another in a manner both convenient and meaningful to each of the individuals concerned.

## THE THREAT TO PRIVACY

The very power of advanced computer systems makes them a serious threat to the privacy of the individual. If every significant action is recorded in the mass memory of a community computer system, and programs are available for analyzing them, the daily activities of each individual could become open to scrutiny.

While the technical means may be available for preventing illegal searches, where will society draw the line between legal and illegal? Will the custodians of the system be able to resist pressure from government agencies, special-interest groups, and powerful individuals? And what about the custodians themselves? Can society trust them with so much power?

These are very difficult questions indeed. For many purposes, information can be depersonalized before it is put into the central file. We can devise means for providing the equivalent of safe deposit boxes for private information. A hierarchical file system, personal and modular on the lower levels, and impersonal and merged on the upper levels, is another possibility. Processing and access by other than the owner could be restricted to the upper levels. In any case, privacy can be preserved if the lower levels are left decentralized.

## THE CULT OF IMPERSONALITY

The use of identification numbers and the issuing of authoritative and authoritarian instructions and answers are associated in the public mind with computers. Of course, these associations are the results of attempts, for the sake of efficiency, to fit people to the capabilities and idiosyncrasies of computers. The attempt to bring computers within easy reach of individuals is in the opposite direction. Proper names and other means of identifying individuals and locations are just as understandable to computers as identification numbers, and are much more pleasant to people. Computer programs can ask and answer questions in a very polite manner, and can even be made to chitchat realistically enough to fool a person for a little while. Computer programs don't have to be authoritarian and can be made to act unpretentiously. They can make suggestions that leave room for choice, simply warn the person that his course of action may be ill-advised, and still allow him to proceed.

There is nothing we can see inherent in the use of computers that will impersonalize, institutionalize, or automate our behavior. The danger lies in ourselves. Through mental laziness, or fear of accepting responsibility, or just plain neglect we may

delegate to computers prerogatives that should remain ours. Computers are literal-minded, as the late Norbert Wiener was never tired of pointing out. They will not take into account any premise, any limitation, or any fact that has not been made available to them. We should never delegate to them either the formulation of our problems, or decisions as to the adequacy of the solutions they produce.

Our institutions are continuously changing, and some of these changes may appear impersonal simply because they are in conflict with the customs ingrained in us from our youth. The widespread availability of a computing and information service will encourage institutions to change in new directions which may well be inconsistent with our present customs. These changes will not be required by the use of computers, but by the needs of institutions themselves. An example we can foresee concerns financial transactions.

Years ago, money consisted of gold and silver coins whose intrinsic value was identical with the nominal value marked on them. With the increasing number of financial transactions, gold coins proved to be too heavy and inconvenient and were relegated to the vaults of banks and to the strongboxes of individuals. Paper currency came into being, and with it a clear separation between the evidence of wealth and wealth itself. The value of paper currency was both guaranteed and enforced by government. Eventually, it became inadequate to the needs of private individuals and businesses, and personal checks came into use. Checks are twice removed from wealth itself, but one can still touch them and carry them in his own pocket. They are still a tangible evidence of wealth.

We are now at the threshold of a further step away from tangible wealth, in our financial transactions. With the same computer system serving banks, stores, business organizations, and private individuals, we will have available a more convenient form of implementing financial transactions. It will no longer be necessary to mail bills and return checks. Yet, each individual will always be able to have a current accounting of his financial affairs and to authorize payments by simply pressing a key. However, will people be willing to accept the reply of a computer system as evidence of their wealth? We think so, given time. But we are also mindful of the fact that many people around the world are still unwilling to accept personal or even travelers' checks, some don't trust banks and hide currency in their homes, and some refuse to accept anything but gold and silver coins.

## UNEMPLOYMENT

Much has been written about unemployment that computer automation has caused, and may cause in the future. An answer often given is that computer automation will create more jobs than it will eliminate. It has been said too that a good man will always find a job, and in any case our affluent society will surely provide a more than adequate livelihood for the jobless. We think such statements miss the mark. The economic aspects of unemployment are only part of the problem. Work is not only a way of making a living, it is also the channel through which one contributes to his family and to society as a whole. Without a job one loses his self respect and the respect of those around him. This is particularly true when the job has been lost to a machine. In our present society, not only must one work to be happy, but one must also feel that he is contributing through some special skill of his own. Competing with a machine is difficult and frustrating, and so is the acquisition of new skills. The most distressing aspect of unemployment is common to the forced retirement of the man who is still physically and mentally fit. Feeling useless in an active society is a sad lot indeed.

Perhaps we can devise better ways of educating people to meet the demands of a changing world and enable them to learn new skills as older ones become obsolete. Perhaps our job-centered society must change many of its present attitudes. In any case, neither of these alternatives seems likely to provide the whole answer. Women have long been faced with "early retirement" to the household, which holds few satisfactions for many. Some women compete with men for jobs effectively; many more spend much of their creative effort in service, social and community or government. Many take up art or music or sports. When all routine and perhaps some nonroutine data collection and processing tasks are performed by computers, many men may have to make similar adjustments. Already our economy is service-oriented. The U.S. Office of Business Economics estimates that today 55 percent of U.S. jobholders are in service industries. The decline in manufacturing jobs began in 1953, but has not produced the expected unemploy-

ment because of an explosive increase in service jobs. Certainly, this is a hopeful sign, but an effort is needed to make a wider range of service jobs socially acceptable.

It has been said that many, perhaps a majority, of people in our society are incapable of anything except routine work. We are unwilling to accept this as a basic premise. Experience shows that people have vast resources, both intellectual and otherwise, which can be brought to the surface by appropriate means. We share the enthusiasm of Dr. George Gallup in the vast potential of people, as yet undeveloped.[1] The limitations we see today in the crystallized part of our population are probably more a result of their past experience than of their basic abilities. One particularly impressive piece of evidence comes from the several high school curriculum revisions undertaken since the middle 1950's. Children are now being taught in high school what their parents or older brothers and sisters were taught as sophomores in college. Typically, it has been found that children can be taught almost anything; the limitations lie in teachers who have difficulty in overcoming their past. The remarkable progress in high school education came from massive efforts in both subject matter and pedagogy. Similar efforts are underway in continuing education and retraining programs. These are vital to solving the problems of people and machines.

## CONCLUSION

We do not pretend to have answers to the many questions raised here. While we have opinions which tend toward the optimistic, we take for granted that the new resources, among them computers, will be abused as well as used. We believe, however, that abuses (namely those uses which rob us of opportunity and individuality) will be recognized as such, for computers can affect our ethics, creeds, or standards only slowly compared to technological change. Preservation of these will, as always, depend upon the thoughtful and conscientious action of individuals and institutions. In the end, exploitation of computers for the benefit of society hinges upon two pivots: education, and responsible considered action by those of the technical community able to exert some influence.

## REFERENCES

1. G. Gallup, *The Miracle Ahead,* Harper and Row, New York, 1964.

# STRUCTURE AND DYNAMICS OF MILITARY SIMULATIONS

Eugene Levine
*Systems Research Group, Inc.*
*Mineola, New York*

The widespread use of military simulations as a tool for predicting and evaluating systems operations has led investigators to ferret out the generic structure underlying the construct of such simulations. Although a good deal of success has been encountered in this effort, it is evident that only a start in this direction has been achieved thus far. In this discussion, we will indicate where the state-of-the-art rests today and attempt to point out further areas of investigation where new inroads may exist.

Many of the concepts presented herein, emerged during the development of the MILITRAN programming system. The codification of these concepts within the MILITRAN system provides a facile means for developing and implementing military simulation programs by utilization of the MILITRAN programming language.

A military simulation will be viewed in sequel as a war game oriented towards digital implementation in which the rules of play are set forth in advance. Inasmuch as the rules of play may provide for a random device for determining the outcome of each individual move in the game, the result of play is generally not deterministic.

In the development of military simulation programs, one can discern many structural and dynamic features which are generic. We will highlight these features and illustrate means of organization which provide for the representation of military situations.

To begin, a simulation program is composed of many phases. Of prime importance is what we call the Simulative Phase and it is that program phase concerned with the implementation of the system dynamics. The function of the Simulative Phase is to determine and record the state transitions which a system undergoes with the passage of time.

A critical entity in the Simulative Phase is the notion of an event. An event is that entity which triggers an interruption in the current state of a system and causes a state transition. There are many facets to the notion of an event, and in what follows, we shall examine a number of these.

First, an event within a digital simulation program is represented by a vector whose components shall be termed variants. One such variant is the event time and it is that time (in the system) at which the event will be evaluated to determine its effect upon the system (if any). From this latter remark, it is implicit that in referring to an event, we are referring to a "future" event or of an entity that only has the potential to interrupt the system.

Another variant which is germane to an event is the event type. The essence of this variant is to indicate the manner in which the system is altered

when a state transition occurs. The remaining variants are generally modifiers of the event type.

The Simulative Phase of a simulation program may then be viewed as a stochastic process which examines each potential event in a time sequential order and alters the system state as dictated by the event type variant (and its associated modifiers).

In order to view this stochastic process more closely, we must divert the discussion momentarily and remark about the representation of a system. The parameters used to represent a system may be categorized into two relatively distinct kinds of quantities. First, there are parameters of one kind aggregated into data arrays which provide the essential characteristics of the objects within the system. Then there are the potential interruptions (or events) to which the system can give rise (as provided by the event variants).

As the stochastic or simulative process is carried out, in other words, as each event is examined in time sequence, one of two things may happen with each event. Namely, either the examination results in an actual interruption of the system or, on the other hand, the potentiality of the event is not realized, that is, no interruption of the current system state occurs.

Random selection is usually an essential feature in determining whether event realization does or does not occur. Of the various means available for the sampling process, it seems that the utilization of the "pseudo" random number device is most prevalent and effective. Inasmuch as knowledge relating to such devices is so well disseminated, we shall not dwell upon the sampling process itself.

To reflect a change upon a current system state when an event is realized, there is of course a modification of the system representation. This modification of system status may either be within the data arrays, or, of more significance, within the potential events which could result from the current system state. In other words, the realization of the current event may either rule out the potential of existing "future" events, or may create new events which have the potential to be realized.

Again, essential to the representation of a system is the aggregation of potential events. The organization of this aggregate is achieved by means of sequencing the event vectors into an event list. Actually, in practice, this organization is usually achieved by utilizing more than one event list. For example, it is quite common to provide one event list for each event type and the advantages achieved by this subdivision are quite numerous. One immediate advantage is that the event type variant is no longer necessary, that is, the event type is implicitly determined by the list in which the event vector is located. A second advantage is in the simplification of determining the next event in time sequence to be examined. A third advantage is that events of the same type generally have the same number of variants. Thus, the separate event lists are homogeneous within themselves which allows for more convenient organization.

The choice for subdividing the events into separate lists as illustrated above is far from arbitrary. There is actually another advantage achieved by this particular separation which penetrates much more deeply into the theory underlying the construct of simulation programs. Namely, as hinted at earlier, events of the same type alter the system state in somewhat the same manner. The state transitions which thus result from events of the same type are determined by a common program entity which is termed a Submodel of the Simulative Phase. We then have for each event list (or, equivalently, for each event type) an associated Submodel. Each Submodel is a procedure and the collection of Submodels together comprise the system dynamics.

Depending upon the nature of the system which is being simulated, the interdependencies between the Submodels can be and usually are quite complex. The variety and complexity of these dependencies have thus far prevented investigators from abstracting and exhibiting a global theory of the dynamics underlying the simulative process. This problem area is one of current concern in behalf of advancing the present state-of-the-art. In restricting the simulation problem to military situations a certain amount of progress has been achieved in this direction.

In the development of military simulations within the framework just outlined, generic features emerge in a special manner. For example, the objects within the system may be organized into categories such as:

- Sensing devices
- Weapons
- Launches
- Targets
- Sources of tactical control

- Platforms (which are the centers of association for sensors, weapons, launchers, etc.).

These object categories are in turn related to certain generic event types such as detection events, identification and/or classification events, launch events, and damage assessment events, to name a few. Not only is each object category related to a generic event type variant, but even further, the system dynamics related to these events may be ferreted out in a systematic manner.

Enumerating the "Generic Submodels" would be too extensive for the context of this discussion. However, as an illustration of generic system dynamics, one would have the realization of a detection event perform:

(a) the removal of the current detection event from the event lists since its potentiality no longer exists;
(b) the creation of a potential identification and/or classification event;
(c) the creation of a potential lost target event.

Such a generic modification would then provide the state transition resulting from detection realization.

Up to this point, we have only singled out the Simulative Phase of the simulation structure. Actually, one may view the total simulation structure in five phases; namely,

1. the Input Phase which transfers input parameters to the simulation,
2. a preprocessing phase which precomputes data required by each pass through the Simulative Phase,
3. an Initialization Phase which generates the initial system state,
4. the Simulative Phase as described above, and finally,
5. the Output Phase which generates or displays the outcome of simulation.

Inasmuch as the outcome of the Simulative Phase is dependent upon random phenomena, one could usually desire many passes in order to provide better quantitative estimates of the expected outcome. Obtaining as many passes as desired is automatically achieved by alternately implementing the Initialization and Simulative Phases.

In this discussion, we are only able to touch upon those features of significance which comprise the structure of military simulation. We may sum up by noting that from the consideration of many specific military simulations, one can distill those common features of significance. The study and codification of these significant features should provide the proper guidelines for the programming organization and language of military simulation.

## ACKNOWLEDGMENT

The MILITRAN programming system was developed primarily under the auspicies of the Naval Analysis Group of the Office of Naval Research. Additional support was provided by the Directorate, Computers Electronic Systems Division, Air Force Systems Command, Hanscom Field, Mass. Their interest in behalf of the MILITRAN system has been most encouraging to those staff members of Systems Research Group, Inc. who participated in this project.

## REFERENCES

1. Systems Research Group, Inc. The MILITRAN Programming Manual, June 1964, Technical Documentary Report No. ESD-TDR-64-320.
2. Systems Research Group, Inc. The MILITRAN Operations Manual for the IBM 7090-7094, June 1964, Technical Documentary Report No. ESD-TDR-64-389.
3. Systems Research Group, Inc. The MILITRAN Reference Manual, June 1964, Technical Documentary Report No. ESD-TDR-64-390.

# ANALOG-DIGITAL DATA PROCESSING OF RESPIRATORY PARAMETERS

T. W. Murphy
*The RAND Corporation*
*Santa Monica, California*

## INTRODUCTION

The data processing application to be described here might be regarded as a somewhat elementary one by those familiar with computer technology, i.e., most of those present. However, it is presented as an example of an application of data processing technology to a field in which most of the data gatherers are unfamiliar with the heights to which the computing art has been raised in its short lifetime.

It is very difficult for the physician scientist to learn how he can use modern computing technology, because he is, due to his medical education, (typically) lacking in mathematical knowledge or engineering knowledge. Another factor militating against his use of computer technology is the language barrier, more bluntly, the jargon. The physician has his own jargon, indeed, the process of medical education is primarily the learning of the medical language. Probably a smaller proportion of the effort in the computer field is directed toward semantics, yet, an individual armed only with a good understanding of the jargon could, with the assistance of an engineer, make significant advances in the application of data processing technology to medical research.

The following study is accordingly presented, not as an earth-shaking advance in computer technology, but rather as an effort to improve communication between the two disciplines of medicine and computer science.

First we present a discussion of the symbols, definitions, and lung model employed in the work.

$\dot{V}(t)$ = flow rate of gas at mouth.

$\dot{V}(t)$ also equals the rate of change of lung volume

$\dot{V}(t) > 0$ represents *inhalation.*

$\dot{V}(t) < 0$ represents *exhalation.*

$F(t)$ represents the $CO_2$ concentration in the gas at the mouth.

for $\dot{V}(t) > 0$ $F(t) \rightarrow F_I(t)$ i.e., *inhaled* concentration.

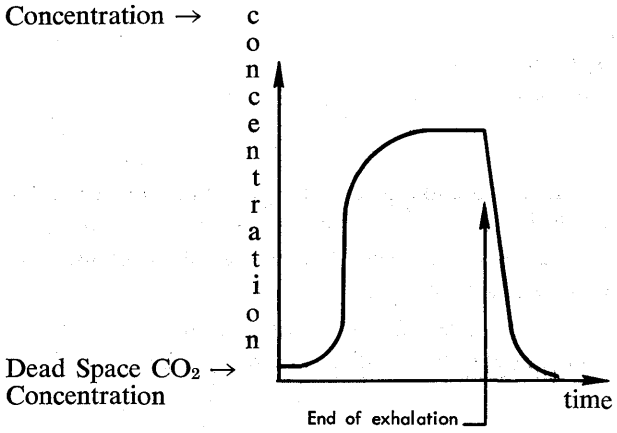
$\dot{V}(t) < 0$ $F(t) \rightarrow F_E(t)$ i.e., *exhaled* concentration.

$F_A(t)$ represents the alveolar concentration of $CO_2$.

We use a two compartment model of the lung.

Compartment 1, The Dead Space
This is purely a region of gas transport. There is no exchange. Its volume $= V_D$.
Compartment 2, The Alveolar Compartment.
This is a region only of gas exchange. There is no transport.

This compartment is assumed to have no concentration gradients. When gas is exhaled, there is some mixing of gas from the two regions, but the first portion of the exhalate represents dead space gas, the last portion represents alveolar gas. Thus we have the following curve of concentration versus time for carbon dioxide in the exhaled gas:




Thus *sup* $[F_E(t)] = F_A(t)$
and *inf* $[F_E(t)] = F_I(t)$, since the dead space gas is simply the gas inhaled on the last breath, and left unchanged. (The inhaled gas is usually considered to be of constant composition during any given inhalation.)

$V_E$ $CO_2(n)$ = volume of $CO_2$ exhaled on $n$th breath.
$V_I$ $CO_2(n)$ = volume of $CO_2$ inhaled on $n$th breath.

$$\text{Obviously, } V_E\ CO_2(n) = \int_{n\text{th breath}} \overset{\circ}{V}(t)\, F_E(t)\, dt$$

$$V_I\ CO_2(n) = \int_{n\text{th breath}} \overset{\circ}{V}(t)\, F_I(t)\, dt$$

The minute volume, the average rate of gas movement out of the lung, can be mathematically defined as

$$MV = \frac{\int_0^t \overset{\circ}{V}(t)\, dt}{T}$$

The tidal volume $V_T$ is the volume of gas moved out of the lung on a given breath.
The alveolar volume $V_A$ is the volume of gas moved out of the alveolar compartment on a given breath. Obviously,

$$V_A = V_T - V_D$$

Similarly, we can define the alveolar ventilation rate,

$$AV = \frac{\sum_0^T V_A}{T}$$

Since the $CO_2$ concentration in the dead space is $F_I$ and its volume is $V_D$, the volume of $CO_2$ exhaled from the dead space $= V_D \times F_I$.
And, volume of $CO_2$ exhaled from the alveolar compartment $= V_A \times F_A$.

$$V_E CO_2 = V_A \cdot F_A + V_D \cdot F_I .$$

If $F_I = 0$, a common case, then

$$V_A = \frac{V_E CO_2}{F_A} .$$

If $F_I \neq 0$, then

$$V_E CO_2 = V_A \cdot F_A + (V_T - V_A) \cdot F_I .$$

But $V_T \cdot F_I = V_I CO_2$.

$$V_A = \frac{V_E CO_2 - V_I CO_2}{F_A - F_I}$$

These two formulae for $V_A$ are known as the Boh formulae. We will now discuss the processing of the above data.

This paper discusses an improved version of two systems previously reported. An attempt has been made here to perform the various operations in the appropriate (analog or digital) section of the equipment instead of doing them all in the "analog" section.

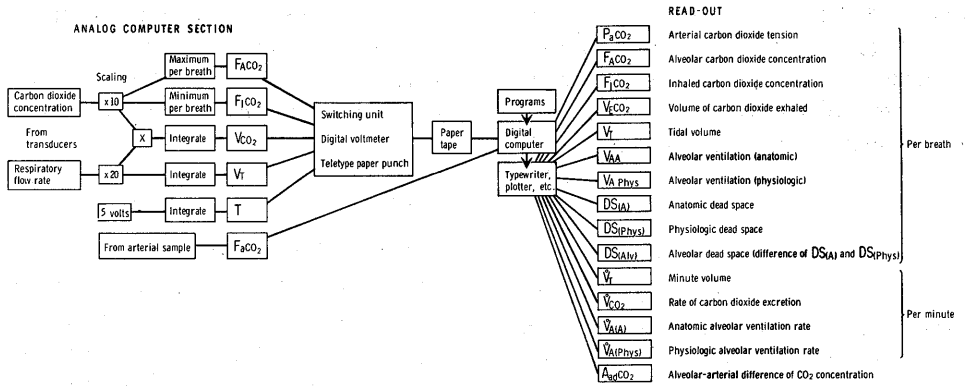
## EQUIPMENT

The system used consists of two transducers, special purpose analog computing equipment with digital read out and digital computing facilities.

The transducers are (a) a pneumotachograph

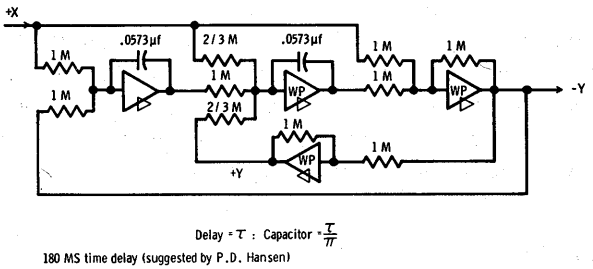
(Fleisch), strain gage (Statham PM 15) and amplifier (Statham CA 9-10), and (b) an infra-red carbon dioxide analyser (Godart).

The computing equipment consists of 25 operational amplifiers, some with chopper stabilization (G.A. Philbrick Researches, K2PA and K2W) and a multiplier (GAP/R, K5M). Plug-in units for the amplifiers were fabricated by the author from modules (K3). Control circuitry was synthesized from digital modules by Tech-Serv (B.R.S.). Read-out equipment is by Hewlett Packard, and the digital computer is a Control Data Corp. 160 A.




*Computation* (Fig. 1)

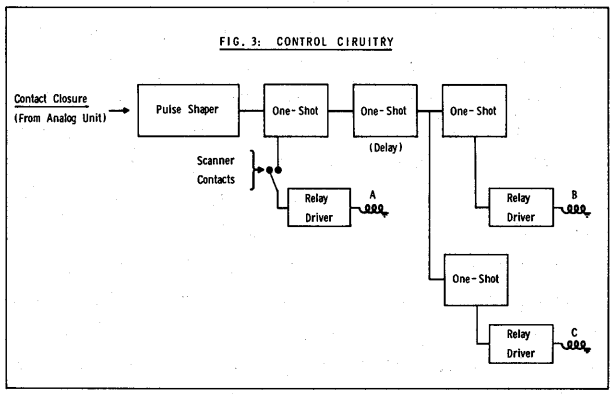
From the pneumotachograph, strain gage and amplifier system a signal arises representing the instantaneous flow rate of the patient's exhalation or inhalation. A small sample (approximately two liters per minute) is taken from this stream and passed through the sampling head of the carbon dioxide analyzer from which is obtained a signal proportional to the carbon dioxide concentration in the gas stream (which is lagged approximately 300 ms). To synchronize the two signals, the "flow" voltage is delayed by an equal amount. This is performed using a (Fig. 2) modification of the Pade approximation devised by Dr. P. D. Hansen. The




Delay = $\tau$ : Capacitor = $\frac{\tau}{77}$

180 MS time delay (suggested by P.D. Hansen)

flow signal is rectified and integrated thereby giving the volume exhaled for that breath. The flow signal and the carbon dioxide signal are multiplied and integrated and this integrand for each breath represents the volume of carbon dioxide exhaled per breath. The peak exhaled or end-tidal carbon dioxide tension is obtained using a peak follower technique and the inhaled carbon dioxide from the inverted curve in the same way. These peak followers are reset after being read out on each breath. A constant voltage is integrated for the period of the breath to give a measure of the time taken for that breath. All five quantities are placed on memory circuits at the end of each breath. The integrators are reset and computation recommences.




*Control Circuits* (Fig. 3)

These consist of a series of digital modules by Tech Serv (B.R.S.). The input pulse to this system is obtained from a voltage crossing detector and relay on the analog unit. This then initiates a pulse train in the digital system which proceeds through a

series of one-shots of variable delay time, each one of which is triggered by the trailing edge of the pulse from the preceding one shot. The time delays are adjusted to the appropriate values. The first one shot operates the relay which connects the integrators to the memory circuits. (A) The second one shot is a delay to allow for closure of this relay, a small dead time, and then operation of the second (shorting) (B) relay which is operated from a third one shot. Another one shot is used to provide a suitable delay between readout of the first channel ($F_A$ $CO_2$) and reset of this unit. (Relay C).

*Calibration*

Calibration of this equipment is rather complex due to the large number of functions performed and every attempt is made to cross-check during calibration. The carbon dioxide analyzer is calibrated with gases of known chemical composition. Its response to these is linear to within plus or minus 1/mm $pCO_2$ at 760 mm barometric pressure. The pneumotachograph, strain gage and amplifier system is calibrated by passing oxygen through a flowmeter which delivers a known amount of gas for any particular position of the rotameter. Stability and linearity of this system are excellent, the only aspect requiring frequent adjustments being the zero level which is sensitive to positional changes of the transducer. The ability of the system to record accurately the volume passing through the pneumotachograph is tested by comparing the computer output with a volumeter and spirometer. Agreement here is excellent. $\pm 3$ percent. Stability and accuracy of the integrators is tested with a sine wave of known dimensions and again here reproducibility and accuracy are better than 2 percent.

Finally the system is tested by the simulation of a dead space. Calculation of dead space is the most revealing calibration statistic of the machine because the dead space represents the differences between two fairly large values, namely the tidal volume, and alveolar ventilation, which only differ by about 20 percent. Consequently errors in these quantities are reflected in an extreme fashion in the dead space calculations. Therefore a homogeneous carbon dioxide mixture is flushed through the pneumotachograph to simulate a zero dead space. Results of this typically indicate an average mean dead space determination of the order of 5 cc for a total volume of 500 cc passed through the pneumotachograph. Artificial dead spaces of 40 and 95 cc have been constructed. Average of the mean of 10 determinations for the 40 cc dead space was 41.6 cc in one instance and in another 45. Average of the mean of 10 determinations for the 95 cc dead space was 97 in one instance and 92 cc in another. These results lead us to have some confidence in the ability of the equipment. On the other hand, this confidence can only be maintained if calibration is conscientiously and frequently performed.

*Readout*

A multiplexing device connects the five memory circuits to a digital voltmeter sequentially. The digitized values are then printed or punched out. The first system, the printing system, is a slow speed unit consisting of a multiplexing device (Dymec C 2900 A) a digital voltmeter (Hewlett-Packard 405 CR) and printer (Hewlett-Packard 561 B). This system can read out five parameter in approximately 2.5 seconds. This speed is adequate as long as we do not have to have an observation on every breath. (If the readout sequence is not completed the integrators are merely reset and computation recommences. The integrators are not connected to the memory units in this situation).

The other system is faster and consists of a similar stepping switch type of multiplexing device (Dymec C 2901 A) which connects the memory circuits through a 5-space digital voltmeter with a 10 ms sampling time, (Dymec 2401). The output of this is put on a punched paper tape by a teletype unit (BRPE-11). This latter system is of course much faster and will read out 5 parameters within approximately 7/10 of a second. This latter format is also much more convenient as it can be read directly into the digital computer (CDC 160 A). With the printing system data must be transferred onto cards, which is rather tedious.

*Programming*

Several programs are then available to us. The first program simply removes the scale factors used in the analog equipment and punches in the conventional units. Several types of manipulation are performed upon the scaled data of which a few examples are as follows.

We might desire a plot of alveolar ventilation rate against end-tidal carbon dioxide tension. This type of plot is useful in studies of the sensitivity of the respiratory center and the effect of drugs upon it. It is usually necessary to smooth this plot. The technique employed is to average the ventilatory rates over five breaths. Similarly the end-tidal carbon dioxide tension is averaged over five breaths (each tension is weighted by the time of the breath to obtain a meaningful average). This type of plot has been used by us extensively in assessment of the depressant effects of narcotics.

Another typical problem is determination of the relationship between tidal volume and alveolar ventilation. The latter is determined by the use of the Bohr formula above. This is a fairly elementary program and a plotting routine is incorporated here also to avoid the tediousness of plotting the large amount of data.

Using the formula for the case when the inhaled concentration of $CO_2$ is not zero, we must compute the net output of $CO_2$ for each breath, as the denominator for the previously derived formula:

$$V_A = \frac{V_E CO_2 - V_I CO_2}{F_A CO_2 - F_I CO_2}$$

For this case the analog equipment is adjusted to compute the product of the concentration signal and all the flow signal, rather than the rectified signal.

By an obvious adaptation of the above program we can plot the net rate of $CO_2$ production against time for any time interval of interest. Such a plot is of interest because this parameter indicates the overall rate at which blood is returning from tissues in a normal metabolic state. A sharp drop in $CO_2$ output would indicate that the rate of return blood to the heart was reduced or that there had been a severe metabolic disturbance. Such information could be useful for an anesthesiologist during a difficult procedure.

Comparisons between the partial pressure of $CO_2$ in the arterial blood and in the lung gases are of interest inasmuch as any great differences reflect inefficiencies in the lung as an exchange device. True comparison is not usually made directly, but rather the 'physiological alveolar ventilation" is determined. This somewhat empiric parameter is the result of replacing the $F_A$ in the Bohr formula by $F_a$, i.e., the fractional concentration corresponding to the partial pressure of $CO_2$ in arterial blood. Comparing the volume so obtained with the "alveolar ventilation volume" $V_A$, defined in the introduction, allows us to express the inefficiency in terms of a volume of the lung (referred to as the alveolar dead space = $V_A$ (phys.) – $V_A$) which receives an adequate blood supply but an inadequate gas supply.

It would be easy to extend the above techniques to obtain many other parameters of respiration, of interest to the respiratory psychologist and the clinician, such as the timed vital capacity, one second expiration, etc.

## CONCLUSION

Techniques are outlined for rapid data processing of respiratory parameters. It is suggested that these techniques are much more efficient than the classical techniques of chemical analysis, etc. Much more data is obtained and the maximum number of parameters can be calculated from an indivdual experiment. It is suggested that we can have a fruitful union of medicine and data-processing technology.

# COMPUTER SIMULATION — A SOLUTION TECHNIQUE FOR MANAGEMENT PROBLEMS

Alan J. Rowe
*Graduate School of Business*
*University of Southern California*
*Los Angeles, California*

Although computers are currently being used primarily for rapid processing of data, there is little doubt that computer-processed information will be a requirement in providing management with timely and accurate data for evaluation, analysis, and as an aid in decision making. At the top management level, decisions are concerned with directing the organization and providing means of assuring its survival. To achieve maximum effectiveness at the operating level, plans and policies must be applied to the available resources, subject to specified constraints and risks. However, there is generally insufficient information for these decisions, and they often cannot be structured as a set of procedures. But, most important, policy decisions are based on a blend of intuition, experience and emotion.

Looking more specifically at the management control process, measurement, reporting, evaluation, decision rules, and feedback are susceptible to computer processing. However, the decision criteria, plans and objectives are still subject to human judgment. At the operating level, where the physical processes in a system are applied, there is the highest opportunity for computer application. Numerous examples exist of automated data processing, essentially in production and inventory control and accounting. Examples of decision rules applied to physical processes can be found in the Journal of Operations Research, Management Science, and Journal of Industrial Engineering. The interesting fact, however, is that in all of these instances of automated data processing, supervisors are still required to deal with the workmen who actually operate the processes. Specific data processing activities can and have been automated, yet management still performs the basic decision functions.

## THE USE OF COMPUTER SIMULATION

In view of the intricately complex nature of large business systems, it is difficult to evaluate new management concept or system designs. Direct experimentation poses almost insurmountable problems due to disruptions, uncontrolled results, length of time required, and possibility of costly mistakes. Computer simulation, on the other hand, has been shown to provide a suitable methodology to study business system behavior under a variety of conditions, and provide a means for analysis of simultaneous interaction of the many system variables to yield valuable insights. In view of its capability of rapid interrogation of system performance, simula-

tion is becoming an integral part of "real time systems."[1]

Computer simulation can be considered as an attempt to model the behavior of a system in order to study its reaction to specific changes. The simulation model is seldom an exact analogue of an actual system. Rather, it is an approximation of continuous time dependent activity. If the properties and elements of the system are properly defined, then the tracing through by the computer of the simultaneous interaction of a large number of variables provides the basis for studying system behavior. A model of the system indicates relationships which are often otherwise not obvious and has the capability of predicting system behavior which results from changes in system design or use of alternate decision rules.

For many years engineers have used scaled models to simulate system response. The armed forces have used exact duplicates of operating systems for training. There have been laboratory studies which can be considered similitude or an attempt to duplicate reality in a laboratory environment. This has been extended to the use of management games where people interact with the output of a computer and make decisions on information received. Computer simulation has been directed toward the use of models of the behavior of a system so that the results correspond to the problem being studied. Abstract mathematical models, on the other hand, are used for problems which correspond with reality to a sufficient degree to produce useful solutions.

Not only has simulation increased in use as a means for studying and understanding new problem areas, but it has a number of distinct advantages. Once a simulation model is completed, the time for experimentation is considerably reduced. The cost of simulation models is now being reduced to the point where for larger problems it is an extremely economical tool. The fact that all the work is done in a computer rather than a laboratory or actual operating environment provides better experimental design and control. The ability to explain the simulation model in terms of a real problem is a far more useful tool than some of the analytic techniques which cannot be described to management or the potential user.

## PROBLEMS IN SIMULATION

Although simulation has many advantages, one should not overlook the difficulty involved in developing a model, programming it on a computer, and utilizing the results. Although computer simulation has been used for a number of years, there are still many pitfalls that must be avoided. One of the greatest difficulties is that of developing a suitable model. Another is the use of computers in the simulation process which poses a number of problems, including computer programming, search techniques, data storage and retrieval, function generators, etc. The computer programming problem has in many instances proven to be a major stumbling block.

In recent years there have been a number of approaches taken to minimize the programming problem. One is the development of models to study a specific area, such as Job Shop Simulation.[2] Using this type model, the user is required to provide appropriate data and a description of the facility to be studied, and the computer program needs little modification. A similar approach has been taken in the Gorden General Purpose Simulator.[3] A somewhat more general approach to this problem has been tackled by the use of the DYNAMO Compiler,[4] in which a set of equations is submitted to the computer, which in turn compiles these and generates a computer program. Therefore, once the model is completed, no further programming is required. As an alternative to writing directly in machine language, a simulation language has been developed called Simscript.[5] Once the model is written, no further programming is required. The Simscript language has all the flexibility of computing language but much of the simplicity of a special purpose approach. Quickscript[6] and programming by questionnaire[7] are extensions of this approach to developing useful simulation languages. Thus, depending on the type of problem being undertaken, it is possible to use a variety of approaches to obtain a computer program. Several of the computer manufacturers have developed standard programs which are readily available and require no further computer programming effort.[8]

A second problem is in the area of experimental design. Considerable effort is often expended in an attempt to obtain information and is often done in an inefficient manner based upon poor input data. It is therefore necessary to consider computer simulation as an equivalent to a laboratory experiment. Before any simulation is undertaken, areas of payoff or urgency should be established and the feasibility of completion of the project with estimates and

budgets should be provided. When defining the problem, there should be careful observations and correct statements concerning what is being studied and discussions held with experienced personnel. Preliminary approaches or brainstorming should be undertaken in order to attempt to define solutions to the problems being studied. Organization of the data, the use of sample vs. exhaustive representation, and the use of statistically designed experiments should all be incorporated. This becomes particularly important when trying to state on a rigorous basis the comparison of one system design to another. Simply because a problem is run on a computer does not mean it is either valid or statistically significant.

A number of fairly significant techniques have been developed for analysis and evaluation of data.[9] Some of these are referred to as Monte Carlo sampling or importance sampling. In these techniques the data are handled in such a way as to minimize the amount of data required and to maximize the information that can be derived from the manipulation of the data. In many applications the use of analysis of variance or regression analysis is very important. It is necessary in evaluating the results of a simulation to have the appropriate criteria and measures of system performance. These, of course, do not depend on the simulation but rather on the user.

The problem of modeling is important since the results of simulation are no better than the model used. A model provides a formal statement of system behavior, in symbolic or mathematical form. The model should be constructed so that the parameters, variables, and forcing functions correspond to the actual system. The parameters should include properties which are sufficient to define the behavior of the system; whereas the variables are the quantities which describe the behavior for a given set of parameters. The forcing function provides the stimulus, external to the system, which causes the system to react. For example, job orders which enter a production system cause men to work, machines to run, queues to form, etc. In this way, job orders become the forcing function for the system. Whatever particular form is used, a model provides the frame of reference within which the problem is considered.

A model need not duplicate actual conditions to be useful. The model should be designed to predict actual behavior resulting from changes in system design or application of new decision rules. Prediction implies an understanding of the manner in which the system reacts; that is, being able to specify the outputs for a given set of inputs.

Models are merely the basis for testing new ideas and should not become ends in themselves. The simpler the model, the more effective for simulation purposes. Tests should be made prior to model building to determine the sensitivity of the characteristics which are incorporated. Typically, certain key characteristics contribute the majority of the information to be derived from simulation. Other characteristics, although more numerous, do not contribute much to the final system design. In this sense, simulation can be considered as sampling the reaction of a system to a new design. It is imperative, therefore, that a representative sample be taken, rather than an exhaustive sample. Thus, the number and type of characteristics to be included should be carefully selected.[10]

The major task of simulation is reached at this point. A logical model, which is merely descriptive, is not suitable for computer simulation. The model must be modified to suit the particular computer on which it will be programmed. Factors such as kind of memory, speed of computation, and errors due to rounding must all be taken into account. Simplification is often necessary due to speed of computation or limitation of the computer memory. The method of filing information and representing time are also significant problems. Which data to accumulate, and at what point in time, often are difficult to decide beforehand. Thus, the program must be flexible and easy to change.

As computer programming proceeds, there is generally feedback which provides the basis for further modification of the model. At the outset, the decision must be made whether to make the program general or special purpose. The type of programming changes radically, depending upon the end use of simulation. Modular programming which treats each section independently provides flexibility at a small cost in computation time and storage. In view of the many logical relations which exist in systems, computer programming represents an important aspect of the problem.

## SUCCESSFUL APPLICATIONS OF SIMULATION IN MANAGEMENT PROBLEMS

A considerable body of literature exists covering

the use of simulation in various applications.[11,12,13] As shown in Fig. 1, simulation should be thought of as a continuum, starting with exact models or replication of reality at one extreme, with completely abstract mathematical models at the other. When viewed in this manner, the breadth of simulation can be appreciated.

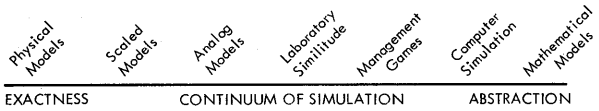



Figure 1.

The wide variety of simulation applications is somewhat astounding. Not only has simulation been extremely successful for purposes of studying physical systems but it has been used for such diverse applications as the study of personality,[14] election results, gross economic behavior, etc. In order to evaluate where simulation is most effective, it is probably best to categorize problems as involving physical and non-physical systems with high and low risk decision alternatives.

As shown in Fig. 2, the area for greatest success is physical problems having very low risks. The poorest applications are nonphysical problems having high risk or little data. This situation may change as the simulation technique is applied to a broader class of problems.

A review of the literature indicates many successful applications of simulation in business.[15,16,17,18,19] A

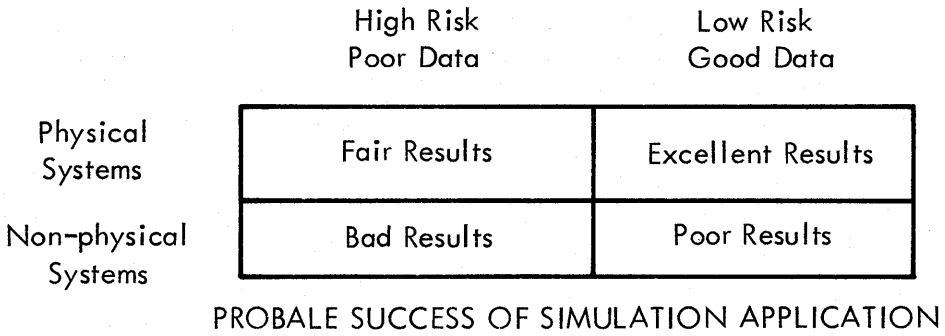



Figure 2. Probable success of simulation application.

publication[20] "Simulation-Management's Laboratory," based on a fairly extensive survey of simulation applications, shows the applicability to a wide variety of management problems. Some of the applications include:

| Company | Management Decision Area Simulated |
|---|---|
| 1. Large Paper Company | Complete order analysis |
| 2. U. S. Army Signal Supply | Inventory decisions |
| 3. Sugar Company | Production, inventory, distribution |
| 4. British Iron and Steel | Steelworks melting operation |
| 5. General Electric | Job shop scheduling |
| 6. Standard Oil of California | Complete refinery operation |
| 7. Thompson Products | Inventory decisions |
| 8. Eli Lilly & Co. | Production and inventory decisions |
| 9. E. I. DuPont | Distribution and warehouse |
| 10. Bank of America | Delinquent loans |

In another survey by Malcolm,[21] the following applications are described:

| *Company* | *Problem Simulated* |
|---|---|

| | | |
|---|---|---|
| 1. | Eastman Kodak | Equipment redesign, operating crews |
| 2. | General Electric | Production scheduling, inventory control |
| 3. | Imperial Oil | Distribution, inventory |
| 4. | United Airlines | Customer service, maintenance |
| 5. | Port of New York | Bus terminal design |
| 6. | Humble Oil | Tanker scheduling |
| 7. | U. S. Steel | Steel flow problems |
| 8. | I.B.M. | Marketing, inventory, scheduling |
| 9. | S.D.C. | SAGE Air Defense |
| 10. | Matson | Cargo transportation |

These lists are not meant to be all-inclusive, but rather indicate the variety and type of problems that have been solved by simulation.

## THE USE OF SIMULATION FOR SCHEDULING JOB SHOPS

Scheduling of job shops has long been considered a critical problem. Extensive work using analytic techniques to find a suitable solution were tried. However, in view of the large-scale combinatorial nature of this problem, no solution was found except for extremely small cases. Extensive models have been developed over a period of years and have evolved into what today is known as the Job Shop Simulator. Although the computer program cost a large sum of money and took almost two years to develop, the Job Shop Simulator has been used successfully in a large number of companies. Notably, it has become an integral part of the manufacturing function at General Electric and has been used extensively in many other companies, including the Hughes Aircraft Company.[22]

The kind of decisions that can be aided by the use of this type of simulation are the following:

1. Establishing required capacity in terms of equipment, facilities, and manpower in order to meet unpredictable customer demand.
2. Examination of alternative types of demands and the capability of the system to respond.
3. Examination of the inventory problem relating equipment utilization to cash requirements and customer demand. (It is possible to meet customer demand by maintaining large inventories.)
4. Development of appropriate scheduling decision rules to maintain a minimum inventory and meet specified delivery requirements.
5. Study the operation of a physical facility through the appropriate use of forecasting techniques, load level techniques, scheduling decision rules, and priority decision rules.

In addition to specific decision areas, there is the information generated from the simulation which provides the basis for feedback on performance so that management can make decisions on the number of shifts to run, need for additional equipment or capacity, or amount of cash to maintain for adequate inventory. The use of this particular program has been extended to an operational system at the Hughes Aircraft Company for real time manufacturing control. The Job Shop Simulator was first used to examine alternative scheduling decision rules. These rules, in turn, provided the basis for developing a supplemental computer program which is used to generate the factory job order status on a daily basis. This computer program, by application of priority decision rules, is used to generate new priority lists each day, taking into account all occurrences for the given day. Thus, the system operates on essentially a daily cycle with all information current and correct as of that point in time. This type of real time application appears to offer considerable opportunity for the use of simulation in industry.[23,24]

## STUDYING BUSINESS SYSTEM BEHAVIOR

Considerable effort has been extended to develop models of the total business system. Several efforts along these lines have been undertaken at SDC,[25] Stanford Research Institute,[26] IBM Corporation,[27] and M.I.T.[28] A notable example of work being undertaken in this area is by the Industrial Dynamics Group at M.I.T. which is concerned with studying total system behavior. The basic premise of this latter simulation is that the dynamic interaction of system variables and information feedback leads to

amplification, oscillation and delays in system performance. The behavior of a system, then, is the result of desired objectives and the decision rules employed to carry out these objectives. Thus, where there is an attempt to make corrections and adjustments in flow rates, there is the possibility of delays or amplification or there may be conflicts between short and long-term objectives. Forrester in his book on Industrial Dynamics describes a number of studies that have been undertaken and describes future studies of total management systems.

In providing the means for experimenting with a total business system, a quantitative formulation of the various behavioral characteristics, component interdependencies, system flows and stochastic functions is required. The formulation is used to develop a simulation program which can trace the activities of the system as they change in time. In this way, a large number of variables can be examined simultaneously, without explicit knowledge of their interdependencies.

A model of a business system is concerned with a "decision network" rather than an explicit characterization of the decision maker per se. The difference stems from studying the information flow and decision rules in the system, as contrasted with studying the behavior of the individual. To the extent that a model faithfully characterizes the behavior of a given business and that suitable decision criteria can be established, a computer model can generate information useful in the study of business problems. Further, the computer is capable of providing summaries of the information generated during the simulation to permit evaluation of experimental designs or to permit useful insights on system behavior.

The characteristic behavior of decision makers in response to information provides the basis for studies concerned with organizational aspects of information flow. The density of communication linkages among decision makers provides data which could be used to establish critical decision points in a system. Further, a communication network linking the managers with the operations provides the means for establishing feedback control loops. In a sense, management is linked with the resources of the business via information flowing through a communications network. Characteristics of the information flow among managers and between the managers and the operators provides one of the basic measures of system behavior. In an actual business, much of the information generated is not pertinent to the direct operation of the business, and often informal communication channels provide useful information. In a sense, the decision maker has surveillance over a given number of decision points, which, when linked to other decision points, defines the underlying operational structure of the business.

## MODELING THE ACTIVITIES AND FUNCTIONS OF A BUSINESS SYSTEM

A schematic representation of the information flows among various functions and activities in a typical manufacturing (resource transforming) business system is shown in Fig. 3.

Central to the system is the decision-communication network, which is normally thought of as the management function. The decision network is linked to the resource transformation or operations subsystem through the various organizational levels. In a computer program, the simulated information generated would be used to execute the decisions in a synthetic manner. Inputs to the decision network include the system contraints or policies, system functions and environmental factors. These too can be represented by flows of information via the communication network.

Inputs to the operations subsystems include environmental factors, customer orders, capital resources, material, etc. In addition, transients or perturbations in the operations subsystem could be used to introduce variability in performance. Outputs of the system enter distribution systems, warehouses or finished goods storage. Although there is considerable detail associated with the operations subsystem, a computer model need merely treat the information aspects as they interact with the decision network.[29]

The study of the behavior of a total system requires an explicit description of the time dependencies among the components. The primary concern in the model discussed here is the conversion of resources into goods and services. By determining appropriate strategies in relation to risks, it is possible to control rates of change of production, work force stabilization, growth rate, cost-pricing, response to demand, and the relation of income to investment.

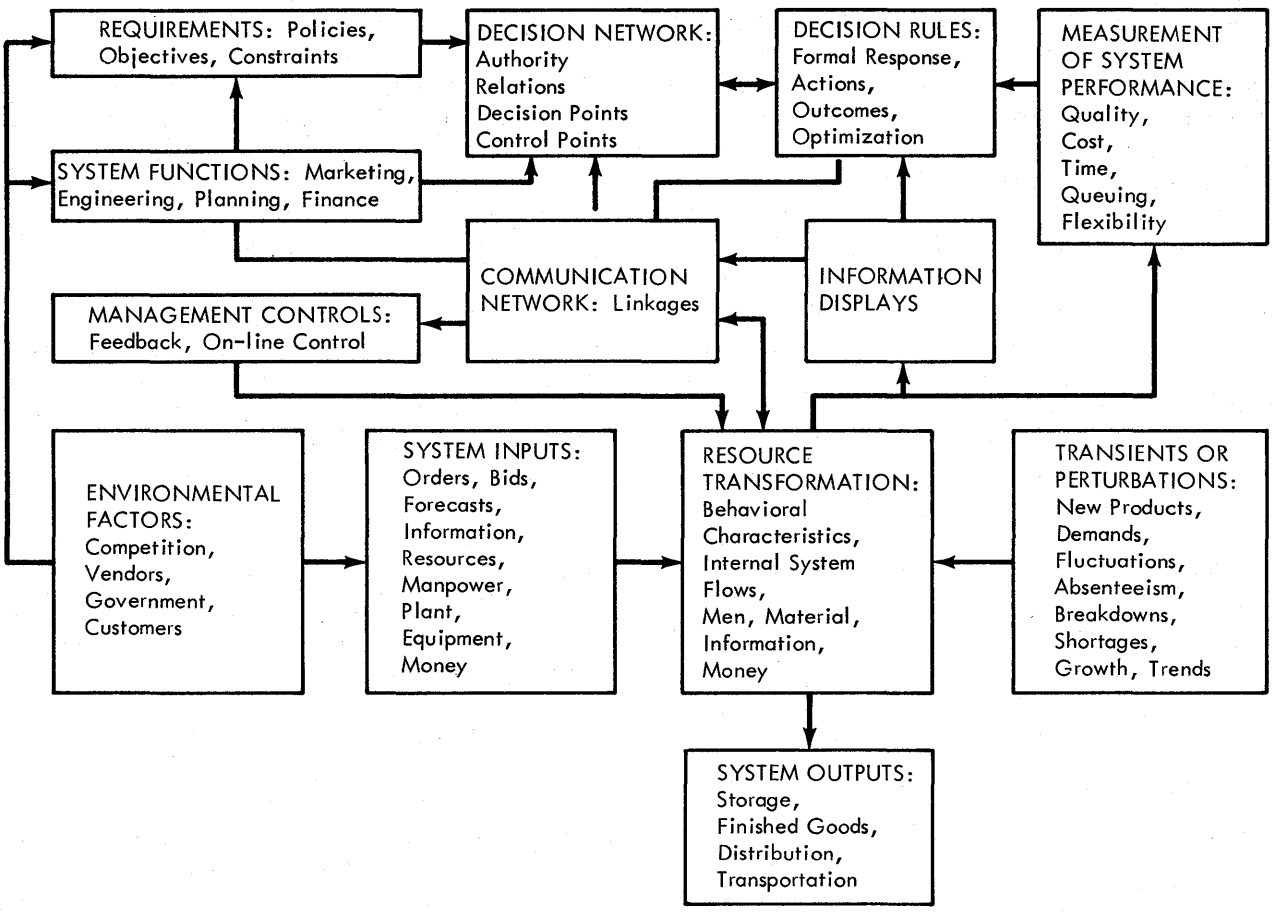



Figure 3. Activities and functions of a business system.

Since computer simulation is used to trace the change in the variables across a time domain, decision rules can be made functions of the state of the variable rather than using expected values. In this respect, simulation differs from dynamic programming or gaming strategies which depend on statistical estimates as the basis for optimization. Forcing functions, which trigger the decision rules, must also be specified and are related to information flow in the system.

Since system optimization involves many variables, it is necessary to consider the many combinatorial effects. Simulation is a means for examining a large number of variables simultaneously. However, the solution is not unique, but provides an estimate of the distribution of expected system performance. Thus, although all the combinations could not possibly be enumerated, sampling results tend to form relatively stable and determinable distributions.

## DEFINING SYSTEM FLOWS

Flows within the system can be separated into information flow (paperwork, reports, etc), material flow, resource flow, and manpower flow. Each has its own characteristics and is therefore modeled differently.

Starting with information flow, the channels or network determines the destination of the information, and the transmission media determine the speed and message type. Information content is a function of the data, format, and timeliness. Transformation, distortion, and errors should be included, as well as the queueing effects at the decision points in the system.

Material flow has received the most attention in simulation and operations research studies. Thus, a considerable body of literature exists which could provide the basis for modeling. Reorder rules, safety stocks, value analysis, collation studies, scheduling rules, and stocking policies have been well doc-

umented. There are a number of additional considerations which might also be of interest, such as surge effects, queueing effects, interdependence of component parts, work-in-process flow rates, and resource utilization.

Resource and manpower flow are more difficult to solve since factors external to a simulation model may have the major influence. Nonetheless, there are aspects of these factors which can be profitably studied. For example, what are the cash flow requirements in a marginally capitalized business? What is the relation between demand variation and capacity? How does capacity and capital requirements change with different products, number of shifts, skill and mobility of manpower? These and similar questions are readily susceptible to study via computer simulation.

The internal system, in addition to the basic flows, has a number of other characteristics. In particular, it is necessary to structure certain behavioral patterns such as:

- demand and shipping patterns
- value distribution among products
- variability in man-machine performance
- learning-curve effects
- various lead-time distributions

Rather than attempt an exhaustive description of the physical characteristics of the internal system, the considerations discussed are intended to provide some measure of the complexity and difficulty in modeling the business system.

There are a number of environmental considerations, as well as system inputs and outputs, which should also be taken into account in the modeling. The number and type of competitors, customer demands, vendor characteristics and legal or civic factors all should be specified in developing a computer model of a business. Furthermore, forecasting of demand, gaming strategies, competitive pricing, and advertising policies are, in effect, the control of system response to variable demand. Thus, for example, maintaining standby capacity in anticipation of orders, having a complete product line, or carrying large safety stock in inventory, are all means of response which are really control of the system.

From a total system viewpoint, the availability of cash affects the above considerations; that is, carrying large inventories may determine the plant capacity required or the advertising budget. In this sense, cash flow permeates all aspects of system behavior and thus affects control. Similar considerations enter into the make or buy question, material and tooling purchases, employment stabilization, etc. It is an explicit treatment of the many interdependencies which provides the basis for total system control.

The modeling of a total business system which incorporates the many considerations discussed would undoubtedly involve numerous details. Thus, where possible, transfer functions or aggregations should be used rather than the precise flows or system characteristics. Not only does aggregation provide considerable savings in modeling, but, often more significantly, it helps reduce the size and complexity of a computer program. The modeling is, after all, designed to answer given questions or explore new areas and, therefore, should be governed by these considerations.

## CONCLUSION

It is apparent from the many successful applications that simulation will continue to grow in importance and become a truly operational tool for management decisions. There is still a vast number of problems that can be tackled, ranging from the study of specific economic problems to total company system problems.[30] Because of its many advantages and because of the need for improved techniques in management, simulation appears as one of the most useful tools that has come on the horizon. There is still much required in the way of improved modeling, reduced cost of programming, improved outputs, etc. However, none of these problems is unsurmountable and the evidence is quite clear that there are continued improvements on all fronts. Thus, we can expect to see the use of simulation as a normal part of business operations in the not too distant future.

## REFERENCES

1. A. J. Rowe, "Real Time Control in Manufacturing," *American Management Association Bulletin,* No. 24 (1963).

2. A. J. Rowe, "Toward a Theory of Scheduling," *Journal of Industrial Engineering,* vol. XI, no. 2 (March-April 1960).

3. G. Gordon, "A General Purpose Systems Simulator," *IBM Systems Journal,* vol. I (September 1962).

4. J. W. Forrester, *Industrial Dynamics,* M.I.T. Press, Cambridge, Mass. 1961.

5. H. Markowitz, B. Hausner and H. Karr, *Simscript: A Simulation Programming Language,* Prentice Hall, Inc., New York, 1963.

6. F. M. Tonge, P. Keller, A. Newell, "Quickscript—A Simscript-Like Language," *Communications of the ACM,* vol. 8, no. 6 (June 1965).

7. A. S. Ginsberg, H. M. Markowitz, R. M. Oldfather, "Programming by Questionnaire," RM-4460-PR, The RAND Corporation (April 1965).

8. H. Markowitz, B. Hausner and H. Karr, "Inventory Management Simulation," I.B.M. Data Processing Information (April 1961).

9. S. Ehrenfeld and S. Ben Tuvia, "The Efficiency of Statistical Simulation Procedures," *Technometrics* (May 1962).

10. A. J. Rowe, "Modeling Considerations in Computer Simulation of Management Control Systems," SP-156, Systems Development Corporation (March 1960).

11. D. G. Malcolm, Editor, "Report of System Simulation Symposium," American Institute of Industrial Engineers (May 1957).

12. W. E. Alberts and D. G. Malcolm, "Report of the Second System Simulation Symposium," American Institute of Industrial Engineers (Feb. 1959).

13. W. E. Alberts and D. G. Malcolm, Report No. 55, "Simulation and Gaming: A Symposium," American Management Association (1961).

14. S. S. Tomkins and S. Messick, "Computer Simulation of Personality," John Wiley & Sons, Inc., New York, June 1962.

15. J. Moshman, "Random Sampling Simulation as an Equipment Design Tool," CEIR (May 1960).

16. A. Rich and R. T. Henry, "A Method of Cost Analysis and Control Through Simulation," Linde Company.

17. H. N. Shycon and R. B. Maffei, "Simulation —Tool for Better Distribution," Harvard Business Review (Dec. 1960).

18. D. G. Malcolm, "System Simulation — A Fundamental Tool for Industrial Engineering," *Journal of Industrial Engineering* (June 1958).

19. K. J. Cohen, "Simulation in Inventory Control," Chapter XV, *Production Planning & Control,* R. H. Brock and W. K. Holsterin, Merrill Books, Columbus, Ohio, 1963.

20. D. G. Malcolm, "Simulation—Management's Laboratory," Simulation Associates, Groton, Conn. (April 1959).

21. D. G. Malcolm, "The Use of Simulation in Management Analysis—A Survey and Bibliography," SP-126, System Development Corporation (Nov. 1959).

22. E. LaGrande, "The Development of A Factory Simulation System Using Actual Operating Data," *Management Technology,* vol. 3, no. 1, (May 1963).

23. A. J. Rowe, "Management Decision Making and the Computer," *Management International,* vol. 2, no. 2 (1962).

24. M. Bulkin, J. L. Colley, H. W. Steinhoff, Jr., "Load Forecasting, Priority Sequencing and Simulation in A Job Shop Control System," unpublished paper, Hughes Aircraft Co. (May 1965).

25. M. R. Lackner, "SIMPAC: Toward A General Simulation Capability," SP-367, System Development Corporation (Aug. 1961).

26. C. P. Bonini, "Simulation of Information and Decisions in the Firm," Stanford University (April 1960).

27. D. F. Boyd and H. S. Krasnow, "Economic Evaluation and Management Information Systems," *I.B.M. System Journal,* vol. 2 (March 1963).

28. E. B. Roberts, *The Dynamics of Research and Development,* Harper & Rowe, New York, Jan. 1964.

29. A. J. Rowe, "Research Problems in Management Controls," *Management Technology,* no. 3, December, 19

30. R. Bellman and P. Brock, "On the Concepts of A Problem and Problems Solving," *American Mathematical Monthly,* vol. 67, no. 2 (Feb. 1960).

# THE ROLE OF THE COMPUTER IN HUMANISTIC SCHOLARSHIP

Edmund A. Bowles
*Department of Educational Affairs*
*IBM Corporation*
*Armonk, New York*

Within the past dozen years or so, the computer has made itself felt in every aspect of our society. One hundred years ago, it was the Industrial Revolution which wrought profound changes in the economic and social fabric of the western world. Today there is an upheaval of comparable force and significance in the so-called Computer Revolution. Indeed, Isaac Auerbach has characterized the invention of the computer as being comparable to that of the steam engine in its effects upon mankind. He predicted that the computer and its application to information processing "will have a far greater constructive impact on mankind during the remainder of the 20th Century than any other technological development of the past two decades."[1]

To cite but one example, the so-called information explosion has affected all areas of knowledge, scientific and humanistic alike, making analyses both increasingly complex and time-consuming. During much of the century, knowledge is estimated to have doubled every ten years, and journals are proliferating at the astounding rate of over three per day. An overriding problem, or challenge, if you will, is the integration of this new knowledge into the existing world of scholarship as well as the dissemination of these new ideas and concepts within the intellectual community at large. Here again, one turns inevitably to the computer. In fact, we have now reached the point where even an anthropologist speaks about the heritage of a culture being stored in physical objects such as books and computer tapes! More important, however, is the existence of data processing and information retrieval as a new and useful tool of tremendous potential; a fact that must be recognized and accepted by the nonscientific community of scholars.

Let us consider for a moment the principal advantages of the computer to humanistic scholarship in general. Its incredible speed allows the scholar to accomplish in a short time what would otherwise take him a whole lifetime of drudgery to accomplish. Its storage or memory constitutes an infinitely more reliable repository than the mind of the proverbial absentminded professor. Its great accuracy is completely dependable even when untold mountains of statistical data require handling. And finally, its automatic operation is not subject to the vagaries of human fatigue, periods of interruption, or even of mood. To the humanist, I would suggest the most important of these advantages is the immense saving of time gained by the use of computers. It is useful to remind ourselves that the *Oxford*

*English Dictionary* took some 80 years to complete with several generations of editors. Jakob and Wilhelm Grimm's monumental *Deutsches Wörterbuch* began to appear in 1854 and wasn't completed until 1960. Similarly, the manual indexing of the complete works of Thomas Aquinas (approximately 13 million words) would take 50 scholars 40 years to accomplish, but thanks to the computer, the total time required by a few scholars working mainly in Italy was less than one year.[2] A concordance to the Revised Standard Version of the Bible was produced on a high-speed computer within a period of several months as compared to the King James Concordance of the last century which took 54 scholars 10 years to accomplish.[3] The deciphering of the Mayan hieroglyphic script by Russian mathematicians, we are told, took only 40 hours of computer time for which a human being would have needed thousands of years to accomplish.[4] All this leads to the inevitable conclusion that there are a number of scholarly tasks—call them the more tedious clerical chores, if you will—that in this age demand the use of the computer. Certainly, one can no longer think of concordances, dictionaries, or projects involving masses of statistical data and numerous cross-correlations without bringing into play the tools of data processing. Thus, the computer's power can be harnessed to relieve scholars in the humanities of some of their most burdensome activity while at the same time providing their research with the benefits of greater speed and accuracy. More important by far, however, are the more creative uses of data processing as an aid in such areas as stylistic analysis. More of this in a moment.

Unfortunately there is a great deal of suspicion, fear, and ignorance on the part of the humanist concerning the computer and its legitimate role in scholarship. Some see the machine as eventually making decisions that man himself should make. Others find sinister implications in every technological advance, maintaining the attitude that the humanities and technology don't mix. Finally there are those who, ignorant of mathematics, fear they are totally and forever incapable of comprehending the computer and therefore dismiss it. Although no one would suggest burning at the stake the maker of a computerized concordance, as was almost the fate of the first person to make a complete concordance of the English Bible in 1544, the computer-oriented humanist does face some formidable oppositions.

Ironically, while the impact of the computer, as stated at the beginning of this paper, may be compared to the influence of the Industrial Revolution one hundred years ago, there is also an analogous reaction among many highly placed scholars to so-called computer-oriented humanistic research. Not that any misguided intellectual will physically attack "the dark Satanic mills," as did their Luddite ancestors, but we do have their counterparts today who, kowing little of the computer's advantages and limitations, damn the machine as not only useless but dangerous to the world of scholarship. To some of the older, more conservative scholars, putting lines of verse into a computer seems profane, like putting neckties into a Waring Blender, as one professor remarked. More seriously, there are academicians who place no value on the scholar's time, like the professor who, when told that data processing would vastly speed up the production of an English-Old Iranian dictionary, went so far as to say that what is *not* needed is a computer but rather enough money for someone to be completely free for several years so he could sit down and do the necessary work. A Scottish minister and mathematician sent an article on the use of the computer in biblical scholarship to a publisher. It was returned promptly with a notation, "I do not understand this, but I am quite sure that if I did understand it it would be of no value." One scholar said a few years ago that, "If you have to use a computer to answer a question, it is not a question which I would care to put." Fortunately for the humanities, things are changing rapidly.

Let us reveal the negative position for what it is as we now examine some representative projects within various humanistic disciplines which have made extensive use of the computer as both an important and productive tool of scholarship.

In the field of archeology, for example, the computer is of use in studies of shards or fragments of artifacts found in the diggings of ruins. In this connection, Jesse D. Jennings of the University of Utah suggests constructing a matrix of coefficients of similarity of one artifact to another, and thus to all others within a given corpus of objects. The two basic problems are classifying shards as to their cultural provenance and reconstructing whole artifacts from broken fragments. Jennings has a body of some 2,600 shards, each of which has 50 attributes. Obviously, this represents an astronomical number

of comparisons to make by hand, and yet for a computer it is a relatively simple task.[5]

Mr. Dee F. Green, a research associate at the University of Arkansas Museum, is using codes and statistical techniques in correlation studies of burial lots from Eastern and Southwestern Arkansas, and in detailed analyses of ceramic, decorative, and technical complexes and traditions exhibited in certain areas of the state. Following the maxim that pottery is "the essential alphabet of archeology," a code was developed for reducing the individual attributes of some 4000-odd pottery vessels to a numerical system for computer handling. Once the material is classified, the various attributes will be sorted into discrete categories and then statistical techniques applied to lump the attributes into statistically meaningful groups, or ceramic types.

Dr. Paul S. Martin and his associates at the Chicago Natural History Museum have been using the IBM 7094 computer at the University of Chicago to process archeological data from the southwestern United States.[6] By this means they have discovered spatial clusters of both pottery types and pottery design elements within a pueblo site. It was found that the clusters themselves tended to be localized in certain well-defined areas of the site. In addition it was found that certain room-types contained specific clusters of artifact types. In each case, the computer was given frequencies or percentages of different artifact or shard types by provenance. The variables were then correlated and submitted to factor analysis. This allowed comparison of room-floors with one another to find out which rooms were similar and which different. This information was then interpreted in terms of room function, social groups, chronology, and so forth.

As can readily be seen, such projects as these involving many thousands of artifacts, each with numerous attributes, as well as the dozens of correlations between them, really demand the use of computers to handle the sheer mass of information and to derive really meaningful results therefrom.

Historians have faced a new impetus for the application of social science research techniques to the analysis of historical political data. The Inter-University Consortium for Political Research at Ann Arbor is amassing a vast amount of raw data transferred to tape storage on American political history. In addition to the formation of a data repository committee, with close ties to the American Historical Association, is the development of an automated data retrieval system to make available to historians and political scientists alike large bodies of information. The American Historical Association has set up an Ad Hoc Committee on the Collection of Basic Quantitative Data of American Political History under the chairmanship of Professor Lee Benson. Election statistics on presidential campaigns from all counties in the United States from 1824 to the present, roll-call votes during each congressional session since 1789, data on federal court cases, and census and ecological information are all being computerized. Further material awaiting such attention exists in the fields of agriculture, business statistics, industry, religion, economic, social, and cultural data, foreign trade, employment, tax data, and housing. The amount of such unpublished information available is staggering. For but one year in American history, the U.S. Census Bureau Catalog includes over 5000 computer tape reels of data in the above-mentioned fields.

The Inter-University Consortium held a training program during the past three summers consisting variously of elementary courses such as "Introduction to Survey Methods," and "Cases in Survey Research," an eight-week Graduate Pro-Seminar in Behavioral Research Methods and Quantitative Political Analysis, and advanced seminars conducted by both the Department of Political Science and the Survey Research Center of the University of Michigan. To the best of my knowledge, this is the first and only attempt within a given field of humanistic endeavor to provide computer training for its constituents.

One of the earliest historical studies involving the use of data processing was the study of Massachusetts shipping during the early Colonial period made by Professor Bernard Bailyn of Harvard University.[7] Faced with the problem of sketching a realistic picture of the subject, he came upon a perfectly preserved shipping record for an 18-year period containing information not only about the vessels registered but about the owners as well; in other words, material of early American social and economic history. Bailyn not only summarized and tabulated this data in comprehensive fashion but used the opportunity to assess realistically the possibilities of applying machine techniques to historical material and to explore the problems of procedure.

The shipping register in question consisted of 1696 entries, each giving information about a ves-

sel and the people who held shares in it. A total of 4725 punched cards were produced which contained all the information available in the register on the ships and their owners. Codes were developed not only for the purely quantitative data but for such qualitative information as names of people and places, vessel types, occupations, building sites, etc. Although numerous problems were encountered along the way, it is significant that Bailyn's book closes with the statement that only with these tools and techniques was the analysis of the register possible at all.

Two other computer-oriented historical research projects involve content analysis. Professor Richard Merritt of Yale University is studying the developing symbols of the American sense of community or identity as reflected in five colonial newspapers. Professors Robert North and Ole Holsti of Stanford University are analyzing the origins of World War I by means of computer techniques for scanning and reporting the appearance of themes and relationships in a large body of historical material pertaining to decision-making during the 1914 crisis.[8] "Communication is at the heart of civilization," but since students of international relations are considerably more restricted in access to data than most social scientists—direct access to foreign policy leaders is severely restricted—one method is to assess their attitudes, values, and assessments by means of a computer content analysis of political documents at times of crisis. North and Holsti constructed a dictionary of words, such as *abolish, accept,* or *armaments*. They are sought out in historical documents and changes in style are noted as the historical crisis grows in severity in terms of verbal effectiveness, strength or weakness, and activity or passivity. By this means one can explore for examantagonism and the degree of cohesion between the ple the relationship between the level of East-West Soviet Union and China.

Professor William Aydelotte of the University of Iowa has used the techniques of data processing to aid in the study of the voting patterns in the British House of Commons in the 1840's.[9] During Sir Robert Peel's ministry, Commons debated and voted on a number of substantial political issues. There were divisions as well on various aspects of religious questions, the army and fiscal reform. There exists an unexploited source in the so-called division lists giving information on all the men in Parliament so far as they voted on the issues in question. The complexity of this material, the richness thereof, the fact that most members of Parliament did not vote consistently "liberal" or "conservative" on all issues made this entire decade of Parliamentary history ripe for computer analysis; a project which resulted in a total of 6441 four-fold tables each of which was punched on a separate IBM card.

Certainly the most well-known use of computers in the field of literature is the construction of verbal indices. Briefly, there are two forms: first, a simple alphabetical list of text showing the frequency or location of the words or both; and second, a textual concordance showing all the words of a given literary work not only alphabetically but in context as well. In this form, each word appears as many times as there are words within the parameter arbitrarily chosen for it along with the relevant passages of which it is a part. Such a concordance is not only of immense value to a literary scholar from the point of view of time saved, but is useful also to those in other disciplines employing literature as source material. For example, concordances to the poetry of American authors have been issued since 1959 by the Cornell University Press. In the case of the works of Matthew Arnold, the lines of verse were punched on IBM cards, one line per card, to which was added the line number and page number from the standard edition and variance from other collations. A separate title card was punched and inserted before each poem. The entire deck of some 17,000 cards was printed out and transferred to magnetic tape. A computer-generated concordance program was then made and ultimately a tape prepared with all the significant words of Arnold's texts arranged in alphabetical order along with their locations. A final print-out formed the basis for the published index.[10]

In similar fashion, Professors Alan Markman and Barnet Kottler of Pittsburgh and Purdue respectively have prepared a computer concordance to five Middle English poems, perhaps the best known of which is *Sir Gawain and the Green Knight*.[11] Dr. John Wells of Tufts University is making a computerized word-index to the Old High German glosses cribbed between some 140,000 lines of medieval Latin text.

Professor Alice Pollin, working with the Computer Center at New York University, produced a guide, or critical index, to the 43 volumes of the *Revista de Filología Española* from 1914 to 1960, cross-indexed by authors, subject matter, and book

reviews. A total of nearly 900 pages was the result, the product of approximately 60,000 punched cards. The printout sheets were reproduced photographically and issued as a bound volume.[12] In this way a methodology for the machine-indexing of periodicals was established.

However, it is in the area of textual analysis that computer-oriented research in literature shows exceptional and exciting promise for the future. The massive comparison of text where there are several or even dozens of sources presents an almost insurmountable problem for the scholar. To compare in complete detail as few as 40 manuscripts might take the better part of a lifetime. It is this type of activity that cries for the use of data processing techniques.

Perhaps the first such effort was the study by Professors Mosteller and Wallace at Harvard University, and actually continuing over a period of years, to solve the authorship question of 12 disputed *Federalist Papers*.[13] Briefly, literary styles of Madison and Hamilton were identified, then matched with the style of each of the disputed papers. Having found such factors as sentence length, vocabulary and spelling to be of no help (the two authors were remarkably alike), it turned out that differences in the use of so-called key function words—particularly those of high frequency such as *from, to, by, upon, also,* and *because* — served to pin down authorship of the papers in question. From a number of computations, Mosteller and Wallace found that most of the disputed documents were written by James Madison. But consider for a moment the problem of Dr. John W. Ellison, a biblical scholar from Massachusetts who, for his doctoral dissertation, studied 309 manuscripts of the Greek New Testament, then went on to prepare a complete concordance of the revised Standard Version of the Bible. Fortunately, he used a computer to assist him in these gigantic tasks. However, there are over 4600 known manuscripts of the whole or part of the New Testament, with cross-fertilization in the copying process that has been going on for a thousand years or more. Here the use of the computer to determine the interrelationships of the manuscripts of the text is mandatory.

Scholars at other universities, while not faced with such vast problems of correlation, are actively engaged in similar work. For example, a definitive edition of the works of John Dryden is being prepared under Professor Vinton Dearing at U.C.L.A. with textual collation aided by the use of the computer.[14] This grew out of an existing corpus of 240,000 *manually* indexed cards. Variant texts of the final section of Henry James's novel *Daisy Miller* are being collated by computer in a pilot project at New York University under Dr. William Gibson. This, too, is designed to aid the editor faced with a number of varying manuscripts or printed editions in making up an *Urtext* or variorum edition by supplying him with a printout indicating the identifies between the versions, sentence by sentence.

Dr. James T. McDonough of St. Joseph's College has demonstrated with the help of a computer that Homer's *Iliad* indeed exhibits the consistency of one poet.[15] His study was in part a response to the persistent question of whether one poet wrote the epic or if it consists of separate, short ballad-type songs by separate authors from various times and places all strung together. McDonough prepared in systematic fashion a metrical index of the *Iliad,* not by spellings but by the rhythmic function of all 112,000 words in their 157 metrical varieties. Once the rhythm of each of the 15,693 lines was coded and punched, an IBM 650 machine was able to isolate the individual words, sort, count, and print out the resulting wordlists in a matter of hours.

Mrs. Sally Sedelow of Saint Louis University has described the use of the computer for a rigorous description and analysis of pattern attributes of text.[16] She makes the observation that while colleagues in linguistics have been making major contributions to such fields as machine translation and information retrieval—and in return, gaining important insights into the structure of language—those in literature have offered very little and gained very little. The aim of such studies is to discover the differences between writers' styles and to shed light on the changes of an individual author's style over a period of time. Known as "computational stylistics," these techniques deal with the parameters of literary style in terms of its constituent elements: rhythm, texture, and form.

Turning now to the field of musicology, Professor Harry B. Lincoln of Harpur College is using the techniques of information retrieval to compile a catalog of musical incipits (that is, brief melodic quotations of the first six to eight notes of a composition), of the entire body of 16th century Italian *frottole,* a known body of some 600 polyphonic compositions. Since the possible permutations and combinations of the beginning notes of such pieces

are almost infinite, both as to pitch and rhythm, such brief quotations represent unique identifications of the compositions from which they are taken. First the incipit is translated into alphanumerical form by means of a code which can activate a photon printer. This is a device with a high-speed rotating disk containing about 1400 characters (in this case, musical), a light source, lens, and photographic film. This code consists of *even* numbers for the spaces, *odd* numbers for the lines of the musical stave, an *H* for a half note, a *W* for a quarter note, and so forth. One punched aperture card with a 35mm photograph of the particular score set in the right-hand side holds information such as the composer, title, voice part, accession or serial number. The second card contains the proper sequence of note representing the incipit coded for the photon device. When computerized, this code causes the printer to raise or lower its focus to the proper line or space and, when the correct note or other symbol is in place, shoot a beam of light through the proper aperture in the disk exposing the film at that time with the desired musical symbol. At the same time, a computer program extracts from this coded information the intervalic order or melodic profile of the particular incipit. This, then, can be compared to other musical sources for instances of borrowings by one composer from another, or from other works of the composer himself.

A second major area is the use of data processing as an aid in the analysis of the structure of music. For example, Professor Bertram H. Bronson at Berkeley has used computer techniques in the study of folk-songs.[17] By means of punched cards, he coded the important elements of folk tunes including range, modal characteristics, prevailing time signature, number of phrases, the nature or pattern of refrains, final cadences, and so forth. In this way, an entire corpus of folk song material can be recorded both fully and accurately. The various elements can then be analyzed for statistical patterns, comparisons, or indeed subjected to any other query consistent with the data.

A computer can also serve to test hypotheses through simulation and models. Dr. Allen Forte of Yale University is applying machine analysis to help provide insights regarding the structure of the atonal music of Arnold Schönberg. The structure of so-called pre-twelve-tone (or nontonal) music is still somewhat of a mystery. Mr. Forte has found the traditional nonmachine forms of analyses lacking and has stated that a structural description of this music would be virtually impossible without the aid of a computer. If I understand the outline of his program, he is formulating a basic working theoretical hypothesis based upon linguistic and mathematical models for musical structure, developing an analytical method to explore his ideas by means of the computer; he then will test the result.

In quite another application of computer techniques in the analysis of musical style, a program can be written to search for meaningful patterns and relationships which, because of the number and quality of variables, might remain obscured and undiscovered if left to the human brain. From these very patterns, the researcher can then develop new and significant hypotheses. An interesting example of this is the proposal of Professor Jan La Rue of New York University to evolve machine language to describe stylistic phenomena in 18th century symphonies, thereby permitting complex correlations and comparisons far beyond the reach of the hand tabulation. Just as the literary scholar has quantified style in terms of form, rhythm, and texture, the musicologist has developed a set of guidelines for the purpose of stylistic analysis, breaking down the various musical elements into sound, form, harmony, rhythm, and melody. This technique is admirably suited to computer procedures when one would wish, for example, to determine whether or not a symphony attributed to one composer was actually written by him. By this technique, the stylistic attributes of a questionable or anonymous symphony could be compared for correspondences with the stylistic quantification of a given composer's known symphonies stored within the computer memory.

Finally, I must mention the Musical Information Retrieval project at Princeton University being carried out under the direction of Professor Lewis Lockwood. This involves a programming language for an IBM 7094 computer by means of which musical data is stored in the computer for interrogation and manipulation.[18] Information representative of each note in its complete context and relationship within the score is coded manually and then stored in anticipation of the questions to be asked. The pilot project at Princeton involves a stylistic investigation of the 22 masses and mass movements of the Renaissance composer Josquin des Près. In its broader aspects this type of program enables the scholar to search for and locate reliably all elements

within a given category, such as accidentals, or all examples of a particular intervalic progression. In addition the computer program can serve as a check on discrepancies between the original manuscript and later editions or transcriptions.

The projects I have just described to you represent but a few highlight from the growing roster of scholars using the computer in humanistic research. A list of such activities published last spring by the American Council of Learned Societies reveals well over a hundred individuals involved with the tools of data processing.[19] When compared to the state of the sciences versus the computer some 10 years ago, the prognosis for the future is good indeed.

In a lecture at M.I.T., entitled, "The Computer in the University," the prediction was made that in a few years the computer may have settled immutably into our thought as an absolutely essential part of any university program in the physical, psychological, and economic sciences. On the basis of what I have said, I think the time has come to amend that statement to include the humanities. Furthermore, within a short time, I believe a knowledge of data processing will become part of the "common baggage" of research tools and techniques required of every graduate student in the liberal arts. I am even tempted to go a step further and state that with the increasing number of courses in programming being offered at our universities the time may come when some students in the humanities may be as fluent in programming as in writing English composition. Certainly, the computer is fast becoming an important and indispensable research tool for faculty and students alike.

The value of such an acquaintanceship can be seen in the case of a professor of art history and archeology at an eastern college. Describing himself as probably the man on campus "least likely to benefit from a computer," he took a short summer course at the college computer center. Later, in reporting on the instruction, he said, "The course profoundly affected the thinking of all of us. This is the important thing—much more important than the machine itself. Of course we know that it is the brains behind the machine that make these miracles possible. Nonetheless, it is a weapon of such power that all intelligent men and women everywhere should know the kind of things it can do. Once we know that, we can devise ways to make use of it." In this connection it is useful to bear in mind Alfred North Whitehead's remarks 40 years ago that "the reason why we are on a higher imaginative level than the 19th century is not because we have finer imagination, but because we have better instruments—which have put thought on to a new level."

Let us, therefore, see the computer as a means of liberation, freeing the humanist scholar from the time-consuming operations of the past; a tool providing him in rapid fashion with a proliferating series of sources in the form of statistics, collations, printouts, cross-references, frequency counts and hypothetical models upon which he may build a research of new dimensions and complexity. Viewed in this light, it is a device the potentialities and applications of which we cannot afford to ignore.

## REFERENCES

1. Isaac Auerbach, "Information: A Prime Resource," in *The Information Revolution, New York Times,* May 23, 1965, p. 4.

2. Paul Tasman, "Literary Data Processing," *IBM Journal of Research and Development,* vol. I, pp. 249-56 (1960).

3. See the preface to *Nelson's Complete Concordance to the Revised Standard Version Bible,* New York, 1957.

4. Felix Shirokov, "Computer Deciphers Maya Hieroglyphs," *UNESCO Courier,* vol. XV, pp. 26-32, (1962).

5. Jesse D. Jennings, "Computers and Culture History: A Glen Canyon Study," Paper delivered at a meeting of the American Anthropological Association, San Francisco (Nov. 21, 1963).

6. See for example J. A. Brown and L. G. Freeman, Jr., "A UNIVAC Analysis of Shard Frequencies from the Carter Ranch, Pueblo, Eastern Arizona," *American Antiquity,* vol. XXX, pp. 162-167, (1964); and Paul S. Martin, "Archeological Investigations in East Central Arizona," *Science,* vol. CXXXVIII, pp. 825-27 (1962).

7. Bernard Bailyn, *Massachusetts Shipping 1697-1714: A Statistical Study,* Harvard University Press, Cambridge, Mass., 1959, esp. pp. 137-141 ("A Note on Procedure").

8. Ole R. Holsti, "Computer Content Analysis as a Tool in International Relations Research," in *Proceedings of the Conference on Computers for the Humanities,* Yale University, New Haven, Conn., 1965.

9. William O. Aydelotte, "Voting Patterns in the British House of Commons in the 1840's," *Comparative Studies in Society and History,* vol. V, pp. 134-163 (1963).

10. Stephen M. Parrish, *A Concordance to the Poems of Matthew Arnold,* Cornell University Press, Ithaca, N. Y., 1959; see also his "Problems in the Making of Computer Concordances," *Studies in Biography,* vol. XV, pp. 1-14 (1962).

11. See Alan Markman, "Litterae ex Machina: Man and Machine in Literary Criticism," *Journal of Higher Education,* vol. XXXVI, esp. pp. 70-72 (1965).

12. Alice M. Pollin and Raquel Kersten, *Guia para la Consulta de la Revista de Filología Española,* New York University Press, New York, 1964.

13. Frederick Mosteller and David L. Wallace, "Inference in an Authorship Problem," *Journal of the American Statistical Association,* vol. LVIII, pp. 275-309 (1963).

14. On methodology, see Vinton A. Dearing, *Methods of Textual Editing,* University of California Pres, Los Angeles, Calif., 1962.

15. James T. McDonough, Jr., "Homer, the Humanities, and IBM," in *Proceedings of the Literary Data Processing Conference,* IBM Corporation Yorktown Heights, N. Y., 1964, pp. 25-36.

16. Sally Y. Sedelow, "Some Parameters for Computational Stylistics: Computer Aids to the Use of Traditional Categories in Stylistic Analysis," ibid., pp. 211-229.

17. Bertrand H. Bronson, "Mechanical Help in the Study of Folk Song," *Journal of American Folklore,* vol. LXII, pp. 81-86 (1949).

18. See Michael Kassler, "A Simple Programming Language for Musical Information Retrieval," (Project 295D, Technical Report No. 3), Princeton University, Princeton, N. J. (1964).

19. "Computerized Research in the Humanities: A Survey," *ACLS Newsletter,* vol. XVI, no. 5, pp. 7-31 (May 1965).

# THE STRUCTURE AND CHARACTER OF USEFUL INFORMATION-PROCESSING SIMULATIONS

Louis Fein
*Synnoetic Systems*
*Palo Alto, California*

## INTRODUCTION

I am curious about how the brain and nervous system work. So I became interested in the properties of useful models—especially of useful information-processing models of psychological and physiological processes.

In my professional pursuits as a sometimes consultant, writer, lecturer, teacher, and designer in the computer field, I have attentively and respectfully listened, observed, read, and conversed about models that were inspired by the desire of the modelers themselves to know how the brain works. I doggedly sought for a *particular* bit of knowledge of how the brain works that was gained by a researcher primarily because he used a putative model of the brain—as opposed to knowledge about the brain gained by direct observation and measurement not predicted or suggested by a model. For years, I have been singularly unsuccessful in finding any neural net models or digital computer program models (i.e., the information-processing models I am familiar with) whose use led to a particular experimentally verifiable piece of knowledge of how the brain works. I have found lots of information about the models themselves; but not about the brain.

I am neither a biologist nor a psychologist. I am merely an interested observer who has paid diligent, respectful, yet disciplined attention to this matter. For a while, I blamed my own ignorance and lack of understanding of psychological and physiological processes and of the terminology used to describe them, for my inability to find what professed brain modelers were learning about the brain itself. But I began to doubt that it was I who was at fault, as I became startlingly aware of the various defensive postures taken by professed modelers of psychological and physiological processes whom I pressed in correspondence and in conversation to tell me what they now knew, that they didn't know before, about a particular psychological or physiological process, now that they had a working model; or even what *kind* of knowledge they expected to gain with the use of a model that was not yet developed. The responses were largely evasive, irrelevent, defensive, offensive, insulting to my person and my ancestry; I was accused of wanting to deprive honest researchers of their livelihoods; I was ignored, or referred to the *vast* literature on the subject; I was admonished to be tolerant and to give this young discipline a chance to develop. Some didn't want to discuss it at all; others sent me reprints. My Freudian analysis of such responses was orthodox; when a

simple question to a patient elicits a noisy and boisterous defense, then there is something wrong with the patient, not the questioner. My conclusion was that a great deal of psychological and physiological modeling was both aimless and fruitless. I started to wonder why neural net models and digital computer models were not as fruitful for gaining verifiable knowledge about psychological and physiological phenomena, especially of the brain and nervous system, as mathematical, computational, and physical models have been for learning about physical phenomena such as wave motion, aircraft behavior, and circuit operation. Indeed, it is rare to find an article or talk on psychological or physiological modeling without the author's observing that since models have helped the engineer and physicist to gain knowledge in his subject then models should be expected to help the author to gain knowledge in his.

So I wondered in what ways the fruitful models are different from the fruitless ones. I assumed that the reason modelers in psychology and physiology were having little luck was that their models didn't have whatever characteristics models should have to suit the modelers' purposes. I tentatively formulated the question: what are the characteristics of an information-processing model that would suit it to the purposes of a psychologist or physiologist interested in some aspect of the brain and nervous system. I reasoned that once I knew the character of useful models, I would find that unsuccessful modelers were using models that did not have certain of these required characteristics. I want, therefore, to present what I think the structure and character of the total modeling process must be in order for it to be potentially useful in psychology and physiology.

Let me give a summary of my viewpoint before systematically presenting it for your consideration.

Recall that we are focusing attention both on what physiological or psychological knowledge, if any, a model can lead to, and of what value such knowledge is to a particular practitioner—a surgeon, psychiatrist, public health specialist, internist. The *utility* of a model depends on how helpful it is in gaining new knowledge; the *value* of new knowledge obtained with the help of a model depends on how much more effective, efficient, and economical it makes a practitioner in doing his job. In this context, "$A$ is a model of $B$" would be an incomplete statement. One must at least say that $A$ is a potentially useful model of $B$, if the use of $A$ can help the simulator predict or suggest new knowledge in $B$ (expressed in a suitable form and language) that could be of some value to some practitioner. But even these three elements (1) the model $A$, (2) the modeled system of interest, $B$, and (3) the new knowledge hypothesized by the simulator about $B$, are not enough. Another element is necessary in the modeling process if one is to measure its utility, i.e., how helpful it is in gaining new knowledge. In order to test these hypotheses—which are predictions or suggestions of answers to particular questions of the simulator, or solutions of his problems, or resolutions of issues—the simulator must design and carry out valid and feasible experiments on the system of interest, $B$, with the use of appropriate instruments and apparatus together with procedures for interpreting the results of the experiment. Without such experiments the hypothesized new knowledge remains conjecture; the simulator cannot give the practitioner confirmed answers to his questions, problems, or issues; hence he can't say anything about the utility of the modeling process.

The crucial issue of whether or not an experiment can be both designed and carried out to test the hypotheses suggested by the model $A$ about the system of interest, $B$, usually depends on whether the controllable parameters and variables in the model $A$ have correspondences in what is modeled, $B$; whether the corresponding parameters and variables in $B$ are accessible to the investigator so he can control and vary their values; and whether the investigator has instruments suitable for observation and measurement of $B$.

Without being able to show correspondences between controllable parameters and variables in the model and in what is modeled, the model is not even potentially useful as an aid to gaining new knowledge; without instrumentation, even if one showed the correspondences, an experiment could perhaps be designed but not carried out to realize the potential utility of a model.

The experiments may be gedanken experiments, i.e., experiments that the simulator would design and carry out if he knew the correspondences and if he had the instruments. Thus, a potentially useful and valuable modeling process has four necessary elements: (1) the model, (2) the modeled system of interest $B$, (3) hypothesized new knowledge about $B$, (4) the experiment.

## THE STRUCTURE AND CHARACTER OF A SIMULATION

I have used the word "model" to introduce these notions because it is a familiar term. In other fields, practitioners use terms such as: mapping, analog, similitude, isomorph, homomorphism, and emulation, often as synonyms of model. But as with the term, model, it is often unclear what these terms mean. Because the word model is so ambiguous, its use will be limited in what follows. To avoid ambiguity and misunderstanding, I start with a statement of what I am not here talking about; then a statement of what I am interested in; then some definitions.

I am *not* here interested in the process whereby an investigator is motivated to design, construct, and use a system that behaves or is otherwise similar to another system—an investigator who, after acknowledging his debt to it for its inspiration, has no further interest in the other system. This species of "model" is not what concerns us. I emphasize this distinction because I find that it is often not made and, if made, misunderstood. If I ask a man who has announced that he is simulating, say, the cognitive process, what he has learned thereby about the cognitive process, and he responds with what to me is a non sequitur, by telling me what a wonderful new computer language he has invented or what clever things his program can do, then this is an instance of the kind of misunderstanding I am trying to avoid by making clear and careful definitions of important and distinguishable notions and objectives.

Nor am I here interested in models that serve as pedagogic vehicles for educating an investigator and presenting to him knowledge already in hand.

I am here interested in the *process* whereby an *investigator* designs, constructs, and uses a certain kind of *instrument* (heretofore variously called model, or mapping, or analog . . .) with the aid of which he forms hypotheses about knowledge of particular *objects, phenomena, properties, functions, events, or thoughts* of interest in his field, and who has designed feasible and valid experiments to test these hypotheses (verify or find them false).

The process must have all four interdependent constituents: (1) the investigator, who (or which) I will call the *simulator*; (2) the instrument which (or whom) I will call the *simulate*; (3) the object or phenomenon, or properties, or events or thoughts of interest to the simulator, which I will call the *simuland,* and (4) the design of the feasible experiments, or the gedanken experiment, which I will call the *experiment*. The process itself, I will call the *simulation* or the *simulation process*.

Note that as knowledge is accumulated in developing fields, such as physiology or psychology, or even information processing, the simuland is rarely a complex system such as the ear, or the brain, or a computer, or the eye. It is more often a part of the ear, or of the brain, or of the computer, or of the eye. One usually constructs a simulate in order to use it as an aid in gaining new information, insight, knowledge, or understanding, about a single property, or a single function, or a single part of a subsystem. Only occasionally, as is the case with "laws" of physics, will a single simulate serve to answer questions about many properties, many functions, a complex of many parts. Thus, we speak of a simulate, useful for finding or calculating or predicting such things as signal transmission modes in nerve fibers; or the logic design of the addressing system of the core memory in a computer. The simuland is here meant to be only that part of the simulator's system of interest about which he seeks knowledge with the aid of a particular simulate. I emphasize this point because I believe that speaking of "brain modeling," to pick only one instance, often confuses both the investigator and those interested in his work. If an investigator specifies an ambiguous objective for himself by saying that he is simulating the brain when he means he is simulating a property of a part of the brain, then his colleagues may misconstrue him, and they may expect results from him that the investigator never intended to obtain.

To make these ideas explicit, I ask the reader to imagine that we have the problem of writing a manual on a simulation process as herein defined, for the preparation of a college laboratory report; or a proposal to a government agency; or the final report to a government agency; or a paper for a professional journal; or a graduate dissertation. The organization and format of such a document will reflect what I consider to be the necessary structure and character of potentially useful and valuable simulation processes. It will at the same time systematically disclose to the reader the value, utility, and progress of the simulation process reported on, and the goals and intentions of the simulator. The following is an abstract of such a manual.

# THE PREPARATION OF A PAPER ON THE SIMULATION PROCESS

This manual is intended to assist authors in the preparation of reports, proposals, papers, articles, and dissertations on simulation.

It should be understood that the specific organization and format suggested herein are arbitrary. However, the literature on simulation is read by many people with different orientations and varying degrees of familiarity with the subject matter. For these people to get the most out of documents on simulation, they should expect the material in them to be presented in a consistent organization, style, and format. Not only does the reader benefit by the application of these rules, the author benefits too; they serve as a goad to logical, critical, explicit, and disciplined thinking and writing.

A paper on a simulation process should always mention the hypothesized knowledge that was or is to be tested; to what purpose the knowledge obtained is being put or might be put; a description of the simulate, a description of the simuland, and a report of the design of the experiments. The emphasis each topic receives will, of course, depend on the pattern of emphasis in the work itself. The question arises: in what order should these topics be covered in a paper, and what specific items should be covered under each topic. The following organization and format is suggested:

*Outline of a Report on the Simulation Process*

I. SIMULAND

A. Statement (in the language of the simulator's field) of what knowledge is already verified or to be assumed about the structure, functions, properties, and theorems of the system of interest to the simulator — the simuland.

B. Statement (in the accepted language of the simulator's field) of a specific question the simulator wants an answer to; or problem he wants solved; or an issue he wants resolved, with the aid of the simulate and the experiment. This is the hypotheses to be tested.

C. Acceptable form and language of the answer, solution, or resolution.

II. SIMULATOR

A. Statement of the purpose and use to which the new knowledge gained in the simulation process is or might be put — the effect, significance, and value of having such knowledge.

III. SIMULATE

A. Description in the language of the field of the simulate, of the structure, functions, properties, and theorems of the simulate.

B. Identification of corresponding controllable and measurable or observable variables and parameters in the simulate and the simuland.

C. The form of addressing the simulate with the simulator's questions, problems, or issues, in the language of the simulate.

D. The procedure for deriving from the simulate the answers, solutions, or resolutions.

E. The correspondence rules by which question/answer or problem/solution, or issue/resolution in appropriate form and language of the simulate are transformed into what the simulator wants, i.e., the corresponding question/answer or problem/solution, or issue/resolution in acceptable form and language of the simuland.

IV. EXPERIMENT

A. The design of valid and feasible experiments (which may be gedanken experiments) whose sole object it is to test the hypotheses of the simulate, i.e., to verify the hypothesized answers, solutions, or resolutions given by the simulate, or to prove them false.

The form of the write-up of these experiments is typical of the form of the write-up of laboratory experiments performed in school.

(1) Object of experiment: to test the hypothetical answers given in the simulate
(2) Procedures
(3) Apparatus and instruments
(4) Results
(5) Conclusions
  (a) Verified hypotheses
  (b) Unverified hypotheses
(6) Unexpected results — fall-out

(7) Suggestions for other
  (a) Experiments
  (b) Questions, problems, issues
  (c) Instruments
  (d) Theory

## THE UTILITY OF A SIMULATION

The utility of a simulation will clearly depend on the extent to which the simulator learned what he wanted to learn through the use of his simulate.

It may be noted that in any simulation, in order for the simulator to learn what he wants to learn, the simulate must bear a certain relation to its simuland. Suppose a simulation system consists of, say, a differential equation simulate from which the simulator can learn about properties of standing acoustic wave patterns in an enclosed cylinder (simuland) and that he can perform experiments in which he can observe or measure correspondences between properties of standing waves in his simulate and simuland. The utility of this simulation can be high. But if the investigator, actually interested in, say, properties of standing acoustic wave patterns in an enclosed cylinder (simuland), first hypothesizes a simulate of this simuland, and then builds a simulate of the hypothesized-simulate-of-this-simuland, then he might thereby gain knowledge about the hypothesized but untested simulate, but he cannot thereby learn anything about the simuland. For instance, to use a digital computer to solve a differential equation *presumed* to simulate wave motion gives information about the differential equation but not about wave motion. Or to write a digital computer program to simulate a hypothesized neural net model of a part of the eye will tell you something about the hypothesized neural network, not about the eye. (On the other hand, if the differential equation has already been well established as a simulate of wave motion, the digital computer can be said to give information about wave motion.)

We will call a simulate of a simuland (or of a well-established simulate-of-the-simuland) a first order simulate; and the simulation process that includes it, a first order simulation process. We will call a simulate of a hypothesized-simulate-of-a-simuland, a second order simulate; and the simulation process that includes it, a second order simulation process, and we will say that only a first order simulation process can have a nonzero utility.

We can now enumerate the major characteristics that a simulation process must have in order for it to have utility:

1. The simulate of the process must be a first order simulate of the simuland. To use the vernacular: you might learn something about the ear from a model of the ear; but you can't learn anything about the ear from a study of a model of a hypothesized-model-of-the-ear.

2. There must be one-to-one correspondences between the accessible, controllable, and measurable or observable variables and parameters of the simulate and simuland.

3. There must be a well-designed experiment for testing the hypotheses.

I mentioned earlier that I sought the characteristics of an information processing model that would suit it to the purposes of a physiologist and psychologist and that once I knew the character of the useful models, I would find that unsuccessful modelers were using models that did not have certain of these characteristics. I have run some gedanken experiments of recasting into the above framework, papers that purport to simulate—by digital computer programs or by neural networks—physiological and psychological processes such as memory, perception, neurosis, and cognition. In papers using both of these types of information-processing simulations, simulators do not show one-to-one correspondences between accessible controllable and measurable or observable parameters and variables in the digital computer program or network and the organism's brain, nervous system, or other organs of interest. Nor do they have the instrumentation either to gain access to or to aid in controlling, varying, measuring and observing these variables and parameters in the organisms themselves. Thus, without the correspondences and without experiments, such simulations are fruitless.

In other instances, digital computer solutions are reported for sets of equations representing a network that was inspired by an organic process, and presumed to be a simulant of it. Obviously such third order simulations have no potential utility for learning anything directly about that organic process.

Finally, I read some papers, presumed to be

about simulation, in which I could not locate a statement of what hypothesis was to be tested, nor a statement about what specific questions or problems the simulator hoped to answer, if his simulation was successful. These were aimless simulations.

I would suggest to laboratory instructors, dissertation advisers, sponsors of research who prepare requests for proposals and who evaluate contractor's or grantee's work, journal editors, teachers, and investigators interested in simulation as herein defined, that they would find it valuable in their professional roles to insist that authors prepare their written material in an organization and format similar to the one presented in this paper.

# THE CATALOG: A FLEXIBLE DATA STRUCTURE FOR MAGNETIC TAPE

Martin Kay and Theodore Ziehe
*The RAND Corporation*
*Santa Monica, California*

The files of data used in linguistic research differ from those found in other research applications in at least three important ways: (1) they are larger, (2) they have more structure, and (3) they have more different kinds of information. These are, of course, all simplifications but not gross ones. It is true that the files that must be maintained by a large insurance company or by the patent office are so large as to pose very special problems, but the uses to which the files are to be put are fairly well understood and their format and organization is not usually subject to drastic and unexpected change. It is also true that the data from a bubble chamber is interesting only if collected in vast quantities, but this is not the only respect in which a bubble chamber is a special kind of tool. A typical linguistic job will bring together a number of files, each very large by the standards of everyday computing: a body of text, a dictionary and a grammar for example. The grammar, if it is anything but a very simple one, will contain a large number of elementary items of information of different kinds, each related to others in a number of different ways. This is what it means to say that the file has a lot of structure. The dictionary may also contain grammatical codes which may consist of characters from one of the languages represented in the dictionary or may be something altogether different. If the dictionary contains alternatives to which probabilities are assigned, then these will presumably be in the form of floating-point numbers. This is what it is like for a file to contain different kinds of information.

The notion of a *catalog** was developed principally with the needs of linguistic computing in mind. It is oriented more to the storage of information on a long-term medium, such as magnetic tape, than to its representation in the high-speed store of a computer. The elementary items of information in a catalog are called *data*. The structure imposed on the data making up a catalog is that of a tree—a hierarchy of sets of information. Let us consider as an example how a bibliography or the acquisitions list of a library—catalogs in the conventional sense—might be organized within this system. Each document or book has an entry in the file containing various items of information about it. One of these can be chosen as the key under which the others are filed. For example, the acquisition number of an item can serve as the key for all information related to the item. Under it there will be sections for author, title, journal if relevant, publisher, and date. In an actual application, there would doubtless be

*The catalog system has been developed through the joint efforts of the Centre d'Etudes pour la Traduction Automatique of the Centre National de la Recherche Scientifique at the University of Grenoble, France, and the Linguistics Research Project of The RAND Corporation.

other sections but these will serve for the example. The author section must be capable of handling cases of multiple authorship without confusion and we may assume that there is provision for giving the institution to which each author belongs. The journal section will also have subsections for volume and issue number.

A pair of entries might appear as in the diagram at Fig. 1. The acquisition numbers of these two documents are 1746 and 1747 respectively. The first has one author belonging to a single institution; the second has two, each belonging to a pair of institutions. Furthermore, the two authors of the second paper belong to one institution in common, but since a catalog must, by definition, have the form of a tree and not of an arbitrary graph, the institution must be mentioned separately for each author. The first document is a journal article and there are therefore two nodes below the one representing the name of the journal; the first gives the volume and the second the number. The second entry is for a book and therefore these two nodes do not appear.

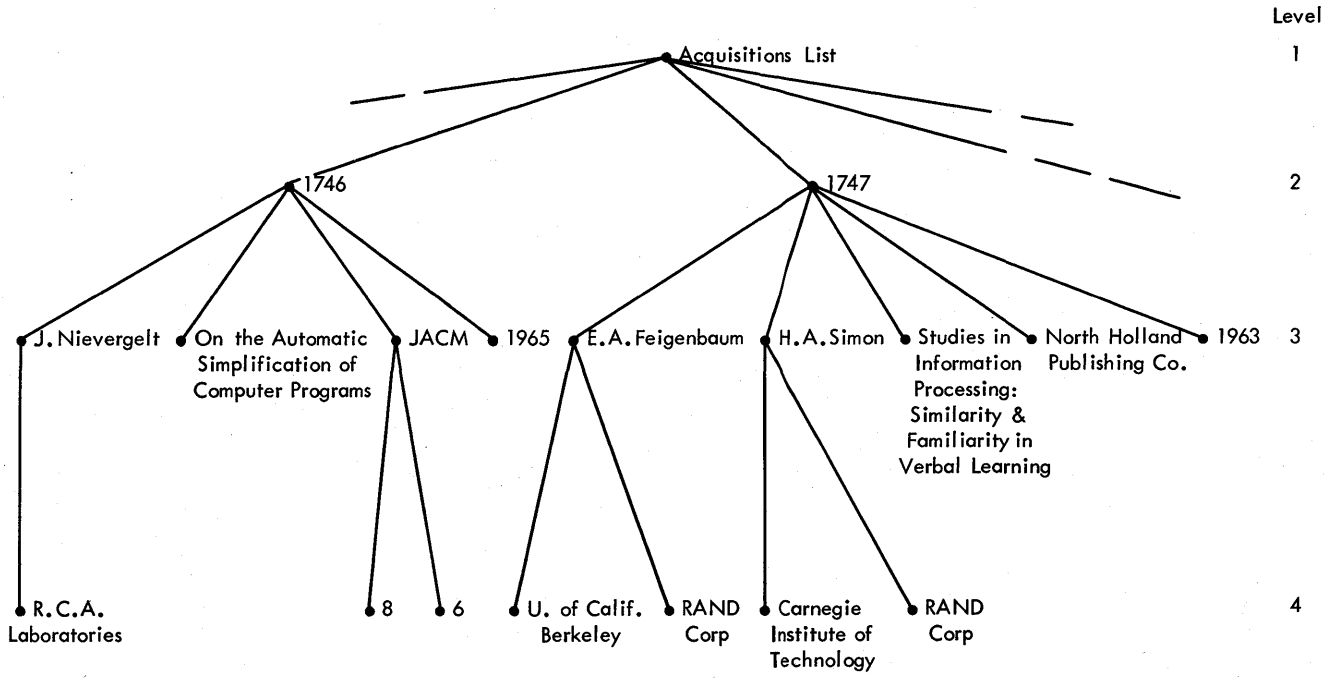



Figure 1. A portion of an acquisitions list.

In this example, the kind of information at each node can be unambiguously determined from the structure of the tree. The nodes on level 2 all represent acquisition numbers. The last three nodes under a given acquisition number represent title, publication information, and date respectively. Any nodes preceding these represent authors. Any nodes below those for authors are for institutions. A journal article is distinguishable by the volume and number nodes which are absent in the case of a book. However, it is not difficult to imagine cases where rules of this kind would not work. Suppose, for example, that the date of each edition of a book were put in, or that when the date was unknown, we wished simply to omit it. To cope with these situations, it would be necessary either to redesign the catalog structure or to label each datum explicitly to show the kind of information it contains. The structural redesign might be as follows. The names of authors are dropped to level 4 and their institutions to level 5. A new kind of datum is introduced on level 3 dominating the author names. This node has no information of its own; it serves only to show where the authors are to be found. A similar node could be inserted above the date, except that in this case the node would have information of its own to provide for the case where the date is missing.

The catalog system in fact requires that each datum be tagged with the name of the *data class* of the information it contains. This is useful for other

reasons than the one we have suggested. Each data class is, for example, associated with a particular set of encoding conventions. Some contain textual material, some floating point numbers, some integers and so on. So, by looking at the class of a datum, we can tell not only what its status is in the catalog as a whole but also how to decode it.

It is convenient to be able to describe the overall structure of a catalog, to the computer as well as to other people, in terms of data classes and the relations among them. This we do by means of a *map*, which, like the catalog itself, is a tree, but in which the data are replaced by the names of the data classes. The map of our hypothetical acquisitions list is shown in Fig. 2. The name of each data class appears exactly once in the map, a fact which is bound up with an important restriction on the way catalog structures may be designed. The members of a given data class always appear on the same level of the catalog—the level on which the name of the class appears in the map—and they always come immediately below members of the same other class —the one whose name appears above theirs in the map. If two classes are shown directly beneath the same other class in the map, then their members must appear in that order in the catalog itself. Thus, in the example, a title may never come to the left of the corresponding author. Cases can arise where these restrictions may seem unduly severe, but, as we shall shortly see, powerful means are available for dealing with them.

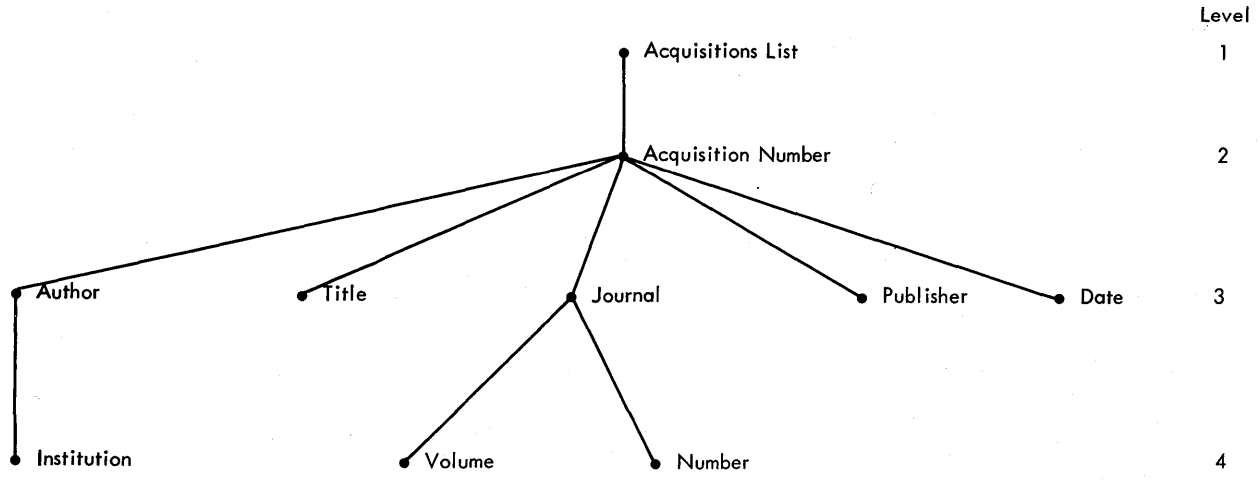



Figure 2. Map of an acquisitions list.

A variant has been proposed for the catalog design we are using as an example in which new nodes would be introduced to represent sets of authors and sets of dates. Since we have data-class names, we do not need to adopt this variant. However, classes of this kind, whose members never contain substantive information, are frequently useful. In other cases, no substantive information is available for a particular datum, but only for the nodes it dominates. In these cases, we speak of *null data*. The catalog system has been implemented in such a way that null data occupy no space whatever on the tape. They can therefore be used freely to lend perspicuity to the structure of a catalog and without regard to economy. In order to see how this is achieved, we must consider the format used for writing catalogs on tape.

A catalog is reduced to linear order in the most obvious way. The datum at each node is written on tape before the data at the nodes beneath, and all these before nodes to its right. Thus the node at the root of the tree goes first, followed by those on its leftmost branch. The first node on a branch is the first to be written, followed by the nodes on *its* leftmost branch. When a branch is finished, the one immediately to the right is taken next. This is the order arrived at by regarding the nodes as operators whose arguments are the nodes immediately beneath them, and writing the expression out in Polish parenthesis-free notation.

A set of programs is being written for moving catalog data between high-speed core and tape. A blocking scheme is used for the data on tape; the block size is set by the user. Each datum is treated

as a single logical record whose size is not restricted to that of the physical block. Each logical record is preceded and followed by a *link word* which gives the lengths of the records on either side of it. The length of the preceding as well as the following record is given in order to make it as easy as possible to backspace. Link words also contain certain other information of importance to the housekeeping of the reading and writing programs, but nothing of the essential structure of the catalog. This being the case, it is possible for a program using the catalog input/output system to make use only of its blocking facilities for certain purposes, calling on the whole system only when required. This plan of building the system as a sequence of layers, each using the facilities provided by the one beneath, has been followed wherever possible in the design. The first two words of each block are an *IOBS control word* and a *FORTRAN control word.* These are used nowhere in the current system but are included for reasons of compatibility.

The first word of a logical record which contains a catalog datum is a *datum control word.* This gives the data class of the datum and its *preceding implicit level* number (PIL). It is the PIL that enables the system to identify the place of null data which, as we have said, are not explicitly represented by a logical record. The PIL is defined as follows:

1. For all data on level 1, it is 0.
2. The PIL of the *first* datum dominated by a null datum is the PIL of that dominating null datum.
3. The PIL of every other datum is the level number of the datum that dominates it, i.e. one less than its own level number.

Informally, we can say that the PIL gives the highest point in the tree encountered on the path from the previous non-null datum to the current one. Consider two adjacent data on level $i$ of some catalog and suppose that the PIL of the second is $j$, where $j \leq i = 1$. $i-j-1$ null data are to be assumed between these two levels $j+1, \ldots, i-1$. Given the class of the current non-null datum, the classes of these null data are uniquely determinable from the map.

It is desirable to be able to write general programs for performing standard operations on catalogs without requiring that the user supply complete information on the structure of the catalog to be treated. For this reason, the map of a catalog is written on tape in the logical record immediately preceding the first datum. The map is represented as a simple list of data-class names paired with level numbers and taken in the order we have described for the catalog itself. Thus, a class whose level is given as $i$ is dominated by the class with level $i$ -1 most recently preceding it in the list. With the name of each data class is also given a code showing what encoding conventions are used for the data of that class.

We have noted that the restrictions imposed on the design of catalogs could become onerous in some situations if means were not introduced for overcoming them. We have been considering how a library acquisitions list might be represented as a catalog. But, of all the lists produced by a library, surely this is the least interesting. Suppose instead that we were to undertake to accommodate the subject catalog. Most subject classifications have the structure of a tree to begin with, so that the job should be easy. One possible strategy would be to examine this tree to determine the length of its longest branch, that is, how many categories dominate the most deeply nested one. We may then construct a map with this number of levels plus 5, which is the number used for the acquisitions list. This will make it possible to put a complete entry of the kind considered in the simpler example beneath the node for the most deeply nested category in the classification scheme. In general, the node for a subject heading will have two kinds of nodes directly beneath it, one kind for more particular categories under that heading and one for documents which cover the whole field named by the current heading. The structure already set up for the acquisitions list is repeated once for each document node in the map, but with different data-class names.

This scheme will indeed work, but it is clearly unsatisfactory in a number of ways. For one thing, subject headings will be in different data classes according to their level in the classification as a whole. For another, the map is unduly large and monotonous and liable to change when some minor part of the classification changes. An alternative strategy rests heavily on the claim made for catalogs that any kind of data whatsoever can be accommodated in a datum. If this is so, then an entire catalog can be included as a single datum within another one. From here, it is a short step to the notion of catalogs with recursive structures. Consider the ex-

ample in Fig. 3. The main catalog has two data classes of which the lower one has data that are other catalogs. To emphasize this, we have shown this node with a square rather than a circle. This simple two-class map is written at the beginning of the tape. When a subheading datum is encountered, the first thing it is found to contain is the map of a subsidiary catalog. In order that the data of this catalog should be correctly processed, the tape format must make special provision for them. In fact, subsidiary catalogs are represented not as single logical records, but as sequences of logical records bounded by special markers. However, the user of the system need not concern himself with these details. As far as he is concerned, the included and the including catalogs can be treated in exactly the same manner.

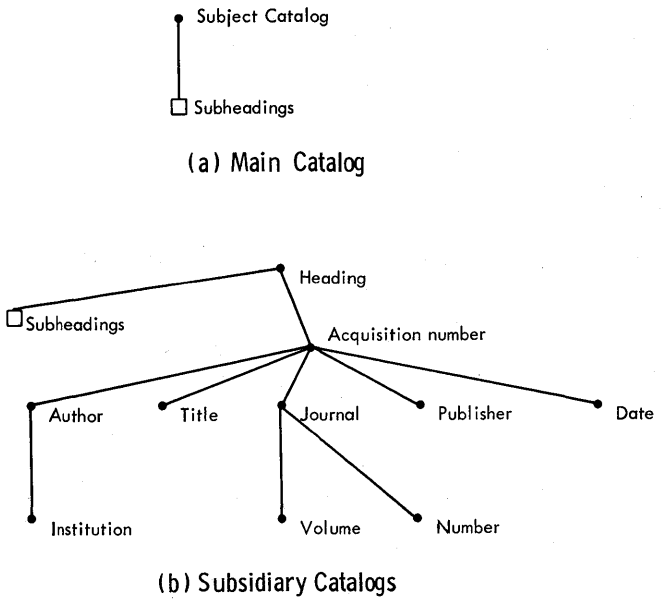



Figure 3. Structure of a library subject catalog.

The subsidiary catalog in Fig. 3 is similar to the main catalog for the acquisitions list except for the addition of a single new class to accommodate a further level of subheadings. This again is a class of catalogs and their structure is exactly the same as that of the subsidiary catalog of which they are members. Fig. 4 shows an excerpt from a catalog built on this plan.

Any scheme devised to fill the role for which catalogs were devised must be measured against three main requirements.

1. It must be easy to update.
2. It must provide for retrieval of information in response to a wide variety of requests.
3. It must allow files to be organized on new principles as research proceeds.

Now, the catalog system is not intended as a full-fledged information-retrieval system, but it does contain something of what any such system would have to provide. In particular, it provides powerful and flexible facilities for addressing data and sets of data. Furthermore, this addressing capability is precisely what is required for an updating algorithm where the principal work consists in identifying the items of information to be treated.

There is no obvious limit to the refinements that could be introduced into a catalog addressing scheme, and our ideas on the subject can be guaranteed to far outpace our ability to implement them. Here, we must content ourselves with a survey of some of the simpler notions.

It will be convenient to distinguish between the *location* of a datum and its *address*. Each datum in a catalog has a unique location which may be thought of as its serial number on the tape, or as anything else which preserves its uniqueness. But a datum may have an indefinite number of addresses, only some of which refer to it uniquely. The location of a datum will not normally be known to the user of a large catalog, but this is of no consequence provided that there is a clear method of

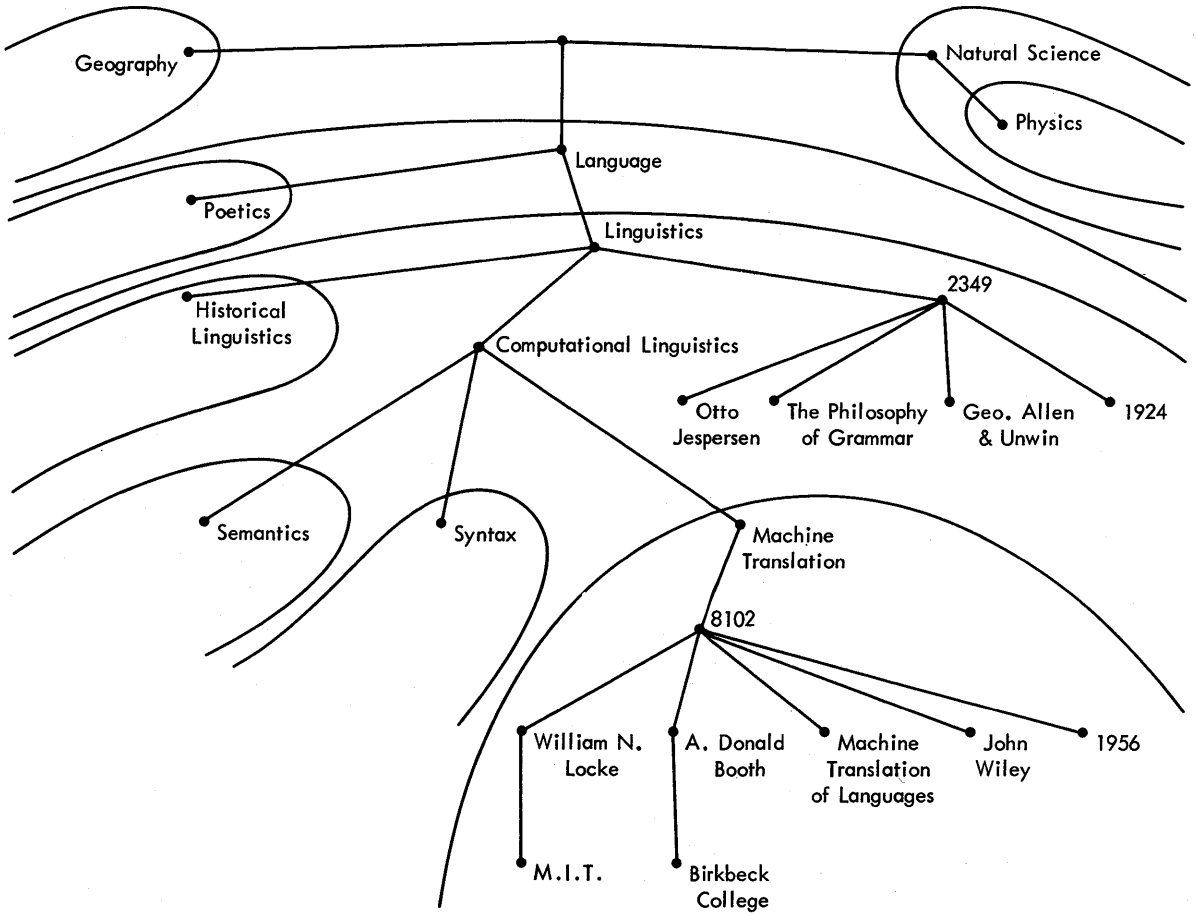



Figure 4. A portion of a library subject catalog.

specifying unique addresses. The notion of location is useful only for understanding how the addressing scheme works.

Some parameters which are useful in identifying a datum are its *data class,* its *class ordinal,* its *level,* its *level ordinal* and its *value.* Data class and level have already been explained. The *i*th datum of a given class dominated by a single datum on the next higher level has class ordinal *i.* The *i*th datum of any class dominated by a single datum on the next higher level has level ordinal *i.* We can now describe the form of an *elementary address.* This is a triple consisting either of a level, a level ordinal, and a value or a data class, a class ordinal, and a value. In either case, any member of the triple may be left unspecified. The following are some examples:

$(1, ,a)$ — The data on level 1 with value $a$.
$(, ,a)$ — All data with value $a$.
$(A)$ — All data of data class $A$.

Informally, we are using the convention that, if the first member of a triple is an integre, it is a level number; if it contains alphabetic characters, it is a data-class name.

An elementary address, like any other address, can be regarded as a function whose value, if defined, is a location or list of locations. Two other useful functions, descendant and ancestor, have location lists both as arguments and values. These are:

Des [L] — All data dominated by the datum or data at L.
Anc [L] — All data dominating the datum or data at L.

A concatenation of addresses is itself an address whose value is the intersection of the location lists referred to by each of them. This machinery is already sufficient to call for data in a number of interesting ways. The following examples refer to the

catalog, part of which is shown in Fig. 1 and whose map is given in Fig. 2.

Des [(2,,1747)] (3,2)
The datum or data on level 3 which are second sons of data on level 2 with value 1747 —the value in this case is H. A. Simon.

Des [(2,1747)] (Author)
Here, "1747" is a level ordinal rather than a value, but we may assume that there is just one entry for each acquisition number. This therefore refers to the authors of the 1747th document in the file—E. A. Feigenbaum and H. A. Simon.

Des [(2,1747)] (Author,2)
This is the same as the previous example except that it selects the second author—H. A. Simon.

Anc [(Institution,,R.C.A. Laboratories)] (Author.
All authors from R.C.A. Laboratories. As far as we know from Fig. 1, this means only J. Nievergelt.

Des[Anc[(Journal,,JACM)] (2)] (Author)
Everyone who has published in JACM.

Anc[Date,,1964)] (Acquisition number)
Acquisition numbers of everything published in 1964.

It is clear that other functions could be added to these two without difficulty.

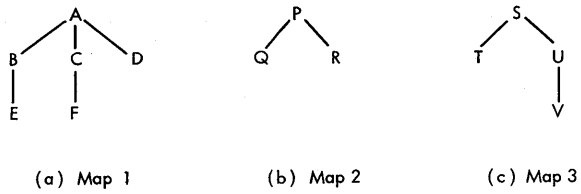
Addressing catalogs, some of whose data contain catalogs poses special problems requiring some new machinary for their solution. At least three new functions, member, recursion, and catalog, are required:

Mem[L,M]—where L is an address or location list and M is an address. The value is defined only if some of the data referred to by L contain catalogs. To these internal catalogs, the address M is applied to yield the final value of the function.

Rec [L,M]—where L is an address or location list and M is an address. This permits data to be located in general recursive catalogs. Its value is a list of locations arrived at by (1) applying M to the top level catalog and (2) applying the whole function to the catalogs identified by L.

Cat [L]—where L is an address. The address L is applied within all catalogs contained in data of the current catalog. The location in the current catalog of any catalog in which a datum is found meeting that address becomes a member of the list which is the value of the function.

It will be easiest to see how these work by reference to a specially constructed example. Fig. 5 shows three maps which are used in the structure of some recursive catalog. Map 1 gives the structure of the main catalog. In this, data of classes B and C contain catalogs with maps 1 and 2 respectively. The catalogs with map 2 have one class containing catalogs, namely Q; class U, of catalogs with map 3, also contains catalogs.




(a) Map 1 (b) Map 2 (c) Map 3

Classes of data containing catalogs

B - Map 1
C - Map 2
Q - Map 3
U - Map 2

Figure 5. A recursive catalog structure.

Mem [(B),(D,,x)]
Data of class D and value x in catalogs in data of class B in the main catalog.

Rec [(B),(D,,x)]
Data of class D and value x anywhere in the recursive system.

Cat [(Q,,x)] Des [(P,,y)]
Data in the main catalog, necessarily of class C, containing a catalog in which there is at least one datum of class P with value y dominating at least one datum of class Q with value x.

Mem [(C),Rec [Mem [(Q),(U)],(R)]]
Data of class R anywhere in the recursive system.

Des [Cat [(D,,y)]] (E,,x)
Members of class E with value x dominated by a datum (which, in this case, can only be of class B) which contains a catalog which has a datum of class D and value y.

Rec [(B),Des [Cat [(D,,y)]](E,,y)]
Same as above, but in the whole recursive system.

Let us return briefly to the example of library catalogs. Suppose that, given an acquisitions list of the kind we have described, we wish to obtain an author index. With a conventional file on, say, punched cards, this would be a simple matter of sorting. With a catalog, two operations are involved. First, a new map must be designed for the index in which "author" occupies the level-one position, and the file must be rewritten with the new structure. Second, it must be sorted, duplicate author entries removed and their dependent nodes thrown together under a single author node. This is one of many processes known collectively as *transformations*. In general, a transformation is any process which produces a catalog with a certain structure from one or more other catalogs with different structures. The variety of transformations which could be applied to a sufficiently complicated catalog is almost endless and the search for algorithms capable of carrying out any one, provided only that it is specified in a terse but perspicuous notation, leads to theoretical and practical problems which would, and probably will, fill several more papers.

There is good reason to suppose that any transformation can be broken down into a series of elementary transformations of which there will be only three or four types. One of these will have the function of decomposing a catalog into *simple catalogs*. A simple catalog is one which has exactly one data class on each level. If two simple catalogs have the same classes on the first $k$ levels, then they can be merged to form a new catalog with one data class on each of the first $k$ levels, and two on the levels below. In the same way, it is clearly possible to merge a pair of catalogs of whatever structures provided only that they have a common class on level one. If they have common classes on levels 2, 3, etc., then the blend will be the more complete. Of course, before the merge can take place, it may be necessary to perform a sort. Now, any catalog in which there are $n$ data classes which have no subordinate classes can be regarded as a compound of $n$ simple catalogs. Many transformations can be effected by decomposing the given catalog either partially or completely, changing the relative levels of the data in some of the simple catalogs, and merging them together again in the same or a different order.

Sorting and merging are common components of transformations as well as other catalog procedures and the overall system must clearly give them an important place. Since it is part of the essence of catalogs that they contain a rich variety of data types encoded according to diverse conventions, more flexibility is required than most sorting procedures provide. In particular, the encoding of the information in a given data class will, in general, not be such that an algebraic sort on the resulting binary number will give the required results. Furthermore, there may be some classes in any catalog in which the order of the data cannot be algorithmically determined; their order may be essentially arbitrary, each under its own dominating element on the level above, or the requirement for the sort may be that the order of the data in the class be preserved from the original input. To provide for contingencies of these kinds, the catalog merge and sort routines allow the user to supply a comparison routine for each data class in any catalog to be treated. This routine takes a pair of data of the specified class and declares which of them should precede the other in the output. The routine also knows what the input order was and may use this in arriving at a result.

Since the objects to be merged and sorted are trees rather than files of independent records, the algorithms must clearly be unconventional in other ways as well. However, the differences are less than

might at first appear. If each node in a catalog were duplicated once for each lowest-level datum it dominated, the catalog would take on the aspect of a great number of chains with a lowest-level datum at the foot of each. Each of these chains could then be treated as a single record to be sorted in a conventional way. Whilst it is never necessary to actually expand a catalog into this cumbersome form, the computer can arrange to retain a datum in memory until all its decendants are passed so that its instantaneous view of the catalog at any moment is as of a chain.

The catalog system could develop in many different ways and it is our intention that it should. For something so pedestrian as a filing system, it is remarkable how it has captivated the stargazer and the theoretician as well as the bookkeeper and the librarian in everyone who has worked on it. And these have been many. However, it is important for the welfare of computational linguistics that catalog systems or something designed to fill the same need should be made available soon. We have therefore resolved to be done with theorizing, for the present at least. What catalogs need is action.

# INFORMATION SEARCH OPTIMIZATION AND INTERACTIVE RETRIEVAL TECHNIQUES*

J. J. Rocchio and G. Salton
*Computation Laboratory*
*Harvard University*
*Cambridge, Massachusetts*

## INTRODUCTION

Automatic information retrieval systems must be designed to serve a multiplicity of users, each of whom may have different needs and may consequently require different kinds of service. Under these circumstances, it appears reasonable that the system should reflect this diversity of requirements by providing a role for the user in determining the search strategy. This is particularly important in automatic systems, where presently used one-shot (keyword) search procedures normally produce poor results.

In an automatic retrieval environment in which the user may be given access to the system—for example, by means of special input/output consoles—this can be achieved by two principal methods:

1. By providing automatic aids to the user in his attempt to formulate effective search requests.
2. By using the results of previous searches to determine strategies likely to prove effective during a subsequent pass.

*This study was supported in part by the National Science Foundation under research grant GN-360.

In either case, the user can be made to control the retrieval process by asking him to furnish to the system information which subsequently determines, at least in part, the mode of operation for a later search.

Several methods may be employed to aide the user in formulating effective search requests. One of the simplest methods consists in providing some kind of automated dictionary which may be used to display certain pertinent parts of the stored information. Thus, the frequency of use in the collection of certain terms in the vocabulary can be displayed to allow the user to make a choice between the use of frequently occurring terms, if "broad" retrieval is desired, and that of rarer terms if "narrow" retrieval is wanted. Alternatively, terms related in various ways to those originally included in a search request may be exhibited, and the user may be asked to choose from among these related terms in reformulating his request. The automated dictionary is then used as an aid in a *manual* reformulation of the request.

The iterative search process can also be mechanized more completely by leaving the search request largely unchanged, but altering instead the information analysis process. In that case, the user furnishes to the system information concerning the

adquacy of a preceding search operation, which is then used automatically to adjust the retrieval process for the next iteration.

In the present study, several alternative search optimization procedures are examined. In each case, the automatic SMART document retrieval process, presently operating on the IBM 7094 computer in a batch-processing mode, is used to simulate the real-time iterative search process.[1,2,3] The automatic evaluation procedures incorporated into SMART are utilized to measure the effectiveness of each process, and data are obtained which reflect the relative improvement of the iterative, user-controlled, process over and above the usual single-pass search procedure.

## THE AUTOMATIC DICTIONARY PROCESS

In a conventional, batch-processing retrieval environment, the user normally relies on his intuition and experience—possibly aided by published references—in formulating an initial search request. Once the general context has been established, the request must be normalized to a form suitable for use by the retrieval system. In a conventional coordinate indexing system, for example, this normalization would consist in a manual transformation of the original search request into an appropriate set of keywords. In certain automatic keyword search systems, a machine indexing process would generate the keywords, and stored synonym dictionaries might be used for normalization. After the analysis process, the normalized identifiers which specify the search request are matched with the identifiers attached to the documents, and correlation coefficients are obtained to measure the similarity between documents and search requests.

In the present section, a system is considered in which a communications link enables the user to influence the normalization process by making it possible for him to choose certain terms to be added and/or deleted from an original search formulation. Four main procedures appear to be of interest for this purpose:

1. A stored synonym dictionary, or thesaurus, may be used, given a set of thesaurus entries, to display all related entries appearing under the same concept category.
2. A hierarchical arrangement of terms or concept classes may be available which, given a set of initial terms, can provide more general concepts by going "up" in the hierarchy, or more specific ones by going "down."[1,2,3]
3. A statistical term-term association matrix may be computed which can be used, given a set of terms, to find all those related terms which exhibit a tendency to co-occur in many documents of the collection with the terms originally specified.[4]
4. Assuming the availability of a set of documents retrieved by an initial search operation, one may add to the terms originally specified in a search request, all those terms which occur in several of the retrieved documents but do not occur in the initial request.[5]

While it is potentially very useful to provide the user with a set of terms which may have been overlooked in formulating the original search request, it is probably even more important to furnish an indication of the usefulness in the retrieval process of each of the query terms. The most obvious indicator of potential usefulness is the density (or absolute number) of documents identified by each of the given index terms. The assumption to be made in this connection is that the usefulness of a term varies inversely with the frequency with which it is assigned to the documents of a collection.

Thus, in a coordinate indexing system, in which the retrieval process is controlled by the number of matches between terms assigned to documents and terms assigned to the search requests, the indexing density provides a straightforward estimate of the number of documents likely to be retrieved in each particular operation. If a correlation function is used to compare keyword sets attached to documents and queries, the relation between number of retrieved documents and the indexing density of query keywords is less obvious. However, the general assumption that a query term with high indexing density will produce "broad" retrieval, whereas one with low indexing density produces "narrow" retrieval is still valid.

It seems reasonable under the circumstances, to require that each dictionary display provided to the user consist not only of the corresponding terms or concepts, but also of the frequency with which the

various terms are assigned to the documents of the collection. The user can then utilize this information to refine the search request by promoting terms deemed important and demoting others which may be ambiguous or otherwise useless in the retrieval process.

As an example of the use of the indexing density of query terms, consider the retrieval process illustrated in Fig. 1. The original text of a request titled "Morse Code" is shown in Fig. 1a. When this text is looked-up in a typical synonym dictionary,* eight distinct concept codes are obtained. The codes, together with the frequency of occurrence in the collection of the corresponding concept classes are shown in Fig. 1b; the full thesaurus entries are similarly included in Fig. 1c.

"Can hand-sent Morse code be transcribed automatically into English? What programs exist to read Morse code?"

| Term Used in Request | Concept Number | Frequency of Concept (405 documents) |
|---|---|---|
| hand-sent | 113 | 12 |
| Morse | 35 | 9 (LOW) |
| code | 281 | 37 |
| transcribed | 570 | 25 |
| automatically | 119 | 70 (HIGH) |
| English | 35 | 9 |
| programs | 608 | 104 (HIGH) |
| exist | 234 | 55 |
| read | 569 | 25 |

Figure 1. Processing of request "Morse Code."
a) Original query for "Morse Code."
b) Terms included in original request.

The user who examines the output of Fig. 1b may notice that concepts 119 (obtained from "automatically") and 608 (from "programs") appear with excessively high frequency in the document collection under study; furthermore, these concepts do not appear to be essential to express the intent of the query of Fig. 1a. These concepts might then usefully be removed from the query statement. The user may note further, from the display of Fig. 1c, that "transcribe" can be replaced by "translate" (concept 570), and "read" by "recognize" (concept 569) in order to render more appropriately the purpose of the request. Finally, the crucial concept 35 (Morse) may be reinforced by increasing its weight. The two reformulations of Fig. 1d reflect the corresponding additions, deletions, and substitutions.

The success of the request alterations may be evaluated by examining the ranks of the two relevant documents (numbers 305 and 394) as shown in Fig. 1e. It may be seen that retrieval results are improved for both modifications 1 and 2 over the original, but that the better result is obtained for the first modification where the relevant documents are ranked fourth and eighth, respectively. The correlation coefficients between the two relevant documents and the search requests are also seen to be

*The dictionary used in the example is the "Harris III" thesaurus available with the SMART retrieval system.[1,2,3]

| Concept Number | Corresponding Thesaurus Entries |
|---|---|
| 113 | hand-drawn, hand-keyed, hand-sent, hand, handsent, manual, non-automatic |
| 35 | English, French, Morse, Roman, Russian |
| 281 | code, pseudo-code |
| 570 | record, reproduce, translate, transcribe, transcript |
| 119 | artificial, automate, machine-made, mechan, semi-automatic |
| 608 | program |
| 234 | behavior, case, chance, event, exist, fall, instance, occur |
| 569 | read, recognize, sense |

Modification 1: "Can hand-sent Morse code be translated into English? Recognition of manual Morse code."

---

Modification 2: Use original query and add "Morse, Morse, Morse".

| Type of Query | Ranks of Relevant Documents | Document Number | Correlation |
|---|---|---|---|
| Original Query "Morse Code" | 7<br>30 | 394<br>305 | 0.29<br>0.13 |
| Modification 1 | 4<br>8 | 394<br>305 | 0.33<br>0.26 |
| Modification 2 | 4<br>16 | 394<br>305 | 0.30<br>0.13 |

c) Thesaurus entries for terms in original request.
d) Modified queries by deletion of common, high-frequency concepts and addition of important low-frequency concepts.
e) Comparison of search results using original and modified queries.

much higher for the modified formulations than for the original.

A second example, and a different dictionary feedback process is illustrated in Fig. 2 for the request labeled "IR Indexing." In this case it is assumed that an initial retrieval operation has taken place, and that the user would like to use information obtained from the retrieved documents in order to reformulate his search request before a second attempt is made. The original query text is shown in Fig. 2a, and the corresponding concept classes and concept frequencies appear in Fig. 2b. The retrieval results obtained by processing the initial query are given in Fig. 2c.

Under the assumption that the user examines the list of retrieved documents, and finds that the 5th and 6th documents (numbers 79 and 80) are useful to him, it is now possible to request that concepts attached to these documents, but *not* included in the original search request, be displayed. This is done in Fig. 1d for concepts jointly included in the relevant documents no. 79 and 80.

It now becomes possible for the user to pick new terms from the list of Fig. 2d—for example, terms like "coordinate," "lookup," and "abstract"—and to use them to rephrase the search request as shown

"Automatic Information Retrieval and Machine Indexing"

Figure 2. Processing of request "IR indexing."
a) Original query text for "IR indexing."

| Original Term Used in Request | Concept Number | Frequency of Concept (405 documents) |
|---|---|---|
| automatic | 119 | 70 |
| information | 350 | 45 |
| retrieval | 26 | 6 |
| machine | 600 | 77 |
| indexing | 101 | 11 |

b) Terms included in original request.

| Document Rank | Document Number | Correlation Coefficient | Relevant |
|---|---|---|---|
| 1 | 167 | 0.41 | no |
| 2 | 166 | 0.38 | no |
| 3 | 129 | 0.33 | no |
| 4 | 314 | 0.33 | no |
| 5 | 79 | 0.33 | yes |
| 6 | 80 | 0.30 | yes |

c) Retrieval results for original query (using version III of Harris Theraurus).

in Fig. 2e. The reformulated request also excludes concepts 119 (automatic) and 600 (machine) which are found to occur with excessive frequency in the document collection. The ranks of the rele-

| Concepts from Documents 79 and 80 | Corresponding Thesaurus Entries |
|---|---|
| 49 | co-ordinate, [coordinate], intercept, ordinate, pole, rectangular-to-polar |
| 108 | consult, look-up, look, [lookup], scan, seek, search |
| 114 | [abstract], article, auto-abstracting, bibliography, catalog, copy, etc. |
| 170 | noun, verb, sentence |
| 497 | science |

"Information Retrieval. Document Retrieval. Coordinate Indexing. Dictionary Look-up for Language Processing. Indexing and Abstracting of Texts."

| Retrieval Results Using Original Query | | | Retrieval Results Using Modified Query | | |
|---|---|---|---|---|---|
| Ranks of Relevant Documents | Document Number | Correlation | Ranks of Relevant Documents | Document Number | Correlation |
| 5 | 79 | 0.33 | 1 | 80 | 0.51 |
| 6 | 80 | 0.30 | 4 | 79 | 0.41 |
| 9 | 221 | 0.29 | 6 | 48 | 0.36 |
| 11 | 126 | 0.28 | 9 | 126 | 0.23 |
| 12 | 48 | 0.27 | 11 | 221 | 0.23 |
| 69 | 3 | 0.10 | 18 | 3 | 0.19 |

d) Concepts common to relevant documents Nos. 79 and 80 and not included in original request.
e) Modified query using terms from relevant documents.
f) Comparison of search results using original and modified queries.

vant documents obtained for both original and modified queries are given in Fig. 2f. It may be seen that the relevant documents have much lower rank, and correspondingly higher correlation coefficients for the modified search request than for the original. The lowest relevant document, in fact, places only 18th out of a total of 405 documents when the modified query is used, whereas it originally ranks 69th.

Corresponding improvements can be obtained by the judicious use and display of hierarchical subject arrangements and statistical term associations.

## REQUEST OPTIMIZATION USING RELEVANCE FEEDBACK

The vocabulary feedback process illustrated in the preceding section appears to be both easy to implement and effective in improving search results. It does, however, put considerable demands upon the user who controls not only what is displayed by the system, but also what is returned in the way of modified information. A variety of search optimization methods should, therefore, be considered which place a much larger burden on the system and a correspondingly smaller one on the user. One such procedure is the *relevance feedback* method.

In essence, the process consists in effecting an initial search, and in presenting to the user a certain amount of retrieved information. The user then examines some of the retrieved documents and identifies each as being either relevant (R) or not relevant (N) to his purpose. These relevance judgments are then returned to the system, and are 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.

The amount of improvement to be obtained from the feedback process depends critically on the manner in which the search request is altered as a function of the user's relevance judgment. The following process which has been used experimentally with the SMART system appears to be optimal in this connection. Consider a retrieval system in which the matching function between queries and documents (or between query and document identifiers) induces a metric, or a monotonic function of a metric, on the space of query and document images (e.g., on the space of keyword vectors).[6] In such a case, it is possible to produce an ordering of the documents with respect to the input query in such a way that increasing distance between document and query images reflects increasing dissimilarity between them.

Let $D_R$ be the nonempty subset of relevant documents from the source collection $D$, relevance being defined subjectively and outside the context of the system. An optimal query can now be defined as that query which maximizes the difference between average distances from the query to the relevant document set, and from the query to the nonrelevant set. In other words, the optimal query is the one which provides the maximum discrimination of the subset $D_R$ from the rest of the collection $(D-D_R)$. More formally, let $\delta(q, d)$ be the distance function used in the matching process between query $q$ and document $d$. The optimal query,, $q_0$ may then be defined as that query which maximizes the function

$$C = \sum_{d \epsilon D_R} \tilde{\delta}(q, d) - \sum_{d \epsilon D_R} \tilde{\delta}(q, d) \tag{1}$$

where $\tilde{\delta}$ is the average distance function, and decreasing distance implies stronger query-document association.

Clearly, Eq. (1) is of no practical use, even under the assumption that the optimal query $q_0$ can be determined as a function of $D$ and $D_R$, since knowledge of the set $D_R$ (the relevant document subset) obviates the need for retrieval. However, if instead of producing the optimal query $q_0$, the relation (1) is used to produce a sequence of approximations to $q_0$, starting with some initial query which identifies a part of the set $D_R$, then a method for automatically generating useful query modifications becomes available. The system can, in fact, produce the optimal query to differentiate the *partial* set of relevant documents, identified by the user, from the remaining documents; using this optimal query to modify the original search request the resultant query can then be resubmitted, and the process may be iterated, as more complete sets of relevant documents become available through subsequent retrieval operations. One may hope that only a few iterations will suffice for the average user; in any case, the rate of convergence will be reflected in the stability of the retrieved set.

In the SMART automatic document retrieval system, the query-document matching function normally used is the cosine correlation of the query vector with the set of document vectors, defined as

$$\rho(\bar{q}, \bar{d}) = \frac{\bar{q} \cdot \bar{d}}{|\bar{q}|\,|\bar{d}|} = \cos\theta_{\bar{q},\bar{d}} \tag{2}$$

where $\bar{q}$ and $\bar{d}$ are the vector images of query $q$ and document $d$, respectively. Since the vector images are limited to nonnegative components, the range for the correlation is $0 \leq \rho \leq 1$, corresponding to an angular separation of $90 \leq \theta \leq 0$. Under these conditions, the correlation coefficient is a monotonic function of the angular distance metric. Furthermore, since the correlation decreases with increasing distance, relation (1) may be rewritten as

$$C = \sum_{\bar{d} \epsilon D_R} \tilde{\rho}(\bar{q}, \bar{d}) - \sum_{\bar{d} \epsilon D_R} \tilde{\rho}(\bar{q}, \bar{d}) \tag{3}$$

where $\rho$ is the average cosine function $\rho$, It can be shown,[7] that in this case $C$ is maximized for

$$\bar{q}_0 = \frac{1}{n_0} \sum_{d_i \epsilon D_R} \frac{\bar{d}_i}{|\bar{d}_i|} - \frac{1}{N-n_0} \sum_{d_i \epsilon D_R} \frac{\bar{d}_i}{|\bar{d}_i|} \tag{4}$$

where $n_0 = n(D_R)$, the number of elements in the set $D_R$, and $N = n(D)$, the number of elements in the collection.

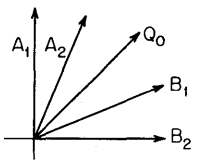
The query modification algorithm employed may now be written in the form

$$\bar{q}_{i+1} = n_1 n_2 \bar{q}_i + n_2 \sum_{i=1}^{n_1} \frac{\bar{r}_i}{|\bar{r}_i|} - n_1 \sum_{i=1}^{n_2} \frac{\bar{s}_i}{|\bar{s}_i|} \tag{5}$$

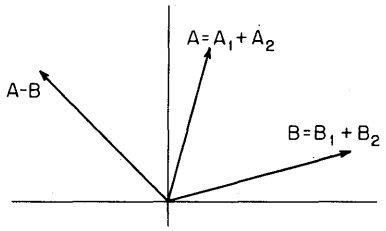
where $q_i$ is the ith query of a sequence, and $R = \{r_1, r_2, \ldots, r_{n1}\}$ is the set of relevant document vectors retrieved in response to query $q_i$, and $S = \{s_1, s_2,$

$...,s_{n2}\}$ is the set of nonrelevant vectors retrieved in response to $q_i$ (the specification of sets $R$ and $S$ constitute the feedback from the user after the $i$th iteration of the process).

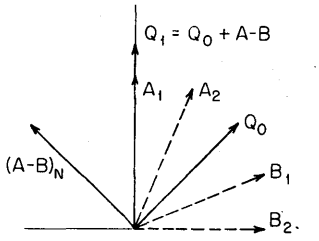
A simple graphical illustration of the application of Eq. (5) for two-dimensional document and query vectors is given in Fig. 3. Figure 3a shows the original query vector ($Q_0$), and the four document vectors $A_1$ and $A_2$ (relevant), and $B_1$ and $B_2$ (nonrelevant). Figure 3b illustrates the formation of the optimal vector used to differentiate between $A_1$ and $A_2$ on the one hand, and $B_1$ and $B_2$ on the other. The resulting, normalized query ($Q_1$) is shown in Fig. 3c. It may be noticed that whereas the original query is approximately equidistant from sets $R$ and $S$, the modified query is much closer to $R$ (indeed, it coincides with $A_1$) than to $S$. This is reflected in the correlation coefficients for both original and modified queries $Q_0$ and $Q_1$ with the four document vectors, as shown in Fig. 3d. It is evident that the modified query will be much more effective in providing retrieval of the relevant document set than the original.




a) Initial Query ($Q_0$), Relevant Docs. ($A_1$,$A_2$) and Nonrelevant Docs. ($B_1$,$B_2$)




b) Sum of Relevant - Sum of Nonrelevant Doc. Vectors




c) Resultant of $Q_0$ + Normalized Sum of Relevant - Sum of Nonrelevant

| Doc. | $Q_0$ | $Q_1$ |
|---|---|---|
| $A_1$ | 0.71 | 1.00 |
| $A_2$ | 0.92 | 0.92 |
| $B_1$ | 0.92 | 0.38 |
| $B_2$ | 0.71 | 0.0 |

d) Correlation of Query Vectors $Q_0$ and $Q_1$ With Doc. Vectors

Figure 3. Geometrical representation of relevance feedback.
a) Initial query ($Q_0$), relevant docs. ($A_1$, $A_2$) and nonrelevant docs. ($B_1$, $B_2$).
b) Sum of relevant — sum of nonrelevant doc. vectors.
c) Resultant of $Q_0$ + normalized sum of relevant — sum of nonrelevant.
d) Correlation of query vectors $Q_0$ and $Q_1$ with doc. vectors.

The query modification process of Fig. 3 was tested by performing two iterations for a set of 24 search requests, previously used in connection with the SMART system. Figure 4 shows the results for a request on "Pattern Recognition." The original retrieval results, using version 2 of the "Harris" thesaurus (synonym dictionary), are given in Fig. 4a. The user identifies documents 351, 353, and 350 as relevant, and 208, 225, and 335 as nonrelevant. The query is then automatically modified, in accordance with the expression of Eq. (5), and retrieval performance is compared in Fig. 4b. It may be seen that drastic improvements are obtained both in the ranks of the revelant documents and in the magnitude of the correlation coefficients. The "recall" and "precision" measures, shown in Fig. 4b, are the normalized evaluation measures incorporated into the SMART system,[8,9] which express the ability of the system to retrieve relevant material and to reject irrelevant matter.

| Document Rank | Document Number | Correlation | User Feedback |
|---|---|---|---|
| 1 | 351 | .65 | Relevant |
| 2 | 353 | .42 | Relevant |
| 3 | 350 | .41 | Relevant |
| 4 | 163 | .36 | - |
| 5 | 82 | .35 | - |
| 6 | 1 | .32 | - |
| 7 | 208 | .27 | Not Relevant |
| 8 | 225 | .25 | Not Relevant |
| 9 | 54 | .24 | - |
| 10 | 335 | .21 | Not Relevant |

| Retrieval Results Using Original Query | | | Results Using Query Modified by User Feedback | | |
|---|---|---|---|---|---|
| Ranks of Relevant Documents | Document Number | Correlation | Ranks of Relevant Documents | Document Number | Correlation |
| 1 | 351 | .65 | 1 | 351 | .66 |
| 2 | 353 | .42 | 2 | 350 | .60 |
| 3 | 350 | .41 | 3 | 353 | .55 |
| 4 | 163 | .36 | 5 | 163 | .37 |
| 6 | 1 | .32 | 6 | 1 | .32 |
| 9 | 54 | .24 | 7 | 54 | .29 |
| 26 | 205 | .17 | 11 | 314 | .23 |
| 27 | 224 | .17 | 16 | 205 | .19 |
| 33 | 314 | .16 | 17 | 39 | .19 |
| 34 | 39 | .12 | 30 | 224 | .16 |
| Recall .972 Precision .864 | | | Recall .989 Precision .923 | | |

Figure 4. Query processing using relevance feedback.
a) Retrieval results using original query for "pattern recognition" (version 2 of Harris thesaurus).
b) Comparison of search results using original and modified queries,

Figure 5 is a typical recall-precision plot giving recall and precision figures averaged over 24 research requests for the original query formulations as well as for two iterations using relevance feedback.* It may be noted again that for a given recall value large improvements are gained in the average precision by using the relevance feedback process. Additional improvements are obtained by identifying further documents as either relevant or nonrelevant during a second iteration.

## AUTOMATIC MODIFICATION OF THE ANALYSIS PROCESS

The last search optimization process to be described depends, like its predecessor, on feedback provided by the user, and results in selective changes in the document and request analysis process. However, instead of furnishing relevance judgments based on the output of a previous retrieval operation, the user makes a qualitative assessment of the effectiveness of an initial search operation. For example, he may find that the documents obtained from the system show that his request was interpreted too narrowly (since all retrieved documents belong to some small subfield of the larger area which he expected to cover), or too broadly, or too literally, or too freely.

Depending on the type of interpretation furnished by the user, the system now proceeds to initiate a new search operation under altered analysis procedures. If the user's verdict was "too narrow," a hierarchical subject arrangement similar to the one mentioned in the second section of this paper might be consulted, and each original query term could be replaced by a broader one; if, on the other hand, the initial search was "too broad," more specific terms might be obtained from the hierarchy. If the interpretation was too literal, the use of a synonym dictionary might provide more reasonable results; and so on.

Automatic retrieval systems are particularly attractive in such a situation, because these systems make it possible to provide at relatively little extra cost, a variety of indexing procedures which may be called upon as needed. The SMART system, in par-

*The method of construction of such recall-precision plots has previously been described in detail.[9,10]

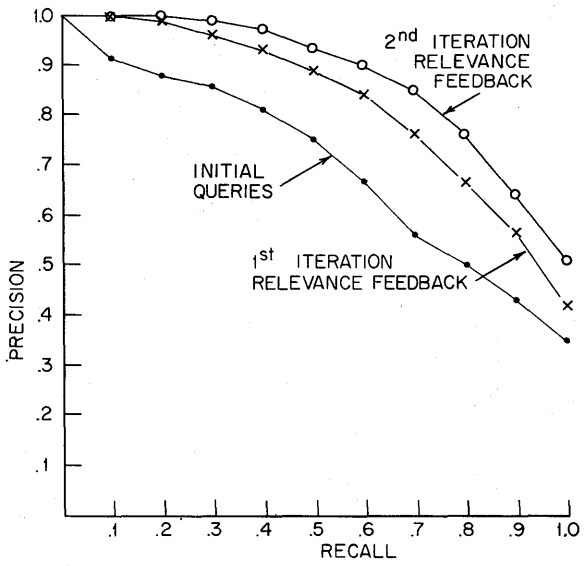



Figure 5. Precision versus recall for initial queries and queries modified by relevance feedback (averaged over 24 search requests).

ticular, provides a large variety of indexing methods including the following:

1. A *null thesaurus* procedure which uses the *word stems* originally included in documents and search requests for content identification.
2. A *synonym dictionary* ("Harris" thesaurus) which replaces original word stems by synonym classes or concept numbers.
3. A *hierarchical arrangement* of concept numbers which can be used, given a set of concepts to obtain more general ones ("hierarchy up"), or more specific ones ("hierarchy down").
4. A *statistical phrase* procedure which is used to replace pairs or triples of co-occuring (related) concepts by a single "phrase" concept (e.g., the concepts "program" and "language" might be combined into "programming language").
5. A *syntactic phrase* process which generates phrases only if the components in fact exhibit an appropriate grammatical relationship.
6. A variety of so-called *merged methods*,[9] in which the system proceeds iteratively through two or three simple processes and combines the output.

Obviously, the ability to generate a multiplicity of distinct index images for each document does not necessarily imply that each modification in the anlysis process results in large-scale improvements in the search effectiveness. Experiments conducted with the SMART system have, however, shown that in many cases a considerable increase in retrieval effectiveness is obtainable when changes in the analysis are adapted to the aims of each particular user.

Consider, in this connection, the evaluation output for a variety of analysis methods produced by the SMART system, reproduced in Fig. 6 and 7. Figure 6 contains output for the search request titled "Automata Phrases," with nine relevant documents. Six simple analysis methods are shown: null thesaurus, "Harris Two" (version 2 of the regular synonym dictionary), statistical phrases, syntax phrases, hierarchy up, and hierarchy down. Thirteen "merged" methods, each including two simple components, are also included in Fig. 6, as well as nine

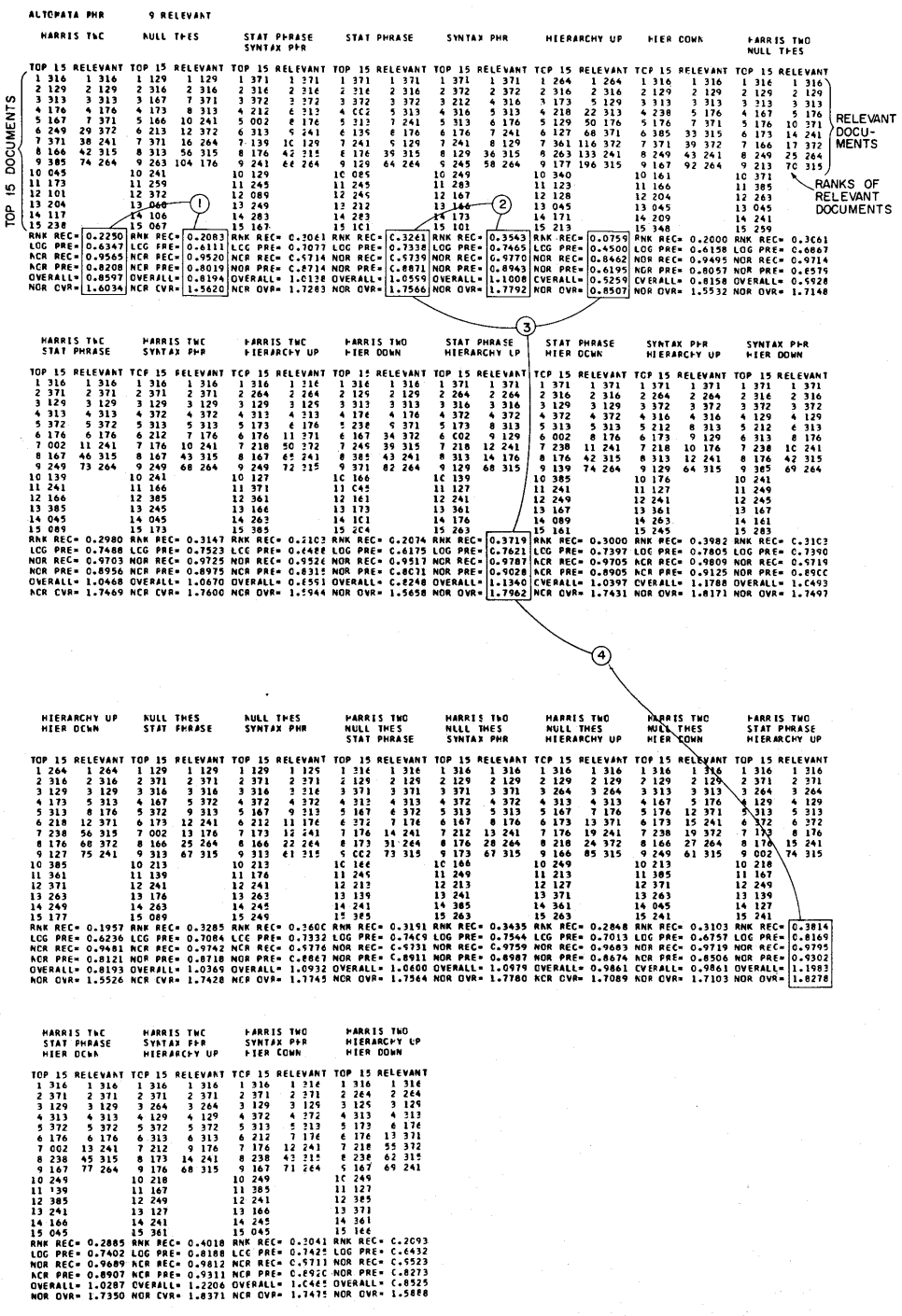



Figure 6. Evaluation output for request "automata phrases" (28 different analysis methods).

triple merges.* For each method, the output is presented in two parts: the left part includes the document numbers of the first 15 documents retrieved by that method, whereas the right-hand side con-

*The ranked document output for the "merged" methods is produced by taking the ranked lists for the individual component methods and merging these lists in such a way that all documents with rank 1 precede all documents with rank 2, which in turn precede documents with rank 3, and so on.

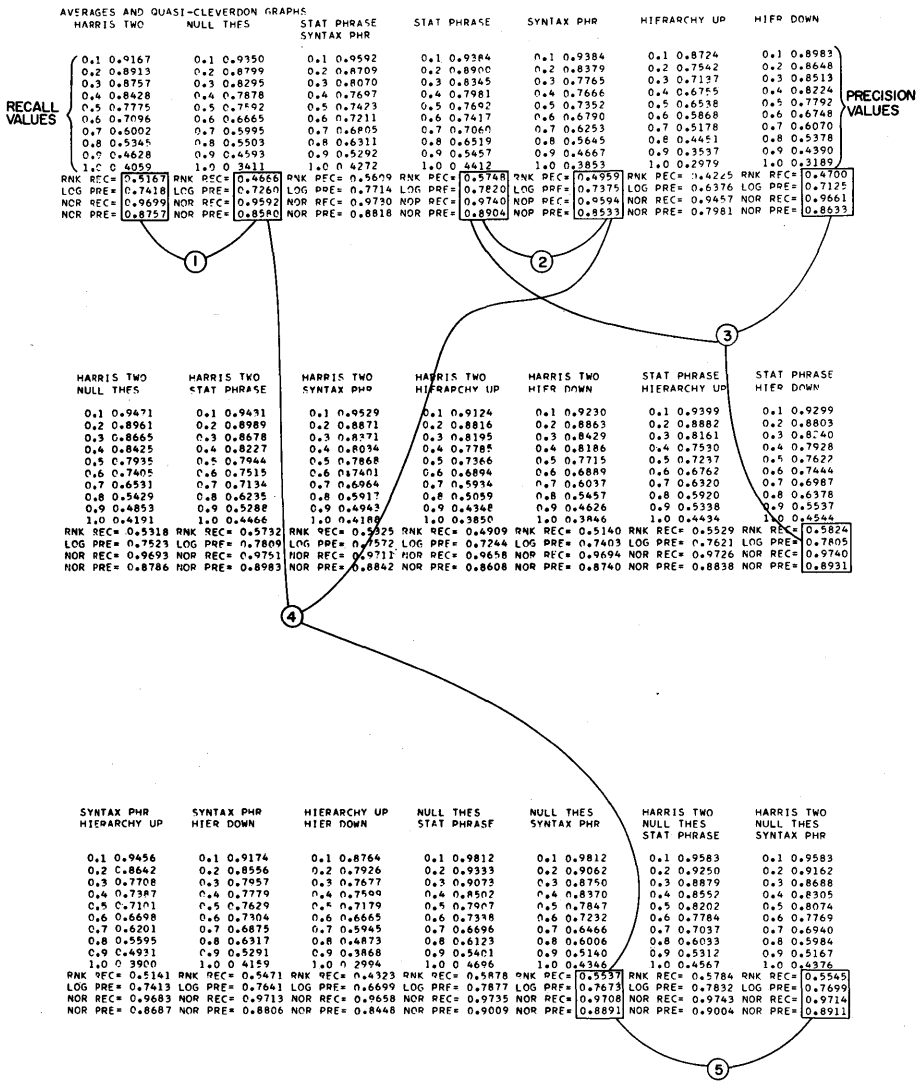



Figure 7. Precision vs. recall plots for 28 analysis methods (averages shown over 17 search requests).

sists of only the *relevant* document numbers and their ranks in decreasing correlation order with the request. Below the lists of document numbers, a variety of recall and precision measures are provided for each analysis procedure, to reflect the effectiveness of the corresponding process.

An examination of Fig. 6 reveals, for example, that for the request on "Automata Phrases," improved retrieval is obtained by switching from the word stem procedure to the synonym recognition process using the regular thesaurus (labeled 1 in Fig. 6). This is reflected both by the magnitude of the evaluation coefficients, and by the ranks of the last relevant document (104th out of 405 for the word stem process (null thesaurus), and 74th for "Harris Two"). An improvement is also obtained by switching from "Harris" thesaurus to the phrase procedures, and from statistical phrases to syxtax phrases (labeled 2). The third example from Fig. 6 shows that the merged procedure which combines the statistical phrases with the hierarchy results in an increase in performance over and above each of the component methods. A further improvement is obtained by adding the regular "Harris 2" thesaurus process to the previously merged pair (example four of Fig. 6).

Figure 7 shows evaluation output obtained for the same 28 analysis methods previously shown in Fig. 6, but averaged over 17 different search requests. The output of Fig. 7 is presented in the form of precision vs. recall graphs, similar to that shown in Fig. 5 (the actual graphs are not drawn but tables are presented instead). The five examples specifically indicated in Fig. 7 again confirm the earlier results that improvements are obtainable from method to method.

Each of the three search optimization procedures described in this study appears to be useful as a means for improving the retrieval effectiveness of real-time, user-controlled search systems. Additional experimentation with larger document collections and with an actual user population may be indicated before incorporating these procedures in an operational environment. Iterative, user-controlled search procedures appear, however, to present an interesting possibility, and a major hope, for the eventual usefulness of large-scale automatic information retrieval systems.

## REFERENCES

1. G. Salton, "A Document Retrieval System for Man-machine Interaction," *Proceedings of the ACM 19th National Conference,* Philadelphia, 1964.

2. G. Salton and M. E. Lesk, "The SMART Automatic Retrieval System—An Illustration," *Comm. of the ACM,* vol. 8, no. 6 (June 1965).

3. G. Salton et al, "Information Storage and Retrieval," Reports No. ISR-7 and ISR-8 to the National Science Foundation, Computation Laboratory, Harvard University (June and Dec. 1964).

4. V. E. Giuliano and P. E. Jones, "Linear Associative Information Retrieval," *Vistas in Information Handling,* P. Howerton, ed., Spartan Books, Washington, D.C., 1963.

5. R. M. Curtice and V. Rosenberg, "Optimizing Retrieval Results with Man-machine Interation," Center for the Information Sciences, Lehigh University, Bethlehem, Pa., 1965.

6. J. F. Rial, "A Pseudo-metric for Document Retrieval Systems," Working Paper W-4595, MITRE Corp., Bedford, Mass. (1962).

7. J. J. Rocchio, "Relevance Feedback in Information Retrieval," Report No. ISR-9 to the National Science Foundation, Sect. 23, Computation Laboratory, Harvard University (Sept. 1965).

8. J. J. Rocchio, "Performance Indices for Document Retrieval Systems," Report No. ISR-8 to the National Science Foundation, Sect. 3, Computation Laboratory, Harvard University (Dec. 1964).

9. G. Salton, "The evaluation of Automatic Retrieval Procedures—Selected Test Results Using the SMART System," *American Documentation,* vol. 16, no. 3 (July 1965).

10. C. W. Cleverdon, "The Testing of Index Language Devices," *ASLIB Proceedings,* vol. 15, no. 4 (Apr. 1963).

# AN ECONOMICAL PROGRAM FOR LIMITED PARSING OF ENGLISH

D. C. Clarke and R. E. Wall*
*IBM San Jose Research Laboratory*
*San Jose, California*

Automatic syntactic analysis has often been proposed as a component of mechanized indexing systems.[1,2] However, up to this time, frequency counting and statistical association techniques have been more favored, since these involve operations which can be performed with great speed on present day computers. Syntactic analysis programs,[3] especially the few which have relatively complete grammars, have suffered from the disadvantage of slow and expensive operation and consequently have seldom been applied beyond the field of mechanical translation. In this paper, we report the design and testing of a limited syntactic recognition program for English which shows promise of becoming accurate enough to aid in mechanized indexing, yet sufficiently inexpensive to make large-scale use practicable.

We originally developed this system as an extension of Baxendale's Title Analyzer Program,[4] which used a small number of syntactic clues and a discard list to select "significant" words and phrases from titles of technical articles for use as index terms. However, the shallowness of an index produced only from titles seriously limited the applicability of the Baxendale program, so it seemed natural to apply similar techniques to abstracts of technical articles as well. Abstracts, like titles, are intended to be concise statements of the information in a document, but by virtue of their greater length should provide more potential index terms.

The syntactic recognition problem with abstracts, however, is much more difficult than with titles. The latter often consist only of noun and prepositional phrases and rarely post-modifying participles or relative clauses. Abstracts, on the other hand, potentially exhibit the full range of syntactic constructions of formal written English, except for interrogative and exclamatory sentences, which are excluded by precepts of style. For this reason, the fairly simple procedures of the Title Analyzer Program were inadequate for dealing with abstracts with any reasonable degree of accuracy. While our initial efforts were directed (as in the Title Analyzer) toward the identification of nouns and their modifiers, the result has been a program written in COMIT[5] yielding a nearly complete parsing of the "surface" syntactic structure of each sentence. The overall sequence of operations in the program is shown in Fig. 1.

In the description of the program which follows we will emphasize those features of the design imposed by the necessity for economical performance in a projected mechanized indexing system.

The program accepts cards in a format which is

*Present address (R. E. Wall): Department of Linguistics, Indiana University, Bloomington, Ind.

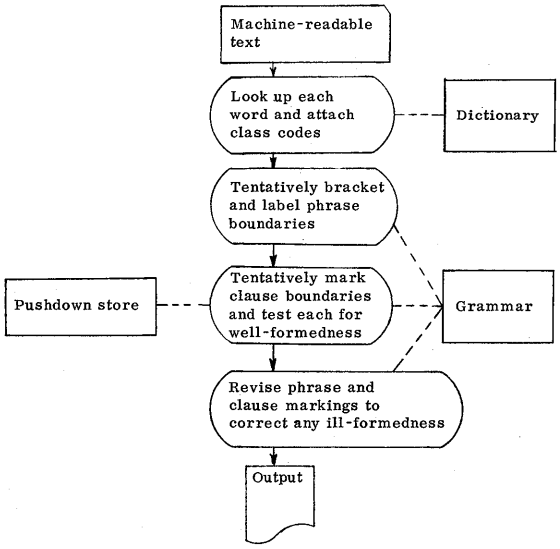



Figure 1. Sequence of syntactic analysis program.

N-type GaAs doped with Te to 3 X $10^{17}$ $cm^{-3}$ and 9.6 X $10^{17}$ $cm^{-3}$ have absorption coefficients of 20 $cm^{-1}$ and 10 $cm^{-1}$, respectively, at 1.475 eV and 77° K.

Printed input text

*N-TYPE *GA*AS DOPED WITH *TE TO *(*( **F *) AND *(*( **F *) HAVE ABSORPTION COEFFICIENTS OF *(*( **F *) AND *(*( **F *) , RESPECTIVELY , AT 1.475 E*V AND 77 DEGREES *K .

Keypunched input text

Figure 2. Sample input sentence.

essentially a character-for-character transcription (Fig. 2). We have accepted the present necessity of keypunching and have therefore concentrated on making the input format convenient for our syntactic analysis program. Nonetheless, we feel that this format will be reasonably compatible with automatically transcribed text whenever such becomes generally available.

## DICTIONARY

The function of a dictionary in a syntactic recognition program is to assign each word* of the input text to one or more word classes (traditionally such

*In this discussion of the dictionary we are referring to words as *tokens* or separate occurrences, not as *types* or the total of different forms.

categories as noun, verb, adjective, etc.). The usual approach is to use a "complete" dictionary, which contains (ideally) every word form in the language. Our program uses a "computational" dictionary of the type described by Klein and Simmons[6] which makes word-class assignments on the basis of orthographic features. The current dictionary consists of three lists which contain about 1,000 entries in all:

1. Common function words—prepositions, articles, conjunctions, etc.
2. Word endings† — *-ing, -tion, -ed, -ous,* etc.
3. Exceptions to the word ending rules — *thing, feed, mention,* etc.

One requirement of the program was that it should be suitable for use in a mechanized indexing system. A complete dictionary operating on technical text would require an addition each time a new word was encountered. Such additions are not compatible with economic automatic processing. We thus chose to use a computational dictionary designed to encode correctly the relatively few types which account for the large proportion of tokens in running text.

Words which are not classified by the computational dictionary are arbitrarily assigned to the noun/verb category. This choice is again influenced by potential use of the program in a mechanized indexing system. The hypothesis is that the importance of nominal constructions in selection of index unit candidates places emphasis on the bracketing of all noun phrases. The grammatical algorithm is designed to deal with the noun/verb ambiguity.

One advantage of our computational dictionary is that it is small enough to be contained in the core storage of an IBM 7094 along with the grammar rules and program instructions. This allows for a binary search through the dictionary lists for every word as it occurs *in text order.* A full dictionary, on the other hand, would have to be stored on magnetic tape (in which case the input text would have to be run in batches, alphabetized, looked up, and re-sorted into text order) or else stored on drums or disks, thereby sacrificing the advantage of a binary search. A further consequence of the small computational dictionary contained in core storage is that the processing of sentences can be open-ended. Since there is no need to handle the text in batches, any number of sentences can be run without interruption once the program has been loaded.

These conveniences, however, are paid for in two ways. One is the obvious limitation that many words will receive the arbitrary noun/verb classification because they were not found in the dictionary, and any misclassification may lead to erroneous phrase bracketings. The other disadvantage is in the lack of refinement possible in certain word classes. For example, although the suffix *-tion* nearly always serves to identify words as noun forms, they cannot be further subclassified by this clue as animate-inanimate, abstract-concrete, or countable-uncountable. Thus, a computational dictionary introduces error in syntactic recognition not only by incorrect word-class assignments, but also by limiting the discrimination which can be made in the grammar.

## GRAMMAR

The grammar gives rules for the allowed combination of word classes into phrases and clauses. Nine types of phrases—nominal, pronominal, adjectival, past participial, present participial, prepositional, verbal, infinitive, and adverbial—and eight kinds of subordinate and relative clauses are recognized. The kind of clause is dependent on the clause introducer and on the alternative structural patterns predicted by that introducer. For example, if a clause is introduced by *which,* the grammar expects that a verb will be found but that a subject is optional. If a verb is not found, the algorithm will search for a re-bracketing to fulfill the requirement for a verb. The output is a labeled bracketing of these phrases and clauses together with the syntactic word class for each word. An example of the output is shown in Fig. 3.

Each phrase is enclosed in parentheses and is followed immediately by an identifying label (NOUP = noun phrase, PREP = prepositional phrase, etc.). A hyphen separates the phrase label from a list of the word classes assigned to each word in the phrase. These classes are based on those of Kuno and Oettinger[8] with many modifications. Although a complete list of the word classes and their defining characteristics would be too long to include here, it would perhaps be helpful to give a few of those appearing in Fig. 3.

†A reverse-alphabetized word list[7] was most helpful in discovering word endings and exceptions.

```
THIS PAPER PRESENTS A GENERAL SYSTEM CONFIGURATION FOR AN ARITHMETIC UNIT
OF A COMPUTER WHICH IS USED TO SOLVE POLYNOMIAL PROBLEMS EFFICIENTLY

     (THIS PAPER ) NOUP-ATES NOUS

     (PRESENTS ) VERB-VZZS

     (A GENERAL SYSTEM CONFIGURATION ) NOUP-ATBS NOVS NOUS NOUS

     (FOR AN ARITHMETIC UNIT ) PREP-PRE ATBS NOVS NOUS

     (OF A COMPUTER ) PREP-PRE ATBS NOUS

     ***BC 1

     WHICH-RL3

     (IS USED ) VERB-VB1S PZ1

     (TO SOLVE ) INFP-TOIS VZZP

     (POLYNOMIAL PROBLEMS ) NOUP-NOVS NOUP

     (EFFICIENTLY ) ADVP-AV1

     ***EC 1
```

Figure 3. Sample of output from syntactic analysis program.

| | |
|---|---|
| NOUS | noun, singular number |
| PRE | preposition |
| VZZS | finite form verb, third person, singular number, transitivity unknown |
| NOVS | noun/adjective |
| ATBS | article modifying singular nominals |
| PZI | past participle homographic with past tense form |
| AVI | regular adverb |

The beginning and end of the relative clause are marked by the symbols ***BC 1 and ***EC 1.

Such an analysis falls short of a "complete" phrase-structure parsing in two ways. First, the labelings of fine structures within phrases are suppressed. For example, in the sentence shown in Fig. 3, the noun phrase *an arithmetic unit* is not overtly labeled as such but is included in the prepositional phrase. Likewise, the finer structure of this noun phrase itself (*arithmetic unit* = NP, etc.) is not given explicitly.

Using phrase-structure tree diagrams we might illustrate the difference as in Fig. 4. The tree diagram at the left represents the phrase marker as it might appear for this 4-word prepositional phrase in a phrase-structure parsing; the tree at the right shows the same phrase as it would be analyzed by our grammar.

The second difference is the failure to mark dependencies between phrases and clauses. For example, in the output in Fig. 3, the point of attachment of the prepositional phrase *for an arithmetic unit* is not specified. The clues for joining such modifiers to the proper head seem in many cases to be purely semantic ones, and the problem is always troublesome in any parsing scheme. Jane Robinson cites the example[9] *I saw the man with the telescope in the park,* which can have several different readings, depending on the words which the prepositional phrases are understood to modify. We do not yet know whether the simple expedient of joining post-modifiers to the nearest allowable preceding structure can be improved upon with the aid of syntactic information alone, nor do we know how much of this interphrase structure will be necessary in order to do the job of indexing. In any case, delimiting the phrase boundaries as we have done is a prerequisite to any attempt to specify these dependencies algorithmically.

The current implementation of our program does not incorporate an explicit, separable grammar. However, a formal description of the grammar in a context sensitive phrase-structure notation has been written to provide documentation.

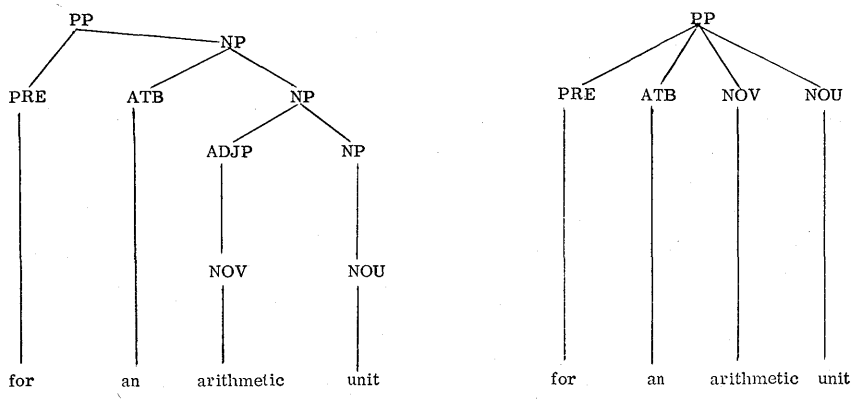



Figure 4. Comparison of phrase-structure tree diagrams.

## ALGORITHM

The algorithm is a set of procedures for assigning an allowable syntactic structure to an input sentence according to the rules set forth in the grammar. A serious problem in implementing parsing algorithms has always been that the processing time tends to increase exponentially with the number of words in the sentence being analyzed (because as sentence length increases, so in general does the number of combinatorial alternatives which must be considered). Consequently, the practical upper limit on sentence length for the best existing programs has been about 40 words, and for most programs it has been much below that. Longer sentences are not at all rare, however, particularly in technical writing, and any parsing system which is intended to be a practical component of an indexing system should be able to handle them. We therefore attempted to design this parsing algorithm so that the total processing time would be directly proportional to sentence length.

The general sequence of operations is as follows. After dictionary lookup is complete, all phrase boundaries are tentatively identified in one left-to-right pass through the sentence. On a second left-to-right pass, clause boundaries are established, and tests for well-formedness in each clause are performed. Nested clauses are handled by a pushdown storage mechanism.[10] Whenever an ill-formed condition is recognized, the algorithm initiates an ordered search for alternatives pertinent to that condition (different word classes or different phrase boundaries) and will choose the first alternative which resolves that condition. This strategy, together with the restriction that no set of alternatives can be tried more than once during the analysis of a sentence, avoids the repetitive tracing of substructures which have already been recognized as well-formed. Thus, even worst-case analysis times will vary linearly (or nearly so) with sentence length. A final series of passes through the sentence serves to link phrases joined by coordinating conjunctions and performs a few minor revisions before output.

We can illustrate one of the parts of the algorithm, the clause well-formedness testing, by using the example of the sentence in Fig. 3. During dictionary lookup, both *paper* and *presents* have received the arbitrary noun/verb coding, but the latter word, because of its *-s* suffix, has been coded as plural noun and third-person singular verb.

After the first pass, the phrases have been bracketed

as they appear in the output (Fig. 3), except for the first three words.


The noun homograph of *presents* has been tried first and has led to the incorrect bracketing shown. On the second pass the lack of a verb for the independent clause is noted (the relative clause is well-formed), and the algorithm then examines the first noun phrase, beginning at the right-most word, for a verb homograph. *Presents* is found to have such a homograph, so the subject is redefined as the remaining noun phrase, *This paper,* and the test for number agreement between subject and verb is made. Since this alternative produces a well-formed clause, the final bracketing is:


Had this alternative not been allowable, the other words in the phrase would have been examined for verb homographs. If none was found, the phrase would have been restored to its original bracketing and the next noun or prepositional phrase to the right in the same clause would have been broken apart and examined similarly. This procedure would have continued until either a verb and subject had been established or else all the relevant possibilities had been exhausted. However, when all the clauses of a sentence have been made well-formed, no more alternatives are tried. Thus, the algorithm arrives at only one syntactic structure for each sentence, in contrast with the multiple parsings generated by a program such as the Harvard Syntactic Analyzer.[8]

While multiple analyses are clearly useful in linguistic research for exposing syntactic ambiguities, in a practical application such as mechanized indexing they are an embarrassment of riches. Having all structural descriptions for each sentence would be of little use since at the present time there is no method for deciding which analyses are also semantically acceptable and, further, which is the one "correct" reading intended by the author. Perhaps one could hope to select instead the "syntactically most probably" parsing[11,12] if adequate statistics of English grammatical structures were available. Since they are not, we have ordered the search for alternatives according to what seem to us, intuitively, to be the most frequently occurring structures.

A useful by-product of the algorithm arises from the fact that all the phrases of a sentence are tentatively recognized in the first pass. Should the analysis of any sentence be terminated because of excessive time, or if no grammatically acceptable parsing for some clause can be found, many substrings of the sentence will still be correctly identified. Thus, in cases of noncatastrophic failure, we are able to get partial results and go on to the next sentence.

Provision is also made for handling parenthetic expressions (the clause well-formedness tests are omitted) and clauses separated by semicolons (treated as separate sentences). Sentences up to 100 words in length can be analyzed. However, some very long sentences, depending on the particular structures they contain, will occasionally require more COMIT "workspace" than is currently available. In such cases, the program writes out the results of the dictionary lookup routine on a tape which can later be used as input to the syntactic analysis portion.

## PROGRAM TESTING

The program was coded in COMIT[5] because of the ease it provides in the design and updating of experimental models. Initial debugging and testing were carried out on a sample of 70 consecutive sentences taken from abstracts in the *IBM Journal of Research and Development*. This text, which we will refer to as IBM #1, was chosen because it contained a fairly wide range of technical subject matter. The results were used to make further refinements to the grammar, and then the program was tested on 5 more texts, each containing 70 sentences, from randomly selected abstracts. One text was taken from another issue of the *IBM Journal* (IBM #2) and the others from the fields of chemistry, physics, acoustics, and, for comparison with the technical material, literary criticism. The accuracy with which phrases were identified* is indicat-

*The method of counting phrases is in accord with our earlier remarks about "complete" parsing. Thus, the object in a prepositional phrase does not count as an additional nominal or pronominal phrase.

Table 1. Accuracy of Identifying Phrases.

| Text | No. of Words | NP | PP | Pn P | Inf P | VP | PMA | PMR | PMP | Av P | Totals | Percent |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| IBM #1 | 1593 | 154/170 | 209/222 | 6/6 | 10/11 | 89/95 | 10/15 | 10/11 | 17/17 | 11/15 | 516/562 | 92 |
| IBM #2 | 1867 | 170/182 | 248/267 | 11/12 | 12/13 | 100/110 | 12/19 | 6/7 | 20/20 | 18/19 | 597/649 | 92 |
| Physics | 1805 | 153/163 | 240/264 | 11/11 | 14/14 | 112/118 | 21/27 | 11/13 | 11/11 | 19/20 | 592/641 | 92 |
| Chemistry | 1716 | 160/174 | 214/241 | 12/12 | 16/16 | 106/114 | 18/20 | 14/15 | 16/17 | 14/15 | 570/624 | 91 |
| Acoustics | 1705 | 174/187 | 233/249 | 5/5 | 18/21 | 117/123 | 10/11 | 8/11 | 22/23 | 14/15 | 601/645 | 93 |
| Literary criticism | 2192 | 193/216 | 249/283 | 54/54 | 27/27 | 133/161 | 24/35 | 4/6 | 10/14 | 37/44 | 731/840 | 87 |
| Total | 10878 | 1004/1092 | 1393/1526 | 99/100 | 97/102 | 657/721 | 95/127 | 53/63 | 96/102 | 113/128 | 3607/3961 | |
| | | 92% | 91% | 99% | 95% | 91% | 75% | 84% | 94% | 88% | 91% | |
| Technical material only | 8686 | 811/876 | 1144/1243 | 45/46 | 70/75 | 524/560 | 71/92 | 49/57 | 86/88 | 76/84 | 2876/3121 | |
| | | 93% | 92% | 98% | 93% | 94% | 77% | 86% | 98% | 90% | 92% | |

NP—noun phrase
PP—prepositional phrase
Pn P—pronoun phrase
Inf P—infinitive phrase
VP—verb phrase
PMA—post-modifying adjective
PMR—post-modifying present participle
PMP—post-modifying past participle
Av P—adverbial phrase

*Total number of words includes those contained in parenthetic expressions. This accounts for the discrepancies between the total and that given for the acoustics text in Table 1.

ed in Table 1. A further test using 1,000 sentences taken from *Nuclear Science Abstracts* gave similar results.

It was of interest to compare the performance of our program with that of one of the most complete parsing programs available, the Harvard Syntactic Analyzer (HSA).[8] We obtained this program from SHARE, modified it to produce only one analysis (i.e., the first found) for each sentence, and tested it on our first text, IBM #1. After homograph codes were supplied for words not found in the HSA dictionary, the program produced an analysis for 59 of the 70 sentences. Seven were rejected as ungrammatical, and four had not been analyzed after at least 5 minutes of running time on each by the SYNTAX subroutine. One sentence was allowed to run for 17 minutes without success. Table 2 shows the comparison of our program with the modified HSA in identifying phrases in the 59 sentences which were analyzed by both.

Table 2. Comparison with Modified Harvard Syntactic Analyzer (59 Sentences from IBM #1; 1244 Total Words).

| | NP | PP | Pn P | Inf P | VP | PMA | PMR | PMP | Av P | Totals |
|---|---|---|---|---|---|---|---|---|---|---|
| Our program | 115/126 | 157/169 | 3/3 | 9/10 | 73/79 | 7/9 | 7/8 | 12/12 | 7/9 | 390/425 |
| | 91% | 93% | 100% | 90% | 92% | 78% | 88% | 100% | 78% | 92% |
| HSA | 104/126 | 147/169 | 3/3 | 9/10 | 73/79 | 9/9 | 5/8 | 5/12 | 6/9 | 361/425 |
| | 83% | 87% | 100% | 90% | 92% | 100% | 63% | 42% | 67% | 85% |

## PROCESSING TIME

The core clock was used to measure the time of each phase of our program for every sentence analyzed. A program kindly supplied by Mr. K. L. Deckert of the IBM Systems Development Laboratory, San Jose, plotted the times against sentence length on a CALCOMP plotter and calculated the slopes of the resulting lines by the method of least squares. The summarized results appear in Table 3.

All the data appeared to be well represented by straight lines except for the second pass (testing for clause well-formedness and trying alternatives), which as expected displayed considerable scattering. The times for sentences containing parenthetic

Table 3. Average Processing Times for Each Phase of Program.

| Phase | sec Time/*word* |
|---|---|
| 1. Input and dictionary lookup | 0.072 |
| 2. Bracketing of phrases | 0.047 |
| 3. Testing clause well-formedness and trying alternatives | 0.019 |
| 4. Rebracketing coordinated structures and other minor corrections | 0.024 |
| 5. Output | 0.017 |
| TOTAL | 0.179 |

expressions, because they are treated in a special way, were also found to deviate markedly from the line. Nonetheless, we found no clear indication that the total time might be increasing exponentially with length for sentences of the order of 50 to 75 words.

The total time for our program to analyze the 59 sentences of IBM #1 (Table 2) was about 4.5 minutes plus 4.2 minutes for compilation (run on the IBM 7094 but adjusted to the IBM 7090 time). The time for the modified Harvard Syntactic Analyzer was about 30 minutes (not measured precisely), which does not include the time for dictionary lookup and updating. We have recently been informed by Dr. Kuno that the running time for the Harvard program has now been reduced substantially.[13] However, this version is not yet available to us for testing.

## DICTIONARY CODING

Since the computational dictionary is a fundamental part of our program, we were concerned with its ability to assign words to word classes compared to the assignments made by a complete dictionary. Klein and Simmons reported that their computational dictionary could correctly assign *unique* word-class codes to about 90 percent of the words in their sample texts. However, this figure measures the results of two operations: first, assigning each word all its possible word-class codes and, second, eliminating ambiguous codes by means of the context. Our dictionary performs only the first of these functions, while word-class ambiguities are resolved in the syntactic recognition routine. Therefore, we counted as "correct" those words which were identified by the computational dictionary or which were not found and were indeed noun/verb ambiguities. Table 4 gives the results for the acoustics text. Words coded arbitrarily received only the noun/verb classification; the fraction of these marked "incorrect" should have been assigned to either noun only or verb only, or else should have had additional codes attached as well. Similar data were collected for all six experimental texts with an earlier version of the dictionary. The percentage of correct coding was somewhat lower (86 percent), but it was nearly the same for each of the texts.

In order to determine the extent to which parsing errors arose from inadequacies in the computational dictionary, we reran two of the texts with corrections to dictionary coding supplied by hand. The overall accuracy in identifying phrases increased from 91 to 93 percent for the chemistry text and from 87 to 90 percent for literary criticism. Thus, using a perfect dictionary with the present grammar and algorithm seems to improve the accuracy by about 2 to 3 percent.

Table 4. Accuracy of Dictionary Coding for Acoustics Text.

| | Words | Percent |
|---|---|---|
| Found among common function words and exceptions to ending list | 1001 | 58 |
| Found in ending list | | |
| Correct | 440 | 25 |
| Incorrect | 13 | 1 |
| Coded arbitrarily (noun/verb) | | |
| Correct | 137 | 8 |
| Incorrect | 143 | 8 |
| Total coded correctly* | 1578 | 91 |
| Total coded incorrectly | 156 | 9 |

## DISCUSSION

The principal result of our work thus far has been to show that the approach to parsing, which we adopted for purely practical reasons, nonetheless succeeds as well *in identifying phrases* as at least one other more sophisticated routine. We were frankly surprised at this result. Because our program was to operate with many handicaps—a minimal dictionary, simple grammar, and severe time and space constraints on the whole program—we did not suppose that it would be able to perform so well.

We must emphasize that the comparison with the HSA (1963 version) should be accepted with reservations. The HSA, as previously noted, provides a more complete syntactic description of each sentence than does our program, and therefore the running times are not directly comparable. Also, the sample for comparison, only 59 sentences, is rather small. One might also argue that selecting only the HSA's first analysis from each sentence may have produced a bias, but there seems to be no reasona-

ble alternative to this choice, and in fact there is some reason to believe that the first analysis (rather than the second, the last, etc.) has a greater probability of being the "correct" syntactic analysis than does any other.[11]

We believe that two factors are chiefly responsible for the degree of success which our program has so far achieved. First, we have made some fortunate guesses about the probability of occurrence of syntactic constructions, at least for the kind of technical writing we have investigated. (Note that the accuracy for the literary criticism text was somewhat lower than for the others despite the fact that the dictionary coded about the same percentage of words correctly in this text.) The second factor is the strategy of searching out alternatives only to remedy a particular syntactic ill-formation. This technique allows most of the previously made "probable" choices to be left intact whenever an error is corrected.

Three kinds of errors were frequently made by our program: (1) incorrect bracketing of coordinated structures around *and* or *or;* (2) unresolved noun/verb ambiguity; and (3) incorrect assignment of words with suffix *-ing,* which may be adjectival (pre- and post-modifying), verbal, or gerundive. It is interesting to note that the pattern of errors made by the HSA differs considerably. A frequent mistake was also the noun/verb confusion, but it usually arose from erroneously finding relative clauses beginning with an elliptical *which* or *that.* For example, a sentence beginning [*The maximum signal*] [*has*] . . . was analyzed as if it has been [*The maximum*] (*which*) [*signal*] [*has*] . . ., with a plural noun occurring later in the sentence being called the predicate verb for the subject *maximum.*

The current version of our program occupies about 29,000 registers of an IBM 7094, thus allowing about 4,000 registers for COMIT "workspace" during analysis. Therefore, no substantial additions to the dictionary, grammar, or algorithm are possible while maintaining the program's present design on the IBM 7094. Some slight improvements in performance can undoubtedly be made at the expense of much more investigation into grammatical refinements. This would undoubtedly lead to an increase in running time and storage space required. We do not know what the limits of accuracy are for our approach, but we estimate that less than half the errors are in theory correctable (by an expanded grammar); the remaining are genuine syntactic ambiguities which presumably can only be resolved by extrasyntactic information. If this estimate holds, it means that an accuracy of about 94 percent in identifying phrases is theoretically attainable.

The point of balance between cost and accuracy will depend on the particular application envisioned for the program. Despite the encouraging results thus far, we cannot claim that our program will guarantee the feasibility of a mechanized indexing system. It is clear that more will be required for automatic indexing than an identification of phrases and clauses. For example, it may be necessary to specify some interphrase dependencies, and a means for the deletion of items deemed nonsignificant will almost certainly be necessary. Also, some syntactic transformations to convert the material into a format suitable for searching (whether by machine or human) will probably be essential. Nonetheless, the prospects for using at least a limited syntactic analysis program in automatic indexing on a large scale now seem much more hopeful.

## ACKNOWLEDGMENT

The authors wish to express their appreciation to Mr. J. Bennett and Miss P. Baxendale for substantial contributions to this work throughout its development. Mr. Bennett assisted materially in the programming and made several modifications to the COMIT system which greatly expedited the debugging.

## REFERENCES

1. W. D. Climenson, H. H. Hardwick and S. N. Jacobson, "Automatic Syntax Analysis in Machine Indexing and Abstracting," *American Documentation,* vol. 12, pp. 178-183 (1961).

2. G. Salton (principal investigator), Report No. ISR-7 to the National Science Foundation, Harvard Computation Laboratory, Cambridge, Mass. (1964).

3. D. G. Bobrow, "Syntactic Analysis of English by Computer—A Survey," *Proceedings of the Fall Joint Computer Conference,* 1963, p. 365.

4. P. B. Baxendale, "An Empirical Model for Machine Indexing," *Machine Indexing Progress and Problems,* papers presented at the Third Institute on Information Storage and Retrieval, American University, 1961, p. 207.

5. V. H. Yngve, "COMIT As an IR Language," *Communications of the ACM,* Jan. 1962, pp. 19-28.

6. S. Klein and R. F. Simmons, *Journal of the ACM,* 1963, p. 334.

7. A. F. Brown (ed.), *Normal and Reverse English Word List,* Univ. of Pennsylvania, Philadelphia, 1963.

8. A. G. Oettinger (principal investigator), Report No. NSF-8 to the National Science Foundation, Harvard Computation Laboratory, Cambridge, Mass. (Jan. 1963).

9. J. J. Robinson, "Preliminary Codes and Rules for the Automatic Parsing of English," Memorandum RM-3339-PR, RAND Corp., Santa Monica, Calif. (Dec. 1962).

10. A. G. Oettinger, "Automatic Syntactic Analysis and the Pushdown Store," *Proceedings of the Twelfth Symposium in Applied Mathematics,* American Mathematical Society, 1961, p. 104.

11. R. E. Wall, "Probabilistic Approach to Ordering Multiple Analyses of Sentences," in Report No. NSF-13, Harvard Computation Laboratory, Cambridge, Mass. (Mar. 1964).

12. K. C. Knowlton, "Sentence Parsing with a Self-Organizing Heuristic Program," Ph.D. thesis, Massachusetts Institute of Technology, Sept. 1962.

13. S. Kuno, "The Predictive Analyzer and a Path Elimination Technique," *Communications of the ACM,* July 1965, pp. 453-462.

# THE MITRE SYNTACTIC ANALYSIS PROCEDURE FOR TRANSFORMATIONAL GRAMMARS*

Arnold M. Zwicky,† Joyce Friedman,†
Barbara C. Hall,† and Donald E. Walker
*The MITRE Corporation*
*Bedford, Massachusetts*

## INTRODUCTION

A solution to the analysis problem for a class of grammars appropriate to the description of natural languages is essential to any system which involves the automatic processing of natural language inputs for purposes of man-machine communication, translation, information retrieval, or data processing. The analysis procedure for transformational grammars described in this paper was developed to explore the feasibility of using ordinary English as a computer control language.

*The research reported in this paper was sponsored by the Electronic Systems Division, Air Force Systems Command, under Contract AF19 (628) 2390. The work was begun in the summer of 1964 by a group consisting of J. Bruce Fraser, Michael L. Geis, Hall, Stephen Isard, Jacqueline W. Mintz, P. Stanley Peters, Jr., and Zwicky. The work has been continued by Friedman, Hall, and Zwicky, with computer implementations by Friedman and Edward C. Haines. Walker has directed the project throughout. The grammar and procedure are described in full detail in reference 1. This paper is also available as ESD-TR-65-127.

†Present addresses of Zwicky, Friedman and Hall are, respectively: Department of Linguistics, University of Illinois, Urbana; Computer Science Department, Stanford University, Stanford, Calif.; Department of Linguistics, University of California, Los Angeles.

### *The Problem of Syntactic Analysis*

Given a grammar‡ G which generates a language L(G), we can define the *recognition problem for G* as the problem of determining for an arbitrary string x of symbols, whether or not $x \in L(G)$. The more difficult problem of *syntactic analysis* is to find, given any string x, all the structures of x with respect to G.

The syntactic analysis problem varies with the class of grammars considered, since both the formal properties of the languages generated and the definition of *structure* depend on the form of the grammar.

A context-free (CF) phrase-structure grammar is a rewriting system in which all rules are of the form $A \rightarrow \varphi$, where $\varphi$ is a non-null string and A is a single symbol; in context-sensitive (CS) phrase-structure grammars all rules are of the form $\psi_1 A \psi_2 \rightarrow \psi_1 \varphi \psi_2$, where A and $\varphi$ are as before and $\psi_1$ and $\psi_2$ are strings (possibly null) of terminal and/or nonterminal symbols. A derivation in a phrase-structure grammar is represented by a tree in which the terminal elements constitute the derived string. In a transformational grammar there is, in addition to a phrase-structure

‡The linguistic concepts on which this work is based are due to Noam Chomsky; see, for example, references 2-6.

component, a set of transformational rules which operates upon trees from the phrase-structure component to produce trees representing sentences in their final form.

For both types of phrase-structure grammars the syntactic analysis problem is known to be solvable. In the case of CF grammars, a number of general recognition and syntactic analysis procedures have been developed and programmed. Several syntactic analysis algorithms for CS grammars given by Griffiths (1964).[8]

The case of transformational grammars is complicated by the fact that they do not correspond exactly to any well-studied class of automata. In fact, a number of decisions crucial to their formalization have yet to be made. This situation makes it impossible to describe a general recognition procedure for transformational grammars without explicit conventions about the form and operation of transformational rules. Since there is no widespread agreement as to the most desirable conventions, it is likely that different people working on the analysis problem for transformational grammars are actually working on quite different problems. A solution to the problem with one set of conventions will not necessarily be a solution to the problem with a different set of conventions. Furthermore, the solution in one case would not necessarily imply the existence of a solution in another.

The area of formal properties of transformational grammars needs more study; the results of this attempt to solve the syntactic analysis problem for a particular one may help in determining the further restrictions needed on the form of transformational rules.

## THE MITRE GRAMMAR

In order to develop an analysis procedure it was necessary to fix on a particular set of conventions for transformational grammar. Many of these conventions agree essentially with the more or less standard conventions in the literature; points on which general agreement has not been reached will be noted.

The grammar contains two principal components: a CS phrase-structure component and a transformational component.* The rules in the phrase-structure component serve to generate a set of *basic trees* that are then operated upon by the rules of the transformational component to produce a set of *surface trees*.

### *Phrase-structure Component*

A CS phrase-structure rule $\psi_1$ A $\psi_2 \rightarrow \psi_1 \varphi \psi_2$ is written in the form $A \rightarrow \varphi / \psi_1 - \psi_2$. $\psi_1$ or $\psi_2$ or both may be null. The rules are ordered; consecutive rules expanding the same symbol in the same context are considered to be subrules of a single rule. A rule in this sense is thus an instruction to choose any *one* of the specified expansions.

The initial symbol of the grammar is SS, and the first phrase-structure rule is SS → # S #.† Further instances of SS and S are introduced by later rules. These instances are expanded during a succeeding pass through the phrase-structure rules during which new instances may be introduced, etc. The result is a tree that may contain sentence-within-sentence structures of arbitrary depth. This version of the phrase-structure component differs somewhat from the more usual versions, but is similar to the version presented in Chomsky.[3]

We shall use the following tree terminology: x is a *daughter* of y, x (not necessarily immediately) *dominates* y, x is the (immediate) *right* (*left*) *sister* of y, x is *terminal,* and the sequence $x_1, x_2, \ldots, x_n$ is a (proper) *analysis* of x. We shall also refer to the (*sub*) *tree headed* by x. These terms are all either standard of self-explanatory.

### *Transformational Component*

*Form of the Rules.* A transformational rule specifies a modification of a tree headed by the node SS or S. Every such rule has two main parts, a *description statement* and an *operation statement.*

The description statement sets forth general conditions that must be satisfied by a given tree. If these conditions are not met, then the rule cannot be applied to the tree. The conditions embodied in a description statement are conditions on analyses of

*There is a third component, the lexicon, which will not be discussed in detail here.

†The first phrase-structure rule in the MITRE grammar differs from this rule by allowing for the conjunction of any number of sentences. SS may then dominate a sequence of conjoined sentences. S, on the other hand, never immediately dominates such a sequence.

sentences (trees headed by SS, # S #, or S‡); a description statement is to be interpreted as requiring that the given tree have at least one analysis out of a set of analyses specified by the description statement.

If a tree satisfies the conditions embodied in a description statement, then the operations apply to the subtrees headed by the nodes in the analysis. The operation statement lists the changes to be made — the deletions, substitutions, and movements (adjunctions) of subtrees.

In addition to a description statement and an operation statement, a transformational rule may involve a number of *restrictions*. A restriction is an extra condition on the subtrees. The extra condition is either one of equality (one subtree must be identical to another) or of dominance (the subtree must contain a certain node, or must have a certain analysis, or must be a terminal node). Boolean combinations of restrictions are permitted.

The form of a transformational rule can be illustrated by the following example:

```
TWH2
(#) (Q) ($NIL NG) (AUXA) ($SKIP NP AP $RES 19)
 1   2       3       4              5
(5) ADLES 4                  $RES 19:    dom WH
ERASE 5
```

The description statement of this rule (TWH2) consists of five numbered and parenthesized *description segments*. Each segment specifies one part of an analysis. When several grammatical symbols (symbols not beginning with $) are mentioned in a segment, the interpretation of the segment is that the corresponding part of the analysis must be a subtree headed by *one* of these symbols. When $NIL is mentioned in a segment, the interpretation is that the corresponding part of the analysis is optional—that is, the corresponding part may be a null subtree; if, however, some analysis can be found in which the corresponding part is *not* null, that analysis must be chosen. The occurrence of $SKIP in a segment is equivalent to a variable between that segment and the preceding one.* $RES must be followed by the number of the restriction to which it refers. There is an implicit variable at the end (but not at the beginning) of every description statement.

In a more informal and traditional notation, the description statement of TWH2 would be written as

$$\underbrace{\#\ +\ Q\ +\ (NG)}_{1}\ -\ AUXA\ -\ X\ -\ \underbrace{\begin{Bmatrix} NP \\ AP \end{Bmatrix}}_{2\quad 3\quad 4}\ -\ \underbrace{Y\ +\ \#}_{5}$$

In our system there is no way of referring to a sequence of subtrees as a single part of an analysis, althiugh there is in the more informal notation.

In outline, the routine that searches through a tree for an analysis that conforms to a given description statement searches from left to right through the tree, attempting (in the case of a segment containing $NIL) to find a real node before assuming that a segment is null, attempting always (in the case of a segment containing $SKIP) to "skip" the smallest possible number of nodes, and checking (in the case of a segment containing $RES n) to see if a restriction is satisfied as soon as a node to which the restriction applies is found. In case one part of the search fails, either because the required nodes cannot be found or because a restriction is not satisfied, the routine backs up to the most recent point at which there remains an alternative (e.g., the alternative of searching for NP or for AP in the fifth segment of TWH2). As each part of the analysis is found, the appropriate subtrees are marked with numbers corresponding to the numbers on the description segments. The tree then undergoes the modifications specified in the operation statement.

The operation statement of TWH2 consists of an (ordered) list of two *instructions*. There are three types of instructions: the *adjunction* instructions, the *substitution* instruction, and the *erasure* instruction. The adjunction instructions are of the form ($\varphi$) AD n, where $\varphi$ is a sequence containing numerals (referring to the marked subtrees) or particular grammatical symbols or both, where AD is one of the four adjunction operations — ADLES (add as left sister), ADRIS (add as right sister), ADFID (add as first daughter), or ADLAD (add as last daughter) — and where n is a numeral referring to a marked subtree. The instruction (5) ADLES 4 specifies the adjunction of a copy of the subtree marked 5 as the left sister of the node heading the subtree marked 4. Substitution instructions are of the form ($\varphi$) SUB n, where $\varphi$ and n are as before. When such an instruction is applied, copies of the elements of $\varphi$ replace the

‡This distinction is not important for our discussion here. See the discussion in reference 1.

*$SKIP and $NIL may not both be used in a single segment.

subtree marked n, and this subtree is automatically erased.*

Erasure instructions are of the form ERASE n (erase the subtree marked n and any chain of nonbranching nodes immediately above this subtree) or ERASE Ø (erase the entire tree). The ERASE Ø instruction permits us to use the transformational component as a "filter" that rejects structures from which no acceptable sentence can be derived.

*Derivations.* The transformational rules are distinguished as being *obligatory* or *optional, cyclical* or *noncyclical,* and *singularly* or *embedding.* The obligatory/optional distinction requires no special comment here.

A rule is cyclical if it can be applied more than once before the next rule is applied. A rule may be marked as cyclical either (a) because it can be applicable in more than one position in a given sentence (say, in both the subject and object noun phrases), or (b) because it can apply once to yield an output structure and then apply again to this output. Otherwise, the rule is marked as noncyclical. In the present grammar case (b) does not occur.

Singulary rules are distinguished from embedding rules on the basis of the conditions placed upon the tree search. In the case of a singulary rule the search cannot continue "into a nested sentence"—that is, beneath an instance of SS or S within the sentence being examined; the search may, of course, pass over a nested sentence. In the case of an embedding rule the search can continue into a nested sentence, but not into a sentence nested in a nested sentence. Singulary rules operate "on one level," embedding rules "between one level and the next level below."

The transformational rules of our grammar are grouped into three sets—a set of initial singularies, a set of embeddings with related singularies, and a set of final singularies.† The rules are linearly ordered within each set.

The initial singularies operate on the output of the phrase structure component; they can be considered as applying, in order, to all subtrees simultaneously, since these rules do nothing to disturb the sentence-within-sentence nesting in a tree. There are numerous ways to order the application of these rules with respect to the nesting structure of a tree, and they are all equivalent in output.

The embeddings and related singularies operate on the output of the initial singularies. These rules require a rather elaborate ordering. Let us define a *lowest sentence* as an instance of # S # in which S does not dominate # and a *next-to-lowest-sentence* as an instance of # S # in which S dominates at least one lowest sentence and no instance of # S # that are not lowest sentences. At the beginning of the first pass through the embeddings and related singularies, all lowest sentences are marked. The rules will be applied, in order, to the marked subtrees. At the beginning of each subsequent pass, all next-to-lowest sentences will be marked, and the rules will again be applied, in order, to all marked subtrees. Characteristically, the embedding rules, when applied during these later passes, erase boundary symbols and thus create new next-to-lowest sentences for the following pass. However, only those subtrees marked at the beginning of a pass can be operated upon during the pass. The process continues until some pass (after the first) in which no embedding rules have been applied.

The final singularies operate on the output of the embeddings and related singularies. They can be considered as applying, in order, to all subtrees simultaneously.

A tree that results from the application of all applicable transformational rules is a *surface tree.* Each surface tree is associated with one of the sentences generated by the grammar.

*Dimensions*

The MITRE Grammar generates sentences with a wide variety of constructions—among them, passives, negatives, comparatives, *there*-sentences, relative clauses, yes-no question, and WH-questions. The dimensions of the grammar (excluding all rules concerned with conjunction) are as follows:

Phrase Structure Component:

Transformational Component:

75 rules
approximately 275 subrules

13 initial singularies

*If $NIL is chosen in the nth description segment, then ($\varphi$) AD n or ($\varphi$) SUB n is vacuous. Null terms in $\varphi$ are ignored; if all of $\varphi$ is null the instruction is vacuous.

†There is also a fourth set, conjunction rules. Because of the treatment of conjunction in the "English Preprocessor Manual"[1] is currently being revised, conjunction has been omitted from this presentation.

26 embeddings and related singularies, including 9 embeddings

15 final singularies

54 rules

## THE MITRE ANALYSIS PROCEDURE

The MITRE analysis procedure takes as input an English sentence and yields as output the set of all basic trees underlying that sentence in the MITRE grammar. If the procedure yields no basic tree, the input sentence is not one generated by the grammar. If the procedure yields more than one basic tree, the input sentence is structurally ambiguous with respect to the grammar.

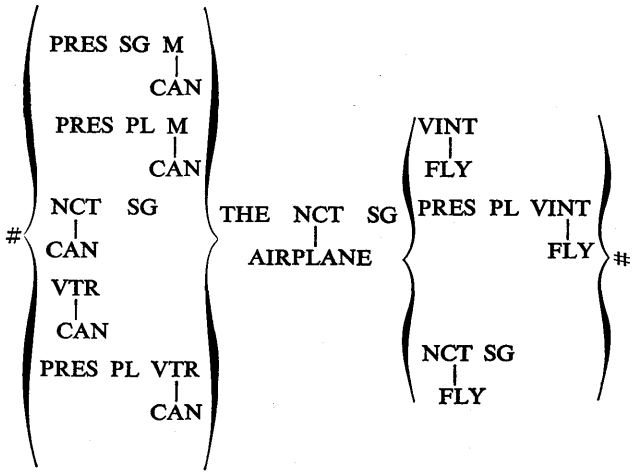
There are five parts to the procedure: lexical look-up, recognition by the surface grammar, reversal of transformational rules, checking of presumable basic trees, and checking by synthesis. These parts are described in detail in the following sections.

*Lexical Look-up*

The first step of the process is the mapping of the input string into a set of *pre-trees,* which are strings of subtrees containing both lexical and grammatical items. The pre-trees are obtained from the input string by the substitution of lexical entries for each word.

A lexical entry for a word may be identical to the word (in the case of grammatical items like A and THE). More often, a lexical entry for a word indicates a representation of the word in terms of more abstract elements (NEG ANY for NONE), a category assignment for the word ( $\frac{\text{ADJ}}{\text{GREEN}}$ for GREEN), or a combination of abstract representation and category assignment ( $\frac{\text{PRES SG VTR}}{\text{OPEN}}$ for OPENS). A word may have several lexical entries.

The number of pre-trees associated with an input string is then the product of the numbers of lexical entries for the words in the string. Thus, the string # CAN THE AIRPLANE FLY # has 15 associated pre-trees, which can be schematically represented as:




Of these 15 pre-trees, only


is a correct assignment of lexical entries to the words in the input string.*

*Recognition by the Surface Grammar*

The surface grammar is an ordered CF phrase-structure grammar containing every expansion which can occur in a surface tree. Unavoidably, the surface grammar generates some trees which are not correct surface trees, even though the corresponding terminal string may be a sentence obtainable by the grammar with some other structure.

In the second step of the analysis procedure the surface grammar is used to construct from each pre-tree a set of *presumable surface trees* associated with the input string. Since the surface grammar is context-free, and context-free parsing algorithms are known to exist, no details will be given here for this step of the analysis.

In the course of recognition by the surface grammar, some pre-trees may be rejected. For example, 9 of the 15 pre-trees in the previous section are rejected in this way. From other pre-trees one or more presumable surface trees will be constructed.

The remaining steps of the analysis procedure are designed to determine, for each presumable surface tree, whether or not the tree is in fact a surface tree for the input sentence.

*Since the MITRE grammar generates neither imperatives nor noun-noun compounds, the interpretation of CAN THE AIRPLANE FLY as analogous to CORRAL THE SADDLE HORSE is excluded.

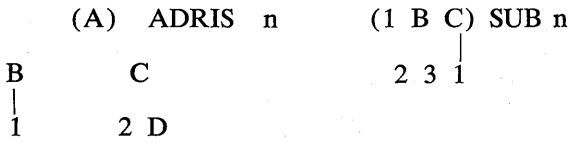
*Reversal of Transformational Rules*

The next step in the analysis procedure reverses the effect of all transformational rules that might have been applied in the generation of the given presumable surface tree.

The "undoing" of the forward rules is achieved by rules that are very much like the forward rules in their form and interpretation. The discussion under Form of the Rules, above, applies to reversal rules as well as to forward rules, with the following additions:

(a) There is a new adjunction instruction, ADRIA (add as right aunt — that is, add as right sister of the parent).

(b) Adjunction and substitution instructions have been generalized to permit instructions like:




Such instructions are used to restore entire subtrees deleted by forward rules.

(c) In the reversal of optional forward rules, a marker OPTN is added as a daughter of a specified node, which in every case is either terminal or else has only a lexical expansion. Some such device is required if the result of the final synthesis step is to correspond to the original input string. The constraint on the placement of OPTN insures that the marker will not interfere with the operation of other reversal rules.

As with forward rules, reversal rules are either cyclical or noncyclical, and either singulary or embedding. All reversal rules are obligatory.*

The reversal rules are grouped together in the same way as the forward rules, and the order of their application within each group is essentially the opposite of the order of the corresponding forward rules. In many cases, one reversal rule undoes one forward rule. There are three types of exceptions, however: (a) several reversal rules may be required to attain the effect of undoing a single forward rule; (b) for some rules, notably the rules with ERASE Ø instructions, no reversal is needed; (c) in some cases the reversing of several forward rules can be combined in whole or in part into a single reversal rule.

Reversed final singularies are first applied to all subtrees. Then reversed embeddings and related singularies are applied in several passes. The first pass deals with the highest sentences in the tree. Later passes move downward through the tree, one level at a time. New lower sentences are created when boundary symbols are inserted during the reversal of embedding transformations; in general, a sentence created on one pass is dealt with on the next. Finally, reversed initial singularies are applied everywhere.

The effect of transformational reversal is to map each presumable *surface* tree into a presumable *basic* tree.†

*Checking of Presumable Basic Trees*

In the next step of the analysis procedure, each presumable basic tree is checked against the phrase-structure component of the (forward) grammar. The check determines whether or not the presumable basic tree can in fact be generated by the phrase-structure component; if it cannot, it is discarded.

Checking by Synthesis. It is possible that transformational reversal and phrase-structure checking could map a presumable surface tree $T_1$ into a basic tree $T_2$ that is not the basic tree underlying $T_1$. For example, the reversal rules map at least one presumable surface tree associated with THOSE PIG IS HUGE into a basic tree underlying THAT PIG IS HUGE. Even under the assumption that input sentences are grammatical, the possibility remains. For example, the reversal rules map at least one presumable surface tree associated with THE TRUCK HAS A SURFACE THAT WATER RUSTS into a basic tree underlying THE TRUCK HAS A SURFACE THAT RUSTS. Similarly, they map at least one presumable surface tree associated with THEY CAN FISH into a basic tree underlying THEY CAN A FISH.

Revision of the present reversal rules and the introduction of rejection rules into the transformational reversal step might make a synthesis step

*Optional reversal rules are required when two distinct basic trees are mapped into identical surface trees by the application of forward rules. No such example occurs in the present MITRE grammar.

†Distinct presumable surface trees may be mapped into identical presumable basic trees; the resultants of distinct presumable surface trees will continue to be processed separately, however.

unnecessary. However, the above examples demonstrate that with the present rules this step is essential.

In the synthesis step, the full set of forward transformational rules is applied to each basic tree that survives the previous checking step. Each optional rule becomes obligatory, with the presence of the marker OPTN (in the appropriate position) as an added condition on its applicability.

The synthesis step maps a basic tree $T_2$, derived from a presumable surface tree $T_1$, into a surface tree $T_3$. If $T_1$ and $T_3$ are not identical, then $T_2$ is discarded as a possible source for the input string. If $T_1$ and $T_3$ are identical, then $T_2$ is a basic tree underlying $T_1$ (and hence, underlying the input string).

*Dimensions*

The dimensions of the additional components of the analysis procedure are as follows:

| | |
|---|---|
| Surface Grammar: | 49 rules |
| | approximately 550 subrules |
| Reversal Rules: | 30 final singularies |
| | 92 embeddings and related singularies |
| | 12 initial singularies |
| | 134 rules |

## AREAS FOR FURTHER INVESTIGATION

We are investigating a number of problems both in the grammar and in the analysis procedure, with the objectives of making the grammar more adequate and the procedure more efficient.

Among the grammatical problems are the use of syntactic features (see Chomsky[3]) and the addition of further rejection rules in the transformational component. The treatment of conjunction is being revised. Other topics requiring investigation include adverbial clauses, superlatives, verbal complements, imperatives, and nominalizations.

We are examining a number of ways to improve the efficiency of the analysis procedure. If the input vocabulary is to be of an appreciable size, an efficient and sophisticated lexical look-up routine will be required. We are using computer experiments to determine the extent to which the use of a CS surface grammar, either as the basis of a CS parsing routine or as a check on the results of CF parsing would improve the procedure by eliminating some incorrect surface trees at an early stage.

Some increase in the efficiency of the reversal step might be achieved by making use of a preprogrammed path through the reversal rules, or by using information that certain surface grammar rules signal the applicability or inapplicability of certain reversal rules. Similarly, the efficiency of the final synthesis step might be improved by making use of a preprogrammed path through the forward transformational rules, or by using information that certain reversal rules have been applied.

*Analysis by Synthesis*

The first analysis procedure proposed for transformational grammars was the "analysis by synthesis" model of Matthews.[9] Basically this procedure involves generating sentences until one is found which matches the input sentence; the steps used in the generation provide the structural description. No attempt to program the analysis-by-synthesis procedure for transformational grammars has been reported in the literature. In its raw form this procedure would take an astronomically long time. One way to refine the procedure would be to use a "preliminary analysis" of some sort, which would have to be extensive to make any appreciable change in efficiency. As a result, there may be no sharp boundary between refined analysis-by-synthesis and direct analysis with a final checking-by-synthesis step. In the case of the MITRE procedure the final synthesis step plays a relatively minor role in the total procedure.

*Petrick's Procedure*

S. R. Petrick[10] has proposed and programmed a general solution to the analysis problem which is similar in many respects to the MITRE procedure. One of the main differences between his approach and ours is that he alternates the use of reversal rules and phrase-structure rules, while we use first the phrase-structure rules of the surface grammar and then the reversal rules. Furthermore, while Petrick's reversal rules are all optional, ours are all obligatory. It follows that although we may have a larger number of structures to consider at the beginning of reversal step, this number does not increase as it does at every step in Petrick's procedure.

At the present time the procedures differ in gen-

erality, for Petrick has shown that there are algorithms for the construction of his surface grammar and reversal rules. In the case of the MITRE procedure, the question of the existence of comparable algorithms has not yet been resolved.

*Kuno's Procedure*

Another approach to the analysis problem has been proposed in Kuno.[11] Kuno attempts to find basic trees, without using reversal rules, by constructing a context-free surface grammar and associating with each of its rules information about the form of the basic tree.

Kuno reported that an experimental program for this system had been written and was being tested on a small grammar. At that time it was not known whether an algorithm for constructing the required phrase-structure grammar existed.

## COMPUTER TESTS

To test the grammar and the procedure a set of FORTRAN subroutines (called SYNN), designed to be combined in several different programs, has been written. In one order, the subroutines carry out the procedure from the stage at which presumable surface trees have been obtained, through the base tree, to the final step of comparison of the derived surface tree with the given presumable surface tree. In other orders they can, for example, convert base trees to surface trees and back, or check surface trees against context-sensitive grammars.

We describe first the subroutines, in groups corresponding to the major components of the MITRE procedures, then some of the programs and the results of running the programs on a subset of the grammar.

*Subroutines*

Because the primary operations are operations on trees, the main subroutines of the SYNN package analyze and manipulate trees. Three of the subroutines treat trees without reference to the grammar: CONTRE reads in a tree and converts it to the internal format, TRCPY stores a copy of a tree for later comparison, and TREQ compares two trees to see if they are identical.

In the SYNN package there are four subroutines that deal with phrase-structure grammars. CONCSG and CONCFG read in context-sensitive and context-free grammars, respectively, and convert them to internal format. CHQCS and CHQCF check the current tree against the indicated grammar by a regeneration procedure.

Most of the subroutines of SYNN are concerned with the transformational components. Separate subroutines read in the transformational rules, control the application cycle, mark levels of embedded subtrees, search for an analysis, check restrictions, and perform the operations.

The application of the forward rules is controlled by the subroutine APPFX, and the application of the reversal rules by APPBX. The application cycles are as described in the section Reversal of Transformational Rules, above, except that each transformational rule has a keyword which is used to bypass the search if the transformational keyword does not occur in the tree.

There is also a generation subroutine GENSR which is best described as a "constrained-random" generator. Within constraints specified by the user the subroutine generates a pseudo-random base tree to which other subroutines of SYNN can be applied.

*Programs*

In initial tests of the grammar and procedure the most useful combination of subroutines was in the program SYN1, which goes from basic tree to surface tree and back to basic tree, checking at every step. This first program is an iteration of the subroutines CONTRE, TRCPY, CHQCS, APPFX, CHQCF, APPBX, CHQCS, TREQ. When all parts are correct, the final result is the same as the input, and this is indicated by the final comment of the TREQ subroutine.

The program SYN2 carries out the steps of the MITRE procedure without the first two steps, lexical look-up and context-free parsing. Its basic cycle is CONTRE, TRCPY, APPBX, CHQCS, APPFX, CHQCF, TREQ. After each of the subroutines an indicator is checked to see if the tree should be rejected.

The program SYN3, which uses the generation subroutine, is like SYN2 except that GENSR replaces CONTRE in the basic cycle. Inputs for GENSR are easier to prepare than those of

CONTRE, so that SYN3 is being used extensively in debugging the grammar.

The lexical lock-up and context-free parsing steps of the procedure have not been programmed. Because algorithms for these steps are known to exist, it was decided that their programming could be postponed and an existing program used.

*Test Grammar*

A subset of the grammar, familiarly known as the JUNIOR grammar, was selected for initial tests of the procedure. Its dimensions are:

| *(Forward) Grammar* | |
|---|---|
| Phrase-Structure | 61 rules |
| Component: | 105 subrules |
| Transformational | 11 initial singularies |
| Component: | 6 embeddings and related singularies, including two embeddings |
| | 3 final singularies |
| | 20 rules |
| *Surface Grammar* | 32 rules |
| | 306 subrules |
| *Reversal Rules* | 6 final singularies |
| | 15 embeddings and related singularies |
| | 11 initial singularies |
| | 32 rules |

Twenty-six sentences (plus some variants) constitute a basic test sample for the JUNIOR grammar. This sample, which includes at least one test for each transformational rule, contains (among others) the sentences:

1. The airplane has landed.
2. Amphibious airplanes can land in water.
3. Did the truck deliver fifty doughnuts at nine hundred hours?
4. Were seven linguists trained by a young programmer for three months?
5. The general that Johnson met in Washington had traveled eight thousand miles.
6. Are there unicorns?
7. John met the man that married Susan in Vienna.
8. There were seven young linguists at MITRE for three months.
9. Can all of the ambiguous sentences be analyzed by the program?
10. The linguist the ambiguous grammar was written by is young.

SYN1 has been run on the full set of sample sentences. The total time for a run with 28 sentences was 5.11 minutes on the 7030 computer.

SYN3 has likewise been run with the JUNIOR grammar. As an example of running time, a typical run generating 20 trees carried all of them through the transformations and reversal rules in a total of 5 minutes. All but one of these trees contained embedded sentences; half of them contained two embeddings.

In another experiment, a CF parser was used with SYN2 to simulate the full procedure. The results for sentences (1), (2), and (6) are:

| | Sentence | | |
|---|---|---|---|
| | (1) | (2) | (6) |
| Pre-trees | 12 | 90 | 1 |
| Presumable surface trees | 8 | 15 | 1 |
| Presumable base trees | 3 | 4 | 1 |
| Correct base trees | 1 | 2 | 1 |

In the worst case encountered, sentence (5), there are 48 presumable surface trees.

It is clear from even these few numbers that if the procedure is to be practical, it will be necessary to incorporate a highly efficient routine for obtaining surface trees and to work on the rapid elimination of spurious ones.

## REFERENCES

1. "English Preprocessor Manual," SR-132, MITRE Corp. 1964, rev. 1965.

2. N. Chomsky, *Syntactic Structures,* Mouton, The Hague, 1957.

3. ———, *Aspects of the Theory of Syntax,* M.I.T. Press, Cambridge, Mass., 1965.

4. ———, "Formal Properties of Grammars," *Handbook of Mathematical Psychology,* R. D. Luce, R. R. Bush and E. Galanter, eds., Wiley, New York, 1963, vol. 2, pp. 323-418.

5. ——— and G. A. Miller, "Introduction to the Formal Analysis of Natural Languages," ibid., pp. 269-321.

6. ——— and ———, "Finitary Models of Language Users," ibid., pp. 419-491.

7. T. V. Griffiths and S. R. Petrick, "On the Relative Efficiencies of Context-Free Grammar Recognizers," *Comm. ACM,* vol. 8, pp. 289-300 (1965).

8. T. V. Griffiths, "Turing machine recognizers for general rewriting systems. *IEEE Symp. Switching Circuit Theory and Logical Design.* Princeton, N. J., November, 1965, pp. 47-56.

9. G. H. Matthews, "Analysis by Synthesis of Sentences of Natural Languages," *1961 International Conference on Machine Translation and Applied Language Analysis,* HM Stationery Office, London, 1962, vol. 2, pp. 531-540.

10. S. R. Petrick, "A Recognition Procedure for Transformational Grammars," Ph.D. thesis, M.I.T., 1965.

11. S. Kuno, "A System for Transformational Analysis," paper presented at the 1965 International Conference on Computational Linguistics, New York City, May 20, 1965.

# COBWEB CELLULAR ARRAYS*

Robert C. Minnick
*Stanford Research Institute*
*Menlo Park, California*

## INTRODUCTION

The *cobweb* cellular arrays are embellishments of the cutpoint[1-3] cellular array that are made by complicating the cell-interconnection structure. This new class of arrays will allow for more economical and efficient logical designs than are possible in cutpoint arrays. As a background to the new arrays, the properties of the cutpoint array[2] will be reviewed.

## CUTPOINT ARRAY

The cutpoint cellular *array* is a two-dimensional rectangular arrangement of *cells*. As shown in Fig. 1*b*, each cell has binary inputs from neighboring cells on the top and the left, and binary outputs to neighboring cells on the bottom and right. In addition to being used in the cell, the input to the left of each cell is bussed to the right output. The bottom output of each cell is set as one of six combinational switching functions of the two cell inputs, or as an R-S flip-flop—in either case by the use of four specification bits, or *cutpoints,* in each cell. By specifying these cutpoints independently for every cell of a cutpoint cellular array, the array is thereby particularized to a required logical property. The table in Fig. 1 *a* includes the logical functions that can be produced at the bottom output of each cell, depending on the particular specification of its cutpoints. The symbol F in the index column indicates an R-S flip-flop.

A 3 × 4 array of cutpoint cells is shown in Fig. 1*b*; the specification bits are indicated as dots. A realization for one cutpoint cell in terms of diode-transistor circuits is shown in Fig. 2. The four cutpoints in this realization are depicted as switches; however, they could be photoresistors, flip-flops, or breaks or bridges in conductors. The DTL realization in Fig. 2 is one of many circuit possibilities for a cutpoint cell.

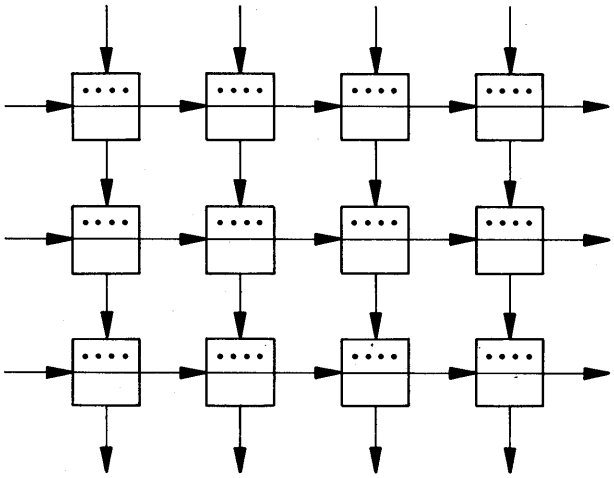
## PROBLEMS WITH CUTPOINT ARRAYS

It has been shown in the previously cited references that arbitrary logical functions can be realized using appropriately specialized cutpoint arrays. However, certain of these realizations tend to be inefficient in terms of the number of required cells. For instance, the best-known realization for a three-bit parallel adder using no more than two cutpoint arrays is shown as Fig. 3. In this figure the two three-bit words ($a_3$, $a_2$, $a_1$) and ($b_3$, $b_2$, $b_1$) are added to form

*The research reported in this paper was sponsored by the Air Force Cambridge Research Laboratories, Office of Aerospace Research, Under Contract AF 19(628)-4233.

| INDEX | a | b | c | d | z |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | y' |
| 2 | 0 | 0 | 1 | 0 | x'+y' |
| 3 | 0 | 0 | 1 | 1 | x'y' |
| 4 | 0 | 1 | 0 | 0 | x+y |
| 5 | 0 | 1 | 0 | 1 | xy' |
| 6 | 0 | 1 | 1 | 0 | x⊕y |
| 7 | 0 | 1 | 1 | 1 | 0 |
| F | 1 | 1 | 0 | 1 | x=S, y=R |

(a)



(b)

RA-641541-82

Figure 1. Cutpoint Array.

the sum word ($s_3$, $s_2$, $s_1$). The input carry to the low-order column is $c_0$, while $c_3$ is the overflow bit. An n-bit parallel adder can be formed in a similar manner to the one in Fig. 3; a total of $(2n + 1)^2$ cells are required in two adjoining arrays for such a realization.

Reference back to Fig. 1*a* shows that cells with an index 1 form the complement of the top input. This one-variable function is convenient to use when transmittal of information vertically in an array is desired. Hence, vertical cascades of Index 1 cells in a cutpoint array indicate that in effect, no logic is being performed, perhaps with the exception of one cell in each such cascade. With this in mind, it is now observed that in the upper 3 × 7 array of Fig. 3 only six cells, roughly along the diagonal from the upper-right to lower-left corners, are used logically. Similarly, in the lower 4 × 7 array in Fig. 3, only cells in the upper-right triangular area are used logically.

A wastage of cells similar to that encountered in Fig. 3 had been observed in several cutpoint-array logical designs, particularly in designs which involve parallel operations. This inefficient use of cells occurs in most of these situations because every bit of one operand word must interact with every bit of a second operand word. In a cutpoint array the only convenient way this interaction can occur is to introduce the bits of one word on the side of the array and the bits of the other word along the top. This orthogonal introduction of the two operand words into a cutpoint array seems necessary because no facility is provided within the array to change the direction of information flow from vertical to horizontal.

It it were possible to redirect the information

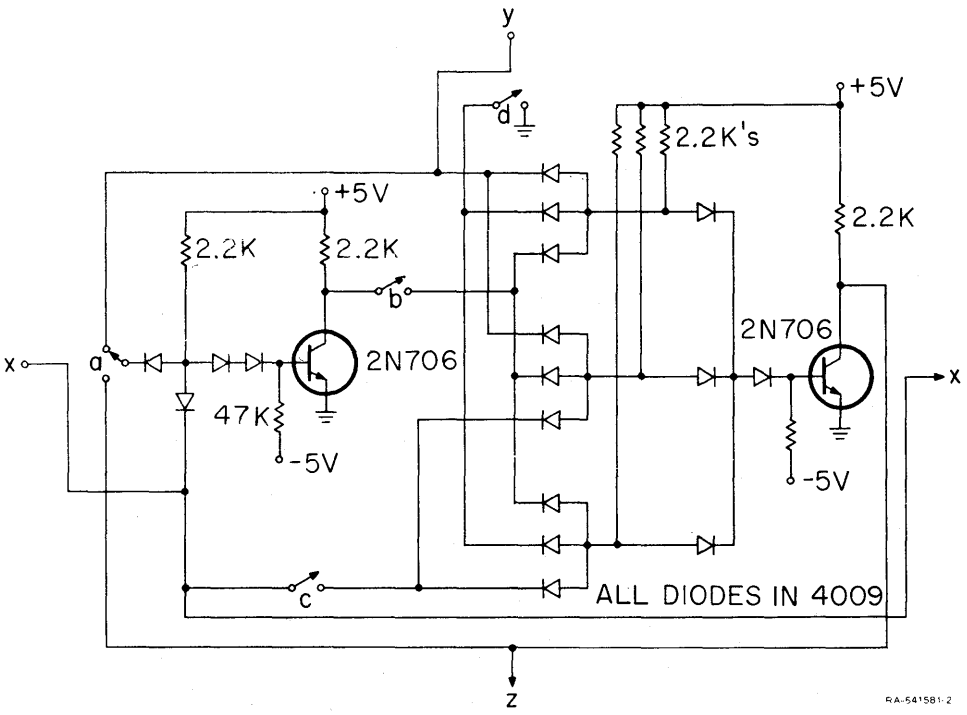



Figure 2. Circuit for one Cutpoint Cell.

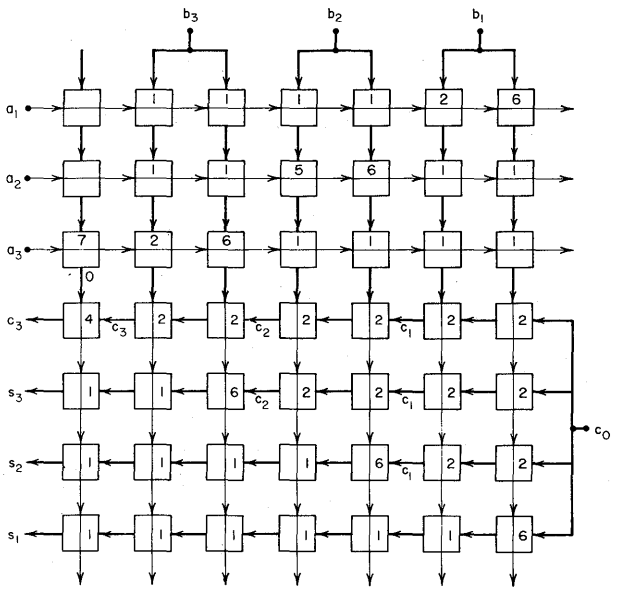



Figure 3. Cutpoint realization for a three-bit parallel adder.

flow inside a cellular array, two n-bit operand words might be applied to the side of the array, (or both to the top), and possibly a significant reduction would result in the number of cells that are required. Instead of requiring $0(n^2)$ cells, the resulting array might require $0(n)$ cells. Thus this lack of control on the direction of information flow constitutes an important problem in the use of cutpoint arrays.

Another problem encountered in practical logical designs using cutpoint arrays is the excessive requirement for jumper connections from edge-output points to edge-input points of the same array. One example of this problem is shown in Fig. 4, which is a five-bit shift register driven by a four-phase clock. For this example four such jumpers are used. Jumpers of this type will be termed *edge jumpers*. A second example is shown in Fig. 5. In this figure, the following three combinational functions of the three variables, $x_3$, $x_2$, $x_1$, are realized in one cutpoint array:

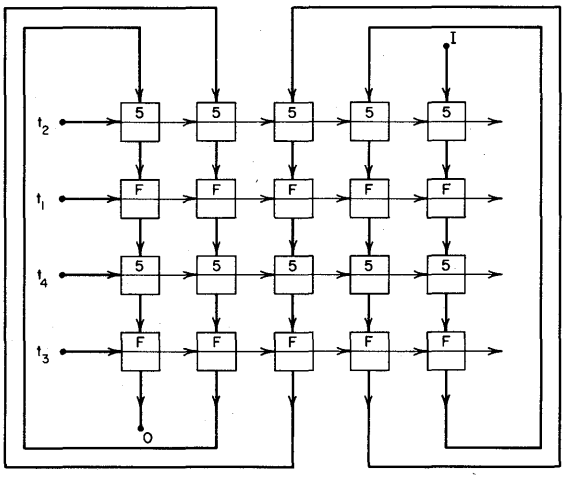



Figure 4. Cutpoint realization for a five-bit shift register.

$$E_{66} = \Sigma(1, 6)$$
$$E_{43} = \Sigma(0, 1, 3, 5) \quad (1)$$
$$E_{129} = \Sigma(0, 7).$$

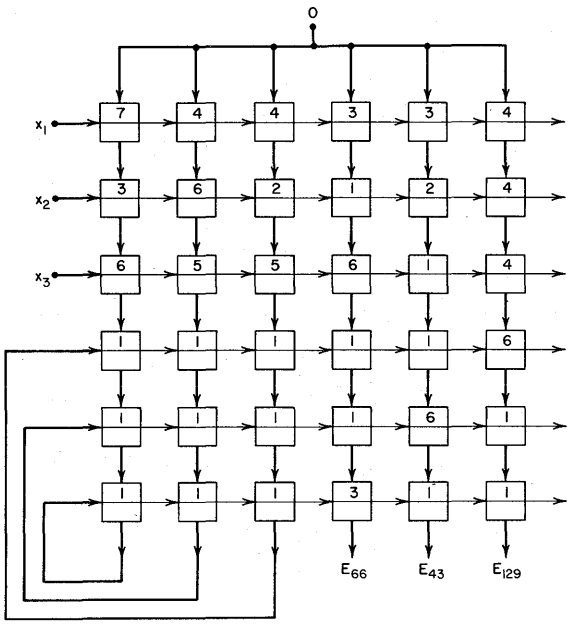



Figure 5. Cutpoint realization for three functions of threevariables.

A requirement for edge-jumper connections in a cutpoint array often carries with it a wastage of cells. In the bottom half of the array in Fig. 5, for example, only three cells are used other than for transmitting signals.

A third problem often encountered when practical logical designs are made in terms of cutpoint arrays is an insufficient number of edge connections to the array. A final problem is the desirability to have the cells isolated from one another during the early part of the production so that it is possible to identify faulty cells by step-and-repeat testing.

The cobweb array is proposed as a means of meeting all of these problems of inefficiency for parallel operations, excessive edge-jumping, insufficient edge connections, and lack of cell isolation.

## COBWEB ARRAY

A 4 × 4 cobwed array is shown in Fig. 6. It is seen that within the array each cell has five possible inputs: two from a horizontal and vertical buss and three from nearby cells. Connections from edge cells to the package terminals are shown on Fig. 6 by peripheral dots. For terminal connections, each cell on the left and bottom edges of the array has one non-bus output connected to terminals, each cell on the right and top edges of the array has one non-bus input connected to terminals, and each horizontal and vertical bus is connected to a terminal. For an M × N-cell cobweb array it is easily seen that $3(M + N) - 2$ package terminals are needed. This compares with M + 2N terminals for the cutpoint array of the same size; for square arrays approximately twice the number of terminals are provided by the cobweb array, while in general the number of terminals in the cobweb array varies from one and one-half to three times the number in cutpoint arrays of the same dimensions.

In Fig. 7*a* one internal cell of this cobweb array is shown with its five inputs labelled as u, v, w, x, and y Fig. 7*b* shows how this cobweb cell can be fabricated from the previous cutpoint cell and fourteen additional cutpoints. Of course, as mentioned previously, technologies other than the diode-transistor method shown in Fig. 7*b* can be used. The added cutpoints are labelled e, f, . . . , r. In order to have all single-throw cutpoints, the double-throw cutpoint a in Fig. 2 is replaced in Fig. 7*b* by two cutpoints, g and e, where $a = g'e$ and $a' = ge'$.

It is anticipated that cobweb cellular arrays will be made by one of the modern batch-fabrication technologies, such as that of integrated circuits. In making these arrays with integrated circuits, the number of deposition steps is of related economic interest. Returning to Fig. 6, and using the nomen-

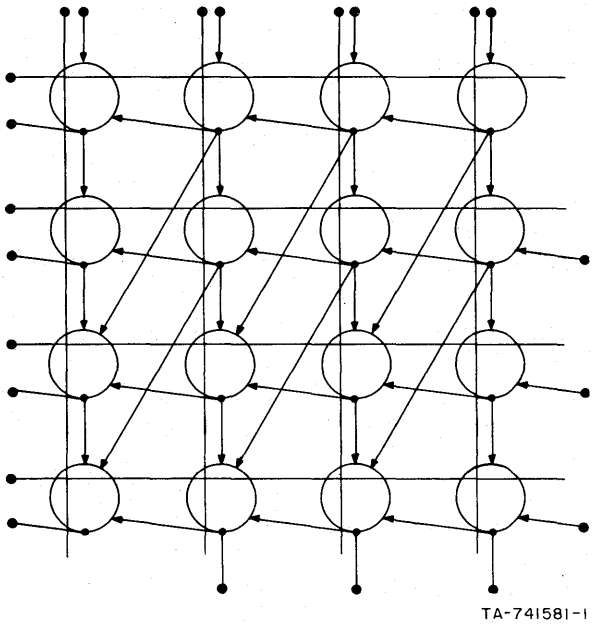



Figure 6. Structure of the cobweb array.

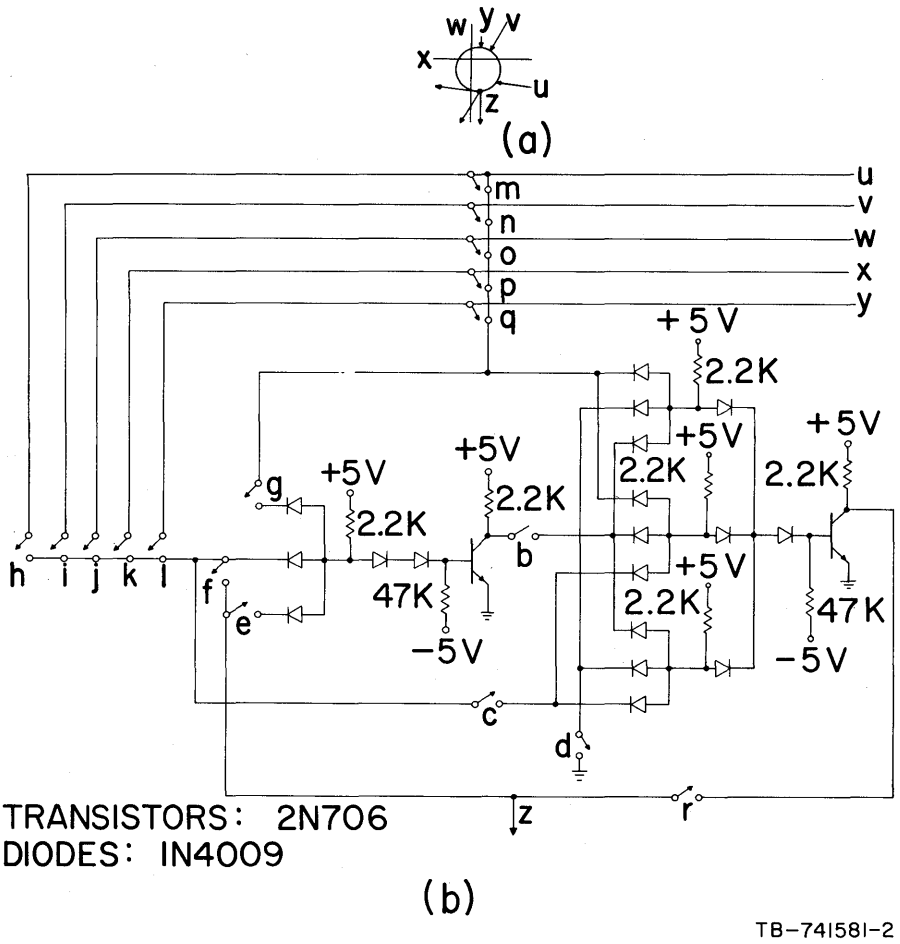



Figure 7. Diode-transistor realization of cobwell cell.

clature of Fig. 7 *a* , it is seen that if the w busses are moved to the right of the center in each cell, all w, v and y interconnections may be deposited simultaneously. After depositing an appropriate insulating layer, the u and x interconnections together with connections for power and ground may be formed as a second deposition layer. Hence the interconnection structure of the cobweb array in Fig. 6 is *two-layered*. Similar reasoning applied to Fig. 1*b* shows that the interconnection structure of the cutpoint array is single layered.

In summary, the cobweb array consists of cells that have the same amount of electronics as cutpoint cells. Each cell in the new array has about four times the number of cutpoints as the cutpoint cell, one and one-half to three times the number of package terminals as a cutpoint array of the same size, and a two-layered rather than a one-layered interconnection structure. It will now be shown that the use of this more complicated cellular array at least partially alleviates the previously-discussed problems of cutpoint cellular arrays.

## LOGICAL DESIGN WITH COBWEB CELLULAR ARRAYS

In cutpoint arrays, switching functions are produced by forming one or more vertical cascades of cells. In the cobweb cellular arrays, these cascades of cells no longer are required to be vertical. Indeed, a *cascade* in a cobweb array may be any chain of cells that follows the arrowheads in Fig. 6. This property of cobweb arrays gives the logical designer a considerable degree of flexibility in forming his design. The need for an increased ratio of edge connections to cells is met in cobweb arrays. By introducing other assumptions on edge connections it is possible further to increase this ratio if additional logical design experience shows this to be desirable.

In the cobweb array it is possible to use some of the cutpoints in a cell in lieu of edge jumpers. For instance, if cutpoints h and k (Fig. 7 *b* ) are closed, this causes the x bus and input u to that cell to be connected together. Similarly, by closing cutpoints j and f, and by opening cutpoint r, the cell output can be jumpered to the w input bus; for this connection, the logical function produced by the cell is immaterial.* It is also possible to jumper as many as all five inputs and the output of a cell together. Indeed, for those cases where sneak paths are not introduced, it is possible to form one jumper path among the cutpoints f, h, i, j, k, l and a second nne among m, n, o, p, q. Cells that are specialized in this way are called *jumper cells*. The jumper connections are designated by circling the inputs (and output) that are jumpered together and by inserting the symbol J inside the cell. If two isolated jumpers are used, triangles will designate the inputs (and output) on the second jumper.

It should be observed that J cells in the cobweb array are logically inactive. That is, jumper cells are used only to make local connections in the array, and not to perform logical operations. It should also be noted that jumper-cell connections can be made in such a way as to allow information flow in violation of the arrowheads in Fig. 6.

It also is possible to use the cutpoints h, i, . . . , q to obtain an OR of two or more of the five inputs to a cell. For instance if cutpoints k, l, m, n and o are closed, then the horizontal input to the cell is x + y, while the vertical input is u + v + w. Care must be taken when using this property to avoid the introduction of sneak paths.

Two decomposition methods will now be shown in order to illustrate the elimination or reduction in the use of edge jumpers by the jumper-cell specialization. It is supposed that a switching function $E = E(x_1, x_2, \ldots, x_n)$ is not producible in one cascade of cells. This function can always be decomposed on one of its variables, $x_i$, in several ways, including a form due to Shannon,

$$E = Gx_i + Hx_i' \tag{2}$$

and a form due to Reed,

$$E = Ax_i \,\theta\, B \tag{3}$$

where $\theta$ is the EXCLUSIVE-OR operator, and each of G, H, A and B is a switching function of no more than n-1 variables, and is independent of $x_i$.

If it is assumed that G′, H′, A′, and B are each producible in one cascade of cells, the cobweb arrays of Figs. 8*a* and 8*b* correspond to the two decompositions of Eqs. (2) and (3), respectively. If one or more of the four (n – 1)-variable functions are not producible in a single cascade, either of the above decompositions may be repeatedly applied until all subsidiary functions are realizable in one cascade.

The J cell in Fig. 8*a* with the y input and the z output circled means that the switches f and 1 (see Fig. 7) are closed and that switch r is open. This connects the y input to the cell output without the use of an external jumper. For cobweb cells that produce one of the functions listed in Fig. 1*a*, a function index is placed inside the cell and the particular inputs (if any) that are connected through the cutpoints h, i, . . . , 1, (Fig. 7), are designated by *circles,* while the particular inputs (if any) that are conected through the cutpoints m, n, . . . , q are designated by *triangles*. For instance, the cell with index 5 in Fig. 8*a* means (b, c, d) = (1, 0, 1), and since a circle is on the x buss and a triangle is on the y input, then (h, i, j, k, l) = (0, 0, 0, 1, 0), and (m, n, o, p, q) = (0, 0, 0, 0, 1), and finally (e, f, g, r) = (0, 0, 1, 1).

As shown in Fig. 8, no jumpers are needed in the cobweb-array realization of either decomposition. Furthermore, it is noted that only one row of cells is needed for each application of the Reed decomposition.

With none of the cutpoints h, i, . . . , q closed, all cells are isolated in the cobweb arrays. Therefore, step-and-repeat testing is possible for cobweb arrays that are fabricated as monolithic integrated circuits, while it is not possible for the originally proposed cutpoint arrays.

The particular interconnection structure of the cobweb array was chosen for several reasons. As the number of potential inputs to each cell is increased, the number of interconnection possibilities also increases. But this increase is obtained at the cost of additional cutpoints in each cell. Hence, it is desirable to introduce only as much interconnection versatility as the typical logical designer would use.

*In principle, cutpoint r in Fig. 7b could be eliminated by setting cutpoints b, c, and d so that the output transistor is nonconducting. However, cutpoint r is necessary for the correction algorithms to be described.

*In order to simplify the artwork, the terminal conventions adopted in connection with the discussion of Fig. 6 will not be explicitly shown on this and on following cobweb-array designs.

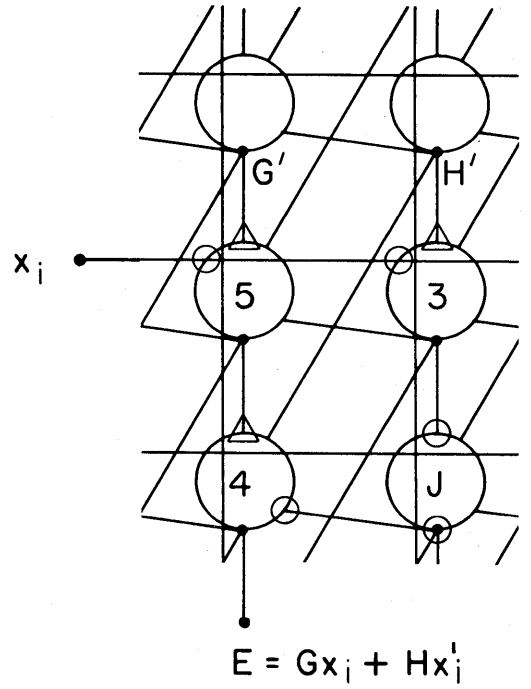



Figure 8. Shannon and reed decompositions using cob-web arrays.

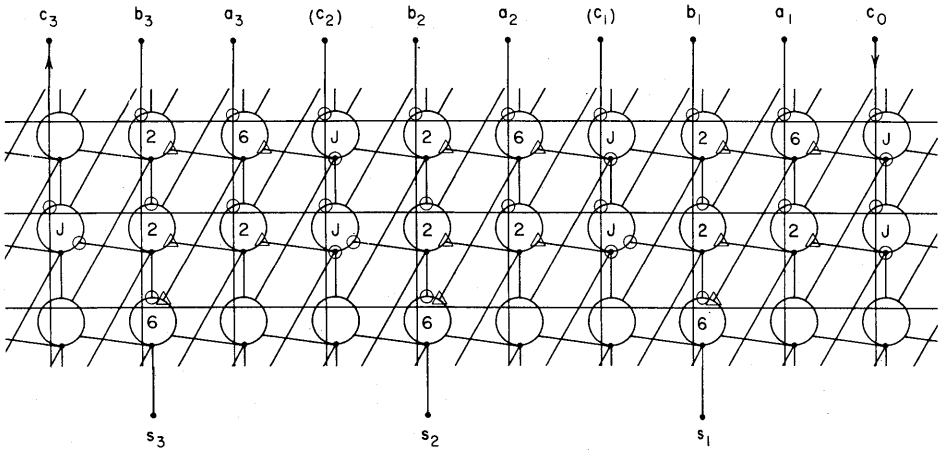



Figure 9. Cobweb realization for a three-bit parallel adder.

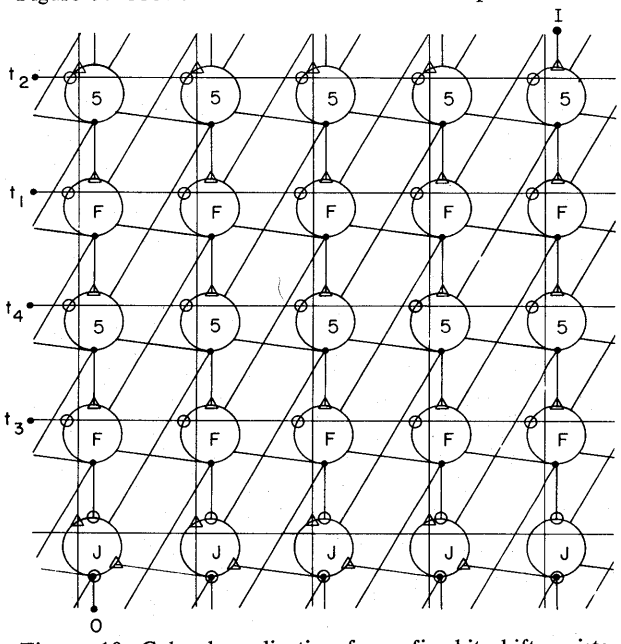



Figure 10. Cobweb realization for a five-bit shift register.

Referring back to Fig. 7a, the x and y inputs are carried over directly from the previous cutpoint array. The u input allows the designer to build up a carry propagation chain within a horizontal row of register cells. The vertical bus allows one to jumper a bottom-cell output of an array to a top-cell input. Finally, the v input is a knight's move away so that it is possible to build up a cutpoint cascade that crosses other such cascades. The desirability of having crossings in cellular arrays has been observed before.*

A number of obvious variations of the cobweb array is possible. For instance input v, or both inputs v and w in Fig. 7a may be omitted in each cell,with a corresponding saving in cutpoints. In the latter variation, a single-layered interconnection structur results. Similarly, it is possible to invent more complicated variations of the cobweb array.

Illustrations of logical designs using cobweb arrays are given as Figs. 9, 10 and 11. These figures should be compared directly with Figs. 3, 4, and 5, respectively. First comparing Figs. 3 and 9, it is seen that an n-bit parallel adder can be synthesized using $9n + 3$ cobweb cells in a single array, while $(2n + 1)^2$ cells in two adjoining arrays are required if cutpoint cellular logic is used. Thus, for example, a 50-bit parallel adder requires 453 cells in a cob-

*By Marvin E. Brooking, private communication.

web array and 10,201 cells in two cutpoint arrays.

A comparison of Figs. 4 and 10 shows that all edge jumpers are eliminated in the cobweb realization of a shift register at the cost of one extra row of cells. Finally, comparing Figs. 5 and 11, it is seen that the three edge jumpers as well as half the total number of cells are eliminated when a cobweb array is substituted for a cutpoint array.

## FAULT AVOIDANCE METHODS

In regard to the cutpoint cellular array, methods have been demonstrated for replacing faulty cells with spare cells.[2] These methods no longer are feasible with the cobweb arrays; therefore, it is necessary now to develop an alternative faculty cell avoidance algorithm. It will be assumed that the faults are "electronic;" that is, a transistor has a low beta, or it has an emitter-collector short, or a diode is open-circuited, etc. All conductors and cutpoints will be considered perfect, and furthermore, the circuit design is assumed to be such that no failure condition will cause the shorting or opening of a conductor or the shorting of a power supply. It appears from a consideration of the integrated circuit technology that these assumptions are realistic.

Two clusters of cells called *supercells* are defined by Fig. 12. The shaded cell in each 2 × 2 cobweb array has five inputs (marked with the symbol I) that are geometrically equivalent to the cobweb cell of Fig. 7. The jumpers between points p and in Fig. 12 are used for transmitting the knight's move interconnections. The supercells of Fig. 12 are arranged in such a way that one may first perform a logical design in terms of a conventional cobweb array, and then replace each cell in the first and all odd-numbered rows with a type $\alpha$ supercell. The cells in the second and all even-numbered rows are replaced with a type $\beta$ supercell.

The effect so far has been to increase the number of cells in the cobweb array by a factor of four. In this supercell array it is possible under most conditions to make focal perturbations of the logical de-

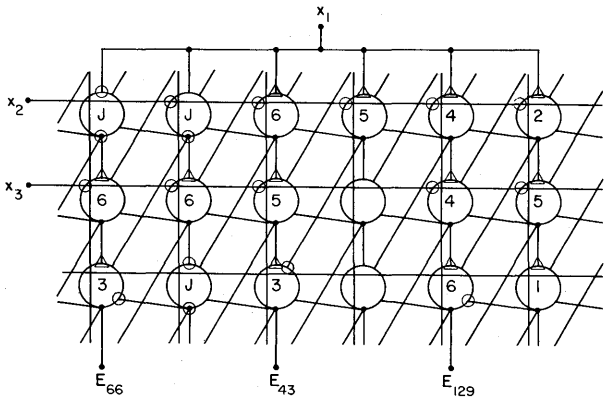



Figure 11. Cobweb realization for three functions of threevariables.

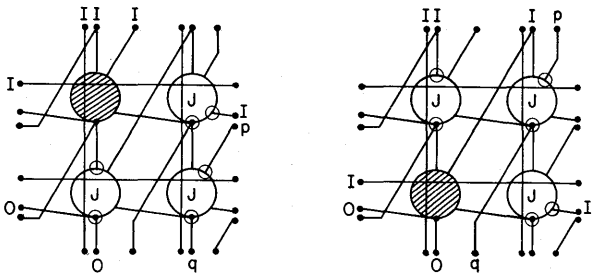



Figure 12. Cobweb supercells.

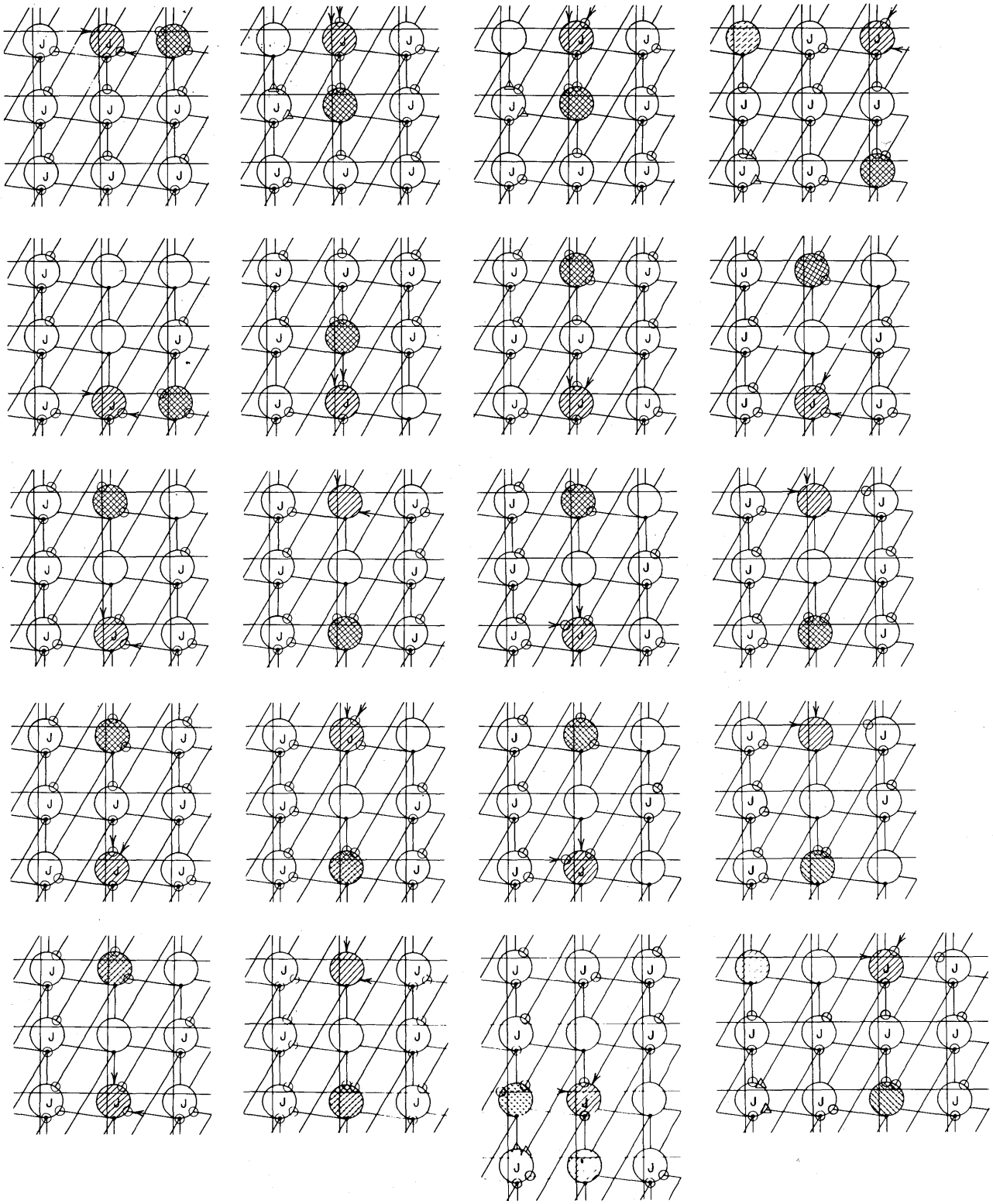
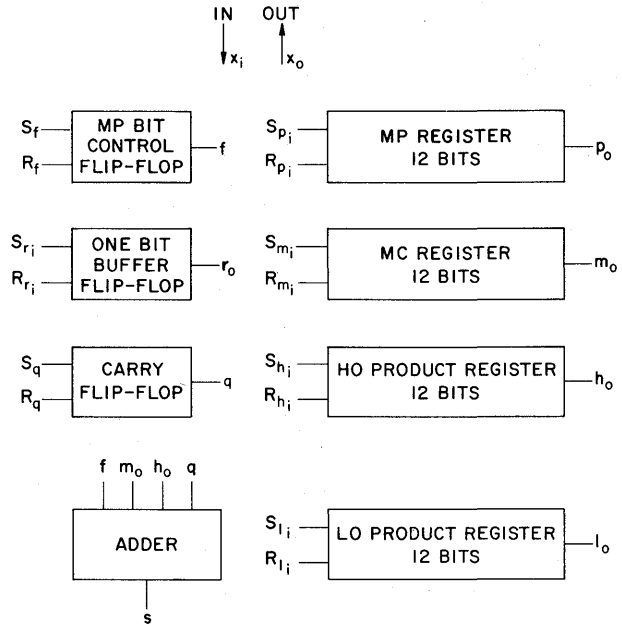


Figure 13. Exhaustive listing of the cobweb array fault-avoidance algorithm.

sign in order to avoid faulty cells. Assuming that only one or two of the five inputs to a given cell are connected by means of cutpoints h, i, . . . , q in Fig. 7, it is necessary to demonstrate a fault-avoidance algorithm for $C_2^5 = 10$ cases (the two-input cases cover the one-input cases). However, the logical cells in a supercell array appear in two geometrically different environments that correspond to the types $\alpha$ and $\beta$ supercells; therefore, a total of 20 cases must be investigated. Proceeding by exhaustion, a fault-avoidance algorithm for each of these 20 cases is shown as Fig. 13. On this figure, a single shading indicates a faulty cell; arrowheads are attached to the two active inputs for that cell. A cell with cross-shading is assumed to be a good cell, and it replaces the faulty cell. If the faulty cell has no symbolism other than the arrowheads and the shading, it is assumed to have been disconnected by having all of cutpoints f, h, i, . . . , r open; if it has a J symbol, it is used as a jumper cell with no connections made at the arrowheads. Cells with a dotted single shading are neighboring good logical cells. Cases where the faulty cell in Fig. 13 occurs on or near the top row correspond to faults in $\alpha$ supercells, while cases where the faulty cell in Fig. 13 occurs on or near the bottom row correspond to faults in $\beta$ supercells.

From this development it should be clear that if a logical cell in a supercell array is bad, it can be logically replaced provided that another cell is good




$$R_f = p_o' T t_1 \qquad S_f = p_o T t_1$$

$$R_{p_i} = (\beta x_i)' t_2 \qquad S_{p_i} = t_1$$

$$R_{r_i} = (\gamma m_o)' t_2 \qquad S_{r_i} = t_1$$

$$R_{m_i} = (r_o + \alpha\beta p_o)' t_2 \qquad S_{m_i} = t_1$$

$$R_q = f m_o T' h_o t_2 \qquad S_q = [(f m_o + T' h_o)' + \alpha'\beta] t_2$$

$$R_{h_i} = (s\gamma + l_o \alpha\beta') t_3 \qquad S_{h_i} = t_2$$

$$R_{l_i} = (l_o \gamma + T h_o \gamma)' t_3 \qquad S_{l_i} = t_2$$

$$s = f m_o \oplus T' h_o \oplus q$$

$$x_o = \alpha\beta' t_3 h_o$$

TB-5087-7

Figure 14. Block diagram for a twelve-bit serial multiplier.

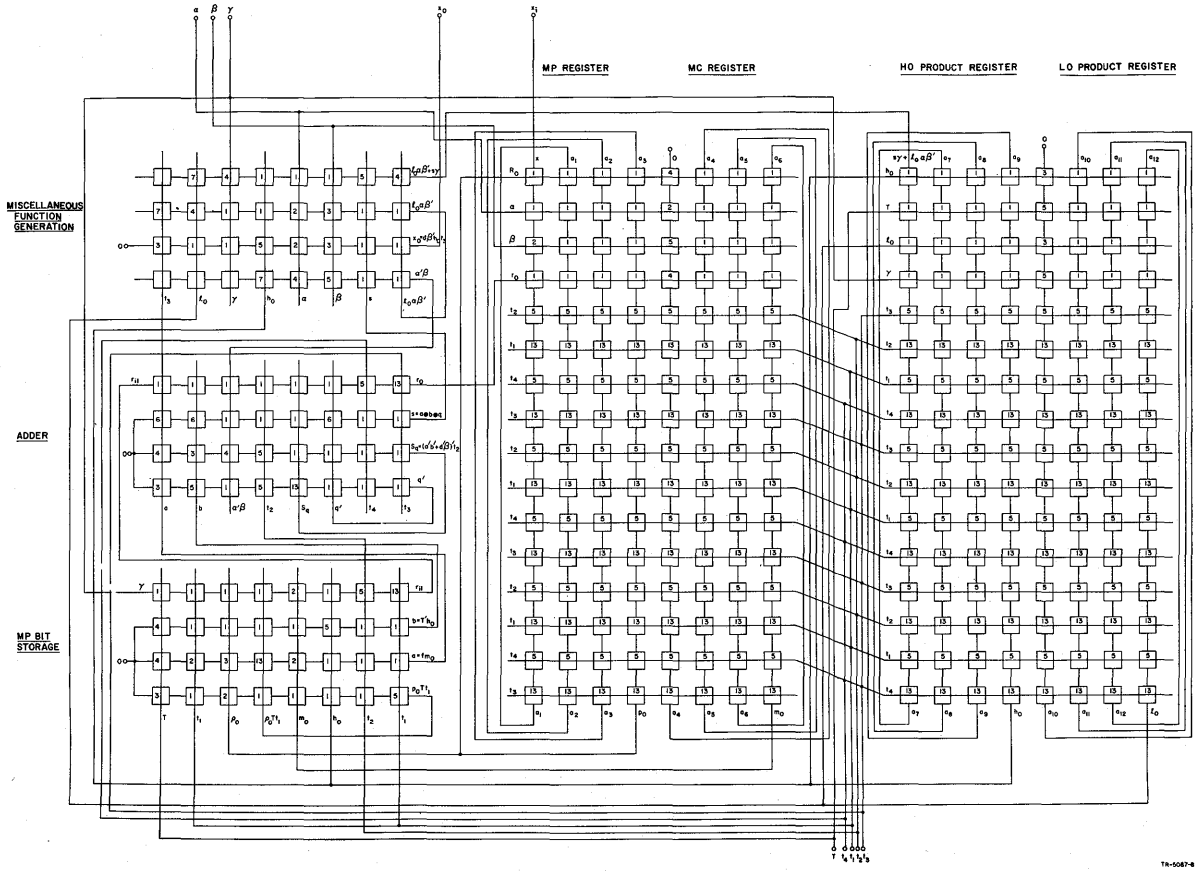



Figure 15. Realization of the multiplier in terms of fivecutpoint arrays.

at a distance one or two cells from it, according to Fig. 13.

Thus a fault-avoidance algorithm for cobweb arrays has been demonstrated. Many variations in the process are possible. For instance, if multiple faults prevent complete avoidance of faulty cells using the $2 \times 2$ supercells, one can replace some or perhaps all cells in the supercell array again with supercells until enough redundancy has been obtained that all faults can be avoided. Similarly, it may be possible to compress a supercell array if, for instance, no corrections are necessary in a particular row or column. Similar fault-avoidance algorithms can be deduced for the simplified cobweb arrays mentioned before.

## LOGICAL DESIGN OF A MULTIPLIER

As a final illustration, a logical design is given for a 12-bit serial multiplier in terms of a single cobweb cellular array. For comparison purposes, the same system has been chosen as was previously reported.[3] The block diagram for this four-register, five-command multiplier is given as Fig. 14, while a previously-reported design in terms of five interconnected cutpoint arrays is shown in Fig. 15.

In Fig. 16, this same system is realized in terms of a single $27 \times 16$-cell cobweb array.

The statistics on these two realizations for the multiplier are as follows:

### *Cutpoint Realization*

There are 352 cells in five cellular arrays, and
100 connections at the edges of the five arrays.
28% of the cells are "1" cells used only for transmitting information,
71% of the cells are used logically, and
1% of the cells are not used.

### *Cobweb Realization*

There are 432 cells in one cellular array, and
10 connections at the edges of the one array.
26% of the cells are jumper cells,
55% of the cells are used logically, and
19% of the cells are not used.

It is seen from the above data that while 26 percent more cells are required for the multiplier in the cobweb realization than in the cutpoint realization, only one cobweb array is used versus five cutpoint arrays; furthermore, the backplane wiring in the cobweb realization is reduced by an order of magnitude.

## CONCLUSIONS

The essential difference between the previously reported cutpoint cellular array and the cobweb array is the more complicated and flexible interconnection structure of the latter array. This flexibility allows the logical designer much more geometric freedom in the embedding of cascade logical realizations. For certain types of digital operations, and in particular for parallel operations, the use of cobweb arrays results in a significant reduction in the required number of cells. Also it is possible to eliminate jumper connections from one edge cell on an array to another edge cell on the same array when cobweb arrays are employed.

## ACKNOWLEDGMENTS

The logical design of the parallel adder in Fig. 9 was provided by Mr. David W. Masters, the supercell concepts were the result of conversations with Mr. Milton W. Green, and some of the ideas on parallel operations in cellular arrays were derived from unpublished work of Mr. Jack Goldberg. Particular credit is due to Dr. Robert A. Short for his critical evaluation of the manuscript and for his many helpful suggestions and comments.

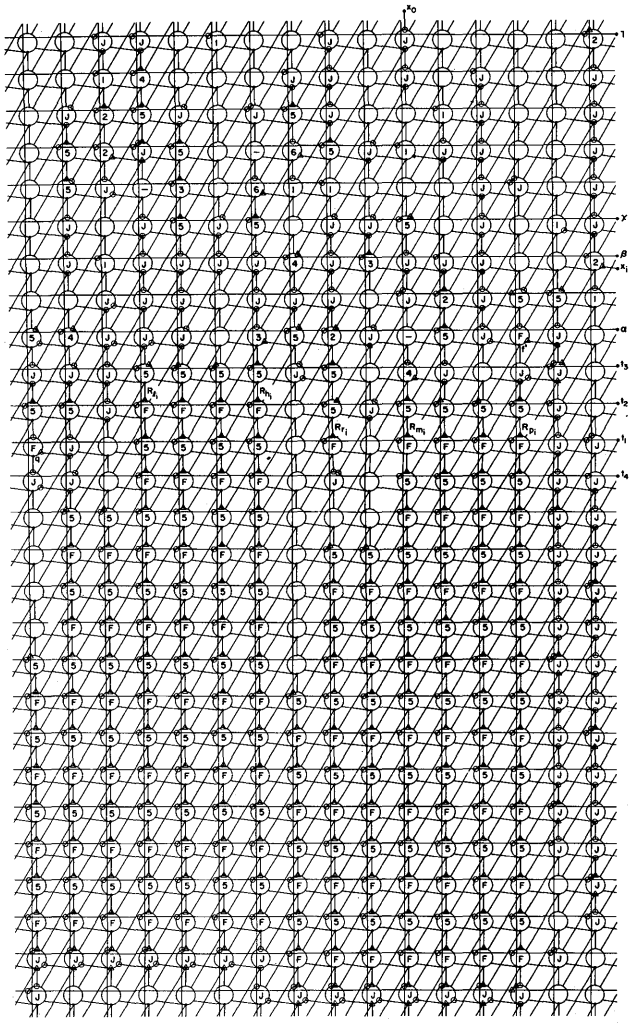


Figure 16. Realization of the multiplier in terms of onecobweb array.

REFERENCES

1. R. C. Minnick and Robert A. Short, "Investigation of Cellular Linear-Input Logic," Final Report, Contract No. AF 19(628)-498; SRI Report No. 4122; Stanford Research Institute, Menlo Park, Calif., prepared for Data Sciences Laboratory, Air Force Cambridge Research Laboratories, Office of Aerospace Research, Bedford, Mass., USAF AFCRL No. 64-6; DDC No. AD 433802 (Dec. 1963).

2. R. C. Minnick, "Cutpoint Cellular Logic," *IEEE Transactions on Electronic Computers,* Vol. EC-13, No. 6, Dec. 1964, pp. 685-698.

3. R. C. Minnick, "Application of Cellular Logic to the Design of Monolithic Digital Systems," presented at a Symposium on Microelectronics and Large Systems co-sponsored by the Office of Naval Research and the Univac Division of Sperry Rand Corp., Washington, D. C., Nov. 17 and 18, 1964. (To be published by Spartan Books, Inc. in 1965.)

# TWO-DIMENSIONAL ITERATIVE LOGIC*

Rudd H. Canaday
*Bell Telephone Laboratories, Incorporated*
*Whippany, New Jersey*

## INTRODUCTION

It is well known that given a suitable Boolean function, a large number of "gates" or "elements," each producing this function, can be interconnected in a regular structure, or "array," to realize any given Boolean function. Furthermore, the structure of the array can be invariant to the function being realized.

One of the simplest such structures is the two-dimensional array of three-input one-output elements shown in Fig. 1. In this paper two methods are presented for using this structure in the synthesis of arbitrary Boolean functions. The following assumptions will be adhered to throughout this paper:

1. All elements in the array are identical.
2. The interconnections between elements in the array are fixed. They cannot be broken or changed in any way.
3. The array will be used as a single output circuit. Only the output of the lower right element of the array is accessible to the outside world.
4. Every element in the array realizes the "majority" function

$$f(A,B,C) = AB + AC + BC$$

of its three inputs.*

As a consequence of assumptions (1), (2), and (4), an array can be described completely in terms of its width $w$ and height $h$. Such an array will be called a "MAJority Array" or "MAJA."

In the remainder of this paper it will be shown, first, how to synthesize an arbitrary "self-dual" function in a MAJA. Then this result will be extended to arbitrary functions and some examples will be given. This is "intersection synthesis." Next a second synthesis technique, "factorization synthesis," will be described, first in a canonic form, through examples, and then in a more general form.

*The material presented in this report is based on a thesis submitted in partial fulfillment of the requirements for the Doctor of Philosophy Degree in Electrical Engineering at the Massachusetts Institute of Technology, September 1964.
The research reported was made possible through the support extended to the M.I.T. Electronic Systems Laboratory by the U.S. Air Force Avionics Laboratory, Navigation and Guidance Division, under Contract AF-33(657)-11311 and, in the earlier phases of this research, under Contract AF-33(657)-8932.

*It is easy to prove[1] that all of the results given here extend directly to arrays of "minority" elements:

$$f(A,B,C) = \overline{AB + AC + BC}$$

This paper is based on the author's Ph.D. thesis.[1] In the present paper space limitations preclude statements of all theorems and proofs on which the synthesis methods are based. These do appear, together with extensions of the results presented here, in reference 1.

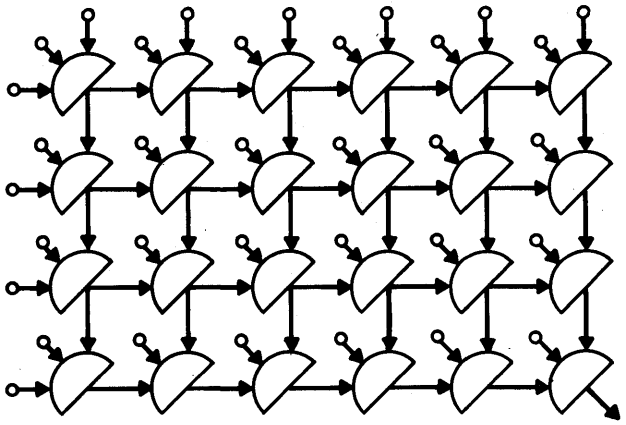


Figure 1. A 4 × 6 array of 3-input elements.

Both synthesis techniques lead to arrays of reasonable size, and embody new synthesis techniques which may prove to be applicable in other forms of synthesis also.

## PRELIMINARY DISCUSSION

Before discussing array synthesis, it is necessary to define some terminology for arrays.

Each element in an array has three inputs, which will be denoted "top," "center," and "left" inputs (signal flow in an array is always left-to-right and top-to-bottom).

The $w$ inputs (for an array of width $w$ ) consisting of the top input to each element in the top row of the array form the "top boundary" inputs to the array.

Similarly, the $h$ inputs (for an array of height $h$), consisting of the left input to each element in the leftmost column of the array, form the "left boundary" inputs to the array.

One particular type of array proves to be of particular interest. This array has, in effect, all top boundary inputs wired together, and all left boundary inputs similarly wired together.

*Definition* : An " $XY$ Standard Boundary Condition majority array" ( $XY$ SBC MAJA) is a MAJA all of whose top boundary inputs carry the signal $Y$ where $Y$ can be a variable or a constant, and all of whose left boundary inputs carry the signal $X$, where $X$ can be a variable or a constant.

Fig. 2 is an example of an $XY$ SBC MAJA. The two synthesis methods to be presented both apply to the SBC MAJA.

Intersection synthesis is given first for self-dual functions, as defined below.

*Definition:* Given a Boolean function $f(x_1, \ldots, x_n)$, then the *dual* $f^d(x_1, \ldots, x_n)$ of the function $f$ is defined as:

$$f^d(x_1, \ldots, x_n) = f(x_1, x_2, \ldots, x_n)$$

By applying deMorgan's theorem one can easily see that if $f$ is expressed using only the operations + (OR), • (AND) and – (NOT), then $f^d$ is obtained by interchanging + and • throughout the expression for $f$.

*Definition:* A Boolean function $f(x_1, \ldots, x_n)$ is *self-dual* if and only if

$$f^d(x_1, \ldots, x_n) = f(x_1, \ldots, x_n)$$

Note that by this definition of dual and self-dual, a function which is a constant is not self-dual since, if $f \equiv 1$ then $f^d = f \equiv 0$.

Any $n$-variable Boolean function $f(x_1, \ldots, x_n)$ can be factored as

$$f(x_1, \ldots, x_n) = Xf_0 + Yf_1 \qquad (1)$$

with $X$ and $Y$ chosen from $\{x_1, x_1, \ldots, x_n, x_n\}$.

If $f$ is a self-dual function, then the existence of the factorization (1) implies that $f$ can be factored as

$$f = Xf_0 + Yf_0^d + XY \qquad (2)$$

where $X$, $Y$, and $f_0$ are the same as in Eq. (1).

Equation (2) is basic to the synthesis algorithm, which is presented in the following two definitions and Theorem 1 below.

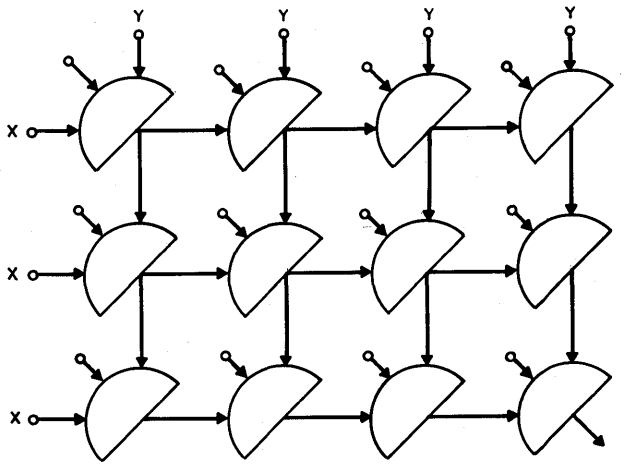



Figure 2. An *XY* SBC MAJA.

## INTERSECTION SYNTHESIS

*Definition:* Given two Boolean functions $f_a$ and $f_b$, and given a sum of products expression for each: $f_a = r_1 + r_2 + \ldots + r_k$; $f_b = t_1 + t_2 + \ldots + t_m$, then an intersection matrix of $f_a \times f_b$ is a matrix with $k$ rows and $m$ columns, in which each entry $e_{ij}$ is the intersection of the literals in $r_i$ with the literals in $t_j$ (i.e., $e_{ij}$ contains a literal $y$ if and only if $y$ is in both $r_i$ and $t_j$).

Note that the intersection matrix for a given $f_a$ and $f_b$ is not unique. It is unique for given sum-of-products expressions (including the ordering of their terms) for both $f_a$ and $f_b$. Now it is possible to define an SBC MAJA to realize any given self-dual function.

*Definition:* Given a self-dual function $f^{sd} = XY + Xf_0 + Xf_0{}^d$, and given a $k \times m$ intersection matrix $f_0 \times f_0{}^d$, with rows corresponding to terms of $f_0$, then an *XY intersection MAJA* for $f^{sd}$ is a $k \times m$ *XY* SBC MAJA with the center input to the $ij^{th}$ element chosen to be any one of the literals in entry $e_{ij}$ of the intersection matrix, for all $i, j$: $1 \leq i \leq k$, $1 \leq j \leq m$.

Again note that one function $f^{sd}$ may have many intersection MAJAs for each factorization (each choice of $X$ and $Y$).

*Theorem 1:* Given any XY intersection MAJA for a self-dual function $f^{sd}$, then the output of the MAJA realizes the function $f^{sd}$.

*Proof:** By construction the MAJA has no constant inputs. Therefore it is sufficient to prove that the MAJA produces all the ones of $f^{sd}$, since a MAJA without constant inputs must realize a self-dual function.[1] It is easy to prove that if the term $XY$ is one the array output is one. Now let a term $Xr_1$ in $Xf_0$ be one. Then every literal in the term is one. Then every left boundary input, and the center input to every element in the $i^{th}$ row, is one. It is not difficult to prove that this condition suffices to insure that the array output is one. Thus the array output is one for every term in $Xf_0$. Similarly if a term $Yt_i$ in $Yf_0{}^d$ is one then every center input to the $i^{th}$ column, as well as every top boundary input, is one. Again this suffices to insure that the array output is one. Thus every one of $f^{sd} = XY + Xf_0 + Yf_0{}^d$ is realized at the output of an intersection MAJA for $f^{sd}$ and so the MAJA realizes $f^{sd}$.

The synthesis algorithm just presented allows one to synthesize any self-dual Boolean function. To extend the result to any arbitrary Boolean function, the "self-dual expression" for a function is defined.

*Definition:* Given any $n$-variable Boolean function $f(x_1, \ldots, x_n)$, and a variable $U$ independent of $(x_1, \ldots, x_n)$, the *Self-Dual Expression* $f^{sd}$ for $f$ is defined as the $(n+1)$-variable function:†

$$f^{sd}(U,x_1,\ldots,x_n) = Uf(x_1,\ldots,x_n) + \bar{U}f^d(x_1,\ldots,x_n).$$

*The proof given here is very sketchy. The detailed proof, which depends on a number of theorems not given here, is in reference 1.

†This is a reformulation of work done by S. B. Akers.[2]

It is trivial to prove that the self-dual expression for any function is a self-dual function. Also, if $f$ is a selfdual function, then

$$f^{sd}(U,x_1, \ldots, x_n) = f(x_1, \ldots, x_n).$$

Clearly this is true if and only if $f$ is self-dual.

To synthesize an arbitrary $n$-variable function, proceed as follows:

1. Find the $(n + 1)$-variable self-dual expression, $f^{sd}$, for the function $f$.
2. Synthesize $f^{sd}(U,x_1, \ldots, x_n)$.
3. Replace every input $U$ to the array by the constant input 1 (one) and every input $\overline{U}$ by the constant 0 (zero).

The resulting array realizes $f(x_1, \ldots, x_n)$ since $f^{sd}(1, x_1 \ldots, x_n) = f(x_1, \ldots, x_n)$ by construction.

The examples to follow show arrays with the inputs $U$ and $\overline{U}$. Thus these arrays, as shown, realize the self-dual expression of the given function. Wherever the inputs $U$ and $\overline{U}$ occur, they can be replaced by 1 and 0 as discussed above to obtain the array for the given function.

Note that the array for a self-dual function contains, by construction, no constant inputs. It can be shown that in any MAJA constant inputs are required if and only if the function being synthesized is non-self-dual.

In an intersection MAJA for a function, every term in the factored expression for the function corresponds to a single row or column in the MAJA. It can be shown[1] that terms in the output function of an SBC MAJA can correspond not only to single rows and columns, but also to inputs (or elements) which do not form a single row or column. Thus it seems that the intersection matrix construction does not make maximum use of the MAJA. In other words, by realizing some terms in the function using a set of elements not from a single row or column, it is possible to realize many functions in an SBC MAJA considerably smaller than an intersection MAJA for the function. By extensions to this work, reduced non-SBC arrays can be derived also, but the methods become much messier and less algorithmic.

It is not possible in the space available here to discuss reduction techniques. However, the following examples show some arrays in reduced form, as well as the original intersection arrays.

While it is possible to construct an intersection array from any factorization of the form

$$f^{sd} = XY + Xf_0 + Yf_0^d$$

with $f_0$ and $f_0^d$ each expressed as a sum of product terms, it is obvious that the smallest array results from choosing $X$ and $Y$ and the expressions for $f_0$ and $f_0^d$ to minimize the number of terms in $f_0$ and in $f_0^d$. This is done in the following examples.

## SYNTHESIS EXAMPLES

Before giving examples of synthesis by Theorem 1, it is useful to define a notation which will be used in examples throughout the rest of this work.

In the many examples to follow in this and succeeding sections it is necessary to show arrays with variables assigned to the inputs. Since the interelement connections in an array are fixed, an array with inputs can be completely specified by giving each boundary input and the center input to each element of the array. These inputs are presented as a matrix, with a line separating top and left boundary variables from the center input variables. Thus the array of Fig. 3 is represented by

| | $B$ | $B$ | $B$ | $B$ | $B$ |
|---|---|---|---|---|---|
| $\overline{B}$ | $\overline{U}$ | $C$ | $C$ | $\overline{D}$ | $A$ |
| $\overline{B}$ | $D$ | $\overline{C}$ | $\overline{C}$ | $U$ | $\overline{A}$ |
| $\overline{B}$ | $C$ | $\overline{A}$ | $\overline{A}$ | $\overline{D}$ | $\overline{C}$ |
| $\overline{B}$ | $\overline{C}$ | $\overline{A}$ | $\overline{A}$ | $\overline{U}$ | $\overline{C}$ |
| $\overline{B}$ | $\overline{C}$ | $\overline{D}$ | $\overline{U}$ | $\overline{D}$ | $\overline{C}$ |

Clearly this representation is completely general; it is not restricted to SBC arrays.

*Example 1:* $f(A,B,C,D) = \Sigma\emptyset,1,4,6,7,8,11,12,13,14.$*

A minimum Sum of Products (MSP) form of the Self-Dual Expression for this function is:

$$f^{sd} = BCDU + ABCDU + ABCD + \overline{A}BCD + \overline{A}BC + \overline{A}BCU + A\overline{B}CU + ABCDU + BDU + CDU$$

This function can be factored on $\overline{B}B$ or $\overline{C}C$ without increasing the number of product terms (1Ø) in the expression, since the term $B\overline{D}U$ can be written $BC\overline{D}U$ or $\overline{C}\overline{D}U$ can be written $\overline{B}\overline{C}\overline{D}U$, without changing $f^{sd}$. Arbitrarily choose the $\overline{B}B$ factorization.

*This notation, defined in Caldwell,[3] defined the rows of the truth table for which the function is one.

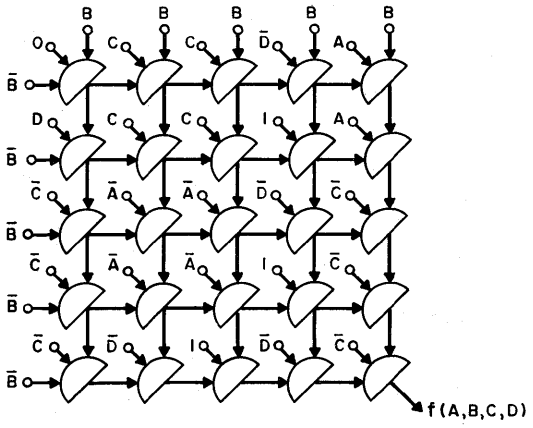



Figure 3. SBC MAJA for Example 1.

The intersection matrix is

| | $B$ $(\bar{C}D\bar{U})$ | $(\bar{A}C\bar{D})$ | $(\bar{A}CU)$ | $(\bar{D}U)$ | $(A\bar{C})$ |
|---|---|---|---|---|---|
| $(ACD\bar{U})$ | $(\bar{U})$ | $(C\bar{D})$ | $(C)$ | $(\bar{D})$ | $(A)$ |
| $(AC\bar{D}U)$ | $(D)$ | $(C)$ | $(CU)$ | $(U)$ | $(A)$ |
| $(A\bar{C}\bar{D})$ | $(\bar{C})$ | $(A\bar{D})$ | $(A)$ | $(D)$ | $(\bar{C})$ |
| $(A\bar{C}U)$ | $(\bar{C})$ | $(A)$ | $(AU)$ | $(U)$ | $(\bar{C})$ |
| $(\bar{C}DU)$ | $(\bar{C})$ | $(\bar{D})$ | $(U)$ | $(\bar{D}U)$ | $(\bar{C})$ |

One intersection MAJA is

| | $B$ | $B$ | $B$ | $B$ | $B$ |
|---|---|---|---|---|---|
| $\bar{B}$ | $\bar{U}$ | $C$ | $C$ | $\bar{D}$ | $A$ |
| $\bar{B}$ | $D$ | $C$ | $C$ | $U$ | $A$ |
| $\bar{B}$ | $\bar{C}$ | $\bar{A}$ | $\bar{A}$ | $\bar{D}$ | $\bar{C}$ |
| $\bar{B}$ | $\bar{C}$ | $\bar{A}$ | $\bar{A}$ | $U$ | $\bar{C}$ |
| $B$ | $\bar{C}$ | $\bar{D}$ | $U$ | $\bar{D}$ | $\bar{C}$ |

By reduction techniques† described elsewhere,[1] a 2×3 non-SBC array can be found to realize this function:

| | $C$ | $\bar{B}$ | $\bar{D}$ |
|---|---|---|---|
| $D$ | $\bar{U}$ | $\bar{A}$ | $U$ |
| $A$ | $A$ | $\bar{C}$ | $B$ |

A 2×3 array is known to be the smallest array capable of realizing this function since no smaller array has enough terminals. The 2×3 array shown here has one element which performs no logical function because it has two identical ($A$) inputs. The 5-element network resulting from removal of element 21 has the adsolute minimum number of 3-input elements for any network capable of realizing this function.

†These techniquees are heuristic, and results obtained depend to some extent on the experience of the person doing the reduction.

*Example 2:* $f(A,B,C,D) = \Sigma 1,4,5,6,7,9,11,12,13,15$

MSP $f^{sd} = BD + \bar{A}\bar{C}D + AB\bar{C} + \bar{A}BU + ADU$ can be written as

$$f^{sd} = BD + B(A\bar{C}+\bar{A}U) + D(\bar{A}\bar{C}+AU)$$

The SBC MAJA is

| | $D$ | $D$ |
|---|---|---|
| $B$ | $A$ | $\bar{C}$ |
| $B$ | $U$ | $\bar{A}$ |

This is the smallest SBC MAJA which can possibly realize this function, since six different literals must appear as inputs, and no smaller SBC array has six inputs.

*Example 3:* $f(A,B,C,D) = \Sigma 1,2,5,7,11$

$$\text{MSP} \quad f^{sd} = \bar{A}\bar{C}D + \bar{A}BD + A\bar{B}CD + \bar{A}B\bar{C}\bar{D} + AB\bar{C}\bar{D}\bar{U} + \bar{B}D\bar{U} + CD\bar{U} + \bar{A}\bar{B}\bar{U} + \bar{A}C\bar{U}$$

Factor on $\bar{D}D$ for the minimum number of product terms in the factored expression

$$f^{sd} = \bar{D}(\bar{A}\bar{B}C+AB\bar{C}\bar{U}+\bar{A}\bar{B}\bar{U}+\bar{A}C\bar{U}) + D(\bar{A}\bar{C}+\bar{A}B+A\bar{B}C+\bar{B}\bar{U}+C\bar{U})$$

A corresponding SBC intersection MAJA is:

| | $D$ | $D$ | $D$ | $D$ | $D$ |
|---|---|---|---|---|---|
| $\bar{D}$ | $\bar{A}$ | $\bar{A}$ | $C$ | $\bar{B}$ | $\bar{C}$ |
| $\bar{D}$ | $\bar{C}$ | $B$ | $A$ | $\bar{U}$ | $\bar{U}$ |
| $\bar{D}$ | $\bar{A}$ | $\bar{A}$ | $\bar{B}$ | $\bar{U}$ | $\bar{U}$ |
| $\bar{D}$ | $A$ | $A$ | $C$ | $\bar{U}$ | $\bar{U}$ |

By reshuffling rows and colums, it becomes possible to remove the row corresponding to term $\overline{A}C\overline{D}\overline{U}$ and the column corresponding to term $CD\overline{U}$

| | $\overline{D}$ | $\overline{D}$ | $\overline{D}$ |
|---|---|---|---|
| $D$ | $\overline{A}$ | $\overline{C}$ | $\overline{A}$ |
| $D$ | $\overline{A}$ | $B$ | $\overline{A}$ |
| $D$ | $\overline{C}$ | $\overline{A}$ | $\overline{B}$ |
| $D$ | $\overline{B}$ | $\overline{U}$ | $\overline{U}$ |

The term $CD\overline{U}$ is realized by the center inputs to elements 31, 42, and 43, and the left boundary. The term $\overline{A}C\overline{D}\overline{U}$ is realized by the center inputs to elements 11, 21, 31, 42, and the top boundary. This is the smallest known SBC array for this function. However, there exists a 2×3 non-SBC majority array for the function:

| | $C$ | $C$ | $D$ |
|---|---|---|---|
| $\overline{B}$ | $\overline{A}$ | $\overline{D}$ | $\overline{C}$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{B}$ | $\overline{A}$ |

Arrays of size 2×3 or 3×3 appear to be typical for non-self-dual functions of four variables.[1]

## THE CANONIC ARRAY

The two major disadvantages of the synthesis method presented above are:

1. The lack of reasonable bounds on the size of array needed to realize an arbitrary function.
2. The inability to apply reduction procedures to functions of more than five or six variables.*

The development of a canonic form for arrays for arbitrary functions of $n$ variables as is done below obviates these disadvantages. This canonic form has the following properties:

1. The canonic array for $n$ variables, for a given $n$, is an array of fixed size, with some inputs fixed and the rest of the inputs chosen for the specific function (typically, well over half of the inputs are fixed). This array will realize any given function of $n$ variables if the nonfixed inputs are properly chosen. An algorithm for determining the inputs needed to realize any given function exists.
2. An algorithm exists for generating the canonic array for any given $n$.
3. The canonic array for $n$ variables is the smallest known array to realize the checkerboard (worst-case) function of $n$ variables, for $n$ even.
4. For most given functions the canonic array is reducible (by methods given in reference 1).
5. The canonic array embodies a technique for embedding arrays within larger arrays, which shows great promise for future work in multiple output arrays and in nonmajority (nonminority) arrays.

The disadvantage of the canonic array is that the array required for a given function usually is larger than the array produced by intersection synthesis and reduction, assuming that the function is small enough to make that synthesis-reduction technique feasible.†

The size of the canonic array is shown in Table 1 as a function of $n$ , the number of variables in the function to be synthesized. In addition a "connection count" is shown for each $n$. This is the number of connections to the array which are not invariant over all functions of $n$ variables, plus one connection for each variable whose input connections are invariant. One can envision building the array with all invariant connections wired together at the time of manufacture. Then the "connection count" is just the number of input terminals to the array to allow it to realize any function of $n$ variables.

Table 1.

| n | Size | Connections |
|---|---|---|
| 3 | 3×3 | 10 |
| 4 | 4×6 | 16 |
| 5 | 7×8 | 26 |
| 6 | 9×14 | 44 |
| 7 | 15×18 | 78 |
| 8 | 19×30 | 144 |
| 9 | 31×38 | 274 |

*Note, however, that the basic synthesis algorithm (Theorem 1) can be applied to arbitrarily large functions, though the resulting arrays generally are unreasonably large.

†Reduction is feasible on most functions of five variables and some functions of six variables.[1]

## CONSTRUCTION OF CANONIC ARRAYS

In this section the canonic array for $n$ variables is presented through examples. In the next section the process of embedding subarrays in an array is considered more generally.

Consider the MAJA

| | $U$ | $U$ |
|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $g_1$ |
| $\overline{U}$ | $g_0$ | $A$ |

Array 1

in which $g_0$ and $g_1$ are input literals chosen from the set $\{B,\overline{B},U,\overline{U}\}$. This array, as straightforward analysis will show, realizes the self-dual function

$$f^{sd} = U[\overline{A}g_0 + Ag_1] + \overline{U}[\overline{A}g_1 + Ag_0]$$

which can be any self-dual function of the three variables $(A,B,U)$. If $U = 1$, Eq. (3) becomes

$$f = \overline{A}g_0 + Ag_1 \qquad (4)$$

which, if $g_0$ and $g_1$ are chosen from $\{\overline{B},B,0,1\}$, can be any function of the two variables $(A,B)$. Thus Array 1 is the *canonic factorization* array for $n = 2$ variables. It is called a "factorization" array because Eq. (4) is a factorization of the function. It is called "canonic" because it is in a standard form, as will become clear later.

Now consider the MAJA

| | $U$ | $U$ | $U$ |
|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $g_{11}$ | $\overline{B}$ |
| $\overline{U}$ | $g_{00}$ | $B$ | $g_{10}$ |
| $\overline{U}$ | $\overline{B}$ | $g_{01}$ | $A$ |

*Array 2*

If $U = 1$, this array realizes the function

$$f = \overline{A}\overline{B}g_{00} + \overline{A}Bg_{01} + A\overline{B}g_{10} + ABg_{11} \qquad (5)$$

If $g_{00}$, $g_{01}$, $g_{10}$, and $g_{11}$ are chosen from $\{C,\overline{C},0,1\}$, then Eq. (5) can be any function of the three variables $(A,B,C)$. Array 2 is called the canonic factorization array for $n = 3$ variables even though its form differs slightly from the canonic construction to be defined.

It would be possible to continue thus to define arrays to realize any function of $n$ variables for $n = 4,5,6$, and so on. However, if one wishes to define a canonic factorization array for an arbitrary number of variables, it is necessary to define a construction method which will result in the canonic array for any given $n$. Here the approach taken is inductive. Given the canonic factorization array for $(n-1)$ variables, it will be shown how to construct the array for $n$ variables. This will be done by embedding two arrays for $(n-1$ variables in the factorization array for $n$ variables. Consider first the case of $n = 4$. The array for $(n-1) = 3$ variables is known (Array 2). Take two of them:

| | $U$ | $U$ | $U$ |
|---|---|---|---|
| $\overline{U}$ | $\overline{B}$ | $g_{011}$ | $\overline{C}$ |
| $\overline{U}$ | $g_{000}$ | $C$ | $g_{010}$ |
| $\overline{U}$ | $\overline{C}$ | $g_{001}$ | $B$ |

*Array 3*

Array 3, with $U = 1$, realizes

$$g_0 = \overline{B}\overline{C}g_{000} + \overline{B}Cg_{001} + B\overline{C}g_{010} + BCg_{011} \qquad (6)$$

| | $U$ | $U$ | $U$ |
|---|---|---|---|
| $\overline{U}$ | $\overline{B}$ | $g_{111}$ | $\overline{C}$ |
| $\overline{U}$ | $g_{100}$ | $C$ | $g_{110}$ |
| $\overline{U}$ | $\overline{C}$ | $g_{101}$ | $B$ |

*Array 4*

Array 4, with $U = 1$, realizes

$$g_1 = B\overline{C}g_{100} + B\overline{C}g_{100} + B\overline{C}g_{110} + BCg_{111} \qquad (7)$$

Combine Array 3 and Array 4 in the canonic factorization array for $n = 4$

| | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ |
|---|---|---|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{B}$ | $g_{111}$ | $B$ |
| $\overline{U}$ | $\overline{B}$ | $g_{011}$ | $\overline{C}$ | $g_{100}$ | $C$ | $g_{110}$ |
| $\overline{U}$ | $g_{000}$ | $C$ | $g_{010}$ | $\overline{C}$ | $g_{101}$ | $B$ |
| $\overline{U}$ | $\overline{C}$ | $g_{001}$ | $B$ | $A$ | $A$ | $A$ |

*Array 5*

where dotted lines have been shown only to clarify the construction of the array. It should be emphasized that Array 5 is a $4 \times 6$ $\overline{U}U$ SBC MAJA, with no modification of its structure. The array contains subarrays only in the sense that the input pattern to portions of the array can be identified with the input patterns to Array 3 and Array 4.

Array 5, with $U = 1$, realizes the function

$$f = \overline{A}g_0 + Ag_1 \qquad (8)$$

where $g_0$ and $g_1$ are the arbitrary 3-variable functions realized by Array 3 and Array 4. To see this, let $U = 1$ (and $U = 0$, of course). Then if $A = 1$, the subarray corresponding to Array 4 (elements 14,15,16,24,25,26,34,35, and 36) has one on its top boundary (elements 14,15,16), and zero on its left boundary (elements 14,24,34). It is not difficult to see that with $A = 1$, Array 5 has the same output as Array 4. Similarly, if $A = 0$, then the subarray corresponding to Array 3 has zero on its left boundary (elements 21,31,41) and one on its top boundary (elements 21,22,23) and it can be shown that Array 5 has the same output as Array 3. Thus Eq. (8) is vertified.

By substitution of Eqs. (6) and (7) into Eq. (8), one obtains

$$f = \overline{A}\,\overline{B}\,\overline{C}g_{000} + \overline{A}\,\overline{B}Cg_{001} + \ldots + A\overline{B}\,\overline{C}g_{110} + ABCg_{111} \qquad (9)$$

If $g_{000}$ through $g_{111}$ are chosen from $\{\overline{D},D,0,1\}$, then equation (9) can be any function of the four variables $(A,B,C,D)$.

By interchanging rows and columns in Array 5 and then interchanging $U$ and $\overline{U}$ and changing the subscripts on the $g$ inputs appropriately, one obtains the MAJA,

| | $U$ | $U$ | $U$ | $U$ |
|---|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $\overline{B}$ | $g_{111}$ | $\overline{C}$ |
| $\overline{U}$ | $\overline{A}$ | $g_{100}$ | $C$ | $g_{110}$ |
| $\overline{U}$ | $\overline{A}$ | $\overline{C}$ | $g_{101}$ | $B$ |
| $\overline{U}$ | $\overline{B}$ | $g_{011}$ | $\overline{C}$ | $A$ |
| $\overline{U}$ | $g_{000}$ | $C$ | $g_{010}$ | $A$ |
| $\overline{U}$ | $\overline{C}$ | $g_{001}$ | $B$ | $A$ |

*Array 6*

which realizes the same function, Eq. (9), as does Array 5. This is the *flipped canonic factorization array* for four variables, "flipped" because it corresponds to Array 5 flipped about its main diagonal (and with $U$, $\overline{U}$ interchanged and the $g$'s renumbered).

To construct the canonic factorization array for five variables one embeds two 4-variable subarrays in a factorization array in exactly the same manner in which Array 5 was constructed. If one uses as subarrays two copies of Array 5, the resulting 5-variable array is $5 \times 12$, with 60 elements. If, however, one uses the flipped array, Array 6, the resulting 5-variable array is $7 \times 8$ with 56 elements:

| | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ |
|---|---|---|---|---|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{B}$ | $\overline{C}$ | $g_{1111}$ | $\overline{D}$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{C}$ | $g_{0111}$ | $\overline{D}$ | $\overline{B}$ | $g_{1101}$ | $D$ | $g_{1110}$ |
| $\overline{U}$ | $\overline{B}$ | $g_{0100}$ | $D$ | $g_{0110}$ | $\overline{B}$ | $\overline{D}$ | $g_{1101}$ | $C$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{D}$ | $g_{0101}$ | $C$ | $\overline{C}$ | $g_{1011}$ | $\overline{D}$ | $B$ |
| $\overline{U}$ | $\overline{C}$ | $g_{0011}$ | $\overline{D}$ | $B$ | $g_{1000}$ | $D$ | $g_{1010}$ | $B$ |
| $\overline{U}$ | $g_{0000}$ | $D$ | $g_{0010}$ | $B$ | $\overline{D}$ | $g_{1001}$ | $C$ | $B$ |
| $\overline{U}$ | $\overline{D}$ | $g_{0001}$ | $C$ | $B$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $A$ |

*Array 7*

Again, the dotted lines are included to clarify the construction. Array 7 realizes the function

$$f = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D}g_{0000} + \overline{A}\,\overline{B}\,\overline{C}Dg_{0001}\,7 \ \ldots + ABCDg_{1111}$$

It is interesting to note that the canonic factorization array is the smallest array known which realizes the "checkerboard" function $f(A,B,C,D) = A + B + C + D$.* The canonic factorization array for this function is

| | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ |
|---|---|---|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{B}$ | $\overline{D}$ | $\overline{C}$ |
| $\overline{U}$ | $\overline{B}$ | $D$ | $\overline{C}$ | $\overline{D}$ | $C$ | $D$ |
| $\overline{U}$ | $D$ | $C$ | $\overline{D}$ | $\overline{C}$ | $D$ | $B$ |
| $\overline{U}$ | $\overline{C}$ | $\overline{D}$ | $B$ | $A$ | $A$ | $A$ |

However, for five variables no function is known which cannot be realized in a reduced intersection MAJA smaller than Array 7. It is true in general that no function is known to be "worst case" for $n$ odd, although the "checkerboard" function is always "worst case" for $n$ even.

The construction of canonic factorization arrays for higher values of $n$ is carried out by successive embedding of $(n-1)$-variable flipped arrays, as was just illustrated for $n = 5$. Let $H_n$ denote the height of the canonic factorization array for $n$ variables, and let its width be $W_n$. Then, by the construction of the canonic factorization array

$$H_3 = 3, \qquad W_3 = 3$$

and

$$H_n = W_{n-1} + 1, \qquad W_n = 2H_{n-1} \qquad \text{for } n > 3$$

These array sizes are tabulated in Table 1. Note that in the canonic factorization array for $n$ variables $\{x_1,x_2,\ldots,x_n\}$, $2^{n-1}$ of the inputs, the $g$ inputs,

*This function is termed "checkerboard" because its Karnaugh Map representation resembles a checkerboard.

depend on the function being realized, while all other inputs are fixed, independent of the particular function being realized and equal to one of the $2n$ literals $\{x_1,x_1,x_2,x_2, \ldots, x_{(n-1)},U,U\}$ $(U, U)$ being in fact constants, of course), so that if all identical fixed inputs are wired together at the time of manufacture, only $2n + 2^{n-1}$ external connections to the array need be provided. This "connection count" is also tabulated in Table 1.

As a final illustration, the canonic factorization array for $n = 6$, $f(A,B,C,D,E,F)$, is shown below, with solid lines indicating the two 5-variable subarrays and dotted lines indicating the four 4-variable sub-subarrays.

| | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ | $U$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\overline{U}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{A}$ | $\overline{B}$ | $\overline{C}$ | $\overline{C}$ | $\overline{C}$ | $\overline{D}$ | $g$ | $\overline{E}$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{C}$ | $\overline{C}$ | $\overline{C}$ | $\overline{D}$ | $g$ | $\overline{E}$ | $\overline{B}$ | $\overline{D}$ | $g$ | $\overline{E}$ | $g$ | $\overline{E}$ | $g$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{D}$ | $g$ | $\overline{E}$ | $g$ | $\overline{E}$ | $g$ | $\overline{B}$ | $g$ | $\overline{E}$ | $g$ | $\overline{E}$ | $g$ | $D$ |
| $\overline{U}$ | $\overline{B}$ | $g$ | $\overline{E}$ | $g$ | $\overline{E}$ | $g$ | $D$ | $\overline{B}$ | $\overline{E}$ | $g$ | $D$ | $C$ | $\overline{C}$ | $C$ |
| $\overline{U}$ | $\overline{B}$ | $\overline{E}$ | $g$ | $D$ | $C$ | $\overline{C}$ | $C$ | $\overline{C}$ | $\overline{C}$ | $\overline{C}$ | $D$ | $g$ | $\overline{E}$ | $B$ |
| $\overline{V}$ | $\overline{C}$ | $\overline{C}$ | $\overline{C}$ | $\overline{D}$ | $g$ | $\overline{E}$ | $B$ | $\overline{D}$ | $g$ | $\overline{E}$ | $g$ | $E$ | $g$ | $B$ |
| $\overline{V}$ | $\overline{D}$ | $g$ | $\overline{E}$ | $g$ | $E$ | $g$ | $B$ | $g$ | $E$ | $g$ | $\overline{E}$ | $g$ | $D$ | $B$ |
| $\overline{V}$ | $g$ | $E$ | $g$ | $\overline{E}$ | $g$ | $D$ | $B$ | $\overline{E}$ | $g$ | $D$ | $C$ | $C$ | $C$ | $B$ |
| $\overline{V}$ | $\overline{E}$ | $g$ | $D$ | $C$ | $C$ | $C$ | $B$ | $A$ | $A$ | $A$ | $A$ | $A$ | $A$ | $A$ |

where the 32 nonfixed inputs $g_{00000}$ through $g_{11111}$ are all denoted simply $g$.

The factorization array, unless it has prewired fixed inputs, can often be reduced for a given function. The factorization array factors a function into subfunctions, each of which is in turn factored until eventually each sub-subfunction consists of a single literal. Each subfunction is realized in a $\overline{U}U$ SBC subarray. Many of the subfunctions may have $\overline{U}U$ SBC MAJA realizations smaller than the one used in the factorization array. These can be substituted for the standard subarray with a corresponding decrease in array size.

## EMBEDDED SUBARRAYS

In this section the process of embedding subarrays will be considered in general. If one has two $\overline{U}U$ SBC MAJA's realizing two self-dual functions

$$g_0^{sd} = Ug_0 + \overline{U}g_0^d$$

and

$$g_1^{sd} = Ug_1 + \overline{U}g_1^d$$

then these two arrays can be embedded in a $WV$ SBC MAJA to realize the self-dual function

$$f^{sd} = VXg_0 + VYg_1 + WXg_1^d + WYg_0^d + VW + WXY$$

where $X$ and $Y$ are any single literals. If the array for $g_0^{sd}$ is denoted

| | $U$ | $U \ldots U$ |
|---|---|---|
| $\overline{U}$ | $\delta$ | $\delta \ldots \delta$ |
| $\overline{U}$ | $\delta$ | $\delta$ . |
| . | . | . |
| . | . | . |
| . | . | . |
| $\overline{U}$ | $\delta$ | $\delta \ldots \delta$ |

where $\delta$ denotes the various inputs of the array for $g_0^{sd}$, and if the array for $g_1^{sd}$ is denoted, similarly,

| | $U$ | $U \ldots U$ |
|---|---|---|
| $\overline{U}$ | $\epsilon$ | $\epsilon \ldots \epsilon$ |
| $\overline{U}$ | $\epsilon$ | $\epsilon$ . |
| . | . | . |
| . | . | . |
| . | . | . |
| $\overline{U}$ | $\epsilon$ | $\epsilon \ldots \epsilon$ |

then the array for $f^{sd}$ is

| | $V$ | $V \ldots .$ | | . | $. \ldots V$ |
|---|---|---|---|---|---|
| $W$ | $X$ | $X$ | $X$ | $\epsilon$ | $\epsilon \ldots \epsilon$ |
| $W$ | $\delta$ | $\delta$ | $\delta$ | $\epsilon$ | $\epsilon$ . |
| . | $\delta$ | $\delta$ | . | | . |
| . | . | | . | | . |
| . | . | $\ldots .$ | | | . |
| . | . | $\ldots .$ | | $\epsilon$ | $\epsilon \ldots \epsilon$ |
| $W$ | $\delta$ | $\delta \ldots \delta$ | | $Y$ | $Y \ldots Y$ |

*Array 8*

In this illustration the arrays for $g_0^{sd}$ and $g_1^{sd}$ have been assumed to be of equal height. This need not be true in general.

The formal definition of this construction follows.

Given a self-dual function $f^{sd}(x_1,x_2,\ldots,x_n)$, express it in a $VW$ factorization as $f^{sd} = Vg + Wg^d + VW$ where the function $g$ may or may not be self-dual. This is always possible, if $V$ and $W$ are properly chosen from $\{x_1,\bar{x}_1,x_2,\ldots,\bar{x}_n\}$. Then factor $g$ as $g = Xg_0 + Yg_1$. Again, this is always possible.

Then

$$g^d = Xg_1^d + Yg_0^d + XY$$

and

$$f^{sd} = VXg_0 + VYg_1 + WXg_1^d + WYg_0^d + VW + WXY$$

with $V,W,X,$ and $Y$ chosen from $\{x_1,x_1,x_2,\ldots,x_n\}$.

Now construct an SBC $UU$ MAJA to realize $g_0^{sd} = Ug_0 + Ug_0^d$. Call this array $A_0$. Let its size be $h_0 \times w_0$. Similarly construct the SBC $UU$ MAJA $A_1$ of size $h_1 \times w_1$ to realize $g_1^{sd} = Ug_1 + Ug_1^d$. Note that there is no restriction on how $A_0$ and $A_1$ are constructed, or on their size. It will now be shown that the two arrays $A_0$ and $A_1$ can be embedded in an SBC $WV$ MAJA, $A$, of size $h \times (w_0 + w_1)$ which realizes $f^{sd}$, where $h$ equals the larger of $h_0 + 1$ and $h_1 + 1$.

Let the center input to element $ij$ of $A_0$ be called $\alpha^0_{ij}$, defined for all $i$, $j$: $1 \leq i \leq h_0$, $1 \leq j \leq w_0$. Similarly, let the center input to element $ij$ of $A_1$ be called $\alpha^1_{ij}$, defined for all $i$, $j$: $1 \leq i \leq h_1$, $1 \leq j \leq w_1$. Then the inputs to the $h \times w$ array $A$ are assigned as follows, where $h$ = the largest of $h_0 + 1$ and $h_1 + 1$ and $w = w_0 + w_1$ and $\alpha_{ij}$ denotes the center input to element $ij$ of $A$:

For $1 \leq i \leq h - h_0$ and $1 \leq j \leq w_0$
$\alpha_{ij} = X$

For $h - h_0 < i \leq h$ and $1 \leq j \leq w_0$
$\alpha_{ij} = \alpha^0_{kj}$ with $k = i - (h - h_0)$

For $1 \leq i \leq h_1$ and $w_0 < j \leq w$
$\alpha_{ij} = \alpha^1_{i(j-w_0)}$

For $h_1 < i \leq h$ and $w_0 < j \leq w$
$\alpha_{ij} = Y$

This specifies every input to A in terms of $X$ and $Y$ and the inputs to $A_0$ and $A_1$. Array 8 is an example. It can be proved[1] that the array just defined has as output the function $f^{sd}(x_1,\ldots,x_n)$.

It is very important to observe that the only restrictions on the arrays $A_0$ and $A_1$ are

1. That they are SBC arrays with $U$ and $\bar{U}$ as boundary variables.
2. That they realize $\overset{sd}{g_0} = Ug_0 + U\overset{d}{g_0}$ and $\overset{sd}{g_1} = Ug_1 + U\overset{d}{g_1}$ respectively.

Condition (2) is *not* equivalent to the condition (2′): That when $U = 1$ and $U = 0$, $A_0$ and $A_1$ realize $g_0$ and $g_1$ respectively.

Since the subarrays $A_0$ and $A_1$ can be any $UU$ SBC MAJA's realizing the functions $\overset{sd}{g_0}$ and $\overset{sd}{g_1}$ respectively, it is possible to construct one or both of $A_0$ and $A_1$ themselves as factored arrays. In fact, the canonical factorization array for $n$ variables is just a factored array with each subarray factored and each sub-subarray factored and so on until each sub-sub . . . subarray is a $3\times3$ canonical array which realizes a function of only three variables.

To illustrate the use of factored subarrays in a factored array in the general case, express $\overset{sd}{g_0}$ and $\overset{sd}{g_1}$ as

$$\overset{sd}{g_0} = UR_0g_{00} + US_0g_{01} + UR_0\overset{d}{g_{01}} + US_0\overset{d}{g_{00}} + UR_0S_0$$

and

$$\overset{sd}{g_1} = UR_1g_{10} + US_1g_{11} + UR_1\overset{d}{g_{11}} + US_1\overset{d}{g_{10}} + UR_1S_1$$

and realize each of them in factored $UU$ arrays, which are used as subarrays in the array for $f^{sd}$. Figure 4 shows the construction of the resulting array. In this array the function $f^{sd}$ has been factored as

$$f^{sd} = VXR_0g_{00} + VXS_0g_{01} + VYR_1g_{10} + VSY_1g_{11} + VYR_1S_1 + WXR_1g + WYR_0g_{01} + WYS_0g_{00} + WXY + VW$$

(For the sake of illustration it has been assumed in Fig. 4 that $g_{00}^{sd}$ can be realized in a $2\times2$ SBC $UU$ MAJA, while $g_{01}^{sd}$, $g_{10}^{sd}$, and $g_{11}^{sd}$, each require a $3\times3$ SBC $UU$ MAJA.)

A study has been made of two-dimensional arrays of three-input one-output gates, or elements, each element realizing the majority function of its three inputs ($f(A,B,C) = AB+AC+BC$). These arrays are functionally equivalent to arrays of minority elements ($f(A,B,C) = AB+AC+BC$).

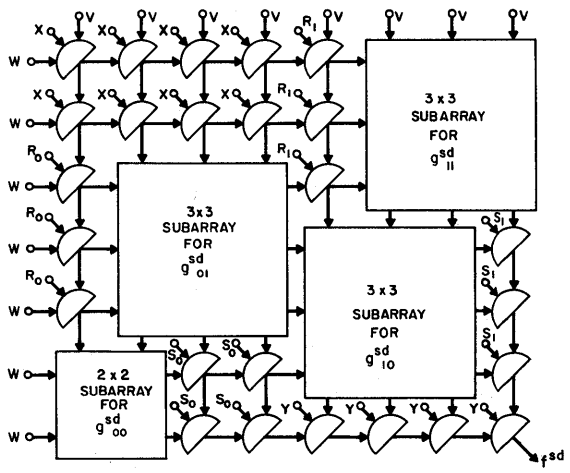



Figure 4. Four subarrays embedded in an SBC factorization MAJA.

## SUMMARY

Two methods are developed for synthesizing any given Boolean function in an array. The first method results in an array whose size depends on the particular function being realized. The second method results in an array whose size depends only on the number of variables in the function being realized. Any 4-variable function, for example, can be realized in an array of 24 elements or less.

The principle result of this work is a simple algorithmic synthesis procedure with the following properties:

1. It is based on building blocks (arrays) which are characterized solely by their width and height, and which contain only simple three-input, one-output elements of *one* type, with a maximum output load of two elements each.
2. It results in arrays obeying a known upper bound on size that seems reasonably small.
3. It permits the synthesis of any Boolean function of n-variables by specifying no more than $2^{(n-1)}$ inputs to the array.
4. It permits the logical decomposition of the array into subarrays, corresponding to a decomposition of the function into subfunctions, with no physical modification of the array.
5. It results in circuits (arrays) with a longer delay, and hence lower speed, than conventional logic circuits.
6. It often requires more elements to realize a given function than do methods less constrained in element type and interconnection.

## REFERENCES

1. R. H. Canaday, "Two-Dimensional Iterative Logic," Report ESL-R-210 Electronic Systems Laboratory, Massachusetts Institute of Technology, Cambridge, Mass., (Sept. 1964). The same material appears in: R. H. Canaday, "Two-Dimensional Iterative Logic," M.I.T. Department of Electrical Engineering, Ph.D. Thesis, Sept. 1964.

2. S. B. Akers, Jr., "The Synthesis of Combinational Logic Using Three-Input Majority Gates"; *Third Annual Symposium on Switching Circuit Theory and Logical Design,* Chicago, October 7-12, 1962.

3. S. H. Caldwell, *Switching Circuits and Logical Design,* Wiley and Sons, New York, 1958.

# TWO-RAIL CELLULAR CASCADES*

Robert A. Short
*Stanford Research Institute*
*Menlo Park, California*

## INTRODUCTION

The increasing importance of integrated circuit technologies has motivated research into the development of systematic and efficient procedures for the design of cellular arrays—that is, arrays of logical assemblies, or cells, that are interconnected in a regular fashion. A useful and analytically attractive approach to the design of two-dimensional, edge-fed cellular arrays for the realization of arbitrary switching functions is based upon the decomposi tion of the arbitrary function into a set of subfunc tions, each of which is independently produced by one of the columns of the array. In this approach, each column of the array might realize, for example, an individual member of a minimum covering set of prime implicants; these subfunctions are then composed by "collecting" the column outputs in a special row of the array whose final output is a realization of the desired function, or alternatively by using edge jumpers in the same array to accomplish the collecting function.

The total number of cells in such arrays typically grows as $Cn2^n$, where the exponential factor reflects the growth of the width of the array, and where $C$ is a constant determined by the particular design algorithm and the logical capabilities of the individual columns. For example, in a minterm composition $C$ may approach the value of ½. The successful reduction of the constant $C$ has been achieved by enlarging the class of functions that can be realized by a single column. In particular, for the simplest possible interconnection structure with which a column, or cascade, can be constructed—one using two-input single-output cells—more sophisticated design techniques have been developed that show that $C$ need not exceed the order of ⅛ for large $n$.

In this paper an augmentation of this simplest interconnection structure will be examined; in particular, provision will be made for two connections between each of the cells of a column in an attempt to increase significantly the number of functions that are realizable within a single column. This simple structural augmentation that provides one more interconnecting lead between cells proves to be significant, for it results in a structure that is functionally complete—that is, any function of an arbitrary number of variables can be combinationally realized within the extent of a single column.

Before summarizing these results, as well as others directed toward reducing the number of cells required for such cascades, a brief review is given below of pertinent prior results.

*The research reported in this paper was sponsored by the Air Force Cambridge Research Laboratories, Office of Aerospace Research, under contract AF 19(628)-4233.

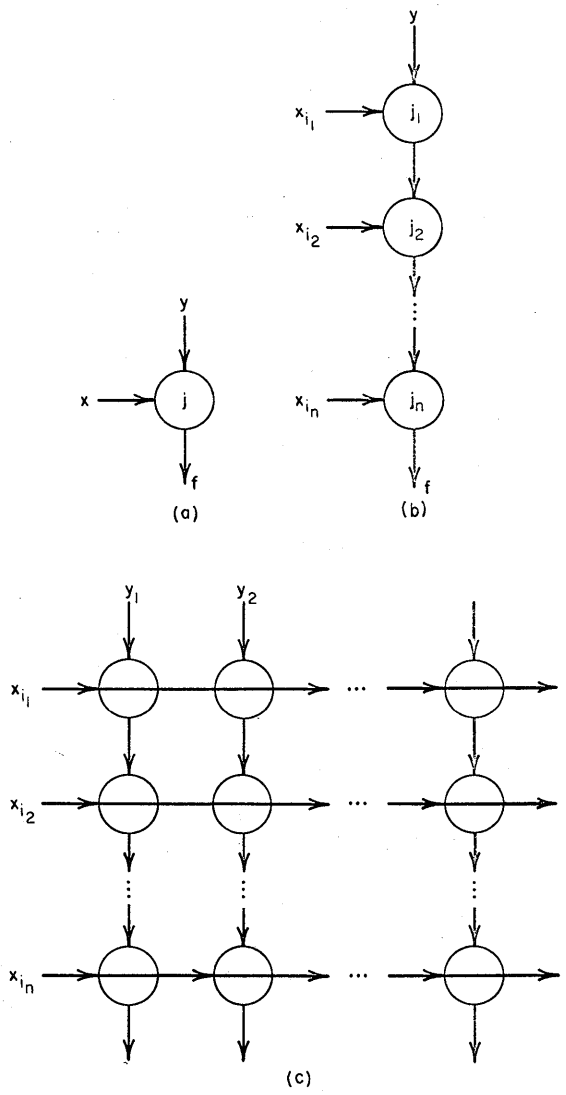



Figure 1. Array composition for simplest cell.

## SINGLE-RAIL CASCADES

The simplest logic cell from which a single column, or cascade, can be constructed is the familiar two-input cell indicated in Fig. 1a. It is assumed that the cell is sufficiently complex that any of the two-variable functions of $x$ and $y$ can be specified for it, as noted by the index $j$ on the cell itself. Thus, for the single cell output, $f = f_j(x,y)$, where $j$ can range over any of the 16 possible 2-variable functions.

The cascade that results from the interconnection of such cells is indicated in Fig. 1b. The independent variables enter the cells directly, e.g., $x_{i_2}$ provides an input to the second cell; for convenience, the topmost free input, $y$, will be taken as an binary constant. The output of the cascade is formed on the single free output at the bottom and is some function of the $i_n$ variables, in particular

$$f = f_{j_n}[x_{i_n}, f_{j_{n-1}}(x_{i_{n-1}}, \ldots)]$$

One way of utilizing such cascades in two-dimensional edge-fed rectangular arrays is indicated in Fig. 1c, where the second output for each cell is formed by reproducing the horizontal input. This scheme has been utilized in the cutpoint arrays developed by Minnick.[1] In these arrays, where the horizontal cell inputs are functions of the independent variables and are not functions of neighboring columns, the basic functional building block remains the column itself. The functional capabilities of cascades

of the type shown in Fig. 1b have been extensively investigated.

The case where the external independent variables $x_1, \ldots, x_m$ correspond one-for-one with the cell inputs $x_{i_1}, \ldots, x_{i_n}$ has been studied by Maitra,[2] Sklansky,[3] and Levy, Winder and Mott.[4] Design algorithms for producing the limited class of functions are available. Obviously the same cascade will suffice for the realization of any function in the class of functions obtained by permuting the independent variables; the total number of such realizable equivalence classes has been determined by Stone.[5] The point of present interest, however, is that the number of such functions comprises an increasingly insignificant proportion of the totality of possible functions with increasing $m$. There are even some three-variable functions that cannot be produced in such elemental cascades, e.g., the majority function.

A refinement which increases the number of functions producible by this single-rail cascade ("single-rail" referring to the single connection between cells of the cascade) was demonstrated by Minnick,[6] who showed that a one-to-many correspondence between the independent variables and the horizontal cell inputs results in a larger class of realizable functions. These cascades have been called redundant cascades, i.e., $m \leq n$, and result in the use of certain of the independent variables to form more than a single cell input. Design algorithms have also been developed for these redundant cascades,[7] but again the pertinent point here is that the class of functions, while larger than the irredundant case for all $m \geq 3$, still forms a vanishing proportion of all functions for sufficiently large $m$. Indeed the redundant cascades still do not quite suffice for the three-variable case.

There is no other way in which the elemental binary two-input single-output cell can be generalized to augment further the number of producible functions. Although arbitrary functional complexity is permitted within the cells,* the interconnection complexity is simply not sufficient for complete functional capability in a single column.

With this fundamental question disposed of, the next-rank question becomes of interest: namely, how does the functional capability of cascades grow with increasing liberality in the allowable interconnection structure? Does any linear cascade exist that does achieve complete functional capabilities?

There would appear to be only two ways in which the interconnection structure of cascades can be liberalized within the basic cascade structure, which requires that a given cell receive inputs (other than external) only from its immediate predecessor, and supplies outputs only to its immediate successor. In one of these ways the number of external inputs to a cell can be augmented, in the other alternative the number of connections between cells can be increased. (A distinction should be made between this latter alternative and the rather intricate interconnection patterns exemplified by the "cobweb" arrays that have been proposed by Minnick.[8] In the cobweb array, each cell has as available inputs not only horizontal and vertical buses, but also inputs from the cell above, the cell to the right, and one of the cells located a knight's move above. Each cell also has an augmented switching network—e.g., cutpoints—by which it selects any two of the five inputs to which it responds, and the particular output that it will activate. The resulting designs achieve a much more efficient utilization of the cells of the array, but the essential function-composition capabilities are still those that characterize the single-rail cascade.)

The first of these alternatives has been investigated by Lendaris and Stanley,[9] and on the basis of their results can be quickly dismissed. The basic cascade with $m$ external inputs per cell is indicated in Fig. 2, and it is assumed that each cell can produce any of the $(m + 1)$-variable functions on its output. In summary, although the class of functions realizable in such a cascade is increased—obviously all $(m + 1)$–variable functions are realizable, for example—it is still deficient* and becomes increasingly so, as noted in the previously mentioned cascades, as the number of variables increases.

The second alternative, an increase in the number of interconnections between cells, is indicated in Fig. 3 for the case of two such cell interconnections, i.e., the "two-rail" generalization of the elemental single-rail cascade previously discussed.[10] Again it will be assumed that the cells are individually as complex as need be and, in this case, that

*Although arbitrary functional complexity in each cell has been assumed, it is interesting to note that such complexity is not necessary. In fact, Minnick[1] has demonstrated that any one of 64 different sets of 6 functions each is sufficient for these purposes. This result serves only to reduce the individual cell complexity (i.e., to reduce the range on $j$ in Fig. 1a), but not to reduce the number of cells required in a cascade.

*It should be noted that these cascades have not been examined in the redundant situation; there is no reason to expect any basic difference in the results, however.

any three-variable function of the cell inputs can be formed (again specified by the $j$-index). If necessary, variable redundancy will also be permitted, and as before the topmost $y$-inputs will be taken as binary constants. However, no greater generalization than the two-rail cascade will be considered; for as will be shown below, the set of functions realizable is not only greatly augmented, but indeed includes the entire class of possible functions. That is, the two-rail cascade is complete.

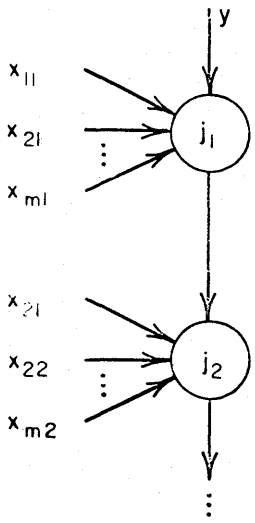



Figure 2. Basic multiple-external-input cascade.

## TWO-RAIL CASCADES

### *Logical Completeness*

The isolated basic two-rail cell is shown in Fig. 4. The cell has three inputs (one is externally supplied) and two outputs. The index $j$ indicate swhich of the three-variable functions of the cell inputs are produced independently by the cell outputs.

The greater flexibility offered by such a cell (which of course must be measured against the greater cell complexity) is suggested by the cascade of Fig. 5 which will realize any four-variable switching function depending upon the specialization of the cells. In particular if the index notation

$$j : f_1(x, y_1, y_2), f_2(x, y_1, y_2)$$

is used to specify the functional outputs of a given cell, then the assignments of

$j_1 : x, 0$
$j_2 : y_1, x$
$j_3 : g_1(x, y_1, y_2), g_0(x, y_1, y_2)$
$j_4$ : arbitrary, $xy_1 + x'y_2 = f(x_1, x_2, x_3, x_4)$

in the cascade of Fig. 4 will realize any

$$f(x_1, x_2, x_3, x_4) = x_4g_1 + x_4'g_0$$

on the second output of the terminal cell. (It may further be noted that this realization follows the previous forms in that only one independent variable is accommodated per cell. If this requirement is relaxed the top two cells are seen to be redundant and only the remaining two cells are necessary.) It further follows that edge-fed rectangular arrays can be constructed utilizing such cells (again simply bussing the $x$-input through to provide the intercolumn connections, and utilizing a Shannon-type composition on the last $n-4$ variables) that would exhibit a growth rate of $Cn2^n$, where $C$ is no greater than 1/16. Thus an improved growth rate compared with the two-input cell is readily exhibited, but it must be remembered that the individual cells are more complex.*

*It also immediately follows that array growth rates characterized by any arbitrarily small value of $C$ can be constructed simply by augmenting the number of vertical rails to the required number. It also follows that in general the individual cell complexity would rapidly increase.

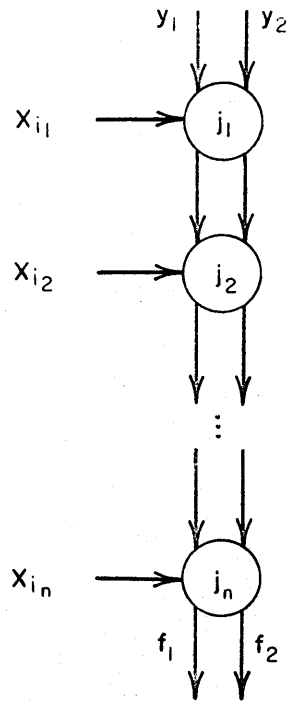



Figure 3. Basic two-rail cascade.

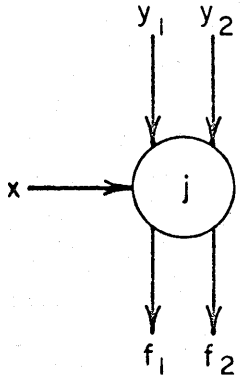



Figure 4. The two-rail cell.

However, the cascade of Fig. 5 exhibits another property of two-rail cells that can be utilized to enhance even further the number of functions realizable in a single column. In particular, the last cell of the cascade has two independent functions of the preceding variables presented on its vertical inputs. Thus the action of the last cell can be selectively applied to portions of the previously generated results; in the example shown the last variable can act selectively on the functions $g_1$ and $g_0$ to obtain $x_4g_1$ and $x_4'g_0$ before composing these in turn for the final result. This capability suggests a different view of the possible tasks of the two rails of a two-rail cellular cascade. Specifically, it is possible that one of the rails could be designated as an "accumulator" to which partially derived results can be added (logically) after they are computed in the functional line of the other rail.

For example, it is well known that arbitrary functions can be composed by simply adding the component minterms. In the two-rail cell it is possible to do this directly. Thus, again let $f_1$ and $f_2$ refer to the two cell outputs and $x$, $y_1$ and $y_2$ to the inputs, and assume that $f_1$ can be specified as any one of

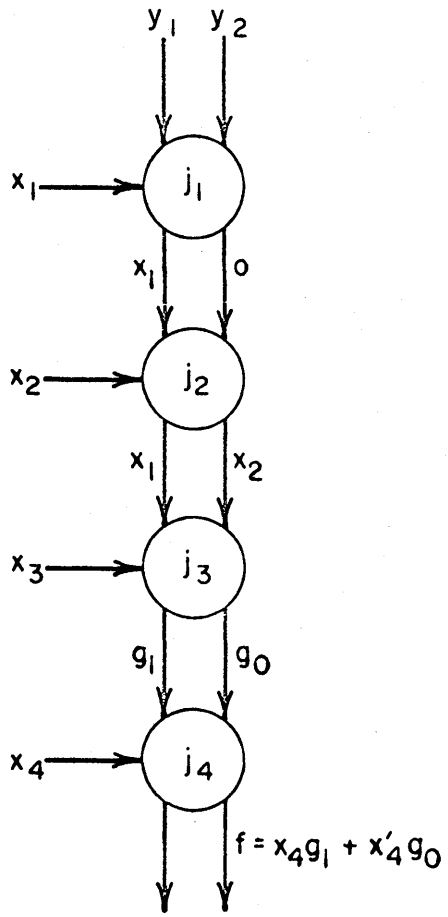



Figure 5. Realization of arbitrary four-variable function.

the set of functions ($xy_1$, $x'y_1$, 1) and that $f_2$ can be specified as any one of the set ($y_2$, $xy_1 + y_2$, $x'y_1 + y_2$). Then any arbitrary minterm (or prime implicant) can be formed on the $f_1$ leads and, when completed, it can be added to the sum being accumulated on the $f_2$ leads. At the same point, a new minterm can be commenced on the $f_1$ leads. It is apparent that the resulting cascades will be redundant in the sense that the variables are required at more than one cell; it also follows immediately that any function is realizable in a single cascade of such cells.

If it is assumed, then, that a complete minterm expansion for a function of $n$ variables is utilized in a single two-rail column, the total number of cells, $N$, required is

$$N < n2^n.$$

In terms of the number of cells required, this growth rate is no improvement on the most elemental rectangular array that also utilizes a minterm expansion. It does, however, achieve this rate within the desired single-column restriction.

Alternative realizations to the basic two-rail cell described above are possible and can be utilized to reduce the cell count somewhat for the functionally complete column.

In the first place, if the functions producible on the second rail are augmented to include the complements of those prescribed above, then the minterm expansion can be utilized to realize either $f$ or $\bar{f}$, whichever is simplest. If a final complementation is necessary to produce the specified function, it can be accomplished in the final cell of the column. Since the number of minterms in $f$ and $\bar{f}$ is just $2^n$, it follows that

$$N \leq \tfrac{1}{2}\, n2^n = n2^{n-1}$$

for such a cascade. The same end can be achieved by augmenting the cell functions so that a product-

of-sums can accumulate in the second rail.

The same growth rate can also be achieved by a two-rail cell based upon the exclusive-OR expansion

$$f(x_1, \ldots, x_n) = c_0 + c_1x_1 + c_2x_2 + \ldots + c_nx_n + c_{n+1}x_1x_2 + \ldots + c_{2^n-1}x_1x_2x_3 \ldots x_n$$

In this case, if the cell oputs can be selected from (again referring to Fig. 4):

$$f_1 : (x, xy_1, 1)$$
$$f_2 : (y_2, y_2 + xy_1)$$

then, as in the minterm case, each product term can be formed in the first rail, to be added or not to the accumulating subfunction in the second rail, depending on the value of the particular binary constant $c_i$. Since the constant $c_0$ can be accommodated by one of the initial boundary constants at the top of the column, only one cell is required for each variable occurrence. Hence the total number of cells can be enumerated as

$$N = \sum_{i=1}^{n} i \left(\frac{u}{i}\right) = n2^{n-1}.$$

Clearly other two-rail cascades can also be developed based on other expansions. For the realization of particular functions the use of prime implicants instead of minterms will suffice, as will various multilevel combinations (that is, greater than the usual two-level AND-OR configuration) that reduce the number of variable occurrences for the particular function to be realized. Since these realizations will result in nonstandard variable orderings and hence are not appropriate for the composition of horizontal-bus rectangular arrays, they will not be further discussed here.

*Efficient Two-Rail Compositions*

Since the two-rail cascade is functionally complete, i.e., any function can be realized within the extent of a single cascade, two minimization questions next become of significant interest. The first seeks an algorithm for the minimum length realization of any given function; the second seeks a minimum form of cascade that will be appropriate for a large class of functions, e .g., canonical cascades which, when the cells are appropriately specialized, will realize any member of the entire class of functions. The latter question will be briefly examined here in terms of efficient cascade realizations for arbitrary $n$-variable functions, where efficiency is measured in terms of cascade length. The development of fixed-variable-order cascades suitable for the class of all functions of $n$ variables is particularly appropriate to the design of rectangular, horizontal-bus arrays, where all the input variables are applied to the columns of the array in the same order, even though redundantly. In such arrays, designs that have been minimized for each function realized in the individual columns generally will need to be augmented in order that each column can be driven within the common variable ordering that feeds the entire array.

Since the minimization criterion of interest is the length of the cascade (i.e., the total cell count), it follows that the complexity of the individual combinational cells is not at issue and is subject only to the logical constraints implied by the two-rail structure. That is, each cell will initially be assumed capable of developing, independently on both of its outputs, any of the three-variable functions of its inputs. This point of view is entirely consistent with the original assumptions made regarding the single-rail cascades; steps directed toward reducing this implied individual cell complexity will be discussed in the following section.

Since each cell is assumed to possess complete three-variable capabilities, the determination of the minimum-length three-variable cascade is trivial, and is, of course, one. Almost as obviously, the minimum length four-variable cascade is two, although it is of special interest because it represents the last instance in which the value is known to be minimum. A particular instance of such a four-variable cascade is indicated in Fig. 6 and is based upon the general exclusive-OR expansion mentioned previously, that is

$$f(x_1, x_2, x_3, x_4) = g_1(x_1, x_2, x_3) + x_4g_2(x_1, x_2, x_3)$$

where $g_1$, $g_2$ are arbitrary functions of three-variables and are realized on the outputs of the first cell in the cascade. This particular expansion is selected with an inductive construction in mind, for it suggests a means for introducing a new variable to the scope of a problem with the expenditure of only one cell

for that particular variable. Indeed any arbitrary function of $n$ variables can be expanded as

$$f(x_1, \ldots, x_n) = g_1(x_1, \ldots, x_{n-1}) + x_n g_2(x_1, \ldots, x_{n-1})$$

where $g_1$ and $g_2$ are arbitrary functions of their arguments, suggesting the general form of realization of an $n$-variable cascade as indicated in Fig. 7. In this figure, the cascade A represents an efficient (and presumed known) realization of the $(n-1)$–variable function $g_2$. (The symbol $\phi$ on an output indicates that the particular function realized thereon may be arbitrarily specified since it does not contribute logically to the result.) The pertinent output of cascade A provides one input for the cell which introduces the new variable, $x_n$, which in turn provides an "accumulated" input for cascade B. The function of cascade B, finally, is to develop the function $g_1$ as efficiently as possible, and to accumulate it on the second rail in order to form the complete $n$-variable function. An economical realization of the B cascade can also be obtained by utilizing the exclusive-OR expansion, this time on only $n-3$ of the variables; thus using $n = 5$ as an example,

$$g_1(x_1, x_2, x_3, x_4) = h_0(x_1, x_2) + x_3 h_1(x_1, x_2) + x_4 h_2(x_1, x_2) + x_3 x_4 h_3(x_1, x_2).$$

A realization of this B cascade corresponding to $n = 5$ is shown in Fig. 8. Since two cells are utilized for each $h_i$ function, plus one more for each variable occurrence of the expanded variables, it follows that in general such a B cascade requires

$$N_B = 2 \cdot 2^{n-3} + \sum_{i=1}^{n-3} \left( \frac{n-3}{i} \right) = (n+1)2^{n-4}$$

cells together. The number of cells in the complete $n$-variable cascade then is certainly bounded by the recursion relation.

$$N_n \leq N_A + N_B + 1 = N_{n-1} + (n+1)2^{n-4} + 1.$$

It follows immediately that the number of cells required for a canonical five-variable cascade, for example, need not exceed fifteen cells. Further simplifications can be made, however, for all cases. Since it is immaterial which of the $n-3$ variables is used in the expansion of $g_1$, it can always be arranged that the last variable used at the bottom of cascade A can provide one input to the top of cascade B, saving one cell. Furthermore, since the particular function $h_0(x_i, x_j)$ is not multiplied (logically) by any of the others, it follows that $h_0$ and any one of the other two-variable functions can be realized in the same two cells of the cascade, thus saving two more cells. (This simplification is indicated in Fig. 9 which can be substituted directly for the bottom five cells in the B cascade of Fig. 8.) Thus for all $n$, the upper bound on the number of additional cells required can be reduced to

$$N_n \leq N_{n-1} + (n+1)2^{n-4} - 2.$$

This represents the best bound that has proved valid for all $n$. Examination of Fig. 8, however, reveals

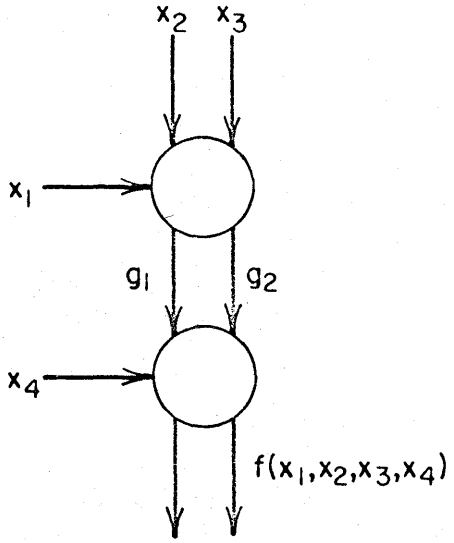



Figure 6. Arbitrary four-variable cascade.

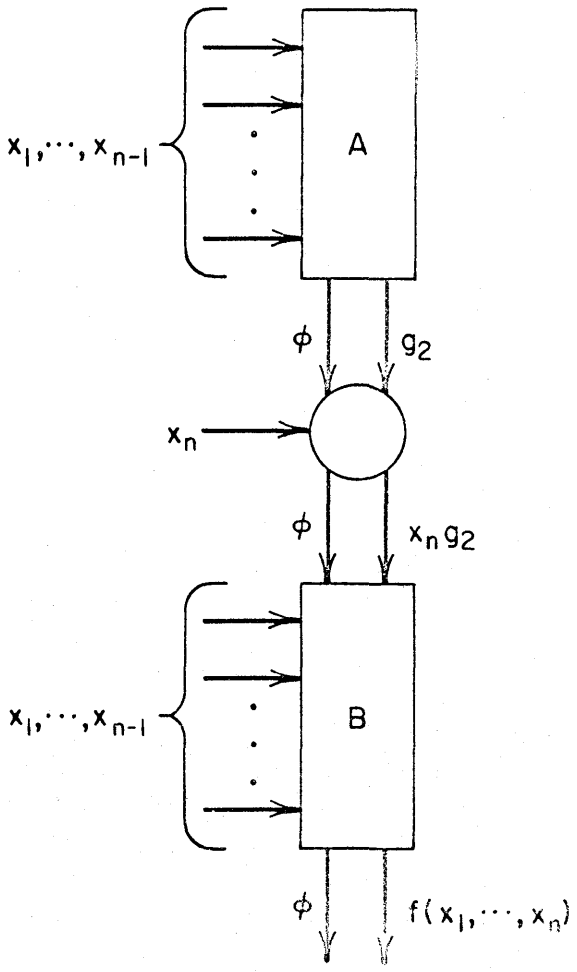



Figure 7. Construction of $n$–variable function.

other $\phi$ outputs that suggest even further savings. For example, the $\phi$ output of the fourth cell in the cascade could be specified as $x_4$ if that variable could be used logically by the fifth cell in the cascade possibly resulting in the elimination of another cell. It turns out that such savings can be easily shown for $n = 5$, and result from a nonuniform expansion of the function $g_1$, that is, an expansion such that the different $h_i$'s may be functions of different pairs of variables. For example, an alternative expansion for $g_1$ for $n = 5$ is

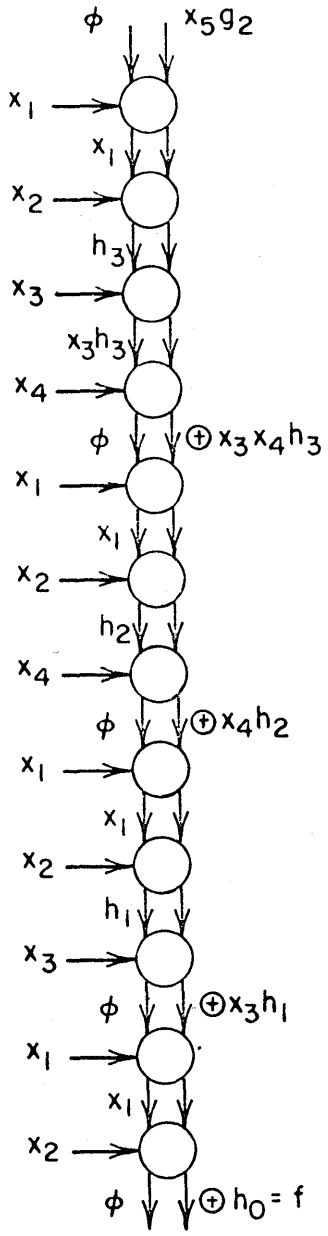
$$g_1(x_1, x_2, x_3, x_4) = \underset{(2)}{h_0(x_1, x_2)} + \underset{(3)}{x_3 h_1(x_1, x_2)} + \underset{(4)}{x_4 h_2(x_2, x_3)} + \underset{(1)}{x_1 x_4 h_3(x_2, x_3)}$$

and if this expansion is realized in the B cascade in the order indicated by the numbering of the terms above, then in each instance a cell can be saved in the transition from one term to the next in the cascade. In Fig. 10 a five-variable cascade is specified that utilizes this B cascade for a total of 10 cells for the entire cascade, and achieves the minimum known length possible for such a general cascade.

Whether such savings, which accrue from appropriate nonuniform expansions, can be achieved for all $n$ is not known. For all values of $n$ through $n = 8$ the existence of such expansions has been shown, however, and if this result can be extended to all $n$, then the upper bound can be reduced as indicated by the recursion

$$N_n \leq N_{n-1} + (n-1)2^{n-4}$$

On the basis of these results it is conjectured that




NOTE: The notation "⊕H" on an output arrow indicates that at that output the function H is added to the previously accumulated function on the second rail.

Figure 8. A realization of the B cascade. The notation "⊕*h*" on an output arrow indicates that at that output the function *h* is added to the previously accumulated function on the second rail.

the above bounds holds for all *n*.

As a summary, the best known values of $N_n$ are shown in Table 1 for low values of *n*. For comparison, the direct minterm expansion values are also shown.

Table 1. Upper-Bounds for Two-Rail Cascades.

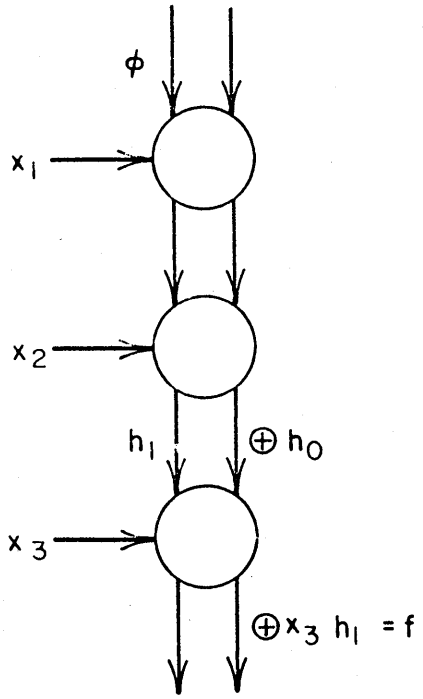



Figure 9. Simultaenous realization of $h_0$ and $h_1$.

| $n$ | Minterm $n2^{n-1}$ | Bound $N_{n-1} + (n-1)2^{n-4}$ |
|---|---|---|
| 3 | 12 | 1 |
| 4 | 32 | 2 |
| 5 | 80 | 10 |
| 6 | 192 | 30 |
| 7 | 448 | 78 |
| 8 | 1024 | 190 |
| 9 | 2304 | 446 |
| 10 | 5120 | 1022 |

*Reduced-Cell Compositions*

Although to this point the only cost criterion has been the length of the two-rail cascade, it is apparent that the individual cell complexity is also of great practical importance. Certainly the general two-output, three-variable cell will be significantly more complex and costly than has previously been described for the single-rail case.[1]

In Fig. 11, however, a summary is found of the six cell types that have been utilized in the efficient two-rail canonical realizations of the previous section. Of these six only the first type, in part (a) of the figure, requires the complete generality assumed for the two-rail cell. This type is only used in the first cell of any such cascade, however, and the remaining types of cells are required to realize only very simple three variable outputs. For present purposes it will suffice, then, to demonstrate the avoiddance of cells of the type in part (a), in particular by realizing the necessary functions in terms of the remaining five cell types, which can then be thought of as a sufficient, reduced cell set.

Although the general cell is used only as the first cell of every cascade, it is convenient to replace the first two cells in every cascade (as indicated by part (a) of Fig. 12 which represents the first two cells of the cascade of Fig. 10) with the five-cell cascade of par (b) of Fig. 12. It can be verified that these two cascades are functionally identical according to the expansion

$$g_2(x_1, x_2, x_3, x_4) = h_0(x_1, x_3) + x_2h_1(x_1, x_3) + x_4(h_2(x_1, x_2) + x_3h_3(x_1, x_2)),$$

and for the price of three additional cells in part (b) of the figure, only reduced-cell types are utilized. ized.

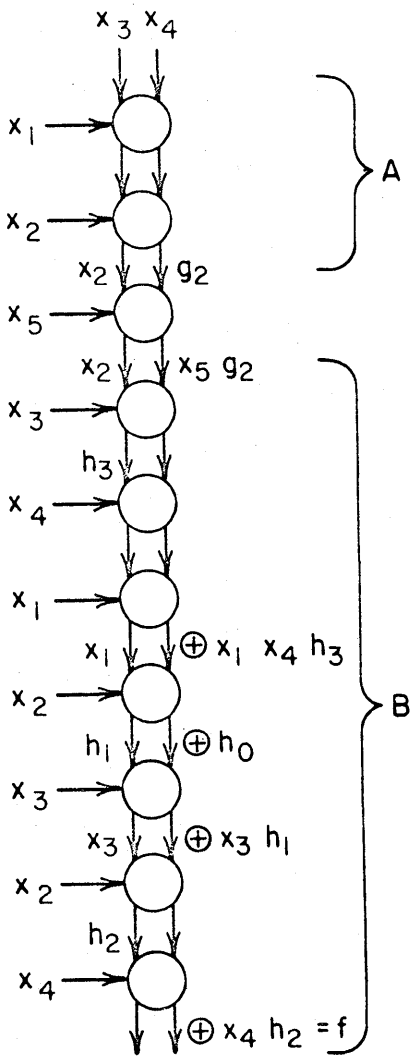



Figure 10. The best-known five-variable cascade.

Thus upper bounds on the number of reduced type cells required for canonical two-rail cascades can be obtained simply by adding three cells total to the more general results previously obtained.

## BOUNDS FOR RECTANGULAR ARRAYS

Since the single-column for two-rail cells has been shown sufficient for the composition of arbitrary functions, it is of interest to ask whether the growth rate of rectangular arrays composed of such columns is thereby enhanced. As one way of structuring such an array, let us consider the conventional Shannon type decomposition

$$f(x_1, \ldots, x_n) = \sum_i g_i(x_1, \ldots, x_m)\,(x_{m+1} \ldots x_n)_i$$

where an arbitrary function of $n$ variables is expressed canonically in minterms of $n-m$ of the variables, each with a coefficient that is an arbitrary function of the remaining $m$ variables. If the $m$-variable functions are realized in a single column in a minterm two-rail realization, then the number of cells in a column, including one for a collector row, will not exceed

$$m2^m + n - m.$$

Since there are no more than $2^{n-m}$ such columns necessary, the total number of cells in the entire array, as suggested in Fig. 13, will not exceed the product, that is

$$N \leq m2^n + (n-m)2^{n-m}.$$

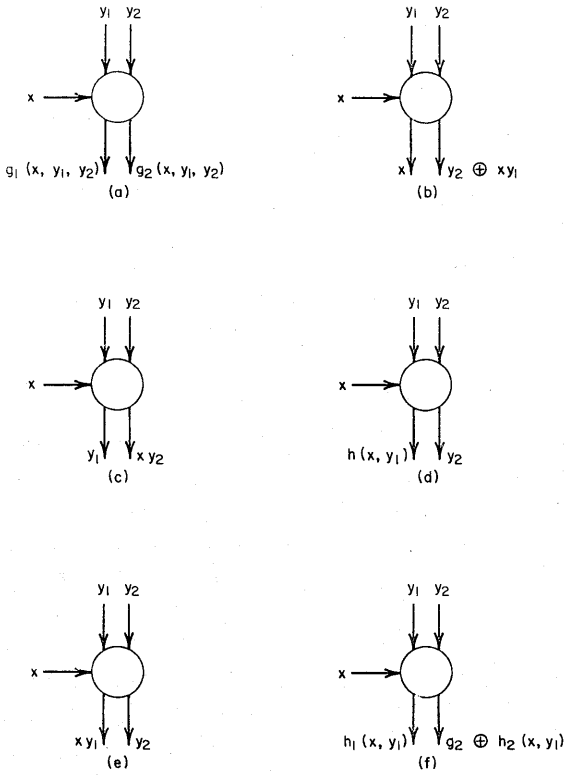



Figure 11. Two-rail cell types.

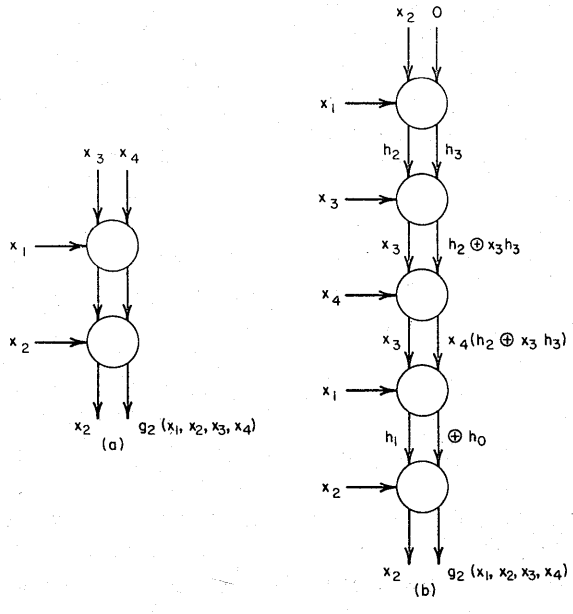



Figure 12. Reduced-cell version of cascade A.

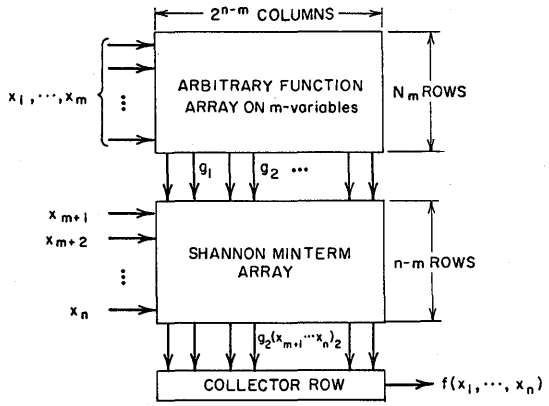



Figure 13. Construction of rectangular array.

If $n$ is chosen so that $m = \log n$ then this growth rate varies asymptotically as $(\log n)2^n$ which is a functional improvement over the rate for previously constructed arrays of the edge-fed, horizontal-bus type.

It should also be noted that the selection of $m$ can also be motivated by the aspect ratio of the resulting rectangular array; in particular the situation where the height varies linearly with $n$, while the width varies exponentially can be avoided by the proper selection of $m$.

Similarly, the more efficient two-rail cells can be utilized directly to obtain reduced cell counts for canonical rectangular arrays of various desired aspect ratios. For example, for $n = 11$ a canonical array could be composed so that $m = 5$, as in the above. Since the five-variable columns require only 10 cells, the array height (counting one row as a collector row) need not exceed 17 cells; and the width would not be greater than $2^6$, corresponding to the Shannon expansion on the remaining 6 variables. Hence the total cell count in the array would be 1088 cells in a $17 \times 64$ array. On the other hand, a nearly square array measuring $36 \times 32$ cells or 1152 cells altogether, results from selecting $m = 6$ in the 11-variable case.

## CONCLUSIONS

It has been demonstrated that the two-rail cellular cascades not only greatly increase the number of functions realizable in a cascade of cells, but that they are indeed functionally complete. No greater interconnection complexity is needed to insure that the columns of horizontal bus cellular arrays can be made to realize as broad a class of functions as may be desired for the solution to the particular logical design problem.

Furthermore, fairly efficient realizations can be specified in canonical form utilizing only a fairly simple class of three-variable functions that need be specified on the outputs of the individual two-rail cells. The resulting columns can be composed into rectangular arrays that exhibit considerable flexibility in the specification of aspect ratio.

Sevral lines for further inquiry are suggested by these results, not the least of which is the remaining question of whether the derived bounds on column length are firm, and whether further reductions can be made. It is quite probable that further reductions, if possible, will require considerably more cell complexity than the designs suggested herein.

Also, the development of minimal realizations for specific functions is of interest, and may ultimately result in rectangular arrays with fewer total cells required.

Further, the two-rail cell, taken as a starting point, suggests other interconnection possibilities besides the individual columnar type of composition. For example, each cell, rather than communicating only with the cell below it in an array (even though with two leads), might instead provide an output for the cell below and one to the right, or any of a variety of other ways. Such interconnection structures provide the possibility for tree-like compositions within the rectangular arrays and may significantly increase the efficiency of utilization of the basically two-dimensional structures. Finally, the generalization of the two-rail cell to three or more rails is an obvious next step. Although the cell complexity in the completely general interpretation would increase phenomenally, experience indicates that selected subsets of the possible functional specifications can be very useful.

## ACKNOWLEDGMENTS

The author gratefully acknowledges the constructive participation of Dr. Robert C. Minnick in all aspects of this study, the contributions of Dr. Bernard Elspas in the development of efficient cascades, and the many helpful discussions with his other colleagues at Stanford Research Institute.

## REFERENCES

1. R. C. Minnick, "Cutpoint Cellular Logic," *IEEE Transactions on Electronic Computers,* vol. 13, no. 6, pp. 685-98 (Dec. 1964).

2. K. K. Maitra, "Cascaded Switching Networks of Two-input Flexible Cells," *IRE Transactions on Electronic Computers,* vol. 11, no. 2, pp. 136-43 (Apr. 1962).

3. J. Sklansky, "General Synthesis of Tributary Networks," *IEEE Transactions on Electronic Computers,* vol. 12, no. 5, pp. 464-9 (Oct. 1963).

4. S. Y. Levy, R. O. Winder and T. H. Mott, Jr., "A Note on Tributary Switching Networks," *IEEE Transactions on Electronic Computers,* vol. 13, no. 2, pp. 148-151 (Apr. 1964).

5. H. S. Stone, "On the Number of Equivalence Classes of Functions Realizable by Cellular Cascades," presented at the Conference on The Impact of Batch-Fabrication of Future Computers, Los Angeles, Apr. 6-8, 1965.

6. R. C. Minnick, "Application of Cellular Logic to the Design of Monolithic Digital Systems," presented at the Symposium on Microelectronics and Large Systems, Washington, C.D., Nov. 17-18, 1964.

7. H. S. Stone and A. J. Korenjak, "Canonical Form and Synthesis of Cellular Cascades," *IEEE Transactions on Electronic Computers,* Dec. 1965.

8. R. C. Minnick, "Cobweb Cellular Arrays," this volume.

9. G. G. Lendaris and G. L. Stanley, "On the Structure-Dependent Properties of Adaptive Logic Networks," G.M. Defense Research Labs., Santa Barbara, Calif., TR 63-219 (July 1963).

10. M. Yoeli, "Generalized Maitra Cascades," *IEEE Transactions on Electronic Computers,* Dec. 1965.

# THE ASSOCIATIVE MEMORY STRUCTURE

B. T. McKeever
*General Electric Computer Laboratory*
*Sunnyvale, California*

## INTRODUCTION

The development of the Associative Structure Computer (ASC) was partly motivated by the opportunities and problems of designing, manufacturing, interconnecting and using batch fabricated devices, particularly cryotrons. Some of these points are discussed in the third paper of this series, "The Associative Structure Computer," which describes the machine organization. ASC is constructed with only two types of modules. One module is described in the second paper of this series, "The Associative Switching Structure." This paper, the first of the series, discusses the other module, the Associative Memory Structure.

The following section sketches the general organization and background in which the Associative Memory Structure operates. The succeeding section describes the Associative Memory Structure and the primitive information manipulation operations that are built into the hardware. It first describes the primitives individually in two classes, bit manipulation operations and word select operations; then describes the primitives in combinations as modes of accessing information in the Associative Memory Structure. The next section presents six fairly detailed examples of the use of sequences of combinations of primitives. It is hoped that these examples will clarify the description of the Associative Memory Structure and its primitive operations, show from several approaches how the primitives work together and illustrate a less familiar technique, parallel sequential logic, for composing computer functions. The examples also illustrate that the small set of primitives is logically powerful and that the Associative Memory Structure has broader applications than just serving as a search memory.

## GENERAL ORGANIZATION

To try to match the general characteristics of batch fabrication technologies, compromises were made in the logic design and machine organization of ASC and so its design does not follow a purely cellular approach—a uniform array of identical cells. The design approach of ASC was partly influenced by an existing and successful cellular system which has about a half billion cells, performs about a quarter million operations per second and handles around a hundred thousand problems simultaneously. It is neither a micro array of gates nor a macro array of computers. It is an array of arrays, not all of which are the same. Even within arrays there are inhomogeneities. Basically the arrays and subarrays consist of some number of signal collecting, buffering and distributing subarrays or cells closely asso-

ciated with some number of information manipulation, transformation and storage subarrays or cells which can operate in parallel, independently of each other. This system is the telephone network.

The design of the Associative Structure Computer tries to take advantage of the concept of hierarchy both in the cellular structure at each level and in the communication/interconnection scheme. ASC is a macro cellular array using two modules rather than just one. Both modules are based on associative logic and both are capable of logic and storage, but one (Associative Memory Structure) is slanted toward storage and the other (Associative Switching Structure) toward logic. Each of the modules is in turn a micro cellular array with two kinds of cells rather than just one. One kind is the Associative cell which has the ability to perform storage and logic actions in parallel with and independently of other cells. The other kind is the Logic Accumulator cell which can collect and combine signals and then distribute the results to an associated group of cells. The two levels of cellular approach are apparent, but it has proven valuable in both levels to avoid a purely cellular approach of a uniform array of a single type of cell and instead borrow from the functional block approach by employing two kinds of working units at each level. This is as much for communication purposes as it is for efficient division of labor by specialization.

Instead of first designing circuits and then trying to find an effective interconnection pattern, the approach can be reversed. First a cellular approach is used to establish a general interconnection of all cells and arrays. Then the logic design problem becomes one of selectively introducing local disconnections. Neighborhood or point-to-point wiring is avoided as much as possible and the interconnection scheme is predominantly bussing which shares the cost of interconnections and associated drive/sense circuitry over many sources and receivers. The resulting array pattern lies between a one- and two-dimensional structure. Cells are connected to form a string of functional blocks. The terminals of these strings are bussed into a higher order string whose terminals are connected into a still higher order string, the chain, which is a major subsystem. The chains are in turn connected at their terminals to form the basic system. Appropriate use of the Associative Switching Structure in the interface of the chain allows such things as peripherals, remote terminals or even existing computer systems to be disguised as chains with nonstandard properties and to be connected to ASC at the highest string level.

Associative logic is used as a basic form for building functions. ASC works with a logic form which compares a single input with a stored constant and produces an output for either equality or inequality. Several of these logic forms operate independently and in parallel. Then results for a group enter a logical accumulator of some kind and an overall group result is calculated. This gives an inherent mixture of logic and storage at a low level in the machine organization. The string of strings hierarchy in the communication/interconnection scheme is used to extend associative logic to the entire machine.

Logical flexibility is emphasized. Only a small set of primitive information manipulating operations are implemented in hardware. Stored logic is used to build up macro functions. The usual limiting of system speed by the read rate of the control memory is circumvented by distributing the stored logic. Rather than storing a single clock gating action in each control word, each word stores instead a cycle code that is interpreted by Associative Switching Structures to generate a sequence of gating actions which is long enough to allow the control memory ample time to read the next word.

The dominant function pattern is parallel sequential logic in two forms. In one form, the function is broken up into pieces representing possible logical cases which are passed against all members of a stored data set—sequential by case, parallel by item. This reduces removals, external modifications and reinsertions in memory. The other form completes the symmetry by passing individual items of data against a set of functions—sequential by item, parallel by case. The functions are broken up into logical cases and expressed in the form of stored logic tables which imitate an easily modifiable diode matrix. This reduces dedicated hardware subject to faults and reduces reentering data for different functions.

## DESCRIPTION OF THE ASSOCIATIVE MEMORY STRUCTURE

The Associative Memory Structure has two kinds of cells—the associative cells making up the storage section of a word and the logical accumulators

making up the word control. There are two kinds of built-in primitive operations—bit manipulations and word selects. Figure 1 shows a group of words, each made up of a number of associative cells, each

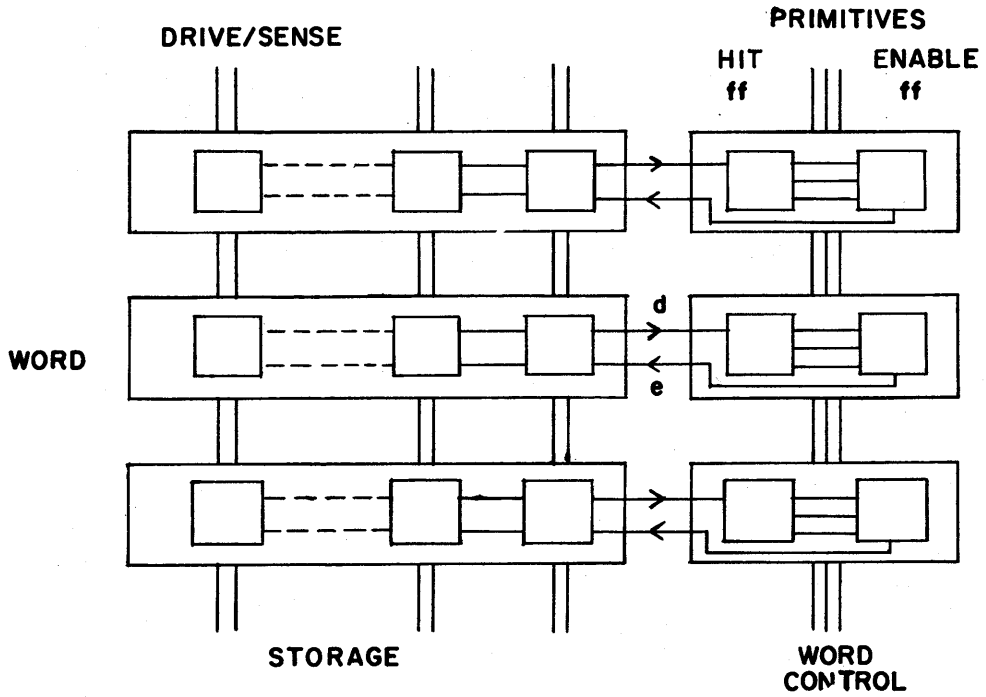



Figure 1. Block diagram of associative memory structure.

a bit in the word. The drive/sense lines of corresponding bit positions are wired together to form an input/output bus. The detect/enable lines of successive bits within a word are wired together to form a bus linking the storage section with the corresponding word control.

*Bit Manipulations*

*Search.* One of the bit manipulations is the Search operation. The drive/sense bus is controlled by a Contents register and a Mask register. In each bit position where the Mask is one, the value of the Contents register, one or zero, is placed on the drive line. Where the Mask is zero, the drive line is neutral. The associative cell may store one of three values: one, zero, and no comment. Figure 2 shows the truth table of an associative cell responding to a Search. Note that only for the cases of a driven one and stored zero or for a driven zero and stored one is a disagreement signal generated. Combinations of a driven one and stored one or a driven zero and stored zero produce agreement. When either the drive line is neutral or the associative cell stores no comment, agreement is produced.

SEARCH

| | | DRIVE 1 | 0 | n |
|---|---|---|---|---|
| STORE | 1 | a | d | a |
| | 0 | d | a | a |
| | X | a | a | a |

Figure 2. Truth table for search.

During the search, each associative cell performs its computation of agreement or disagreement in parallel with and independently of all other associative cells. If any associative cell disagrees, the detector bus assumes the zero state. If all cells agree, the detector is one.

The final phase of the Search operation consists of transferring the value of the detectors to the Hit flip-flops (Hit ff) in the corresponding word controls. There are two control lines, transfer one and transfer zero. If both are turned on, the value of each detector is copied into the corresponding Hit ff. If only transfer one is turned on, then the Hit ff

is set to the one state if the detector is one, and the Hit ff retains its initial state if the detector is zero. If only transfer zero is turned on, then the Hit ff is reset to the zero state if the detector is zero and the Hit ff retains its initial state if the detector is one. If the Hit ff were initially zero and a sequence of searches using only transfer one were applied, then at the end of the sequence, the Hit ff would be zero if all the searches failed, one if any succeeded; the searches would be ORed. If the Hit ff were initially one and a sequence of searches using only transfer zero were applied, then at the end of the sequence, the Hit ff would be one if all the searches succeeded, zero if any failed; the searches would be ANDed. Combinations of these sequences could be applied. In addition to association between bits within a word, we have the capability of association between searches in a sequence.

The other two bit manipulations, Read and Write, differ from the Search in one important respect. During a Search, every word participates. During a Read or Write, only words with the enable bus in the one state participate. Words which are not enabled do not affect and are not affected by the drive/sense bus.

*Write.* The Write operation is relatively straightforward. A valuable feature is individual bit write. If the drive line is in the one or zero state, the associative cell takes on the driven state in any and all enabled words. If the drive line is neutral (or if the word is not enabled), the associative cell retains its initial state. Any bit, therefore, can be used as a tag ff to store the result of a search or sequence of searches. Since these tags are bits just like others in the word, their values can be specified when desired in subsequent searches. This allows formation of quite general associations between searches whether or not they are in sequence and whether or not the value of a search result or its complement is of interest.

The associative cell is discussed as if its circuitry allows three physical states to represent one, zero, and no comment. Equally valid circuit approaches may be based on circuitry which couples adjacent bits within a word to represent the three states or on circuitry which couples adjacent bits between two consecutive words (and their word controls) to allow the lower bit to act as a mask or bypass for the upper bit. Depending on which circuit approach is taken, a variation of the normal write cycle will be used to sort the third state. For ease of explanation, it will be assumed that in a block of words and bits having all three states, a separate clear-write operation stored no comments everywhere within the block and then the normal write cycles were used to superimpose patterns of one and zero.

*Read.* The Read operation is really a logical comparison like the Search except that only enabled words participate instead of all words, that the results are collected by the sense line along bit positions instead of along the word and that agreements are of interest instead of disagreements. Figure 3 shows the truth table for the Read operation. Note

READ

| | | DRIVE | | |
|---|---|---|---|---|
| | | 1 | 0 | n |
| STORE | 1 | 1 | 0 | 0 |
| | 0 | 0 | 1 | 0 |
| | X | 0 | 0 | 0 |

Figure 3. Truth table for read.

that only for the case of a driven one and stored one or a driven zero and a stored zero is there a positive output on the sense line. For a driven one and stored zero or a driven zero and stored one, there is zero output. For a neutral drive or a stored no comment, there is zero output. If more than one word is enabled, then along each bit position, the sense line will OR one outputs of the associative cells during a read.

If exactly one word is enabled, then by reading for ones (placing one on all drive lines) the result is a normal read. Reading for zeros (placing zero on all drive lines) produces a complemented read. Placing the bit values of a word on the drive lines produces the bitwise comparison of the external word and the enabled internal word (noted as A:B, reading A with B, identity function of A and B, complement of the exclusive or).

If for some reason it was intended to enable several words, then reading for ones will produce the bitwise OR and reading for zeros will produce the bitwise NAND which can be complemented externally to produce the AND. Also, all bit positions but one can be held in the neutral state by placing zeros in the mask register and that one bit position can

be read first for ones, then for zeros. The two reads will indicate whether a given bit position contains only ones, only zeros or both ones and zeros for use in a Lewin type algorithm. It is also possible that neither ones nor zeros are present, which will indicate that in that bit position of all enabled words, the no comment is stored. If for some reason it is desired to completely read a word which can have stored no comments, then two reads, a read for ones and a read for zeros, must be performed and bit positions which gave no output for either case hold no comments. More often, multiple read will be used when several words are enabled at the end of a logic sequence where each word is the computed result for one subfield of the desired output and in each word, the bits of subfields of the output which belong to other words are filled with no comments. In this case, the transparency of no comment to the read operation is a distinct advantage.

From the viewpoint of the overall connection scheme, this third state, the no comment, gives the ability to locally introduce disconnections. This in turn allows breaking up functions into parts which can be treated independently and exercised in parallel without interference between the parts. When blocks of the associative memory are used for logic purposes, the three state storage allows imitation of diode logic networks with stored ones and zeros acting as a connection of variables or their complements and stored no comments acting to disconnect variables from a given term.

*Notation for Bit Manipulation.* It might be useful to introduce some notation for search, read and write. A search employing both transfer one and transfer zero is written as S(a/A, 1/Data). The (a) refers to a value placed in the contents register. The (A) refers to a value placed in the mask register in bit positions associated with (a). Sometimes ("a") is used, meaning the name of (a) as distinguished from the value (a) itself. As many terms of specification as desired may be used, separating each by commas and using the slash to separate contents and mask values. The mask is sometimes specified by the name of a defined field such as (Data) rather than directly with a bit pattern. The use of a single bit value (a constant or the output of an expression) over a multibit field implies that value is placed in all bit positions. S1 (a/A) denotes a search only using transfer one. S0(a/A) denotes a search only using transfer zero. W(a/A) denotes write (a) into field A of all selected words. WX(1/field 2, 0/field 2) denotes the clear-write, storing no comments in field 2 of all selected words. R(a/A) reads field (A) with the value (a).

*Word Selection*

The approach taken in ASC emphasizes logical flexibility. In the word control, the function of keeping track of the response of a word to searches is separated from the function of selecting a word to be enabled for manipulation in read/write operations. The Hit ff, already discussed, records the history of search responses. There is an Enable ff whose value is transferred to the enable bus in all words whenever a read or write occurs. Setting the enable bus in a word resets the Hit ff. If the Enable ff is zero, a read or write will be ignored by the word and the Hit ff will be unchanged. In all cases, the state of the Enable ff is not changed by read, write or search operations and so, any number of reads and writes can be performed without losing the word. There are a variety of word select operations which transfer information between the Hit and Enable ffs.

There are five word select operations—Select First Hit (SFH), Partition (PRT), Select Next Word (SNW), Select Previous Word (SPW) and Select All Hits (SAH). Select First Hit and Partition employ the same circuit but use different initial conditions. In most hardware implementations, this circuit is expected to be very slow compared to the other word select operations and is used sparingly—when there is no reasonable alternative or when there is a clear advantage in speed.

*Select First Hit.* Select First Hit (SFH) assumes the Enable ffs are initially zero and that there are a number of words with a one in the Hit ff. Geometry and the direction of certain control signals impose a physical order on the words—top to bottom, first to last, previous to next. Action of the operation can best be described as imagining a pointer which quickly scans the Hit ffs, top to bottom. As long as the pointer encounters a zero, it goes on to the next word. As soon as the pointer encounters one in a Hit ff, it sets the associated Enable ff to one and stops the scan. The Hit ff is unchanged. If, for some reason, not all the Enable ffs were initially zero, then we have two cases. As the pointer scans for the first Hit, it resets Enable ffs to zero in all words until the first Hit. Once the first Hit has

been found, the scan stops and therefore the state of Enable ffs below the first Hit will be unchanged.

When we have reason to expect multiple matches and must treat them one word at a time, then SFH allows resolution of the multiple matches. With the proper initial conditions (0 in all E ffs), assume a search results in multiple matches. Applying SFH, the first hit is selected, but the Hit ffs are unchanged. Then applying R or W, the first word is enabled for participation while the other hits remain unenabled. After R or W, SFH is applied again. Since the Hit ff of the first match word was reset to zero by the R/W, the second match word is now the first hit. The Enable ff of the first match word will be reset to zero and the Enable ff of the second match word, which is now the first hit, will be set to one. This procedure continues until all the match words have been processed.

There are two test signals available. Hit Test indictates if any Hit ff is one. Read Test indicates if any bit position produced a positive output (stored one and driven one or stored zero and driven zero) during a read.

*Partition.* Partition (PRT) assumes the Enable ffs are initially one and that there is exactly one hit. All the Enable ffs above the hit will be reset to zero. The Enable ff associated with the hit and all the Enable ffs below will be set at one. The memory will be partitioned into two blocks on the basis of the hit. If there is no hit, all of the words will be in the upper block. If there is more than one hit, division will take place on the basis of the first one and the other hits will be ignored. If not all the Enable ffs are one, then Enable ffs in the zero state below the first hit will remain zero. The Enable ff of the first hit will be forced to one in any case and all the Enable ffs above the first hit will be reset to zero.

Figure 4 shows a typical short sequence using PRT. Searching for Header-5 gives a hit at the top of the block. Partition selects all of the words from the header down. Write places ones in all the marker bits (M) from the header down. Searching for Trailer-5 gives a hit at the bottom of the block. The second PRT selects all of the words from the trailer down. The second Write restores all the marker bits from the trailer down to zero. This leaves ones in the marker bits of only the words bounded by the header and trailer. This allows a very general association between words as well as association between bits within a words and association between searches. This kind of sequence can be concatenated using additional marker bits to access structured information such as a condensed file or file index with a hierarchy of headers, sub-headers, sub-sub-headers, etc. This avoids the necessity of storing the header values for each level of the hierarchy in the same word along with the item label and data which could lead to difficult requirements such as thousand-bit words.

| Data | M | HE | |
|---|---|---|---|
| Header-5 | 0 | | |
| Element A1 | 0 | | S(Header-5/-),PRT,W(1/M) |
| Element A2 | 0 | | S(Trailer-5/-),PRT,W(0/M) |
| Element A3 | 0 | | |
| Element A4 | 0 | | |
| Trailer-5 | 0 | | |

Figure 4. Example of association between words.

*Select All Hits.* Select All Hits is the workhorse of the word selects. It is also the simplest and, for most hardware implementations, is expected to be the fastest select operation. The state of all Hit ffs is copied into all corresponding Enable ffs. The Hit ff retains its state.

Whenever the nature of the program indicates that exactly one hit (or none) exists and it is desired to process an individual word, SAH provides a much faster word selection than SFH. Whenever the nature of the program allows manipulation of a set of hits in exactly the same way independently of how many there are, SAH provides a means for selecting those words in parallel and performing the manipulation once for every hit simultaneously.

*Select Next Word and Select Previous Word.* Select Next Word and Select Previous Word are essentially the same function except that the direction of control information movement is reversed. SNW and SPW are fairly simple and reasonably fast. When Select Next Word is turned on, the states of all Enable ffs are transferred to the Hit ffs of the words immediately below (if a word is faulty, rewiring at the edges can remove power from the word and make the two adjoining words act as if they were directly connected). The state of the last Enable ff leaves from the bottom and is lost. The normal state input to the first Hit ff from the top is zero. Where noted as an exception, it may be a one. When the Select Previous Word is turned on, the states of all Enable ffs are transferred to the Hit ffs

of the words immediately above. The state of the first Enable ff leaves the top. The normal state input to the last Hit ff from the bottom is zero. When noted as an exception it may be a one.

*Access Modes*

Another way of looking at the word select operations, in conjunction with each other and with the bit manipulations, is to consider access modes. ASC has four modes: Direct Access, Indirect or Conditional Access, Inverted Access and Parallel Access.

*Direct Access.* Direct Access has two cases—random and sequential access. SFH can be used to provide random access when it is not known how many hits there are. SAH provides random access when it is known that there is one hit or none. After the starting point of a list or sublist of consecutive entries has been found in a random access mode, other entries can be found in a sequential access mode. Pairing SAH & SNW or SAH & SPW will cause the Hit and Enable ffs to act as a two-phase shift register, moving an access pointer from an Enable ff to the Hit ff of the word above or below with the SPW or SNW and then from the Hit ff to the Enable ff of the same word with the SAH. A common use of sequential access is reading lists such as instruction in a subroutine where the individual instructions are not labeled but access is started at a header word which contains the name of the subroutine.

*Indirect or Conditional Access.* Another mode is Indirect or Conditional Access. In the discussion of Partition used in conjunction with individual bit write for parallel tagging, access to a block of contiguous words conditional on the values stored in boundary words was illustrated. This form of conditional access can be concatenated using additional tag bits and searching on the original tag as well as second level boundary words and allows very general association between as well as within words. It also offers the possibility of accessing information using structural information about a data hierarchy to supplement incomplete knowledge of contents.

Another form of Indirect or Conditional Access is based on extending sequential access. This gives a less general but much faster association between adjacent words. This form can be based either on a SAH & SNW sequence or a SAH & SPW sequence. Figure 5 illustrates a SAH & SNW sequence. The initial search with double sided transfer finds all headers and establishes the starting points of all

```
H
    A1          S(H/-), SAH, SNW
    B1
    C4          SO(A1/-), SAH,SNW
    D6
H
    A2          SO(B2/-), SAH, SNW
    B2
    C3          W(C3/-)
    D6
H
    A1
    B2
    C5
    D2
```

Figure 5. Example of conditional access.

strings. The SAH & SNW makes the first word and only the first word of every string a hit and thus a candidate. Since the words above the headers did not satisfy the initial search, their Enable ffs are zero and therefore after the SNW, the Hit ffs of the header words are zero even though the Enable ffs of the headers are still one. The search for A1 in the first word is satisfied by the first and third string and therefore the single sided transfer leaves their Hit ffs still one. The first word of the second string does not satisfy the search and therefore the second string goes out as a candidate. The subsequent SAH sets the Enable ff of the first word of the first and third string, leaves the enable ff of the first word of the second string zero and resets the Enable ffs of the headers since their Hit ffs were zero. The SNW makes the second word of all strings still active (first and third) candidates. The search for B2 in the second word of all strings is satisfied by the second and third strings but not the first. So the third stays on and the first goes out. Even though the second word of the second string satisfies the search, the single sided search does not set the Hit ff and therefore the string does not get turned on again. Once a string goes out, it stays out. Therefore if the third

word of any string is still on, it is because the first word satisfied the search A1 AND the second word satisfied the search for B2. So now it is possible to combine association within words, association between words and association between searches to get a rather powerful Indirect or Conditional Access mode.

A useful application of Indirect or Conditional Access is the Vertical Character String Memory. Six (or eight) bits plus a few markers are set aside from all words in the memory. The unit of storage is the character. As information enters the system it is stored vertically serial by character using the natural punctuation as delimiters, perhaps supplemented by machine generated symbols not in the normal character set. This forms a simple associative memory which can be searched, read and written but serially by character. However, it can be parallel by string or sub-string using punctuation as headers and sub-headers. Furthermore, there are no such things as word boundaries. For highly formatted, regular nearly fixed length data, this approach is not very impressive. For irregular, variable length data, the Vertical Character String Memory can be quite useful. Other bit fields of the words can be used as additional string memories or as normal word-organized units with a reduced number of bits per word. The two forms can share the memory and exist simultaneously.

*Inverted Access.* Inverted access simply implies an inversion of the normal application of a single search criterion and its mask against all data units in the associative memory, that is, the storage of a number of search criteria with their masks and the application of a single data unit against them. With the required internal masking, the selection and tagging of all search criteria satisfied by the applied data unit can be done simultaneously. This mode can be applied to periodically running a file too large to contain in the memory against batched requests, particularly if there is sufficient multiprogramming capability so that between arrivals of information from a slower speed mass store, other useful work can be performed. The ability to store a mask along with a search criterion avoids the necessity of storing not only the basic search criterion but also a version for each of the combinations allowed by the don't care bits which is otherwise necessary since it must be assumed the different batched search criteria will be masked in arbitrary fields and therefore the data cannot be masked when it is passed through the memory.

Inverted access also has application to less grand scale problems than information retrieval from bulk store. Input conversion or format transformations can be performed in modes combining Inverted and Parallel access. Where the data unit breaks up into subfields on which independent operations can be performed, each operation is expressed as a set of masked searches examining only the subfields of interest, each set of masked searches operating in parallel. Associated with each logical case as represented by a stored search and mask is a response which may be stored in a field not common to any of the data subfields or in an adjoining word which is accessed indirectly using SNW or SPW and SAH.

There is a programming technique, using branch tables discussed in a later paper, which makes use of an N-way branch. This N-way branch is easily implemented by the inverted access mode. A stored search and mask represents a logical classification of the test, status and control information leading to one of the branch terminals. Inverted access avoids the necessity of storing all possible combinations and allows storing only those system conditions, along with don't cares represented as no comments, that actually lead to a branch terminal.

A quite exciting combination of Inverted and Parallel access can be used to build up group characteristic profiles of a data set by passing each data unit against a number of stored searches and masks representing characteristics of interest and "counting" the hits in parallel. An example of on-line process control will be discussed later.

*Parallel Access.* The Parallel Access mode is the workhorse often used in conjunction with other modes. The properties of individual bit write and SAH give the ability to tag in parallel. This avoids removing each hit, changing bits in an external register, reinserting the word and repeating for the next hit. Tagging can be generalized from manipulating a special field of marker bits to any modification of a subset of words that is identical for all members of the subset. Being able to search on the tags allows modifications to be built up from several related taggings.

## EXAMPLES

It might be beneficial to go through some examples to try to clarify the details of the primitives

and structural properties and develop insight about some of the vaguely defined concepts such as associative logic, parallel sequential forms, modes of access, etc.

The first three examples show the use of the Associative Memory Structure to perform conventional addition on a pair of numbers. Their main importance is an illustration of the logical power that is available and a demonstration of the speed/storage tradeoff that is available to the designer. The next example is a serial-by-bit addition that may be performed on many pairs of numbers simultaneously. It illustrates one of the parallel sequential logic forms. The fifth example is a formatting operation. It illustrates some of the logic power available for nonnumeric functions and the other form of parallel sequential logic. The sixth example illustrates several functions and several capabilities of the Associative Memory Structure tied together by a significant piece of a useful problem from the context of process control of a cryotron manufacturing operation. It shows the application in parallel to many data groups of a sequential process, each step of which consists of parallel bit manipulations and suggests that parallelism may be extended upward through the language hierarchy to the user.

*A Simple Carry Register Addition*

Figure 6 shows the truth tables and memory setup of the stored logic table for an addition algorithm shown in Fig. 7. This is a simple straightforward algorithm imitating the action of a conventional adder based on the old von Neumann carry register technique. A pair of numbers are fetched from memory and inserted into the registers (M1 & M2) of a stored logic table adder. A computation loop (steps 5,6,7) produces the partial sum in one register and the carries in the other. The carries are shifted and the loop repeated until there are no more carries. The final sum is read out and put away.

STORED CARRY LOGIC

| G | H | S | C | M1 | M2 | M1' | M2' | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | no change |
| 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | exchange M1 & M2 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | no change |
| 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | reset M1 |

LOGIC TABLE

| Access Code | Label | OV | BN-B1 | M1 | M2 |
|---|---|---|---|---|---|
| 5 | Add 3 | X | XXXX1 | | |
| 5 | Add 3 | X | XXX1X | | |
| 5 | Add 3 | X | XX1XX | | |
| 5 | Add 3 | X | X1XXX | | |
| 5 | Add 3 | X | 1XXXX | | |
| 5 | Add 3 | 1 | XXXXX | | |

Figure 6. Simple carry register add–logic.

```
1. S(2/access code, "G"/label), SAH,R(1/data)

2. S(5/access code, Add 3/label, 0/ov, G/B1-N), SAH, W(1/M1)

3. S(2/ac, "H"/lbl), SAH, R(1/data)

4. S(5/ac, Add 3/lbl, 0/ov, H/B1-N), SAH, W(1/M2)

5. S(5/ac, Add 3/lbl, 0/M1, 1/M2), SAH, W(1/M1, 0/M2)

6. S(5/ac, Add 3/lbl, 1/M1, 1/M2), SAH, W(0/M1, 0/M2), SNW

7. SAH, W(1/M2), TEST
   Hit:  loop to 5.  No Hit: continue

8. S(5/ac, Add 3/lbl, 1/M1), SAH, W(0/M1), R(1/ov, 1/B1-N)

9. S(2/ac, "F"/lbl), SAH, W(F/data)
```

Figure 7. Simple carry register add–algorithm.

The first truth table displays the correct Sum (S) and Carry (C) for individual bits of the pair of numbers (G & H). The second truth table shows the same logic assuming G & H are placed in M1 & M2 (initially zero) and the Sum and Carry are developed in M1 & M2, respectively. Only two cases require action.

Step 1 fetches G. Step 2 enters G into M1. The search includes a requirement for access code 5 and Add 3 in the label field. This ensures that nothing which might have the same bit configuration of Add 3, but which is not a machine defined stored logic table (designated by access code 5) will be selected and that none of the stored logic tables except Add 3 will be selected. The search for (0/ov) ensures that the overflow word will not be selected during loading. The second word of the logic table has stored no comments in every bit position of the data field except the second. Therefore, the value of the search requirement in the data field (G) for bits 1 and 3 through N will have no effect on the selection of the second word. There is a one in the second bit so the second word will hit or miss according to the value of the second bit of G.

In general, the Kth word of the logic table fol-

lows the Kth bit of the data field. The diagonal of ones embedded in the block of no comments acts as a "logical mirror or deflector" to swing a data field from a horizontal to a vertical format.

Step 5 searches for all instances of a logical case, 01 in M1, M2. However many there are, they are selected in parallel and the write exchanges M1 and M2. If there were no hits, no words would be selected. The write would take place but no words in the logic table are affected.

Step 6 searches for 11 in M1, M2 and resets M1, completing the process for the unit step of the carry register adder. This illustrates in a simple way the pattern of logic called parallel sequential. The function is broken up into four cases. Two need no further processing. The remaining two cases are treated sequentially but all instances of each case are treated in parallel.

Since the carry must be propagated, step 6 also performs part of the propagation operation, first clearing M2 and then shifting the carry to the Hit ff of the next register position. Step 7 writes any shifted carries into M2 and also tests for their occurrence. A partial sum and carry is indistinguishable from a new G & H in M1 and M2, so if there are any carries, they can be handled by looping back to step 5.

Step 8 selects words corresponding to register positions for which the sum is one and reads the data field ORing the outputs along bit positions. The Kth word has stored no comments in every bit position except the Kth so its readout can only influence the Kth bit of the data field. The Kth bit position has stored no comments in every word except the Kth so a one output is possible only if the Kth sum is one. The multiple read collects the results for each sum bit computation into a single result word.

For a N bit data field, plus overflow bit the storage requirement of the stored logic table is $N + 1$ words. The worst case time in normalized units is $6 + 3N$. The average is $6 + 3[\log_2 N]$ (next integer greater than the binary logarithm of N).

### *A Partitioned Carry Register Addition*

Figures 8 & 9 show the stored logic table and algorithm for an improvement of the carry register add. The carry register is partitioned into K blocks of M bits each (KM is greater than N, the number of bits in the data field). At the expense of a larger

| Label | OV | BN-B1 | M1 | M2 | M3 | M4 |
|---|---|---|---|---|---|---|
| Add | X | XX XXX XX1 | | | 1 | 0 |
| Add | X | XX XXX X1X | | | 1 | 1 |
| Add | X | XX XXX 1XX | | | 1 | 1 |
| Add | X | XX XX1 XXX | | | 1 | 0 |
| Add | X | XX X1X XXX | | | 1 | 1 |
| Add | X | XX 1XX XXX | | | 1 | 1 |
| Add | X | X1 XXX XXX | | | 1 | 0 |
| Add | X | 1X XXX XXX | | | 1 | 1 |
| Add | 1 | XX XXX XXX | | | 1 | 1 |
| Add | X | XX XXX XX1 | 0 | | 0 | 1 |
| Add | X | XX XXX 11X | 0 | 0 | 0 | 0 |
| Add | X | XX XX1 XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XXX X1X | 0 | | 0 | 1 |
| Add | X | XX XXX 1XX | 0 | 0 | 0 | 0 |
| Add | X | XX XX1 XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XXX 1XX | 0 | | 0 | 1 |
| Add | X | XX XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XX1 XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XX1 XXX | 0 | | 0 | 1 |
| Add | X | XX 11X XXX | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX X1X XXX | 0 | | 0 | 1 |
| Add | X | XX 1XX XXX | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX 1XX XXX | 0 | | 0 | 1 |
| Add | X | XX XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XXX 1XX | 0 | | 0 | 1 |
| Add | X | XX 111 XXX | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XXX X1X | 0 | | 0 | 1 |
| Add | X | XX 111 1XX | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |
| Add | X | XX XXX XX1 | 0 | | 0 | 1 |
| Add | X | XX 111 11X | 0 | 0 | 0 | 0 |
| Add | X | X1 XXX XXX | 0 | 0 | 0 | 0 |

Figure 8. Partitioned carry register add–logic.

```
1.  S("G"/label), SAH, R(1/data)
2.  S(Add/label, 0/ov, G/B1-N, 1/M3), SAH, W(1/M1)
3.  S("H"/lbl), SAH, R(1/data)
4.  S(Add/lbl, 0/ov, H/B1-N, 1/M3), SAH, W(1/M2)
5.  S(Add/lbl, 1/M3, 0/M1, 1/M2), SAH, W(1/M1,0/M2)
6.  S(Add/lbl, 1/M3, 1/M1, 1/M2), SAH, W(0/M1,0/M2), R(1/B1-N),
    SNW, SAH, W(1/M2)
7.  S(Add/lbl, 0/M3, 1/M4, "carries"/B1-N), SAH, W(1/M2)
8.  S(Add/lbl, 1/M3, 1/M1), SAH, R(1/B1-N)
9.  S(Add/lbl, 0/M3, 1/M4, 1/M2), SAH, W(0/M2), SNW
10. SO(Add/lbl, "propagates"/B1-N), SAH, SNW, SAH, R(1/B1-N)
11. S(Add/lbl, 1/M3, "interblock carries"/B1-N), SAH,W(1/M2)
12. S(Add/lbl, 1/M3, 0/M1, 1/M2), SAH, W(1/M1, 0/M2)
13. S(Add/lbl, 1/M3, 1/M1, 1/M2), SAH, W(0/M1, 0/M2), SNW
14. SO(1/M3, 1/M4), SAH, W(1/M2), TEST
    Hit: Loop to 12. No Hit: Continue
15. S(Add/lbl, 1/M3, 1/M1), SAH, W(0/M1), R(1/ov, 1/B1-N)
16. S("F"/lbl), SAH, W(F/data)
```

Figure 9. Partitioned carry register add–algorithm.

logic table, the time to perform the addition depends on M instead of N. As before, a logic table (1 in M3) imitates a set of carry register adders. There is an additional logic table (0 in M3) that examines the sum and carry results after the first cycle, produces all the carries that will cross block boundaries and inserts them immediately into the beginning of each block. The simple carry register add then resumes, operating in parallel on the bits of each block and in parallel on all blocks.

The first four steps fetch the two operands and insert them into the registers, M1 & M2. Steps 5 and 6 perform the first cycle of addition.

The interblock carries are determined by a second logic table consisting of triplets. The first word of each triplet responds to a particular carry case. The second word responds to the required bit configuration of the partial sum bits to propagate that carry across the block boundary. The third word represents the output carry value to be inserted at the beginning of each block (1 in M3, 0 in M4).

The first triplet is activated by a carry from the first bit of the first block. The second word of the first triplet checks if the sum bits are one from the second to last bit of the first block. If so, the third word outputs a carry to the second block. Note that at the same time, the ninth triplet checks for a carry from the first bit of the first block but looks for a sum pattern that takes the carry through the second block into the third.

The fourth triplet performs the same logic as the first except for the control of a carry from the second to third block. Note the stored no comments in the second subfield of the first triplet and the first subfield of the fourth triplet. The no comment is used as a logical disconnection so that the first and fourth triplets may perform their computation in parallel and independently of each other without any possibility of interference.

Step 7 takes the carries while they are at hand and stores them temporarily in M2 of the entry word of each triplet. Step 8 goes back to retrieve the partial sum and holds it for subsequent driving. Steps 9, 10, 11 use the Conditional Access mode to select the correct triplets and insert the interblock carries. Steps 12, 13, 14 are the simple carry register loop. Step 14 is modified to also require a 1 in M4 so that the last carry of a block does not cross over into the next block, duplicating the insertion of interblock carries.

The worst-case time in normalized units is $13 + 3(M-1)$. The average is $13 + 3([\log_2 M]-1)$. The storage requirement is $N+1+(3/2)K(K-1)M$.

*A Simultaneous Carry Generation Addition*

Figures 10 and 11 show the truth table and stored logic table for a third addition algorithm. This is a logical descendant of the partitioned carry register approach for a block size of one bit. All the carries that will occur are calculated and absorbed into the sum together. Considerable shortcutting is achieved by employing the result word as a scratchpad for an intermediate result and by making use of the logic power inherent in the read operation.

The truth table displays the possible cases for each bit position of G and H, the carry coming in (ignoring how it got there) and the correct sum F. The next column displays the result of reading H with G. The last column shows the result of reading G:H with the carries. This equals the sum.

Step 1 fetches the first operand. Step 2 selects the second operand and reads it with the first. Step 3 stores the result temporarily in the result word. G is transferred to the Contents register and G:H to the Mask register in the data field. Step 4 searches in the logic table for the entry word (1 in M4) of

| G | H | C | F | G:H | (G:H):C |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 |

```
1. S("G"/label), SAH, R(1/data)

2. S("H"/label), SAH, R(G/data)

3. S("F"/label), SAH, W(1/ov, G:H/data)

4. S(Add/lbl, 1/M1, G/G:H in Bl-N), SAH, SNW

5. SO(G:H/G:H in Bl-N), SAH, R(0/Bl-N)

6. S("F"/lbl), SAH, R(carries/data), W(G:H:C/data)
```

Figure 10. Simultaneous carry generation add–algorithm.

| Label | OV | BN-B1 | M1 |
|---|---|---|---|
| Add | X | XXX1 | 1 |
| Add | X | XX01 | 0 |
| Add | X | XX11 | 1 |
| Add | X | X001 | 0 |
| Add | X | X111 | 1 |
| Add | X | 0001 | 0 |
| Add | X | 1111 | 1 |
| Add | 0 | 0001 | 0 |
| Add | X | XX1X | 1 |
| Add | X | X01X | 0 |
| Add | X | X11X | 1 |
| Add | X | 001X | 0 |
| Add | X | 111X | 1 |
| Add | 0 | 001X | 0 |
| Add | X | X1XX | 1 |
| Add | X | 01XX | 0 |
| Add | X | 11XX | 1 |
| Add | 0 | 01XX | 0 |
| Add | X | 1XXX | 1 |
| Add | 0 | 1XXX | 0 |

Figure 11. Simultaneous carry generation add–logic.

each doublet with the masked data field. Only where the Mask is one will the bits be driven. This coincides with bits where G and H are equal and their value is given by G in the Contents register. The search drives bits where carries are generated or can be absorbed.

Step 4 concludes by making the output words of doublets whose entries satisfied the first search candidates for conditional access. Step 5 searches only the candidates with zero in those bit positions which only propagate carries and concludes with a read complement.

The entry of the first doublet will be selected if the first bit is a carry generate or propagate. The output word will remain selected in step 5 if the first bit is a generate. If the second bit is another generate or a carry absorb, it will not be driven in step 5. If it is a propagate, it will be driven with a zero. If the first bit is a propagate, then the output word will fail the search and a false carry into the second bit will be avoided. There is similar behavior in doublets 5, 8, 10.

The entry word of the second doublet will be selected as long as neither the first nor second bits are carry obsorbs. If both bits or the first bit only is a carry propagate, the output word will fail the search in step 5, avoiding false carries. If both bits are generates, the output word will pass the search, as will the first and fifth doublets. If the first bit is a generate and the second a propagate, the output word will provide not only a carry into the second bit, duplicating the first doublet, but also a carry into the third bit. If the third bit were a carry absorb, no other doublet would provide this necessary output.

If a carry is generated in the first bit, propagated through several bits and then absorbed, the first group of doublets will be selected until the leading one of an entry word coincides with the driven carry absorb.

Step 6 selects the result word which is temporarily holding G:H and reads it with the carries from the logic table. The result is written back into the sum word.

The logic table storage requirement is $N(N + 1)$ words for an N bit data field. The time in normalized units is a constant, six.

### *Speed/Storage Tradeoff in Addition*

These three examples of addition are not too important in their own right. They are examples of conventional addition algorithms, operating on one pair of numbers at a time, which can be performed by the Associative Memory Structure if desired. Furthermore, artithmetic capability is not the critical problem in computers now. However, these examples may bring home the considerable logic power inherent in the primitive information manipulations and help to clarify the earlier discussion of the primitives and features of the Memory Structure. These examples do illustrate an important characteristic of ASC. There is a fairly broad tradeoff of storage requirements and performance of functions available to the designer. This can be used not only to optimize the machine initially, but since the logic is stored rather than wired in, the machine can be optimized over its operating life as its problem environment changes by reading in new, more appropriate algorithms.

### *Serial By Bit, Parallel By Word Pair Addition*

Figure 12 shows the truth table and algorithm for a serial by bit addition which uses no logic table and uses the data words as scratch pad registers.

| M1 | M2 | dj | M1' | dj' |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |

```
0.  Start with j = 1

1.  S("A"/lbl, 1/dj), SAH, SNW, SAH, W(1/M2)

2.  S("B"/lbl, 0/M1, 1/M2), SAH, W(1/M1, 0/M2)

3.  S("B"/lbl, 1/M1, 0/dj, 0/M2), SAH, W(0/M1, 1/dj)

4.  S("B"/lbl, 1/M1, 1/dj, 0/M2), SAH, W(0/dj)

5.  S("B"/lbl, 1/M2), SAH, W(0/M2)

6.  increment j, test j > N:  no-loop to 1, yes-done
```

Figure 12. Bit serial, word parallel add.

There are two contiguous words A and B, A above B, and the sum is formed in B. There are two marker bits. M1 holds the carry in B. M2 in B receives the bit to be added from A. The jth digit position is designated by dj, j referring to an implicitly defined counter in the Mask register running from the least to most significant bit.

The truth table displays the cases of M1, M2, dj and the new values of M1, dj. The first four cases require no change in M1 or dj. The sixth case has the same result as the fifth and can be converted by exchanging M1 & M2. The eighth case has the same result as the seventh and can also be converted by exchanging M1 & M2. Note that the seventh case ends in the same state of M1 and dj that begins the fifth case, so the fifth case must be handled before the seventh.

This algorithm is a simple illustration of one form of parallel sequential logic. The function is broken up into cases which are passed against the data. As an individual digit of A and B is examined, not all cases will select and so some operations will be dummies. However, all possible cases are covered. Suppose there were several pairs of numbers with a portion of their label field identical. Then in executing the algorithm, for each digit position, the numerical values alone would determine at each step whether none, one or several of the pairs would be selected. Every digit position would fall into some case. All pairs in the same case would be processed simultaneously. When the last bit had been processed, all word pairs would have been added. The serial by bit algorithm is parallel by word. The parallel sequential logic treats the individual cases of the function sequentially but handles all instances of each case in parallel.

### *A Format Transformation*

Figure 13 shows the logic tables and algorithm for another example of parallel sequential logic but for the second form-sequential by item of data, parallel by function. This time the example is nonnumeric, format transformation. The input is an N character packed number, calculated by some process, to be printed out on a check. The rightmost two characters specify cents, delimited by a period. The remaining characters are grouped right to left in fields of up to three characters, each field separated by commas. A dollar sign is placed to the left of the most significant character and the spaces to the left of the dollar sign are filled with asterisks. There is an exception. Any amount less than one dollar is printed as dollar sign, zero dollars, point, and the appropriate cents.

In a conventional machine, this would probably be programmed as a set of subroutines to perform the justifying, deletion, shifting, superposition, insertion, testing for most significant digit, asterisk fill and check for special case. In this example, the operation will be viewed as two functions, FORMAT SPREAD 2 and FORMAT FILL 3, which operate in parallel on the data. The two functions have a common label so that different SPREAD or FILL functions can be called when appropriate. The use of no comments in the second and third label fields allows independent calling.

Both logic tables consist of doublets. There is a bit of shorthand used in displaying the logic table. There is an enlargement of the third doublet of the first logic table to the side. The positions refer to the characters, not individual bits. In the third doublet, the entry word (actually six words, one for each bit of the character) copies the third character. The associated output word (actually six

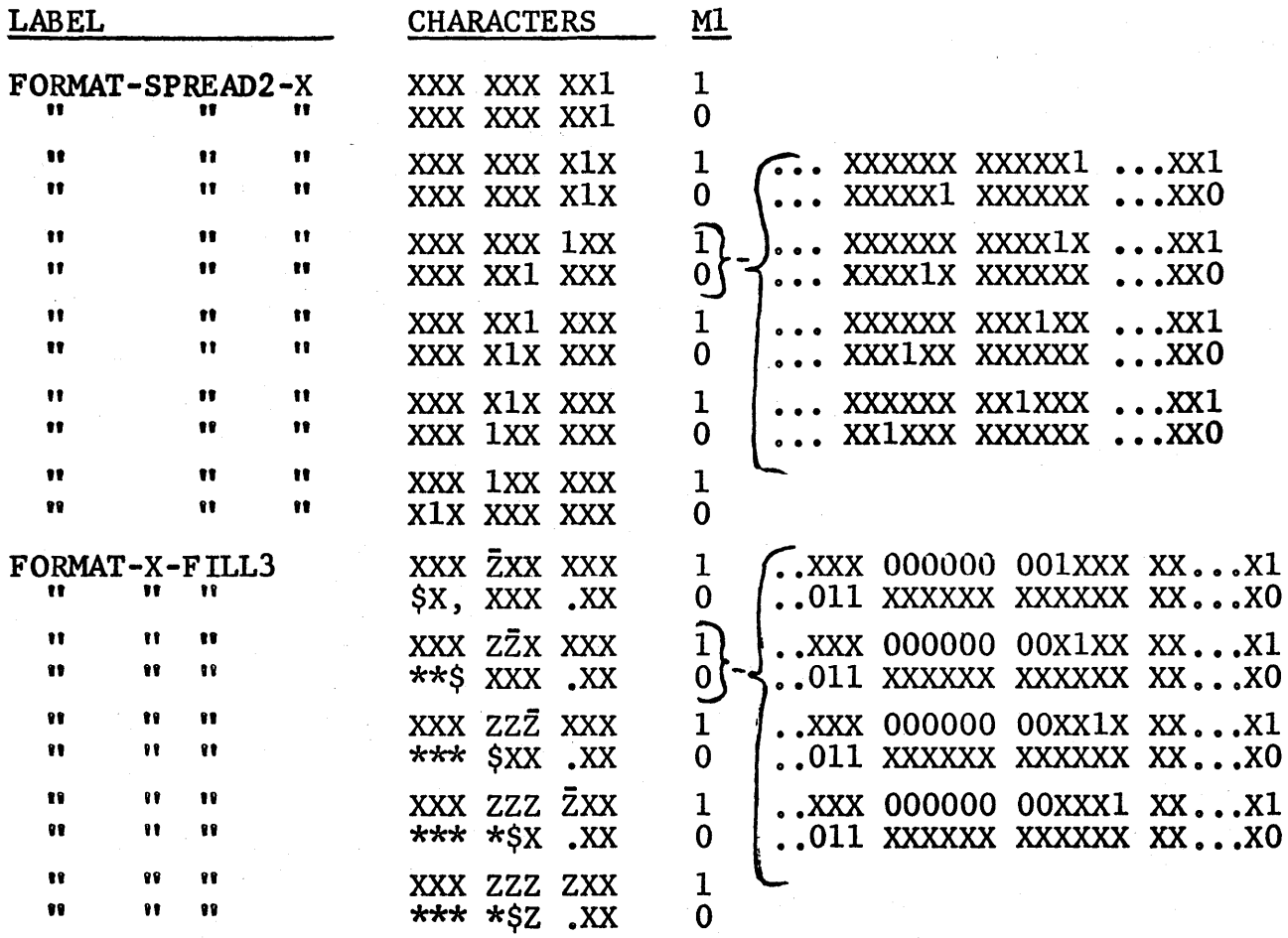



Figure 13. Example of format transformation.

words) reproduces the third input character but in the fourth character position. As the enlargement indicates, the entry words for each bit of the character are interleaved with the output words. The readout spreads the packed number out as required by the format transformation assuming all characters are significant.

The second logic table also uses some shorthand in display. The first four doublets find the most significant character. The fifth doublet checks for the exception case. In the second doublet, the sixth character position is the code for the number zero which is all zeros as shown in the enlargement. The fifth character position represents the logical condition of some code for a number other than zero. This breaks up into four subcases (zone bits are zero for numerics). If the fifth character is not zero, then at least one of the four low order bits is one and at least one of the correct output words will be selected, giving the correct fill.

The structure of the logic tables and the algorithm have been chosen so that both logic tables respond in parallel to the data and format function call. The time in normalized units is one and the required storage is 24N-22 words, where N is the number of 6-bit packed numeric characters.

*A Process Control Problem*

Figures 14, 15, 16 and 17 illustrate an example of operations that might be appropriate to a process control problem. It is assumed that there is a manufacturing process which fabricates cryotron plates of a repetitive structure. The direct cost of fabrication is quite low, but the cost of testing plates could be quite large, especially if plates which are ultimately rejected are fully tested since their test cost must be amortized over the plates that are finally accepted. The test process is divided into two parts: a final cold test which determines whether or not the plate is used in a system and a room temperature test which follows the fabrication and precedes the cold test. The room temperature test is designed to reject

at low cost a large percentage of the plates that would ultimately be rejected. It is automated and steps a set of contacts across the repeated cells of the plate, performs some tests on parameters, computes the pass/fail components of a unit cell test vector and transmits this vector to a computer. There is a set of logic-storage modules, one for each parameter of the test, which keep track of the number of cells tested on the plate, the number of failures and calculate for each parameter whether the number of failures for the given number of tests falls above a reject limit or below an accept limit of a sequential test chart as shown in Fig. 14. If a parameter leads to rejection, plate is rejected and a

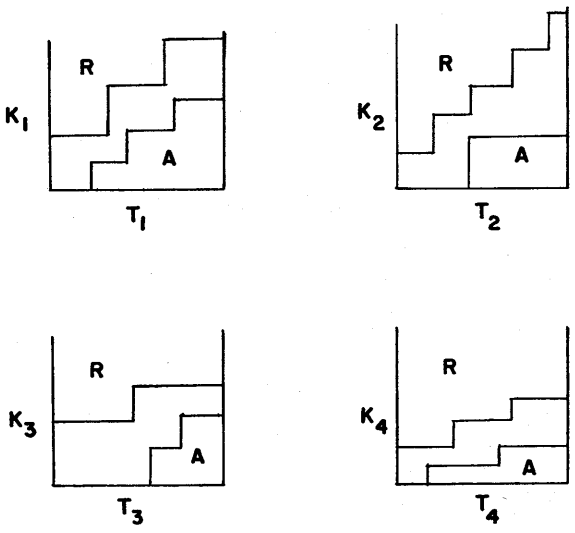



Figure 14. Example of process control limit charts.

new one started in test. If all parameters lead to an acceptance, the plate is passed on for cold test. If no parameter has led to rejection, but not all parameters have reached a decision yet, the automatic tester steps over to the next set of contacts and performs a new test of the same plate. If an individual parameter rejects or accepts, its count is frozen while the others go on.

The format of the logic-storage module for a parameter is shown in Fig. 15. It consists of four contiguous words. The modules may be separated arbitrary distances but the components of the modules must be together. Each word has a common label, AUTO TEST, and a 1 in MO, signifying parameter test. M1 and M2 distinguish the individual words.

The second word accumulates the number of failures in its count field and compares this to its limit field to see if the parameter indicates rejection. The first word accumulates the total number of tests in its count field and compares this to its limit field to determine when to trigger both itself and the second word to add the increment fields to the respective limit fields. The fourth word compares failures to an accept limit. The third word checks number of tests to determine when to adjust the limits of itself and the fourth word.

The key field identifies the parameter being examined by the module. A failure of a parameter is communicated by driving a one in the key field. A zero is driven if the parameter passes. The second and fourth words will respond only to failures. The first and third words will respond to any test. The flag field is used in readout to identify whether the rejection (0) limit or the acceptance (1) limit was crossed.

The initialize algorithm clears markers 3-8 and the limit field in all modules and then serially transfers the initial value field to the limit field.

The count algorithm drives the vector of parameter pass/fail values into the key field and marks responses with a 1 in M6, the trigger. M5, the freeze bit, must be zero which ensures that modules measuring parameters which have crossed a limit will not do any further counting. The second and third steps count up by one serially by bit but parallel by word.

The compare algorithm uses M3 & M4 to indicate in each module whether a count was greater than (10 in M3,M4), equal to (00 in M3,M4) or less than (01 in M3,M4) its associated limit. This algorithm proceeds from least to most significant bits. At any stage where the count and limit bits are equal, M3 & M4 are unchanged. So at the end of the cycle the value of M3 & M4 corresponds to the inequality value of the highest order unequal bits of the count and limit.

The check-limits algorithm uses the results of the compare to determine if any failure limits have been crossed and if so, freezes those modules.

The adjust limits algorithm examines total test count of unfrozen modules to see if these limits have been crossed. If so, both the total test count word and its associated failure count word are triggered to add the increment to the limit. Figure 16 shows the truth table of this addition in stages corresponding to steps 2, 3, and 4.

| Flag | Key | Count | Limit | Increment | Initial Value | M0 | M1 | M2 | M3 | M4 | M5 | M6 | M7 | M8 | Label |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| XXX | XXX | t | TU | dTU | TU0 | 1 | 0 | 0 | | | | | | | AUTOTEST |
| X0X | X1X | k | KU | dKU | KU0 | 1 | 0 | 1 | | | | | | | AUTOTEST |
| XXX | XXX | t | TL | dTL | TL0 | 1 | 1 | 0 | | | | | | | AUTOTEST |
| X1X | X1X | k | KL | dKL | KL0 | 1 | 1 | 1 | | | | | | | AUTOTEST |

INITIALIZE

1. S(1/M0), SAH, W(0/Limit, 0/M3—M8)
2. S(1/M0 , 1/Initial Value bit j), SAH, W(1/Limit bit j)-repeat till field exhausted.

COUNT

1. S(1/M0, 0/M5, test vector/Key), SAH, W(1/M6)
2. S(1/M0, 0/M5, 1/M6, 0/Count bit j), SAH, W(0/M6, 1/Count bit j)
3. S(1/M0, 0/M5, 1/M6), SAH, W(0/Count bit j)
   Repeat 2 & 3 for all Count bits.

COMPARE

1. S(1/M0, 0/M5, 1/Count bit j, 0/Limit bit j), SAH, W(1/M3, 0/M4)
2. S(1/M0, 0/M5, 0/Count bit j, 1/Limit bit j), SAH, W(0/M3, 1/M4)
   Repeat 1 & 2 for all bits of Limit field, least to most significant.

CHECK LIMITS

1. S(1/M0, 0/M1, 1/M2, 0/M5, 10/M3M4), S1(1/M0, 1/M1, 1/M2, 0/M5, 01/M3M4), SAH, W(11/M3M4, 1/M5)
2. SPW, SAH, W(1/M5)
3. S(0/M1, 1/M5), SAH, SNW, SAH, W(1/M5), SNW, SAH, W(1/M5)
4. S(1/M1, 1/M5), SAH, SPW, SAH, W(1/M5), SPW, SAH, W(1/M5)

ADJUST LIMITS

1. S(1/M0, 0/M5, 0/M2, 10/M3M4), SAH, W(1/M6), SNW, SAH, W(1/M6)
2. S(1/M0, 1/M6, 1/Increment bit j, 0/M7), S1(1/M0, 1/M6, 0/Increment bit j, 1/M7), SAH, W(1/M8, 0/M7)
3. S(1/M0, 1/M6, 1/M8, 1/Limit bit j), SAH, W(0/M8, 0/Limit bit j, 1/M7)
4. S(1/M0, 1/M6, 1/M8), SAH, W(0/M8, 1/Limit bit j)
   Repeat 2, 3 & 4 for each bit of Increment Field
5. S(1/M0, 1/M6), SAH, W(0/M6)

Figure 15. Response to parameter test vector.

| a | b | c | SC | a'c'M8' | a"c"M8" | a"'c"'M8"' | | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 00 | 0 0 0 | 0 0 0 | 0 0 0 | S=a,C=c | no change |
| 0 | 0 | 1 | 10 | 0 0 1 | 0 0 1 | 1 0 0 | S=$\bar{a}$,C=$\bar{c}$ | |
| 0 | 1 | 0 | 10 | 0 0 1 | 0 0 1 | 1 0 0 | S=$\bar{a}$,C=c | |
| 0 | 1 | 1 | 01 | 0 1 0 | 0 1 0 | 0 1 0 | S=a,C=c | no change |
| 1 | 0 | 0 | 10 | 1 0 0 | 1 0 0 | 1 0 0 | S=a,C=c | no change |
| 1 | 0 | 1 | 01 | 1 0 1 | 0 1 0 | 0 1 0 | S=$\bar{a}$,C=c | |
| 1 | 1 | 0 | 01 | 1 0 1 | 0 1 0 | 0 1 0 | S=$\bar{a}$,C=$\bar{c}$ | |
| 1 | 1 | 1 | 11 | 1 1 0 | 1 1 0 | 1 1 0 | S=a,C=c | no change |

Figure 16. Logic of simultaneous limit increment addition.

Figure 17 shows the stored logic table (one word) and algorithm to check the sample. Step 1 reads the key and flag of all reject/accept words frozen because a limit was crossed. Initially, M7 is zero and M8 is one. Step 2 will select the logic table if and only if all parameters have crossed the

| Label | Flag | Key | M7 M8 |
|---|---|---|---|
| Sample Test | 111 | 111 | |

CHECK SAMPLE

1. S(1/M0,1/M2,1/M5,11/M3M4),SAH,R(1/Key,1/Flag)
2. S(Sample Test/Label, Key readout/Key, Flag readout/Flag), SAH, W(1/M7)
3. S(Sample Test/Label, 0/M7, Flag readout/Key readout in Key Mask), SAH, W(0/M8)
4. S(Sample Test/Label), SAH, R(1/M7,M8),W(0/M7,1/M8)

Figure 17. Accept/reject sample evaluation.

accept limit. Step 3, searching over the key field with the flag readout in the Contents register and the key readout in the Mask and for 0 in M7, will select if and only if not all parameters have crossed a limit and that none that have crossed were rejections. The last step selects the logic table, reads M7 & M8 and initializes them. The automatic tester interprets the readout of 1 in M8 as a signal to step the test and go on to the next plate; 0 in M8 as a continue signal. The plate is passed for 1 in M7, rejected for 0 in M7 if the test is stopped.

The logic of the example can be concatenated by providing another set of modules with 0 in M0 to indicate process test. Each time a plate test is stopped and the plate passed or failed, the final cumulative vector used in the check sample algorithm can be driven into the process test modules to monitor the overall fabrication process to check if it is doing well, so-so, or about to go out of control. The key/flag criteria can involve combinations of parameters as well as individual parameters themselves for more sophisticated measures of the performance of the fabrication process. The limit/increment logic of either the parameter or process test modules can be made more complex with additional fields and/or additional words in the module (to adjust the increments of the limit fields for example) and additional steps to allow more complex sequential testing limit curves which might be defined by second, third, etc., order difference equations.

## CONCLUSIONS

The Associative Memory Structure is a versatile storage and processing building block for computers and is not limited to the role of a search memory. While a cellular approach may be taken to alleviate interconnection problems of batch fabricated devices, there is some advantage to departing from a purely cellular system of identical elements in a uniform array. Hierarchy in communication and distributed control allow selective parallelism. Some cells will respond to control information by determining whether or not a group of associated cells will respond to a subsequent control sequence.

The small set of primitive information manipulation operations is quite powerful and much more flexible than built in macro commands. Computer functions can be composed with the primitives using the parallel sequential logic approach. The batch fabrication technology is not required to be able to implement very complex macro commands and fewer plate or chip types need to be designed and stocked.

## ACKNOWLEDGMENT

The author wishes to acknowledge the contribution of John W. Bremer and Dwight W. Doss, coinventors of the cryotron associative memory structure.

## REFERENCES

*Companion Papers*

B. T. McKeever, "The Associative Switching Structure," *IEEE Trans. on Electronic Computers,* vol. EC-15, to be published in 1966.

———, "The Associative Structure Computer," ibid.

NOTE: A more extensive set of references for all three papers will appear in "The Associative Structure Computer."

Cryogenic Electronics Issue, *Proc. IEEE,* vol. 52, Oct. 1964.

Integrated Electronics Issue, ibid., Dec. 1964.

*Proc. Symposium on the Impact of Batch Fibrication on Future Computers,* Los Angeles, Apr. 1965.

W. T. Comfort, "A Modified Hollard Machine," *Proc. FJCC,* Nov. 1963, pp. 481-488.

J. Sklansky, "General Synthesis of Tributary Switching Networks," *IEEE Trans. on Electronic Computers,* vol. EC-12, pp. 464-469 (Oct. 1963).

R. C. Minnick, "Cutpoint Cellular Logic," ibid., vol. EC-13, pp. 685-698 (Dec. 1964).

D. L. Slotnick, W. C. Borck and R. C. McReynolds, "The SOLOMON Computer," *Proc. FJCC,* Dec. 1962, pp. 97-107.

R. Fuller and G. Estrin, "Algorithms for Content-Addressable Memory Organizations," *Proc. Pacific Computer Conference,* Pasadena, Mar. 1963, pp. 118-130.

R. G. Ewing and P. M. Davies, "An Associative Processor," *Proc. FJCC,* Oct. 1964, pp. 147-158.

B. A. Crane and J. A. Githens, "Bulk Processing in Distributed Logic Memory," *IEEE Trans. on Electronic Computers,* vol. EC-14, pp. 186-196 (Apr. 1965).

W. Shooman, "Parallel Computing with Vertical Data," *Proc. EJCC,* Dec. 1960, pp. 111-115.

J. Atkin and N. B. Marple, "Information Processing by Data Interrogation," *IEEE Trans. on Electronic Computers,* vol. EC-11, pp. 181-187 (Apr. 1962).

# COMPUTER EDITING, TYPESETTING AND IMAGE GENERATION

M. V. Mathews and Joan E. Miller
*Bell Telephone Laboratories, Incorporated*
*Murray Hill, New Jersey*

## INTRODUCTION

The programs which we will describe were developed to provide a practical system for editing and publishing text with a digital computer. The system consists of an electric typewriter, a computer, a cathode ray tube output unit, and a camera. Text and editorial instructions may be entered into the computer from the typewriter. The computer executes the instructions and prepares a corrected, justified text. The text may be written on the cathode ray tube, photographed by the camera, and published by standard photo-offset printing. Alternatively, it may be written on the typewriter by the computer, or printed on the computer printer.

There has been much recent interest in computer editing programs. Among others, extensive work has been done by M. P. Barnett at M.I.T., by P. F. Santarelli at the IBM Systems Development Laboratory in Poughkeepsie and at Project MAC at M.I.T., by R. P. Rich at the Johns Hopkins Applied Physics Laboratory, and by C. R. T. Bacon at the University of Pittsburgh. In addition, some machines are being developed to present graphic arts quality images on cathode ray tubes such as the Merganthaler-CBS Laboratories Linotron machine. Oscilloscopes of lower quality are available as output devices for numerous computers.

Human engineering of both the programs and the equipment was a prime design consideration and led to objectives of:

1. Providing a typewriter as good and as simple as an ordinary secretary's machine. The typewriter can be located in the office of the user.
2. Providing a good upper and lower case type font with flexibility for adding letters and figures.
3. Allowing the intermixing of text and editorial instructions. Thus corrections may apply to a text while it is being written.
4. Providing a comfortable vocabulary of editing instructions.

When the editing program was written, facilities at Bell Telephone Laboratories did not provide instantaneous real-time interaction between a typewriter and the computer on which editing was done. Consequently, the program was written so as not to require interaction. This decision strongly affected the way in which lines in the text are located for correction purposes. When interaction is available, parts of the program may be modified.

The following section of this paper describes the editing program, and the succeeding section the

typesetting and genration of images on the cathode ray tube.

## THE EDITING PROGRAM

### *Description*

The purpose of this program is twofold: (1) to allow for input and correction of typewritten material; and (2) to store material for future editing and/or processing.

The significance of such a system lies primarily in point (2) for although typists and proofreaders can prepare manuscripts and typesetters can produce aesthetically pleasing copy, no one relishes the problem of altering previously prepared material. Consequently, the ability to store text in a form that can be easily updated represents a great convenience. And if the material is stored in a computer, then it is amenable to other helpful forms of processing such as alphabetizing or sorting.

Therefore, the aim of this project has been to develop a system of editing in which the user can provide input to the computer by a device which is familiar and easy to use (i.e., a typewriter), can easily correct the errors he makes while inputting, and can easily modify in the future material he has produced in the past. Furthermore, the design of the system is such that corrections to currently typed material can be made interchangeably with corrections to preexisting material. The corrections themselves constitute current input and do not require a second pass operation for their execution. It is felt that this particular feature adds great flexibility and power to the technique of computer editing.

### *Structure of System*

The editing program is diagrammed in Fig. 1. The manuscript is typed at an IBM Selectric correspondence typewriter and transmitted to the IBM 7094 computer. The data is in the form of 6-bit characters, one for each of the 44 typewriter keys and one for each of several operations such as carriage return, backspace, upper case shift, lower case shift, etc. This input data consisting of both text and instructions is pre-processed by the pre-edit pass of the program. Case shift characters are removed from the data stream and case information is added to each character. Read requests for material stored on the permanent disk file of the 7094 are

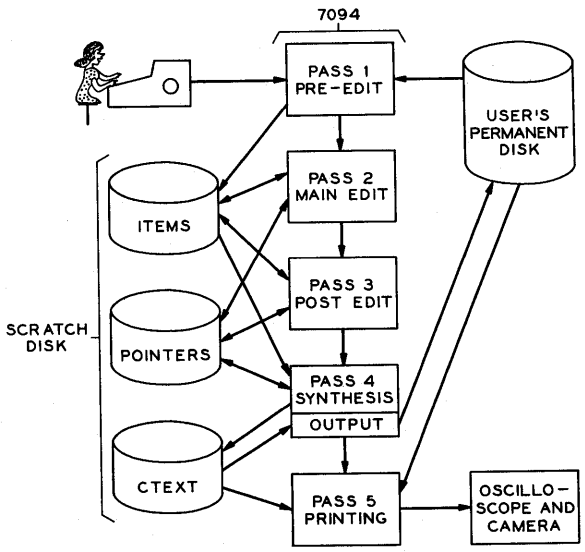



Figure 1. Block diagram of editing program.

executed. All data is stored on scratch disk (ITEMS) for examination by main edit pass.

The sequencing of characters is indicated by a list of pointers which specifies for each character the location of the character which follows. The instructions which call for modifications to the text are therefore executed by resetting the appropriate pointers. Pass 2 executes all such instructions.

The post edit pass is logically equivalent to the main edit but is included as a separate pass to provide space for routines of the users' own design. In pass 4 the characters are sequenced in their proper order by following the pointers. The corrected text is written on scratch disk CTEXT and transmitted to its appropriate place in the users' permanent disk file.

The fifth pass deals with the printing of the text. A galley proof is prepared on microfilm, unless otherwise specified, in order that the user may know the state of the material which has been generated and stored in his permanent disk file. Final copy may be requested by instruction and the format is dictated by instructions still present in the body of the text.

### *Organization of Material*

All typewritten material is subdivided into units called *items*. This organization is determined by the user with the restriction that an item be no larger than 1,800 typewriter characters, which corresponds approximately to one page of typing. Items are

identified by user-assigned decimal numbers of up to six integral places and at most two decimal places. These numbers are typed at the beginning of the item.

The next larger organizational unit of material is the *standard text*. There can be at most 2,000 items in one standard text. A standard text is identified by a (at most) 24-typewriter-character name of the user's choice.

Standard texts (an arbitrary number of them) constitute *standard files,* which are identified by an up to 6-BCD-character name. These files correspond to files of disk storage in the computer. An author preparing an opus of volumes might use a file for a volume, a standard text for a chapter, and each item for a paragraph. Most applications, however, will probably involve one file of one standard text only.

*General Conventions*

A single run of the editing program produces one standard text. The *first line* of typing indicates the name and destination (file) of the standard text being generated. If this standard text is not to be stored on permanent disk, as may be the case when only a printed copy is desired, then the file should be specified as 0.

Editing instructions must be distinguished from text. The presence of every instruction or control is announced to the program by means of the *left square bracket*. The typewriter provides this character in both lower and upper case and thus simplifies the typing of instructions by eliminating extra case shifts. Each instruction has its own code of some single character and this code character must immediately follow the left square bracket. Instructions are terminated in general by a slash.

The text begins with the assignment of an *item number* of up to six integral places and at most two decimal places. Integers need not have decimal points. The assignment of this number requires the code of lower case i and therefore [i1/ labels the characters that follow as item number one.

The privilege of *backspacing* has been preempted by this system in order to provide an effective means of making erasures. A sequence of backspaces will eliminate the sequence of previously typed characters of equal length. The usual customs on underlining and overtyping must be abandoned inasmuch as any backspacing will destroy the input. Provisions for underlining are made by an instruction but, at present, overtyping is not allowed.

As will be seen in the discussion of the instructions which follows there are special break characters such as the left square bracket and the slash which have been preempted by the system. In order to exempt these characters from their usual role should they be desired in some other context, they may be effectively placed in quotation marks or *"super quoted."* An example will be discussed in the illustration which follows.

*Instructions*

Those instructions which are used to edit the text or control options in the program are referred to as Class I instructions. They are identified by a code which is an upper case character and are distinguished from Class II instructions, which specify the output format. A list of instructions presently available is given in Fig. 2. In the general form of the instructions as shown in this list, www and xxx are used to denote item numbers and yyy and zzz are used to denote line numbers. These parameters are considered to be right adjusted in fields marked by commas or the slash. The character # or 0 may be typed in most contexts to denote current item or line. Furthermore, line numbers may be back-referenced by using minus signs if and only if the instruction refers to the item in which it is contained.

*Example of Editing*

Figure 3 is an unrealistic though illustrative example of an original text prepared at the typewriter to demonstrate Class I instructions. The requisite format of the first line is shown.

The first item is numbered 10. It is desired that the content of this item be printed in a fixed format and, hence, the first instruction is [F. The word Objectives is underlined by using the instruction [U#,#/ carriage return . . . underlining . . ./carriage return typed on the same line. Note that the second slash appears first but was, in fact, typed after the underlining was complete. The normal mode of format is resumed after the instruction [J. It is intended that the second line of item 10 be shortened to "II. Description" by use of the instruction [Ø . . . . However, the line number is erroneously stated as 1. (This difficulty will be remedied later.)

Item 12 contains several errors. A C-type instruction is used to eliminate the extra occurrence

CLASS I

| | |
|---|---|
| [E xxx/ | Erase item xxx. |
| [E xxx,yyy,zzz/ | Erase line yyy of item xxx. |
| [E xxx,yyy,zzz/ | Erase lines yyy through zzz of item xxx. |
| [I xxx,yyy/...insertion.../ | Insert the text between the slashes after line yyy of itemxxx. |
| [O xxx,yyy/...substitution.../ | Overwrite line yyy of item xxx with the text between the slashes. |
| [S xxx,yyy/ expression A (substitution) expression B/ | |
| [S xxx/ expression A (insertion)/ | |
| [C xxx,yyy/carriage return.../carriage return | Correct line yyy of item xxx on a character by character basis. |
| [U xxx,yyy/carriage return.../carriage return | Underline specified characters in line yyy of item xxx. |
| [R FILE, Text, N,www,xxx/ | Read request. |
| [G FILE, Text,www,xxx/ | Galley proof request. |
| [T FILE, Text,www,xxx/ | Type final copy. |
| [F | Fixed format. |
| [J | Normal mode (justify). |
| [P | Paragraph (used in normal mode). |
| [X www,xxx/ | Randomize the order of items www through xxx. |
| [M www,xxx,n/ | Multiprocess items www through xxx using routine PROC n. |

CLASS II

| | |
|---|---|
| [v n/ | Space vertically n times |
| [h n/ | Space horizontally n times. |
| [m a,b/ | Margins: a spaces on left, b spaces on right |
| [p | Go to new page. |
| [l n/ | Type on every nth line |
| [e (x,n)...(y,m)/ | Equate character x to symbol number n, etc. (to extend character set) |

Figure 2. Instructions for the editing program.

of the characters "t ype" and to change x to c two lines back from the instruction. The importance of this type of instruction is that the modifications can be made at the location of the error. After the first slash and carriage return, the platen is rolled back two lines, and the space bar used to find the proper position. Then the erroneous characters are deleted by typing minus signs, are ignored by spacing, or modified by overtyping. The second slash and carriage return terminates this instruction and typing resumes. The instructions [S#/ and p(r)/ at the end of the item inserts the missing r in the word "poofreaders". Note that a line number is not mandatory in the S instruction, and when it is missing, the search for the first occurrence of expression A begins with the first line of the specified item.

Item 14 gets off to a false start and is erased by [E#/. The erasing in this case goes only as far as the slash of this instruction and does not wipe out the material which follows in this item.

```
(MEMO)(DESCRIPTION)
[1 10/[F
II.  Description of System
Objectives/ [U#,#/
The purpose of this system is three-fold:
(1) to allow for input and correction of typewritten text;
(2) store material for future editing and/or processing;
(3) to provide high quality typographical output.
[0#,1/II.   Description/
[112/[JThe significance of such a system lies primarily in point
two for although typists and poofreaders can prepare manuscripts
and t-ype-typesetters can produce aesthetically pleasing copy,
no one relishes the task of altering previously prepared material.
[C 12,-2/              /
Consequently, the ability to store text in a form that can
be easily updated represents a great convenience.  Therefore,
the aim of this project has been to develop a system of
editing in which the user can provide input to a computer
by a device which is familiar and easy to use.  [S#/and p(r)/
[1 14/Furthermore, the [E#/The corrections themselves
constitute input.  [S12,2/point (@"(2)" )for/
[19.5  E10,8/[010,2/II.  Description/
[1 11.5 [S12,8/venience.([P      )There/
[1 20 [R MEMO,DESCRIPTION,-1,6,8
[1 25 [RMEMO,DESCRIPTION,100,28.5/
[1 30 [R MEMO,MANUAL,0,0,0/
```

Figure 3. Example of input text for editing.

The next instruction, which begins [S12,2/, gives an example of super quoting. It is intended that there be a substitution of (2) in place of the word two in the second line of item 12. However, this substitution involves parentheses which will interfere with the interpretation of the instruction. Therefore, the three characters (, 2, and) need to be super quoted. For this purpose a character other than (, 2, or), for example, the " is chosen to surround the phrase and the surrounded phrase is preceded by @. The @ and the two occurrences of " in the resulting string @"(2)" will not appear in the text, but they will cause the PRE EDIT program to flag the super quoted characters with an extra bit thus preventing the MAIN EDIT from making its usual interpretation. The printing programs, however, will ignore the extra bit and will treat these characters properly.

It has been pointed out that an important aspect of this editing system is that the material is stored on the disk of the computer for future modification and processing. The provision for requesting preexisting material, which is to be updated or quoted, is made by means of a read instruction. These read statements must be made as items in themselves. They specify the file name, the text name of the desired items, and up to three parameters, which specify how and what items are to be read. The first parameter indicates how the desired items are to be numbered. A negative value indicates that the numbering should start with number of the item in which this read statement occurs, a value of zero indicates that the item should be given the number it already possesses, and a positive value indicates that the new item should be given a number equal to its original number incremented by the value of this first parameter. This last technique of numbering is useful when merging items from several texts into a new one. The second and third parameters indicate which items are to be read. If, however, the third one is omitted, only one item will be recovered, and if both of these parameters are zero, all items in the text will be recovered. In the example of Fig. 3, item 20 calls for the items from text DESCRIPTIØN whose numbers range through values 6 to 8. They will be numbered sequentially 20, 21, . . . under the assumption that it is known that there are no more than five items since the next read request is in item 25.

### *Corrections to Instructions*

Corrections to instructions are handled in several ways. First of all, the backspace-retype method, which is effective for all errors, is the one and only means for correcting errors in the first line or read statements. This restriction is due to the fact that these statements are interpreted in the PRE EDIT and do not enjoy the benefits of the editing facilities. Several instances of apparent overtyping may be noted in Fig. 3. They are, in fact, cases of erasure and retyping. Errors can and will be made in the typing of other instructions, and these mistakes may not be discovered until some time later after which the backspace-retype means of correction is no longer practical. Corrections to Class I instructions (those designated by an upper case code) must be made ahead of the erroneous instruction. Facility for allowing instruction A to be executed in advance of instruction B is provided by placing instruction A in an item whose number is less than that of the item containing instruction B. The data is scanned in order of increasing item number, and therefore, the instructions will be encountered in their proper order. The decimal numbering system of the items allows for the prenumbering of items.

In this way corrections are effectively made on the future as well as on the past.

The mistake made in the overwrite instruction of item 10 (Fig. 3) is corrected by item 9.5. Here the old instruction is erased and the proper form given.

*Paragraphing*

Paragraphing must be handled by instruction when typing in the normal mode since the program is concerned about the associated blanks indicating indentation. A desired paragraph is indicated by the instruction [P which may or may not be followed by a carriage return. In either case the blanks which follow will be regarded as the indentation. Should the user decide upon a paragraph as an afterthought, he may "wire in" the instruction. For example, item 11.5 of Fig. 3 shows [P and five blanks being inserted after the word convenience in item 12.

Changes which control format of output, i.e., corrections to Class II instructions, offer no problems as to when they are inserted relative to their proper position since their execution occurs in the print pass after the editing has been completed. Hence, these instructions can be "wired in or out" at any time in the editing program by use of the correction instructions. The S type is particularly useful for this purpose.

Figure 4 shows the galley of the edited text of Fig. 3, and Fig. 5 is the final copy as produced by the output program.

## OSCILLOSCOPE WITH DIGITAL CONTROL FOR TYPESETTING AND GRAPHIC ARTS

*General Description*

This section describes a system for the digital control of a high-quality oscilloscope for the purpose of generating graphic arts quality images such as are needed for printing text and line drawings. In general the images will be photographed and the resulting pictures reproduced by the standard methods of offset printing. The input information which specifies the image will come from a digital magnetic tape or a computer. On the input, the image is described entirely in numerical form in the manner

FILE- MEMO STANDARD TEXT - DESCRIPTION---------- 07/12/65 21:39:12

10.00 [F

II. Description

Objectives

The purpose of this system is three-fold:

(1) to allow for input and correction of typewritten text;

(2) store material for future editing and/or processing;

(3) to provide high quality typographical output.

12.00 [[The significance of such a system lies primarily in point (2) for although typists and proofreaders can prepare manuscripts and typesetters can produce aesthetically pleasing copy, no one relishes the task of altering previously prepared material. Consequently, the ability to store text in a form that can be easily updated represents a great convenience.[P

Therefore, the aim of this project has been to develop a system of editing in which the user can provide input to a computer by a device which is familiar and easy to use.

14.00 The corrections themselves constitute input.

Figure 4. Galley proof.

## II. Description

### Objectives

**The purpose of this system is three-fold:**
**(1) to allow for input and correction of typewritten text;**
**(2) store material for future editing and/or processing;**
**(3) to provide high quality typographical output.**

**The significance of such a system lies primarily in point (2) for although typists and proofreaders can prepare manuscripts and typesetters can produce aesthetically pleasing copy, no one relishes the task of altering previously prepared material. Consequently, the ability to store text in a form that can be easily updated represents a great convenience.**

**Therefore, the aim of this project has been to develop a system of editing in which the user can provide input to a computer by a device which is familiar and easy to use. The corrections themselves constitute input.**

Figure 5. Justified text.

to be discussed below. The motion of the beam of the scope is completely determined by the numerical data. The intensity and off-on times of the beam are similarly controlled.

A block diagram of the system is shown in Fig. 6. The general mode of operation is as follows: Data, in numerical form, which specifies the next

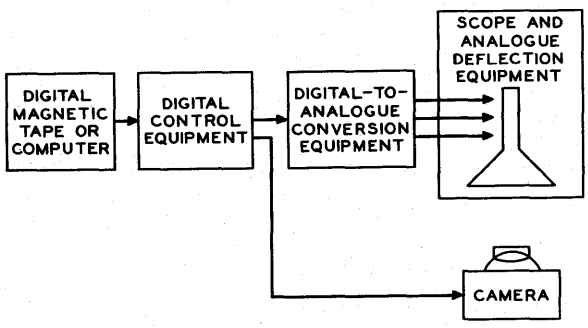



Figure 6. Block diagram of image-generating equipment.

letter or diagram to be generated is read from the digital magnetic tape or computer. The digital control equipment then determines the sequence of movements of the scope beam to produce the image. Typically, many (10 to several hundred) beam movements will be needed to produce a single character. Digital signals from the digital control unit are passed through digital-to-analog converters to obtain the voltages to be applied to the $x$ and $y$ scope deflections and to the beam brightness and off-on controls. The digital control also advances the film in the camera at appropriate times. The camera film is stationary and the shutter is left open until a full scope face of material is exposed. This amount will usually correspond to a page of printed text. The film is then advanced to provide a fresh film for the next page.

### *Preliminary Experiments*

A set of preliminary experiments has been carried out using a Stromberg-Carlson 4020 Microfilm Printer and an IBM 7094 computer. The SC 4020 contains a scope and camera of sufficient quality to test the feasibility of producing printing and drawings. A sample alphabet produced on microfilm on the SC 4020 and reproduced photographically is shown in Fig. 7. Here each of the letters in the bottom 6 lines was produced by a number of short vec-

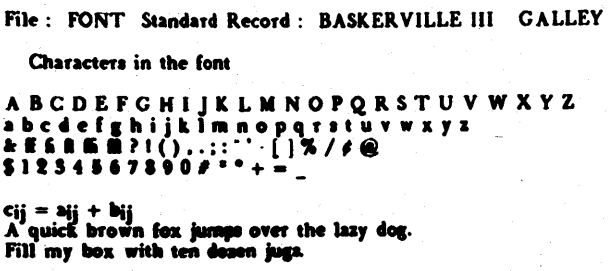



Figure 7. Example of type font.

tors, an average of about 16 vectors being required for each letter. In this way a highly readable output is obtained with both upper and lower case letters. Furthermore, an unlimited number of new fonts can be introduced simply by reading a new set of vectors which describe the letters into the IBM 7094. Also graphs and line drawings can be treated just as any other character.

The quality of the letters produced with vectors is not as good as the usual printing. Readability probably compares with an ordinary typewriter. However, the SC 4020 scope is far from the best available, and better equipment should greatly improve the quality.

*Estimates of Resolution Requirements*

The SC 4020 can plot points, dots (slightly larger than points), or draw vectors. The points and dots can be placed at any position on a 1024 x 1024 raster. The vectors can start at any raster point and extend up to 64 grid spaces in either or both $x$ and $y$ directions. Measurements on the image of the SC 4020 indicate resolutions of

Point . . . . 2.6 grid spaces diameter in 1024 raster
Dot . . . . . 2.8 grid spaces diameter in 1024 raster
Vector . . . 2.3 grid spaces width in 1024 raster

The letters shown in Fig. 7 were produced on a matrix 16 raster points high and an average of 10 raster points wide. Consequently the letter resolution can be defined as 4.3 vector widths wide and 7 vector widths high. This resolution is sufficient for a quality whose readability is probably comparable to typewriting (although we have made no tests of this point). However, the quality is substantially less than that of good printing. We estimate that first-rate printing could be achieved by tripling the SC 4020 resolution and producing the letters on a 21 × 13 resolution unit grid. On the SC 4020 this would correspond to a 49 × 30 raster point grid using the vector mode or a 59 × 36 grid if the dot were used.

In addition to resolution compared to letter size another factor is very important. This is quantizing or the comparison of raster space to dot diameter. In the vector mode the ratio is

$$\frac{1 \text{ Raster Space}}{1 \text{ Vector Width}} = 0.43$$

This ratio is too large for some purposes. In particular, it would be nice to be able to change the size of a letter simply by multiplying the lengths of all involved vectors. Failure of attempts to change size indicate that a quantizing ratio of 0.43 is much too large to allow this scaling, and hence a smaller ratio is desirable. The exact ratio needed may also depend on the resolution raster used for the letter. We estimate that the ratio should be no larger than 0.25.

*A High-Quality Scope*

The SC 4020 scope has a basic resolution of at best 2.3 parts in 1,024. The scope has a built-in character mask and was not designed for ultimate resolution. Better scopes are now available. The best specifications quoted by scope manufacturers indicate beam widths of 0.0005″ to 0.001″ are available in 5″ to 10″ diameter scopes. Assuming that a 5″ usable deflection can be obtained with a beam width of 0.001″, then a basic resolution of 1 part in 5,000 can be achieved. This is 11 times better than the SC 4020 and should produce excellent graphics if it is properly used.

*Proposed Raster and Resolution for Quality Printing*

As a result of the preliminary experiments described above, and the specifications of currently available scopes, specifications for a high-quality graphics system can be set down. Scheme #1 consists of a raster of 32,768 points across a tube with a resolution of 1 part in 5,000, and the letters being generated on a 140 × 85 matrix of raster points. This would give a 21 × 13 matrix of resolution circles for each letter. The quantizing ratio would be 5,000/32,768 = 0.15, which should be adequate for smooth size multiplication.

A simulation was carried out on the SC 4020 in which letters were generated from points on a 64 × 38 raster giving a 25 × 15 matrix of resolution circles. The results shown on Fig. 8 indicate quite acceptable printing quality.

**An electronic system drew, then set in words, sentences, and justified columns the letters you are now reading.**

Figure 8. Example of better quality type.

*Image Formation From PATCH's*

In the work previously described, which has moderate resolution, images were formed from the sum of short vectors or from the sum of dots. The vector width or dot diameter is the resolution of the scope. With high resolution scopes, the beam is so small that it is no longer practical to use a single beam image as the basic area unit. Instead, special

deflection equipment must be added to the scope to cause the beam to sweep over a basic area unit or PATCH (Parameterized Area To Construct Holograph). All images are constructed as the sum of a number of PATCH's. In this way the number of digits required to describe the image is reduced to a reasonable number, and the speed with which the image is generated is increased.

PATCH's must meet certain requirements. They must:

1. Fit together without leaving interior spaces. Circles would be unsuitable by this criteria.
2. Provide a good approximation to an image with a small number of areas.
3. Be describable with a reasonable number of numerical parameters.
4. Be generatable with reasonably simple analog equipment.
5. Be enlargeable or reducible to allow changes in font size.

An area which meets these requirements is shown in Fig. 9a. The area is bounded by straight lines at its top and bottom and second order curves at its left and right edges. Adjacent PATCH's can be stacked on their straight sides. Simple circuitry has been designed to generate the PATCH. If desirable the PATCH can be rotated 90° to provide vertical straight sides. Rectangles and trapezoids are special cases of this area. Eight parameters are required to describe a PATCH; width—a, height—h, curvature and slope at left and right bounds—$C_1$, $S_1$, $C_1$, $S_1$, and the coordinates of one corner—$X_0$, $Y_0$.

A sketch of a generation for the lower case letter "r" is shown in Fig. 9b. Nine PATCH's are required. Some PATCH's have been rotated 90° as shown.

Preliminary experiments indicate an average of 10 PATCH's is required for each good quality letter. Thus a font of 100 leters would require 8,000 numbers for its description. This number is substantial, but not too large for currently available computer memories.

The letters on Fig. 8 were produced with PATCH's.

### *Use of Sub-Areas*

In order to reduce computer memory requirements, it may be possible to take advantage of identical sub-areas possessed by several letters. Thus the loops on the "b," "p," "d," and "g" may be identical (after suitable rotations) in some fonts. If so, then these areas need be described only once in terms of PATCH's, and suitable equipment devised to repeatedly call for these standard areas. The concept is similar to the use of subroutines in computer programs.

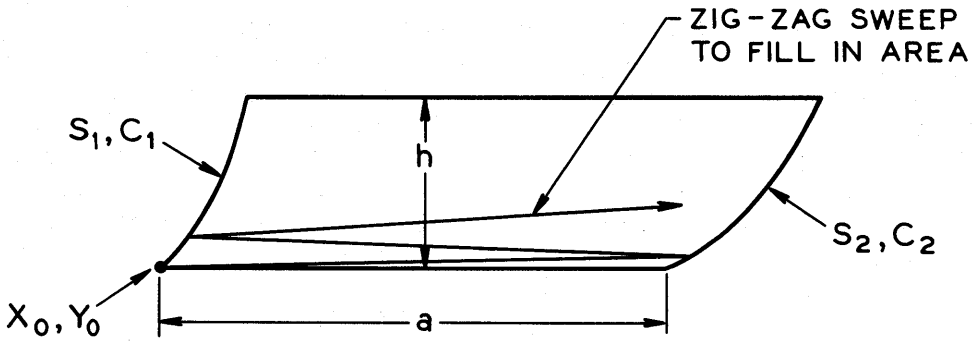
## CONCLUSIONS

The general philosophy behind the development of the editing program has been to provide a human-engineered facility for producing text in machine readable form so that a computer can be used for editing, sorting, and printing. It is hoped that timesaving in composing will be effected by eliminating much proofreading time inherent in a system involving human copying. The ability to modify and republish existing material is probably the most valuable feature in the system.

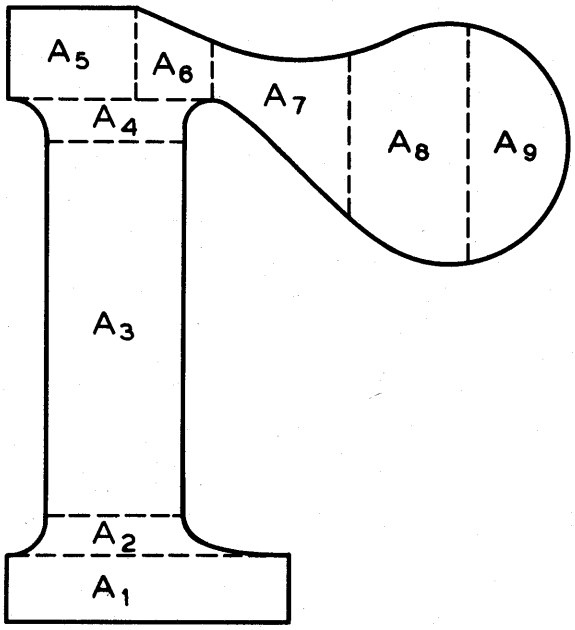
It is planned to first use the program for texts which must be issued in many slightly different editions or which must be frequently modified. Certain instructional programs and literature describing computers are prime examples.

The image generation programs have not yet been tested with the best available scopes. However, the experiments with the SC 4020, the specifications of the best scopes, and the currently available digital equipment make it appear possible to build a high-quality, high-speed, graphic arts display device. The unit can produce on a single scope face high quality printing with as much as 200 lines of 350 letters each. Such a scope face would contain more than an entire page of newsprint. Letters could be produced at speeds of 500 to 5000 letters/second. These speeds are 10 to 1,000 times faster than existing photographic or hot metal typesetting equipment. Furthermore, since the image is described in complete generality digitally, line drawings, mathematical equations, musical scores, and an unlimited number of type fonts can be produced by the same, completely standard, means.

Gutenberg invented printing with movable type in the 15th century, thus superseding handdrawn letters. We are now ready to replace movable type with drawn letters. The pen is in the hand of a computer. Altogether, we believe the computer is now ready to provide great assistance to human written communication.




(a) PATCH




(b) FORMATION OF LOWER CASE "r" BY PATCHS

Figure 9. Generation of images from PATCH's.

# THE LEFT HAND OF SCHOLARSHIP: COMPUTER EXPERIMENTS WITH RECORDED TEXT AS A COMMUNICATION MEDIA

Glenn E. Roudabush,
Charles R. T. Bacon,
R. Bruce Briggs, James A. Fierst, and
Dale W. Isner
*The University of Pittsburgh*
*Pittsburgh, Pennsylvania*
and
Hiroshi A. Noguni
*The RAND Corporation*
*Santa Monica, California*

To paint a broad though much simplified picture, let us suppose at the outset that scholarship begins with the collection of facts. These facts are of two distinct kinds. The first are observations and they consist, for example, of the results of controlled experiments or observations for field work in the case of science or, perhaps, they are derived from the study of historical documents in the case of history, and so on. The second kind of facts are the *reported* observations, descriptions of phenomena or events, or the theories provided by contemporary scholars. In aggregate, let us refer to the first kind of facts as "data" and the second as "information." From the confluence of these two kinds of facts in the mind of the scholar, new descriptions and theories are born. When he makes these public, then new information is generated.

Scholarship, strictly conceived, is this activity in the mind of the scholar. On its right hand are sources: data and information. On its left are publications: the products of this activity made public. But these two sides of scholarship are closely related. What to one scholar is a publication, to another is information. Every scholar stands both to the right and to the left of every other one.

In our text processing work at the University of Pittsburgh, we look upon our computers and our developing system of programs as a tool designed to extend the abilities of the scholar, on the one hand to collect, sort, and understand information, and on the other to disseminate to others the information that he generates. In other projects and for most of the users at our Center, our computers and systems of programs are seen as a tool to extend the ability to process and analyze data. These systems are, of course, well developed. In analyzing data, one's concern is to reduce, to simplify, and to summarize, preserving only the most significant aspects of the data. While in processing information, we wish to preserve every jot and tittle, allowing no character-

istic of any significance to go unrecorded or untransmitted. Finally, in the research we ourselves do that utilizes natural language text, we come full circle and again use our systems as data processors and analyzers, treating the information we have collected as data.

Figure 1 shows schematically the overall design of our text processing system. Four kinds of input

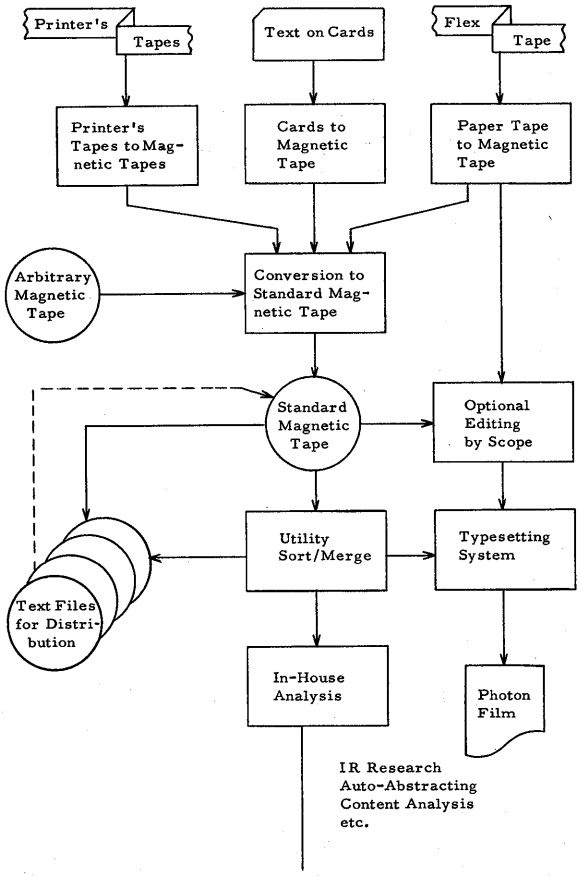



Figure 1. Block diagram of the general text processing system.

are shown. The text on magnetic tape in any arbitrary format may be material obtained from other centers or from any source that produces text on tape. One day this source may include material read by optical character recognition equipment. The printer's control tapes are paper tapes obtained from printers and publishers which were originally used to control some kind of typesetting equipment. We have locally constructed a paper tape reader that will accept 5, 6, 8, 15, and 31-channel paper tape and, through an IBM 1401 computer, write magnetic tape. This work was completed under a Department of Defense Advanced Research Projects Agency grant and has been reported elsewhere.[1,2] The text punched on cards or on Flexowriter-type paper tape would normally represent material prepared at our Center.

The block labeled "conversion to standard magnetic tape" represents the encoding of all forms of natural language text into a particular format according to a schema devised by Martin Kay and Ted Ziehe of the Rand Linguistics Research Group. A relatively complete, but still preliminary description of this format has been published as a Rand Memo.[3] The use of magnetic tape for storage of text and the use of this standard format are prominent in our system and more will be said about this in a moment.

Some source text in exceptionally good condition may, after encoding in this standard form, be ready for distribution to other centers requesting it or for use in our own research. Characteristically, however, some additional processing will be required and this is represented in the block labeled "utility." At the bottom of this figure, our use of text as data is represented. Under "in-house analysis" we have listed information retrieval research, auto-abstracting, and content analysis as examples of this kind of work.

The series of blocks down the right side of Fig. 1 show the normal sequence of operations for photocomposition. Material to be photocomposed will, in most cases, be specifically keyboarded for that purpose. This material will be under good control from the beginning and can go directly into the typesetting system unless it will be used for other purposes as well. Sorting, editing, and other processing will generally not be required so that the conversion to standard format can be bypassed. Both kinds of input to the typesetting system are allowed. An expanded block diagram of the typesetting system itself will be shown in a later figure.

Our system depends to a large extent on the efficient processing of large amounts of natural language text on magnetic tape and this aspect of our system will be described in somewhat greater detail. Magnetic tape is, of course, an economic storage medium and is easily shipped between geographically separated centers. Encoding all text in one standard format becomes important when many different kinds of text from many different sources must be processed and shared. When standardized

input can be expected, a smaller number of general programs can be written and a useful library can begin to be accumulated. The standard adopted must be flexible enough to handle any material one may encounter. The Rand format seems to fill all of our current and anticipated requirements and we have adopted it for our system.

On seven-channel magnetic tape, the minimum unit is a six-bit pattern plus a parity bit. In a one-to-one character representation, only 64 unique characters can be defined. In order to extend the number of different characters that can be represented on tape, either more than one six-bit pattern can be assigned to each character to be represented or else, as in the Rand standard format, some of the available 64 patterns can be used to change the meaning of the patterns that follow them on tape. These mode change patterns or characters are of two kinds: "flags" and "shifts." The flags change the interpretation of succeeding patterns to a new alphabet, while the shifts retain the same alphabet, but mediate changes to, for example, upper case, italics, larger type size, and so on.

Fifteen of the available 64 patterns are permanently assigned as alphabet flags in the Rand system. These 15 patterns along with the blank (octal 60) and a filler character (octal 77) are not a part of any alphabet and their interpretation never changes. There are, then, 47 patterns which can be assigned meanings in each of the 15 alphabets. In each of the 15 alphabets, some of the available 47 patterns will be assigned mode change functions as shift characters. In the Roman alphabet, for example, nine patterns are used in this way. The remaining 38 patterns can accommodate the 26 letters, 10 diacritic marks, and the apostrophe with one pattern left unassigned. Notice that separate alphabets must be used for punctuation, the numerals, and other symbols occurring frequently in the English text.

This encoding system gives a flexible representation of the micro-characteristics of text. Larger units of text, however, have a hierarchical organization which also requires representation. This is accomplished in the Rand system by the "catalog" format. The fundamental unit in this system is the datum, which can be thought of as a manipulable unit of information. A datum may be a text entry consisting of one physical line of text if from a previously printed source, or one sentence, or one word if that is convenient, or it may be a title or a caption from an illustration, or an annotation or description of another datum added at a later time. Each datum belongs to a particular class and at the beginning of each reel of tape following a label record, a map of the corpus is given describing the various classes of material contained in the file. Each datum is coordinated with this map and its proper identification assured by a system of control and label words accompanying every datum. A representation of the Rand encoding system will be shown later in our second typesetting example.

We in Pittsburgh became interested in automatic photocomposition when, in October of 1964, we acquired a Photon S-560 photocomposition machine from the National Institutes of Health. This machine had previously been used by Michael Barnett at the Massachusetts Institute of Technology under an NIH grant. The Photon is an electromechanical device driven by punched paper tape. It consists essentially of a movable glass disk with 1400 characters etched on it and a lens system for projecting these characters onto roll film. The disk can accommodate 16 different type fonts arranged in eight concentric circles or levels around the disk. The paper tape is punched with double character codes, the first giving the character position within disk level and the second giving the escapement for that character. There are additional codes for advancing the film, positioning the film carriage horizontally, affecting lens shifts for size control, and effecting shifts to new disk levels for font changes.

When wereceived the Photon, we also acquired the PC6 system of automatic photocomposition programs developed under the direction of Barnett while he was at M.I.T.[4-8] The PC6 system is typified by the TYPRINT program which requires text containing fixed typesetting control codes as input. These codes are set off from the text by square brackets, which are reserved, and have fixed meanings as shown in the following examples:

[NP ] New Paragraph
[DL6] Shift to Disk Level 6 (Highland type face)
[VL2] Leave 2 Blank Lines

In using this system, we soon found that the insertion of fixed codes can be laborious, that changes in format require changes throughout the text, and that many desirable formats are impossible to achieve. We felt that a more flexible and more gen-

erally useful system of programs could be written. We still believe, however, that the PC system was a successful first step toward automatic photocomposition and, in general, the typesetting system we have developed is an outgrowth of our experiences with it.

The input to our system is either magnetic tape in an arbitrary format produced from paper tape punched specifically for typesetting or else magnetic tape in the Rand standard format. The output is again paper tape that will drive the Photon. A schematic diagram of this system is shown in Fig. 2. In this figure, the two forms of magnetic tape input

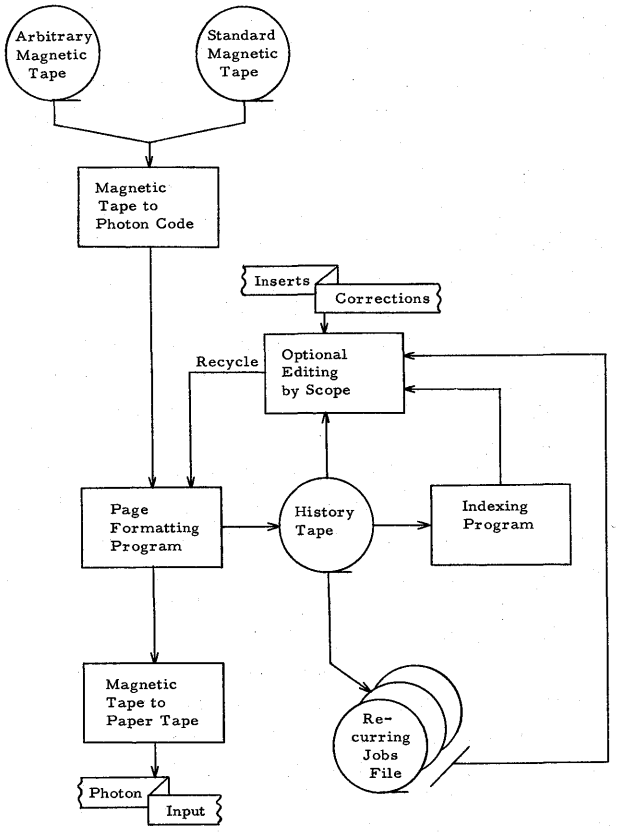



Figure 2. Block diagram of the typesetting system.

are shown at the top. The typesetting program is shown as two separable functions. The first part, which translates text into the double character Photon code, is relatively independent of the second part, but is quite dependent on the particular photocomposition device being used, that is, on the Photon. This part would be largely rewritten if a new piece of equipment were obtained. It is, however, a rather simple and straightforward program. The second part, labeled the "page formatting program," represents a real departure from the PC6 system and other typesetting systems we have seen. In this program, a full page of text is set before outputting is begun.

The page formatting program shows two forms of output. The first is a magnetic tape which contains Photon input that will be converted to paper tape. The other form of output labeled the "history tape," is a magnetic tape containing the original text characters with their associated Photon codes, all of the material added by the page formatting program, page and line numbers, and sufficient parametric information to reset the material exactly as it was originally done. This tape can be recycled through the page formatting program with corrections or additions to the text or simply with changed parameters if the format is to be changed. Since page numbers, tables, captions for figures, titles and subtitles, and so on are all in their proper place on this tape, it can be used as input to a program that produces indices and tables of contents. Finally as shown, this tape might simply be stored for a period of time and then recycled when a new edition is to be set.

This history tape is an important by-product of computerized typesetting and may well be a critical factor in making the adoption of an automatic system economically feasible. This tape is essentially an exact copy of the printed material, less illustrations which cannot be handled in our system, and is a compact, machine-readable counterpart of the standing type that occupies space in some print shops and warehouses. Any material in this file can be simply addressed by page and line number from the corresponding printed document and changes made. If a change is made that affects the remainder of the file, for example an insertion that affects the pagination, all of the file will automatically be corrected.

In designing this system, we came to the conclusion that typesetting control codes in the text to be set are necessary if any format flexibility is to be obtained. They, therefore, appear minimally in our system. We have tried at the same time to ease the burden of keyboarding these codes and of changing their meaning in pre-prepared text by making them entirely arbitrary. The text-dependent codes can be though of simply as markers. The actions to be taken when particular codes are encountered are separately specified as parameters to the system.

These parameters can be inserted anywhere in the text ahead of the markers to which they refer, or they can be punched on parameter cards. If they are keyboarded with the text, they are normally marked off by dollar signs or some other specified reserved symbol. The form of the printd output can be completely changed by changing these parameters with no re-editing of the text itself.

In our system, we wished to include the ability to control as much as possible the layout and final form of the pages in the manuscript. We felt that the deficiencies of other systems in this respect stemmed from their line-by-line typesetting. The attempt to visualize a page by as yet undefined lines is difficult and usually leads to a number of unnecessary trial runs. To ease this difficulty on the programming level, we set full pages. On the conceptual level, we conceive of a page as a collection of subpages or "boxes." A box is a string of fixed text delimited by two markers. The material within a box can be set independently of other material as though it were itself a page and then the box of fixed material placed in its proper position on the page. The box system is recursive so that boxes may be defined within boxes and for most functions, overlapping is allowed.

The parameters used to control the system are of three types: (1) general parameters, (2) text boundary parameters, and (3) box parameters. A list of the general parameters is shown in Fig. 3.

Most of these parameters control the general appearance of the printed output. They include the specification of page size, number of columns on the page, type face, point size, and so on. The parameters specifying running page headers include a provision for incorporating page numbers that are automatically incremented. The last two parameters are provided to make the keyboarding somewhat simpler. The DLIM code allows the specification of any character to mark off parameters when these are included in the text in place of the preset dollar sign. The DEL code allows any character to be specified as a deletion code. It causes a character over which it is typed to disappear from the input string. Only those parameters that are to be different from their preset values need be specified.

The following list of general parameters:

$ PSIZ(8. 5, 11), TFAC(SCOTCH), TSIZ(10), HEAD(Page /1/), COL(3.5, 1.5, 3.5) $

would specify 8½ by 11 inch pages to be set in

| SYMBOL | MEANING | NOTES |
|---|---|---|
| PSIZ(x,x) | Page SIZe | Page size is width by height. |
| COL(x,x,x...) | COLumns | Column widths and margins alternate. |
| JUSV(s) | JUStification-Vertical | Reserved words such as center,spread,etc. |
| JUSH(s) | JUStification-Horizontal | are used to indicate action desired. |
| TFAC(n,i,b,B) | Type FACe | n,i,b,B are names of type fonts. |
| FONT(f) | FONT | Used to indicate italic or bold type. |
| TSIZ(p) | Type SIZe | Type size is given in points. |
| BGND(p) | Back GrouND size | Background size is also in points. |
| TAB(x,x,x...) | TAB | Tab setting measured from left margin. |
| MWS(p,p) | Minimum Word Spacing | Minimum distance between words. |
| XWS(p,p) | maXimum Word Spacing | Maximum distance between words. |
| HEAD(t) | HEADer | Headers may be any string of text,it |
| LHEAD(t) | Left HEADer | will be set on both pages. LHEAD and |
| RHEAD(t) | Right HEADer | RHEAD are set on respective pages only. |
| DLIM(c) | DeLIMiter | Used to surround instructions in text. |
| DEL(c) | DELetion character | Removes unwanted characters when backspacing. |

x is a dimension expressed in inches.

p is a dimension expressed in points.

t is any string of text and may include any boundary or box markers.

c is any keyboardable character and should be one which is not normally used in the text.

n,i,b,B are the names of type faces available. They determine which type face will be used for normal,italic, bold-face and bold-face-italic letters.

s may be one of the following reserved character strings CNT(CeNTer), LFT(LeFT), RGT(RiGhT), SPR(SPRead), BTM(BoTtoM), TOP.

f may be one of the following NOR(NORmal), ITAL(ITALIC), BOLD, or BOLD-ITAL.

Figure 3. List of general parameters.

10 point Scotch with two 3½ inch wide columns separated by 1½ inches. The running heads "Page 1," "Page 2," and so on would print at the top of successive pages. Since the background size and the minimum and maximum word spacing were not specified, reasonable values for these would be computed by the program based on the type size and line length. Hyphenation would occur if the lines could not be justified within these computed limits.

The general form of the text boundary parameters and the box parameters are shown in Figure 4. The text boundary parameters specify a particular arbitrary text marker and a list of actions that are to occur when that marker is encountered in the text. The box parameters specify two particular arbitrary text markers which will delimit fixed strings of text to be treated as a box and a list of actions describing the way material in the box is to be set and the placement of the box on the page. The lists of actions in each of these two parameters can include any of the general parameters or any of the additional actions listed in this figure.

FORM of the TEXT BOUNDARY PARAMETER:

$ AT k)CS(y P,P,P,...,P $

FORM of the BOX PARAMETER:

$ FROM k)CS(y to k)CS(y P,P,P,...,P $

CS is any arbitrary character string.

k is the letter O if the string between the closed and open parentheses specifing the text marker is in octal representation, otherwise it is blank.

y is blank if the marker is not also part of the text, it is S(Save) or SIN(Save IN box) if it is a part of the text.

P may be any general parameter or any of the following.

| SYMBOL | MEANING | NOTES |
|---|---|---|
| TAB(w) | TAB | Allows indenting to a predefined tab. |
| SKIP(z) | SKIP | Allows Vertical spacing. |
| MAR(x,x) | MARgin | Allows margins to be reset. |
| BSIZ(x,x) | Box SIZe | Specifies dimension of box. |
| BTAB(w) | Box TAB | Determines horizontal position of box. |
| BSKIP(q) | Box SKIP | Determines vertical position of box. |
| UNIT | UNIT | Forces box to be put on one page,i.e., not split. |

z may be one of the following reserved symbols NL(New Line), nL(number of Lines), NC(New Column), nC(number of Columns), NP(New Page), nP(number of Pages), nI(number of inches), nPT(number of PoinTs).

q may be the same as z plus TOP BTM(BoTtoM) or CNT(CeNTer) which means the box should be placed at top, bottom or center of the column or page.

w may be a number referring to the nth tab defined plus (for BTAB) LFT(LeFT), RGT(RIGhT) or CNT(CeNTer) which means box should be even with the left, center, or right side of current column or page.

x may be a number (inches) or T (depends on text).

Figure 4. Form of the text boundary and box parameters.

The text markers are defined in these two kinds of parameters as the binary coded decimal equivalent of the character string appearing between the close paren on the left and the open paren on the right. The octal equivalent of the six-bit binary character may also be placed between the parentheses, in which case the letter "O" preceeds the marker specification. A marker may be any string of characters that will not be confounded with text material. They may themselves be a part of the text to be set. If this option is desired, the letter S or SIN (for Save or Save IN) is appended to the marker specification. If S is used, the characters making up the marker are considered to come before the marker or outside the box. If SIN is used, they are considered to come after the marker or inside the box. These conventions give some format control over material that has no keyboarded codes at all.

As a first example of the operation of this system for a straightforward problem, we have taken a part of the "Recent Publications on Computational Linguistics" section of *The Finite String* for June 1965. This monthly newsletter is a publication of the Association for Machine Translation and Computational Linguistics and the short bibliography section has been photocomposed at our Center since April of this year. The procedure we actually use with this material differs somewhat from this description because the text is keyboarded on a Dura Machine 10 at the Rand Corporation rather than at our Center. The differences, however, are minor. In Fig. 5, the Flexowriter hard copy is shown with the parameters appearing at the top of the page. Only

```
$ PSIZ(6.75,10), TFAC(HIGH,HIGH ITAL,CENT BOLD), TSIZ(8),
  XWS(100,100) $


$ FROM  )**(  TO  )**(  TSIZ(9), BSKIP(3L), FONT(BOLD) $
$ FROM  )*ENT(  TO  )*ENT(  SKIP(2L), UNIT $
$ FROM  )[(  TO  )](  FONT(BOLD) $
$ FROM  )/(  TO  )/(  FONT(ITAL) $

**Computational linguistics: Glossaries**

*ENT [Nozaki, A.] "On the Dictionary Preparation,"
manuscript, presented at the U.S.-Japan Seminar on Mechanical
Translation, New York, May 1965. *ENT

*ENT [Lehmann, W.P., and Pendergraft, E.D.]
/Quarterly Progress Report,/
1 November 1964 - 31 January 1965, LRC 65 NSF-23,
Linguistics Research Center, The University of Texas,
Austin, Texas, January 1965. *ENT

*ENT [Nagao, Makoto] "Japanese-English Translation Regarded as
Sentence Generation," manuscript, presented at the
U.S.-Japan Seminar on Mechanical Translation, New York, May 1965. *ENT

*ENT [Otkupshchikova, M.I.] "II simpozium po mashinnomu perevodu"
("2nd Symposium on Machine Translation"), /Nauchno-Tekhnicheskaya
Informatsiya,/ No. 12 (December 1964), pp. 34-36. *ENT

*ENT [Pfafflin, Sheila M.] "Evaluation of Machine Translations
by Reading Comprehension Tests and Subjective Judgments,"
/Mechanical Translation,/ Vol. 8, No.2(February 1965),pp.2-8. *ENT

*ENT [Reitz, Gerhard (ed.)] /Improved Syntactic Flowcharts - Research
Output Format,/ Progress Report No. 9, The Bunker-Ramo Corporation,
Canoga Park, California, 31 March 1965. *ENT

*ENT [Sakai, Toshiyuki] "Procedure for the Analysis of Japanese
Texts," manuscript, presented at the U.S.-Japan Seminar on
Mechanical Translation, New York, May 1965. *ENT

*ENT [Satterthwait, Arnold C.] "Sentence-for-Sentence
Translation: An Example," /Mechanical Translation,/
Vol. 8, No.2 (February 1965), pp. 14-38. *ENT

*ENT [Tosh, L.W.] "Development of Automatic Grammars,"
/Linguistics,/ No. 12 (March 1965), pp. 49-60. *ENT
```

Figure 5. Parameters and text for the *Finite String* example.

general and box parameters are required. The general parameters set the page size to 6¾ by 10 inches, the normal type face to Highland, the italic type face to Highland Italic, the gold type face to Century Bold, the type size to 8 points, and the maximum word spacing to 100 points (to preclude hyphenation). Since the background size and minimum word spacing are not specified, computed values will be used. Four boxes are defined. The first encloses subtitles which are spaced three lines below preceeding material and printed in bold face and somewhat larger type size. The second encloses whole bibliographic entries. The associated actions

cause each entry to be treated as a unit, not to be slit between pages, and a line space is left between them. The third encloses the author's name which is to be set in bold face and the last encloses the title of the publication which is to be set in italics. The photocomposed result is shown in Fig. 6.

This first example was shown to illustrate the simplicity of the system when limited format con-

**Computational linguistics: Glossaries**

**Nozaki, A.** "On the Dictionary Preparation," manuscript, presented at the U.S.-Japan Seminar on Mechanical Translation, New York, May 1965.

**Lehmann, W. P., and Pendergraft, E. D.** *Quarterly Progress Report*, 1 November 1964 - 31 January 1965, LRC 65 NSF-23, Linguistics Research Center, The University of Texas, Austin, Texas, January 1965.

**Nagao, Makoto** "Japanese-English Translation Regarded as Sentence Generation," manuscript, presented at the U.S.-Japan Seminar on Mechanical Translation, New York, May 1965.

**Otkupshchikova, M. I.** "II simpozium po mashinnomu perevodu" ("2nd Symposium on Machine Translation"), *Nauchno-Tekhnicheskaya Informatsiya*, No. 12 (December 1964), pp. 34-36.

**Pfafflin, Sheila M.** "Evaluation of Machine Translations by Reading Comprehension Tests and Subjective Judgments," *Mechanical Translation*, Vol. 8, No. 2 (February 1965), pp. 2-8.

**Reitz, Gerhard (ed.)** *Improved Syntactic Flowcharts - Research Output Format*, Progress Report No. 9, The Bunker-Ramo Corporation, Canoga Park, California, 31 March 1965.

**Sakai, Toshiyuki** "Procedure for the Analysis of Japanese Texts," manuscript, presented at the U.S.-Japan Seminar on Mechanical Translation, New York, May 1965.

**Satterthwait, Arnold C.** "Sentence-for-Sentence Translation: An Example," *Mechanical Translation*, Vol. 8, No. 2 (February 1965), pp. 14-38.

**Tosh, L. W.** "Development of Automatic Grammars," *Linguistics*, No. 12 (March 1965), pp. 49-60.

Figure 6. The *Finite String* bibliography.

trol is required. Our second example is intended to show a wider range of the possibilities inherent in the system and in particular, the degree of format control that can be obtained. This example consists of the first three pages of an eight-page booklet on postpartum care prepared for the University-affiliated Magee-Womens Hospital in Pittsburgh. After the first two pages, the booklet has a two-column format with captioned illustrations and the author had an exact picture in mind of the way in which each page was to appear. It therefore formed a good test of the formatting capability of our system.

The illustration was prepared in the following steps: (1) the straight text was keyboarded on a Flexowriter without parameters, codes, or markers of any kind; (2) appropriate text boundary and box markers were added using a display scope editing program to be described in a moment; (3) the text with markers was then converted to the Rand standard format; and (4) the typesetting programs were run using this as input. The text editing program used in step (2) is implied by the blocks labeled "optional editing by scope" in both Figs. 1 and 2. The text editor is a general editing program, not specific to the typesetting system, but we have found it very useful in preparing material for photocomposition. We shall give only a brief account of this program here, since a complete description can be found in Bacon.[9]

The text editor program is written for a small Digital Equipment Corporation PDP-4 computer with 4K words of core storage, a cathode ray tube and light pen, a paper tape reader and punch, and a teletype keyboard. This small computer is interfaced into our IBM 7090 giving it access to the tape units, disk file, and core storage of the larger computer. The interface was constructed locally by Russell Ranshaw of our staff. Input to the text editor can be keyboarded directly or read from paper tape or magnetic tape via the interface. Output can be typed, punched on paper tape, or written on magnetic tape.

The text editor continuously displays selected sections of text on the cathode ray tube and editing functions can be performed on the displayed text using the light pen and keyboard. The display is in two parts as shown in Fig. 7. Along the bottom of the screen, stationary symbols are shown which function as push buttons when touched by the light pen. The remainder of the screen is used to display the text being edited. The size and intensity of the characters in the display as well as the vertical and horizontal dimensions of the display itself can be varied. All of the text held in the computer at one

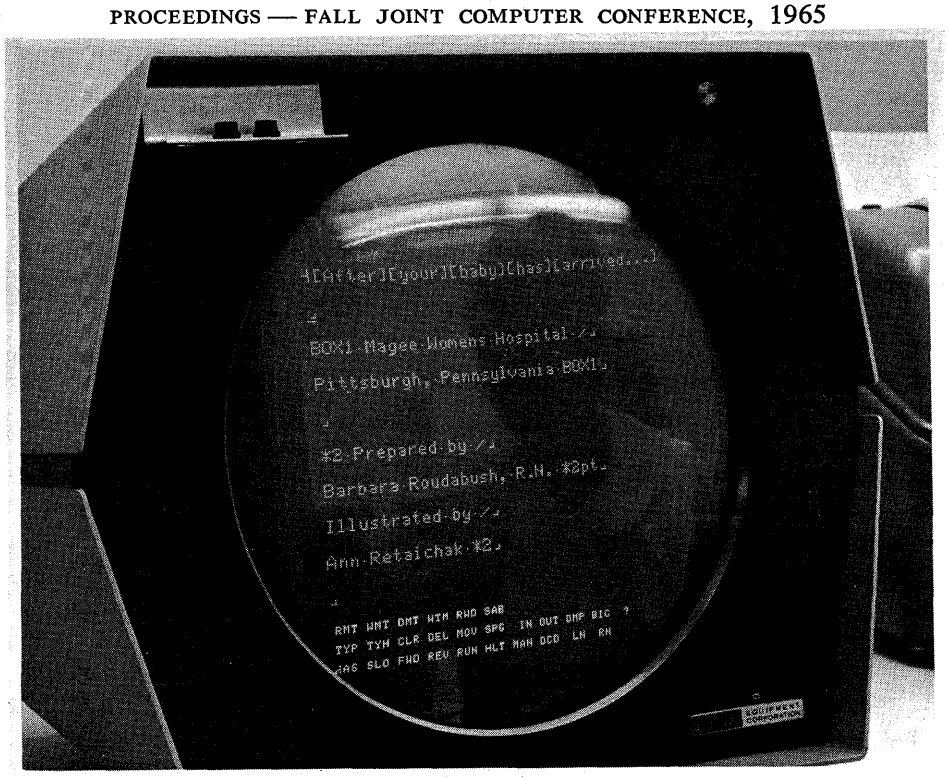



Figure 7. Text editor display.

time can be caused to move down the face of the scope or in the reverse direction with a speed controlled in increments over a wide range of values. This movement of text, its direction, and speed are controlled by the light pen and "push buttons," as are all input and output functions. A complete list of the push-button symbols and their functions is given in the Appendix.

The light pen can be used to place any one of three markers under particular characters in the display. One of these, the cursor, is used to mark a particular point in the text, while the other two, the left and right delimiters, are used to mark off sections of text for deletion or movement. The movement of delimited material to the point marked by the cursor or the insertion of material from the keyboard is controlled by the light pen and push-button symbols. In our illustration, all of the text boundary and box markers were inserted using this program. Figure 8 shows this being in our office by a secretary who has had some experience prepareing material for photocomposition. The text of the illustration as prepared on a Flexowriter is shown at the top of Fig. 9 and then with the required text boundary and box markers inserted at the bottom of this figure.

The next step in the processing of this example was to convert the text with its markers to the Rand standard format. We may suppose that this was done in order to make it available for distribution or to use it for some purpose other than typesetting. A representation of the text in this standard format is shown in Fig. 10. A proper representation would consist simply of a long string of paired octal digits, but that would not illustrate the encoding scheme very well. Here the encoded material is shown on two levels. The upper line shows that shifts and flags while the lower contains the text proper. To the left of each dattum, a text label is shown. This six-character label has the text type indicator as its first character, while the remaining five characters are specific to the entry. This label uniquely identifies the datum. In this figure, the types indicated are T for title, H for heading, A for author, B for body, and C for caption. This information could be used in the typesetting system to control the course of the typesetting process, but in this case, the information is redundant. The flags shown are represented as B for Boundary alphabet, R for Roman alphabet, and P for punctuation alphabet. There are no accepted graphics for the alphabet flags, since they are non-Hollerith (nonprinting) six-bit patterns. In the Roman alphabet, shifts are assigned the numerals 1 through 9. The


Figure 8. Editing text with scope and light pen.

```
After your baby has arrived...

Magee Womens Hospital
Pittsburgh, Pennsylvania

Prepared by
Barbara Roudabush, R.N.
Illustrated by
Ann Retaichak

Your body returns to normal...

Through a natural process called involution, organs altered by
pregnancy return to normal.

The extra tissues of the uterus and breasts that have built up
during pregnancy are absorbed by the body.

The doctor will measure the progress of this by pressing
lightly on your abdomen and saying how many "finger" widths
the top of the uterus is above or below the navel.

Positions of the uterus after delivery.
```

```
[After][your][baby][has][arrived...]*NP

BOX1 Magee Womens Hospital /
Pittsburgh, Pennsylvania BOX1

*2 Prepared by /
Barbara Roudabush, R.N. *pt
Illustrated by /
Ann Retaichak *2*NP//

B4Your body returns to normal...B4

*5 Through a natural process called involution, organs altered by
pregnancy return to normal. *5

*5 The extra tissues of the uterus and breasts that have built up
during pregnancy are absorbed by the body. *5

*5 The doctor will measure the progress of this by pressing
lightly on your abdomen and saying how many "finger" widths
the top of the uterus is above or below the navel. *5*NC

*B6 Positions of the uterus after delivery. *B6
```

Figure 9. Text before and after editing for the booklet example.

```
        BR1 9    B  R   B  R   B  R  B  R      P  B
T00001: 1  A fter2 lyour2 lbaby2 lhas2 larrived...2N

        BR1 9    1 9     1 9        BR1 9          P
H00001: 3  M agee W omens H ospital 4  P ittsburgh ,

      R1 9           B
        P ennsylvania 3

        BR1 9           BR1 9      1 9        PR1 9PR1 9P B
A00001: 5  P repared by 4  B arbara R oudabush,  R .  N . 6

      R1 9              BR1 9  1 9         B
        I llustrated by 4  A nn R etaichak 5NA

        BR1 9                            P  B
B00001: 7  Your body returns to normal...7

        BR1 9                                          P R
B00002: 8  T hrough a natural process called involution, organs

                                                     PB
        altered by pregnancy return to normal.8

        BR1 9
B00003: 8  T he extra tissues of the uterus and breasts that

        have built up during pregnancy are absorbed by the

            PB
        body.8

        BR1 9
B00004: 8  T he doctor will measure the progress of

        this by pressing lightly on your abdomen and saying how

             PR     PR
        many "finger" widths the top of the uterus is above or

                    PB
        below the navel.8C

        BR1 9                                          PB
C00001: 9  P ositions of the uterus after delivery.9
```

Figure 10. Representation of the text of the booklet example in standard format.

numeral 1 represents a shift to upper case and the numeral 9 is a shift terminator. The text boundary and box markers appear in the boundary alphabet where the character assignments are arbitrary except for two characters used to delimit sentences and paragraphs. If the parameters had been keyboarded within the text, they would appear either in the Hollerith alphabet or else as text descriptions.

In Fig. 11, the parameters for typesetting this material are shown. The first parameter, $RFORM$, tells the program that the input is in standard format. The general parameters see the page size to $8 \times 5$ inches; the type faces used will be Century Italic, Century Bold, and Century Bold Italic; the type size is set to 12 points, and tab positions are set at 1, 2, 3, 4, and 5 inches from the left edge of the page. When the standard format is being used, all text boundary and box markers are assumed to be single characters in the boundary alphabet unless otherwise indicated. In this case, an occurrence in the boundary alphabet of an N causes

```
$ RFORM, PSIZ(8,5), TFAC(CENT,CENT ITAL,CENT BOLD,CENT BOLD ITAL),
  TSIZ(12), TAB(1,2,3,4,5) $

$ AT )N(  SKIP(NP) $
$ AT )4(  SKIP(NL) $
$ AT )C(  SKIP(NC) $
$ AT )6(  SKIP(10PT) $
$ AT )A(  COL(3.75,.5,3.75) $


$ FROM )1( TO )2(  FONT(BOLD ITAL), TSIZ(24), BSKIP(.75),
  BTAB(CNT) $

$ FROM )3( TO )3(  TSIZ(8), BSKIP(CNT), BTAB(CNT) $

$ FROM )5( TO )5(  TSIZ(8), BSKIP(BTM), BTAB(RGT) $

$ FROM )7( TO )7(  FONT(BOLD ITAL) $

$ FROM )8( TO )8(  UNIT $

$ FROM )9( TO )9(  TSIZ(8), BSKIP(BTM), BTAB(CNT) $
```

Figure 11. Parameters for the booklet example.

a skip to a new page, a 4 causes a skip to a new line, a 6 causes a skip down of 10 points, an A causes a two-column format to be initiated (this occurs after the second page), and a C causes a skip to a new column. Each word on the title page is put in a separate box of the same kind. These are set in 24 point bold italics and successive boxes are skipped down ¾ inch and moved to the next tab position. On the second page, the name and address of the hospital are put in a block and centered on the page. The names of the author and illustrator are put in a box and placed in the lower right corner of the page. On page three, the subtitle and each of the three paragraphs are treated as boxes and equally spaced in the left hand column. In the right hand column, space is left for an illustration with the caption centered beneath it. Figure 12 shows these three pages in their final printed form.

With this illustration, the description of our current typesetting system is complete. The typesetting system itself, however, is not now complete, nor will it be until it is abandoned to a dusty completed projects file to rest unused. Some improvements and extensions are planned for the coming months, while others that seem promising will wait for improved hardware. One improvement, for example, will be in the ability of the system to handle tabular material, not only tables of numbers or of words, but also tables of contents and indices. Then again, there is no provision in our system for setting complex mathematical expressions. We have, in fact, no way to represent such forms in a linear string which would allow their efficient reconstruction in two dimensions on paper. This problem, however, is not of great importance to us, since the equipment we have could handle only the simplest formulas.

As our last example has shown, this system has

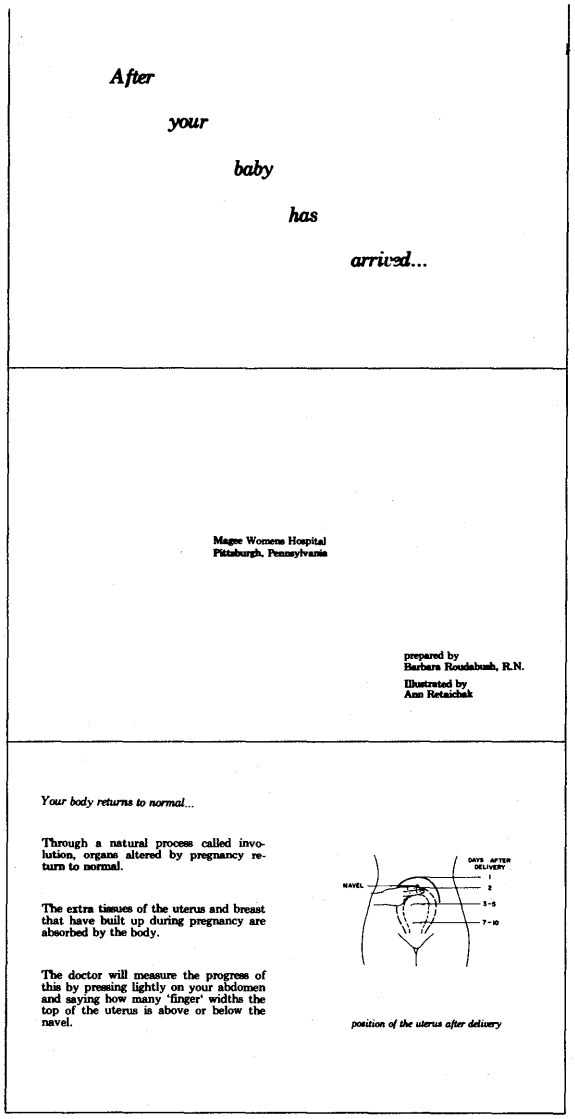



Figure 12. The first three pages of the postpartum care booklet.

some facility in controlling page format in detail. This facility is not as complete nor as easy to use as we would like it to be. We can expect some improvements in the language we use to specify formats and typesetting operations, but significant improvements will wait for new equipment. For typesetting the kind of material exemplified by our booklet illustration, no system will be entirely satisfactory that does not include a manipulable visual display. In such a system, material to be formatted would be punched simply as straight text along with the general parameters and a few text boundary parameters and associated markers, perhaps only marking page boundaries. The system would read the text and display one page at a time on a scope. The author or editor would then move this material about on the face of the scope, changing type size and font at will, until the exact format he wants is obtained. Then, with a push of a button, the page would be written out along with the appropriate codes to set it in that form. The display in such a system would not have to have high graphic arts quality, but the resolution would have to be great enough to provide exact point size and letter spacing representation for the fonts being used.

In the beginning of our discussion, we asserted that the purpose of our text processing system was to extend the abilities of the scholar in performing his work. Whatever else a scholar may do, it seems essential that he be able to: (1) make accurate observations, (2) collect, sort out, and understand the information in his field, (3) integrate his observations with current knowledge to produce new information, and finally (4) make this new information public. We have been concerned with the last of these describing in some detail one particular system intended to aid in the publication of information. The characteristics of this system are derived from our straightforward attempts to use modern computing equipment and programming techniques to duplicate as well as possible the work that is done by printers and publishers. If we are successful, the printed material we produce will be nearly as good as that we are trying to duplicate, but done much faster. If this is the extent of our own scholarly work, then surely we have been unimaginative.

Imagine a system of publication that has the following characteristics. First, a scholar publishes in this system by making his work available on magnetic tape. His publication is then "seen" by other scholars only when a computer has made the decision that his work is both pertinent to and important for some request for information. We assume that the computer's decisions in these matters is less fallable than the scholar's own. Suppose that there are many more subscribers to this system than to any current journal and that the coverage available is just as broad or as narrow as the interests of any individual scholar. Finally suppose that publication in this system is nearly immediate. If this system were in existence, then there would be no further need of scholarly publications in printed form, except perhaps for vanity.

Can the computer do all of this? There are those

of us who think it cannot. But what of the scholar? Can he continue to function for long when the information he must collect and sort out and understand expands exponentially? We may be certain that the scholar will continue to function on some level; that he will continue to generate information. The computer can aid in processing this information. That we already know. The computer alone may not be able to evaluate the importance of a document to some line of investigation, but a computer can hold statistics and the interactions between men and computers may easily generate evaluations. In our research, we are interested not so much in what the computer can do, but rather what the computer and scholar together can do better than either can do alone.

## REFERENCES

1. L. Ohringer, *"Computer Input from Printing Control Tapes,"* paper presented at the 16th meeting of the Technical Association of the Graphic Arts, Pittsburgh, June 3, 1964.

2. ———, *"Progress in Computerized Typesetting,"* paper presented at the 17th meeting of the Technical Association of the Graphic Arts, Toronto, Ont., Canada, June 1, 1965.

3. M. Kay and T. Ziehe, *"Natural Language in Computer Form,"* Memorandum RM-4390-PR, RAND Corp. (Feb. 1965).

4. M. P. Barnett, K. L. Kelley and M. J. Bailey, "Computer Generation of Photocomposing Control Tapes, Part 1. Preparation of Flexowriter Source Material," *American Documentation,* vol. 13, pp. 58-65 (1962).

5. E. J. Desautels, *"The Tabprint I System,"* Technical Note No. 33, Cooperative Computing Laboratory, Massachusetts Institute of Technology (May 1963).

6. M. P. Barnett and D. A. Luce, *"The TYPRINT System,"* Technical Note No. 34, Cooperative Computing Laboratory, Massachusetts Institute of Technology (May 1963).

7. ———, ———, and Moss, D. J., *"The RHOPRINT System,"* Technical Note No. 35, Cooperative Computing Laboratory, Massachusetts Institute of Technology.

8. ———, ———, and C. R. Morgan, *"Instructions for Operating the PC6 System,"* Technical Note No. 38, Cooperative Computing Laboratory, Massachusetts Institute of Technology, Jan. 1964.

9. C. R. T. Bacon, *"Text Editing Display,"* Technical Report, The Computation and Data Processing Center, University of Pittsburgh, 1965.

## Appendix

### THE TEXT EDITOR PUSH-BUTTON SYMBOLS

RUN Causes the text to be set into motion.

FWD Causes the motion of the text to be from bottom to top.

REV Causes the motion of the text to be from top to bottom.

FAS Accelerates the motion of the text as long as the light pen is held on this symbol.

SLO Decelerates the motion of the text as long as the light pen is held on this symbol.

HLT Halts the motion of the text.

MAN Causes the text to move when the light pen is held on it, if it is otherwise halted, and vice versa.

_C_ Cursor.

_L_ Left delimiter.

_R_ Right delimiter.

The above three symbols control the ability of the light pen to move one or another of the underlines. The symbols themselves vary, with the middle letters C, L, and R remaining constant. An initial letter D shows which of the three underlines may be moved by the light pen, and a final letter D or N tells whether or not the given underline is defined. The cursor is always defined, and hence its symbol always appears as CD or DCD.

TYP Starts typing on the Teletype the block between the left and right delimiters.

TYH Causes typing to stop immediately.

DEL Causes the block between the left and right delimiters to be deleted, operates only if the left and right delimiters are properly defined.

MOV Causes the delimited block to be moved to the point immediately to the right of the cursor.

CLR Causes the text area to be cleared.

SPG This function, the symbol pattern generator, gives rise to a different display. The entire alphabet is displayed and each symbol may be selected for change with the light pen or the Teletype. An enlarged replica of the five-by-seven dot pattern is altered by the light pen to create a new pattern. A light patch in the upper right corner returns the program to the normal text display.

? This is the interlock symbol and it has no function of its own, but serves to activate whatever other function has caused this symbol to be changed. When an interlocked function's symbol is sensed, it is made to appear in place of the question mark. When this new symbol meets the light pen, the function is initiated, and the question mark returns. The CLR function and all input/output functions are interlocked in this way.

IN Reads paper tape and appends the information to the end of the text until the text storage area is almost full, a stop code, or two successive carriage returns have been read.

OUT Causes text to be punched, in Flexowriter format, starting at the beginning of the text and continuing until the character above the cursor has been punched.

DMP Functions the same as the OUT symbol but deletes the text punched.

BIG Causes the letter patterns themselves to be punched and may be used to produce readable titles.

RMT Causes a single 120-character record to be read through the 7090 interface.

WMT Causes a single 120-character record to be written through the 7090 interface.

DMT Causes the entire text area to be written, then cleared, by repetition of the WMT function.

WTM Causes a tape mark to be written, on the output tape.

RWD Rewinds the input and output tape, 7090 logical tapes 2 and 3.

SBC Causes input and output tapes to be logically interchanged. The "SBC" symbol then becomes the "SCB" symbol.

DMR Causes the cursor to be moved to the end of the text, executes the DMP function, and then the IN function.

# MATHLAB: A PROGRAM FOR ON-LINE MACHINE ASSISTANCE IN SYMBOLIC COMPUTATIONS

C. Engelman
*Mitre Corporation*
*Bedford, Massachusetts*

## INTRODUCTION

### The purpose of MATHLAB

A mathematical scientist experiments. Today, his test tube and his breadboard are blackboard and paper. He may, it is true, have available a computer, but its role is numerical and its results are delivered not today, not tomorrow, but the day after the final programming bug is corrected. The computer is not present during the most creative phases of the scientist's labor. The purpose of MATHLAB is to provide the scientist with computational aid of a much more intimate and liberating nature.

What sort of aid? The basic goal is to provide facilities for those operations which are mechanical. Among the most common of these are the addition of expressions and equations, the substitution of sub-expressions within a larger expression, differentiation, integration, Laplace transforms, multiplication of matrices, and the solution of simple equations. While the greater part of a scientist's time is spent on these mechanical pursuits (in fact, an appreciable portion is probably spent in simply checking answers and in the eternal bookkeeping problems of getting minus signs and $2\pi$'s right), we must keep in mind that most of the tedious computations associated with the creative aspects of his work are of a symbolic, rather than a numerical, nature. If we are to free the scientist from his routine mathematical chores and conserve his energies for the more properly human activities of interpretation, analysis, planning and conjecture, then we must mechanize the passage from $r^2/r$ to r in addition to that from $1+1$ to 2.

### Requisites for a Mathematical Laboratory

I should like to outline here the properties I feel are required of a mathematical laboratory, not in terms of the range of mathematical operations available, but rather in terms of its spirit and feel.

1. It should be capable of ordinary numerical computation. This implies the ability to perform arithmetic, to compute functions or to look up their values in tables, and to draw graphs.
2. It should be capable of a wide spectrum of symbolic computations.
3. The user commands should be simple. MATHLAB is intended for a physicist, not a programmer. The commands should be no more complicated than the user's thoughts. If he wishes to enter an equation

into the computer, he should need only to type the equation in a notation like that of ordinary mathematics. If he should then wish to differentiate that equation with respect to $x$, he should have to give a command no more complicated than "differentiate $(x)$."

4. It must be expandable by the expert. The language, functions, and subroutines of the laboratory must be such that it will grow as an organism. If today we write programs for symbolic differentiation, we should expect, tomorrow, to employ them in programs for power series expansions. The opportunity to expand the programs should be open to anyone who masters a well-defined and common computer language.
5. It should be extensible by the user. While the ability of the physicist to augment the existing programs will no doubt be severely limited compared to that of the programming expert, he should be provided tools for doing certain simple things for himself, such as changing notational conventions or teaching the machine the derivatives of his favorite functions.
6. The computer, as viewed by the user, must be intimate and immediate. The user should have next to his desk a console consisting of a typewriter or, preferably, a typewriter and a scope. Economy might, in some cases, dictate the substitution of a plotter for the scope. These are connected to a large, fast, on-line, time-shared digital computer. He communicates with that computer by typing messages on his typewriter or by means of a light pen on the scope. The computer replies by means of the same machines. It types both messages and equations. On the scope it displays both equation and graphs. Above all, the response time to the user's requests must be short.

## Quick Summary of Existing Programs for Symbolic Mathematics

The first program we should mention was written for Whirlwind I by J. H. Laning and N. Zierler[1] and was not really a program for symbolic computation at all. But, even though it was capable only of numerical computation, it could accept programs written as simple symbolic mathematical expressions and perform them for a user with no machine language experience.

A later program that could accept instructions in the form of symbolic expressions, but also limited to numerical computations, was, of course, FORTRAN.[2]

Probably the most fundamental development, to date, for the adaptation of computers to symbolic computation is the design by J. McCarthy, for just such purposes, of a language called LISP (for LISt Processor).[3,4] Almost all of the symbolic computation programs extant are written in this language.

The foremost problem of content facing the constuction of a mathematical laboratory today is probably the writing of a satisfactory program for the simplification of mathematical expressions. Simplification is the unconscious of mathematics. We all simplify expressions every day, making choices appropriate to the occasion but of which we are almost totally unaware. For this reason, it is much simpler to program a more "advanced" formal computation, such as differentiation, which has exact rules of which we are conscious, than it is to have the machine simplify the answer after it completes the differentiation. We cannot afford to enter into a detailed discussion of the current attempts to solve this problem, but we would like to mention, in chronological order, the authors of three LISP programs for the simplification of mathematical expressions: T. Hart,[5] D. Wooldridge, Jr.,[6] and W. A. Martin.[7]

A LISP program for symbolic differentiation was written by K. Maling.[8] While it suffered from some weakness of simplification and from its input/output being restricted to well-formed LISP expressions (which are not nearly as legible as those written in ordinary mathematical notation), it was certainly a dramatic early demonstration of the ability of LISP to handle formal mathematical computation.

In my opinion, the most impressive example of symbolic mathematics yet performed by machine is the formal (or indefinite) integration program of J. Slagle written as a doctoral thesis under M. Minsky at MIT.[9] Perhaps the most interesting aspect of this program is that it was heuristic rather than algorithmic. It possessed a small table of integrals and

tried to reduce the problem to one or several found in that table by the same bag of tricks possessed by a good college freshman.

The remaining programs we shall discuss do not, like those above, represent attempts to perform particular symbolic processes (simplification, differentiation, integration) on a computer. They are, rather, attacks on the whole problem of building a system of symbolic computations.

The first such program we should like to mention is the mathematical laboratory project of W. A. Martin, a work-in-progress as a doctoral thesis at MIT under Professor M. Minsky. This work is by far the closest known relative to our MATHLAB. At present it appears that the main difference of emphasis will be that Martin's work will stress very broad input/output capabilities. He is, for example, working on input from scopes achieved by signifying with a light pen an interesting subexpression of a previously "printed" expression, as well as anticipating using the scopes for handwritten input. The emphasis in our program is more in the direction of continually increasing mathematical powers. Since both programs are written in LISP, a good deal of exchange should be possible.

Another approach to the mathematical laboratory problem is a project, under the direction of L. C. Clapp (while at Bolt), Beranek, and Newman.[10] The programs are not written in LISP but in a much simpler and weaker list-processing language of their own invention, called SIMPLIST. While this language and program might serve well for rapid performance of simple computations, e. g., adding two symbolic equations or evaluating a function entered symbolically, it seems unlikely that they will be capable of difficult symbolic procedures such as a powerful simplification program or symbolic integrations. These limitations stem primarily from the fact that their work is married to the PDP-1, a small computer.

The final system we should like to discuss before going on to our own MATHLAB is the IBM FORMAC system. This is an extension of FORTRAN which provides it with the ability to perform a certain amount of symbolic computation. It is under development in Cambridge under a group headed by J. E. Sammet.[11] It is quite capable, possessing such abilities as simplification, substitution, expansion of products of sums, factoring out power of a given variable, and differentiation. For a number of reasons, however, it does not seem well suited to a mathematical laboratory. It is program oriented. One does not give a simple command such as "differentiate" but rather one writes a program. True, the language, like FORTRAN, is easy for a nonprogrammer to learn, but it will not have the universal accessibility of, say, the simple commands of our MATHLAB. The programs can, of course, be run on-line. But they cannot be run, so to speak, line by line. It is necessary to write an entire program, for the program has to be compiled as an entity just like a FORTRAN program (in fact, it employs the FORTRAN IV compiler). It seems like a useful tool for symbolic computation when you know at the outset what you want to do, but quite inconvenient for experimentation.

## MATHLAB—ON THE SURFACE

Mathlab is our current attempt at realizing a mathematical laboratory of the sort we have been discussing. The program, which has been developed on the time-shared system of PROJECT MAC at MIT and on the IBM 7030 at the MITRE Corporation, is continually growing, and the following description is accurate as of April 30, 1965. We do not feel that MATHLAB has, as yet, sufficient mathematical powers to be of aid to a general user, except with respect to special and occasional problems. It does, however, possess most of the qualities postulated in the previous section as requisities for a mathematical laboratory:

1. *Numerical computations.* It is very weak in this department because we decided at first to study symbolic computation as it represented the crux of our problem. It cannot draw graphs or evaluate common transcendental functions. We can evaluate algebraic expressions with numerical arguments in a variety of ways.
2. *Symbolic computations.* Here we can perform many common tasks. We can simplify, substitute, add equations, differentiate, integrate a little, solve equations, etc.
3. *The user commands are simple.* If the user has stored an equation called *e*1 and wishes to differentiate both sides of it with respect to $x$ and call the resulting equation *e*2, he need only type:
   differentiate (el $x$ e2).

4. The program can be expanded by any LISP programmer. In fact, we are doing this all the time.
5. MATHLAB can be extended a little by the user. He can teach the machine the derivatives of functions and change the names of system commands.
6. It is intimate. The user types in some initial equations; the computer thanks him; the user requests certain symbolic manipulations; the computer performs them and types back the answer; the user types in some expressions or numbers and requests the computer to substitute these for certain variables in a previous equation; the computer types back the answer; etc.

## Mathematical Notation

In this section and the next we wish to describe MATHLAB as it appears to the user. There are some minor differences between MATHLAB as it exists at PROJECT MAC and on the STRETCH at MITRE. Where such differences occur, we shall describe the situation at MITRE. First, what sort of expressions may a user type to denote mathematical quantities? The answer is: those expressions, composed in the ordinary way, of the following entities:

1. Numbers: 1, 5/2, 2.5
2. Words, representing symbolic variables, composed of strings of letters and digits, the first of which is not a digit: x, distance, x1, x1sub2.
3. Operation symbols:
   + (addition)
   — (subtraction or minus)
   * (multiplication)
   / (division)
   ↑ (exponentiation)
4. Parentheses: "(" and ")". These have two functions. The first is to ensure the desired interpretation of certain expressions, e.g., to distinguish 5*(x+y) from 5*x+y. The second use of parentheses is for functional notation, e.g., sin(x).

All blanks are ignored. The rules of precedence of the operational symbols are conventional (FORTRAN) except that, in the absence of parentheses, e↑x↑2 denotes e↑(x↑2) not (e↑x)↑2 = e↑(2*x).

For example, if the user wishes to enter the mathematical expression (in conventional notation): $\sin(5x + y^2)$, he need only type (in MATHLAB's notation): sin(5*x + y↑2).

## The System Commands

The program as we have developed it accepts two types of symbolic quantities, called *variables* and "equations," stored in the computer and which can be referred to and manipulated by "name." A variable is a mathematical expression together with its (one word) name. An equation is really two mathematical quantities, one for the left and the other for the right side of the equation, together with a (one word) name. The initial variable and equations in an experiment are entered into the computer by a function called "denote." For example, one might type:

```
denote nil
d = 1/2*a*t↑2,
e1 = = r↑2 = x↑2 + y↑2,
,
```

The first two commas signify the end of individual definitions and the third comma tells the computer that this is all the information we choose to give it for the time being. The effect of this input is to store in the computer a variable whose name is d and whose value is 1/2*a*t↑2 and an equation whose name is e1, whose left side is r↑2, and whose right side is x↑2+y↑2. From this point on the user never again has to type in r↑2 = x↑2+y↑2 but can simply refer to e1. Incidentally, the response to the above instruction (we shall, from now on, use the convention that we speak in lower case and the computer in capitals) is:

THANKS FOR THE VARIABLE D AND THE EQUATION E1.

In terms of these basic constructs of "variable" and "equation," we shall describe the various system commands:

repeat(x)

This repeats *x* to the user. *x* may be the name of either a variable or an equation. The format for a variable is similar to that of *denote*. For an equation, if *x* were *e*1 above, then "repeat" would print:

(E1) R↑2 = X↑2+Y↑2

This same format is used when any of the succeeding commands are called upon to print an equation.

please simplify(x y)

Simplifies $x$ and names it $y$. In this, as in the following commands, the name "y" may be the same as the name "x." In this case, the old $x$ is lost. If $x$ is name of an equation, both sides are simplified independently.

forget(x)

does.

substitute((v1 v2 . . . vn) x y).

The first argument "(v1 . . . vn)" must be a list of names of *variables*. The *value* of each variable is substituted in $x$ at each occurrence of its *name*. The new equation or variable (depending on $x$) is named $y$.

At this point we should like to state more precisely the meaning of the denote and substitute instructions. Should we give the command:

```
denote nil
z = x+y,
,
```

x and z would be quantities of quite different natures. We shall refer to "x" as a "formal symbol"; it is without meaning. "z," on the other hand, is the name of an official "variable"; its meaning, which we normally refer to as its "value," is the expression "x+y" constructed of the formal symbols "x" and "y." If we now type the instruction:

```
denotes nil
x = 5*t,
,
```

a variable whose name is "x" and whose value is 5*$t$ would be created but this would in no way affect the status of $x$ in the value x + y of the variable whose name is $z$. That $x$ remains a formal symbol. The fundamental connection that can be established between these two occurrences (of different types) of the character "x" is through the instruction "substitute." If we now type:

substitute ( (x) z w)

the program will look for a variable whose name is x, find that its value is 5*t, look for a variable whose name is z, discover that its value is x + y, substitute the expression 5*t (containing the formal symbol t) for all occasions of the formal symbol x in x + y (obtaining the expression 5*t + y), simplify this to 5*t + y, and create a new variable whose name is w and whose value is 5*t + y.

Substitutions take place only upon command, never automatically. This is as it should be. The user may have previously informed the computer that x = r*cos(t), but he might like to type x without having it automatically changed to r*cos(t). Automatic substitution schemes are not only undesirable, but prone to interminable loops.

This description may seem too detailed but an understanding of the distinction between a variable and a formal symbol as well as the function of the substitute instruction is fundamental to an understanding of MATHLAB. We shall presume the extension of these concepts to equations and to other commands (e.g., differentiate) are apparent.

add((q1 q2 . . . qn)name)

The q's can be equations, variables, or numbers in any order. If there is at least one equation among them, *name* is an equation; otherwise, it is a variable. Equations are added by adding left sides and adding right sides independently. Variables or numbers are added to an equation by adding them to both sides of the equation.

multiply((q1 . . . qn name) . . .

Similar to above.

subtract(x y name) . . .

Similar to above, but only two equations, variables, or numbers are subtracted instead of an indefinite number as in add and multiply.

division(x y name) . . .

Similar to above.

raise(x y name) . . .

Similar to above. "name = $x^y$"

negative(x y) . . .

"y = –x"

invert(x y) . . .

"y = 1/x"

flip(x y) . . .

x must be the name of an equation. y becomes the name of that equation with the left and right sides interchanged. This is useful if we wish, say, to add the left side of one equation to the right side of another.

makeequation(x y) . . .

x must be the name of a variable. An equation is formed whose name is y, whose left side is the name "x," and whose right side is the value of x. For example, using the variable "d" of "denote" above, if the instruction makeequation (d e2) were given, the computer would respond:

(E2) D = 1/2*A*T↑2

makevariable(e) . . .

e must be an equation whose left side is a single word. Then a variable is formed whose name is the left side of e and whose value is the right side of e. For example, "makevariable(e2)" would now produce:

D = 1/2*A*T↑2

which is where we started.

expand((x1 x2 . . . xn)) . . .

This produces no immediate result but affects all succeeding simplifications. Whenever one of the x's (which are formal symbols) occurs in a product of sums, that product is multiplied out. The following dialogue will clarify this.

```
denote nil
e3 = = y↑3 + y*y = (x + z) * (u + y),
,
THANKS FOR THE EQUATIONS E3
pleasesimplify(e3 e4)
(E4) Y↑3+Y↑2 = (X+Z)*(U+V)
expand((x u))
YES
pleasesimplify(e3 e5)
(E5) Y↑3 + Y↑2 = X*U+X*V+Z*U+Z*V
```

The "expand" command affects not only the command "pleasesimplify" but other commands such as "substitute" or "add," which always simplify their answers.

factor((x y . . . )) . . .

Like "expand," this produces no immediate result but affects all future simplifications. It causes the collection of all terms containing, as a factor, a power of x and similarly for y. The order of formal symbols in the list "(x y . . . )" implies precedence. In this case, the term "x*y" will count as y occurrences of x rather than x occurrences of y.

We give an example:

```
denote
mitre = a*x+2*x+x*y+y
  +b*y+4*x↑2+c*x↑2,
,
THANKS FOR THE VARIABLE MITRE.
pleasesimplify(mitre bedford)
BEDFORD = A*X+2*X+X*Y+Y
  +B*Y+4*X↑2+C*X↑2
factor ((x y))
YES
pleasesimplify(mitre bedford)
BEDFORD = (2+A+Y)*X+
  (4+C)*X↑2+(1+B)*Y
```

By calling the functions expand and factor at different times with different arguments, the user can maintain control over the form of his answers.

differentiate(y x yprime) . . .

Differentiates y with respect to x and calls the resulting variable or equation (depending on y) yprime. At present, all differentiation is explicit.

learnderivative . . .

Allows the user to teach the computer the derivative of a new function. The format of this command is best explained by an example:

```
learnderivative nil
arctan,
x,
1/(1 +x↑2),
```

The need for this function has, to some extent, been obviated by an improvement in the differentiation program. If the computer is asked to differentiate arctan(x)↕2, it will decide that the answer is twice arctan(x) times the derivative of arctan(x). If it should then discover that it does not know the derivative of arctan, it will try to obtain it from the typewriter. If it succeeds, it will complete the differentiation and remember the derivative of arctan for future use.

integrate(v x w) . . .

v must be a variable which is a rational function of x with (rational) numerical coefficients.

w is its indefinite integral. For a more precise discussion, see the following section. We give an example. If

```
v = (x+1)/((x↑2+1)*(x↑2+
        x+1)↑3),
```

then the command

"integrate(v x w)"

yields the result:

```
W = (2/3*X↑3+1/2*X↑2+1/3*X—1/2)
    /(X↕4+2*X↑3+2*X↑2
    +2*X+1)+1/2*LOG
    (X↑2+1) — ARCTAN(X) —
    1/2*LOG(X↑2+X+1)+7/(3*SQRT(3))
    *ARCTAN((2*X+1)/SQRT(3))
```

solve(e x) . . .

e must be an equation which is equivalent to a rational function (with symbolic coefficients) of x. At present it can handle only those equations which are really (although not necessarily explicitly) quadratic or linear in x. The reply to this command, excepting those cases which the program cannot solve, takes one of three forms depending on whether the computer analyzes the equation to be linear, quadratic with distinct roots, or quadratic with a double root. The following three examples illustrate these different responses.

(1) If e is the equation: y = a*x+b, then "solve(e x)" yields:

```
THIS EQUATION HAS A SINGLE ROOT.
X = (Y — B)/A
```

(2) If e is the equation: a*x↑2+b*x+c = 0, then "solve(e x)" yields:

```
THIS EQUATION HAS TWO ROOTS,
  NAMED FIRSTROOT AND SECOND
  ROOT.
FIRSTROOT = 1/2*( — B+SQRT(B↑2 —
  4*A*C))/A
SECONDROOT = 1/2*( — B — SQRT(B↑2
  — 4*A*C))/A
PLEASE RENAME THOSE WHOSE PRES-
  ERVATION YOU WISH TO ENSURE.
```

(3) If e is the equation: x↑2 — 2*b*x+b↑2 = 0, then "solve(e x)" yields:

```
THIS EQUATION HAS A DOUBLE ROOT.
X = B
```

We give a fourth example which, though simple, exhibits more of the meat of the program:

(4) If e is the equation: 1/(x↑2 — 1) = 1/(x — 1), then "solve(e x)" yields:

```
THIS EQUATION HAS A SINGLE ROOT.
X = 0
```

There are two points here. First, the equation does not appear, at first glance, to be linear but the program analyzes it as such. Second, a naive attempt at solving the equation, e.g., inverting both sides, could yield what we feel is an extraneous root at x = 1. For an explanation of how the program avoids this trap, see the next section.

rename(x y) . . .

The variable or equation that had the old name x obtains the new name y.

newname(A B) . . .

This command differs from all previous commands in that it affects, not the data, but the system commands themselves. It creates a new system command B whose effect is identical to A and which exists in addition to A. For example, if the user tires of typing "differentiate," he can give the command

"newname(differentiate d)"

after which the command

"d(y x yprime)"

will have exactly the same effect as

"differentiate(y x yprime)."

Before digging beneath the surface of MATHLAB, it might help clarify some of the preceding if we give a very short sample session possible today.

```
denote nil
e1 = r↑2 = s*t,
s = x↑2 * y,
t = log(w)/x,
,
THANK YOU FOR THE VARIABLES S T
  AND THE EQUATION E1.
substitute ( (s t) e1 e2)
(E2) R↑2 = X*Y*LOG(W)
denote nil
w = sin(x↑2+y↑2),
,
```

```
THANK YOU FOR THE VARIABLE W.

substitute ((w) e2 e3)
(E3) R↑2 = X*Y*LOG(SIN(X↑2+Y↑2))
differentiate(e3 x e3prime)
(E3PRIME) 0 = Y*LOG(SIN(X↕2+Y↕2))
  + 2*X↑2*Y*COS(X↑2 + Y↑2)/SIN(X↑2
  + Y↑2)
denote nil
,
THANKS FOR NOTHING
```

This last crack on the computer's part is indicative of a fact that we have not mentioned heretofore. Most of the commands have very heavy error protection. If the user makes a mistake in constructing a variable or an equation or tries to give a variable a name already assigned to an equation, etc., he will receive an instructive error return.

## MATHLAB — BELOW THE SURFACE

### LISP

The entire program is written in LISP.[4] The system commands are all addressed to the LISP evalquote operator, e.g., the command "differentiate(h t dh)" presents the evalquote operator with a couple consisting of the function "differentiate" and the list of arguments (h t dh)."

### Numbers

The LISP system, written by R. Silver and P. Heckel[12] for the IBM 7030 (STRETCH) at MITRE, contains only one type of number, namely, rational numbers, i.e., ordered pairs of integers. If any of the numbers 12/5, 24/10, or 2.4 is typed in, it is converted to the rational 12/5.

### Internal Representation of Mathematical Expressions

The internal representation of any mathematical expression is a well-formed LISP S-Expression in a prefix notation. If the user types:

```
denote nil
v = t↑2+sin(pi*t),
,
```

then there is stored on the property list of the atom "v" the property "variable" followed by the S-Expression:

(PLUS (EXPT T 2)(SIN(TIMES PI T))).

Should the user then type:

```
differentiate(v t dv)
```

then there is stored, upon completion of the differentiation and simplification, on the property list of the atom "dv" a pointer to the S-Expression:

(PLUS(TIMES 2 T)(TIMES PI (COS(TIMES PI T)))).

This is translated back into the original infix notation and the typewriter prints:

DV = 2*T + PI*COS(PI*T)

Equations are stored similarly to variables, except there are two pointers, one to the S-Expression representing the left side of the equation and one to the right.

Besides the obvious need for well-formed LISP expressions, there are two reasons for our choice of this internal representation of mathematical expressions. First, this representation has become fairly standard and this allows us to exchange programs with other workers. Second, the prefix notation turns out to be well suited to our applications. Consider the differentiation we have just discussed. About the easiest and fastest thing for LISP to tell us about any list is what is the first item on it, in this case: "PLUS." But this is precisely the first thing our differentiation program would want to know so as to invoke the rule that the derivative of a sum is the sum of the derivatives. Both the input (infix→prefix) and the output (prefix→infix) translation programs are written in LISP, the former employing the character reading functions.

Other internal representations of expressions also occur, e.g., polynomials as lists of coefficients and rational functions as dotted pairs of lists of coefficients. All such alternative representations have translation programs connecting them in both directions with the standard prefix representation.

### Simplification and Differentiation

The internal programs for simplification and differentiation have been borrowed from the Stanford Simplify Program.[6] They have been modified in two ways.

Simplify has been enlarged to handle a family of simplifications typified by the transformation: (EXPT (MINUS X) 4/3)→(EXPT X 4/3), i.e., $(-x)^{4/3} \rightarrow x^{4/3}$. This simplification was impossible in the original Stanford system because 4/3 could only be represented by an approximating decimal such as 1.3333333 and nothing can be done with $(-x)^{1.3333333}$.

Differentiation has been modified so as to look to the typewriter for the derivatives of new functions.

Integration

The program for the integration of rational functions with numerical coefficients was written by M. Manove at MITRE in the summer of 1964.[13] It is based on a theorem of Hardy[14] that states that the integral of such a function is of the form $R_1 + \int R_2$ where $R_1$ and $R_2$ are rational and $R_2$ has only simple poles. The program always finds $R_1$ and does the best it can with $\int R_2$, that best depending on its ability to factor the denominator of $R_2$. It is good enough to have found several errors in published tables of integrals.

Solve

Solve first brings the equation it has to solve over on one side. It then combines the various terms into a single rational expression with one numerator and one denominator, employing greatest common divisor routines to eliminate common factors from numerators and denominators. The roots of the original equation are then the roots of the numerator of the constructed rational function. If that numerator is quadratic, its roots are found by the quadratic formula.

## REFERENCES

1. J. H. Laning and N. Zierler, "A Program for Translation of Mathematical Equations for Whirlwind I," Engineering Memorandum E-364, Instrumentation Laboratory, MIT (Jan. 154).

2. *Reference Manual, 709/7090 FORTRAN Programming System,* IBM, 1958.

3. J. McCarthy, "Recursive Functions of Symbolic Expressions and Their Computation by Machine," Communications of the ACM, Apr. 1960.

4. *LISP 1.5 Programmer's Manual,* MIT Computation Center and Research Laboratory of Electronics, Aug. 1962.

5. T. Hart, "SIMPLIFY, Artificial Intelligence Project," RLE and MIT Computation Center, Memo 27 (n.d.).

6. D. Wooldridge, Jr., "An Algebraic SIMPLIFY Program in LISP," Stanford Artifical Intelligence Project, Memo No. 11 (Dec. 27, 1963).

7. W. A. Martin, "Hash-Coding Functions of a Complex Variable," MIT Project MAC, Artifical Intelligence Project, Memo 70, Memorandum MAC-M-165 (June 25, 1964).

8. K. Maling, "The LISP Differentiation Demonstration Program," Artificial Intelligence Project, RLE and MIT Computation Center, Memo 10 (n.d.).

9. J. R. Slagle, "A Heuristic Program that Solves Integration Problems in Freshman Calculus, Symbolic Automatic Integrator (SAINT)", MIT Mathematics Department, Ph.D. thesis, 1961.

10. L. C. Clapp and R. Y. Kain, "A Computer Aid for Symbolic Mathematics," *Proceedings of the Fall Joint Computer Conference,* 1963.

11. J. E. Sammet and E. R. Bond, "Introduction to FORMAC," *IEEE Transactions on Electronic Computers,* Aug. 1964.

12. P. Heckel, forthcoming MITRE Technical Report.

13. M. Manove, "INTEGRATE: A Program for the Machine Computation of the Indefinite Integral of Rational Functions," MITRE Technical Memorandum, TM-04204.

14. G. H. Hardy, *The Integration of Functions of a Single Variable,* Cambridge University Press, 1958.

# AN INTEGRATED COMPUTER SYSTEM FOR ENGINEERING PROBLEM SOLVING

Daniel Roos
*Department of Civil Engineering*
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*

Computers should provide the mechanism that enables *engineers* to do *better engineering*. By permitting faster and more accurate and complete problem analysis to be performed, computers assist the engineer in his computational and *decision making* roles. The engineer today is faced with problems of increasing magnitude and complexity where the effects and interrelationships of all relevant information must be considered. *The computer provides the coordinating and integrating mechanism for the problem information needed by the engineer in the decision making process.* Computers enable engineers to perform *total problem solutions* where all pertinent information is properly considered.

This role of the computer in engineering is achieved only when the computer is adequately *integrated* in the problem-solving environment. This integration must be both *external* and *internal*. The computer must be integrated externally with the engineer using it and the programmer developing it, and internally through the proper coupling of hardware and software.

The engineer must do more than use the computer. He must actively participate in the computer solution. To do this he needs a *language* to *communicate* with the computer, physical *accessibility* to the computer, and a mechanism for obtaining *engineering-oriented results* from the computer. The communication language must be oriented to the problem rather than the machine, and must allow the engineer to easily specify his problem-solving requirements. Accessibility, provided through some type of remote computing facility such as time-sharing, permits the engineer to *interact* with the computer during the problem solution. Meaningful results are obtained using *scopes, plotters* and other output devices.

Integration of the programmer and the computer is provided through a powerful programming language and the necessary system programming capabilities to support the language. The programming language must be dynamic with respect to both problem solution and computer memory requirements. It must allow total problem solutions where the type and amount of data can vary. To satisfy these requirements the system must have *dynamic memory allocation, alternate forms of data structure,* and a *data management* and *transfer mechanism* so that the same data can be used in all aspects of the problem solution.

A modular, machine-independent system design provides the integrating mechanism for hardware and software. The system can then be implemented on any machine configuration to take full advantage of the available hardware.

ICES (Integrated Civil Engineering System) satisfies the above requirements. It is a modular computer system, designed to enable the engineer to easily communicate and interact with the computer. The programmer user the ICETRAN (*ICES*-FOR*TRAN*) programming language to develop and modify the necessary components of the system. Dynamic memory allocation, alternate data structures and data transfer and management facilities are available to the programmer. These features combine to make ICES an integrated computer system for total civil engineering problem solving. ICES is currently being developed at a cost in excess of 2 million dollars by a group of over 50 people from the Civil Engineering Systems Laboratory at M. I. T. It will provide the necessary mechanism to permit civil engineers to efficiently solve engineering problems.

What are the characteristics of these problems and how do they affect the design of an integrated computer system? Each engineering problem has certain unique characteristics that differentiate it from other similar problems. The solution to all problems however is obtained using a series of basic mathematical and engineering operations, where the *order* in which they are performed *varies* with the specific problem. These fundamental operations can be considered basic computational building blocks. A problem solution represents a certain combination of these building blocks. *An integrated computer system must therefore include a set of subroutines for these building blocks and some mechanism for specifying the sequence of operations for a given problem* (i.e., a mechanism for putting the blocks together).

There are three different ways this sequence specification can be made, namely:

1. It can be included in the program.
2. It can be specified by the engineer as part of his data.
3. A combination of the above two methods can be used where the engineer can specify some sequencing and the computer will perform the remainder.

The mode of operation used is a function of the complexity of the problem being solved and the ability of the engineer. It is highly unlikely that method 1 will be used exclusively since it requires a considerable amount of programming to insure that all possible cases that could arise in the problem solution are accounted for.

Methods 2 and 3 require the engineer to commnicate the sequence of operations to the machine. A language is therefore needed to allow the engineer to communicate with the computer. This language should be oriented toward the engineer and not toward the machine. The engineer should be able to communicate with the machine in much the same way he communicates with another engineer. He must instruct the computer rather than follow instructions that have been imposed by the machine. Man and machine must form a working partnership to effectively arrive at a problem solution.

A communication language satisfying the above requirements may be classified as a *problem-oriented language*. The vocabulary of a problem oriented language consists of a series of *commands,* where each command represents an *operation or group of operations* the computer is to perform. A command consists of a *command name* and *data* relating to the requested operation. The command name is a *technical term* that has meaning to the engineer. The data is *free format* so the engineer need not be concerned with cumbersome card column restrictions. Examples of simple commands are:

```
DISTANCE                    2    3
```

which is used to find the distance between points 2 and 3 and,

```
LOCATE/AZIMUTH  1 2 500.26 50 23 10
```

which is used to locate point 2 at a given distance (500.26) and azimuth (50 degrees, 23 minutes, 10 seconds) from a known point (1) (see Fig. 1).

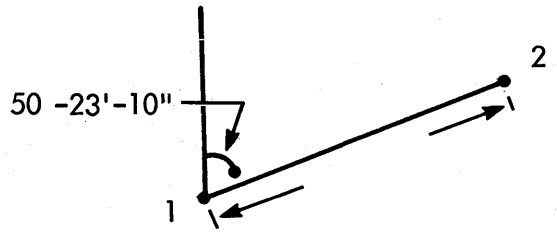



Figure 1. The COGO LOCATE/AZIMUTH command.

The two example commands are part of the COGO (CO ordinate GeOmetry) problem oriented language

used for the solution of geometric problems and developed by Professor C. L. Miller of the Department of Civil Engineering at M.I.T.

The important characteristics of the COGO language are:

1. Programs can be written by the engineer in a matter of minutes. No computer programming knowledge is required.
2. A complete problem can be solved by writing a single COGO program. Because of the small investment of time and money involved in writing a program, a new program is written for each problem and then discarded when the problem is solved.
3. The engineer must choose the necessary commands to obtain the problem solution. The quality and efficiency of a COGO program is dependent on the engineering ability of the user.
   Two engineers may write completely different COGO programs to obtain the solution to the same problem.

It is interesting to note the impact of COGO on the civil engineering profession since its introduction in 1960. It is used by many of the State Highway Departments and private consulting firms. One State Highway Department uses COGO for over 50 percent of its computer runs. The one COGO system supercedes several hundred special purpose geometric programs.

COGO allows the engineer to actively participate in the computer solution. It has introduced many engineers to the computer because it is something they can understand.

The acceptance and use of COGO and a companion problem-oriented language STRESS (STRuctural Engineering System Solver), also developed by the Department of Civil Engineering at M.I.T., are proven examples of the power of problem-oriented languages. They have suggested how a proper balance between engineer and computer can be achieved.

A problem-oriented language is necessary because it is impractical to completely pre-program an engineering problem. Part of the programming must be performed at execution time by the engineer. A *problem-oriented language enables the engineer to program the machine without actually realizing he is programming*. He can specify the characteristics of the problem, the desired method of solution, the requested sequence of operations, and the desired output (what he wants, where he wants it received, and how he wants it displayed) by using the commands of the problem-oriented language. Although the commands appear as a program to the computer, they appear to the engineer as a logical problem statement in engineering terminology.

FORTRAN and other compiler languages have sometimes been referred to as problem-oriented languages. However, FORTRAN contains considerable computer terminology (COMMON, GO TO, DIMENSION, etc.) and is more procedure-oriented than problem-oriented. It, therefore, appears more realistic to reserve the term problem-oriented language to a language with the following attributes:

1. It is oriented to the user rather than the programmer.
2. No computer programming knowledge is required to effectively use the language.
3. The language is command structured where the commands are composed of technical terms.

In certain cases the engineer can use a problem-oriented language to write all the commands for a problem solution, and then submit his run for processing. It is not always possible or desirable, however, for the engineer to function in this manner. Quite often he will formulate part of the solution and then base the remaining portion on the results of the first part. This mode of operation is incompatible with typical batch processing operations. Instead, the engineer must be able to communicate with the computer in an interactive environment where he can:

1. Request an operation.
2. Examine his results.
3. Determine the next operation to be performed based on the previous results.

This suggests that a problem-oriented language functions best in a time-sharing environment. With time-sharing the engineer can try many alternate designs and compress into one session the same work that would require many individual sessions over a long period of time in a batch processing mode.

Engineering is typically performed under severe time restraints. The turn-around time problem in-

herent in batch processing can totally negate the otherwise beneficial results of computers. The engineer needs immediate access to the computer. Time-sharing or some other form of remote computing offers him this access. The combination of the accessibility of remote computing and the communication capability of problem-oriented languages provides the engineer with the necessary tools for effective man-computer communication and interaction.

COGO and STRESS represent a significant step forward in effective computer utilization. However, both of these systems are limited to problems or portions of problems of a particular discipline. Civil engineering problems generally involve the consideration of many disciplines.

Even in a relatively small problem such as the design of a highway interchange, the engineer must consider the highway location and design (highway engineering), settlement, stability and foundation conditions (soil engineering), highway bridges (structural engineering), drainage (hydraulic engineering) and traffic flow (transportation engineering). As engineers we recognize each of these separate disciplines and the necessary interactions that must exist for effective problem solving. *In an integrated computer system each of these disciplines must be present as a subsystem and the subsystems must be able to interact with one another.* The engineer must be able to procede in his problem solution by using the necessary subsystems at the proper time. At any point in his problem solution, he can leave one subsystem, enter another to perform calculations, and then reenter the original subsystem using the results just obtained.

In the past the complexity of engineering problems and the large amount of data often forced the engineer to unnaturally decompose his problem into non-interactive tasks. Many of the feedback aspects of the problem had to be overlooked. The computer now offers the mechanism to permit total problem solutions where all relevant factors and interactions are considered.

One *data file* residing on secondary storage can be associated with the total problem solution. Certain portions of that file can be used by each of the subsystems. A *data management* facility must supervise the organization of the entire data file and a data transfer mechanism must transfer appropriate data between the sybsystems.

What about this data associated with the subsystems? It is difficult to classify engineering data since it is of many types and varieties. In some cases only the numeric value of the data is important, where in other cases, hierarchical and structural relationships between the data are also important. Data with different external characteristics requires different internal computer representation. Some engineering data is best stored as arrays, other as lists, and other as combinations of lists and arrays. In the past, computer systems have generally been limited to one predominant form of storage (either list or array). An integrated engineering system must have alternate forms of data structure; namely, arrays, lists and array-lists. The programmer will then choose the proper internal structure for his data based on considerations such as space requirements, access time, information content, manipulation ability and system overhead. The alternate forms of data structure will allow new representations of engineering problems to be achieved on the computer.

Most engineering data is best represented in the computer in array form. To achieve optimum capability and remove the restrictions presently associated with normal FORTRAN DIMENSIONed array storage, arrays should be dynamically allocated. Dynamic allocation of data achieves the following:

1. Arrays are allocated space at *execution* time rather than at *compilation* time. They are only allocated the amount of space needed for the problem being solved. The size of the array (i.e., the amount of space used) may be changed at any time during program execution. If an array is not used during the execution of a particular problem, then no space will be allocated.
2. Arrays are *automatically* shifted between primary and secondary storage to optimize the use of primary memory.

Dynamic memory allocation is a necessary requirement for an engineering computer system capable of solving different problems with different data size requirements. *A dynamic command structured language requires a dynamic internal data structure.* The result of dynamic memory allocation is that the size of a problem that can be solved is virtually unlimited since secondary storage becomes a logical extension of primary storage.

Dynamic memory allocation would extend to *programs* as well as *data*. Programs can then be brought into primary memory only when they are needed. The allocation of programs and data must be properly balanced so that the use of primary memory is optimized.

The memory allocation problem has been considerably simplified as a result of newly announced third generation computer hardware, in particular new random access storage devices. The new machines are faster, more powerful, larger and cheaper than their second generation counterparts. The large primary and secondary storage available at reduced prices should increase the size of problems that can be solved on the computer and decrease the cost of the problem solution.

Briefly summarizing, then, an integrated engineering computer system should have the following computer oriented characteristics:

1. A flexible powerful problem-oriented language for the engineer, to communicate with the computer.
2. Remote computing to make the computer accessible to the engineer.
3. An orientation toward total problem solving, not just computation.
4. An interaction between discipline areas—or subsystems—for solving a problem which encompasses more than one engineering discipline.
5. A data management scheme to organize the data and a data transfer mechanism to pass the data between subsystems.
6. A modular internal building-block structure.
7. An efficient internal data structure where alternate forms of data can be represented.
8. Dynamic allocation of data and programs based on the problem being solved.
9. Computer hardware to accommodate the necessary software.

All these features have been incorporated into a computer system for civil engineering called ICES (Integrated Civil Engineering System). The engineer uses ICES to obtain solutions to the problems he faces. The internal capabilities of ICES provide the mechanism for efficient and powerful problem solutions.

To function correctly, these internal computer capabilities must be properly linked together. ICES, therefore, includes a programming language ICETRAN to create the engineering subsystems, and an *operating system* to coordinate and supervise ICES computer runs, both of which incorporate the necessary internal computer capabilities.

ICETRAN and the ICES operating system programs are unique because they are developed by people who are expert in both the information sciences and civil engineering. In the past a schism has existed between the computer language developers and application users. The language developers could not completely appreciate the needs of the users, and their languages therefore were not well suited for the intended users. ICES, however, is developed for civil engineers by engineers.

Every computer system needs a basic programming language. With respect to engineering, the FORTRAN language contains many desirable features, and could serve as the basis for the ICES programming language. However, additional capabilities are needed. For this reason, FORTRAN has been extended into a language called ICETRAN which will be used to program the ICES subsystems. ICETRAN contains all of the FORTRAN statements plus additional capabilities to facilitate the problem-solving features of ICES. These additional capabilities are imbedded in the normal FORTRAN structure. No new programming restrictions are imposed on the programmer.

One of the additional capabilities contained in ICETRAN is dynamic memory allocation. A typical ICETRAN program illustrating the use of dynamic memory allocation is shown below:

```
      SUBROUTINE ADD (I)
      COMMON A, B, C, . . . .
      DYNAMIC ARRAYS A, B, C
      DEFINE C, I, HIGH
      DO I L=1, I
    1 C (L)=A(L) + B(L)
      DESTROY A
      RELEASE B
      RETURN
      END
```

This subroutine adds two one-dimensional dynamic arrays (A and B) together to form a new dynamic array (C). To understand the ICETRAN dynamic memory allocation statements it is first necessary to briefly summarize the ICES dynamic memory allocation scheme.

Associated with each dynamic array is one COMMON location known as a codeword-pointer. This location contains the following information about the dynamic array:

1. The size of the array.
2. The residence of the array (primary storage, secondary storage or space not yet allocated).
3. A pointer to the beginning location of the array in the *data pool*. All dynamic arrays are allocated space in a data pool, which consists of the *unused primary memory space* at program execution time.
4. The status and the priority of the array, to be used for *memory reorganization*. Dynamic memory allocation implies that the array space requirements are constantly changing. If the data pool becomes full and more space is needed, then a memory reorganization must be performed. This reorganization is based on the status (active, released, or destroyed) and the assigned priority (high or low) of the array. A sufficient number of arrays are transferred to secondary storage to make room for new active arrays. If an array on secondary storage is later referenced and therefore needed it will automatically be brought back into primary memory.

Returning now to our sample program, the DYNAMIC ARRAY statement is used to specify all dynamic arrays. The statement does not cause any instructions to be generated by the compiler. The DEFINE statement is used by the programmer to specify information about a dynamic array that will be stored in the codeword-pointer. The DEFINE statement in the above example causes the size (I) and priority (HIGH) of dynamic array C to be inserted in the codeword-pointer of array C. Dynamic arrays A and B have already been DEFINED in the subprogram that called subroutine ADD.

The DEFINE statement does not cause allocation of space for the array. The *allocation* of space is delayed until the *first reference* to an array element is made (statement 1 in the above example). The first time statement 1 is referenced "I+1" contiguous unused locations in the data pool will be located. If they are unavailable a memory reorganization will occur. The pointer of the codeword will then be set to a point to the first of the I+1 locations, which will contain a *backpointer* to the codeword. The remaining I locations will be used to store the array. If the array is shifted in the data pool during memory reorganization the pointer of the codeword is appropriately adjusted.

After the DOP loop is completed, array A is destroyed (DESTROY A) and array B is temporarily released (RELEASE B). An array should be destroyed if it is no longer needed and released if it is not presently needed. Intelligent use by the programmer of the DESTROY and RELEASE statements decreases the likelihood of memory reorganization and increases the efficiency of one when it does occur.

The dynamic memory allocation scheme is quite powerful and can handle arrays of any number of dimensions. *An n dimensional array is treated internally as a partitioned set of subarrays.* This offers extreme flexibility since

1. Only the subarray the engineer is working on need be in core.
2. The *size* of each subarray can differ. Figure 2 shows a two dimensional array where the size of each subarray (column) varies (2, 4 and 3).

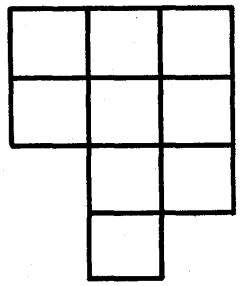


Figure 2. Array size variation.

3. The *structure* of the subarray can vary so that tree structures can be represented using array notation. A tree structure with the associated subscript notation is shown in Fig. 3.

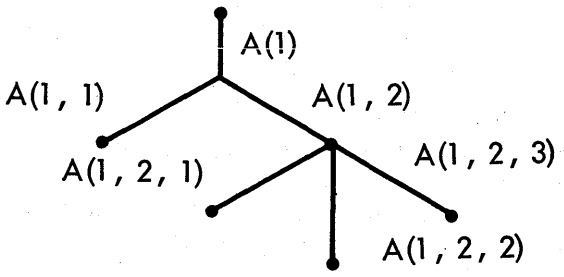



Figure 3. Array structure variations.

The programmer can easily set up desired structures using the DEFINE command and then operate

on the structure using normal FORTRAN array notation.

Dynamic memory allocation capability is merely one of many features available with ICETRAN. Others include list processing, data management, subsystem data transfer, and matrix manipulation.

A completed ICETRAN program must first be processed by the *ICES precompiler,* which will translate the ICETRAN statements into legitimate FORTRAN statements. Although the generated FORTRAN statements might appear somewhat confusing to a FORTRAN programmer, they will accomplish the requested operations.

The resulting FORTRAN program is then processed by the conventorial FORTRAN compiler. The use of the precompiler eliminates the necessity of modifying the FORTRAN compiler or developing a totally new programming language.

The ICES precompiler is but one of the system programs comprising the ICES operating system. Other are described separately below.

ICES Executive

The principal form of input that engineers will use with ICES is problem-oriented language commands. Each ICES subsystem will have a command vocabulary associated with it. The ICES executive processes the problem-oriented language commands that the engineer issues by performing the following operations.

- Read and encode command.
- Analyze, convert and store data.
- Perform consistency checks.
- Transfer control to routine for command execution.

The running of a submitted program therefore essentially consists of two phases, the analysis of the command by the ICES executive, and the execution of the command by the appropriate processing subprograms. A command is completely processed before the next command is read. In this regard the system operates somewhat as an interpreter, except that the commands do not cause computer instructions to be generated. Instead programs that have previously been translated to machine language instructions are used.

The executive has been designed to allow many powerful features to be included in the ICES problem-oriented command input language that were not incorporated in the previously discussed COGO problem-oriented language. It may be recalled that with COGO only numeric data given in a specified order was permitted. The new expanded problem-oriented language capabilities and executive features of ICES include:

1. The data associated with a command may be identified by labels and specified in any order. If the engineer prefers, he may omit the labels and enter the data in a standard order (as in COGO). An example of a command with labeled data items is

   STORE POINT 10 *X* 50 Y 750 . 63

   which cause the X and Y coordinates of a point to be stored.
2. The engineer may omit a command data item and a standard value will automatically be used by the executive. This value may be either:
   (a) *permanently present* by the programmer when the command is initially set up;
   (b) *temporarily present* for a problem by the engineer at the beginning of the problem; or
   (c) may be the value of the data item *presently stored* in the computer. The programmer who initially sets up the command indicates which of these options should be followed. If a standard value is not associated with the data item, the executive will indicate an error condition whenever the data item is omitted by the engineer. The use of standard values reduces the amount of unnecessary data an engineer must include with a command, since input entries are made only when a nonstandard value is encountered.
3. Data values may be alphanumeric as well as numeric. For example, in a geometric problem the quadrant of an angle may be identified as NE, SE, SW, or NW. The executive will automatically set a switch based on the alphanumeric value given by the user. This switch can then be interrogated by the command processing routines.
4. Incremental as well as total processing of the input command is permitted. If so instructed, the executive will process part of the data, transfer control to execute that portion of the data, and then return to repeat

the cycle, continuing until the data is exhausted. Incremental processing minimizes storage requirements when the data field contains a variable number of data items.

5. The executive can initialize variables and incremental counters that are specified by the programmer at command definition time. These variables and counters can then be used by the command processing routines.
6. The command data field can contain *command name modifiers* which permit complicated *tree structured commands* to be set up by the programmer.

To illustrate command name modifiers consider the three versions of the INTERSECT command shown below. The command data consists of the POINT number assigned to the intersection point and the two geometric objects (ARC, LINE) being intersected. In the last two commands a known point NEAR the desired intersection point must be specified since a line and an arc or two arcs can intersect at two different points:

```
INTERSECT  POINT 5  LINE 10  LINE 20
INTERSECT  POINT 5  LINE 10  ARC 20    NEAR 3
INTERSECT  POINT 5  ARC 10   ARC 20    NEAR 3
```

The engineer thinks of these as the same command (INTERSECT) but the programmer must think in terms of three different commands, since each has a different type and amount of data associated with it and each requires different subroutines for execution. The programmer views the command as the tree structure shown in Fig. 4, where the labels ARC and LINE serve as command name modifiers. These modifiers determine the appropriate branch of the tree and the data and subroutine requirements associated with that branch.

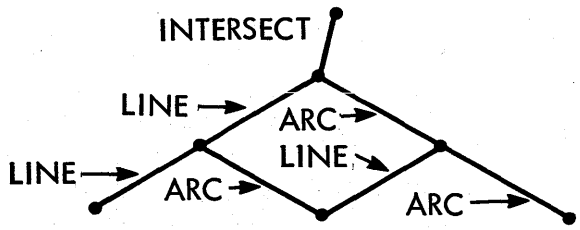



Figure 4. The use of command name modifiers.

Command name modifiers minimize the number of necessary commands and increase the capabilities of the commands.

With ICES the structure of the problem oriented language input has been generalized so that the same executive can be used for all ICES subsystems. The command vocabulary of each subsystem differs but the command structure is identical. Each subsystem has associated with it a set of internally stored tables which define the commands to the executive. When the executive processes the engineer's commands, it uses the appropriate set of command tables for the sybsystem the engineer is working with.

A programmer uses a *command definition language* to set up the subsystem commands and generate these command tables. This command definition language is a problem-oriented language designed for the subsystem developer. *ICES therefore includes a problem-oriented language to generate problem-oriented languages.* In addition to generating new problem-oriented languages, the command definition language can be used to easily add, modify or delete commands from a subsystem that already exists. This ability to easily modify a subsystem is a necessary requirement in engineering organizations where both the problems and the organization itself can change.

The simple example below illustrates some of the features of the ICES executive and the command definition language. The STORE command used to define the X and Y coordinates of a known point will be added to the COGO subsystem of ICES. A typical STORE command as entered by the engineer could appear:

```
STORE POINT 10 X 1000.53 Y 960
```

The command definition program below adds the new command to the COGO vocabulary. The underlined words are the vocabulary of the command definition language and the information in quotes is the input data that the programmer supplies.

```
SYSTEM 'COGO'
ADD 'STORE'
ID 'P' INTEGER 'N POINT' REQUIRED
ID 'X' REAL 'XCOORD' STANDARD 0
ID 'Y' REAL 'YCOORD' STANDARD 0
EXECUTE 'STORE'
FILE
```

SYSTEM specifies the ICES subsystem (COGO) being modified, and ADD specifies the type of modification (addition of a command) and the name of the command (STORE). ID gives the characteristics of the data items associated with the command. One ID entry is required for each of the three data items of the STORE command. The information included as part of the ID entry includes:

1. The identifier for the data item. The identifier defines permissable labels that the engineer can use to label the data items. A permissible label is one that begins with the specified identifier. For example the identifier of P permits the engineer to use P, PT, POINT etc. as labels for the point number. The identifiers for the X and Y coordinates given in the second and third ID commands are X and Y.
2. The internal machine representation of the data item (REAL or INTEGER).
3. The computer location where the data item will be stored (NPOINT, XCOORD, YCOORD in the above example).
4. The action to be followed if the data item is omitted by the engineer. A REQUIRED data item must be entered by the engineer or an error will be indicated by the executive. The STANDARD entry is used to specify a preset data value that will be used by the executive if the data item is omitted by the engineer. Since the coordinates of a point are quite often (0, 0), they have been preset to these values in the above example. If the action field of the ID command is left blank (this does not occur in the above example), then the data item will retain its current value if omitted from the command by the engineer.

EXECUTE specifies the subroutine to be entered (STORE) to execute the command, and FILE concludes the command definition. If no errors have been detected by the command definition program, the necessary tables for the newly defined command will be generated and the command will be added to the COGO subsystem of ICES. The above system modification can be run as a normal ICES job.

This example is for a trivial command and does not demonstrate many of the capabilities of the executive and the command definition language. Unfortunately, space does not permit more interesting commands to be discussed. The reader is referred to reference 13 which contains a complete description of the executive capabilities and the command definition language. Many examples of different types of command structures are included in that paper.

## Dynamic Memory Allocation

A series of programs has been developed to facilitate the dynamic allocation of data and programs. With regard to data, these programs define, allocate, destroy and release arrays by operating on the codeword-pointers and data pool. They retrieve and store referenced array elements, transfer arrays between primary and secondary storage and reorganize the data pool whenever necessary.

Dynamic allocation of programs is accomplished by system programs which generate program segment modules from compiled subprograms, and then load and relocate these modules as they are needed during command execution.

## Data Management

Data management system programs control the organization of the data files which reside on secondary storage and are associated with the engineer's problem solution. The data management programs allow the *same data file* to be used by *all the subsystems of ICES,* and each subsystem to use *different names* to refer to the *same data.* Variable and array names in one subsystem are *mapped* into different variable and array names in the other subsystems. Whenever an engineer switches from one subsystem to a different subsystem, *data transfer programs* use the mapping function to automatically rearrange the data to reflect the new subsystem.

Data management also allows the engineer to easily operate on data files. He can print, modify or delete any of his files. The data files can serve as *permanent documents* of completed problems. Space considerations will of course influence how much information the engineer should keep on secondary storage and for how long this information can be retained.

The data management facility will permit public files to be created so that several engineers can have access to the data to work on the same problem. Suitable protection features will be provided to ensure that unauthorized users may not examine another person's files.

## System Generation and Modification

Necessary utility programs have been developed to set up the ICES system on a computer configuration and then make modifications to the system.

The modifications are performed as normal jobs under ICES so that the system is self-modifying. ICES is *modular* so that each organization can adapt the system to their problem solving requirements and hardware capabilities.

ICES has been designed to be manufacturer- and machine-independent. The original system has been developed for the IBM System 360. Minimum hardware requirements for ICES include:

65 k bytes core storage (any model of the System 360 may be used);
two 2311 disk drives (three drives will result in a superior system);
card input/output facilities.

The usefulness of ICES is substantially improved when on-line plotting devices, graphical input/output devices and remote computing are available. The ICES system being implemented on the model 40 System 360 of the Civil Engineering Systems Laboratory at M.I.T. features all of these capabilities. A substantial portion of the civil engineering community, however, cannot afford these added capabilities, and others who can afford them do not feel they are economically justified. As a result none of these capabilities are required in a minimum ICES system. The inclusion of any or all of the features will result in a far superior ICES system.

## ICES Supervisor

The ICES supervisor coordinates all of the ICES system programs and supervises the use of the computer. It calls on the proper system program or user program based on the requested operations.

Some people will no doubt express concern over the time overhead of the above system programming capabilities. A system objective should be the *minimization of time* required for problem solution. However, this must not be done at the expense of finding the optimal or best solution to a problem. Quite often an increase of several microseconds is completely justified by the benefits obtained from the operations performed. For example, dynamic memory allocation does involve additional machine cycles, but it also eliminates many bookkeeping requirements and the packing of information that are necessary without its use. It also allows increased problem-solving capabilities.

One principle of ICES is that no system programming capabilities are forced upon the programmer. He is, for example, given the choice between normal dimensioned arrays and dynamic arrays, between the input capability of the executive or normal FORTRAN READ statements. The programmer must consider the problem being solved and the tradeoffs associated with each of the available alternatives, and then make his choice.

Let us now reexamine the ICES system in total. One approach is shown in Fig. 5. Each of the vertical boxes represents a subsystem of ICES. These subsystems utilize the basic engineering building-block routines represented by the horizontal box at the bottom, and the system programming capabilities of the ICES operating system (top horizontal box). This framework enables subsystem programmers with no system programming capabilities to easily develop high-performance subsystems using the ICETRAN language.

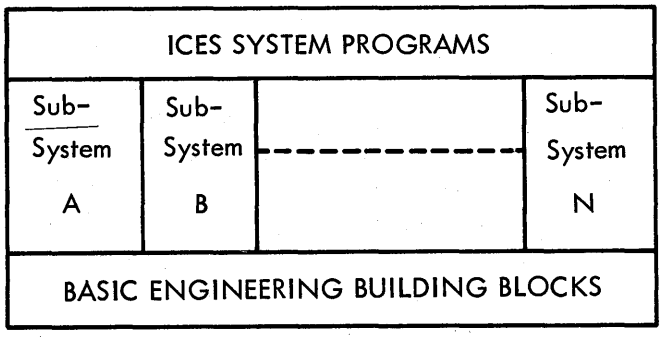



Figure 5. The ICES system.

The Civil Engineering Systems Laboratory at M.I.T. is currently developing a comprehensive series of ICES subsystems. (See, for example, references 4 and 5 and 8 for a description of subsystems being implemented for ICES). ICES subsystem development should not, however, be restricted to M.I.T. Instead, ICES should be considered as a framework for civil engineering computer work, where all members of the profession make contributions. It is the integrating mechanism for the programs that have been developed in the past and the work that will be performed in the future.

User groups such as SHARE have demonstrated how needless duplication of programming effort can be reduced. The ICES concept is a natural extension of this idea. The user organizations provided a framework for rational development and distribution of the necessary component programs. ICES provides a framework for uniting these components in an inte-

grated system. We can now, therefore, think of *system coordination* as well as *program coordination.*

What then is ICES? ICES is many partnerships. A partnership between the third generation hardware and the ICES programming software, a partnership between ICETRAN and programmers, a partnership between the problem-oriented language command structure and the engineer, and a partnership between the integrated computer system and the engineering organization. *It is an integrated system for civil engineering where each of the necessary components (hardware, software, engineer, manager, and programmer)*, properly contribute and interact.

## ACKNOWLEDGMENTS

A project such as ICES is the result of careful planning and represents a natural evolution of previous work. For the past ten years Professor C. L. Miller as director of the Civil Engineering Systems Laboratory at M.I.T. has demonstrated how computers can effectively be used in engineering practice and education. This past work serves as the foundation for ICES, which was conceived by Professor Miller and is now being developed by individuals who were trained and inspired by him. These individuals include Robert Logcher, Jay Walton, Alan Hershdorfer, Alden Foster, Ron Walter, and Richard Goodman. The work reported in this paper was formulated by these people. The author is also indebted to Miss Betsy Schumacker of IBM for her advice and contributions.

## REFERENCES

1. J. A. Champy, "Man Machine Computer Systems in a Public Works Agency: A Management View," *Industrial Management Review,* Spring 1965.

2. S. J. Fenves et al, *Stress: A Users Manual,* M.I.T. Press, 1964.

3. ———, R. D. Logcher and S. P. Mauch, *Stress: A Reference Manual,* M.I.T. Press, 1965.

4. ———, "The Form and Use of an Interactive Problem Oriented Language," this volume.

5. ———, et al, "A Users Manual for the On-Line Use of the Structural Design Language," Internal Project MAC Memo M-234, M.I.T.

6. C. L. Miller, "Man-Machine Communications in Civil Engineering," Department of Civil Engineering, T63-3, M.I.T. (June 1963).

7. ——— and R. Walter, "Communicating with Computers in Civil Engineering Design," Department of Civil Engineering, T65-4, M.I.T. (Mar. 1965).

8. P. O. Roberts and J. H. Suhrbier, "Highway Location Analysis, An Example Problem," Department of Civil Engineering, R62-1, M.I.T. (Mar. 1962).

9. D. Roos and B. Schumacker, "ICES: Integrated Civil Engineering System," American Association of State Highway Officials Conference on Improved Highway Engineering Productivity, Boston (May 1965).

10. ——— and C. L. Miller, "COGO-90: Engineering Users Manual," Department of Civil Engineering, R64-12, M.I.T. (Apr. 1964).

11. ——— and ———, "The Internal Structure of COGO-90," Department of Civil Engineering, M.I.T. (Feb. 1964).

12. ——— and ———, "COGO-90 Time-Sharing Versions," Department of Civil Engineering, M.I.T. (May 1964).

13. R. A. Walter, "A System for the Generation of Problem Oriented Languages," this volume.

# AESOP: A PROTOTYPE FOR ON-LINE USER CONTROL OF ORGANIZATIONAL DATA STORAGE, RETRIEVAL AND PROCESSING

Edward Bennett, Edward C. Haines, and John K. Summers
*The MITRE Corporation*
*Bedford, Mass.*

## OVERVIEW

AESOP is an experimental on-line information control system realized in the Systems Design Laboratory of The MITRE Corporation. It serves as a prototype for a class of management or command information systems capable of giving the members of the using organization as much on-line control over system performance as possible. It is a CRT display-oriented system in that the user experiences the information system primarily through his CRT displays and exercises his control through his light pencil. This control is not limited to that level of the organization responsible for programming the system, but applies upward to the highest level of executive personnel interested in obtaining direct access to the system.

The current version of the AESOP prototype operates on an IBM 7030 (Stretch) computer (65K memory with 64-bit words) with a 353 disk storage unit holding two million words.[1] Each of the four user stations consists of an on-line Data-Display-13 display console with a photoelectric light pencil, an on-line typewriter, and a Stromberg-Carlson 3070 medium-speed printer.

The AESOP system is designed to take advantage of the range of capabilities implied by this central processor and the user station equipment. However, the design philosophy is tailored to the general characteristics of management and command users rather than to specific characteristics of either the equipment or a specific organizational job.

The following description of the prototype, and of the conceptual foundations underlying the design, deals mainly with those aspects of the system which emphasize the user's on-line control capability. The description is divided into two sections, the first dealing with the long-range conceptual objectives which influence the design of the AESOP prototype and the second dealing with the system characteristics as they are experienced by the user of the currently operational A/1 Model.

## CONCEPTUAL OBJECTIVES

The flexibility of an on-line information processing system often determines not only its ability to respond to new and changing demands upon it,

*Software characteristics appear in the Appendix. The program is configured to permit continuous modification of system performance, both on-line and off-line, with minimum recoding requirements. As a result, the models of the system change regularly and the model operating at the time of the FJCC meeting may contain capabilities other than those realized and operating at the time this report was prepared.

but may, in fact, also determine how extensively the system will be used outside of the limited fraternity of professional scientists and engineers involved in its development and management. System flexibility is particularly critical when the on-line capability in question is intended to be more than a multi-access, remote station computer sharing facility. When operating in an organizational context, the users of an on-line system need to do more than develop, store and execute their own job-oriented programs. They must also share the common data base, common data retrieval and updating routines, and the various on-line processing procedures designed to use this common data base for various organizational purposes.

Further, in the case of an organizational system, flexibility involves not only the system's ability to be changed off-line to meet changing on-line requirements, but also its ability to be changed on-line to meet the constantly changing needs and time constraints of its various users. In addition, an organization-wide system must interface with users of widely varying background, experience, needs, and position. A system tailored only to the needs of junior users is no better than one tailored only to the needs of the executive suite.

The fixed information-processing procedures usually employed at the lower levels of an organization, for financial accounting, billing, payroll, and so forth, may leave little room for creativity or imagination on the part of on-line users of the system. In most cases, in fact, there is little need for such creativity. On the other hand, the hardest task masters, with the greatest demand for on-line control, are the military or industrial planners, ranging in organizational level and function, but working collectively on the same total data base. Especially at the higher levels of an organization, where the organization's data base is used for global planning and decision, there is a need for creative approaches to information processing. The senior executive who might use an on-line information system for situation monitoring, resource allocation planning, or for insight into the timing of executive decisions, benefits from as much freedom to mold and configure the performance of his information system as is possible within the limits of his job.

The concept of on-line information control implies the ability of such users of the system to change the performance of the system to meet their own changing needs or wishes. With adequate control, they can experiment with the display of alternative data formats or configurations, with alternative sequences of data retrieval, with alternative formulae for summarizing, processing, or analyzing data.

The on-line user needs to be able to express his desires by communicating with the system is an easily usable language. The language itself must have a syntax and vocabulary tailored to increase the executive user's interest in the system, his creativity in using it, and his flexibility and facility in accomplishing any changes he wishes to make. This implies that the language should have a syntax and vocabulary complex enough to permit a flexible range of alternatives. However, the system should make the on-line use of that language little more than a matter of the on-line user sensing, choosing and pointing out his communication desires. Keeping track of well-formed syntax and vocabulary constraints and implications need not be the on-line user's problem.

An advantage of on-line control for information storage is in the generation and editing of personal files. Such personal files, from the user's point of view, are not files as they exist in fact within the machine structure. The user's concept of his file is more likely conditioned by his past experience with manual or mechanical files; and his impression of the files within a computer system can be, therefore, mainly a matter of how he experiences these files at his own input-output station, how he generates or edits his file, and how he retrieves data from such files. For an on-line user, this experience can be influenced by the way in which data are reported to him by the system, through his own display device, in his own work environment. He can come to "understand" a file as simply the set of possible reports derived from the file, the lists, tables, and matrices, as they appear before him.

This implies that, no matter what logic is applied to the management of the data base inside the machine, the surface appearance of the data base must be under the user's on-line control. He should be able to set up the file display, the object and property names, the formats of these files, including sequence and arrangement of data, as he desires. He should be able to generate, store and revise the structure of his displays, on-line, in much the same fashion as he generates, revises and stores any

other portion of his personal data base, and using the same general on-line control language.

In addition to the user's need for on-line control over the generation, retrieval, and display characteristics of his data base, he needs control over the generation, revision, naming and execution of his job-oriented processing routines, as well as the storage and retrieval of such processing routines as part of his data base. He needs to be able to manipulate logical and arithmetic operators, planning factors and parameters, models and projections of relevant job-oriented operations. The system executive program must carry the burden of making his job-oriented generation of algorithms little more than a matter of looking at his display, choosing among acceptable alternatives and pointing to express his desires.

The need for on-line control is not necessarily limited to any level of organization. On the other hand, senior executives should not have to use this control to create data of low aggregation or detailed procedures normally the responsibility of system analysts. However, if an executive so desires, he should be able to use his control to access the system at the lower organizational levels when he wants to.

## THE AESOP (MODEL A/1) PROTOTYPE

The AESOP A/1 prototype in the form described following has been operational since August 1965. From a conceptual standpoint the system contains two major features. The first is an information retrieval capability which can be operated entirely by means of a CRT display and a light pencil as shown in Fig. 1. The second is a facility to

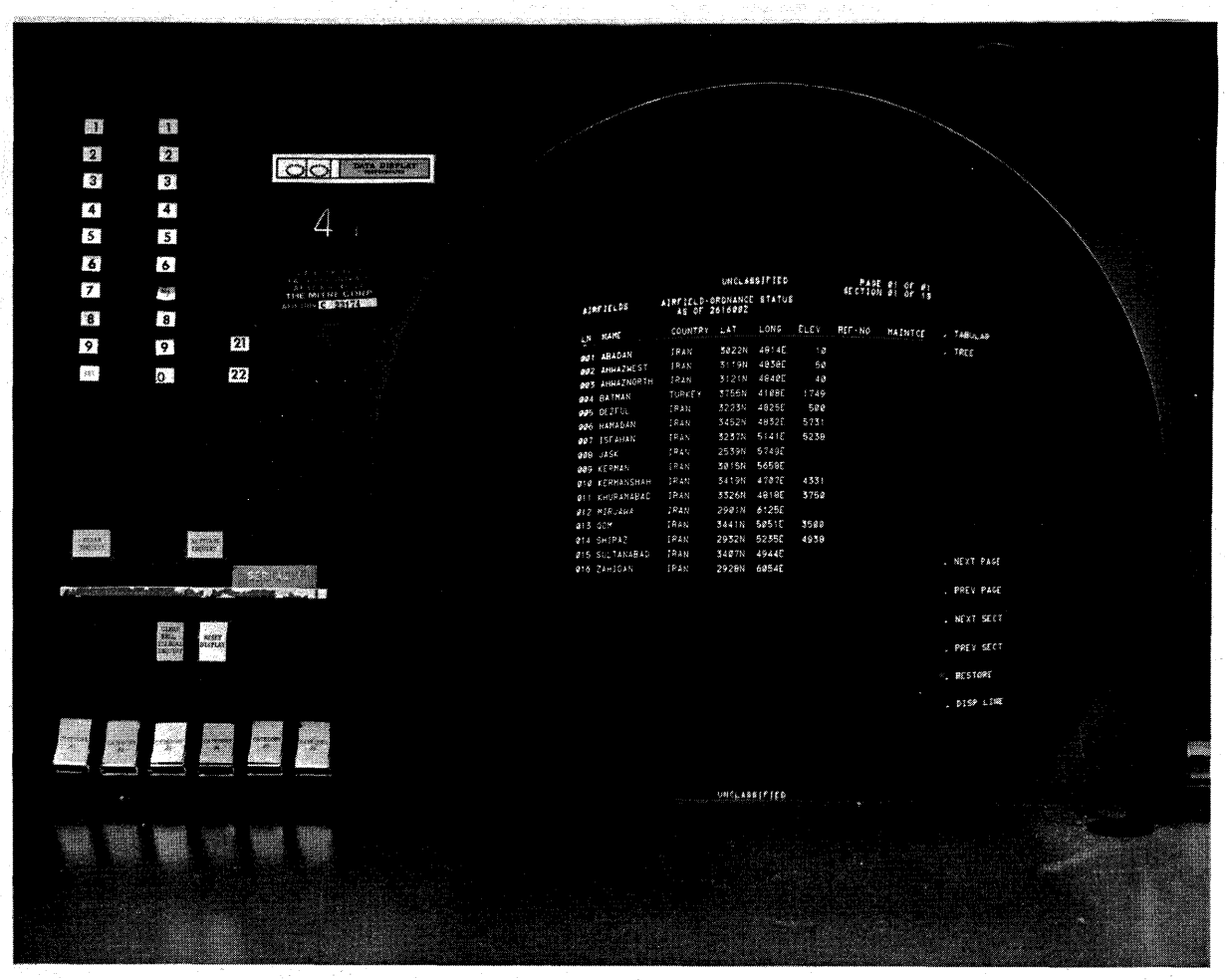



Figure 1. The Data-Display-13 Console with light pencil on the right and notebook display on the CRT.

construct and operate on-line logical and mathematical procedures, again using only a CRT display and a light pencil.

Since an understanding of the user's view of this data base is essential to an understanding of many of the other system capabilities to be described, it will be presented first. It will be followed by a description of the communication tree which the user uses to "talk to" the system. The final capability described will be the on-line procedure generation capability.

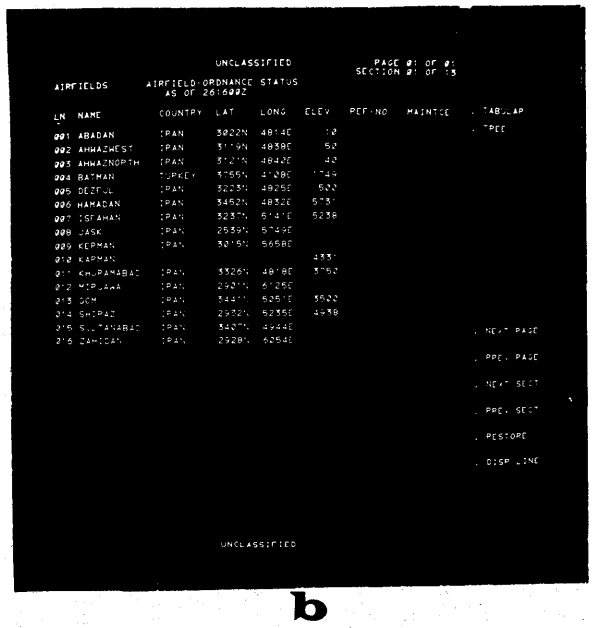
UNCLASSIFIED PAGE #1 OF #1
SECTION #1 OF 13

AIRFIELDS AIRFIELD-ORDNANCE STATUS
AS OF 261600Z

| LN | NAME | COUNTRY | LAT | LONG | ELEV | REF-NO | MAINTCE |
|---|---|---|---|---|---|---|---|
| 001 | ABADAN | IRAN | 3022N | 4814E | 10 | | |
| 002 | AHWAZWEST | IRAN | 3119N | 4838E | 50 | | |
| 003 | AHWAZNORTH | IRAN | 3121N | 4846E | 40 | | |
| 004 | BATMAN | TURKEY | 3755N | 4108E | 1749 | | |
| 005 | DEZFUL | IRAN | 3223N | 4825E | 500 | | |
| 006 | HAMADAN | IRAN | 3452N | 4832E | 5731 | | |
| 007 | ISFAHAN | IRAN | 3237N | 5141E | 5238 | | |
| 008 | JASK | IRAN | 2539N | 5749E | | | |
| 009 | KERMAN | IRAN | 3015N | 5658E | | | |
| 010 | KARMAN | | | | 4331 | | |
| 011 | KHURAMABAD | IRAN | 3326N | 4818E | 3750 | | |
| 012 | MIRJAWA | IRAN | 2981N | 6125E | | | |
| 013 | QOM | IRAN | 3441N | 5051E | 3500 | | |
| 014 | SHIRAZ | IRAN | 2932N | 5235E | 4930 | | |
| 015 | SULTANABAD | IRAN | 3407N | 4944E | | | |
| 016 | ZAHIDAN | IRAN | 2928N | 6054E | | | |

UNCLASSIFIED

a




b

Figure 2. Stromberg-Carlson 3070 print copy (a) of CRT display (b).

lines numbered consecutively 1-30, 31-60, and so forth. There are as many pages in a file as there are such sets of 30 lines. Each display is 64 characters wide. These 64 characters are divided into columns. The number of columns depends upon how many characters constitute a column and how much space is desired between columns. A set of columns is referred to as a section of a file. There are as many sections as there are sets of columns. The user's selection of a specific page and section of a file specifies a subset of lines and columns. Customarily, but not necessarily, lines identify objects, columns identify properties.

The user's notebook contains as many files as are required to hold the organization's data base. It also contains an arbitrary number of files with completely blank pages (see Fig. 3). These blank pages

## THE USERS' NOTEBOOK

The users of the system experience the total data base through tabular displays on their DD-13 cathode ray tubes. Any tabular display can be reproduced in hard copy, on call, on the S-C 3070 medium-speed printer (see Fig. 2). The hard copy is always a precise reproduction of the CRT tabular display.

Each display is a section of a page of a file in the user's notebook. Each display consists of 30

serve as the environment within the notebook in which the individual user builds his private data base and personal report formats. In such cases, the resulting personal files and reports, developed either on-line or off, are retained in the large computer-based notebook.

In AESOP, the distinction between a public file and a private one, a temporary file and a permanent file, a file and a report, becomes somewhat artificial. Procedural rules establish that a public file should not be revised by unauthorized members of the organization. Working files are simply private files that the user intends to erase at some later time, but which need not be destroyed if he changes his mind. Private files are generated, either on-line or off, from outside the existing data base or by "copy" operations from other files, public or

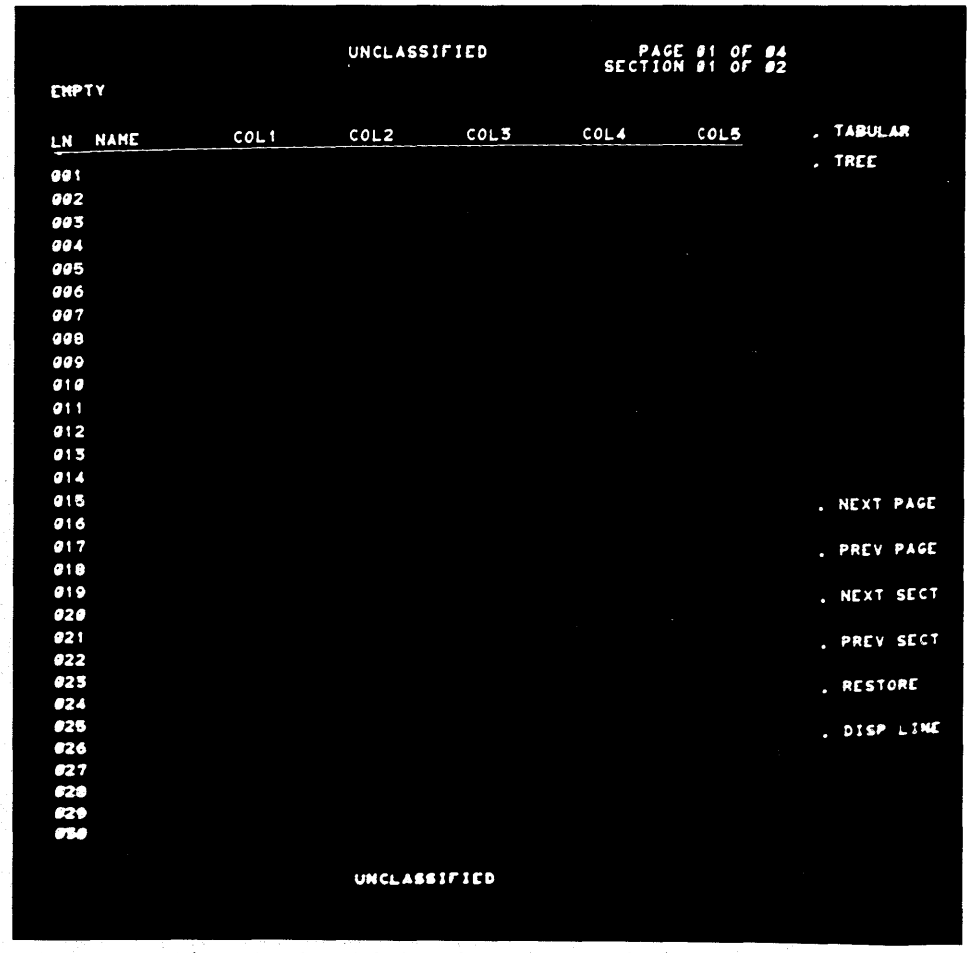



Figure 3. CRT Display of an empty notebook page/section. Light pencil commands appear at the upper and lower right.

private. Both files and reports, from the user's viewpoint, are simply CRT tabular notebook displays or their hard-copy reproductions.

The user of the system can browse through a file by firing his light pencil on either a page-turning, a section-turning, or a line-display command in the lower right-hand corner of the tabular display. He can turn pages and sections both forward and backward to scan all of the objects and properties of the file. He can also light pencil a specific line number or name, after having light penciled a request for the display of a line, and have all of the columns of data and the column headings displayed for him (see Fig. 4).

The format of each page and section of each file, which in effect is the format of a specific tabular CRT display, is under the on-line control of the system users. Among the parameters of the display open to on-line control are the following:

(a) whether or not the line (object) names are to be displayed;
(b) the number of columns (properties) to be displayed in each section of the file;
(c) the specific columns (properties) that are to be displayed and their sequence;
(d) the location and the size of the column in which each property is to be displayed;
(e) the choice of right, left or decimal justification of the data for any property.

Format is determined by data contained in one of the standard notebook files in exactly the same fashion as any other portion of the data base. In this "display adaptation" the names of all of the other files of the system are recorded as lines or objects. The columns contain all of the format parameters

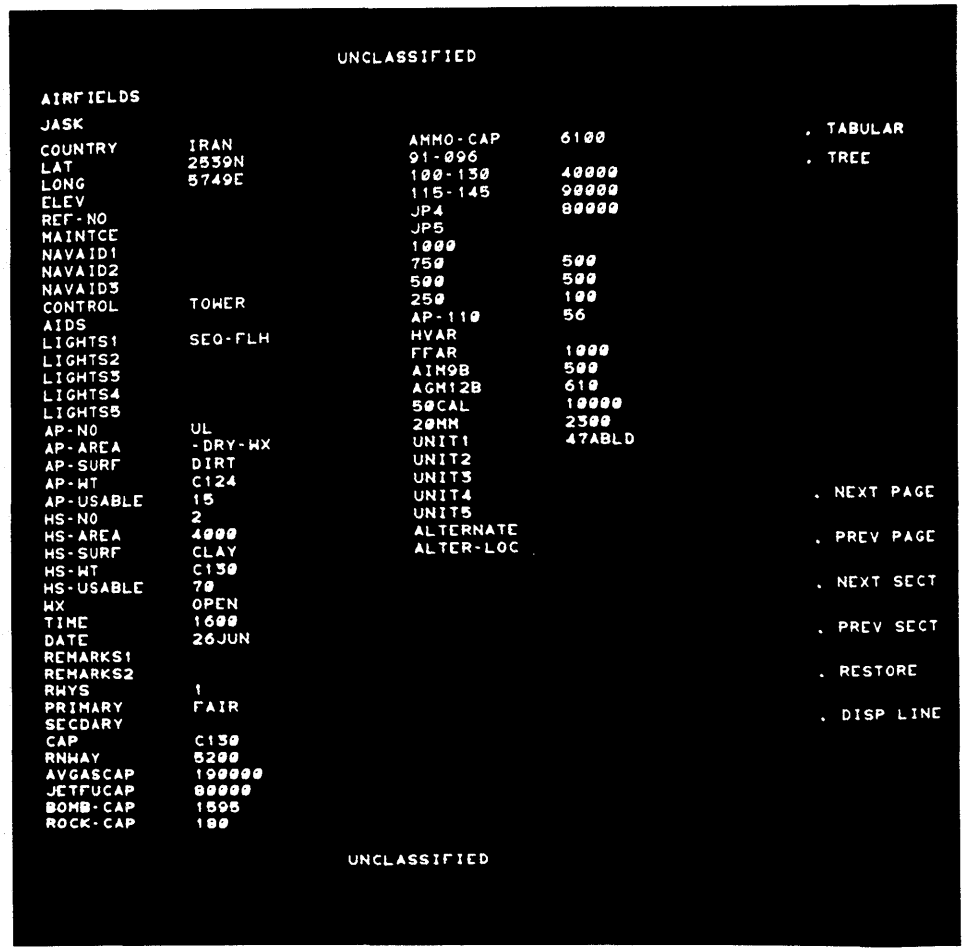



Figure 4. CRT Display of all column names and values for a specific line (name JASK) of a file (name AIRFIELDS) selected by use of the display line command (DISP LINE).

open to on-line control. Values of those parameters are interpreted by the system to determine the specific formats displayed to the user. Thus, when this "display adaptation" file is changed or updated on-line, in the same fashion as any other file in the system, there is an immediate resulting alteration of the display formats. (Compare Fig. 5 with Fig. 1).

The on-line ability to reformat the notebook display provides a report generation capability. When a particular report is desired, one or more notebook pages are formatted to reflect the desired report structure. Data can then be copied or generated into that portion of the notebook and a hard copy can be made of the resulting display. As a final step, the display can be returned to its original form.

## COMMUNICATION TREE

Communication with the system for the purpose of data retrieval, browsing through the notebook, data generation or format control is mainly accomplished by means of the light pencil and the display of the communication tree (see Fig. 6). The communication tree not only handles all of the problems of syntax and vocabulary necessary to access the data base, but it also shows the user all of the linguistic implications down each of the branches of the tree. In addition, at the top of the display as a message is being constructed, it shows him all of his previously selected options to serve as a short-term memory aid.

The branching points heading down the tree indicate legitimate options. The user indicates his de-

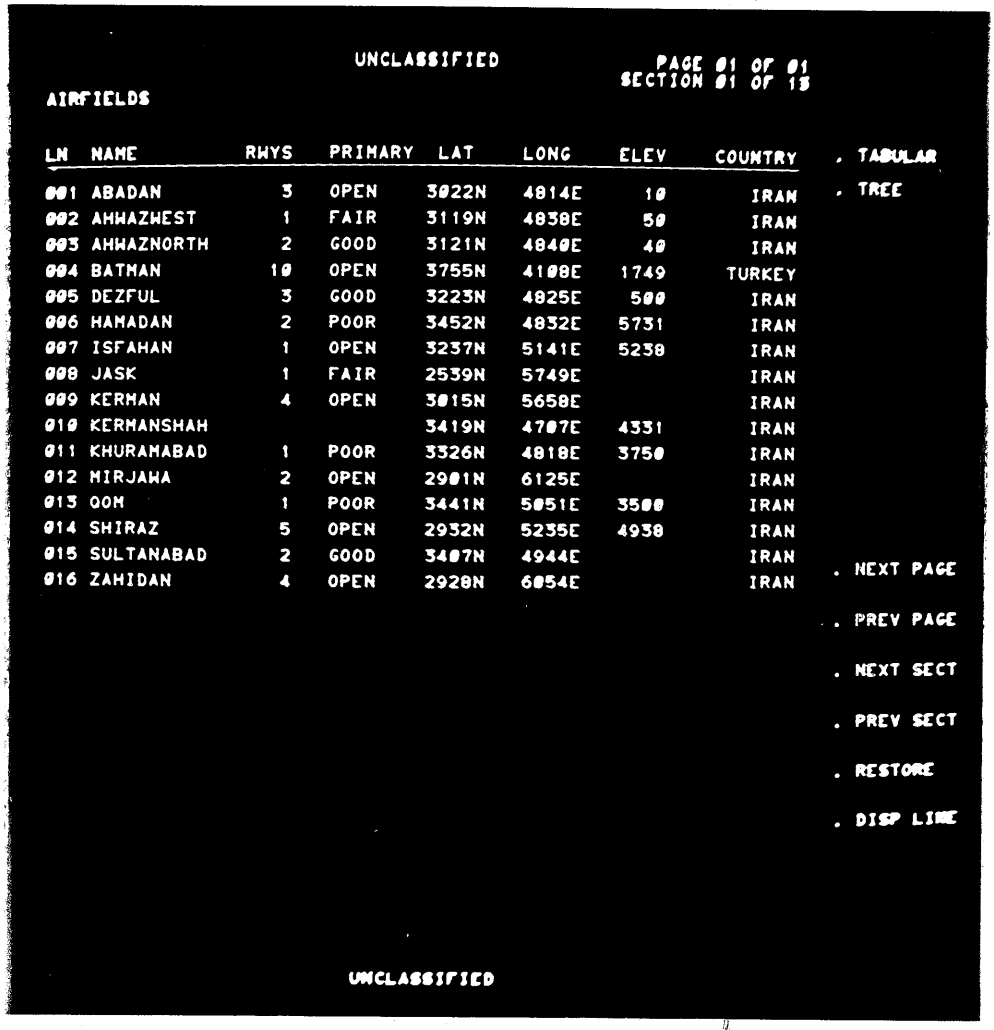



Figure 5. On-line reformatted display of page 1, section 1 of the AIRFIELDS file, originally formatted as in Figure 2. Columns have been rearranged, new columns added, spacing altered and right-hand justification of the COUNTRY column has been substituted.

sires by pointing the light pencil at the desired option currently displayed at the top of the tree. Each light pencil action (1) generates a display record of the part of the message constructed to date, and (2) moves the tree upward to indicate the next legitimate choice to be made or (3) displays a list of possible options to be taken at that particular point in the construction of his communication. Using the tree, the user need have no prior knowledge of either the current system index, vocabulary, or syntax, and he need not keep a record of the communication he is in the process of constructing.

For example, as in Figs. 7 and 8, to construct a call on the file, the user first requests the communication tree by light penciling the word TREE in the upper right-hand portion of his display. He then light pencils "GET" at the top of the tree. This action writes GET at the top of the display and moves FILENAME to the top of the tree (Fig. 7a). Light penciling FILENAME then shifts the display to supply a list of all the names of the files currently in the system (Fig. 7b). He points his light pencil at the file he wants, such as AIRFIELDS, which (1) puts GET AIRFIELDS at the top of the display, and (2) returns the communication tree display with the next

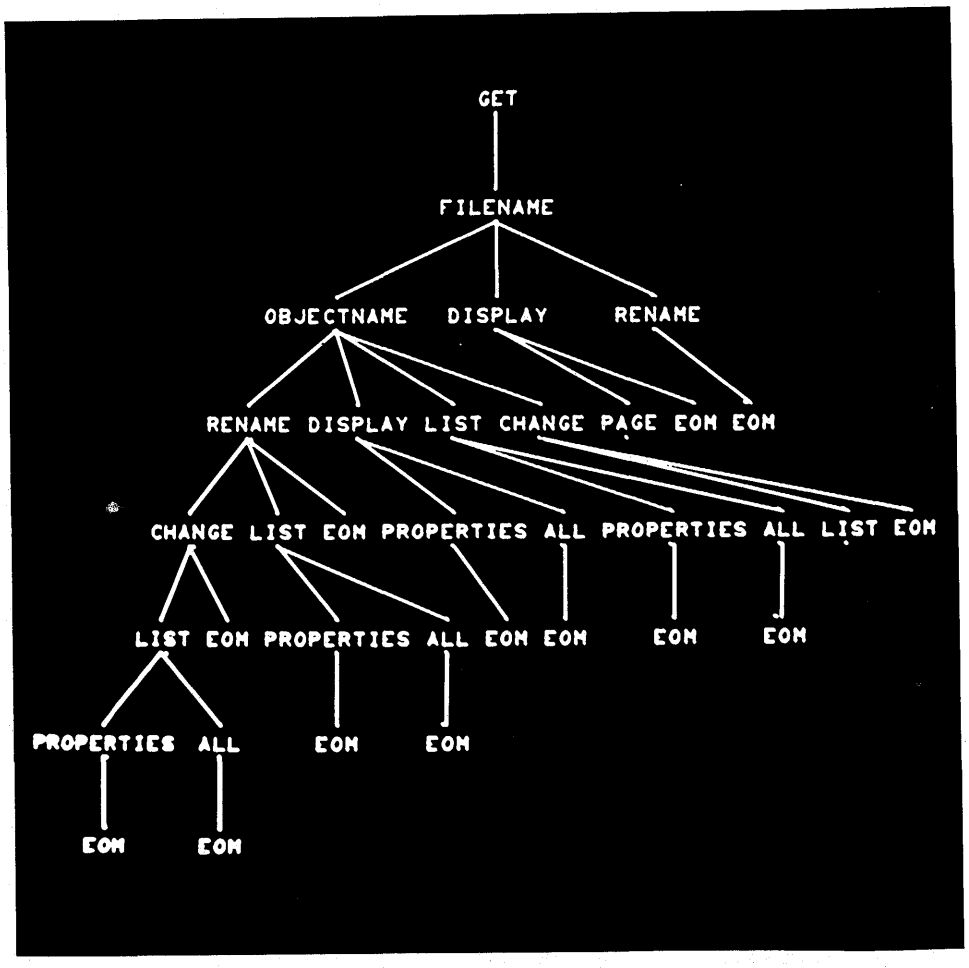



Figure 6. The basic communication tree with all well-formed control statements indicated by downward branches.

valid option displayed at the top of the tree (Fig. 7c). If he chooses OBJECTNAME, all of the current object names in the AIRFIELDS file are displayed, enabling him to choose the desired object by light penciling it (Fig. 7d). The object name, for example KARMAN, is then transferred to the top of the display and the communication tree shows the next valid options (Fig. 7e).

To generate a string of characters for inclusion in the notebook, a matrix of alphanumeric characters and an accumulator are displayed simultaneously. The user selects characters by pointing the light pencil at them in sequence. His choices are recorded in the accumulator and errors can be canceled by a suitable light pencil command. When the appropriate character string has been generated, the command PROCESS moves the display to the next option (Figs. 7f and 8a).

By the alternate selection of appropriate column names and the use of the alphanumeric matrix and accumulator, the user can modify his data base on-line (see Figs. 8b, c, d, and e). The result appears as in line 10 of Fig. 8f (see line 10 of Fig. 2 for prior file appearance).

In this fashion, messages are composed, displayed, and executed by the user who need know nothing more about the communication language than is necessary to point out his desires. By using the tree to review the catalog of currently available file names, object names within files, and property names within files, the user is able to browse through the data structure as well as to familiarize himself with the range of options and implications available to him.

The sophisticated user is not restricted to this mode of communication if his knowledge of the

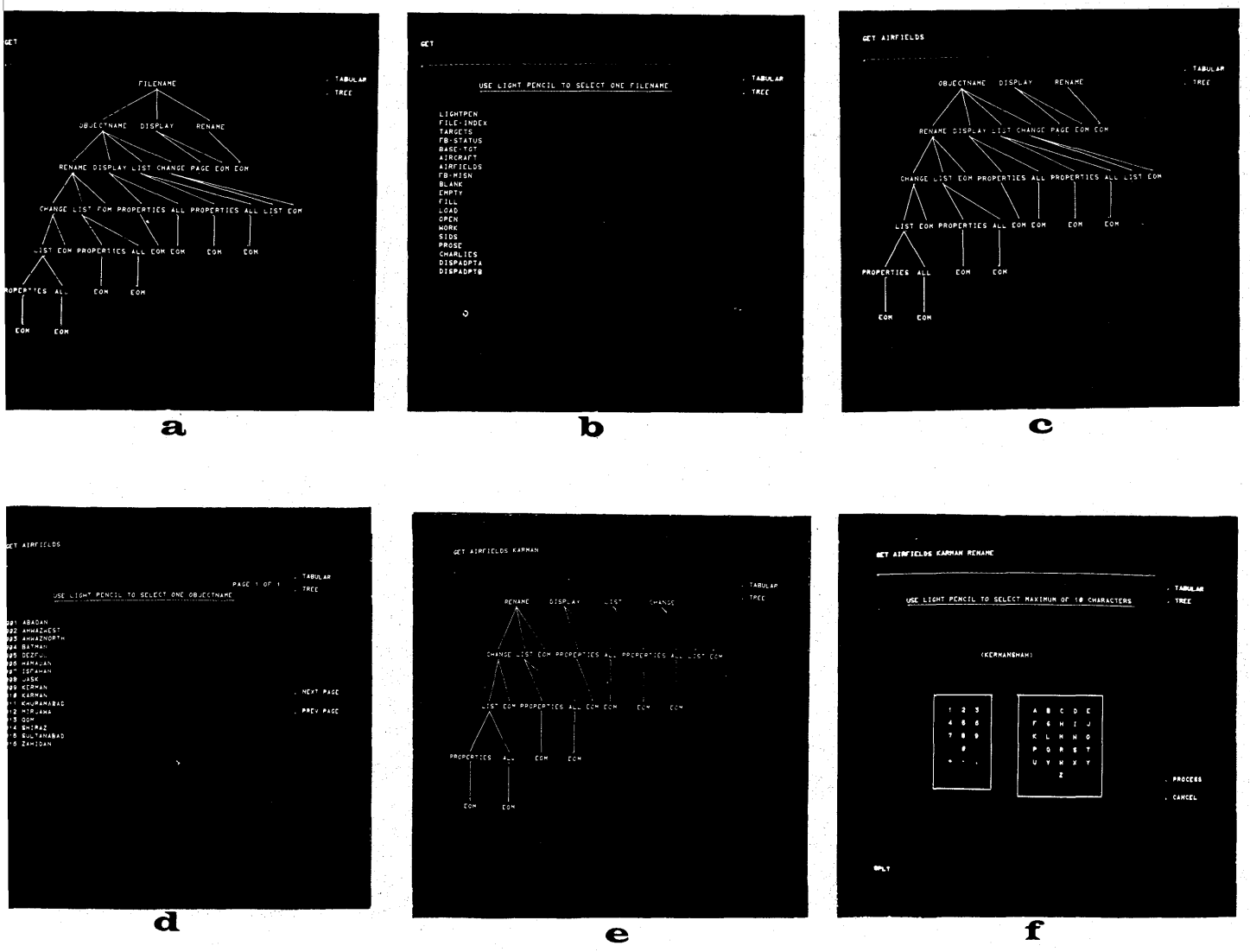



Figure 7. Steps in the on-line construction of a file modification command as described in detail in the text.

system exceeds that implied by the look, choose, and point philosophy. Users who know the syntax and vocabulary of the system may directly compose on the on-line typewriter any message that can be generated by means of the light pencil and communication tree. In fact, at any time in the evolution of the AESOP prototype, the typewriter user has a more extensive language, portions of which, in sequence, are transferred to the tree as this proves useful. Currently, requests to print hard-copy, to erase or to transfer portions of the notebook into other portions are accomplished only by means of the typewriter. System error messages are primarily provided on the typewriter.

## PROCEDURE GENERATION

The user can also process the data of the system, either by executing or modifying established routines or by constructing new routines using his light pencil and CRT display. In order to do this, he calls up an on-line algorithm construction display

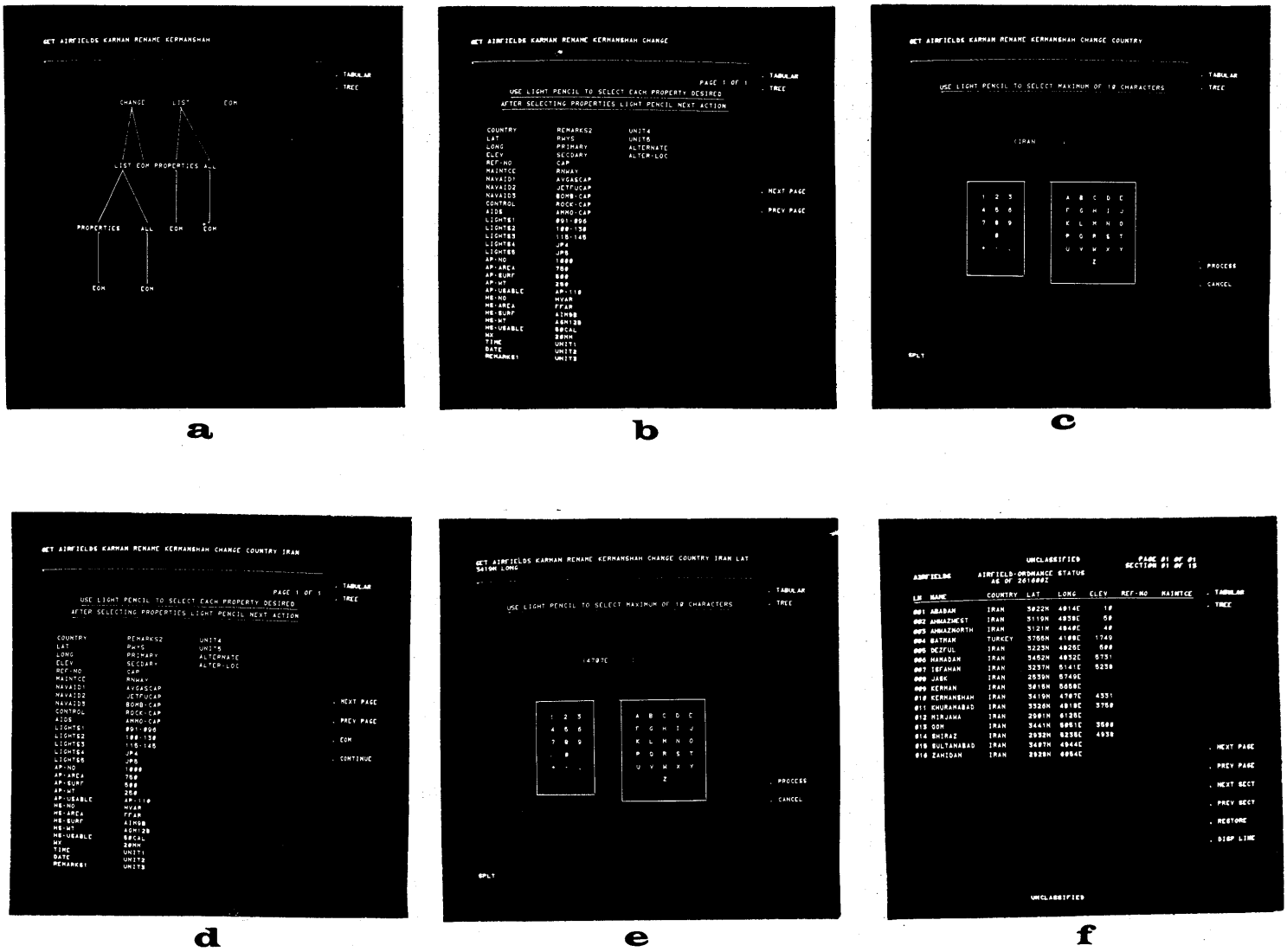



Figure 8. Figure 7 continued.

called OAK-TREET. The display appears as a tree with one branch displaying commands which can be light penciled to build and execute a program. A second branch displays some basic classes of operators and operands or previously established functions to be used as parts of this program. A workspace within which to build a tree representation of the desired procedure exists as another branch (Fig. 9a).

Each time a user light pencils a command, it appears in the upper right-hand corner of the display (Fig. 9b). When he light pencils a class of operators, the specific operators within that class are displayed on a separate branch of the tree (Fig. 9c). When he light pencils one of these specific operators, it also appears in the upper right-hand corner of the display (Fig. 9d). Once a command and an operator are so displayed, he can then light pencil any portion of a tree in the workspace and the command will be executed, using the operator in question, at that specific location in the workspace (Fig. 9e). In this general fashion, the user builds,

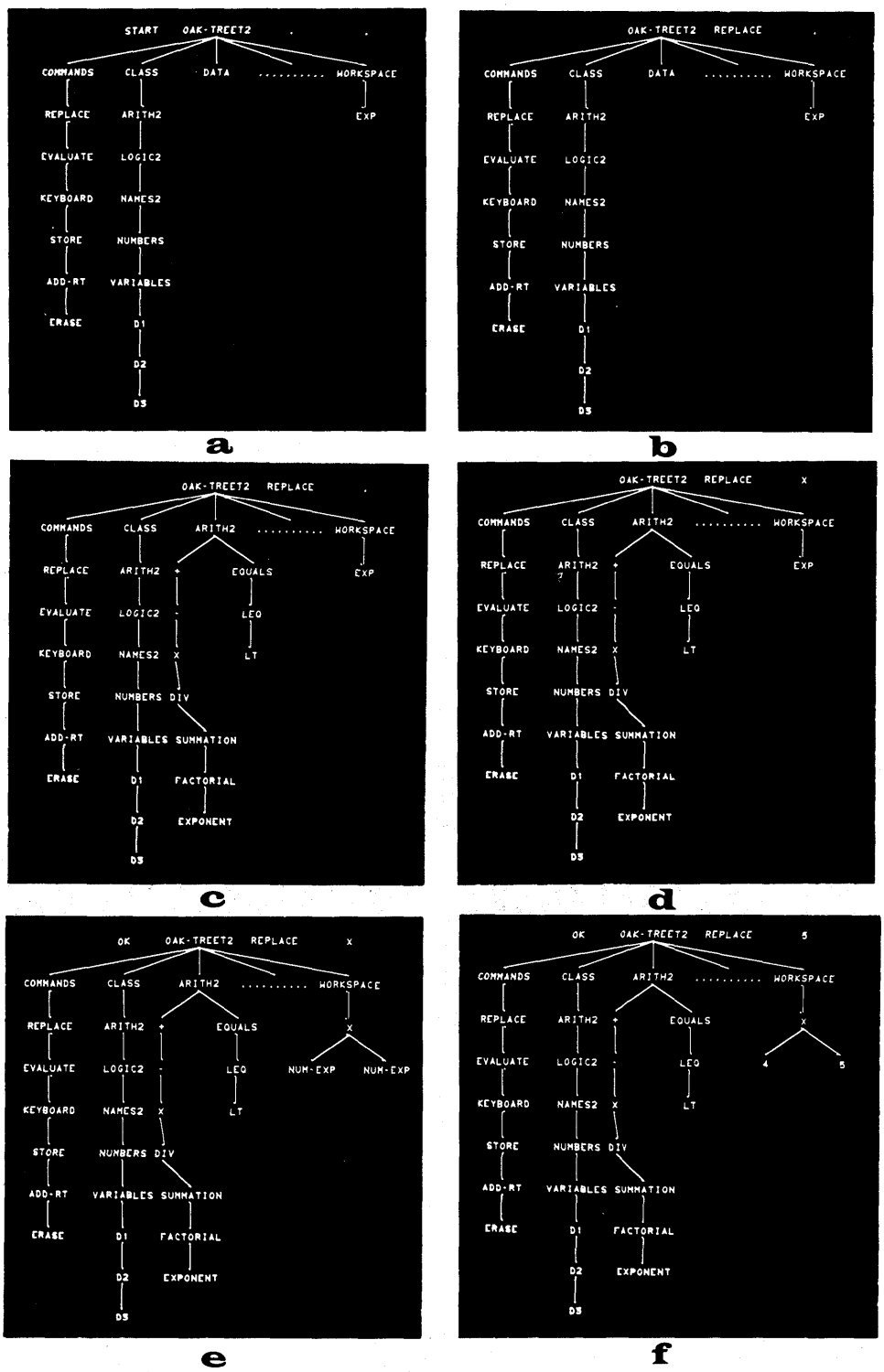



Figure 9. Steps in the on-line construction of a primitive procedure to multiply 4 by 5.

stores, retrieves, modifies, links, and in other ways develops a catalog of logical and mathematical expressions for use with his data base.

Numbers are generated by means of a special display of a tree of numerical characters which can be added in an accumulator by light pencil selection. These numerical strings are then transferred to the upper right-hand accumulator in the OAK-TREET display for further use.

The OAK-TREET capability operates in two modes C. The first displays the commands, operators and workspace simultaneously. This mode may crowd the workspace but the display logic drops portions of the workspace tree off the display face, leaving branch markers to indicate what parts are gone whenever the workspace density exceeds a critical value. The second mode is used for inspection. It turns the entire display face over to the workspace tree. In this mode, any point on the construction can be brought to the top of the display by means of the light pencil in order to expose more of the details of the tree below that point.

As an example of the use of the OAK-TREET feature consider Figs. 10, 11 and 12. A user interested in a gross estimate of fuel consumption as a function of distance for a high performance aircraft in level flight might build the following planning procedure. Using the command REPLACE and a previously stored subroutine with the name JT1, he puts this routine into the workspace by pointing the light pencil at the node immediately below the word WORKSPACE (Fig. 10a). The previously defined routine JT1 is then displayed at that point in the workspace. In this example, the routine is a gross calculation of fuel consumption and is called POL-LBS. It involves multiplying the fuel consumption rate by the ratio of the distance flown over the average speed of the aircraft. The user may substitute parameters into this procedure (see Fig. 10b) and then call for it to be executed by pointing the light pencil at the command EVALUATE. The result of the calculation will be printed by the on-line typewriter.

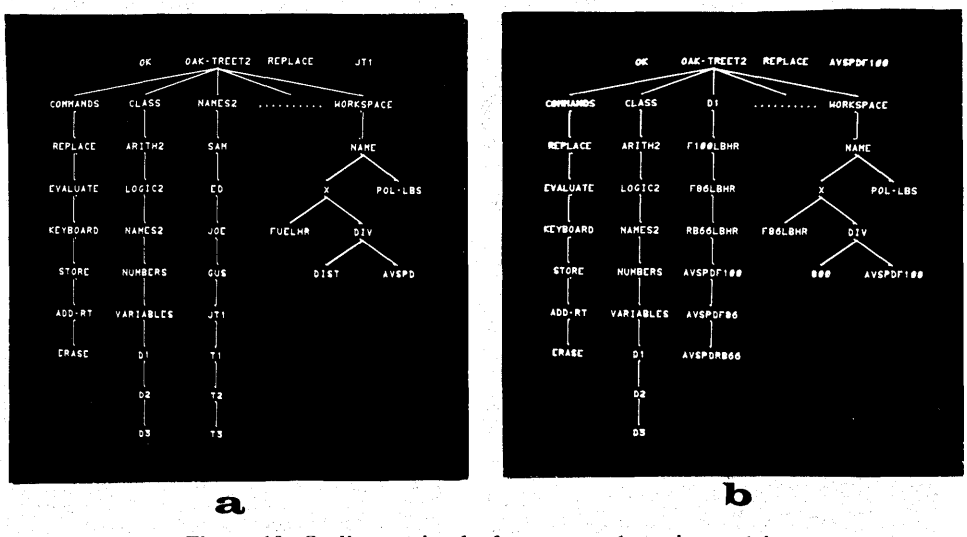



Figure 10. On-line retrieval of a prestored routine and insertion of parameters.

To set up a more complex expression for evaluation, the user now changes the distance flown to the variable C multiplied by 100. He does this by replacing 800 with the arithmetic operator X (multiply), then setting the branches to C and the number 100. Both are done with the command REPLACE (see Fig. 11a). The user now temporarily stores this modified routine under the arbitrary name GUS. He then uses the workspace to establish a new variable D set to the value of the routine stored under the name GUS. He does this by putting the logical operator = immediately under the word WORKSPACE (thereby erasing everything else) and then adding under it the variable D and the expression stored under the name GUS (see Fig. 11b).

This new expression is now stored temporarily under the name JOE. A conditional expression is then built in the workspace by putting the logical operator

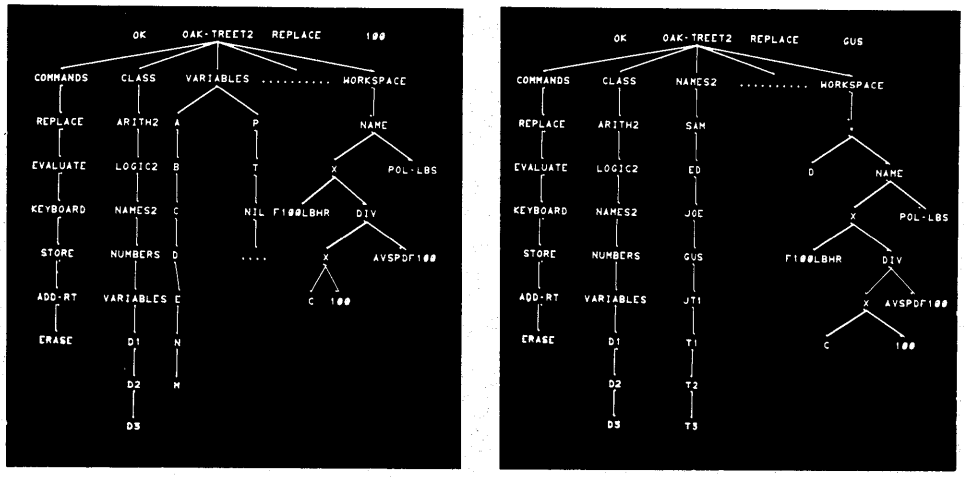



Figure 11. On-line modification of a procedure, storage and use as a subprocedure.

IF-THEN-ELSE immediately below the word WORKSPACE. The user then introduces the logical operator TYPE on both terminal branches of the conditional to indicate that both results of the conditional test should be typed by the on-line typewriter. He then adds the variable D to the true branch, and the same plus a marker of four dots to the false branch (see Fig. 12a).

The conditional test is set as a comparison to be made by the operator LEQ meaning less than or equal to. The comparison is to be made between the value of D and the number 400 (see Fig. 12b). This portion of the routine is now stored under an arbitrary name ED and a next routine is constructed in order to execute the previously defined routines.

This next routine involves the use of the operator DO. It is used to first find the value of D and then perform the conditional test. This is accomplished by putting the routine previously stored under the name JOE as a first branch for DO and then using the command REPLACE, inserting the routine previously stored under name ED (see Fig. 12c). This DO procedure is now stored under the name SAM and a next higher order routine is built.

This next routine involves the operator FOR which is used to run through a sequence of values for a given variable. The variable in this case is the C which was previously used as the variable for distance in the fuel calculation. C will be set to values from 1 to 10. The previously defined DO expression, stored under the name SAM, is then added to be evaluated for these values of the variable C (see Fig. 12d).

The system can be shifted to the second or inspection mode and the complete routine can be expanded for viewing of any part of the tree (see Fig. 12e). The light pencil is used to bring any portion of the tree to the top center (see Fig. 12f).

When the command EVALUATE is executed, the typewriter prints the results of the fuel calculation from 100 to 1000 miles in increments of 100 miles followed by the indication of four dots when the fuel requirement exceeds 400.

The total routine can now be stored as a permanent part of the data base and later used in its entirety. The routine can also be returned to the workspace for modification or to detach subroutines for other purposes. As long as the total routine remains filed in the data base, it is available for inspection and application on-line.

As complex or extensive routines are constructed, it becomes increasingly undesirable to display the total structure to the operational user of the routine. In such cases only the name of the routine and the names of the insertable parameters need be displayed in the OAK-TREET workspace. The appropriate values for these parameters can then be inserted by the user and the named routine evaluated on call. (See Figure 13 for example.)

In this example a routine called STATUS requires the insertion of a destination code (AREA), the type of aircraft (AC-TYPE), and number of aircraft (NO-AC) to be flown to that destination, and the number of originating airfields to be checked for the availability of these aircraft (NO-AF). The routine

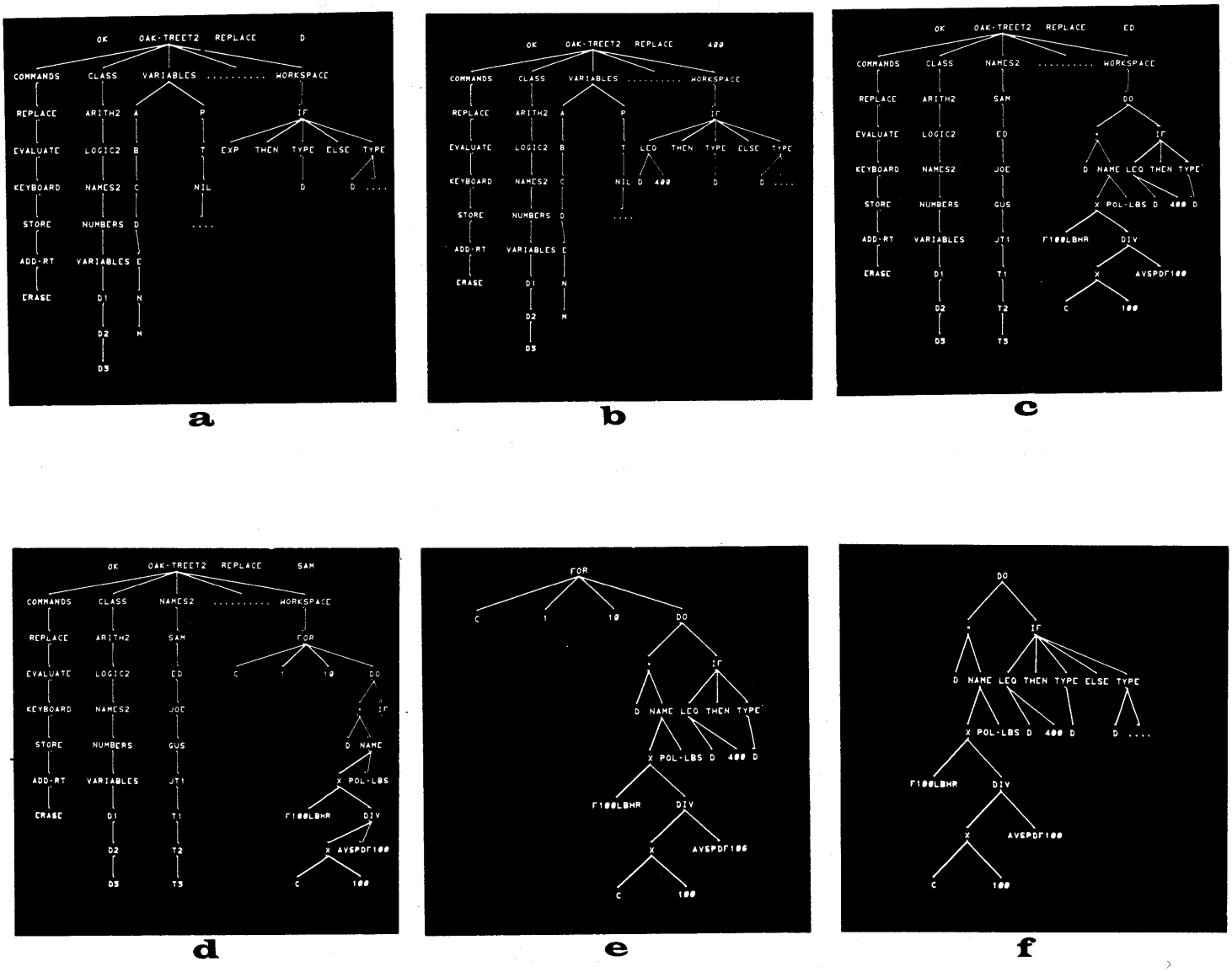



Figure 12. Steps in the on-line construction of a complex procedure using previously on-line defined and stored subprocedures. Expansion for inspection in (e) and (f). See text for a detailed description of the procedure.

then locates up to this number of originating airfields with sufficient aircraft available, in order of increasing distance from the destination. It lists the airfields, the number of aircraft available, the organizational designation of the aircraft, the distance and time required to reach the destination.

The detailed structure of user routines such as STATUS is examined and debugged in another display environment called DEBUG. (See Fig. 14.) In the DEBUG function, the structure of the routine is displayed as the second leftmost limb of a tree along with a set of commands which permit the programmer to examine any selected portion of the routine in question and then modify and redefine it.

A routine is brought to the top of the second leftmost limb of the DEBUG tree by an on-line typewriter command DEBUG (name). Any subtree of that routine is then moved to the top of that limb by pointing the light pencil at the node on top of the substructure in question. The structure can

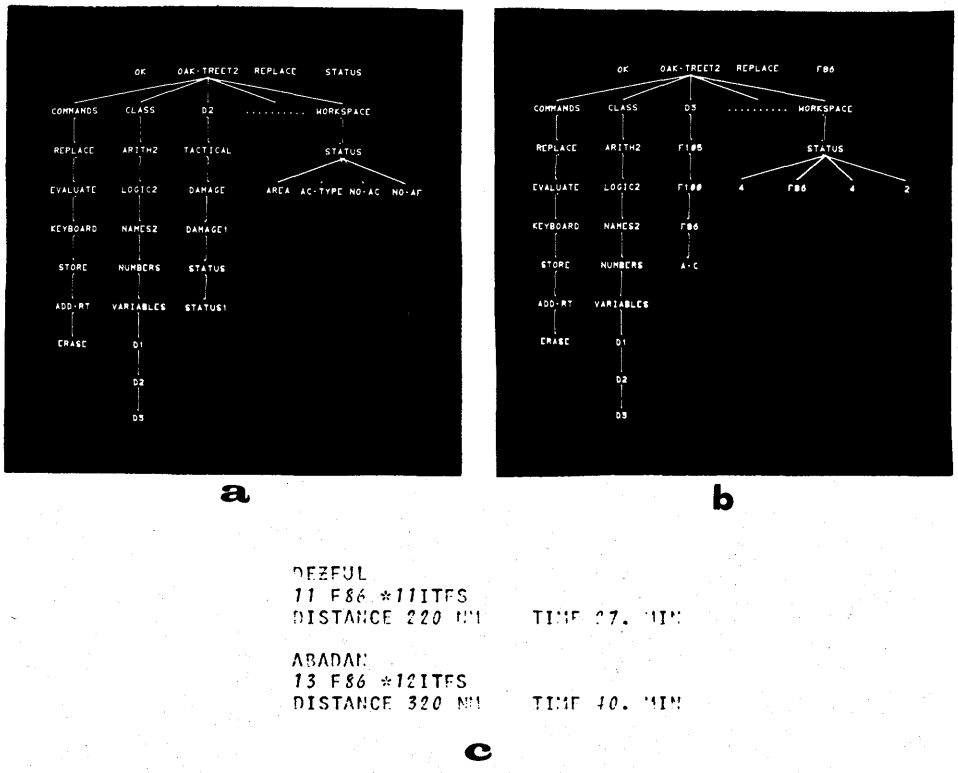



Figure 13. The use of OAK-TREET for parameter insertion in fixed procedures.

also be displayed, in whole or in part, by itself, in either a multinode or single node state.

By light pencil, using the command OAK, the substructure displayed on the second leftmost branch of the DEBUG tree is moved into the workspace of OAK-TREET. In this position it can be modified using the standard OAK-TREET capabilities. The modified structure is then moved back to the second leftmost limb of the DEBUG environment by light penciling the OAK-TREET command KEYBOARD. The modified structure replaces the previous structure in the system by light penciling the redefinition command DEFTR. The revised and redefined function can then be evaluated using the command TEST.

This procedure for on-line program development and revision applies to all of the user and system functions constructed in and interpreted by the AESOP list processor, TREET (see the Appendix for details). For example, it is possible to DEBUG (FACTORIAL) as in Fig. 14(a), (b), (c) and (d) or DEBUG (STATUS) as in (e) and (f). It is also possible, for example, to DEBUG (OAK) or DEBUG (DEBUG).

In previous examples, data used in the execution of a user's routine were inserted by means of the user's light pencil. The results of executing the routine were printed by the on-line typewriter. However, it is also possible to have the routine call for data directly out of the computer-based notebook and then place the results of its execution in the notebook. In effect, the TREET processor can be considered one of the on-line users of the notebook-based file. The TREET processor retrieves data from the notebook using any of the system retrieval capabilities available to the other on-line users of the notebook. It stores data in the notebook using the same range of storage capabilities available to human users of the system. It uses the same communication language.

For example, in Fig. 15, the file BLANK is organized as a scratchpad for keeping track of the input and output data derived from the evaluation of the STATUS1 routine. The routine STATUS1 differs from STATUS (see Fig. 13) mainly in its use of the notebook for its data base. Working headings are established in BLANK for COL1 through COL4 to fix the column locations for input to STATUS1. Column headings are established in

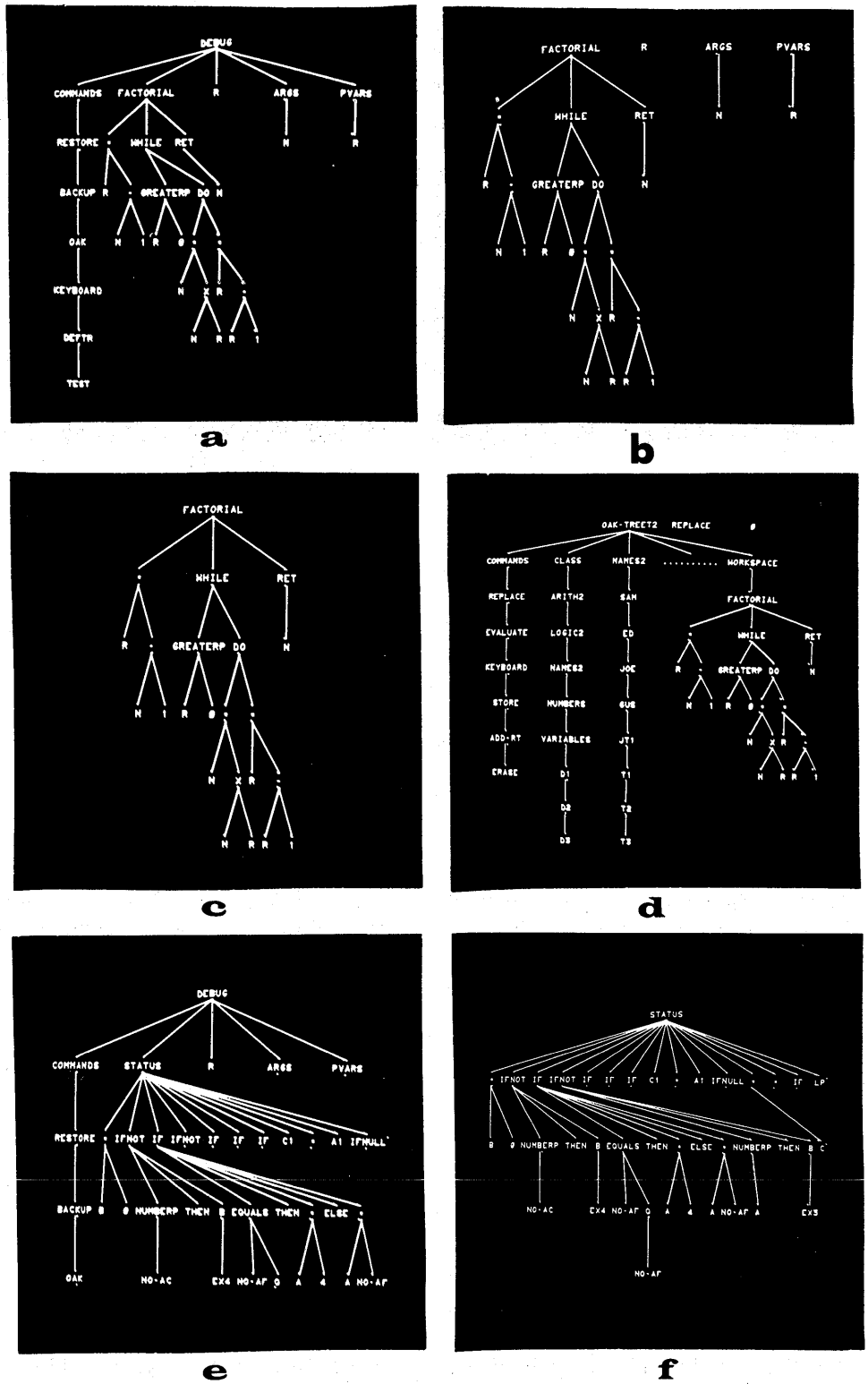



Figure 14. Steps in the generation or modification of an interpreted system or user procedure. See text for details.

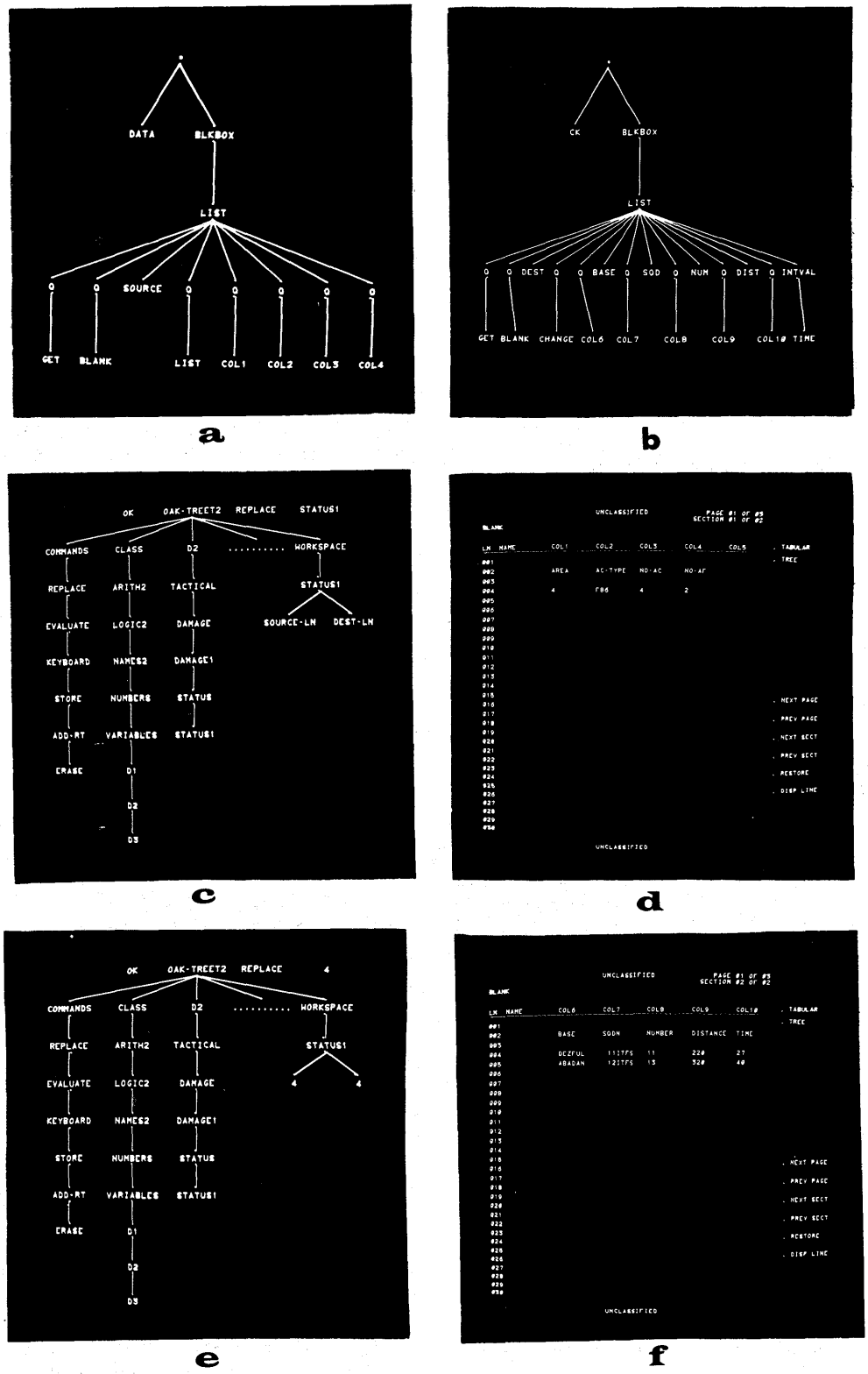



Figure 15. Steps in the use of the notebook for data input to and data output from user procedures. See text for details.

COL6 through COL10 for recording outputs from STATUS1. STATUS1 is constructed to include a data call on the file BLANK in order to list input data from COL1 through COL4, as shown in Fig. 15(a). It also includes a data change order to insert output data in COL6 through COL10 as in Fig. 15(b). Source and destination line numbers are supplied by the on-line user in OAK-TREET, as in Fig. 15(c). Input data are then either generated on-line or transferred from other portions of the notebook into BLANK, as in Fig. 15(d); the appropriate line numbers are inserted into STATUS1, as in Fig. 15(e); and the execution of the STATUS1 routine updates the notebook as in Fig. 15(f).

Using this type of on-line capability, in conjunction with capabilities for moving data from one location in the notebook to another, the human user of AESOP can execute routines, store resulting data in temporary notebook files, review these results, and then select those for transfer into permanent organization files and/or reports.

## APPENDIX

### *Aesop Software Structure*

The Aesop prototype is an evolutionary experimental system. As such it is incremental in growth. To facilitate this growth an attempt has been made to make the system modular as much as seemed practical. There are certain elementary system functions such as data retrieval, data updating and data display. At another level there are additional functions that are preprocessors or switching routines for the elementary functions. As an example there is a COPY action that will copy data from one system file into another. It does this by using the data retrieval and then the data updating routines in succession. Fig. 16 outlines the functional organization of the system.

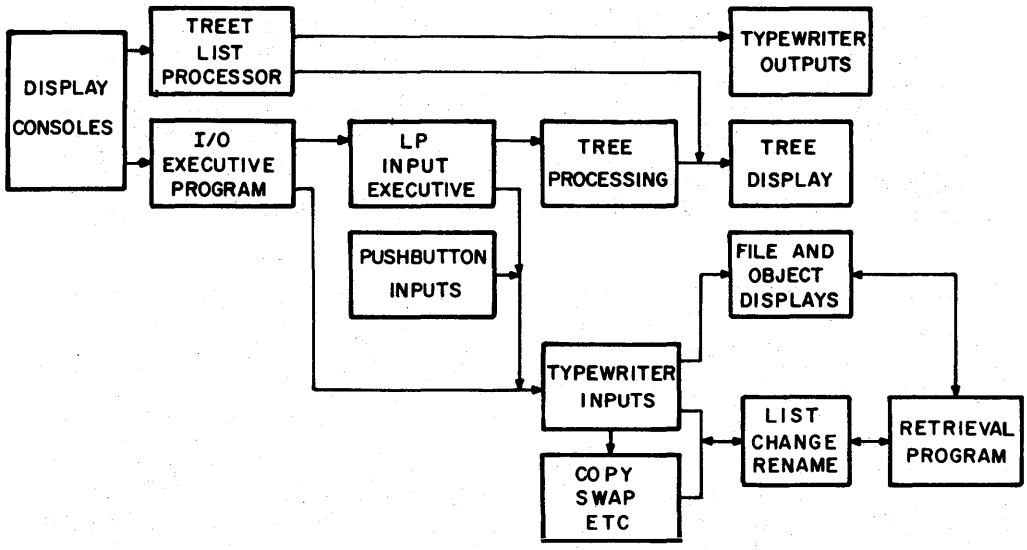



Figure 16. Simplified schematic of the AESOP A/1 software interrelations.

The total system occupies about 35,000 computer words. The remainder of the computer core memory is used as free storage space for list structures. The two prime considerations in the construction of the system were:

1. Ease of extension to permit the system to grow as new capabilities are added to it.
2. Speed of operation to give the operator fast response to all of his inputs.

Economy of storage has not been a prime consideration. All of the system data base is stored in the disk unit.

TREET is a general-purpose list processing system* written for the IBM 7030 computer at the

MITRE Corporation. All programs in TREET are coded as functions. A function normally has a unique value (which may be an arbitrarily complex list structure), a unique name, and operates with zero or more arguments. A function may or may not have an effect on the system. Some functions are used for their effect, some for their value, and some for both. The OAK-TREET function as it appears to the operator has commands, data classes and data which can be used for procedure construction.

What follows is a simplified explanation of the principal nodes of the limbs of the OAK-TREET tree. OAK-TREET is constructed in TREET and OAK-TREET expressions are evaluated by the interpretive list processor.

## OAK-TREET

### COMMANDS

REPLACE — The effect of this command is to place in the workspace, at the point indicated by the light pencil, the expression, symbol, or structure indicated by its argument.

EVALUATE — When this command is signalled by the light pencil the expression in the workspace is evaluated.

KEYBOARD — When this command is signalled by the light pencil, the system will expect the next command to come from the on-line typewriter.

STORE — A copy of the expression which is indicated by the light pencil is maintained in the system under the name of the argument.

ADD-RT — The expression indicated by the first argument of this command will be added to the workspace with the same parent as the node indicated by the light pencil.

*See E. C. Haines, "The TREET List Processing Language." SR-133, The MITRE Corporation (April 1965).

ERASE — Removes the node (and all nodes dependent upon it) indicated by the light pencil from the workspace.

### DATA

ARITH2 — Arithmetic Operators.

+ — Computes the sum of its arguments.

– — Computes the difference of its arguments.

X — Computes the product of its arguments.

DIV — Computes the quotient of its arguments.

SUMMATION — Sums an expression while a variable goes from some number to another in increments of one.

FACTORIAL — Computes the factorial of its argument.

EXPONENT — Raises its first argument to the power indicated by its second argument.

EQUALS — A predicate which checks for equality of its two arguments.

LEQ — A predicate which is true if its first argument is less than or equal to its second argument.

LT — A predicate which is true if its first argument is less than its second argument.

LOGIC2 — Logical Operators.

TYPE — Types out the value of its arguments on the typewriter.

DO — A convenient way of grouping several expressions under one node.

EVAL — Evaluates its argument which must be an expression in Cambridge Polish notation.

CONS — Computes the list of its second argument augmented by its first argument.

MEM1 — Computes the first member of its argument which must be a list.

REM1 — Computes the list of its argument with its first member removed.

AND — Logical Intersection. Value is TRUE if both arguments are not NIL.

OR — Logical union. Value is TRUE if either (or both) argument is not NIL; NIL otherwise.

NOT — Logical negation. Value is TRUE if its argument is NIL.

ATOM — A predicate which asks whether its argument is atomic.

EQUALS — A predicate which checks for equality of its two arguments.

PROG2 — Has as its value its second argument. It is useful for attaching a different value to a computation.

WHILE — Evaluates its second argument while its first argument is (evaluates to) true.

IF-THEN — IF (a1 THEN a2) The value of this expression is the value of a2 if a1 evaluates to true; NIL otherwise.

IF-TH-ELSE — IF (a1 THEN a2 ELSE a3) The value of this expression is the value of a2 or a3 depending on whether a1 is true or false.

FOR — Provides a convenient way to execute an expression (its fourth argument) for a numerical range (between the values of its second and third arguments) of a variable (its first argument).

= — This is the assignment operator. It sets its first argument (which must be a variable) to the value of its second argument.

Q — Quotes its argument.

ADL — The second argument of ADL must be a variable which evaluates to a list. ADL sets that variable to CONS (EVAL-($arg_1$)$arg_2$) thus effectively adding something to a list.

CHOP — The single argument of CHOP must be a variable which evaluates to a list. The value of CHOP is the first member of that list. The variable is set to the remainder of the list.

FNA — The value of FNA is the value of its argument considered as a function applied to no arguments. Its only purpose is to represent a function of no arguments in tree structure.

NAME — The value of NAME is the value of its first argument. The second argument is ignored. Name is used to label an expression.

NAMES

This is a set of undefined symbols which may be used as the name under which a routine is stored for later use.

NUMBERS

This calls up a function which permits any integer to be constructed by pointing the light pencil to its digits in sequence.

VARIABLES

This is a set of variables which can be used in an expression in the workspace.

## DEBUG

DEBUG is a function which allows other functions to be displayed and changed on-line. It works with any interpreted function but cannot display a machine language coded function. Most of the work of debugging a function is done within DEBUG using the light pencil. In the DEBUG display are five branches from left to right, (1) COMMANDS, (2) name of the function to be examined, (3) type of function, (4) ARGS, and (5) PVARS.

### VIEW ACTIONS

If any node other than one in COMMANDS limb is light penciled, then the multiple rooted subtree headed by that node replaces the tree structure in the second leftmost branch to the right of the COMMANDS branch. This feature allows one to view all of a function that is otherwise too large to fit on the display, to concentrate attention on a particular substructure of a function, or to select which part of the tree will be taken into OAK-TREET for modification.

### COMMANDS

RESTORE — This restores the display to its original position thereby cancelling all previous view actions.

BACKUP — This command cancels the last previous view action (if any have been performed since the last RESTORE).

OAK — The tree on the second leftmost branch of DEBUG, to the right of the COMMANDS branch, is placed in the workspace of OAK-TREET and the OAK-TREET function is entered. Changes to the tree may be made as desired. DEBUG is then reentered with this modified structure by light penciling the command KEYBOARD in OAK-TREET.

KEYBOARD — This command returns control to the on-line typewriter keyboard. If the control has been returned to the on-line typewriter, the DEBUG function may be reentered without starting over again by typing R(). Changes may also be made using keyboard tree changing functions.

DEFTR — DEFTR redefines the function according to the present (modified) configuration of the tree. (Changes made to the displayed version are not reflected in the function itself until this is done.)

TEST — This command initiates the processing for execution of the function displayed. The value of the function is printed by the on-line typewriter.

### BRANCH TWO

On the second leftmost branch appears the name of the function being examined, and under it a list of statements and location symbols. Location symbols are represented by a node containing that symbol. Statements are represented by trees in the same fashion as in OAK.

### TYPE

The function type is normally regular type R. Type F indicates that the arguments of the function should not be evaluated prior to evaluating the function itself. Type U functions allow an arbitrary number of arguments to be specified; the arguments are collected in a single list and given to the function as one argument. A type FU function is the combination of types F and U.

### ARGS

The list of arguments specifies which symbols will be set as the values of the arguments of the function when it is called. The old values (if any) of the symbols are automatically saved and restored.

### PVARS

The value of an argument in the program variables list is automatically saved and restored by the function.

# STRUCTURING PROGRAMS FOR MULTIPROGRAM TIME-SHARING ON-LINE APPLICATIONS*

Kenneth Lock
*California Institute of Technology*
*Pasadena, California*

## INTRODUCTION

The modern art of computation has developed from plugboard programming through the stored machine instruction programs controlled by the users on the consoles, then to problem-oriented symbolic programs computed in the batch mode, towards the on-line computing during which the users have a large amount of control over their programs. The lower cost per computation and flexibilities of a large capacity high-speed computer naturally lead us to consider the provision of on-line computing service to several users on a single high-performance machine in a time-sharing mode, rather than several smaller machines, one for each individual. To maximize the efficiency of a man-machine team working in an on-line computing mode, it is desirable to let the man choose the language—say English—for communication and to let the machine do the translation. This idealistic goal is not impossible, but is currently impractical. A good compromise is to select as the user language a formal language such as ALGOL, FORTRAN or LISP which has a set of explicit syntactical rules and a small set of basic vocabulary. The user then may extend the vocabulary by declarative statements and communicate with the machine in the extended vocabulary. Due to frequent message exchanges between the man and the machine during on-line computing, the machine representation of users' programs must be easy to modify at the source language level. The technological trend towards large random access memory suggests the retention of several users' programs in core simultaneously, hence mutual memory protection must be ensured.

This paper describes a scheme of structuring the users' ALGOL programs in accordance with the syntactical unit of a *statement*. The scheme enables the user to make modifications to his source language program at the statement level without recompiling the complete program. The same structure is used to provide the logic sequence of executing statements and to ensure memory protection among users. The next section describes the operating environment of on-line computing which justifies the scheme presented in this paper. The following section reviews the recursive definition of a *statement* in ALGOL as a syntactical unit which is used as the unit of communication from the user to the machine as well as the building block of the program structure in the machine. The next to the last section

*The study is partly supported by National Science Foundation Grant GP4264.

describes the statement-oriented program structure, and the final section shows the role played by the program structure for multiprogram time-shared on-line computations.

## THE ENVIRONMENT IN ON-LINE COMPUTATIONS

There are two modes in on-line computation: constructing the program, and executing the program. Since the programmer is constructing the program on-line piece by piece, it is desirable to specify a minimum number of rules—either things the programmer is not allowed to do, or actions the programmer may take. In either case the programmer will not be burdened with remembering many rules. When executing the program on-line, the user must be able to exercise controls to start and stop the computation at will. The construction of a machine code program on an operator's console imposes a very simple rule on program modification, namely that any single instruction may be independently changed. The execution of a program on an operator's console provides complete control at the machine instruction level, namely that the program may be started or stopped at any specific instruction, or that the program may be stepped through. However, the direct use of an operator's console for on-line computing was discarded on account of the weakness in using a machine code language for program construction and the wastefulness of computer time due to human intervention. The introduction of high-level programming languages and batch operation eliminated the above shortcomings and at the same time ruled out the features of on-line computation.

From the above analysis, we can say that an acceptable on-line computing system must offer each user an input/output device. From this device, he may construct his program piece by piece in a high-level programming language, in which a statement is the building block. When executing the program, he may control the sequence by starting, stopping or stepping through his program at the statement level. For economy, the system must be time-shared among several users to minimize system idle time. From a user's point of view, he enjoys the advantages of an operator's console and a high-level programming language. In this environment, the following are taken as the design specifications:

1. The programming language must be easy to learn and powerful in expressing algorithms. The syntax of the language must allow easy extension to cope with applications such as symbol manipulations.[1] ALGOL 60[2] is considered as a promising language.
2. The source language program should not be completely compiled into a single machine code program such that local changes in source program only require local modifications to its machine representation. An incremental compiler is required.
3. The communication between the user and the machine should be machine independent. For example, the user may ask for the values of variables by specifying their symbolic names, rather than the actual locations in memory.
4. Several users on-line should time share the processor, and all users' programs and data should be retained in core whenever possible, to minimize swapping.
5. A statement is taken as the basic unit of processing such that the user may start or stop his program at specified statements or may execute his program one statement at a time in a "step" mode.

## A STATEMENT IN ALGOL

Since the publication of the "Revised Report on the Algorithmic Language ALGOL 60,"[2] suggestions[3] have been made to generalize it. The following generalization of the definition of a statement is introduced here to give a simpler syntax and render it more suitable for on-line computations.

<program> :: = <statement>
<statement> :: = <unlabeled statement> | <label> : <statement>
<unlabeled statement> :: = <block> | <declarative statement> | <assignment statement> | <conditional statement> | <for statement> | <go to statement> | empty | <line number>
<block> :: = begin <list of statements> end
<list of statements> :: = <statement> | <list of statements>; <list of statements> | (<lower limit>, <upper limit>)
<declarative statement> :: = <declaration>
<conditional statement> :: = if <boolean expression> then <statement> else <statement>

The inclusion of <line number> as unlabeled statement and (<lower limit>, <upper limit>) as

a list of statements is to facilitate on-line program manipulation. <line number> denotes a statement already compiled and assigned a unique line number for identification by the compiler. (<lower limit>, <upper limit>) denotes the list of already compiled statements bounded by the two limits which are identification line numbers. The section on Incremental Statement Compilation gives some examples of using line numbers to modify and reshuffle statements in a program.

All other undefined metalinguistic variables in the above have the same definitions as those given in the official report.[2]

The important differences between the above definition of a statement and that in reference 2 are the following: (1) No distinction is made between a compound statement and a block. (2) All declarations are treated as declarative statements and are allowed to appear anywhere in the block, not necessarily at the beginning. It is conceivable that in on-line computing, the user will benefit from such freedom in making declaration in the middle of a block. (3) Only one type of conditional statement is used: the (if then) is a redundant form of the (if then else) in which the statement that follows else is empty. The statement that follows then is no longer restricted to unconditional statement.

The above definition of a statement will be used as the smallest building block during the construction of a program on-line. The user may insert, add, delete or reshuffle any number of *statements* at a time, and the program structure is designed so that only local modifications are invoked.

## THE STATEMENT-ORIENTED PROGRAM STRUCTURE

By using a statement as the building block of a program, we can structure the ALGOL program internally in the computer memory as a collection of multiply linked elements, one for each statement. An element has a set of structure parameters in the form of pointers that link the element to other elements so that the program-imposed ordering among statements is preserved. If the element represents a statement which can be decomposed into other statements such as a block, a conditional statement, a for-statement or a procedure declaration, the structure representing its component-statements are linked to the element by pointers in it. Hence the number of pointers in an element depends on the type of the statement it represents.

An element in the program structure is composed from the following quantities:

1. A type indicator to specify the type of statement it represents.
2. A set of structure pointers to connect it into the program structure.
3. The compiled machine code for the statement: All references from one statement to others are made through the system interpretation on the program structure.
4. The source statement image: The statement image is retained to provide positive documentary features.
5. An identification for referencing: This system edited identification takes the form of a line number such that the user may refer to any specific statement in making a program change or in initiating execution.
6. Internal editing numbers: All quantities in an element are packed into a contiguous block of words. The size of the block depends on the statement. Two integers are used to specify the size of the block and the field that contains the statement image.

All quantities in an element are completely relocatable except the set of structure pointers. The blocks of words, one block for each element, are dynamically allocated in the memory. Words are taken from an available storage block of unused words to form new elements. The available storage block is shared by all users. It can be reclaimed when exhausted by a scheme similar to that of the garbage collector in LISP.[4] In reclaiming the available storage block, all blocks of words being linked in the program structure are relocated with appropriate changes of structure pointers in the element so that a single available block of consecutive words can be reclaimed.

A simple example is given in Fig. 1 to illustrate the use of structure pointers as shown in Fig. 2.

In Figs. 1 and 2 we use *S* to denote a nondeclarative statement, B to denote a block, D to denote a declaration, BEX to denote a Boolean expression and FCL to denote a for-clause. The complete program can be treated as a single statement in the form of a block B enclosed by the statement brackets: begin in line 1. and end in line 1.*. This block consists of a list of 4 statements $S_1$, $D_2$, $B_3$, $S_4$ where $B_3$

is a block between line 4. and 4.*, and $D_2$ is a declaration.

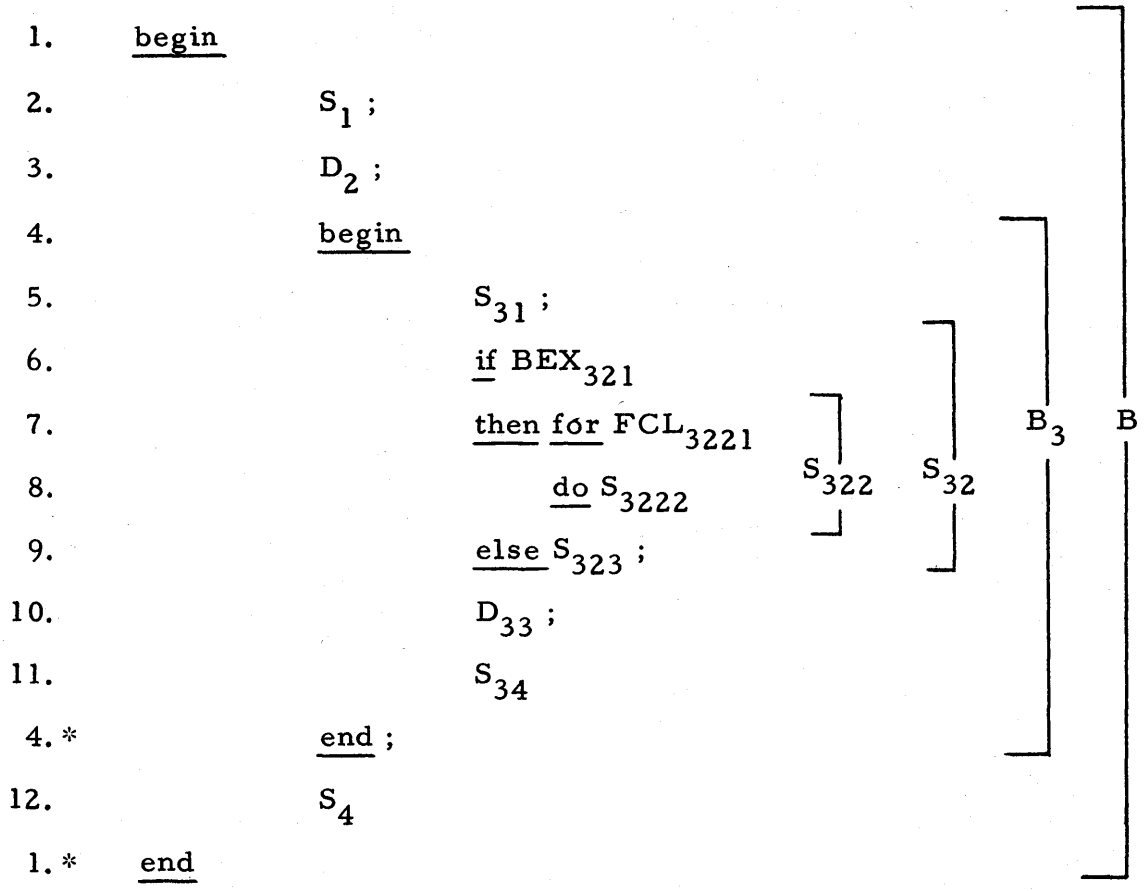



Figure 1. A simple program in which the following abbreviations are used: (1) B, S, D, BEX, and FCL represent respectively the syntactical units of block, statement, declaration, boolean expression, and for-clause; (2) Xijk is a component in the syntactical unit Yij.

d points to the list of declarative statements in B;
t points to the list of nondeclarative statements that are referred to in the original ALGOL report[2] as the tail;
h points to the element representing the block head of which B is a subblock — for this example h is empty;
f points to the element representing the statement that logically follows B in the same block.

The element representing the statement $S_1$ has pointers $f_1$, $h_1$.

The element representing the conditional statement $S_{32}$ between line 6. and 9. has pointers $f_{32}$, $h_{32}$, $ts_{32}$, $fs_{32}$ where $ts_{32}$ points to the element representing the statement to be executed if $BEX_{321}$ is true. $fs_{32}$ points to the element representing the statement to be executed if $BEX_{321}$ is false.

The element representing the for-statement $S_{322}$ has pointers $f_{322}$, $h_{322}$ and $ds_{322}$ where $ds_{322}$ points to the element representing the statement that follows do.

The f pointer of the last statement in a block is a block return BR. The f pointer in the statement within a conditional statement points to if-return IR. The f pointer in the statement that follows do in a for-statement is a for-statement return, FR. The f pointer in the last element of a list of declarative statements is a declaration return, DR. The f pointer in the element representing a procedure body is a procedure body return, PBR. Using the above set of pointers, the structure of the program in Fig. 1 can be constructed as shown in Fig. 2.

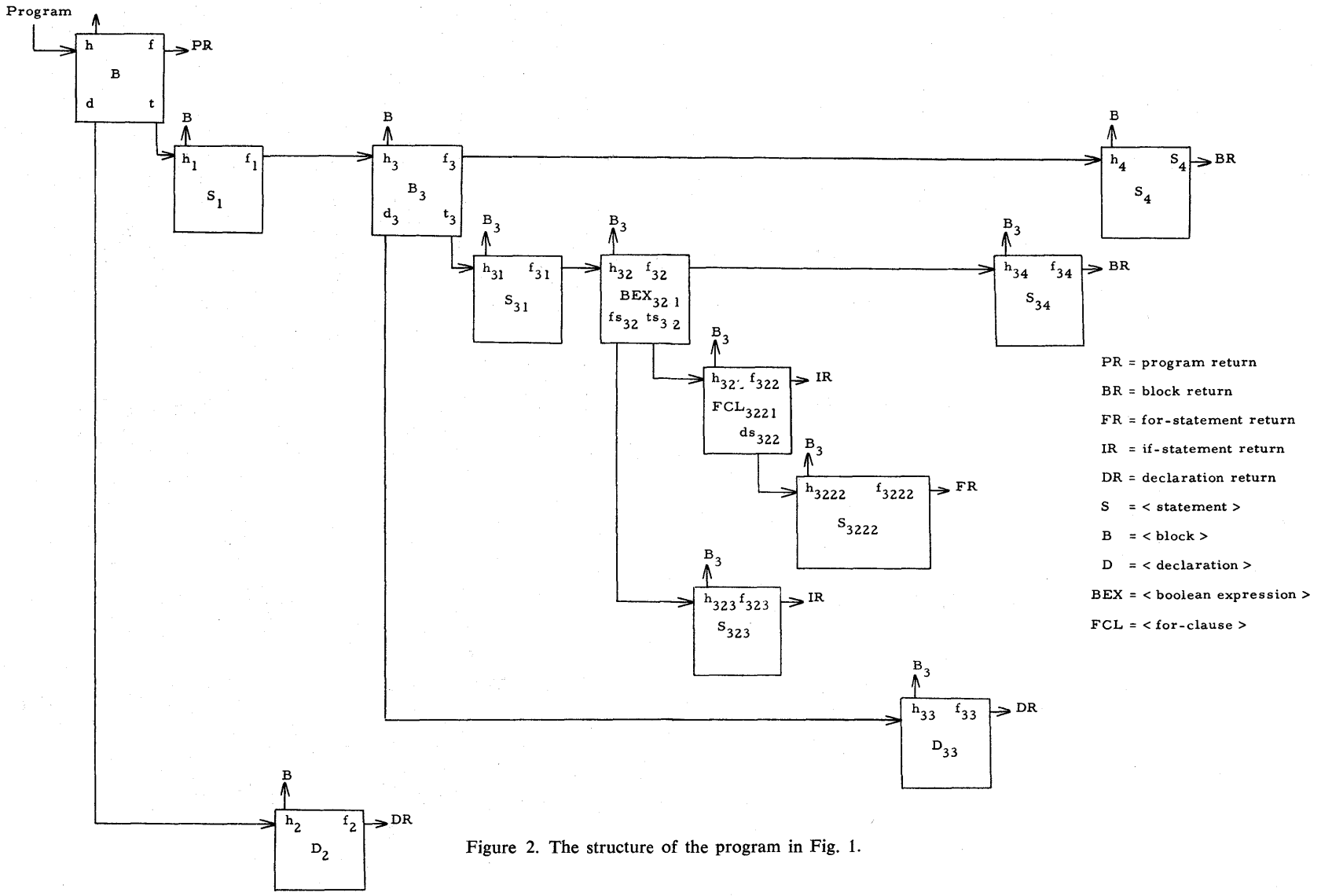



Figure 2. The structure of the program in Fig. 1.

Figure 3 shows an element in the program structure. It is a block of $\gamma$ words. The various quantities in an element are listed as follows:

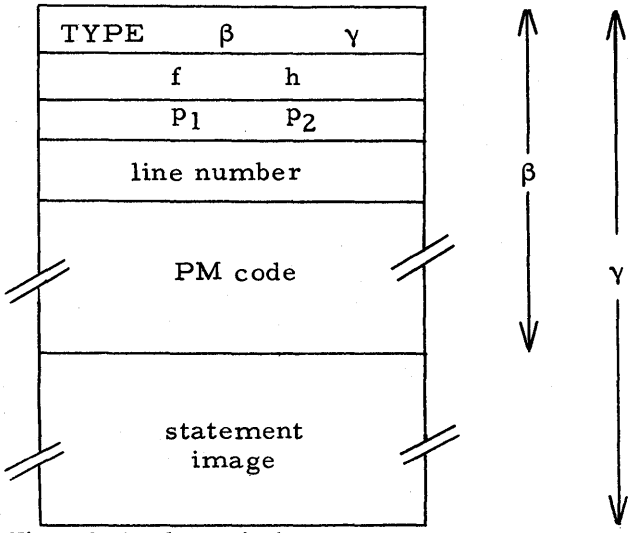



Figure 3. An element in the program structure.

TYPE : a type indicator from block, declaration, procedure, if-statement, for-statement, others.

$\beta$ : an integer specifying the location of the first word in statement image field relative to the first word in the element.

$\gamma$ : an integer that specifies the number of words in the element.

f : a pointer to the element representing the next statement.

h : a pointer to the element representing the block in which the given element is a statement.

$p_1$, $p_2$ : pointers depending on the TYPE according to the following table:

| TYPE | $p_1$ *points to* | $p_2$ *points to* |
|---|---|---|
| block | list of declaration | list of statements |
| declaration | not used | not used |
| procedure | not used | procedure body |
| if-statement | statement after then | statement after else |
| for-statement | not used | statement after do |
| others | not used | not used |

line number : a unique number for each element used for reference by the user.

PM code : Pseudo-machine code for the compiled statement. The PM code differs from the absolute machine code in the following ways:
(1) All references to identifiers are indirectly addressed through non-relocatable entries in the user's symbol table.
(2) The last instruction in a PM code block always returns control to the execution monitor in the system which selects the next element in the program structure for execution.

statement image : The source statement image is retained in its symbolic form.

## THE ROLE OF PROGRAM STRUCTURE IN MULTIPROGRAM TIME-SHARED ON-LINE COMPUTATIONS

In this section, we will describe how the statement-oriented program structure in the last section can be used in multiprogram, time-shared on-line computations. The first part of this section describes the list-structure-like operations on the program structure during the program statement input and modifications. The second part shows the use of program structure during execution in keeping track of the next statement to be executed. In the final part the dynamic nature of the program structure is demonstrated to be extremely desirable in applications that involve frequent man-machine interactions and dynamic data structure.

### *Incremental Statement Compilation and the Statement-Oriented Program Structure*

Conventional compilers translate the source language programs into relocatable codes, and the loader converts them into absolute code. This scheme usually produces an efficient object code; however, the complete process has to be repeated if any changes, however small, are made in the source language program. An incremental compiler is characterized by its ability to compile each statement independently, so that any local change in a statement calls only for recompilation of the statement, not the complete program. When the compiled program is structured as in the preceding section, statement insertions, deletions or modifications are han-

dled by adding, removing or replacing some elements in the program structure with appropriate changes in structure pointers. The dependencies between any two statements lie only in the common set of identifiers that appear in them and their relative location within a program. The latter is encoded into the set of structure pointers in the program structure. The identifier dependency among statements is made indirect through reference entries in the symbol table. Only one reference entry is used for each distinct identifier such that all statements can be independently compiled into PM codes. The contents in the reference entries are set dynamically during execution according to the declaration on the identifiers. Figure 4 shows the indirect dependence among statements through the symbol table and the program structure.

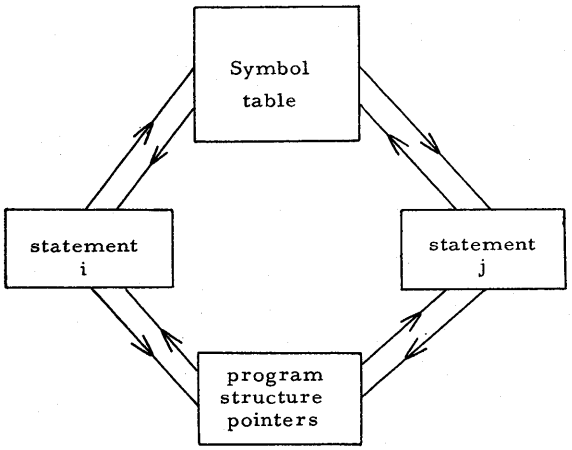



Figure 4. Indirect dependence among statements through symbol table and program structure.

When the program is incrementally constructed on-line, some building code must be specified. With our statement-oriented program structure and the definition of a statement given earlier, the rule becomes very simple:

> Any integral number of *statements* in the program structure, called "out-statements," can be replaced by any integral number of newly specified *statements,* called "in-statements."

Figure 5 shows several examples that represent integral number of statements and also some examples that do not represent integral number of statements.

Replacing no out-statements amounts to inserting in-statements. Specifying no in-statements amounts to deleting the out-statements. Some method of specifying the out-statements in the program for replacement must be provided. One way is probably to display the source program on a CRT and let the user mark, by light pen, the limits that enclose the out-statements. Another method is to associate each element in the program structure with an identification line when the statement for that element is compiled and connected into the structure. The user may subsequently refer to any element in the structure by its line number. The out-statements can be specified by a pair of line numbers ($l_1$, $l_2$) which represent the first and the last of the out-statements, or an insertion point in the program structure when out-statements are empty. For ease of cross referencing, successive statements are assigned line numbers in an increasing order. All statements inserted between the statements numbered n. and n + 1. are numbered into sub-levels n. 1., n. 2., etc. Syntactically the out-statements can be defined as follows:

<out-statements> :: = <insertion point> | (<lower limit>, <upper limit>)
<insertion point> :: = <line number> + | <line number> –
<lower limit> :: = <upper limit> :: = <line number>
<line number> :: = <unsigned integer>. | <line number> <line number>

*Examples:*

(1., 2.4.)
1.2. +
2.3.4. –

*Semantics:*

(<lower limit>, <upper limit>) denotes the set of statements enclosed by <lower limit> and <upper limit> inclusively, e.g., (2., 3.) in Fig. 6.

<line number> + specifies the point in program structure that follows the f pointer in the element identified by <line number>, e.g., 2. + and 3. + in Fig. 6.

<line number> – specifies the point in the program structure that precedes the element identified by <line number>, e.g., 3. –, 4. – and 5. – in Fig. 6.

The in-statements that replace the out-statements can be syntactically defined as follows:

<in-statement> :: = <statement> | <line number> (<lower limit>, <upper limit>) | <in-statement>; <in-statement>

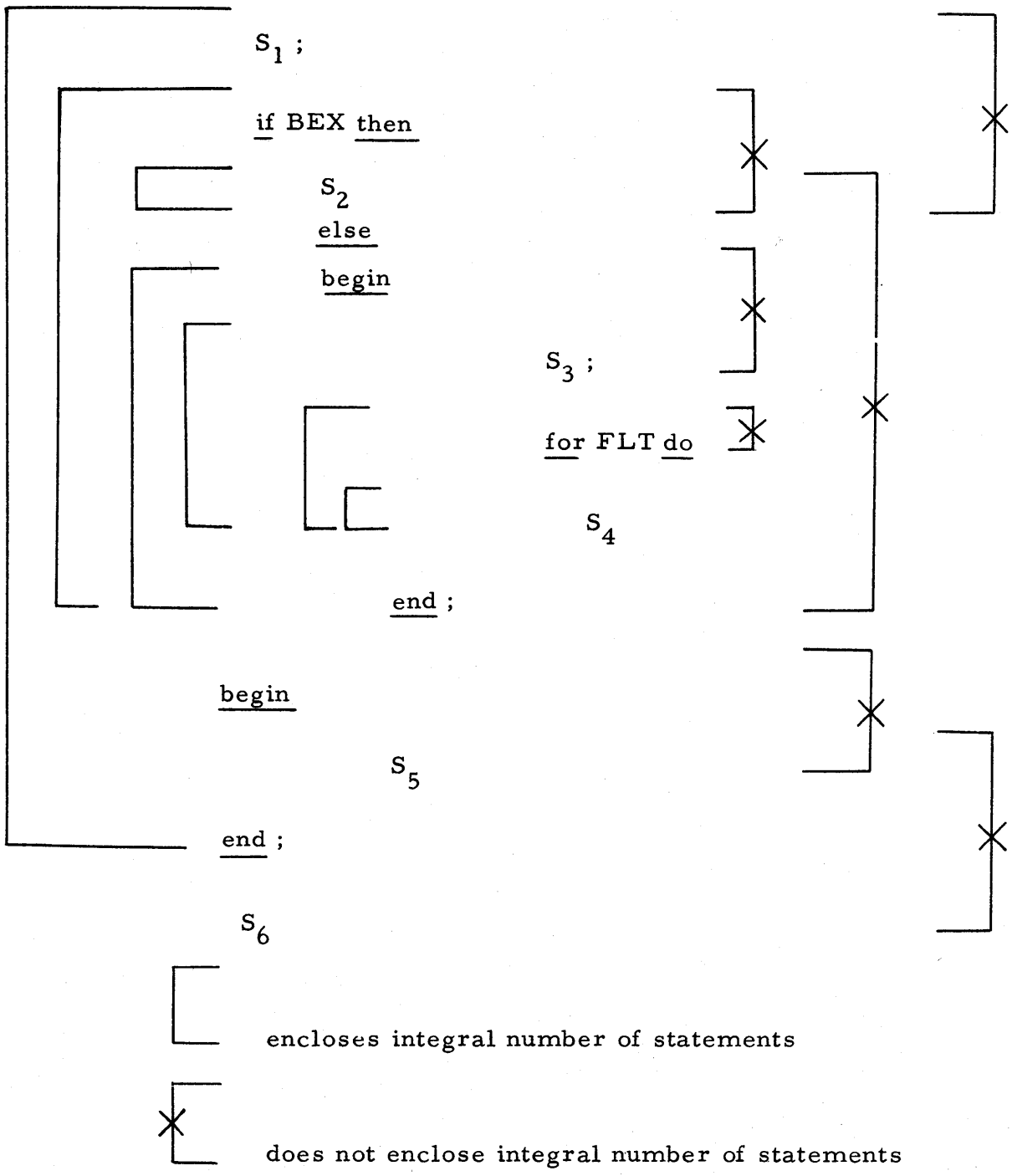



Figure 5. Examples of specifying integral number of statements.

*Example:*

A: = B + C; (1., 5.); begin C: = 0 end; 7. 1.

*Semantics*

<statement> can be any ALGOL statement as defined above (A Statement in ALGOL).

<line number> denotes the statement already in the program structure identified by <line number>.

(<lower limit>, <upper limit>) denotes the

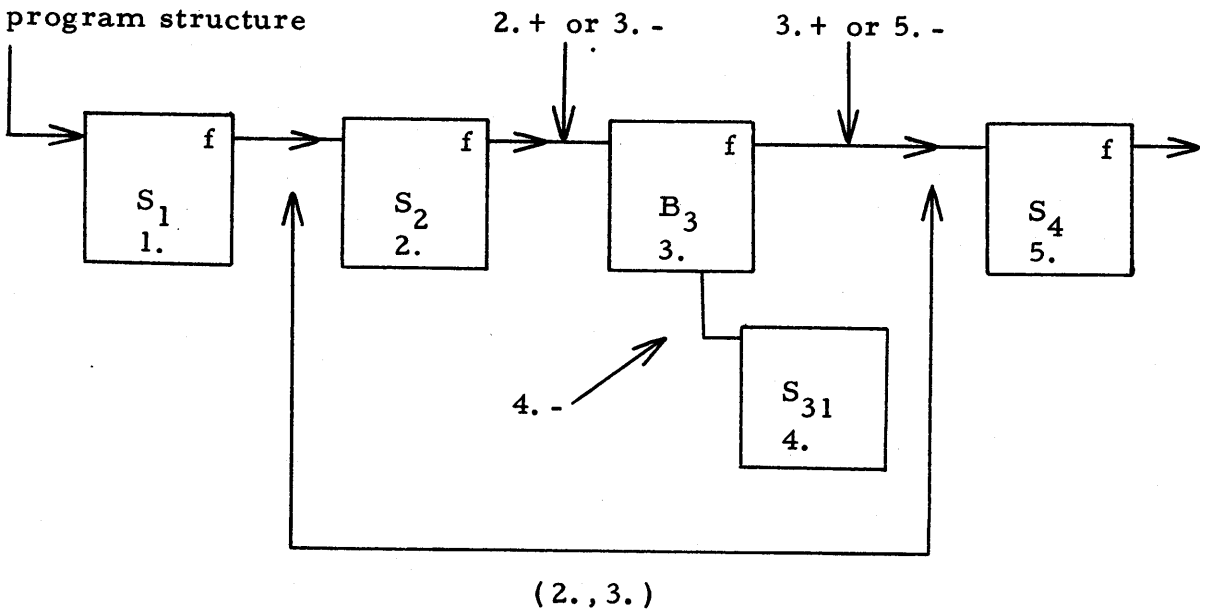



Figure 6. Illustration of insertion point and (lower limit, upper limit) in specifying an integral number of out-statements during program modifications.

list of statements in the program structure that are inclusively enclosed by <lower limit> and <upper limit>.

When line numbers are used in forming an in-statement, they represent the statements already in the program structure. Copies of these elements are incorporated into new locations in the program structure; they are not automatically deleted from their old locations.

A compile command that alters, builds, or manipulates the program structure takes the form:

<compile command>::=compile<out-statement>, <in-statement> EØM

EØM is an action on the input device that will interrupt the machine and cause the monitor in the system to respond to the message.

*Example:*

Let Fig. 7a be some program structure, then the compile command

compile (1., 3.), A:=B; B:=0 EØM

changes the program structure into the form in Fig. 7b.

compile (1., 2.), begin (1., 2.) end EØM

changes it into the form in Fig. 7c which can be transformed to Fig. 7d by

compile 1. +, if BEX then 6. else C:=0 EØM

By using independent statement compilations and the program structure described in the preceding section, the user may manipulate his program quite freely provided that a statement is taken to be the smallest unit for manipulation. Since the user has to be familiar with the definition of a statement in ALGOL before he can express the problem algorithms in the language, the program manipulation rules based on the concept of a statement should become very natural and easy to apply for the user. This is analogous to a user manipulating his machine code program on a console, in which case the smallest unit he may change in his program is a single machine code instruction.

*On-Line Control over Program Execution*

After the source language program is converted into the statement oriented program structure, interactions among statements are made indirectly through the reference entries for identifiers in the symbol table and the set of structure pointers in the program structure. The last instruction in the pseudo-machine code for a statement always returns con-

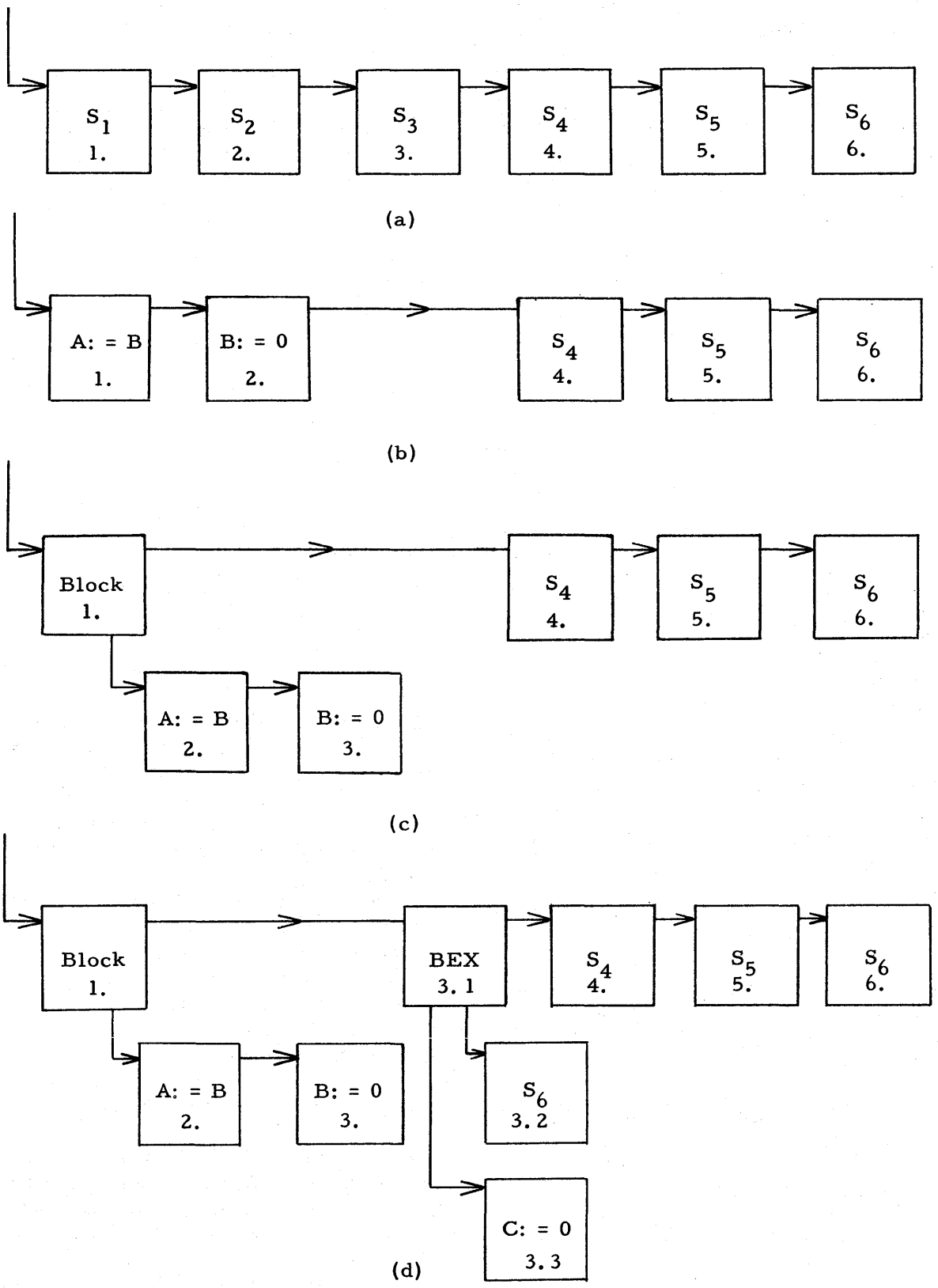

S1
1.
S2
2.
S3
3.
S4
4.
S5
5.
S6
6.
(a)
A: = B
1.
B: = 0
2.
S4
4.
S5
5.
S6
6.
(b)
Block
1.
S4
4.
S5
5.
S6
6.
A: = B
2.
B: = 0
3.
(c)
Block
1.
BEX
3.1
S4
4.
S5
5.
S6
6.
A: = B
2.
B: = 0
3.
S6
3.2
C: = 0
3.3
(d)

Figure 7.

trol to the execution monitor which, from the pointer to the element for the statement just executed, selects the next statement for execution. Due to the recursiveness of a statement in ALGOL, a push down list called ESL for execution status list is maintained for each user. The top element in ESL points to the current statement being executed, the element next below in ESL points to the statement of which the current statement is a component. For example, in Fig. 7d, when the statement A : = B is being executed, the top element in ESL points to the element numbered 2. and the element next below in ESL points to the element numbered 1. . Depending on the type of the element, the last instruction in the pseudomachine code returns control to different points in the execution monitor which takes action depending on the user's operation mode.

The user's operation mode is set by an execution command.

*Syntax:*

<execution command> :: = <start> | <step> | <stop>
<start> :: = execute <execution bounds> EØM
<execution bounds> :: = empty | (<starting point>, <stopping point>)
<step> :: = step <starting point> EØM
<stop> :: = EØM
<starting point> :: = <stopping point> :: = empty | <line number>

*Examples:*

execute EØM
execute (1. 1., 3. 5.) EØM
execute (3. 1.,) EØM
execute ( , 4. 5. 6.) EØM
step EØM
step 5. 6. EØM
EØM

*Semantics:*

A user's program can be either in "execute mode" or "step mode." A <start> will set the user into the execute mode. If a nonempty <execution bound> is specified, the program will start from the <starting point> and stop at the <stopping point>. An empty <starting point> implies the top element in ESL, and an empty <stopping point> implies an infinite line number. A <stop> sets the user's program into step mode. If the <execution bounds> is empty, the program will continue from the statement currently being pointed by the top element in ESL and will come to a halt only if <stop> is initiated from the input device or it comes to the previously specified <stopping point> or a program stop. In step mode, execution is halted after each statement. A <step> instructs one statement to be executed. If the <starting point> in <step> is empty, the element in the program structure pointed by the top element in ESL is executed. Otherwise, <starting point> is set to be the top element in ESL with appropriate pop ups and push downs in ESL to maintain the proper block level being referenced, then the element pointed to in ESL is executed. In step mode, the execution of each statement provides the user certain trace information on the on-line output device such as the value of an expression. At this point we can again see the analogy to the control a programmer can exercise on his machine code program in the computer from an operator's console. The <start>, <step> and <stop> commands are analogous to the start-, step- and stop-push buttons. The <execution limit> is analogous to setting the instruction counter which is, in our system, generalized into a push down list ESL. The control unit in a computer that maintains the correct execution order from one machine instruction to the next is conceptually extended in our system into the "execution monitor." However there is the difference that in our system the user communicates in a problem-oriented language.

The execution monitor's operation is described below by using a set of ALGOL-like statements. The following terminology is employed:

ESL [1]: the top element in the push down list ESL.

t(ESL [1]): the t-pointer in the element representing a block pointed by ESL [1]. t points to the structure that represents all the nondeclarative statements in the block.

f(ESL [1]): the f-pointer in the element representing a statement pointed by ESL [1]. f points to the next statement, namely the statement that follows the statement separator.

ts(ESL [1]): the ts-pointer in the element representing a conditional statement pointed by ESL [1]. ts points to the statement that follows then.

fs(ESL [1]): the fs-pointer in the element representing a conditional statement pointed by ESL [1]. fs points to the statement that follows else.

ds(ESL [1]): the ds-pointer in the element representing a for-statement pointed by ESL [1]. ds points to the statement that follows do.

push down A into B: all elements in the push down list B are pushed down one level and the quantity A becomes the top element in B, i.e., B [i]:=B [i-1] for i≥2 and B [1]:=A.

pop up B: all elements in the push down list B are popped up one level, i.e. B [i]: = B [i+] for i≥1.

The original top element in B is lost.

return control to the user: the user's program is halted and the system is ready to receive a message from the input device.

return control to PM(ESL [1]): go to execute the pseudomachine code compiled for the statement which is represented in the program structure as an element pointed by ESL [1].

output trace information: when in the step mode, the execution of each statement provides information on the execution result such as the value of an evaluated expression, and displays the next statement to be executed upon receiving step EØM.

initiate block entry procedure: save all current machine addresses for the identifiers declared in this block and load their new local machine addresses into their reference entries. If the block is entered recursively, savings are implemented into push down lists.

initiate block exit procedure: restore the machine addresses for the identifiers declared in this block to their values in the outer block which for recursively entered blocks were the top elements in their push down lists.

set up ESL in accordance with the designational expression: transfer out of a block is allowed in which case all the top elements in ESL will be popped up until the pointer to the block in which the designated statement is a component appears as the top element in ESL, then the designated statement is pushed in ESL. Each time a pointer to the block is popped up from ESL, the block exit procedure is initiated.

set up actual parameters: save all current machine addresses for the identifiers used as formal parameters in the procedure, into push down lists when it is recursively called, and load the machine addresses of words containing the actual parameters into the reference entries of these formal parameters.

initiate procedure exit: restore the machine addresses for the identifiers used as formal parameters in this procedure to their values in the block that initiated this procedure call.

The following ALGOL-like statements describe the execution which offers the user extensive controls over his program execution on line.

```
return from go-to statement:
  if in step mode
  then output trace information
  else;
  set up ESL in accordance with the designational
       expression;
  go to execute next statement;
return from block:
  if in step mode
  then output trace information
  else;
  initiate block entry procedure;
  push down t(ESL [1]) into ESL;
  go to execute next statement;
return from if-statement:
  if in step mode
  then output trace information
  else;
  if the Boolean expression is true
  then push down ts(ESL [1]) into ESL
  else push down fs(ESL [1] into ESL;
  go to execute next statement;
return from for-statement:
  if in step mode
  then output trace information
  else;
  if all elements in the for list are serviced
  then ESL [1]:=f(ESL [1])
  else push down ds(ESL [1]) into ESL;
  go to execute next statement;
return from procedure call:
  if in step mode
  then output trace information
  else;
  set up the actual parameters;
  push down pointer to the procedure body into
       ESL;
  go to execute next statement;
return from all other statements:
  if in step mode
  then output trace information
  else;
  ESL [1]:=f(ESL [1]);
```

```
    go to execute next statement;
execute next statement:
    if in execution mode and ESL [1] not equal to
        to <stopping point>
    then go to continue
    else begin enter step mode;
        return control to the user
        end;
continue:
    if ESL [1] is a program return PR
      then begin enter step mode; return control to
                the user
           end
      else
    if ESL [1] is a block return BR
      then begin initiate block exit procedure;
                pop up ESL;
                ESL [1]:=f(ESL [1]);
                go to execute next statement
           end
      else
    if ESL [1] is an if return IR
      then begin pop up ESL;
                ESL [1]:=f(ESL [1]);
                go to exceute next statement
           end
      else
    if ESL [1] is a for return FR
      then begin pop up ESL;
                return control to PM(ESL [1])
           end
      else
    if ESL [1] is a procedure body return PBR
      then begin initiate procedure exit;
                pop up ESL;
                return control to PM(ESL [1])
           end
      else
    ESL [1] is a pointer to an element in the pro-
    gram structure:
      return control to PM(ESL [1])
```

*The Statement-Oriented Program Structure Used in Time-Shared Multiprogramming and Its Compatibility with Dynamic Data Structures*

The dynamic nature of on-line computing calls for a dynamic data structure as well as dynamic program structure. Nonnumerical applications such as analytical expression manipulations on computers[5] will increase in efficiency and effectiveness if they can be performed on-line. ALGOL can be easily extended to manipulate list-structure-like data.[1] The use of a dynamic program structure is completely compatible with dynamic data structure. The same dynamic memory allocator will service all users' programs and data structures.

Figure 8 shows the configuration for the multiprogram time-shared system. Each user's activity in the system is represented by an I/O device, its program structure, symbol table, data structure and operation status, all properly linked under the user's pointer. Since storage allocations for program and data structures and the execution of their programs are all under the control of the multiprogram time-shared system, memory protection against each other is assured. The system consists of an incremental compiler, an execution monitor, an available storage block manager and a system monitor that coordinates various phases of operations. Fig. 9 shows the organization that incorporates the self-optimization technique of adapting a set of monitor system parameters in accordance with the operation environment. Such system parameters may, for instance, cause the monitor to operate in one of several possible modes. In multiprogram time-shared on-line computations, there is always the question of whether all users' programs and data should be retained in core, or should only one be in core with swap between users. Our solution is to let the operation environment dictate the mode: if all users' programs and data can be comfortably accommodated in core, they will all remain in core; otherwise they will be divided into groups and swap among groups. The actual rules used for adapting the monitor parameters are still subject to experimentation. Since the multiprogram time-shared system should be in core all the time, it should be constructed so that read-only memory can be used to store them.

## CONCLUSION

A multiprogram time-shared system based on the concept presented in this report has been under implementation as an experimental project at the California Institute of Technology. Invariably many of its details have been modified to suit the particular hardware which consists of an IBM 7040 computer, a 7288 multiplexor and several Institute-developed typewriter consoles.

In conclusion, we believe that the use of incremental compilation, system-controlled execution, dynamically structured programs and data can offer the users the power of programming in a high-level algorithmic language and the advantage of interacting with the machine by means of an on-line console. The time-sharing mode further makes such operation economically acceptable.

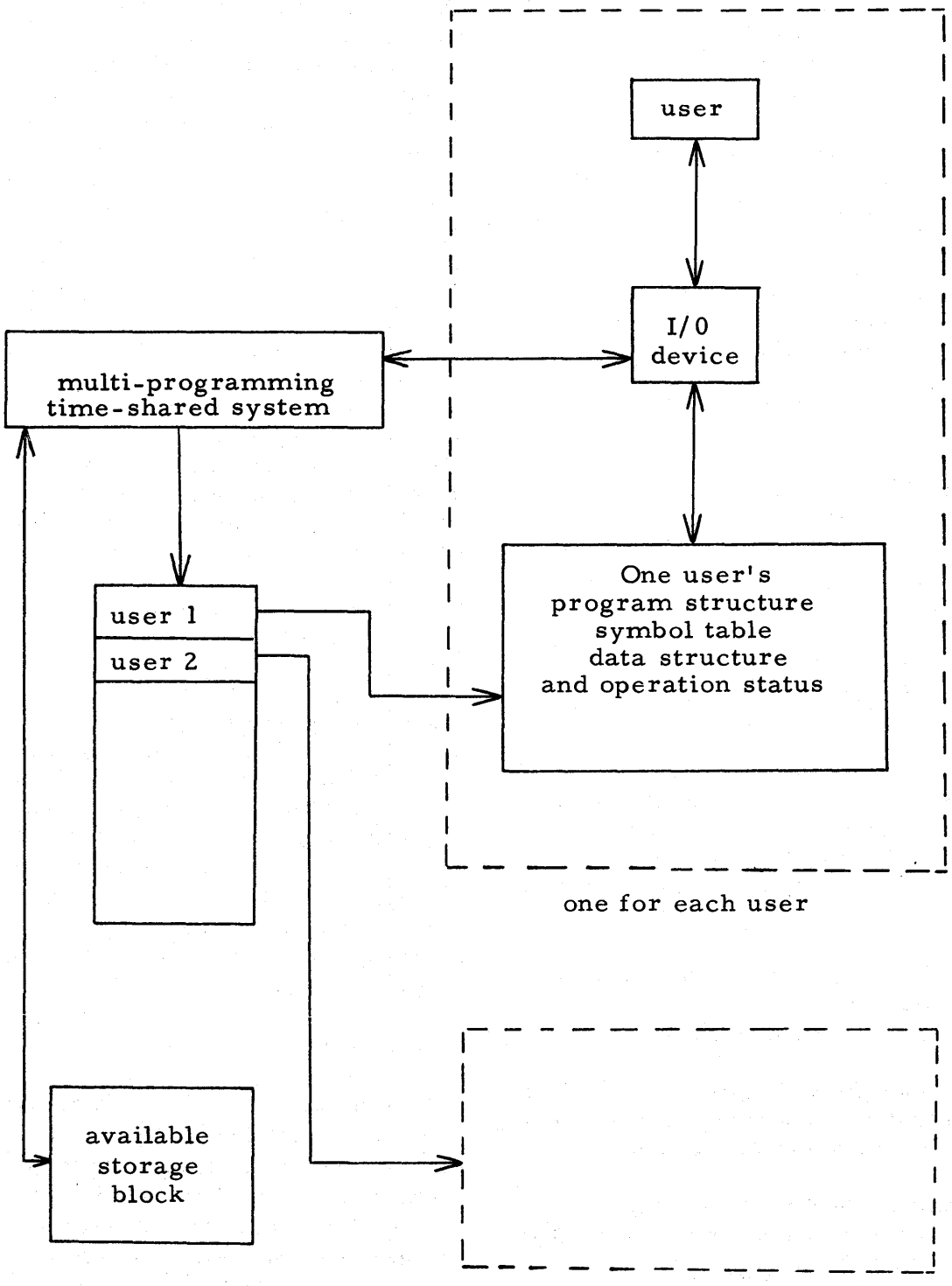



Figure 8. The multiprogram time-shared system configuration.

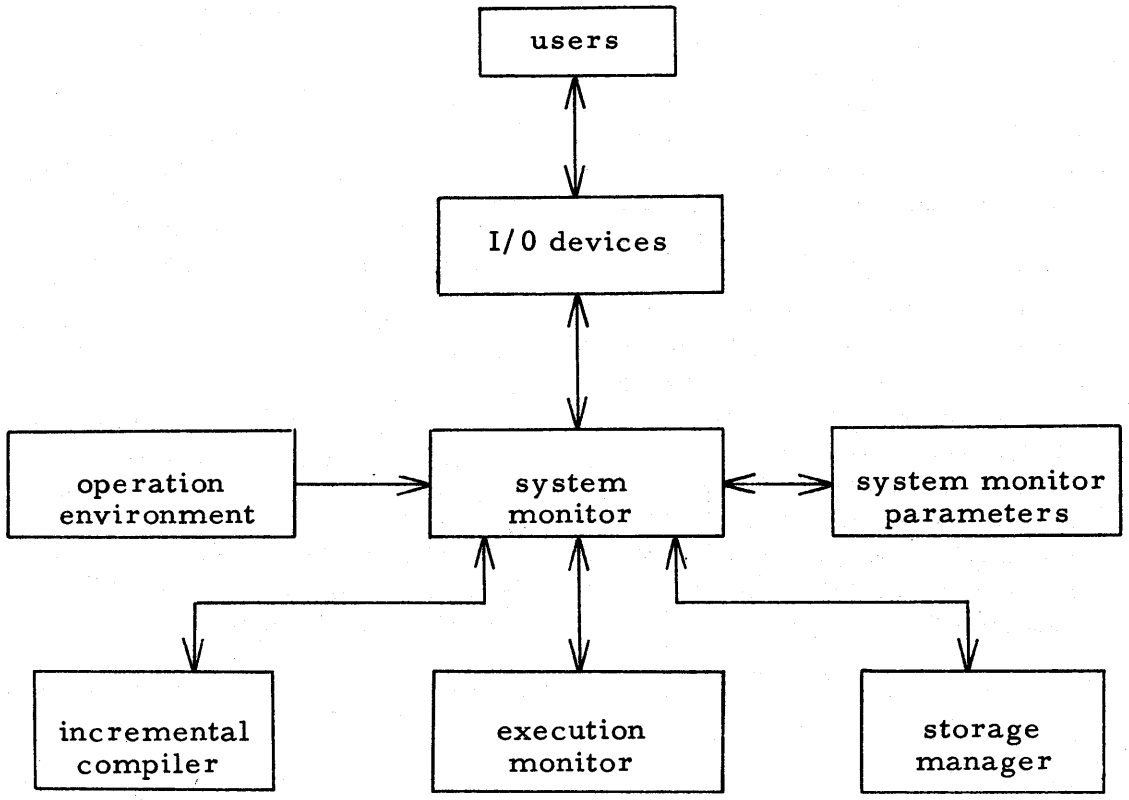



Figure 9. Organization of an adaptive system monitor for multiprogram time-shared on-line computations.

The differences of this system from other similar systems[6] are the following:

1. Statements in our system are compiled incrementally into directly executable codes. System interpretation is called for only between statements.
2. Several users may be accommodated simultaneously in core memory.
3. Easily extendable to cope with applications that call for dynamic data structure such as algebraic expression manipulation.

The study reported in this paper also reflects study of the computer organization for on-line time-sharing applications. Some applications are given below.:

1. The incremental compilation achieved by indirectly addressing all operands through their reference entries suggests a small very-high-speed memory, functioning much like the index registers, to be used by all identifiers' reference entries.
2. The dynamic nature of multiprogram on-line computation should have a strong influence on memory organization. The algorithms trying to maximize the utilization of computer memory without sacrificing computing speed and programming flexibility should be investigated for possible direct incorporation into hardware configurations. For the same reason that arithmetic unit is used to perform arithmetics and data channels for input and output, special processors should be designed to allocate and relocate users' areas in memories possibly in parallel with the main computation.
3. The central control unit in a computer used for multiprogramming should be re-

sponsible for scheduling various programs to various processors. The organization of the central control unit must also reflect the nature of man-machine interactions and the types of control statements in the programming language.

4. The encoding of information, numeric or symbolic, into computer words should include type indication such that, for example, arithmetic operations performed on nonnumeric quantities can be detected as errors. This redundancy in information representation can be used to provide some error check during execution as well as to provide a simpler machine instruction set. For example, the same arithmetic instruction can be used for both floating point and fixed point numbers if the number representation suggests its type and whose indication is decoded accordingly in the arithmetic unit.

## REFERENCES

1. A. J. Perlis and R. Iturriaga, "An Extension to ALGOL for Manipulating Formulae," *Communications of the ACM,* vol. 7, pp. 127-130 (Feb. 1964).

2. P. Naur et al, "Revised Report on the Algorithmic Language ALGOL 60," *Communications of the ACM,* vol. 6, pp. 1-17 (Jan. 1963).

3. N. Wirth, "A Generalization of ALGOL," *Communications of the ACM,* vol. 6, pp. 547-554 (Sept. 1963).

4. J. McCarthy et al, *LISP 1.5 Programmer's Manual,* Massachusetts Institute of Technology Press, 1961.

5. J. E. Sammet and E. R. Bond, "Introduction to FORMAC," *IEEE Transactions on Electronic Computers,* vol. EC-13, pp. 386-394 (Aug. 1964).

6. "IBM 7040 / 7044 Remote Computing System," IBM System Reference Library file no. 7040-25, form C28-6800-0.

# INTERACTIVE MACHINE LANGUAGE PROGRAMMING

Butler W. Lampson
*University of California*
*Berkeley, California*

## INTRODUCTION

The problems of machine language programming, in the broad sense of coding in which it is possible to write each instruction out explicitly, have been curiously neglected in the literature.[1,2] Granted that less than half of the binary instructions generated in the past year had their origin in assembly language, it remains likely that much more than half the instructions executed originated in this way. There are still many problems which must be coded in the hardware language of the computer on which they are to run, either because of stringent time and space requirements or because no suitable higher level language is available.

It is a sad fact, however, that a large number of these problems never run at all because of the inordinate amount of effort required to write and debug machine language programs. On those that are undertaken in spite of this obstacle, a great deal of time is wasted in struggles between programmer and computer which might be avoided if the proper systems were available. Some of the necessary components of these systems, both hardware and software, have been developed and intensively used at a few installations. To most programmers, however, they remain as unfamiliar as other tools, which are presented for the first time below.

In the former category fall the most important features of a good assembler; macro instructions implemented by character substitution, conditional assembly instructions, and reasonably free linking of independently assembled programs. The basic components of a debugging system are also known but are relatively unfamiliar.[3] For these the essential prerequisite is an *interactive* environment, in which the power of the computer is available at a console for long periods of time. The batch processing mode in which large systems are operated today of course precludes interaction, but programs for small machines are normally debugged in this way, and as time-sharing becomes more widespread the interactive environment will become common.

It is clear that interactive debugging systems must have abilities very different from those of off-line systems. Large volumes of output are intolerable, so that dumps and traces are to be avoided at all costs. To take the place of dumps, selective examination and alteration of memory locations is provided. Traces give way to breakpoints, which cause control to return to the system at selected instructions. It is also essential to escape from the

*The work described in this paper was supported by the Advanced Research Projects Agency of the Department of Defense under contract SD-185.

switches-and-lights console debugging common on small machines without adequate software. To this end, type-in and type-out of information must be symbolic rather than octal where this is convenient. The goal, which can be very nearly achieved, is to make the symbolic representation of an instruction produced by the system identical to the original symbolic written by the user. The emphasis is on convenience to the user and rapidity of communication.

The combination of an assembler and a debugger of this kind is a powerful one, which can reduce by a factor of perhaps five the time required to write and debug a machine language program. A full system for interactive machine language programming (IMP), however, can do much more and, if properly designed, need not be more difficult to implement. The basic ideas behind this system are these:

1. Complete integration of the assembler and the debugging system, so that all input goes through the same processor. Much redundant coding is thus eliminated, together with one of two different languages serving the same purpose: to specify instructions in symbolic form. This concept requires that code be assembled directly into core (or into a core image on secondary storage). Relocatable output and relocatable loaders are thereby done away with.

   *A remark on terminology:* It will be convenient in the sequel to speak of the "assembler" and the "debugger" in the IMP system. These terms should be understood in the light of the foregoing: different parts of the same language are being referred to, rather than distinct languages.
2. Commands for editing the symbolic source program. The edit commands simultaneously modify the binary program in core and the symbolic on secondary storage. Corrections made during debugging are thus automatically incorporated into the symbolic, and the labor of keeping the latter current is almost eliminated.
3. A powerful string-handling capability in the assembler which makes it quite easy to write macros for compiling algebraic expressions, to take popular example which can be handled in a few other systems, but rather clumsily. The point is not that one wants to write such macros, but that in particular applications one may want macros of a similar degree of complexity.

These matters are discussed in more detail in the following. We consider the assembler first and then the debugger, since the command language of the latter makes heavy use of the assembler's features.

Before beginning the discussion, it may be well to describe briefly the machine on which this system is implemented. It is a Scientific Data Systems 930, a 2-microsecond, single-address computer with indirect addressing and one index register. Our system includes a drum which is large enough to hold for each user all the symbolic for a program being debugged, together with the system, a core image of the program and some tables. Backup storage of at least this size is essential for the editing features of the IMP system. The rest of the system could be implemented after a fashion with tapes.

## THE ASSEMBLER

The input format of the IMP assembler is a rather unusual one. Originated on the TX-0 at MIT, it has been adopted by DEC for most of its machines, but is unknown or unpopular elsewhere in the industry. Although it looks strange at first, it has substantial advantages in terms of simplicity, both for the user and for the system. The latter is a non-negligible consideration, often equally ignored and overemphasized.

The basic idea is that the assembler processes each line of input as an *expression* (unless it is a directive or macro call). The expression is evaluated and put into core at the word addressed by the location counter, and the location counter is advanced by 1. Expressions are made up of operands (which may be symbols, constants — numeric or alphanumeric — and parenthesized subexpressions) and operators. Available operators are: $+,-,\times,/,\wedge,\vee,\sim$ with their usual meaning and precedence; $=,<$, and $>$, which are binary operators with precedence less than $+$, and which yield 0 or 1 depending on whether the indicated relation holds between the operands; and $\neq$, a unary operator with lowest precedence which causes its operand to be taken as a literal — i.e., it is assigned a storage location, which is the same as the location assigned to other literals with the same value, and the address of this location is the value of the literal. Blanks have the following significance:

any string of blanks not at the beginning or end of an expression is taken as a single plus sign.

It is not immediately clear how instructions are conveniently written as expressions, and in fact the scheme used depends on the fact that the object machine is a single-address, word-oriented computer with a reasonable number of modifiers in a single instruction. It would work on the PDP-6, but not on Stretch.

The idea is simple: all operation code mnemonics are predefined symbols with values equal to the octal encodings of the instructions. On the SDS 930, for instance, LDA (load A) is defined as 7600000 (all numbers are in octal). The expression LDA + 200 then evaluates to 7600200. When the convention about spaces is invoked, the expression

LDA 200

evaluates to the same thing, which is just the instruction we expect from this symbolic line in a conventional assembler.

Modifiers are handled in the same spirit. In the 24-bit word of the 930 there is an index bit, which is the second from the left, and an indirect bit, which is the tenth. With the predefined symbols

I = 40000
X = 20000000

the expression LDA I 200 X evaluates to 27640200. In more conventional form it would look like this:

LDA * 200,2

There is little to choose between them for brevity or clarity. Note that the order of the terms in the expression is arbitrary.

The greatest advantages of the uniform use of expressions accrue to the assembler, but the programmer gains a good deal of flexibility. Examples will readily occur to the reader.

Using this convention the implementation of the basic assembler is very simple. Essentially all that is required is an expression analyzer and evaluator, which will not run to more than three or four hundred instructions on any machine. Because all assembly is into core, there is no such thing as relocatability.

Two rather conventional methods are provided for defining symbols. A symbol appearing at the left edge of a line is defined as the current value of the location counter. Such a symbol may not be redefined. In addition, a line such as

SYM←4600

defines SYM. Any earlier definition is simply overridden. The right side may of course be any expression which can be evaluated.

The special symbol . refers to the location counter. It may appear on the left of a ← sign. Thus, the line

```
A       .←.  40
```

is equivalent to

```
A       BSS  40
```

in a conventional assembler.

There remains one point about the basic assembler which is crucially important to the implementation: the treatment of undefined symbols. When an expression is encountered during assembly, there is no guarantee that it can be evaluated, since all the symbols in it may not be defined. This is the reason why most assemblers are two pass; the first pass serves to define the symbols. Becaue the IMP assembler must accept typewriter input, it cannot be two pass and must therefore keep track of undefined expressions explicitly.

There is a general way of doing this, in which the undefined expression, translated for convenience into reverse Polish, is added to a list of such expressions, together with the address of the word it is to occupy. At suitable intervals this list is scanned and all the newly defined expressions are evaluated and inserted in the proper locations. For complex expressions there is no avoiding some such mechanism, and it has the advantage of simplicity. It is, however, wasteful of storage and also of time, since an expression may be examined many times while it is on the list before it can be evaluated. One important special case can be treated much more efficiently, and this is the case of an instruction with an undefined address, which includes at least 90 percent of the occurrences of undefined expressions.

For example, when the assembler sees this code:

```
X    BRU  A    branch unconditional
     LDA  B
A    STA  C
```

the instruction at X has an undefined address which becomes defined when the label A is encountered. This situation can be kept track of by putting in the symbol table entry for A the location of the first word containing A as an address. In the address of this word we put the location of the second such word, and so build a list through all the words containing the undefined symbol A as an address. The list is terminated by filling the address field with ones. When the symbol is defined we simply run

down the chain and fill in the proper value. This scheme will work as long as the address field contains only A, since there is then no other information which must be preserved. Note that no storage is wasted and that when A is defined the correct address can be filled in very quickly.

The description of the basic assembler is now complete, except for a few nonessential details, and we turn to the macro facility. Macros are handled in a standard manner, which the following example should sufficiently illustrate:

```
STORE   MACRO ARG1,ARG2
        IRP TEMP=ARG2    indefinite repeat
        ST'ARG1 TEMP
        ENDR
        ENDM STORE
```

called with

```
        STORE A,(S1,S2,S3)
```

becomes after argument substitution

```
        IRP TEMP=S1,S2,S3
        STA TEMP
        ENDR
```

That is, this string of characters is seen by the assembler as though it were in the symbolic input.

A macro may be defined with more arguments that it is called with, in which case the extra arguments are made either null strings or generated symbols. No more arguments are collected than are called for by the definition. An argument is normally collected literally, character by character, but a colon appearing before a macro name will cause it to be expanded. To provide additional flexibility, two directives called STACK and UNSTACK are provided which respectively suspend the analysis of the current expression and resume it.

Some unusual things may be done with this much machinery. Consider the macro

```
LIT     MACRO  ARG,GEN
        STACK
        TEMP←.
        .←LITERALS
GEN     ARG
        LITERALS←LITERALS 1
        .←TEMP
        UNSTACK
        GEN
        ENDM LIT
```

Called with

```
        LDA LIT 20
```

it will assign a storage location, say LITERALS+10, put 20 in it, and assemble

```
        LDA LITERALS+10
```

There are many other ways of writing this macro using the list features discussed below.

The IRP operation used above is not new, but it is not well known. It causes the lines in its range, which is delimited by a matching ENDR, to be processed repeatedly by the assembler. Each time around the argument, TEMP in this case, is replaced by one of the subarguments, which are the character strings following the = sign and separated by commas. The entire process is rather similar to a macro call, and subarguments are processed according to the rules for macro arguments, except that parentheses are not removed. Thus the IRP generated by the expansion of the macro discussed above in turn expands into

```
STA S1          TEMP replaced by S1
STA S2          TEMP replaced by S2
STA S3          TEMP replaced by S3
```

Two extensions of this device:

```
IRP A,B=A1,B1,A2,B2,A3,B3 $ C=C1,C2
A,B,C
ENDR
```

expands into

```
A1,B1,C1,
A2,B2,C2
A3,B3,C2
```

We illustrate with another macro definition:

```
MOVE    MACRO ARG
        IRP TEMPA,TEMPB=ARG
        IRP TEMPC=:TEMPA $ TEMPD
          =:TEMPB
        LDA TEMPC
        STORE TEMPD
        ENDR
        ENDR
        ENDM
```

Called by

```
        MOVE (A1,(B1,C1),A2,B2)
```

this expands into

```
        LDA A1
        STA B1
        STA B2
        LDA A2
        STA B2
```

Suppose that we have some two-word data structures which we wish to manipulate. We can define each of them as a macro, using another macro to do the definition and reserve storage:

```
TW      MACRO   ARG,GENA,GENB
GENA    0
```

```
GENB    0
ARG     MACRO
        GENA,GENB
        ENDM ARG
        ENDM TW
```

Now, if we call TW:

```
        TW A
        TW B
```

we can then use the newly defined macros A and B in the move macro. In fact

```
        MOVE (A,B)
```

after character substitution both in the macro body and in the first IRP body is

```
IRP TEMPA,TEMPB=A,B
IRP TEMPC=.G0001, .G0002 $ TEMPD
  =.G0003, .G0004
...
```

which expands to

```
        LDA  .G0001
        STA  .G0003
        LDA  .G0002
        STA  .G0004
```

There are two other repeat directives,

```
        RPT expression
        . . .
        ENDR
```

which repeats its scope the number of times specified by the expression, and

```
        CRPT expression
        . . .
        ENDR
```

which repeats its scope, reevaluating the expression each time until it is $\leq 0$.

Finally, there is a conditional directive:

```
        IF expression
        . . .
        ELSF expression
        . . .
        ENDF
```

which causes the lines between the first IF or ELSF whose argument is $>0$ to be assembled and everything else to be ignored.

The implementation of all this is quite straightforward, and very similar for macros and repeats. The body of the macro definition or repeat is collected as a character string, with markers replacing the arguments, and saved away. Each time it is called, the routine which delivers characters to the assembler, which we will call CHAR, is switched from the input medium to the saved string. Arguments for a macro call or IRP are likewise saved as strings. The characters coming from a definition are monitored for the argument marker, and if it is found CHAR is switched again, to the argument string. Whenever any of these strings ends, CHAR is switched back to the string it was working on before.

All this machinery is of course recursive. The only restriction is that macro definitions and repeats must be properly nested. Note that because of the implementation technique just described a macro definition may contain anything, including other definitions. The other definitions of course are not made until the macro is called.

The most novel feature of this assembler is the string or *list*-manipulating features available to the programmer, which allow him to define macros to perform functions normally regarded as the prerogative of a compiler. A list may be assigned to a symbol as its value by

```
SYM←[any string not containing an
    unbalanced right bracket]
```

The string is saved literally as the value of SYM, with one exception: the character : causes the following symbol to be expanded if it is a macro or list name, just as it does in macro arguments. The structure :[string] is equivalent to the string alone.

Once SYM has been equated to a list, any use of it is exactly equivalent to writing the contents of the string, including the brackets. *Exception:* If SYM appears within brackets or as a macro argument and is not preceded by : it is transmitted literally. In most contexts a string enclosed in brackets has the same effect as one not so enclosed.

Thus the sequence

```
SYMA←[A]
SYMB←[B]
SYMC←[:SYMA:SYMB',CD]
```

will leave SYMC with the value [AB,CD]. The ' has the same function as it does in a macro definition.

A symbol equated to a list (or, as always, the explicit list itself) may be *subscripted* in two ways. In the above example, SYMC[2] is equivalent to B (i.e., a subscript in bracket selects a single character). More generally, SYMC[2,5] is equivalent to B,CD.

The other form of subscripting selects a segment of a list delimited by commas: SYMC(1) is the same as AB.

To illustrate the use of these features we consider the following macro to compile an expression

with the operators + and –, single character variables and parenthesization:

```
ARITH   EXPR
        ARG←EXPR
        SB←0
        L←LENGTH(ARG)
        ARITH1
        ENDM ARITH
ARITH1  MACRO OPA,OPB,OP
        OP←0
        OPA←0
        CRPT SB < L 1 V OP=–1
        SB←SB 1
        C←":ARG[SB]"
        IF OP=0
        IF C="("      this branch if operator
                      not yet found
        ARITH1
        OPA←1
        ELSF C="–"
        OP←2
        ELSF C="+"
        OP←1
        ELSF C=")"
        OP←–1
        ELSF 1
        OP←–2
        OPB←[:ARG[SB]]
        ENDF
        ESLF C="("    this branch if operator
                      found
        IF OPA=1      this branch if second
                      operand is ()
        TIDX←TIDX 1
        STA T'NUM (TIDX)
        OPB← [:T'NUM(TIDX)]
        ARITH1
        TIDX←TIDX–1
        ELSF 1
        ARITH1
        ENDF
        IF OP=2
        CNA    complement A register
        ENDF
        ADD OPB
        ELSF 1
        IF OPA=2
        LDA OPB
        ENDF
        IF OP=1
        ADD ARG[SB]
        ELSF OP=2
        SUB ARG[SB]
        ENDF
        OPA←1
        OP←0
        ENDF
        ENDR
        ENDM ARITH"
```

NOTES

LENGTH is a function which gives the length of the list which is its argument.

NUM evaluates its argument and replaces itself with the decimal encoding of the value. It is useful for constructing a series of symbols over which one wants considerable control.

Double quotes enclosing a string turn it into an alphanumeric constant.

This macro, called by

```
ARITH [(A+B)–(C–D)]
```

would generate

```
LDA  A
ADD  B
STA  T1
LDA  C
SUB  D
CNA
ADD  T1
```

Note that there are only six lines in the definition which actually generate code.

With this example we conclude our discussion of the assembler. The implementation of lists is quite straightforward, though a certain amount of care must be taken about the treatment of colons calling for expansion. A few minor points have been glossed over.

## THE DEBUGGING SYSTEM

A good interactive debugging system must be diffcult for the beginner to master. Its emphasis must be on completeness, convenience and conciseness, not on simplicity. The basic capabilities required are quite simple in the main, but the form is all important because each command will be given so many times.

One essential, completely symbolic input and output, is half taken care of by the assembler. The other half is easier than it might seem: given a word to be printed in symbolic form, the symbol

table is scanned for an exact match on the opcode bits. If no match is found, the word is printed as a number. Otherwise the opcode mnemonic is printed, indirect and index bits are checked and the proper symbols printed, and the table is scanned for the largest symbol not greater than the remainder of the word. This symbol is printed out, followed if necessary by a + and a constant.

The most fundamental commands are single characters, possibly preceded by modifiers. Thus to examine a register the user types

```
/x1-3;  LDA I NUTS+2
```

where the system's response is printed in capitals. This command may be preceded by any combination of modifiers:

| | |
|---|---|
| C | for printout in constant form |
| S | for printout in symbolic form |
| O | for octal radix |
| D | for decimal radix |
| R | for relative (symbolic) address |
| A | for absolute address |
| H | for printout as ASCII characters |
| I | for printout as signed integer |

The modifiers hold until the user types a carriage return.

For examining a sequence of registers, the commands ↑ and ↓ are available. The former examines the preceding register, the latter the following register. In the absence of a carriage return the modifiers of the last examination hold. The → command examines the register addressed by the one last examined.

The contents of a register may be modified after examination simply by typing the desired new contents. Note that the assembler is always part of the command processor, and that debugging commands are differentiated by their format from words to be assembled. (This is not difficult, because the only thing which may occur at the beginning of a line of assembler code is a label.) Furthermore, debugging commands may occur in macros, so that very elaborate operations can be constructed and then called on with the two or three characters of a macro name.

To increase the flexibility of debugging macros, the unary operator is defined. The value of SYM 3 is the *contents* of location SYM 3. With this operator macros may be defined to type out words depending on very complicated conditions. A simple example is

```
TG      MACRO A, B
        TEMP←A
        STOP←1
        CRPT STOP
        IF TEMP>B
        STOP←O
        ELSF 1
        TEMP←TEMP 1
        ENDF
/TEMP;
        ENDM TG
```

called with

```
TG 100, 20
```

it will type out the first location after 100 with contents greater than 20.

Another important command causes an expression to be typed in a specified format. Thus if SYM has the value 1253 then

```
=sym;           1253
```

would be the result of giving the = command. All the modifiers are available but the normal mode of type-out is constant rather than symbolic. If no expression is given, the one most recently typed is taken. Thus, after the above command, the user might try

```
s=;             SYM
```

For convenience, ← abbreviates S=.

It is often necessary to search storage for occurrences of a particular word. This may be done with a macro, as indicated above, but long searches would be quite slow. A faster search can be made with

```
$expression;
```

which causes all the locations matching the specified expression to be typed out. The match may be masked, and the bounds of the search are adjustable. This command takes all the type-out modifiers as well as

```
E
```

which searches for a specified effective address (including indexing and indirect addressing) and

```
N
```

which searches for all words which do *not* match. For additional flexibility the user may specify a macro which will be executed each time a matching word is found.

In addition to being able to examine and modify his program, the user also needs to be able to run it. To this end he may start it at a specified location with

```
;G location
```

If he wishes to monitor its progress, he may insert

breakpoints at certain locations with the command

;B location

This causes execution of the program to be interrupted at the specified location. Control returns to the system, which types some useful information and awaits further commands. An alternate form of this command is

;B location, macro name

which causes the specified macro to be executed at each break, instead of returning control directly to the typewriter. Very powerful conditional tracing may be done in this way.

After a break has occurred, execution of the program may be resumed with the ;P command. The breakpoint is not affected. To prevent another break until the breakpoint has been passed n times the form n;P may be used.

To trace execution instruction by instruction the command ;N may be used instead of ;P. It allows one instruction to be executed and then breaks again. n;N allows n instructions to be executed before breaking. A fully automatic trace has been deliberately omitted, but presents no difficulties in principle.

There remains one feature of great importance in the IMP system, the symbolic editor. The debugger provides facilities, which have already been described, for modifying the contents of core. These modifications, however, are not recorded in the symbolic version of the program. To permit this to be done, so that reloading will result in a correctly updated binary program, several commands are available which act both on the assembler binary and on the symbolic.

This operation is not as straightforward as it might appear, since there is one-to-one correspondence between lines of symbolic and words of binary. Addresses given to the debugger of course refer to core locations, but for editing it is more convenient to address lines of symbolic. To permit proper correlation of these line references with the binary program, a copy of the symbolic file is made during loading with the address of the first and last assembled words explicitly appended to each line. Since the program is not moved around during editing, these numbers do not change except locally. When a debugging session is complete, the edited symbolic is rewritten without this information.

We illustrate this with an example. Consider the symbolic and resulting binary

```
S1  MOVE A, B     (200,201)     S1  LDA A    200
                                    STA B    201
    ADD C         (202,202)         ADD C    202
    STORE D, E    (203,204)         STA D    203
                                    STA E    204
S2  BRU S1        (205,205)     S2  BRU S1   205
```

and the editing command

```
;I        S2–1      insert before line S2–1
SUB F
```

which gives rise to the following:

```
S1  MOVE A, B     (200,201)     S1  LDA A        200
                                    STA B        201
    ADD C         (202,202)         BRU .END     202
    SUB F       (1513,1513)         BRU .END 1   203
    STORE D, E  (1514,204)          STA E        204
S2  BRU S1        (205,205)     S2  BRU S1       205
                                    . . .
                                END ADD C        1512
                                    SUB F        1513
                                    STA D        1514
                                    BRU S1 4     1515
                                    BRU S1 5     1516
```

All the BRU (branch unconditional) instructions are inserted to guarantee that the right thing happens if any of the instructions causes a skip. Multiple skips, or subroutine calls which pick up arguments from subsequent locations, are not handled correctly. The alternative to this rather simple-minded scheme appears to be complete reassembly, which has been rejected as too slow. The arrangement outlined will deal correctly with patches made over other patches; although the binary may come to look rather peculiar, the symbolic will always be readable.

To give the user access to the readable symbolic the command

;S symbolic line address [,symbolic address]

(where the contents of the brackets is optionally included) causes the specified block of lines to be printed. Two other edit commands are available:

;D symbolic line address [,symbolic line address]

which deletes the specified block of lines, and

;C same arguments

which deletes and then inserts the text which follows. Deleting S1 1 would result in binary as follows

```
S1      LDA A
        BRU .END
        BRU .END 1
        STA D
        STA E
```

```
S2      BRU S1
        . . .
.END    BRU S1 3
        BRU S1 4
```

The implementation of these commands is quite straightforward. One entire edit command is collected and the new text, if any, is assembled. Then the changed core addresses are computed and the appropriate record of the symbolic file rewritten.

The scheme has two drawbacks: it does not work properly for skips of more than one instruction or for subroutine calls which pick up arguments from following locations, and it leaves core in a rather confusing state, especially after several patches have been made at the same location. The first difficulty can be avoided by changing large enough segments of the symbolic. The second can be alleviated by reassembly whenever things get too unreadable.

The only other published approach to the problem of patching binary programs automatically is that of Evans,[4] who keeps relocation information and relocates the entire program after each change. This procedure is not very fast, and in any event is not practical for a system with no relocation.

## EFFICIENCY

The IMP system depends for its viability on fast assembly. The implementation techniques discussed in this paper have permitted the first version of the assembler to attain the unremarkable but satisfactory speed of 200 lines per second. Simple character-handling hardware will be installed shortly on our 930; it is expected to double assembly speed on simple assemblies and to produce even greater improvement on programs with many macros and repeats.

Using the latter figures, we deduce that a program of 10,000 instructions, a large one by most standards, will load in 25 seconds. This number indicates that the cost of the IMP approach is not at all unreasonable—far more computer time, including overhead, is likely to be spent in the debugging operations which follow this load. When only minor changes are made, it is, of course, possible to save the binary core image and thus avoid reloading.

In spite of the speed of the assembler, it is possible that a relocatable loader might be a desirable adjunct to the system. There are no basic reasons why it should not be included.

As to the size of the system, the assembler is about 2,500 instructions, the debugger and editor about 2,000.

## ACKNOWLEDGMENTS

The ideas in this paper owe a great deal to many stimulating conversations between the author and L. Peter Deutsch of the University of California.

## REFERENCES

1. G. Mealy, "Anatomy of an Assembly System," RAND Corporation (Dec. 1962).

2. The MIDAS Assembly Program, MIT, Cambridge, Mass.

3. A. Kotok, "DEC Debugging Tape," Memo MIT-1 (rev.) MIT, Cambridge, Mass. (Dec. 11, 1961).

4. T. G. Evans and D. L. Darley, "DEBUG—An Extension to Current On-line Debugging Techniques," *Comm. ACM,* vol. 8, p. 321 (May 1965).

# RESPONSIVE TIME-SHARED COMPUTING IN BUSINESS ITS SIGNIFICANCE AND IMPLICATIONS

Charles W. Adams, President
*Charles W. Adams Associates, Inc.*
*and KEYDATA Corporation,*
*Boston, Massachusetts*

## SIGNIFICANCE

Of the many thousands of businesses in the United States today, it is probable that no two use identical office and accounting procedures. Yet in general it would be safe to say that in any business incoming data are processed, with reference to a file, using established procedures, to yield six broad types of results:

1. Updated file records.
2. Operational documents, such as invoices, purchase orders, pay checks, and the like.
3. Exception notices, status reports, and responses to inquiries regarding the standing of such items as accounts receivable, inventory or personnel records.
4. Historical documentation required by custom, law, auditors, tax officials or boards of directors, whether in the form of printed reports, microfilm or magnetic tape.
5. Reports required by management in addition to, or preferably in place of, the historical documentation mentioned above, consisting primarily of a listing of situations which vary substantially from established norms (that is, exception reports intended for executive policy-making as opposed to those needed by operating personnel).
6. Analytical results, such as sales forecasts and answers (obtained through simulation) to the question "What would happen if ... ?" so frequently asked by management and important to effective policy decisions.

Each of these types of results involves different volumes of information and requirements of frequency and currency. Yet all ensue from the processing of data according to established procedures, and all require access to essentially the same file of information on the history, status, and organizational objectives of the company with which they are concerned.

The purpose of this paper is to discuss the significance of responsive time-shared use of electronic data processing equipment for processing business data to produce the results mentioned above. As used here, responsive time-sharing refers to a multiprogrammed system intended to provide commercial services to many tens or hundreds of remotely located users. In this system information is

introduced through keyboards or other manually operated media. The input data are received and collected into "messages" either by satellite computers or other message buffering devices, or by the central processor or processors. The input is processed promptly upon receipt and the results appear at the remote console quickly enough to influence the subsequent actions of the operator. No paper tape or punched cards are involved.

Such systems tend to entail two added expenses when compared with more conventional, nonresponsive time-sharing on a minute-by-minute or hour-by-hour basis: manual data entry over a telephone line makes inefficient use of that line, and a certain amount of "overhead" is inherent in time-shared processing on a moment-by-moment basis. Both of these costs can be and are being reduced, the first by data compression and traffic concentration, and the second by improvements in the design of hardware and software. But even with existing techniques, responsive time-sharing can demonstrably increase the operating effectiveness of many types of businesses through improved currency, cost reduction, and control in the processing of their data. Each of these is discussed below.

*Currency*

The urgent need for current or timely information was of course the spur which led to the development of the early responsive systems for airline reservation handling, stock market transaction processing, and the like. Currency on a daily, weekly or monthly basis is also essential to top-level management, but it is usually obtainable (though not always obtained) by any data processing system, whether responsive, conventional, or even manual. On the other hand, inventory management, credit checking, production control, cost control and the like are problems encountered to some degree by virtually every business. In these a responsive system usually has a substantial advantage even when the requirement is not urgent enough in its own right to warrant much added expense.

*Cost Reduction*

The potential for cost reduction from the use of responsive computing stems from a variety of sources. An obvious one is the fact that electronic data processing, whether conventional or responsive, offers to the larger organization inherent economies that have not been fully realized by smaller companies because computers are, by their very nature, mass-production devices, and despite significant strides in this direction they have not been available in sizes small enough to meet the needs of many businesses. While the smaller business has had the alternative of time-sharing a larger computer system through conventional service bureaus, the practical or psychological disadvantages of taking business documents (or cards punched from them) to the local service bureau have prevented widespread use of this medium.

Responsive time-sharing can give even the quite small user the effect of having a full-fledged computer of his own through a terminal device located on his premises and connected by a phone line to a responsive data processing system. This advantage of bringing to the user (and charging him for) only the computing capacity he needs, when and where he needs it, is just as basic in engineering and scientific computation as in business data processing.

Another area in which responsive data processing offers recognized economy and convenience to the scientist or engineer is ease of learning, ease of use, and ease of making corrections. What is not always appreciated is that these same advantages, albeit on a somewhat smaller scale, accrue to the business user as well. Training is greatly facilitated by a responsive system, not only because procedures are reduced to minimal simplicity (partly due to freedom from the constraints of the 80-column card) but also because a properly designed system can function as a teaching machine, indicating mistakes or misunderstandings as they occur. This benefits both the experienced operator, by guiding her through extraordinary or unusual procedures, and the neophyte, by allowing her to learn by doing, at her own rate and without the cost or embarrassment of a teacher standing by. Corrections are similarly simplified because mistakes are discovered when they arise and while the source document is still in front of the operator.

An incidental economy, often of more importance than might be thought, is the reduction in space required since few computers are as small as the terminal device of a responsive time-shared system.

In many businesses operational documents such as invoices must be prepared almost as soon as the

information is available. Without a responsive system it is necessary either to use relatively clumsy by-product card or tape punching during the initial handling of the data, or later to re-key part or all of it a second time. Hence, while the repeated handling of data can be eliminated in other ways, a responsive system achieves it almost effortlessly.

A final area of manpower economy is the reduction in cost of managing the data processing function, a by-product of the centralized systematization and control implicit in a time-shared service as discussed later.

Time-sharing, moreover, lowers the effective cost of data processing equipment itself through more effective utilization of it. The user, as previously stated, is charged only for actual usage and not for idle time. Economy of scale—getting more computing per dollar the more one spends for a machine—is a basic factor for the very small user (although not as significant as previously for the larger user since Grosch's Law* no longer applies to computers which rent for more than a few thousand dollars per month). A factor related to economy of scale, and far more important to today's technology, is the potential for effective use of storage hierarchies, keeping in high-priced core storage only the data needed at the moment, with other data, programs and files kept in storage devices lower in both cost and accessibility.

In addition to these, there are two other hardware economies worth considering. First, a properly designed procedure-oriented language for describing the procedures to be followed for one company will permit many others to use the same basic programs even though there will generally be differences in the details of file record layout, output formats, exception handling, and other elements of processing. This permits savings not only in the preparation of procedures but also in the storage of them, a factor of particular significance for frequently used procedures which tend to become resident in core memory. Second, techniques for randomly addressing a disc or magnetic card file are usually more efficient for large than for small files. By combining many files in one file-handling system, a substantial amount of statistical smoothing is obtained and double lookup and/or excess capacity requirements thereby reduced.

*Named after Dr. H. R. J. Grosch, who was the first either to quantify the relationship or at least to state it forcefully enough to clothe it with the mantle of law. It states that, empirically, the ratio of the computing capacities of any two electronic digital computers equals approximately the square of the ratio of their costs. Thus a $10,000 a month computer could in an earlier day have been expected to produce results one hundred times as fast as another renting for $1,000 per month.

*Control*

Perhaps the greatest advantage of a responsive data processing system to the small or medium-sized business is improved control, systematization and management of the data processing function. Conventional service bureaus offer the same advantage, of course, but to a lesser degree. It is also standard policy for suppliers of bookkeeping machines to aid their customers in establishing well-controlled procedures. But one of the biggest sources of difficulty in the business use of larger-scale electronic data processing equipment has been the abdication by the suppliers of their responsibility for providing customers with business systems rather than business equipment alone. The supplier of responsive data processing services, however, can most readily and thoroughly aid the same business manager by providing and maintaining an efficient and well-controlled system.

While it would be possible to allow or even expect the business user to design and program his own system, just as the engineering user of time-shared equipment currently does, the information utilities will best serve their purpose by furnishing to the user not merely a data processing capability but a data processing system service. The cost of systems design, programming, and data processing management is and will remain high. It therefore seems sensible to share these costs among many users just as the equipment is shared by many. And certainly the time-shared responsive system not only facilitates but virtually cries out for such sharing of the cost of system development.

Central time-shared facilities permit close and continuous monitoring of the data processing operations of all its subscribers. There is no excuse, then, for not offering the subscriber a well-designed system and a guarantee that it will continue to be used as planned. This would differ from many systems provided by conventional suppliers or consultants which would work well if used correctly but, once left to the users' own devices, would tend to degenerate through changes made by those performing the job to suit their own tastes without a full understanding of the implications.

This is not to say that a packaged service can or should be developed with the intention of having any substantial number of businesses using it without modification. It is essential that the general system design and its implementation permit "cutting and pasting" of user programs to meet the requirements and desires of the individual subscriber. But the responsive system supplier has a very real responsibility to provide his subscribers not only with what they want—and no more—but with what they need as well. There is, for example, no aspect of the responsive data processing services offered by KEYDATA Corporation that appeals more to prospective subscribers than the fact that they can have the data when they want it and, in addition, can rely on KEYDATA to see that nothing is overlooked with regard to the efficiency and control of their data processing activities.

Finally, for effective management control the convenience of having both computing capacity and the corporate files accessible, essentially at one's finger tips, for study, analysis and forecasting is of tremendous significance. It means that managers will be encouraged to consider facts and analyze trends in a way which few if any of them, aside perhaps from the planning staffs of very large corporations, can or will do today.

## IMPLICATIONS

The responsive business data processing service which, within the next few years, seems so certain to blossom into a computer utility with almost universal appeal to the business world has many implications for the hardware and software required to provide it. Increasingly effective equipment and programming techniques must be found to improve terminal devices, to lower communication costs, and make highly efficient use of storage hierarchies. The integrity of the user's file data must be unequivocally ensured; that is, he must be confident that data will not be accidentally or maliciously lost, altered, or revealed to his competitors. He must also be guaranteed continuity of service without the expense of duplicate equipment; thus it is essential that the system be operational a very high percentage of the time, that interruptions of service be short in duration, and that recovery from equipment malfunction be relatively painless and essentially foolproof. It is therefore clear that the system itself must be truly "clobber proof"; that is, no user may be capable of changing anyone else's program or tying up the system in any way that would disrupt normal service to other users. For maximum efficiency the programming techniques used to supply commercial users should capitalize on every degree each user may want to do things a little differently of standardization that can be encouraged between and among different users. At the same time, since than any other user, adaptability of the system is equally important.

A thorough discussion of all these elements would, of course, be more appropriate in a book than in a short technical paper. But like anyone else who provides or plans to provide responsive time-shared service on a commerical basis, the group the author represents has had to find workable solutions to all the problems implicit in the listing above. Some of these solutions are far from elegant; many will improve rapidly as more and more hardware and software designers direct increasing attention to the problems. Therefore, without attempting even to discuss let alone resolve all of these problems, a few remarks about some of the more interesting design decisions represented in the KEYDATA system may be in order.

### *Terminal Devices*

Certainly the design of the user's terminal is important; but far too much emphasis seems to be placed on high style and novelties to the detriment of dependability and economy. A Model 28 Teletypewriter may seem ancient and unattractive; it may have fewer keys than a new model typewriter, lack lower-case letters, have only a few symbols, and be comparatively slow and somewhat noisy. But until something comes along that is as dependable and inexpensive,* has a higher speed, richer character set and preferably a bit more style, the Model 28 serves the purpose quite well. Full-duplex operation,† permitting easy correction of mistakes and the use of a third level, a control shift, on the keyboards in the KEYDATA system, has proved especially satisfactory.

*The total cost includes, of course, modulating-demodulating equipment if needed, as it is not in the case of Model 28 Teletypes leased from New England Telephone & Telegraph Company.

†The keyboard is connected to the computer and the computer to the printer; but there is no direct connection between the keyboard and the printer.

*Scheduling*

In the matter of response time, the KEYDATA system permits three different approaches: (1) normal full-duplex semiresponsive message processing, in which the user needs the output generated from the input as rapidly as possible, but the only conversational effect is for verification; (2) truly responsive conversational operation, in which the user must wait for a response to one input before he can determine what the next input should be; and (3) job-queued quick-turnaround operation, in which the objective is to give relatively rapid service on jobs in which the amount of processing is so great that the user cannot realistically expect to wait for it without going on to another task in the meantime.

The emphasis of most present time-shared systems is on the conversational mode, which is perhaps the most important and at the same time the most difficult to handle effectively. Since the amount of computing required to generate a response in any conversational situation can sometimes be substantial and is often unpredictable, the crux of the matter is a scheduling procedure that will give each user a "fair shake." The most obvious method, an equal time slot for each user, seems eminently fair but tends to achieve equality by making every user's reponse equally bad. (Suppose, for example, 10 users request a 5-second job at the same instant. Most "fair" systems would give all these users a response after 50 seconds. Serving each in turn to completion, however, would reduce the average response time from 50 to 27.5 seconds without penalizing any one, though perhaps at the expense of making user number 10 resentful of the better service given the other 9.)

The scheduling method used in the KEYDATA system has many of these defects but it does overcome one serious problem. It allows conversational users to reserve a level of service for minutes or hours at a time and guarantees that they will receive at least the agreed level of services on a minute-by-minute basis. In keeping with the doctrine adopted by Vyssotsky at Bell Telephone Laboratories, KEYDATA prefers to deny service rather than degrade it.

Job queuing on a minute-by-minute and hour-by-hour basis is a desirable adjunct to any time-shared system. It means that a user can set up data and procedures, and try them out if necessary, using responsive or semiresponsive services. Then he may, through the same console, request execution of the processing task he has set up, get an estimate of when the job will be finished, keep track of where he is in the queue (if this matters to him), be told when his job is finished (or forego this if he expects to be using his console for some other purpose), examine the results of the completed run (in whole or in part) and, when substantial output is involved, have it printed or recorded on tape at the central computer facility for later delivery to him.

Semiresponsive full-duplex operation is particularly appropriate in commercial applications. Consider, for example, the preparation of invoices using the KEYDATA system. The operator enters a customer number and the computer responds with a full heading for the invoice, including the name and address of the customer, invoice number, date, and other pertinent information. The operator then enters the stock item number and quantity, and the system responds with a full line containing not only the stock number and quantity but also a description of the item, unit price, extension and possibly discount or other information.

This kind of operation differs from the conversational requirement of the engineering user in three important ways: the processing of each message usually involves random access to a large user file but seldom requires much actual computation; output volume (as in job-queued engineering use) is several times that of input and must be attractive in format and free of visible corrections or typeovers; and the operator usually wishes to enter data as rapidly as she can strike the proper keys and need not see the output resulting from one input in order to go on to the next. It is nonetheless desirable that the output be produced soon after the input so that a finished invoice is immediately ready to permit the shipment of an order; inconsistencies in input are called to the operator's attention by the system the moment they occur so that the data can be corrected or verified while the source material is at hand, and basic files are constantly kept up to the minute in currency.

In the KEYDATA system, input data are processed within a very short time (usually a fraction of a second) after being entered, but the output is adequately buffered so that the operator need not wait for the printer before proceeding with the next entry. Since the amount of processing required by

each input message is small and relatively predictable, scheduling in such a system can ordinarily be handled by the simplest possible method: first come, first served.

*Procedure Language*

User programs (that is, the description of the procedures to be followed in processing data for any application by any user) are written for the KEYDATA system in a more or less problem-oriented language, called KOP-3 (for KEYDATA On-Line Processor number three, the first two having been used in a smaller KEYDATA facility which went into operation in July 1963). KOP-language programs can be stored compactly and dealt with interpretively, thus saving storage and facilitating adaptability and integrity. Because they are written by KEYDATA's own staff, these procedures can be expected to make efficient use of the language and equipment. (Many a sound design has grounded on the rocks of inadequately informed or incompetent use!) They can also be cut into quite small segments by the programmer who, in the final analysis, is much better prepared than any 1965-vintage executive routine to determine the natural segments of his program.

KOP-language programs are, of course, pure procedures: that is, the instructions cannot modify themselves in storage and all variable status information is kept in small areas uniquely assigned to each subscriber. Procedure segments (pages of 48 words) are kept on magnetic drum and called into core as needed. Automatic core allocation and scheduled drum accesses (omitting a drum read whenever the desired segment is already in core) result in balanced utilization of core and drum and a high level of central processor utilization.

## CONCLUSION

Responsive time-sharing to serve the business as well as the technical user is here to stay. It requires much more systems work to define and solve the problems of applying it in new areas. It also needs more collaboration between software and hardware designers than has been evinced thus far. Considerably greater attention should be focused on questions of economy in deciding on the tradeoff between hardware and software, in using storage hierarchies, and in determining the kind of service which the time-sharing user should be offered. The next few years will indeed be fascinating ones.

# CIRCUIT IMPLEMENTATION OF HIGH-SPEED PIPELINE SYSTEMS

Leonard W. Cotten
*Fort George G. Meade, Maryland*

## INTRODUCTION

The implementation of high-speed pipeline systems as described in this paper arose as a direct consequence of a large scale Department of Defense developmental effort initiated in 1962. The objective of the effort was to develop and make available a complete capability for producing individual special-purpose systems on a fast reaction basis. From 1962 to the present time attention was focused on all aspects of circuit and packaging technology, automated or computerized design aids, feasibility vehicling, systems design studies, and advanced memory development. As a result of strong industry impetus in these directions it now appears that 1 to 2 nanosecond hybrid integrated or full integrated logic circuits, practical fabrication of transmission line interconnections, packaging densities of 5000 logic gates per cubic foot in the machine environment, 100 to 150 nanosecond cycle time DRO thin film main memories, and 23 to 40 nanosecond integrated scratchpad memories will be made available for systems being constructed over the next one to three year period.

Considering the high potential systems performance promised by these developments and the complexity of nanosecond logic realization it would seem that the systems designer-builder or technology user is faced with a bilateral challenge. First, and perhaps foremost, the user must exploit to a high degree each favorable performance characteristic made available by technology developers. Emphasis will be placed on the traditional engineering compromises; however, increased sophistication and complexity will be necessary. For example, the propagation of a 1 to 2 nanosecond rise time in a logic net requires far more attention to detail and fundamental understanding than say a 10 to 20 nanosecond rise time. Because of the large number of special situations that arise in design, it is important that efficient generalities or ground rules be used. Many considerations suggest that it would be wise for the user to develop straightforward but efficient approaches to recurrent systems problems. As more systems are built, well-known approaches might be augmented such that the designer is able to minimize redundant design effort by not having to start from the beginning with each new logic design. This notion is particularly amenable to a fast reaction capability for one-of-a-kind machines if it is more desirable to expend effort on the problem at hand rather than on details of logic implementation. The second challenge facing the user is that of tightening feedback loops with technology producers and other users. The time is not too distant when a sizable number of interconnections will be

made at the integrated subsystems level. This has occurred at the circuits level and the trend will continue.

In light of the above factors it will be the purpose of this paper to address those areas relating to the implementation of high-speed pipeline systems at the circuits level. The ideas presented are considered to be valid in systems containing up to 50,000 basic gates. Since the circuit packaging and other details have been described in the literature[1-4] only those circuit characteristics relating to pipeline implementation will be discussed. While the concept of pipelining, or possibly streaming, has existed for years* and has been applied in earlier systems, the term appears not to have been broadly established. For this reason examples will be given to permit a better understanding. Main attention will focus on exploiting advanced technology in the pipeline systems environment. Emphasis is given salient features of implementation that will ultimately relate to the pipeline system timing and control structure. The specific logic for a pipeline process may be quite general and is left to the designer of individual systems. For the benefit of advanced technology users it should be noted that many of the concepts presented herein relate to systems implementation in general and may be of interest in applications somewhat removed from pipeline systems.

## THE RATIONALE FOR PIPELINE OPERATION

With the present state-of-the-art in systems organization and technology it appears that pipelining is a powerful approach to a particular variety of large data processing problems. The implementations of systems to process such problems usually display one or more of the following characteristics.

1. Thruput or Byte flow rate is important. The total time required to flow information through a stream of processing logic is less critical than the rate of flow. Since pipeline fill-up time is small compared to total processing time the productivity depends largely on rate. The Byte acceptance and/or output rate is optimized and consequently is allowed to constrain the system.
2. Input/output (I/O) speed is a critical factor even though care is taken to minimize I/O and to keep data buildup and reduction internal to the processor logic. This consideration is normally resolved by developing a memory hierarchy which frequently calls for advancing the memory art significantly.[6-8] The most critical rate exists at the pipeline I/O interface. In many cases this may be described by the following relationship.

$$\text{Memory cycle time} \leq \frac{\text{Memory word size}}{(\text{Pipeline byte size})\ (\text{pipeline byte input rate})}$$

3. High system productivity is achieved by a high degree of parallelism. Bookeeping for example may be performed in a pipeline parallel to the main data processing stream. Serial control is often used to sequence and interleave pipelines. Concurrence is evidenced by the fact that almost all of a system is actively processing data at any given time.
4. Timing is usually performed in the synchronous mode. While the relative merits of asynchronous timing are well known[9] the advantages of synchronous operation normally outweigh the drawbacks in the systems under discussion. This may be attributed to several considerations. First, numerous internal process interactions must be predicted and controlled efficiently. An example is time and spatial injection of new information into a running pipeline process. Secondly, evidence exists to suggest that recent technology is relatively more predictable. In one published report the variance of raw circuit propagation delay was less than 0.5 nanoseconds while typical logic net (circuit, line and loading) delay was approximately 5 nanoseconds. The components of delay associated with transmission line propagation time and load driving are predictable, and typically may account for over half the delay in a net in advanced systems. Also, well-known theorems in statistics[10] may be employed in strings of gates to avoid anticipation of so called worst-case delays. Fi-

*The term pipeline has been used for over 5 years by by designers to describe maximal rate processing of the form discussed. However, the author has so far been unsuccessful in determining the origin of the term as used in this context. See reference 5.

nally, the maximum rate objectives of parallel processors usually rule out lengthy ripple structures of the type where asychronism might be profitably employed. Instead, lookahead gating is incorporated into structures for example.

The concept of pipeline operation may be illustrated by using an adder as a vehicle for discussion. First, total time for performing addition could be minimized by using parallel structures and carry lookahead logic. Add time for 12-bit numbers might be $4T$. Secondly, the same adder could be constructed with fewer gates by using a ripple carry structure. Add time would be approximately $12T$. Thirdly, if the ripple carry structure of the second adder is clocked, and if each successively higher order bit pair from the input numbers is delayed in time by $1T$, then the adder is capable of producing a sum for each period $T$. The skewing of the input numbers and the deskewing of the output sums could be accomplished by delay lines, clocked registers, or by specially ordered memory storage patterns. The third adder is of course a form of pipeline operation, and might be advantageous if many pairs of numbers are to be added at a maximal rate.

A pipeline system would probably result if a special machine was designed to correlate long columns of numbers, $A_i$ and $B_i$, as rapidly as possible. From the relationship shown it is apparent that many products must be formed and accumulated at the same rate. If points $(A_i, B_i)$ where:

$$\rho_{12} = \sum_{i=1}^{n-k} A_i B_i + k$$

were sampled in time it is seen that the process would be repeated for many values of $k$, or time lags. Finally it is observed that a careful balance of memory capability, parallel logic, and system organization would be necessary to achieve the maximum practical rate.

## BASIC PIPELINE OPERATION

A basic pipeline structure is illustrated in Fig. 1. The pipeline is characterized by a succession of register and gate sections. The first section, $(A)$, contains a register which is made up of elements $A_{11}$, $A_{21}, \ldots A_{r1}$ and a gating structure composed of an $(r)$ by $(k\text{-}1)$ element array. Limits have not been have not been placed on $r$ and $k$ as this is a somewhat generalized structure. In practice definite limits are established by logic and timing considerations, and not all elemental positions are necessarily populated. The elements which form registers are considered to be basic register cell structures. The elements which occupy positions in the gate arrays are basic circuits that perform logic and consume time in the process. Information flows through strings of these circuits in an unclocked fashion. For this reason gate strings between registers are kept within specified lengths. Special attention is given the gate-register and register-gate interfaces, especially as timing constraints become critical. The control delay $(CD)$ elements are structured from standard logic circuits and are used to form timing chains, which sequence the pipelines. Timing chains usually account for 10 percent or more of the gates in typical pipeline systems.

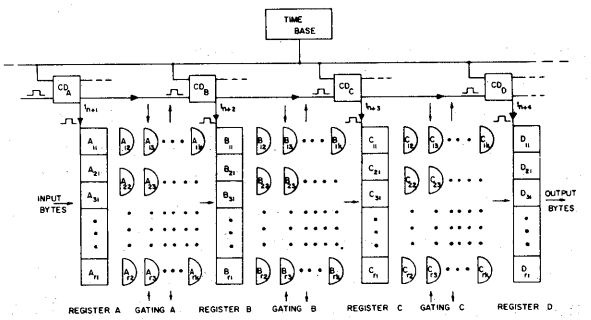



Figure 1. Basic register and gating structure.

An examination of data flow and the timing-control interaction of a basic pipeline will help to explain the operation. Figure 2 illustrates the basic timing for one input BYTE that is subsequently processed through the pipeline. At time $t_{n+1}$ the BYTE is clocked in parallel into the first register $A$. The information appears at the register $A$ output and moves unclocked through logic gating $A$ to the input of the next register $B$. At some time after time $t_{n+1}$ and before time $t_{n+2}$ pulse moves from the first control delay $CD_A$ to set the second control delay $CD_B$ such that a clock pulse from the time base is

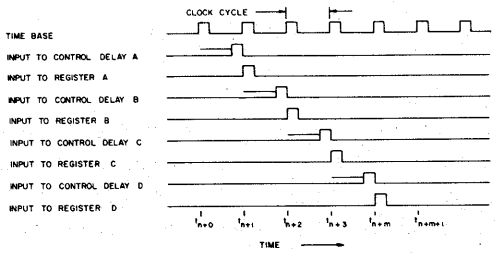



Figure 2. Timing for one input BYTE.

gated through to register $B$ at time $t_{n+2}$. The timing process is repeated through the remainder of the registers. Information arrives at the output end of the pipeline and is available to be clocked into the last register or output device at time $t_{n+m}$. A total of $m$ registrations or clock cycles is required to tranverse the m section pipeline. Four points might be emphasized. First, after introduction of the BYTE into the pipeline the timing chain automatically sequences the information through at the basic clock rate. Second, Figure 2 suggests that the pulse paths between adjacent control delays are noncritical over time periods approaching the clock period. Third, Fig. 3 shows the timing sequence for a burst of six input BYTES. During a burst it is possible that all $m$

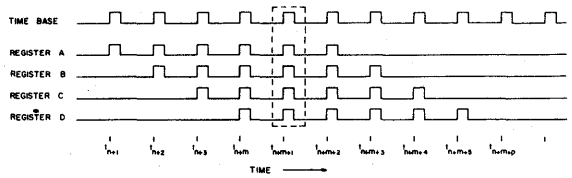



Figure 3. Timing for six input BYTES.

registers may be clocked simultaneously; moreover, this is the desired situation in normal processing. One obtains maximum productivity by minimizing time gaps between bursts. Lastly it is observed that for $p$ BYTES and $m$ pipeline registers a total of $m + p - 1$ clock periods is needed to clock out all information. Now that concepts fundamental to pipeline operation have been set forth, it is meaningful to consider implementation with newly developed logic circuitry.

## CIRCUITS AND LOGIC GATING

### *Basic Circuits*

The circuits being used for implementation purposes are of the well-known current mode switching or emitter coupled logic (ECL) variety. The circuits and detail characteristics have been described in the literature,[11-13] therefore only those properties needed to explain logic implementation will be examined. Figure 4 shows the basic circuit, logic, and an electrical behavior chart. The circuits perform standard NOR-0A(+) or NAND-AND(−) logic depending upon the logic polarity significance. The parenthetical symbmols (+) and (−) are used to indicate which

*The IEEE logic symbol standard is the "American Standard Graphic Symbols for Logic Diagrams," approved by the American Standards Association on Sept. 26, 1962. See *Computer Design,* May 1965, pp. 6-7.

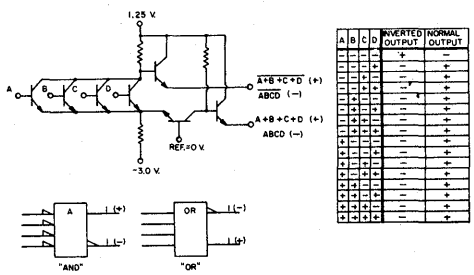



Figure 4. Circuit logic and electrical behavior.

signal excursion is to be identified as a logical ONE. For logic polarity symbols* the open right triangle will be used to indicate negative logic significance and a straight line will indicate positive significance.

Since for purposes of this discussion the reference voltage is zero volts or ground, it is seen that the high swing is a positive voltage and the lower swing is a negative voltage, nominally + 0.4 and − 0.4 volts. Because of the low leakage current associated with the silicon transistors, unused inputs need not be connected. A floating base effectively takes the respective transistor out of the circuit. One other point should be mentioned. The emitter followers are not tied to resistors at the circuit level; however, each logic net in the system will contain a current source resistor and a transmission line terminator in an R (resistor) pac. These provisions make possible the very useful wired OR or DOT OR (+), provided one is able to sacrifice the identity of the individual variables that are shorted together electrically. This connection also forms a DOT AND (−) which proves quite useful. From a logical design standpoint the connection permits two levels of logic in the time slot normally required for one decision. Additional constraints such as placement must be considered, but many useful functions may be realized. For example, the EXCLUSIVE OR, $f = AB + AB$, or the carry equation, $C_n = A_n B_n + A_n C_{n-1} + B_n C_{n-1}$ may be realized in one gate delay for either positive or negative significance if both polarity input variables are provided.

### *Circuit Response in the Systems Environment*

Industry progress in improving circuit response has been quite encouraging; however, the user still must consider numerous design alternatives brought on by bandwidth limitations and line propagation velocity. This is illustrated by the fact that 1 to 2 nanosecond circuit propagation delays have to be

added to 2 to 4 nanoseconds of average interconnection and loading delay in typical cases. Since loading by its very nature produces both lumped and distributive effects, reasonably accurate calculations are less than straightforward. Placement of circuit modules and interconnection paths further complicate the logic design in highly synchronous systems that are optimized for speed. In order to deal with these problems efficiently the designer must be supplied with valid rules of thumb and ground rules. Experience has taught that this is indeed possible, but that the user must be sold on the increased complexity above that encountered in slower speed logic.

The total delay of logic net, consisting of the circuit, interconnection lines and loading, is determinel by a number of contributing effects.[1] For a first order estimate it is necessary to consider only four components. It should be mentioned that these quantitative results are based on (a) effective base input capacities in the 5 to 10 picofarad range, (b) terminated transmission lines that are capacitively loaded so as to present a load of approximately 50 ohms to the driving circuit, (c) logic nets of several inches, and (d) a specific set of circuit design compromises.

1. The first component of interest is raw circuit delay. For the circuits being discussed propagation delay is 1 to 2 nanoseconds. Voltage transition times are defined as between -0.2 volts and 0.2 volts, and the propagation delay is measured from the start of the input transition to the start of the output transition. Since input transition time is a parameter with respect to this measurement, approximately 1 nanosecond is assumed (best case). This is approximately the circuit output transition presented to a 50 ohm resistive load.
2. Second, a transmission line propagation delay of 0.18 nanoseconds per inch must be considered. This is established by the glass epoxy dielectric used in the laminated interconnection cards which accept the circuit modules and the interconnection boards which accept the cards.
3. The third component of net delay arises from attaching base loads to the transmission line and appears in the form of an increase in line propagation delay. This effect and the one below (4) has been graphically described by Flynn.[1] In practice the addition of a base to a moderately loaded line of a few inches in length may add as much as 0.25 nanoseconds of delay.
4. The last delay component to be considered is an increase in circuit propagation delay attributed to input transition times being greater than the original one nanosecond in the best case. Circuit propagation delay increases roughly by one-half the increase in the input transition time from the original one nanosecond. The actual input transition time is determined by base loading on the driving net. In practical situations the addition of one base onto the driving net may cause the propagation delay of circuits being driven to increase 0.15 nanoseconds.

It should be noted that base loading caused delay in two separate ways. These response considerations give some insight into decisions that must be made by a designer using high speed logic of this type. Considerable freedom exists as to possible choices for a given logic implementation, but appreciable time may be spent in arriving at the best set of decisions. For these reasons it would seem desirable to have some standard approaches to recurrent situations in pipeline systems particularly. At the present time it seems reasonable to strive to get total delay per net down to approximately 4 nanoseconds in pipeline applications.

### *Register Cell Implementation*

Historically, the NOR implementation of a gated set-clear flip-flop has been widely used. The NOR implementation may be seen in Fig. 5 along with two more recent developments in the evolution of basic register cells. The NOR flip-flop is well known; therefore, a review of operational characteristics will suffice. Input consists of a negative gat-

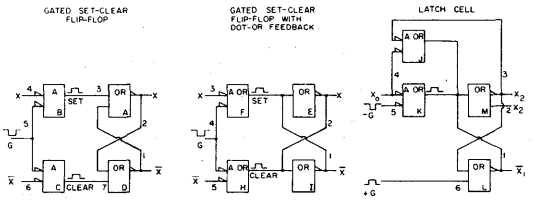



Figure 5. Register cell structures.

ing pulse ($-G$) and normal ($X$) and complemented ($\overline{X}$) data input levels. If the symbol $T$ is used to represent a nominal circuit propagation delay, it is seen the $2T$ best case and $3T$ worst case is required to obtain valid outputs from the time that the gating pulse is initiated.

A useful modification of the NOR flip-flop is the DOT OR feedback. It is seen in Fig. 5 that feedback lines 1 and 2 are wired to lines entering circuits E and I such that additional base inputs are not required. Propagation delay through this flip-flop differs from the first example in a basic respect. Since one of the output lines initially rises with the SET or CLEAR pulse the outputs are available earlier. In general, output information is available in $1T$ best case and $2T$ worst case.

The register cells just mentioned have disadvantages in high-speed pipelines and the latch cell, Fig. 5, has much to offer both in terms of speed and logic implementation. The latch as a functional logic array has been realized as a high-speed module and was mentioned by Flynn.[1] It is the redundant feedback path through circuit $J$ that contributes to many outstanding cell characteristics.

1. Redundant data input information is not required. This frequently saves time and simplifies logic.
2. The $+G$ and $-G$ gating pulses may be skewed either way with respect to each other without impairing latchup. In practice $+G$ and $-G$ should coincide as closely as possible for maximum pipeline speed.
3. A close examination of the latch will reveal that the data outputs ($X_2+$, $X_2-$), lines 2 and 3, will always occur in $2T$. The complemented output ($X_1-$), line 1 is clearly available always in $1T$ because it is the only line seen to enter circuit $M$.

A curious and useful DON'T CARE function is displayed by the latch under certain conditions. To realize this condition the $-G$ pulse is put into the data input and the $-G$ pulse input is permitted to float.

(a) If the $-G$ pulse arrives later than or with the $+G$ pulse the normal data outputs assume the logical ONE state.

(b) If the $-G$ pulse fails to appear then the $+G$ will cause the normal data outputs to assume the logical ZERO state.

(c) If the $+G$ pulse fails to arrive then the $-G$ pulse will cause the normal data outputs to assume the logical ONE state.

The apparent indifference as to the arrival of a $+G$ pulse when the logical one state is to be assumed and the lack of dependence on the arrival of a $-G$ pulse when the logical zero state is to be assumed are both useful in implementing the control delay function that will be considered later.

Register trigger or gating pulse requirements must be established prior to the design of the synchronization for high-speed pipelines. Of interest is the determination of the minimum width gating pulse that must be supplied to registers to insure correct information storage. Each of the flip-flops or register cells previously considered has an almost identical closure path. An examination of all closure paths has resulted in some interesting observations; however, it might be more practical here to consider a worst-case path for each register cell.

1. Consider the first NOR flip-flop in Fig. 5 with a ONE stored and a ZERO about to be gated in and stored. The $-G$ pulse causes the $C$ output on line 7 to rise. The $D$ output on line 1 falls causing the $A$ output on line 2 to rise. Line 7 is then permitted to fall. To simplify calculations the effects of loading and line propagation delay are included in maximum and minimum circuit delay. If symbols $G$, $C$, $D$ and $A$ are used to represent times associated with pulse width and circuit propagation delay then an expression for the minimum gating pulse width, $G$, requirement may be stated as:
$$G = C - C + D + A \qquad (2)$$

2. Consider next the DOT OR flip-flop with the same initial conditions as above. The closure path HIEI is analogues to CDAD even though the outputs of $H$ and $E$ are wired together.
$$G = H - H + I + E \qquad (3)$$

3. The latch has a redundant feedback path; therefore, one path is temporarily negated. Consider the latch as containing a ZERO initially with a ONE on the input data line

4. The $+G$ pulse negates the feedback path $LML$. The closure path is $KMJM$. One may suspect from this example that

something in the latch is wasted, but a consideration of all binary possibilities proves otherwise. The minimum width requirement placed on the $-G$ pulse is:

$$G = K - K + M + J \tag{4}$$

From these example cases it is seen that the minimum gating pulse width is determined by pulse width reduction through one circuit and the maximum delay through two other circuits. Care must be taken in measuring the possible pulse width shrinkage through a circuit because pulse width is a parameter in the limiting case. The interesting phenomenon is question is due to a finite recovery time inherent in all high-speed switches. There may even be an external recovery due to line reflections. In effect a circuit may turn on slowly only to turn off quite rapidly because the internal currents did not complete a full swing to the d-c rest point. If $T$ and $T$ are maximum and minimum circuit propagation delays in our domain of interest then for registers of the type discussed the maximum pulse width requirement will be:

$$G = 3T - T \tag{5}$$

As $T$ is made to approach $T$, $G$ will approach $G$, and $G$ may be approximated by saying:

$$G = 2T \tag{6}$$

An analogy may be drawn between (a) the time a circuit output is seen to switch and the time the circuit is internally stabilized and (b) the time information appears on the output of a register and the time the register is fully latched.

*Logic Gating In Pipelines*

Logic gating in pipelines is usually considered to be gating between registers. For this reason the constraints on maximum and minimum numbers of gates in the critical path are determined respectively by maximum clock rate considerations and data race conditions. Between these two limits the number of gates is usually noncritical. If $T$ is used again to represent a nominal unit of circuit, line propagation, and loading delay, then in 50 megacycle pipelines $4T = 18$ nanoseconds has been established as a reasonable design objective in most cases. This suggests that if each gate was treated separately, then 4.5 nanoseconds would be the maximum $T$ ; however, in practice the 18 nanoseconds may be applied to the total string of gates, lines, and loading for the maximum case. While 4 circuits are normally used, it should be said that 3 circuits under heavy average loading and long paths or 5 circuits in short and lightly loaded paths are not excluded. From the previous analysis of registers it is seen that out of the $4T$ delay almost half the time may be spent in a register. For this reason special attention must be given to (a) optimal methods of gating in and out of registers and (b) ways to obtain as many logical decisions as time permits between registers.

Examples of special register input and output gating may be seen in Fig. 6. The DOT OR flip-flop provides for a level of input gating with circuitry, or two levels of decision gating are possible if the DOT OR is used judiciously. The time for this gating is included in the $2T$ worst-case register propagation time. In practice, to realize this gating, one of three things must usually be done: (1) Completely separate SET and CLEAR equations may be factored. (2) The register may be automatically cleared with a simpler logic expression. (3) If neither of the first two approaches is feasible then it is usually possible to insert a redundant feedback path of the form used in the latch. Gating our of a register in the shortest possible time may be done by using the NOT output ($X_1$) which is available from the latch in $1T$. In Fig. 6 the output of Gate $N$ actually is available as soon as $X_2$ appears on the latch output. The power of this latch—$N$ gate usage is best demonstrated in the selection table example which follows.

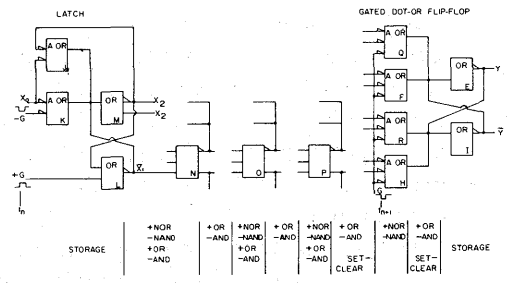



Figure 6. Gating between registers.

Suppose one wishes to store a 4 bit address in a register and use this information to select one out of 16 possible lines in the shortest possible time slot. Assume further that special circuit packaging is a consideration and a geometrical constraint exists on the maximum separation of the outputs in the DOT OR (+) which is also the DOT AND (−). Such a set of considerations is typical, and a solu-

tion is seen in Fig. 7. The DOT AND (-) was used; therefore, the selected line and only this line will go to a negative voltage level for the input conditions shown in the table. Since we are in a negative (-) logic system the low level voltage is interpreted as a logical ONE (-). A careful examination will show that each line is wired to four individual circuit outputs, and when compared to other implementations for this speed range the approach seems reasonably efficient. The relative merits of this solution can best be evaluated with respect to (1) timing, (2) packaging and (3) further logic considerations.

1. The shortest time possible to clock information through the latch and use it to select a line is found to be $2T$. Depending upon the spacing between the registers and the matrix, 2 to 3 nanosecond time must be budgeted to drive 8 external loads from the register. The time for the DOT decision is neglected.
2. It is observed that a module ( module is a 0.5-inch square multiple circuit package with 16 pins on 0.125-inch centers) made up of 4 1-input circuits could be most efficiently utilized. This is not essential, however, because a separate line goes to each input. Using either approach the geometrical constraint on wired (DOT ANDED) emitter spacing is noncritical here. The maximum distance between the two dots on the line to be selected is well within an inch, which in turn is well under the maximum permitted. If advantageous the entire matrix could be partitioned vertically, as shown by the segmented line, and each half could be placed manually or by an automatic placement routine. The module count for this example, including 4 latches, could be as low as 12 modules.
3. From a logic standpoint other facts should be mentioned. In order to get correct outputs from the latches in a $1T$ time delay, complements were fed in. These complements could just as well have been normals. If this were the case, as may be seen from the Fig. 7 table, line 1 would become 16, line 2 would become 15 and so forth. Another alternative would be simply to interchange the normal and inverted outputs from the basic circuits. These considerations are somewhat trivial, but are cited to show that negative (−) or positive (+) input logic presents no proble min this case. For larger address decode problems it is suggested that one explore the possibility of using more than one input on the last level of circuits, granted this is not the only consideration. In passing it should be clear that if the 16 lines are each to be ANDED with a variable of interest by driving another circuit input, then the negative (−) logic significance is needed, and this is found to be true in most cases.

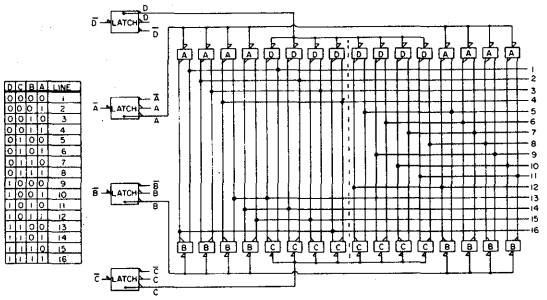



Figure 7. Implementation of selection table.

Now that some optimal methods of gating in and out of registers are known it is desirable to determine the maximum number of consecutive decisions that might be made between registers when a maximum of two circuits may be inserted into the average critical path. Figure 6 shows a maximum critical path of $G_n \frac{K}{L} NOPG_{n+1}$ where $G_{n+1}$ is the next pulse in time (20 nanoseconds later, neglecting skew, for a 50 megacycle clock rate), which gates the resultant of all the path decisions into the DOT OR flip-flop. A count of the Fig. 6 listing shows that storage plus eight serial logic decisions are possible. The seventh and eighth may be weakened by possible SET or CLEAR requirements. If the last two are essential decisions it might be necessary to incorporate redundant feedback into the last register to ease SET-CLEAR requirements. With the DOT OR (+) or DOT AND (−) one does sacrifice identity of component variables, but if this can be tolerated then the result is a decision in approximately zero additional time. Lastly, it is known that levels of NAND (−) NAND (−) are equivalent to OR (+) AND (+) or AND (−) OR (−), as could be shown by successive applications of DeMorgan's theorem.

## PIPELINE TIMING CONSIDERATIONS

Timing analysis indicates that the maximum rate single phase pipeline operation is possible only after (a) register pulse requirements have been met and (b) the race paths have been eliminated. It will be shown that condition (b) is analogous to saying $4T - W_c - S_c \geq 0$ where $T$ is the minimum circuit-line-load delay, $W_c$ is the original clock pulse width, and $S_c$ is the maximum possible clock skew. Actual skew is contained within the interval $0 \leq$ skew $\leq S_c$. Further considerations suggest that in order to meet the basic conditions one might find it advantageous to alter $T$ and $T$ in specialized cases such as in registers, and to carefully consider detailed factors associated with each system.

### *Data Race and Latching — Theoretical Considerations*

The insight provided by a theoretical treatment of the data race problem aids eventually in the application of a practical solution. Since two adjacent registers in a single phase pipeline may be thought of as receiving the same clock pulse the hazard exists that data clocked into register 1 (point *a* in Fig. 8) may race ahead through a short, fast path and appear

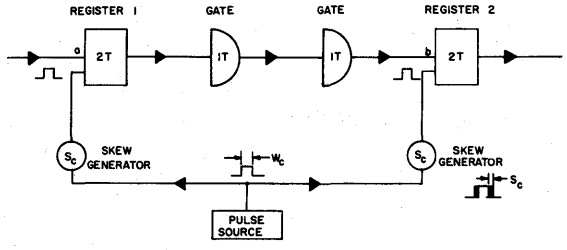



Figure 8. Timing model.

at the input of register 2 (point *b*) before the original clock pulse disappears. This is hazardous in two respects. First, the clock may correctly gate in both the original data and also the unwanted data from the malfunctioning critical race path. Second, the input data must remain a level for the duration of the clock pulse to insure correct gating and original storage in the register. The undesirable consequence of data races may be avoided by designing enough delay into the shortest path to insure data arrival at the next register after the clock pulse has disappeared. In order to determine an adequate amount of additional delay one must first determine the latest possible time after both registers have been clocked that the clock may still be present. If for a pulse of width $W_c$ the total effects of differences in path lengths, loading and distribution circuits are included in the skew term $S_c$ then the latest possible pulse disappearance time is $W_c + S_c$, from the earliest appearance at either reference point *a* or *b* in Fig. 8. If the fastest logic circuit path is $4T$ then an inequality to insure race free operation may be stated.

$$W_c + S_c \leq 4T \text{ or } W_c \leq 4T - S_c \qquad (7)$$

From Eq. (5) it is seen that to insure correct data input clocking a pulse of $3T - T$ nsec of width is required. If the term $S_c$ is a random variable controlled only within limits it follows that the minimum pulse input to a register is $W_c - S_c$; therefore, an inequality to insure correct register pulsing may be stated.

$$3T - T \leq W_c - S_c \text{ or } 3T - T + S_c \leq W_c \qquad (8)$$

Considering race hazard elimination and minimum pulse requirements it is possible to combine Eqs. (7) and (8) to establish an interval for $W_c$.

$$3T - T + S_c \leq W_c \leq 4T - S_c \qquad (9)$$

A solution for $W_c$ in Eq. (9) exists if $T/T \geq 1$ for actual values of $T$ and $S_c$. For 50 megacycle pipeline systems using the technology under discussion a maximum system clock skew of $S_c = 2$ nanoseconds has been established as feasible. The maximum $4T$ delay is then 20-2 or 18 nsec and $T = 4.5$ nsec. Based on this information Eq. (9) may be solved for $T$, a maximum value of the ratio $T/T$, and pulse width $W_c$.

$$T = \frac{3T + 2S_c}{5} = \frac{3\ (4.5 + 2\ (2)}{5} = 3.5 \text{ nsec} \qquad (10)$$

Therefore: $T/T = 4.5/3.5 = 1.28 \geq 1$ and is theoretically acceptable. And: $W_c = 12$ nsec by substitution.

### *Data Race and Latching — Applied Considerations*

While the theoretical treatment in the preceding section produced a solution it is not too surprising that in practice the considerations of $T$ and $W_c$ are not straightforward. This follows from the fact that usually an objective of an advanced circuit developmental effort is to produce the fastest possible stable circuits. The actual value of $T$ is based on population sampling and more detailed considerations. For the circuits being discussed $T$ may be considered as

considerations as there are designers. One consid- 2 nsec when at least one base load is present along with a finite interconnection line and an average input transition time. The determination of $W_c$ could probably be based on as many or more individual eration that has some merit is that presented below.

$$\text{Clock period} = 1/f = 2\ (W_c + S_c) \quad (11)$$

Relationship (11) has implications with respect to clock systems implementation and systems usage. For one thing it permits a two phase clock with zero overlap at the maximum frequency of interest, and thus would permit shifting betwen adjacent registers with no intermediate serial gating.* From Eq. (11) it is seen that $W_c = 8$ nsec for $f = 50$ mc and $S_c = 2$ nsec. Some additional consideration should be given to establishing a reasonable maximum effective clock pulse shrinkage in the system. That $W_c - S_c$ is a reasonable minimum pulse width may be countered as follows.

(a) From Fig. 9 it is seen that if the pulse starts at time $t = 0$ then the minimum width pulse has to be $W_c$ nominal or 8 nsec.

(b) If the pulse starts at time $t = 0 + S_c/2$ or midway in the skew interval it is seen that the minimum width pulse is $W_c - S_c/2$ or 7 nsec.

(c) For a hypothetical pulse width of $W_c - S_c$ the start surely has to occur at time $t = 0 + S_c$. For a pulse to have been delayed originally by this amount the maximum excess line and load delay would have been encountered. Since line and load delay tends to slow both the leading and trailing edge of a pulse it is suggested that any resulting pulse from $t = 0 + S_c$ is greater than $W_c - S_c$ by an appreciable amount.

Statements (a), (b), and (c) do not of course constitute a rigorous proof of the hypothtesis that a pulse of width $W_c - S_c$ cannot occur. It is considered that sufficient grounds were established to invoke a minimum pulse restriction on the clock system of $W = W_c - S_c/2$ without appreciably affecting clock system implementation.

Practical race and latch constraints may now be stated for a realizable pipeline system with many factors considered. To eliminate the race hazard, relationship (7) is modified into the Eq. (12) race constraint.

$$(T_1 + T_2 + T_3 + T_4) \geq W_c + S_c \quad (12)$$

for a string of 4 gates in a race path. Equation (12) differs from (7) in that the minimum of the sum of 4 delays is used rather than a sum ($4T$) of 4 minimum delays. For $W_c = 8$ nsec and $S_c = 2$ nsec, it follows that the effective $T$ average, for 4 gates in a race path, must be equal to or greater than 2.5 nsec. Considering that $T = 2.0$ nsec and taking into account of the effects of line and load delay, the minimum average delay of 2.5 nsec is reasonable.

To insure correct register closure or data latching, relationship (8) is restated in Eq. (13), the data latching constraint.

$$(T_1 + T_2 + T_3) - T \leq W_c - S_c/2 \quad (13)$$

for a closure path. It follows that the maximum average delay for 3 circuits in a register closure path is 3.0 nsec, besed on previous assumptions. This average is easily achieved with a functional module.

If one is justified in assuming that the first circuit, the $-G$ input gate, in the latch has negligible $T$–$T$ for itself and immediate loading then Eq. (6) can be modified to (14).

$$(T_2 + T_3) \leq W_c - S_c/2 \quad (14)$$

for a register closure path with negligible $T$–$T$ for the input gate.

Under Eq. (14) conditions the maximum average delay for the remaining two gates in the closure path is 3.5 nsec. This figure suggests the possibility of placing heavy logic loads close to the register. Also, the possibility of paralleling a latch with a load driving circuit tied to the wired node should be considered for load driving that in any way threatens reliable closure.

*Pipeline Clock Rate Equations**

Pipeline clock rate equations may be developed directly from the timing diagram seen in Fig. 9. For analytical purposes three regions exist. The region widths $W_1$, $W_2$ and $W_3$ are stated as follows.

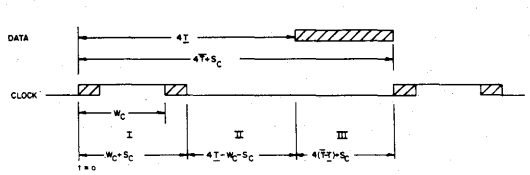



Figure 9. Timing diagram.

*This point was brought out in a private discussion with J. Alton and F. Miller of Remington Rand Univac Corp., St. Paul, Minn., April 27, 1965.

Region I:

$$W_1 = W_c + S_c \tag{15}$$

Region II: This region is established as:

$$W_2 = 4T - W_1 = 4T - W_c - S_c \tag{16}$$

Region III: This is the data arrival interval. That the width is not affected by inserted fixed delay may be seen if a dummy variable $Z$ is used. The earliest possible data arrives at register 2, Fig. 8 after a delay of $4T + Z$. The longest arrival delay is $S_c + 4T + Z$.

$$\begin{aligned} W_3 &= (S_c + 4T + Z) - (4T + Z) \\ &= 4\,(T - T) + S_c \end{aligned} \tag{17}$$

To simplify matters the term $S_c$ in $W_3$ will be added to $W_4$ for frequency calculation purposes in:

$$f = 1/(W_1 + W_2 + W_3) \tag{18}$$

$$= \frac{1}{(W_c + 2S_c) + (4T - W_c - S_c) + 4\,(T - T)}$$

A determination of operating frequency is of interest for three special cases.

*Case 1.* When $4T - W_c - S_c > 0$ the terms in the denominator of Eq. (18) may be added directly. It is tacitly assumed that register closure requirements have been met by $W_c$ and $S_c$. From Eq. (18):

$$f = 1/(S_c + 4T) \tag{19}$$

*Case 2.* When $4T - W_c - S_c = 0$, it is seen that Eq. (18) becomes:

$$f = \frac{1}{(W_c + 2S_c) + 4\,(T - T)} \tag{20}$$

At first glace Eq. (20) appears to be significantly different from (19). If $4T - W_c - S_c = 0$ then $T = (W_c + S_c)/4$. If this $T$ is substituted back into Eq. (20):

$$f = \frac{1}{(W_c + 2S_c) + 4T - 4\,(W_c + S_c)/4} = \frac{1}{(S_c + 4T)}$$

It is seen that $T$ here was a restatement of the data race consideration in Eq. (12).

*Case 3.* When $4T - W_c - S_c < 0$ a race hazard exists and may be eliminated by adding delay of some value $k$ such that:

$$k \geq W_c + S_c - 4T \tag{21}$$

It is recognized that adding a delay $k$ increases $W_2$ in Eq. (16). Since $W_2 = 4T + k - W_1 = 4T + k - W_c - S_c$, as a result, the frequency becomes $f = 1/(S_c + 4T + k)$. In order to recoup speed $T$ might be reduced by an amount $k/4$.

If $T = 4.5$, $T = 2.5$, $W_c = 8$, and $S_c = 2$ nanoseconds in Eq. (19)—and (20)—then $f = 50$ megacycles.

It is emphasized that the clock rate calculations herein presented are based exclusively on single phase operation. In reality the period $4T + S_c$ is valid for single phase clocking and $W_c + 2S_c + 4\,(T - T)$ is theoretically realizable with some multiphase clock scheme. With the numbers given for 50 megacycle operation both yield the same frequency. If one could add sufficient delays to all fast nets such that the $T$ specification could be increased and if the required phasing could be supplied then Eq. (20) will predict the theoretical performance. At first glance multiphase operation from a speed standpoint appears to have merit; however, investigators* have ventured opinions that in the systems under discussion it may not be worth pursuing. Such opinions exist because: (1) precise minimum circuit delay above some natural set of limitations can be difficult to predict with confidence, (2) phase control requirements in the clock system may complicate the organization, and (3) design constraints become more restrictive in the inherently complicated nanosecond system. These points will not be explored further, but the interested reader is encouraged to pursue the matter.

## TIMING CHAIN IMPLEMENTATION

The role of timing chains in pipeline systems was established in the discussion on pipeline operation. Figures 1, 2, and 3 further establish the dependence of pipeline operation upon timing chains in more or less conventional systems. The following discussion will consider (a) the control delay cell implementation using current mode switches without delay lines, (b) detailed aspects of time base implementation for highly synchronous systems, and (c) some standard applications of control delays to illustrate interesting capabilities.

### *The Control Delay Cell*

For purposes of this discussion the control delay* may be thought of as having three main functions:

*The author acknowledges many fruitful discussions with other investigators who were concerned with details of similar calculations. Notable among these were personnel at RCA, Camden, N.J., and at IBM, Poughkeepsie, N.Y., and Dr. H. H. Loomis of the University of California, Davis, Calif.

*Private correspondence from Dr. H. H. Loomis of the University of California, Davis, Calif.

(1) It receives a set pulse and at a predictable time later produces output pulses of both polarities and with fixed width and skew characteristics. (2) The control delay cell furnishes fan-out to clock registers and also furnishes certain control outputs to drive slaves for even greater fan-outs. (3) Finally, the control delay performs all functions in such a way that it may be gated by logic commands.

To enhance systems realizations it is advantageous to construct a basic control delay cell (Fig. 10) from standard logic circuits without the benefit of

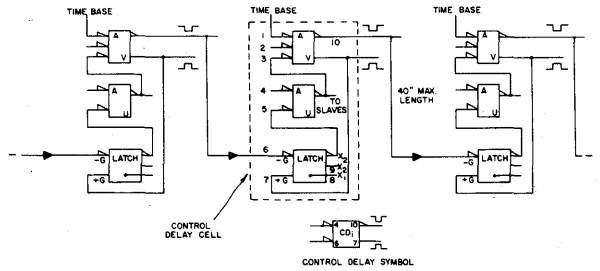



Figure 10. Three control delays in a timing chain.

fixed delay lines. Control delays may be better understood by making reference to the timing in Fig. 11. Operation consists of feeding the $V$ circut with pulses on line 1 from the time base. If line 2 is floating

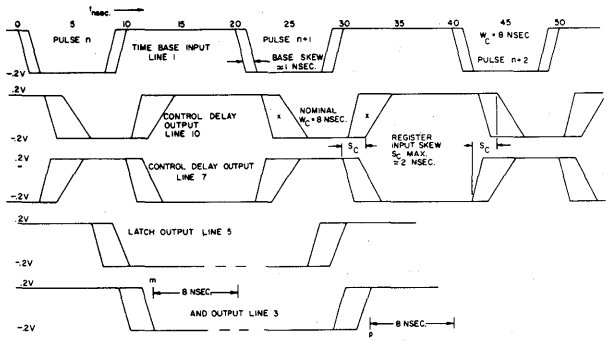



Figure 11. Control delay timing.

(unused) and line 3 is at a low voltage level then positive and negative output pulses appear on lines 7 and 10. For output pulses to have occurred the latch output, $X_2$, had to be a logical ONE. The line 7 positive output is used to furnish a $+G$ latch reset such that no more output pulses are produced unless a $-G$ latch set pulse is received from a previous control delay. Note that no conflict need exist for simultaneous $-G$ and $+G$ pulses due to the DON'T CARE property discussed in the section on registers. Since the $+G$ path is made short, the $-G$ path is always longer, and it takes precedence over a possible $+G$. Circuit $U$, the AND (–) gate, performs two functions: (1) It serves as a buffer delay such that a $+G$ cannot race around and literally reduce its own output width by prematurely causing the $V$ circuit to be inhibited during the negative portion of the time base. Also, the delay is provided such that a $-G$ input cannot race ahead and gate out an earlier portion of the time base than is desired. Note that the line 3 waveform shows that both of these conditions were met; moreover, the output of $U$ could have been delayed 8 nanoseconds and yet have met these conditions. This brings us to the second function of the $U$ gate. (2) The input line 4 may be held high to inhibit the control delay output indefinitely. Some examples of this usage will be described below. Also due to the 8 nanosecond possible delay of line 3, or the $U$ gate output, up to approximately 40 inches of line could exist between adjacent control delays with no serious consequences. Since the control delay effectively gates out a known portion of the time base, the negative pulse portion, the skew throughout a system remains under control. Total skew consists of that prevalent in the time base and the $V$ gate with the output line and load differences. If the output loads per $V$ circuit or slave are held within 1 to 4 registers with no more than 4 inches of effective line difference between any 2 then total skew appears to be approximately 2 nanoseconds when referenced at register inputs throughout a large system. This was calculated for a wide range of load placement conditions and is shown in Fig. 11, lines 7 and 10 waveforms. It should be mentioned that this representation of pulses should not be interpreted as suggesting very narrow possible output pulses. Rather, what is shown are the pulse extremes for the shortest lightest loaded path and the longest heavily loaded path. Pulse shrinkage or stretching would be held to no more than 1 nanosecond.

A wide range of useful functions are possible when slaves are attached to control delays. By way of explanation a slave in considered to be another $V$ circuit tied to line 3 and the time base. For one thing, more slaves can be added for load driving without affecting skew. Another interesting possibility arises if one desires to inhibit the $-G$ command otherwise proceeding from a previous control delay. Simply DOT-ANDING (–) to line 6 could interfere with the load driving of the previous control delay. If a

*The term control delay in this context is believed to have been originated by Remington Rand Univac engineers, some time prior to 1960. See also reference 14.

slave drove line 6 then no such problem need exist. Lastly, since a slave may itself be gated it is possible to inhibit control-delay slave outputs and yet furnish a $+G$ to clear the control delay, without driving registers in the process.

*Time Base And Pulse Forming*

In order to provide a basis for discussion the conceptual time base distribution scheme of Fig. 12 will be considered. Two problems in time base distribution are holding skew within acceptable limits and providing sufficient drive capability at many remote points in space. One solution is to drive a number of cables of uniform electrical length from a common active or passive, frequency insensitive, power divider or power divider-multiplier. In one case that was implemented it was possible to use a commercial power amplifier, a resistive divider, and tunnel diode clippers to furnish square waves with less than 1 nanosecond rise-fall times into approximately 100 50-ohm cables with approximately 0.2 nanoseconds total skew. In the 50,000 gate system layout of Fig. 12 the cables would terminate in amplifiers that would shape pulses and provide drive over some local sector within acceptable skew limits. If skew in each local sector was within limits of 0 and $S$ nanoseconds then it can be seen that any two points over the total $68'' \times 51''$ area would differ by no more than $S$ nanoseconds of time skew, provided cables are of equal length and divider skew is negligible.

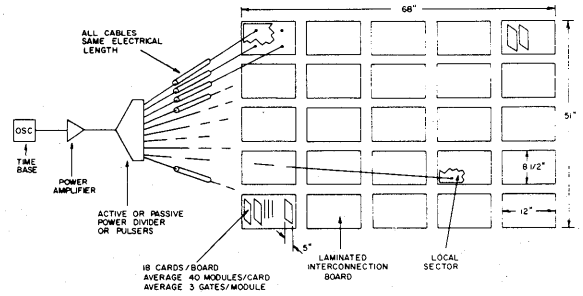



Figure 12. Time base distribution.

Figure 13 shows two methods of producing fixed width time-base pulses by use of delay lines. In method (1) the time-base square wave is brought into an amplifier that furnishes normal and inverted outputs as shown. By delaying one path for $d$ nanoseconds, the $V$ circuit can be made to produce $d$ nanosecond pulses as shown in the waveforms, pin

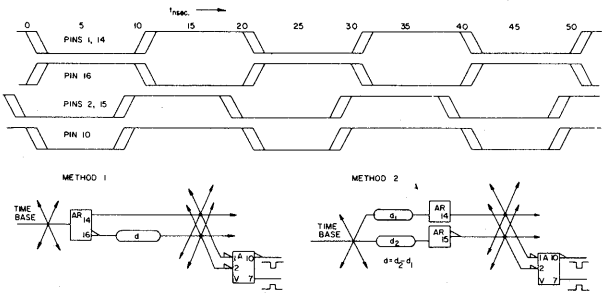



Figure 13. Pulse forming techniques.

10. The $V$ circuit is actually part of the previously discussed control delay cell and the delay $d$ is a fixed delay placed in the system. Method (2) effectively doubles the number of time-base input lines, but permits delay $d$ to become differences in these incoming lines and thus requires no fixed delays in the logic portion of the system. Both methods have the desirable characteristic of independence between pulse width and frequency up to some maximum frequency and maximum pulse width. Below these maximums changing one would not appreciably affect the other. A third method is to design special pulse circuits and drivers which would probably operate at the sector level. In general a capability for adjustments in the suggested areas is necessitated by anticipated improvements in circuit performance, unpredictable factors in final circuit loading and placement, and design policies that could prove to be either overly optimistic or conservative.

*Special Capabilities Realizable With Control Delays*

Several useful capabilities in systems are made possible by the feedback and gating capabilities offered by control delay cells. In the simple case a control delay output (Fig. 10, line 10) could feedback into the input ($-G$ or line 6). Once such a control delay is set it will run at the clock rate and furnish output pulses until the feedback is interrupted. The last control delay output may be delayed indefinitely by holding line 4 positive, or by clearing the latch, a subsequent output could be prevented. By appropriately selecting feedback points from successively farther down a control delay timing chain it is possible to produce pulse trains with apparent frequencies of 1, ½, ⅓, ¼, . . . $1/n$ times the basic clock rate. A wide range of phase relations are possible by selecting desired outputs. A more practical control delay capability is seen in Fig. 14. The application of

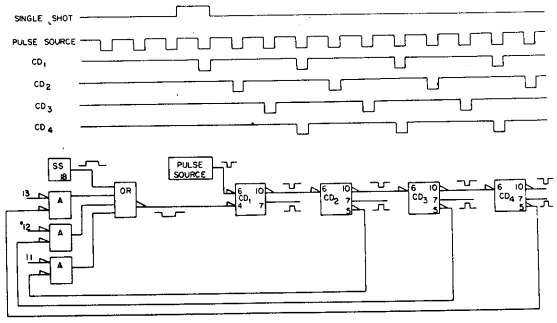



Figure 14. Control delay feedback and gating.

a logical ONE (−), or negative voltage level, to either line 11, 12, or 13 and a subsequent pulse from a single-shot pulser (see Fig. 15) will cause the chain to operate respectively at 1, ½, or ⅓ the clock rate. The waveforms at the top of Fig. 14 would result from using input 13. The gating used in this example is AND-OR, and operation can best be explained by use of the timing diagram of Fig. 11. Assume that operation is in prograss and one wishes to examine the means by which $CD_1$ is enabled to pulse simultaneously with $CD_4$. Assume from Fig. 11 that $CD_3$ is furnishing output pulses from time base pulse $n$. The $CD_3$ negative pulse serves as the $-G$ to $CD_4$. Line 5, the latch output of $CD_4$, assumes the negative level at the time shown. This negative level ANDS(−) with line 13 and propagates through to establish a ONE(−) on line 4 of $CD_1$. Figure 10 shows that line 4 and the latch output ($X_2$) of $CD_1$ is ANDED(−) such that gate V will pass the next negative pulse furnished by the time base. This is pulse n + 1 and outputs from $CD_1$ and $CD_4$ result in time sync. From Fig. 11 the AND(−), line 3, output could have been delayed up to 8 nanoseconds from points $m$ and $p$. It is assumed that the gating delay was within this interval. Had more than 8 nanoseconds been required then the line 8 output ($X_1$) which occurs $1T$ earlier would have been used, as is illustrated in the next example.

Frequently it is neccessary to design a large, synchronous, and parallel system such that in the process of high-speed operation it is possible to stop

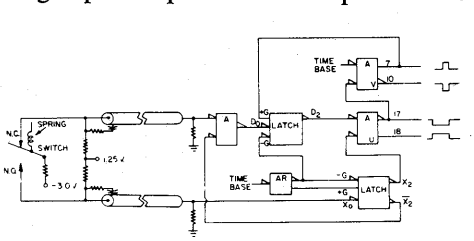



Figure 15. Single shot implementation.

the system and to operate in a manual step mode. Figure 16 shows one method of incorporating the step mode into a large system. The single-shot output sets a series of control delays that are in turn fanned-

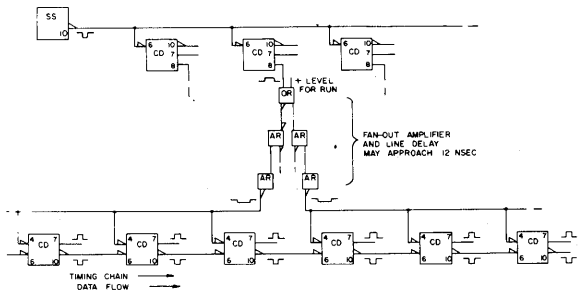



Figure 16. Step mode implementation.

out to other control delays throughout the system. The last level of control delays functions normally unless the line 4 inputs are held in the ZERO(−) state (positive voltage). The control delays receiving set or $-G$ signals from the single-shot furnish ONES(−) at the line 4 inputs for one pulse interval. Since the control delays in the timing chains contain storage the system is enabled to remain dormant for many time cycles and still function correctly for one cycle on a single-shot command.

The three levels of fan-out circuits in Fig. 16 may consume up to approximately 12 nanoseconds. Since the line 4 input to a control delay may be delayed 8 nanoseconds from the time the latch output ($X_2$) occurs and since the complemented latch output ($X_1$) occurs $1T$ earlier, a total of approximately 12 nanoseconds of delay is possible if two provisions are made. First, the $X_1$ must go through a chain containing an odd number of inverters to obtain the logical ONE(−) needed at the line 4 input, and second the $-G$ signal must not have been appreciably delayed in arriving from the single-shot. If still more time is necessary, because of heavy loading or line length, then it would probably be necessary to gate the control-delay $V$ circuit.

A better appreciation of step mode operation is possible by considering the events of Fig. 3. If the pipeline had been stopped after time $t_{n+m}$ then a step command from the single-shot would cause the events associated with time period $t_{n+m+1}$ to occur, but at a much later time. After running for one time cycle the pipeline would again be stopped.

## PIPELINES IN SYSTEMS

Pipelines in systems are tailored to the particular data processing jobs at hand. Both system organization and subsequent implementation are oriented toward doing useful processing at an optimal rate. As information is sequenced through a pipeline process, the register widths, or numbers of parallel registers, may be increased or decreased according to requirements. Information may be extracted from and injected into the process at points along the way. If an arithmetic operation requires more time than gating between registers permits then an attempt might be made to partition that operation over more than one pipeline section. Also, section averaging, where one section expands in time at the expense of its neighbors, is not ruled out when appropriate delays are available to adjust the timing. Pulse widths are altered by standard techniques as needs arise. Many forms of feedback may be incorporated into the structure, but attempts will usually be made to eliminate or minimize any slowdown in the output rate. Lastly, the physical layout for high speed (50 mc) systems will be done so as to obtain as many packaging advantages as possible with structures of the pipeline variety.

## CONCLUSION

Pipeline systems, operation, and specialized implementation areas have been described. Several example solutions to special areas of implementation were presented along with some design considerations to permit an understanding. It is hoped that solutions to common problems may be considered as design tools so that designers do not have to reinvent the wheel, so to speak, for each new system. A majority of the ideas contained herein are quite well established; however, new technology and the compressed time frame has required special treatment. It was no mere coincidence that delay lines were bypassed in favor of circuit solutions in many instances. Integrated circuits are approaching the speed requirements and integrated subsystems are a distinct possibility. Numerical solutions to design considerations were orientated towards the 50 mc systems that will be realized over the next 1 to 3 year time period. As time and technology advance and more complex structures of economical, densely packaged logic are realizable then the possibility may exist that long strings of interacting pipeline logic structures have something to offer in more sophisticated systems of the future.

## ACKNOWLEDGMENT

The author is grateful for the encouragement, help, and many valuable suggestions received from his colleagues in the Department of Defense.

## APPENDIX: GLOSSARY OF SYMBOLS

BYTE: $N$-bit word used for processing.

CD: Control delay.

DOT: A wired-gate decision.

$f$: Logic sense—function
Timing sense—frequency.

$-G$: Negative gating pulse.

$+G$: Positive gating pulse.

$k$–1: Number of circuits between registers.

(–), (+): Negative voltage = ONE,
Positive voltage = ONE.

nsec: Nanosecond, $10^{-9}$ sec.

picofarad: pf, $10^{-12}$ farad.

$r$: Number of bits in a register.

Right triangle (small symbol in logic drawings): Negative voltage = ONE.

$\rho_{12}$: Measure of correlation.

SS: Single shot, or one shot.

$S_c$: Clock skew measured at register inputs.

Slave: See "The Control Relay Cell" above.

$t$: Time.

$T$: Nominal circuit, line, and loading delay.

$T$: Maximum circuit, line and loading delay, unless otherwise specified.

$T$: Minimum circuit, line and loading delay, unless otherwise specified.

$W_c$: Original clock pulse width. Also, nominal final clock pulse width.

$X$: Logic sense—complement of the logic variable $X$.
Timing sense—maximum value of the time variable $X$.

$X$: Minimum value of the time variable $X$.

$X_1$: Latch output available in $1T$.

$X_2$: Latch output available in $2T$.

## REFERENCES

1. M. J. Flynn, "Engineering Aspects of Large, High-Speed Computer Design; Part I—Hardware Aspects," presented at the Office of Naval Research Symposium on High-Speed Computer Hardware, Washington, D.C., Nov. 17-18, 1964.

2. G. M. Amdahl, "The Model 92 as a Member of the System/360 Family," *AFIPS Conference Proceedings,* vol. 26, part II, 1964 Fall Joint Computer Conference, pp. 69-72.

3. E. M. Davis, "The Case for Hybrid Circuits," 1965 *International Solid State Circuits Conference (ISSCC) Digest of Technical Papers,* pp. 32-33 (Feb. 1965).

4. Harding and Schwartz, "An Approach To Low Cost, High Performance Microelectronics," presented at the Western Electronics Show and Convention, Aug. 22, 1963.

5. W. Bucholz, *Planning a Computer System—Project Stretch,* McGraw-Hill Book Co., New York, 1962, p. 204.

6. Pyle, Chavannes and MacIntyre, "A 10 Mc NDRO Biax Memory of 1024 Word, 48 Bit Per Word Capacity," *FJCC, AFIPS,* vol. 26, pp. 69-80 (1964).

7. H. D. Toombs and R. F. Abraham, "A Large High Speed Magnetic Film Memory System," presented at the symposium Les Techniques des Memory Colloque Internationale, Paris, Apr. 5-10, 1965.

8. Crawford et al, "Design Considerations for a 25 Nsec Tunnel Diode Memory," *FJCC,* 1965.

9. Maley and Earle, *The Logic Design of Transistor Digital Computers*, Prentice-Hall, Englewood Cliffs, N. J., 1963, chap. 11.

10. J. E. Freund, *Mathematical Statistics,* Prentice-Hall, Englewood Cliffs, N. J., 1964, p. 176.

11. J. R. Turnbull, "Some Aspects of Digital Circuit Design," *ISSCC Digest of Technical Papers,* p. 58-9, Feb. 1963

12. Narud, Seelbach and Miller, "Relative Merits of Current Mode Logic Implementation," ibid., pp. 104-5.

13. H. S. Yourke, "Millimicrosecond Transistor Current-Switching Circuits," *IRE Transactions on Circuit Theory,* Sept. 1957.

14. *CADY CARDS* (manual of CADY logic circuit family), Department of Defense, Mar. 1963, pp. 35-41.

# HIGH-SPEED LOGIC CIRCUIT CONSIDERATIONS

W. H. Howe
*General Electric Computer Department*
*Phoenix, Arizona*

## INTRODUCTION

This discussion is confined to circuits operating at switching speeds sufficiently fast to require the use of terminated transmission lines for all logic interconnections other than to an adjacent device. The discussion is further confined to significant factors affecting circuit decisions in a high volume commercial/industrial environment. Laboratory curiosities operating at absolute maximum speeds are not considered in view of the extremely distorted economics associated with experimental technologies. The factors under discussion are technology considerations, economic considerations, logic arrays, power dissipation, and packaging media constraints. The discussion is not intended to be a gross prediction of future practice, but rather a snapshot of today's design considerations imposed by present technology and Mother Nature's rather rigid philosophy concerning the speed of light. Since the transmission time through the interconnecting media is significant when compared to propagation delay time of the logic device, the physical size of the system has some bearing on the definition of high speed. This discussion is concerned with relatively large organizations such that a propagation delay time of 2 to 5 nanoseconds may be considered high speed. More dramatic speed improvements may come with machine organizations which consume large amounts of circuits. These organizations are now becoming feasible due to increased reliability and the availability of low cost devices through the semiconductor industry.

## TECHNOLOGY

The selection of a technology for circuit fabrication is heavily dependent upon timing, anticipated volume, cost and required performance. The following sequence of events usually occur in the development of a circuit family.

*Device Availability.* Either as a result of a specific development contract or in the normal course of funded research and development programs, an improvement in mask technology, process sequencing, etc., permits a higher speed device to be fabricated on an experimental basis. Historically, the device has been first implemented as a transistor.

*Circuit Design.* The new device is exploited by the user, as well as the manufacturer, to produce a desirable logic circuit. These circuits are generally different since the user is not aware of all the process constraints nor the economic tradeoffs required

for eventual success in the market place as a microcircuit.

*User Selects Technology.* If the user must exploit his faster circuit as soon as possible, he may choose immediate circuit implementation through the use of one of the hybrid circuit technologies. The penalty for quick turn-around is higher cost in volume. Meanwhile, the semiconductor manufacturer has begun development of a silicon integrated circuit.

*Manufacturer Announces Silicon Integrated Circuit.* The silicon circuit version may prove to be as fast, or faster than the hybrid equivalent since he retains control of all process optimization. The circuit development path is somewhat longer, but is being reduced by the growing tendency on the part of manufacturers *not to disclose advanced technologies until the silicon integrated circuit is well on its way to market.*

Let us, therefore, discuss technology in the light of silicon integrated circuits. Silicon integrated circuit technology has been frequently described as a cure-all for cost. Once, the circuits are in use, volume increases, causing costs to go down, increasing the volume, etc., until the cost extrapolation goes through zero. Some of these effects may be observed in today's market where circuit costs are near one dollar even at modest volumes. At one of the recent conferences, several authors lamented the fact that the longed-for impact of integrated circuits on the computer business simply hadn't happened. A more meaningful statement may well be that the impact of the impact of the computer business on integrated circuits has not yet happened. The economic success of these devices is highly volume-dependent; so much so, in fact, that, to date, commercial computers are the only logical market place for the latent high volume all manufacturers may readily achieve. This relationship has caused a heavy emphasis on logic circuit and logic array development to reduce cost to the fullest extent. Extensive research efforts have been initiated to search for logic arrays which are highly efficient, low in cost, and high in speed to satisfy the needs of the computer industry. However, the tradeoffs in speed, cost, logic complexity, and technology are inherent to the design of *systems* and are not separable in spite of the good intentions of the semiconductor manufacturers or the abstract logicians. We would like to point out briefly some of the tradeoffs available. One of the most interesting and significant paradoxes of the new technology is the apparent reconciliation of a desire to achieve high speed and low cost. The parameters which yield high speed, i.e., low parasitics, small device geometry, also yield lowest ultimate production cost in silicon integrated circuits. Past circuit design practice has equated high speed with high cost. The first step in the assessment of circuit constraints is a careful analysis of the technology used to make circuits and the latent cost significance of the variables.

The following example has been normalized to prevent easy identification of a given semiconductor device. The analysis technique was developed by Mr. W. D. Turner[1] of General Electric and will be explored in more detail in a forthcoming paper. Two characteristic fabrication processes were analyzed and they may be generally described as diode isolation and oxide isolation. Mask technology has a critical effect on cost as will be demonstrated.

Table 1. Circuit Economics.

| | A | B | C | D |
|---|---|---|---|---|
| Isolation | Oxide | Diode | Diode | Diode |
| Tpd. | 4 ns | 3 ns | 2 ns | 5 ns |
| Circuit/chip | 1 | 1 | 2 | 2 |
| Critical area ratio* | 0.296 | 0.460 | 0.256 | 0.40 |
| Relative yield | 3.38 | 2.17 | 3.90 | 2.50 |
| Normalized yield | 0.87 | 0.56 | 1.00 | 0.64 |
| Circuits per wafer | 160 | 218 | 766 | 316 |
| X normalized yield | 139 | 122 | 766 | 202 |
| Wafer cost | 1.00 | 1.00 | 1.00 | 1.00 |
| Oxide isolation premium | 0.25 | — | — | — |
| Relative chip cost $\times 10^{-3}$ | 9 | 8.2 | 1.3 | 5.0 |

Assuming reasonable cost levels for the cost of a wafer and also assuming a reasonable value for absolute yield, one may compute the cost of a chip:

$$\text{Wafer cost} \approx \$50$$
$$\text{Yield} \approx 0.5$$

$$\text{Chip cost} = \textit{Wafer cost} \times \textit{relative chip cost}$$

*Critical area ratio is the result of obtaining the chip area of a given circuit and dividing by the area which is critical to yield. Areas which are critical are those where an oxide fault or misregistration may result in a circuit failure.

| | | Yield | |
|---|---|---|---|
| A | B | C | D |
| $.90 | $0.82 | $0.13 | $0.50 |

Assembly, test and package costs must be estimated to complete the comparison, but for the purpose of this paper, assume a constant $.30 for the sum total of these factors.

| A | B | C | D |
|---|---|---|---|
| $1.20 | $1.12 | $0.43 | $0.80 |

*Cost per circuit (C&D contain 2 circuits per chip)*

| | | | |
|---|---|---|---|
| $1.20 | $1.12 | $0.22 | $0.40 |

These basic costs are marked up in accordance with the profit motive to produce quoted selling price. It is possible, of course, that a comparison of selling price may result in an inversion or distortion of the rank of the products. It is equally possible that a distortion of the basic economics of the process as a result of the battle in the market place may cause slipped schedules, price renegotiations and poor quality parts. Selling price alone is not an adequate parameter.

As a final comment, the smallest device, which was also the fastest, had the least latent cost.

Now it must be recognized that a study of this nature has certain inaccuracies, but it is important that these studies be made to ascertain the inherent speed/cost relationship which may be entirely different from the quoted costs received from semiconductor marketing organizations.

## LOGIC ARRAYS

One factor of growing significance, as circuit size is reduced, is the increasing amount of surface area consumed by areas devoted to interconnections and pads for interconnections. There have been marginal improvements over the past few years, but no startling improvements have been made in comparison to reductions in the basic device geometry. As has been pointed out in many recent papers, the consumption of real estate may be reduced by interconnecting the logic circuits with the narrow lines allowed by the masking technology, thus reducing to a minimum the area requirements for external lead pads. At this point, the semiconductor manufacturer relaxes and says in effect to the computer designer: Reduce your logic to a few standard configurations, and I will reduce costs by a large factor. Hence, we have a search for magic standard logic functions. Other approaches such as varying the final step of the masking process to provide special logic connections over a matrix of logic circuits has been proposed. This has resulted in difficult layout requirements and a challenging problem of computing optimum ratios for connecting leads, logic parting problems, and so forth. Other methods are variations on providing a circuit/logic matrix where bad elements are disrupted and the logic restructed through adjoining elements usually at the expense of speed. All of these approaches seem to neglect the basic overriding economic significance of device geometry.

Figure 1 illustrates the normalized economy of chip size vs array complexity. As can be seen, the smaller the chip, the larger the array which may economically be placed on the chip. With a given mask technology, most economy is achieved by having a high number of chips per wafer which will set definite limits on logic complexity per chip. Most manufacturers are concerned with having a given chip, good or bad, rather than going through complex "rescue" operations involving additional processing even though the metallic interconnection step is relatively inexpensive. The point here is that small size permits high speed with lower costs and that logic arrays are apt to be most effective when they are small, thus producing chip sizes amenable to the economics of a given mask technology. The largest stumbling block then is the logic configuration itself.

Unfortunately, we haven't achieved either a magic logic function or a magic insight into the solution of the problem of finding a relatively small set of standard functions. However, we have carefully analyzed a number of products and have classified the logic groupings obtained in Table 2.

Table 2. Gate Efficiency.

| Group | Gates per package | Efficiency | Loss |
|---|---|---|---|
| 1 (non-functional) | 1-3 | 100% | — |
| 2 (functional) | 4-8 | 96 | 4% |
| 3 (functional) | 46-71 | 87 | 13 |
| 4 (functional) | 106 | 84 | 16 |

Comparing these efficiency figures with cost economy ratios exhibited in Fig. 1 reveals that array technology is economically attractive and war-

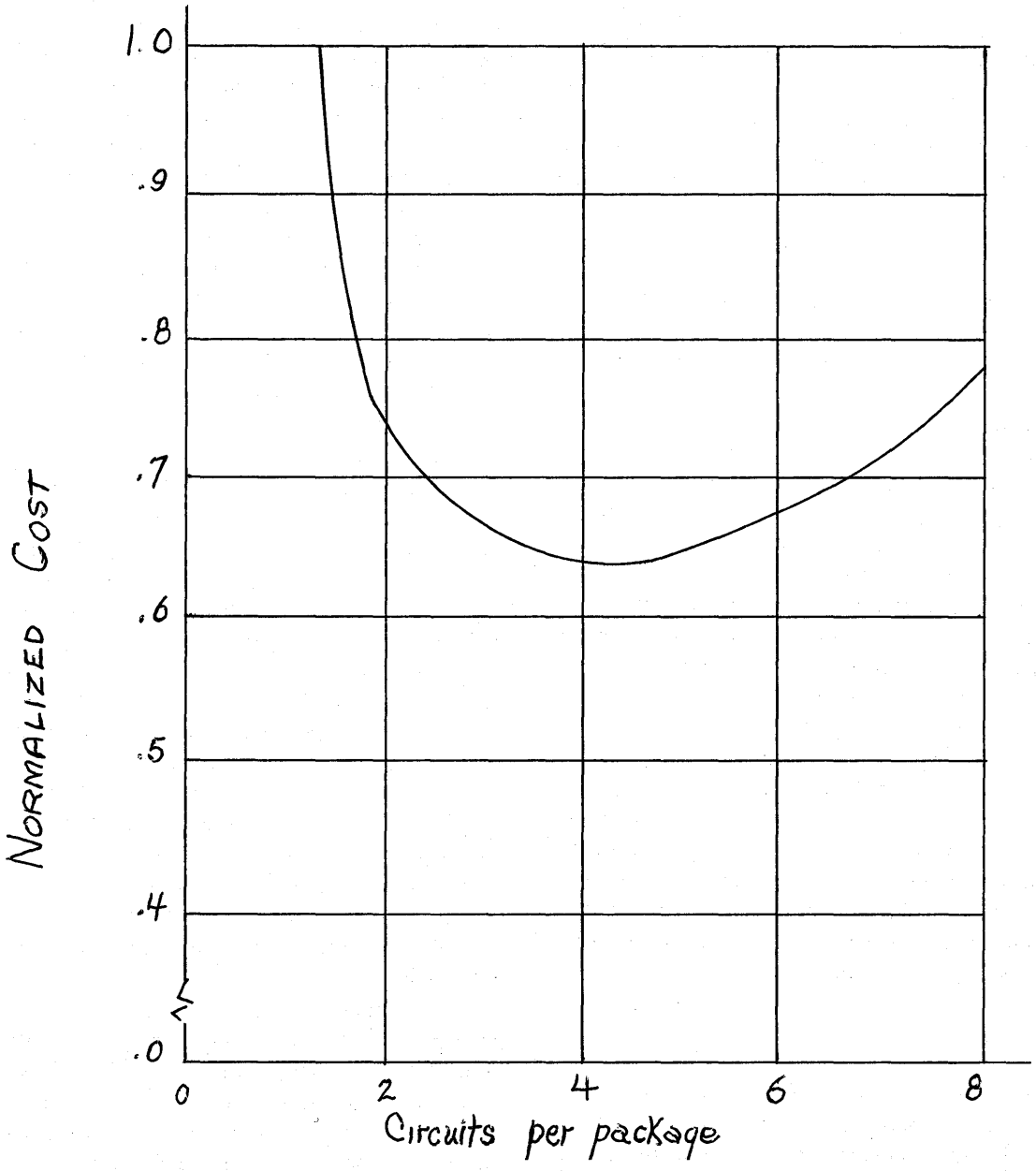



Figure 1

rants extensive development effort. This is even more significant for high-speed circuit applications, since the array permits extremely short runs, unterminated, for a large portion of the logic. It is also interesting to note that these efficiencies were achieved with a relatively small number of different logic configurations. The number of different arrays were, in fact, less than the number of different circuits available in most microelectronic logic sets.

## POWER CONSTRAINTS

The latent difficulty in power dissipation is the prime importance junction temperature rise has on reliability. The logic array is very likely to compound what is already a difficult situation in a normally isolated and mounted single or double circuit chip. Array size is very influential in making a bad situation worse. For example, in a four gate struc-

ture with no internal interconnections, the external leads provide a reasonable path to whatever heat sink is provided. However, many eight and more gate structures contain buried elements which must dissipate through the silicon chip body or through the small area metallic surface connections. The situation is compounded if oxide isolation is used, since the oxide is such an excellent thermal barrier.

Since the array is to be embedded in a transmission line environment, termination resistors must be provided for inputs and literally line drivers provided for outputs. What is generally overlooked is the absence of these requirements for circuits operating in the highly localized environment inside the array. The loads and distances are well known, the geometry is well controlled, crosstalk can be designed to a relatively low value and noise can be locally reduced. In fact, the interior portions *should be* quite different from the exterior portions from the design standpoint. Power dissipation is directly proportional to the transport energy required; hence, internal voltage swings and noise immunity should be reduced to the lowest possible level.

The lead technology will eventually prove to be the most constraining factor for conventional mounting techniques such as the flip chip or bonded wire methods. At present, we do not normally consider making surface connections to reduce the problems associated with power dissipation; but it is apparent that this approach is not too far off.

Consequently, the circuit selected must be capable of operating as a driver of internal logic elements as well as being driven by a very small swing on its inputs. There are a few circuits capable of this type of operation; most level detector designs will function in this fashion and, of course, the popular current mode logic can be designed to produce these desirable characteristics. Since the circuit must operate from, and drive, transmission lines, the swin/power relationships imposed by the transmission line media are of more than a passing interest during the circuit design.

## MEDIA CONSTRAINTS

A common approach to the design problem is to assume the highest possible transmission line impedance to minimize circuit power and reduce heat dissipation. Of the general spectrum of transmission line media available, the most frequently used is the multilayer stripline and in view of its high popularity, the discussion is confined to this type of interconnection scheme. The multilayer structure has a very significant set of economics which must be satisfied to produce acceptable manufacturing costs. Table 3 represents the type of analysis which must be performed to relate transmission line characteristics to the manufacturing capability and to the required run density. Obviously, the transmission media must connect logic; therefore, the economic achievement of density is a prime requisite. As may be imagined, the density and number of layers required to achieve a given logic organization are key factors for attaining reliability, low lamination costs and high overall media yield.

Table 3. Transmission Line Characteristics.

| Impedance *ohms* | Mech. Tolerance $\Delta W$ *mils* | Mech. Tolerance $\Delta t$ *mils* | Mech. Dim. *wn mils* | Mech. Dim. *tn mils* | Cross-talk *percent* | Line Density *runs/in* |
|---|---|---|---|---|---|---|
| 100 ± 10% | ±1 | ±1 | 6 | 12 | 10 | 40 |
| 100 ± 5% | ±1 | ±1 | 11 | 22 | 10 | 20 |
| 100 ± 10% | ±2 | ±2 | 14 | 24 | 10 | 20 |
| 50 ± 10% | ±1 | ±1 | 16.6 | 9 | 7.5 | 30 |
| 50 ± 10% | ±.5 | ±.6 | 9.3 | 5 | 9.2 | 60 |
| 50 ± 10% | ±2 | ±2 | 30.5 | 16.5 | 13 | 20 |

The most important variable in the determination of cost is the mechanical tolerances which must be held in the fabrication of both the signal carrying conductors and the insulating dielectric. With a high quality, tight tolerance process, line density may be significantly increased causing the number of layers to drop rapidly for a given level of crosstalk. A low tolerance process may cause an increase in crosstalk and an increase in the number of layers required to do a given job. In addition to the importance of crosstalk, the absolute impedance variation affects cost since the impedance mismatches in the system cause reflections, which in turn may affect the required noise immunity of the circuits.

Assuming that manufacturing cost is entered into a similar table as a function of tolerance and process variables, a selection that is economic may be made. The cost picture is not yet complete without some provision for logic changes even after the normal debug and prototype cycle. Variable logic must be allowed and it may not be possible to have transmission line impedances which are identical to line impedances used in the multilayer boards. Two pro-

cesses are assumed: transmission lines and perhaps terminated short-run open wires. The circuit selected must be capable then of operating with two different impedances. The overall cost balance achieved is very much a function of the machine size and organization, since maximum flexibility in hardware will almost certainly result in higher costs.

The packaging technique utilized is constrained by the need for allowing full interconnection with stripline, a small but finite number of variable connections, efficient low reflection connectors and some timing problems which will be briefly mentioned. If the machine is a large one, the timing problems are compounded by the fact that the interconnecting line lengths are not the same. In some areas, this may be lumped as a clock skew problem and be adequately solved by control of the clock widths necessary to handle timing/length delays. This is not always efficient and the clock system itself may suffer from the same problem. The packaging system must therefore be designed to accommodate slack to control run length differences. At present, the use of slack to provide a variable degree of delay appears quite feasible in the design and control of some segments of the machine.

## SUMMARY

The variables briefly discussed here are significant in the selection of a logic circuit to be used for a high speed commercial/industrial application. The underriding implication is clearly the achievement of high circuit volume, as is typical in most computer applications, and low cost. The tradeoffs in technology selection have frankly been preoccupied with silicon integrated circuits since in a high volume environment this basic technology has demonstrated definite cost advantages. In an absolute maximum speed environment, constrained by a desire to deliver at the earliest possible time, other hybrid technologies have a slight performance edge. This performance-time gap is rapidly closing, although it will not, in all likelihood, become zero. It is of prime importance that a technology be selected which has a maximum opportunity of achieving low cost. This normally points to excellence in masking tolerances, large production wafers and the absence of process sequences which prohibit the subsequent diffusion of logic arrays. In today's market place, we must plan for a maximum possibility of cost reduction over the life of the product.

Logic arrays are significant in making technological cost reductions possible. If an organization is structured in array fashion, then arrays may be substituted in place as they become available from the suppliers of the technology. This capability may not be quite so effective in a high-speed machine since the operating speed itself causes the arrays to become somewhat different to reduce the effects of power dissipation. The circuit must be capable then of operating over a wide range of drive conditions with power dissipation considered a variable.

The connection media is a strong function of the manufacturing capability to produce precision lines at low cost with commensurate high skill in the art of quality lamination techniques. These manufacturing cost and reliability variables will affect the final nominal impedance choice drastically. To a lesser extent, the variation of impedances will also partially determine noise immunity requirements of the circuits, which in turn affect power dissipation, etc. Impedance reflections do cause a significant noise problem.

While discussing circuit considerations, we have yet to discuss circuit design; and I think it proper to avoid this subject. Of all the factors influencing the selection of a circuit, the design variable is for all practical purposes one and the same with the technology. After all, the design which has the least number of components and the highest component tolerance immunity, coupled with speeds high as compared to basic device Ft, has always been a good choice. The difference today is that the good choice now has become the economic choice as a result of the continuing advances of the semiconductor industry.

## ACKNOWLEDGMENTS

The extensive studies and evaluation analysis, which have caused these conclusions to be reached, were performed by C. A. Combs, W. H. Herndon, J. W. Tarzwell, and W. D. Turner, all of the General Electric Computer Department, Phoenix, Arizona. The author is grateful for their helpful comments in the preparation of this paper.

## REFERENCES

1. W. D. Turner, "An Analysis of the Effects of Technology on Integrated Circuit Shop Costs," G. E. Computer Department internal paper.

# CROSSTALK AND REFLECTIONS IN HIGH-SPEED DIGITAL SYSTEMS

A. Feller, H. R. Kaupp and J. J. Digiacomo
*Radio Corporation of America*
*Computer Advanced Product Research*
*Camden, New Jersey*

## INTRODUCTION

Present state-of-the-art computer circuits have signal transition time and propogation delays on the same order of magnitude as the interconnection delay between them. At these speeds it is mandatory that the transmission line properties of the interconnection media be considered. This permits the designer of high-speed digital equipment to include the effects of signal reflection on his system.

Until recently crosstalk in digital systems was primarily a problem encountered in bundled wires between racks or between distant equipments. With the dramatic increase in circuit speeds, combined with denser packaging, crosstalk problems have spread to the backplane wiring, circuit boards and circuit modules themselves. The availability of high-speed integrated circuits aggravates these problems.

In this paper crosstalk and reflections are discussed from the digital designer's point of view. A qualitative description of the nature of the crosstalk signals on pulse-driven interconnecting lines is presented. This is followed by the general expression for the transient crosstalk signal voltage between lines having a nonhomogeneous medium. The detailed derivation is included in the Appendix. Reflections are also discussed from the digital point of view. A criterion is suggested for rapidly determining whether a given interconnection network should be treated as a distributed or lumped circuit. The relationship between reflections and the signal waveform characteristics, the line lengths, the line impedance, the number of loads and the spacing of the loads is discussed. Expressions permitting quick quantitative estimations of the maximum reflections are presented and illustrated.

## CROSSTALK

### *Qualitative Description*

Consider a pulse propagating on one of two parallel transmission lines as shown in Fig. 1. First only the incremental effects at point $x$ will be examined. The capacitance and inductive mutual coupling between the two lines are represented by $C_m$ and $L_m$ respectively. The arrival of the pulse at point $x$ on the driver line causes a capacitance current, $i_c$, to flow in the sense line. This current divides into two equal and opposite currents which propagate toward each end of the line. Simultaneously, by Lenz's law, an inductive current $i_L$ will be induced in the sense line. Both $i_L$ and $i_c$ are proportional to the derivative of the driving function.

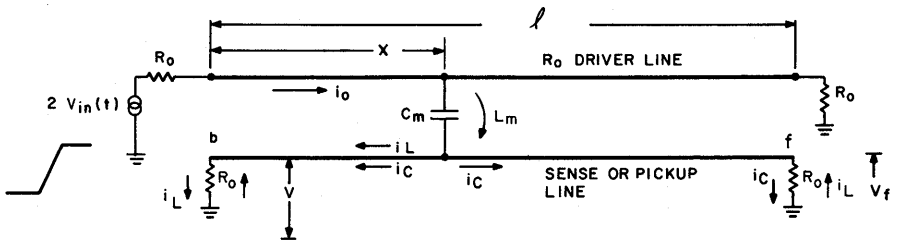



Figure 1. Polarities of instantaneous induced currents.

Several immediate observations can now be made. The currents $i_L$ and $i_C$ at terminal $b$ of the sense line are additive and produce a terminal voltage which is of the same polarity as the driving function $V_{in}(t)$. In addition, since terminal $b$ continues to receive current in a time-sequential fashion as the driving function propagates along the driver line, the voltage pulse formed at $b$, $V_b$, will be a time integral of the incremental currents received at $b$. This will be identified as the backward crosstalk voltage.

At terminal $f$ the capacitance and inductive currents are in opposition. For a homogeneous medium the resultant voltage produced at terminal $f$ of the sense line will be zero.[1] This is the case generally discussed in the literature in connection with the theory of transmission line directional couplers.[2,3] For the case of interest here, transmission lines of nonhomogeneous mediums such as printed transmission lines, the opposing currents will not produce total cancellation at terminal $f$. This resultant voltage at $f$, $V_f$, will be identified as the forward crosstalk voltage.

Further insight into the nature of $V_b$ and $V_f$ and their relationship to the characteristics of the driving signal and to the length of the lines can be obtained by referrering to Fig. 2. The expression, "length of the lines," refers to the region of interaction between the two lines.

A pulse is applied at $g$ as shown. The pulse propagates at some velocity $vp$ toward the receiving end of the line, $r$. $T_d$ is the time required for the signal to propagate the full length of the line. It is assumed that both lines are lossless and terminated in their characteristic impedance, and that the coupling is loose enough that there is negligible degradation of the driving signal as it propagates the full length of the line. These assumptions generally hold in practical applications if the line lengths are not excessively long.

Let us consider that instant of time when the input pulse is at $x_1$. Two rectangular pulses, proportional to the derivative of $V_{in}(t)$ on the driver line, will be induced on the sense line at $x_1$. One of these pulse, $Vb_1$, is a function of the sum of $i_L$ and $i_C$ and will propagate toward terminal b of the sense line. The other pulse, $V_{f1}$, is the result of the difference of $i_L$ and $i_C$ and will propagate toward $f$, the receiving end of the sense line. For this illustration it is assumed that $i_L > i_C$ resulting in $V_{f1}$ being negative. At a time $\tau$ later, $V_{in}$ has moved to $x_2$, inducing corresponding pulses $V_{b2}$ and $V_{f2}$. Simultaneously, $V_{f1}$ has also propagated to $x_2$. Therefore, $V_{f1}$ and $V_{f2}$ exist at $x_2$ at the same instant of time and add directly. This is a continuing process (at a incremental level) until $V_{in}$ arrives and is absorbed at terminal $r$. Therefore, the induced voltage seen at terminal $f$ will be a direct function of the length of the line. Hence, $V_f$ is not only proportional to the derivative of the driving function $V_{in}$, it is also proportional to the length of line over which the interaction exists.

Now the incremental backcross pulse will be examined. Referring again to Fig. 2, note that during the time interval $\tau$, when $V_{in}$ moved from $x_1$ to $x_2$ in the direction of $r$, $V_{b1}$ traversed an identical distance toward $b$. At any instant of time $V_{b1}$ and $V_{b2}$ are $2\tau$ seconds apart. Consequently a continuing series of pulses, each one delayed by a differential time, will arrive at $b$. The last pulse will arrive $2T_d$ seconds after the time of initial application of $V_{in}$. Therefore, the resultant backward pulse at $b$ will start to decay at $2T_d$ seconds. This then defines the width of the backward pulse. Produced by a positive going wavefront it is $2T_d$ seconds wide, or twice the

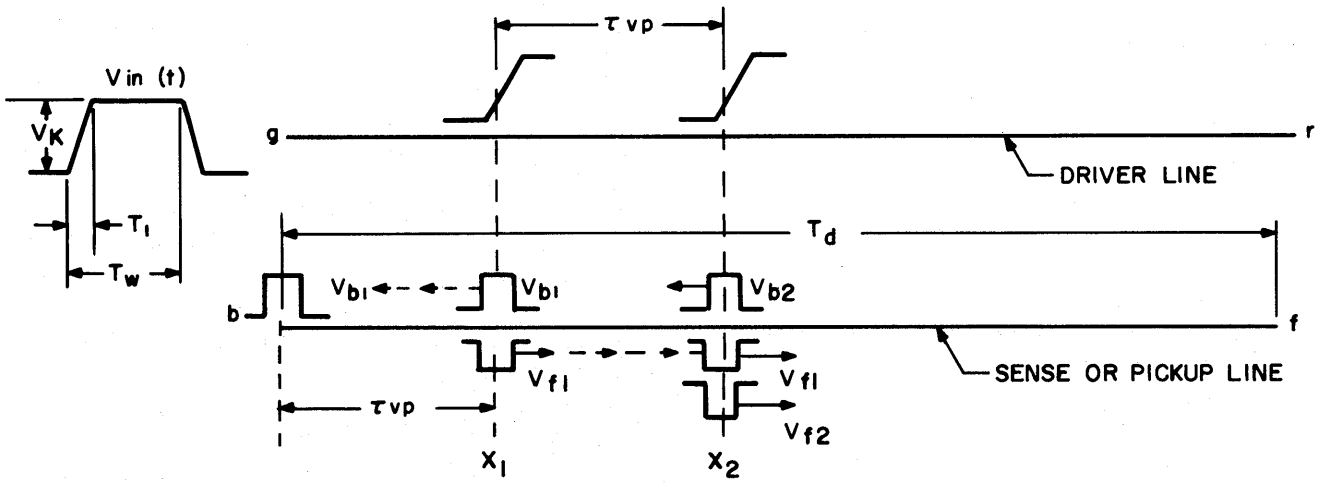



Figure 2. Forward and backward traveling waves.

propagation time of the coupled section of the lines. The resultant amplitude of $V_b$ will be independent of time provided $2T_d$ is greater than the rise time of $V_{in}$, $T_1$. The backward pulse is an attenuated replica of $V_{in}$ for $t$ less than $2T_d$. This can be demonstrated, for the driving signal of Fig. 2, by summing several rectangular pulses $T_1$ seconds wide that are incrementally displaced along the time axis.

It should be emphasized that the amplitude of $V_b$ is independent of the slope of the input signal (for $T_1 < 2T_d$) and the length of the line, while $V_f$ is a direct function of these parameters. Thus, for the case that $T_1 < 2T_d$ the forward and backward induced pulses on the sense line have decidedly different characteristics and sensitivities. By altering the geometrical cross section and permittivity of the dielectric (i.e., its nonhomogeneous characteristics) it is possible to make $V_f$ positive or negative. As was pointed out, in the homogeneous case, $V_f$ will be zero. However $V_b$ will always be of the same polarity as $V_{in}$. Again, this assumes the lines are terminated in their characteristic impedance.

*Crosstalk Equations*

The general expression for the instantaneous voltage induced anywhere on a line which is coupled to another line driven by a signal $V_{in}(t)$ is given by Eq. (1):

$$V(x,t) = Kf\,x\,\frac{d}{dt}\left[V_{in}\left(t-\frac{Td\,x}{e}\right)\right] + Kb\left[V_{in}\left(t-\frac{Td\,x}{l}\right) - V_{in}\left(t-2Td+\frac{Td\,x}{l}\right)\right] \quad (1)$$

where $Kf$ = forward crosstalk constant $= -1/2\left(\frac{Lm}{zo} - Cm\ Zo\right)$,

$kb$ = back crosstalk constant $= \frac{1}{4Td}\left(\frac{Lm}{Zo} + Cm\ Zo\right)$,

$\ell$ = physical length of the coupled region, and

$Td$ = time for signal to propogate a distance $\ell$.

The complete derivation of Eq. 1 is given in the Appendix. As indicated in Fig. 3, $V(o,t)$ is the back crosstalk voltage while $V(l,t)$ is the forward crosstalk voltage. The two crosstalk constants are functions of the geometry and materials of the coupled lines. Each of these constants can be determined by a single measurement. With the determination of these constants the crosstalk voltage at any point on the coupled sense line can be evaluated by Eq. (1) for any line length and for any amplitude pulse with a linear rise time. The forward and back crosstalk will now be more closely examined.

*Back Crosstalk*

At $x = 0$, the beginning of the coupled line, the back crosstalk voltage is

$$V(0,t) = Kb\,[Vo(t) - Vo(t-2Td)] \quad (2)$$

To better understand the relationship of the back crosstalk pulse waveshape to the driving signal waveform let $Vo(t)$ be the function shown in Fig. 4, that is,

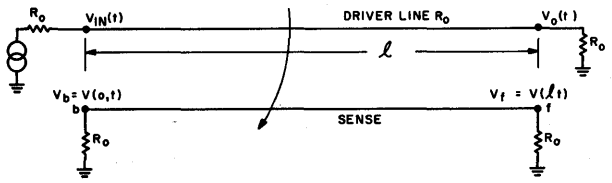



Figure 3. Coupled transmission line.

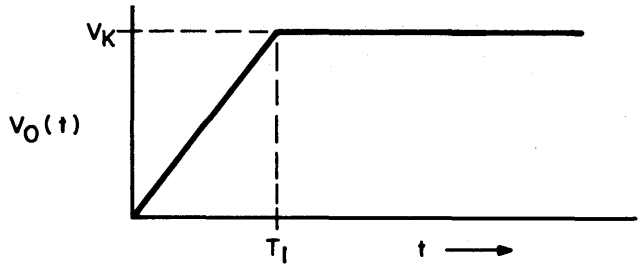



Figure 4. Input driving function.

$$Vo(t) - fo(t-T_1) \qquad (3)$$

where $fo(t) = \frac{Vk}{T_1} t.$

Substituting Eq. (3) into Eq. (2), the back crosstalk voltage is

$$V(o,t) = K_b[fo(t) - fo(t-T_1) - fo(t-2T_d) + fo(t-T_1-2T_d)] \qquad (4)$$

*Long Line Case.* Assume initially that the rise time $T_1$ is less than twice the propagation time of the region of interaction. This situation defines a long line. For this case the plus waveform described in Eq. (4) is as indicated in Fig. 5. Note that the back crosstalk voltage is directly proportional to the input driving voltage $Vo(t)$. In addition, the maximum amplitude, $K_bV_k$, is independent of the coupled length. As discussed in the qualitative analysis, the backcross pulse width is equal to twice the propagation delay of the line.

*Short Line Case.* If the rise time of the input driving function is greater than twice the propagation time of the coupled lines, the situation is defined as the "short line." For this case the back crosstalk pulse, still described by Eq. (4), is graphically shown in Fig. 6. The maximum amplitude occurs at $t = 2T_d$ and from Eq. (4), the maximum back crosstalk voltage is

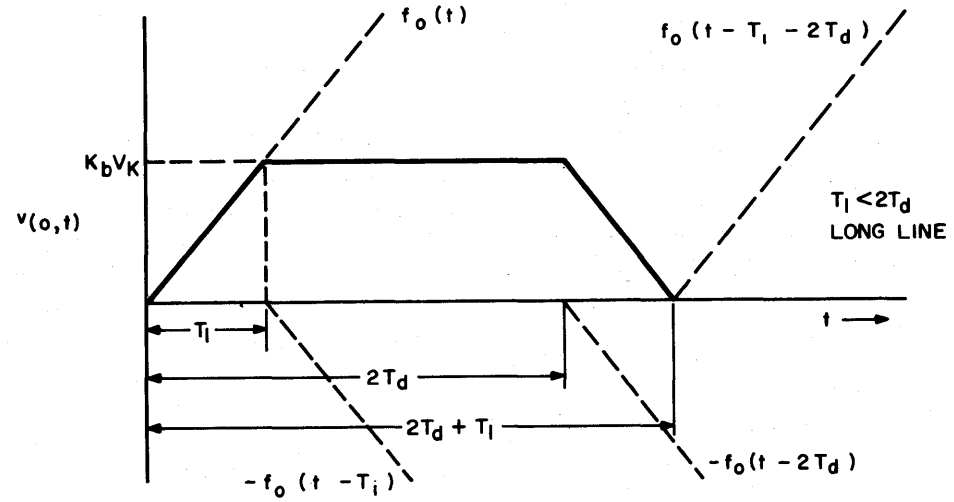



Figure 5. Back crosstalk pulse in long line case $T_1 < 2T_d$.

$$V(0,t)_{\max} = \frac{2K_b V_K T_d}{T_1} = 2K_b T_d \frac{d}{dt}(V_0(t)) \qquad (5)$$

Here, as opposed to the long line, the back crosstalk voltage is proportional to the slope of the driving function and the electrical length of the coupled line. Note that the maximum amplitude is decreased from that of the long line by the factor $2T_d/T_1$. The duration of the backward pulse for the short line is $T_1$ as indicated by Fig. 6.

*Forward Crostalk*

Now consider the crosstalk at the other end of the coupled line, $X = \ell$ in Fig. 1. This forward crosstalk, in response to the input driving function, $Vo(t)$, of Fig. 4, is

$$V(\ell,t) = K_f \ell \left[\frac{d}{dt} f_0(t-T_d) - \frac{d}{dt} f_0(t-T_d-T_1)\right] \qquad (6)$$

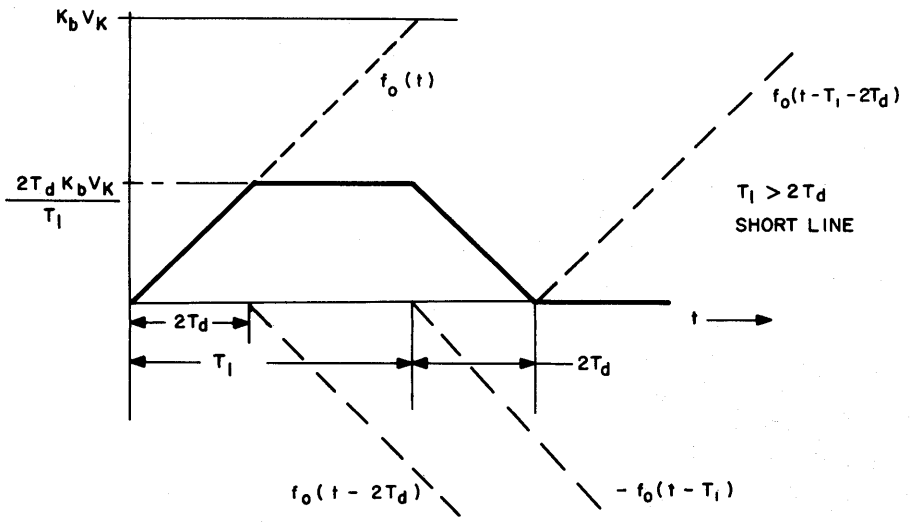



Figure 6. Back crosstalk pulse in short line case $T_1 > 2T_d$.

$$= 0 \text{ for } 0 < t < T_d$$

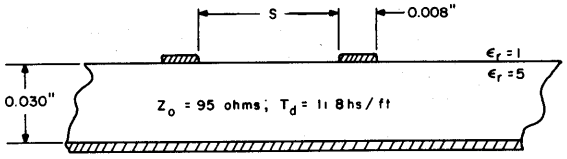
$$= \frac{K_f V_{k1}}{T_1} \text{ for } T_d < t < T_d + T_1$$

$$= 0 \text{ for } t > T_d + T_1$$

Equation (6) yields a rectangular waveform. The amplitude of the forward crosspulse, $V(\ell, t)$, is proportional to the slope $(V_k/T_1)$ of the driving signal and the length of the coupled line. The pulse width is equal to the rise of the driving signal.

The polarity of the forward crosstalk signal depends on the sign of the forward crosstalk constant $Kf$. If $L_m > C_m Z_0^2$ the forward crosstalk pulse will have a polarity opposite to that of the driving signal. If $L_m < C_m Z_0^2$ it will have the same polarity. Only in the case of a homogeneous medium of propagation does $L_m = C_m Z_0^2$.

*Experimental Waveforms.* An example of crosstalk in a nonhomogeneous medium can be demonstrated with printed microstrip transmission lines. Measurements were made on two sets of lines with the cross-sectional geometry shown in Fig. 7. The setup is as shown in Fig. 3, with the line length, $\ell$, equal to 16 inches. The waveforms are shown in Fig. 8 for the ines spaced 0.120 inches apart and in Fig. 9 for a 0.015-inch spacing.




Note: Lines and ground plane are approximately 0.003-inch thick.

Figure 7. Cross section of microstrip transmission line. Lines and ground plate are approximately 0.003-inch thick.

On both Fig. 8 and Fig. 9 the forward crosstalk pulse is of the opposite polarity to the driving signal. This occurs because the ratio of self to mutual capacitance is greater than in the comparable homogeneous case. As previously described, if the dielectric medium was homogeneous, the forward crosspulse would be zero.

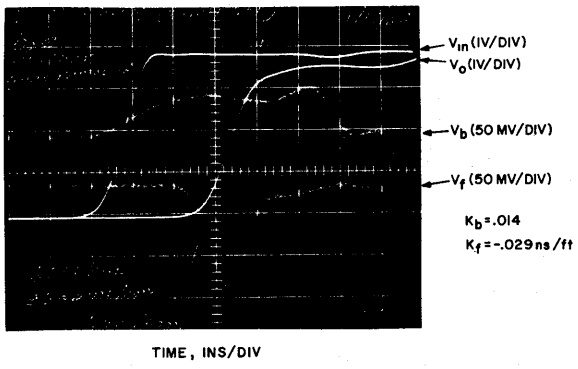



Figure 8. Crosstalk waveforms of Fig. 7: $S = 0.12$ inch, $l = 16$ inch.

From the waveforms of Figs. 8 and 9 the crosstalk constants $K_b$ and $K_f$ can be determined by use of Eq. (1). In determining these constants it is better to use the output voltage rather than the input voltage of the driven line. This is true because the output of the drive line and the crosspulses are subject to line losses and distortion. Thus, to obtain $K_b$, take the ratio of $V_o$ to $V_b$; to obtain $K_f$ take the ratio of

the maximum value of $V_f$ to the maximum slope of $V_{out}$ for a given line length. For the loose spacing of Fig. 8 the backcross talk constant, $K_b$, is 0.014, while the forward crosstalk constant, $K_f$, is 0.029 nanoseconds per foot. With the tightly spaced lines of Fig. 9 $K_b$ is 0.16 and $K_f$ is 0.09 nanoseconds per foot. These constants can now be used to predict the crosstalk waveforms for an arbitrary length of coupled line (of the given crossectional geometry) and any driving signal waveform.

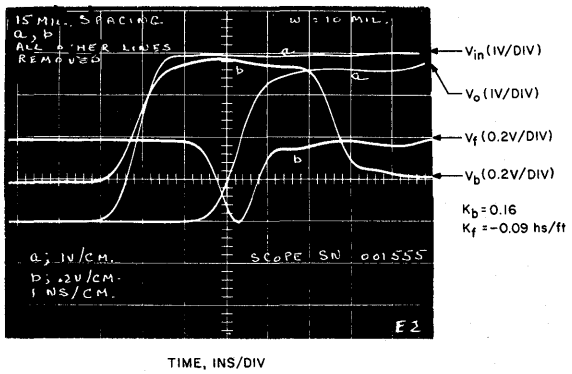



Figure 9. Crosstalk waveforms of Fig. 7: $S = 0.015$ inch, $l = 16$ inch.

## REFLECTIONS

### *Definition—Long and Short Lines*

When a pulse propagates along a transmission line that is not terminated in its characteristic impedance, a signal of amplitude no greater than the original will be reflected toward the point of original incidence. Of special interest in high-speed digital processors is the question of when is it necessary to treat an interconnection path as a transmission line or simply as a lumped element. Although the exact criterion is an exact function of various parameters, a useful criterion can be evolved using the following development.

In the discussions to follow, the high-speed logic gate loads are represented by lumped capacity. Many of the very high-speed logic circuits have input transistors which are operated in the linear mode resulting in input characteristics which are essentially capacitive. In Fig. 10a, $Td$ is the time required for the signal to propagate the length of line. If $T_1$, the rise time of the input signal $E_{in}$, is less that 2 $Td$, the waveform at the sending end of the line will be as shown in Fig. 10b. The reflection signal, due to the capacitor termination, will arrive at the sending end after the input signal has completed its transition. Those signal interconnecting lines that exhibit this property will be defined as "long lines" and will be treated as transmission lines.

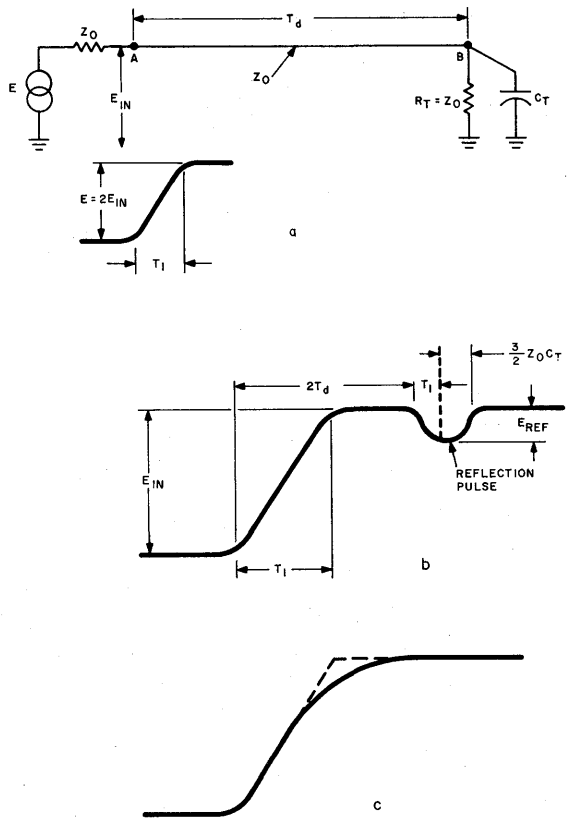



Figure 10. *a.* Transmission line terminated with capacitor and resistor. *b.* Signal at point $A$ for a long line ($T_1 < 2T_d$). *c.* Signal at point $A$ for a short line ($T_1 > 2T_d$).

Conversely, if $T_1$ is greater than 2 $Td$ the effect of the reflection is to degrade the rise time of the input signal as demonstrated in Fig. 10c. Signal lines that demonstrate this property are defined as short lines. Such lines can be treated as lumped capacitances.

As will be demonstrated shortly, a given transmission line, loaded with randomly spaced logic gates, can exhibit the properties of both long and short lines. In these cases, the short and long line theories can be separately applied to ascertain the overall effect.

## REFLECTION FORMULA

Although there are rigorous methods using computer programs for analyzing the waveforms on a transmission line loaded with capacitors, these problems can be analyzed by the novel application of distributed line theory. This technique is an approximate method which yields accurate results for the heavily loaded line, which for the most part, represents the limiting criteria insofar as reflections are concerned. Requiring only a few minutes of the designer's time, this method provides him with further insight and understanding into the nature of the reflection phenomena.

This method of analysis is best illustrated by an example. Consider Fig. 11, where a line having a characteristic impedance of 74 ohms to two 100-ohm lines. Assume that each 100-ohm line pair has a propagation time $Td$ and the 74-ohm line pair has a propagation time $T_2$. If a pulse with rise time $T_1$ is applied as shown in Fig. 11 the waveform at $A$ will be as shown in Fig. 12. Thus, using the standard definition for the reflection coefficient $\Gamma$ yields

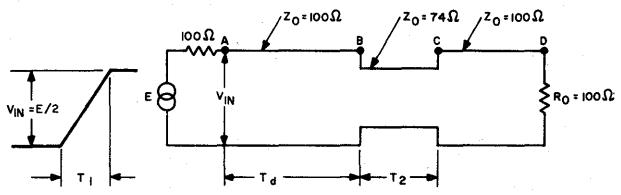



Figure 11. 74-ohm line driven and terminated by 100-ohm line.

$$\Gamma = \frac{Z_L - Z_0}{Z + Z_0} = -0.15$$

Consequently the mismatch between the 100-ohm and the 74-ohm transmission lines produces a 15 percent reflection. Note that the reflection at point $B$ is negative while that produced at point C is positive.

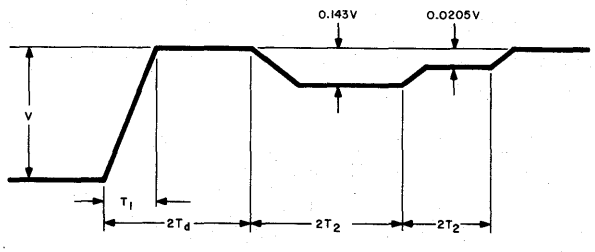



Figure 12. Waveform seen at point $A$ in Fig. 11.

Now consider the 100-ohm line alone. Assume that the signal propagation velocity is 0.5 feet per nanosecond, which is approximately the case for a microstrip line. The distributed inductance and capacitance per unit length of line can be derived from the well known expressions

$$v_p = \frac{1}{(LoCo)^{1/2}} \tag{7}$$

and

$$Zo = \left(\frac{Lo}{Co}\right)^{1/2} \tag{8}$$

where

$v_p$ = velocity of signal propagation,
$Lo$ = distributed inductance per unit length,
$Co$ = distributed capacity per unit length, and
$Zo$ = characteristic impedance.

Substituting in Eq. (7) and (8), the distributed parameters for this particular 100-ohm line are

$$Co = 20.3 \text{ picofarads per foot}$$

and

$$Lo = 0.203 \text{ microhenrys per foot}$$

Now assume that 17.5 pF is uniformly added to line segment $B$-$C$ as indicated in Fig. 13. The distributed inductance remains the same. However the distributed capacitance is increased from 20.3 picofarads per foot to 37.8 picofarads per foot. Then from Eqs. (7) and (8) the impedance of this section is reduced to 75 ohms, and the propagation time is increased from 2 to 2.72 nanoseconds. The waveform seen at $A$ in Fig. 13 would be that of Fig. 12. The amplitude of the reflection would be 0.15 and the width of the reflected signal would be twice propagation delay of the line section $B$-$C$ or 5.44 nanoseconds.

Now, a discrete number of capacitors placed on this line segment can be approximated by a distrib-

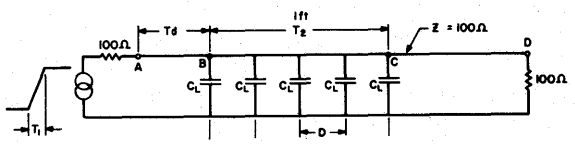



Figure 13. Simulated distributed line.

uted line whose characteristic impedance is to be a function of the spacing between the capacitors. The closer the spacing, the lower the characteristic impedance.

The reflection on the loaded line segment in Fig. 13 can be determined as follows:

Let $Zo$ = characteristic impedance of unloaded line
$Co$ = capacitance per unit length of unloaded line
$Lo$ = inductance per unit length of unloaded line
$v_0$ = velocity of light – in free space
$v_p = Kv_0$ = velocity of signal in unloaded line
$C_L$ = capacitance per unit load
$n$ = number of unit loads per unit length
$D$ = physical spacing between loads
$C_T = Co + C_L$ = total capacity of loaded line per unit length
$E_{ref}$ = maximum amplitude of reflected signal
$E_{in}$ = incident voltage
$\Gamma p = \frac{E_{ref}}{E_{in}}$

$\Gamma p$ is used to define the ratio of the peak amplitude of the reflected signal to the incident voltage when the loads on the transmission line have capacitive components. When the loads are purely resistive $\Gamma p = \Gamma$.

The impedance of the loaded line segment is:

$$Z_1 = \left(\frac{Lo}{C_T}\right)^{1/2} \tag{9}$$

The percent voltage reflected from point $B$ back toward the generator end of the line $A$ is

$$\Gamma = \frac{Z_1 - Zo}{Z_1 + Zo} \tag{10}$$

Substituting Eqs. (7), (8), and (9) into Eq. (10), we get

$$\Gamma = \frac{1 - \left(1 - \frac{\eta C_L}{Co}\right)^{1/2}}{1 + \left(1 + \frac{\eta C_L}{Co}\right)^{1/2}} \tag{11}$$

Equation (11) gives the reflection coefficient in terms of the distributed capacity of the unloaded line and the added capacity (per unit length) due to shunt loads. Note in Eq. (11) that the total added capacity per unit length appears as the product of $\eta$ and $C_L$. Thus, the number of loads and the capacitance of the unit load $C_L$ can be inversely altered without changing the magnitude of the reflected voltage.

Since, in general, the capacitance of the load is fixed by the logic circuit input characteristics, the number of loads per unit length $\eta$ will determine the reflected signal for a given $Co$, or more generally a given $Zo$. A more useful form of Eq. 11 is

$$\Gamma = \frac{1 - \left(1 + \frac{C_L}{DCo}\right)^{1/2}}{1 + \left(1 + \frac{C_L}{DCo}\right)^{1/2}} \tag{12}$$

where $D$ is the physical spacing between loads.

For a given transmission line, Eq. 12 reflects the reflected signal to the load capacitance and the spacing between the loads. Conversely for a specified load and a predetermined maximum tolerable reflection the spacing between equally spaced loads can be determined. It is important to remember that Eq. 12 is most accurate for a large number of loads. The predicted reflections are pessimistic when only a few loads are considered. However, the system must be able to perform properly in the presence of the maximum reflections; thus, it is for many loads that accuracy is required.

Some indication of the relationship between signal reflections and loads spacing can be seen in Fig. 14 where $\Gamma$ is plotted as a function $D$ for $Co$ equal to $C_L$.

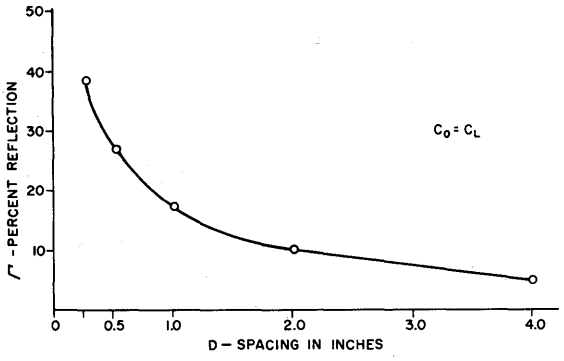



Figure 14. Percent reflection vs spacing for $C_0 = C_L$.

As an illustration of the usefulness of these concepts assume a 100-ohm line with a velocity propagation factor of ½ is loaded with circuits whose input characteristics can be represented by a 5-picofarad capacitor. If the reflection is not to exceed 15 percent the closest allowable spacing can be determined by solving Eq. (12) for $D$. That is,

$$-0.15 = \frac{1 - \left(1 + \frac{5}{D20.3}\right)^{1/2}}{1 + \left(1 + \frac{5}{D20.3}\right)^{1/2}}$$

where $Co$ = 20.3 pf/ft from Eqs. (8) and (9). On

solving the above equation, $D$ is found to be 3.56 inches.

*Short Line*

In the discussion following Fig. 10 a line was defined as short when the driving signal transition time is greater than twice the propagation delay of the line ($T_1 \geq 2T_d$). This definition applies to the idealized trapezodial waveform. In most applications, however, the input waveform will not have a linear rise time but an exponential one. Thus the end of the transition period is difficult to define. Experimentally, it has been found that if $T_1$ is taken as the time required for the source voltage to rise to about 85 percent of its ultimate value, the relation $T_1 > 2T_d$ can stil be used to define the short line.

For those interconnecting lines to which the "short line" criterion applies, the rise times, delays and waveforms are calculated directly by conventional lumped component techniques. In this case the transmission line is represented by an equivalent lumped capacitance as determined by Eqs. (7) and (8). This equivalent capacitance is added to the capacitance presented by the loads to determine the total capacitance.

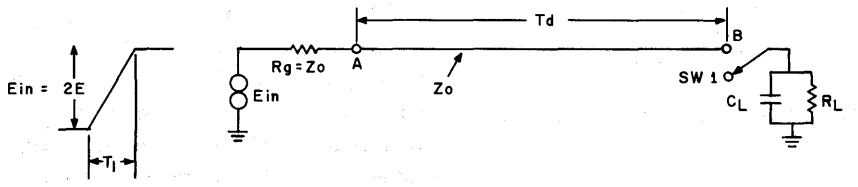
*Single Loads or Widely Spaced Loads*

For those cases where the line is loaded with a single load or with widely spaced loads, the reflection amplitude and the time delay can be calculated by the application of the thevinin theorem. Consider Fig. 15. If an input signal $E_{in} = 2E$ is connected as shown, a signal of amplitude $E$ volts will propagate along the line. $Td$ is the time required for the signal to propagate the length of the line. If $SW$ 1 is open, thus open-circuiting the line, the amplitude of the incident signal will jump to $2E$ which can be considered the equivalent thevinin open circuit voltage. The thevinin impedance is obtained by short-circuiting the input voltage and determining the impedance looking into the $B$ terminal towards the generator. This impedance is the characteristic impedance (except for low frequencies). Thus, the equivalent circuits of Fig. 15b will give a complete and accurate description of the reflection along the line as well as the waveform at the load. If $Rg \neq Zo$, then multiple reflections will exist. In this case the equivalent circuits shown in Fig. 15b apply only in the interval equal to $2Td$ seconds. A new thevinin voltage is determined for succeeding $2Td$ interval based upon the reflected voltage from the generator end of the line. This reflected voltage is determined by the same method, i.e., apply thevinin theorem to the generator end of the line. For short lines ($T1 > 2Td$) the equivalent circuits of Fig 15b still apply provided that the capacity of the line is added to $C_L$.

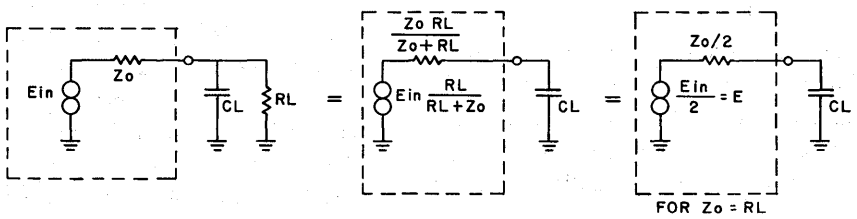
Using the above approach, the reflection voltage is given by

$$E_{REF}(t) = \frac{-EZoC_L}{2T_1}\left(1 - e^{\frac{-2t}{ZoC_L}}\right) \quad 0 < t < T_1 \tag{13}$$

$$E_{REF}(t) = \frac{EZoC_L}{2T_1} e^{\frac{-2t}{ZoC}}\left(1 - e^{\frac{2T_1}{ZoC_L}}\right) \quad T_1 < t < \infty \tag{14}$$




a. Transmission Line




b. Thevenin Equivalents

Figure 15. Thevenin equivalent circuits of transmission lines.

The maximum percent reflection is given by:

$$\Gamma p = \frac{-Zo\,C_L}{2\,T_1}\left(1 - e - \frac{2\,T_1}{ZoC}\right)$$

The time delay from $A$ to $B$ as measured at the 50 percent points is

$Td + (ZoC_L/2)$.

*Reflections Due to Shunt Resistive Components*

Because some logic circuits such as DTL's have nonlinear input characteristics and signficant d-c input components, it is extremely difficult to define the input impedance in terms suitable for use in the reflection formula. Defining *Rin* of the gate as the ratio of the voltage change to the current change is inaccurate since the total current variation usually occurs in some interval of the voltage swing. This assumes that the voltage swing reflects noise immunity properties which is generally the case. However, since the reflection on a line is proportional to the shunt load current, it is possible to define the percent reflection directly in terms of the input current level change which can be easily measured.

Consider Fig. 16 which shows a section of transmission line in which an incident current $I$ *in* is applied. The line is loaded at point $P_1$ by $Z_L$ through which a current $I_L$ flows.

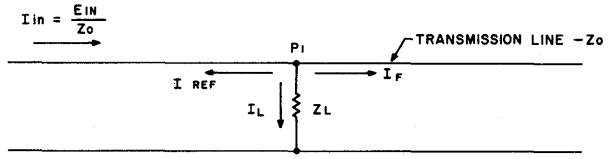



Figure 16. Current distribution at tapped node.

Summing up the currents at node $P_1$ and using the definition $\Gamma_p = \frac{\overline{E_{REF}}}{E\ in}$ we obtain

$$\Gamma_p = -\tfrac{1}{2}\,\frac{I_L}{I\ in} \qquad (16)$$

Thus we have the reflection expressed in terms of the load current and the incident current, both readily determined parameters for DTL type of loads. Although Eq. (16) was derived for resistive loads its application can be extended to capacitive loads by representing $I_L$ by $C\left(\frac{dv}{dt}\right)^{\max}$.

*Experimental Results*

Table 1 shows the comparison between the predicted percent reflections and the measured percent reflections for various capacitance loads and spacings. The table is incomplete simply because all combinations were not measured. As indicated earlier the accuracy between the predicted and observed reflections for the higher numbered loads is within a few percent. As the number of loads decrease (for the closely spaced loads) the predicted value exceeds the observed value. For slower rise times the accuracy of Eq. (12) will maintain itself for the higher numbered loads whereas the difference between the predicted and observed reflections will increase with the fewer numbered loads. This follows from the fact that a lumped line more closely approximates a distributed line as the number of sections are increased.

## DESIGN CONSIDERATIONS

In general the designer will initially endeavor to first specify the characteristics of the transmission line. This happens because the signal transmission system, being an integral part of the packaging, has many physical, mechanical and cost considerations in addition to its electrical properties. Certain types of lines, like coaxial lines, will be eliminated from general use because of their cost and poor package density. Once a line has been chosen consistent with cost and packaging considerations, the natural tendency will be to select as high an impedance as is practical in order to reduce the current requirement of the driver circuitry. However, unless the dynamic and static input current requirements of the logic circuitry is exceedly low, the reflections will be excessive. Excessive reflections in this case means reflection voltages in excess of the predetermined noise immunity of the circuitry. Even if the designer takes corrective steps by lowering the line impedance or increasing the noise immunity of the circuit (which generally increases the delay) it must still satisfy the requirements of the logic designer whose fan out and fan in requirements directly affect the reflections. Then still to be considered is the inherent conflict of increased package density versus higher reflections since squeezing a fixed number of loads close together increases the reflections as indicated in Fig. 14. Increased package density usually results in closer spacing between

Table 1. Comparison of Measured and Calculated Values of $\Gamma_p$.

| Capacitance per Load (pf) | Spacing (inches) | Measured $\Gamma_p$ for $n$ Loads (percent) | | | | | | Calculated $\Gamma$ (percent) |
|---|---|---|---|---|---|---|---|---|
| | | $n=2$ | $n=3$ | $n=4$ | $n=6$ | $n=11$ | $n=12$ | |
| 5 | 1.5 | — | 16.0 | — | 19.0 | — | 26.0 | 26.8 |
| | 3.0 | — | 15.0 | — | 17.5 | 18.5 | — | 17.3 |
| | 4.5 | 12.0 | — | 12.4 | 13.4 | — | — | 12.9 |
| | 6.0 | 10.5 | — | 10.5 | 10.5 | — | — | 10.1 |
| 10 | 1.5 | — | 29.0 | — | 33.0 | — | 37.5 | 38.0 |
| | 3.0 | — | 22.0 | — | 26.0 | 26.0 | — | 26.8 |
| | 4.5 | 16.6 | — | 18.4 | 20.3 | — | — | 20.8 |
| | 6.0 | 17.5 | — | 17.5 | 17.5 | — | — | 17.3 |
| 15 | 1.5 | — | 38.5 | — | 41.5 | — | 43.5 | 45.0 |
| | 3.0 | — | 27.5 | — | 32.0 | 32.0 | — | 33.2 |
| | 4.5 | 25.8 | — | 27.0 | 29.0 | — | — | 26.8 |
| | 6.0 | 21.5 | — | 22.5 | 22.5 | — | — | 22.6 |

*Note:* Rise time = 1 nanosecond.

the signal transmission line increasing crosstalk between the signal lines. This affects the noise immunity. Thus, the packing denseness required, the electrical and mechanical properties of the signal transmission system, the noise immunity, the current-speed capability and input impedance of the logic circuitry, the logical fan in and fan out required and the spacing of the loads all effect and are affected by the reflections (and other noise producing sources). Therefore, these various system parameters should be evolved concurrently with the various disciplines cooperating.

## CONCLUSIONS

The interaction of two coupled lines, when one is driven by a transient signal, will generally result in induced signals at either end of the undriven line, even when the lines are properly terminated. For the special case where the dielectric medium is homogeneous, the induced signal at the receiving end of undriven line will be zero when the lines are properly terminated.

When the driving pulse having a linear rise time is applied to one of two coupled lines having a nonhomogeneous medium, a crosstalk pulse will be induced at each end of the coupled sense of pickup line. The forward crosstalk at the receiving end of the sense line will be proportional to the slope of the driving pulse and to the length of the coupled region. The back crosstalk pulse at the sending end of the sense line has a waveform that is related to the relationship between the rise time of the driving pulse $T_1$ and twice the propagation time $T_d$ of the coupled region. For $T_1 < 2T_d$ the amplitude of the back crosstalk pulse is independent of rise time and the length of coupled region. The width of the pulse is approximately $2T_d$. For $T_1 > 2T_d$ the maximum amplitude of the back crosstalk pulse is proportional to the time derivative of the driving pulse and to the electrical length of the coupled region.

The equations presented for the back and forward crosstalk apply to lines having either homogeneous or nonhomogeneous mediums. Following the determination of two constants by direct measrement, these equations can be used to predict the forward and backward crosstalk voltage for variable driving pulse rise times, driving pulse amplitude and length of the coupled region.

In any given system the magnitude of the reflection can be a function of the signal transistion time, the length and characteristic impedance of the transmission line, the input conductance and capacitance of the circuit loads, and the spacing and number of the loads.

A transmission line uniformly loaded with logic circuits whose loading characteristics can be represented by capacitors can be represented by a distributed line with a reduced characteristic impedance. This approximation is most accurate as the number of loads increase and/or faster signal rise time. The maximum reflections seen on these types of lines can be determined by the relationship

$$\Gamma = \frac{1 - \left(1 + \frac{C_L}{DCo}\right)^{1/2}}{1 + \left(1 + \frac{C_L}{DCo}\right)^{1/2}}$$

where $C_L$ = effective capacity of load,
$Co$ = capacity per length of transmission line, and
$D$ = spacing of loads on line.

For those lines which are loaded with logic circuits whose loading characteristics are essentially resistive (including those which are nonlinear) the maximum reflections can be calculated by the relationship

$$\Gamma = -\tfrac{1}{2}\,\frac{I_L}{I\,in}$$

where $I_L$ = current level change of logic circuit, and
$I_{in}$ = incident current on line.

The effect of the various system design parameters on each other is pointed out. The need for cooperation and compromise between the requirements of the packaging, signal transmission system, noise immunity, current-speed capability and input impedance, logical gain of the logic circuits, and the spatial distribution of the loads is discussed. The importance of unified design philosophy grows as machines become faster and grow smaller.

## REFERENCE

1. B. M. Oliver, "Directional Electromagnetic Coupling," *Proc. IRE,* vol. 42, no. 11 p.1686 (Nov. 1954).
2. R. C. Knechtli, "Further Analysis of Transmisssion Line Couplers," *Proc. IRE,* vol. 43, no. 7, p. 867 (July 1955).
3. W. L. Firestone, "Analysis of Transmission Line Directional Couplers," *Proc. IRE,* vol. 42, no. 10, p. 1529 (Oct. 1955).
4. F. C. Yao, "Signal Transmission Analysis of Ultra High Speed Transistorized Digital Computers," *IEEE Trans. on EL,* pp. 372-382 (Aug. 1963).
5. H. Gray, *Digital Computer Engineering.*

## Appendix

## MATHEMATICAL ANALYSIS OF TRANSIENT SIGNALS ON COUPLED TRANSMISSION LINES

### *Transmission Line in a Zero Field*

The equations describing $V(x)$ and $I(x)$ for the transmission line shown in Fig. 17 can be found in any standard text on transmission line theory such as reference 6. With $V$ and $I$ positive and $x$ measured from the receiving end as shown, $V(x)$ and $I(x)$ are given by

$$V(x) = Ae^{\gamma x} + Be^{-\gamma x} \qquad (1)$$

$$I(x) = \frac{A}{Z_0}\,e^{\gamma x} - \frac{B}{Z_0}\,e^{-\gamma x} \qquad (2)$$

where $Z_0 = \sqrt{\frac{Z}{Y}} = \sqrt{\frac{R + j\omega L}{G + j\omega C}}$.

$A$ and $B$ are constants determined by boundary conditions and $R$, $L$, $G$ and $C$ are defined on a per unit length basis.

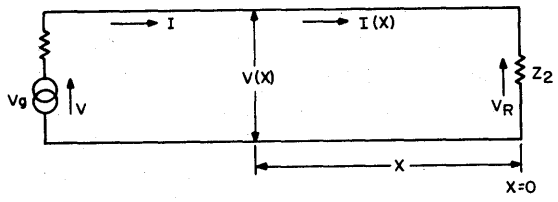



Figure 17. Transmission line.

The instantaneous voltage and current equations can be obtained from Eqs. (1) and (2) by multiplying them by $e_{j\omega t}$, which gives

$$v(x,t) = Ae^{j\omega t + \gamma x} + Be^{j\omega t - \gamma x} \qquad (3)$$

and

$$i(x,t) = \frac{A}{Z_0}\,e^{j\omega t + \gamma x} - \frac{B}{Z_0}\,e^{j\omega t - \gamma x} \qquad (4)$$

Those terms containing expressions of the form $f_1(\omega t - \gamma x)$ indicate a wave traveling in the $+x$ direction while those terms containing the form $f_1(\omega t + \gamma x)$ indicate waves traveling in the $-x$ direction. Note that the $+x$ direction is from the receiving end of the line back toward the generator end.

Evaluating the constants from the boundary conditions at $x=0$, $V(0)=V_R$ and $I_0=V_R/Z_2=I(R)$, Eqs. (1) and (2) become in hyperbolic form

$$V(x) = V_R\,(\cosh \gamma x + \frac{Z_0}{Z_2}\sinh \gamma x) \qquad (5)$$

and

$$I(x) = I_R\,(\cosh \gamma x + \frac{Z_2}{Z_0}\sinr \gamma x) \qquad (6)$$

For convenience, the substitution is made that

$$Q_2 = \frac{V_R}{Z_0}; \; P_2 = I_R$$

This gives

$$V(x) = Z_0 \, (Q_2 \cosh \gamma x + P_2 \sinh \gamma x) \quad (7)$$

and

$$I(x) = P_2 \cosh \gamma x + Q_2 \sinh \gamma x \quad (8)$$

The above equations describe the voltage and current distribution when an exial electric force is applied to a transmission line in a zero field environment. Consider now the effects of placing the transmission line in a nonzero field environment.

## Transmission Line in a Nonzero Field

The voltage and current distribution along a transmission line will be derived by applying the principle of superposition. First, an induced voltage is introduced into the line and the resulting voltage and current distributions are calculated. Then an induced current is introduced into the line and similar calculations are made. The two results are summed, producing the total distribution. The derivation that follows is based on the approach taken in reference 4. It is assumed that the degree of coupling between the two lines is low enough so that there is negligible degradation of the driving waveform as it propagates along the line.

*Voltage and Current Distribution from an Induced Voltage.* Figure 18 shows a transmission line in which there is an induced voltage, $\hat{V}$, at a point $x = \xi$. It is assumed that the internal impedance of the voltage generator $\hat{V}$ is zero. The polarity of $\hat{V}$ is as shown in the figure. Also shown in Fig. 18 are the directions of the conductive and displacement currents.

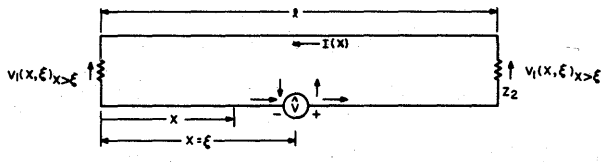



Figure 18. Transmission line with an induced voltage at $x = \xi$.

Let $V(x, \xi)$ and $I(x, \xi)$ be the voltage and current at $x$ when a voltage $V$ is impressed at $x = \xi$. Since there are no impressed voltages in the region $0<x<\xi$ and $\xi<x<\ell$, Eqs. (7) and (8) may be applied to each of these regions. The following equations take note that $x$ is measured with respect to the new reference.

$$V_1(x, \xi) = Z_0 \, [Q_2 \cosh \gamma \, (\ell - x) + P_2 \sinh \gamma \, (\ell - x)]; \quad x>\xi \quad (9)$$

$$I_1(x, \xi) = P_2 \cosh \gamma \, (\ell - x) + Q_2 \sinh \gamma \, (\ell - x); \quad x>\xi \quad (10)$$

$$V_1(x, \xi) = -Z_0 \, [Q_1 \cosh \gamma x + P_1 \sinh \gamma x]; \quad x<\xi \quad (11)$$

$$I_1(x, \xi) = P_1 \cosh \gamma x + Q_1 \sinh \gamma x; \; x<\xi \quad (12)$$

Four boundary conditions are now required to determine the constants $P_1$, $Q_1$, $P_2$ and $Q_2$. Two of these conditions are derived from the terminal conditions of the line, namely, $V(0,\xi)/I(0,\xi) = -Z_1$ and $V(\ell,\xi)/I(\ell,\xi) = Z_2$. The other two conditions result from an evaluation of the voltage discontinuity at $x = \xi$.

From Eqs. (11) and (12)

$$\frac{V(0,\xi)}{I(0,\xi)} = -Z_1 = \frac{-Z_0 \, Q_1}{P_1}$$

since $\sinh(0) = 0$ and $\cosh(0) = 1$,

$$P_1 = PZ_0 \quad (13)$$

$$Q_1 = PZ_1 \quad (14)$$

where $P$ is introduced to simplify manipulation.

From Eqs. (9) and (10)

$$\frac{V(\ell,\xi)}{I(\ell,\xi)} = Z_2 = \frac{Z_0 \, Q_2}{P_2}$$

$$P_2 = Q \, Z_0 \quad (15)$$

$$Q_2 = Q \, Z_2 \quad (16)$$

where $Q$ also is introduced to simplify manipulation.

From the voltage discontinuity at $x = \xi$ the remaining two boundary conditions are obtained.

$$I_1(\xi^{+0},\xi) - I_1(\xi^{-0},\xi) = 0 \quad (17)$$

$$V_1(\xi^{+0},\xi) - V_1(\xi^{-0},\xi) = \hat{V} \quad (18)$$

Substituting Eqs. (13) and (14) into Eqs. (11) and (12) and Eqs. (15) and (16) into Eqs. (9) and (10), respectively, produces

$$I_1(x, \xi) = P \, [Z_0 \cosh \gamma x + Z_1 \sinh \gamma x]; \quad x<\xi \quad (19)$$

$$V_1(x, \xi) = -Z_0 \, P \, [Z_1 \cosh \gamma x + Z_0 \sinh \gamma x; \quad x<\xi \quad (20)$$

$$I_1(x, \xi) = Q \, [Z_0 \cosh \gamma \, (\ell - x) + Z_2 \sinh \gamma \, (\ell - x)]; \quad x>\xi \quad (21)$$

$$V_1(x,\xi) = Z_0 Q [Z_2 \cosh \gamma (\ell - x) + Z_0 \sinh \gamma (\ell - x)]; x > \xi \quad (22)$$

Substituting in Eqs. (17) and (18), gives

$$Q = \frac{P [Z_0 \cosh \gamma \xi + Z_1 \sinh \gamma \xi]}{[Z_0 \cosh \gamma (\ell - \xi) + Z_2 \sinh \gamma (\ell - \xi)]} \quad (23)$$

$$\hat{V} = \frac{Z_0 P [Z_1 \cosh \gamma \xi + Z_0 \sinh \gamma \xi]}{[Z_0 \cosh \gamma (\ell - \xi) + Z_2 \sinh \gamma (\ell \xi)]} \quad (24)$$

With aid of hyperbolic identities, Eq. (24) is reduced to

$$P = \frac{\hat{V} [Z_0 \cosh \gamma (\ell - \xi + Z_2 \sinh \gamma (\ell - \xi)]}{Z_0 [(Z_0^2 + Z_1 Z_2) \sinh \gamma \ell + (Z_0 Z_2 + Z_0 Z_1) \cosh \gamma \ell]} \quad (25)$$

From Eq. (23) $Q$ can be determined

$$Q = \frac{\hat{V} [Z_0 \cosh \gamma \xi + Z_1 \sinh \gamma \xi]}{Z_0 [Z_0^2 + Z_1 Z_2) \sinh \gamma \ell + (Z_0 Z_2 + Z_0 Z_1) \cosh \gamma \ell]} \quad (26)$$

Substituting $P$ and $Q$ in Eqs. (19) through (22)

$$V_1(x,\xi) = \frac{-\hat{V}}{\Delta} \{[Z_0 \cosh \gamma (\ell - \xi) + Z_2 \sinh \gamma (\ell - \xi)] [Z_1 \cosh \gamma x + Z_0 \sinh \gamma x]\}; x < \xi \quad (27)$$

$$I_1(x,\xi) = \frac{\hat{V}}{Z_0 \Delta} \{[Z_0 \cosh \gamma (\ell - \xi) + Z_2 \sinh \gamma (\ell - \xi)] [Z_0 \cosh \gamma x + Z_1 \sinh \gamma x]\}; x < \xi \quad (28)$$

$$V_1(x,\xi) = \frac{\hat{V}}{\Delta} \{[Z_0 \cosh \gamma \xi + Z_1 \sinh \gamma \xi] [Z_2 \cosh \gamma (\ell - x) + Z_0 \sinh \gamma (\ell - x)]\}; x > \xi \quad (29)$$

$$I_1(x,\xi) = \frac{\hat{V}}{Z_0 \Delta} \{[Z_0 \cosh \gamma \xi + Z_1 \sinh \gamma \xi] [Z_0 \cosh \gamma (\ell - x) + Z_2 \sinh \gamma (\ell - x)]\}; x > \xi \quad (30)$$

$$\Delta = [(Z_0^2 + Z_1 Z_2) \sinh \gamma \ell + (Z_0 Z_2 + Z_0 Z_1) \cosh \gamma \ell]$$

Eqs. (27) to (30) give the voltage and current distribution on the line in response to a zero impedance voltage source induced at $Z = \xi$.

*Voltage and Current Distribution from an Induced Current.* Fig. 3 shows a transmission line of characteristic impedance $Z_0$, terminated with $Z_1$ and $Z_2$, in which an infinite impedance current source $\hat{I}$ is induced at $x = \xi$. The direction of the induced current $\hat{I}$ is as shown in the figure. The currents in the line will flow in opposite directions as shown by the two opposing arrows.

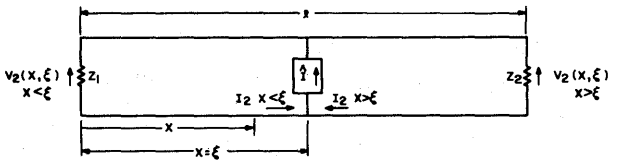



Figure 19. Transmission line with an induced current at $x = \xi$.

A procedure, identical to the one previously used for an induced voltage, yields the voltage and current distribution in response to a current, $\hat{I}$, induced at $x = \xi$. The four boundary conditions required for the solution of the simultaneous equations are

$$\frac{V_2(0,\xi)}{I_2(0,\xi)} = -Z_1; x = 0 \quad (31)$$

$$\frac{V_2(\ell,\xi)}{I_2(\ell,\xi)} = -Z_2; x = \ell \quad (32)$$

$$V_2(\xi^{+0},\xi) - V_2(\xi^{-0},\xi) = 0; x = \xi \quad (33)$$

$$I_2(\xi^{-0},\xi) - I_2(\xi^{+0},\xi) = \hat{I}; x = \xi \quad (34)$$

As before, the current in the direction of $x$ is a positive current. Therefore $I_2(\xi^{-0},\xi)$ is a positive current while $I_2(\xi^{+0},\xi)$ is a negative one. The boundary conditions at $x = 0$ and $x = \ell$ both generate negative impedance values because the currents and voltages are in opposite directions. This is done to allow compatibility with the previously discussed problem with an induced voltage.

Thus, starting with Eqs. (9) through (12) and using Eqs. (31) through (34) in the same manner as before, we arrive at the votage and current distribution at $x$ due to an induced current $\hat{I}$ at $x = \xi$.

The solutions are

$$V_2(x,\xi) = \frac{-\hat{I} Z_0}{\Delta} \{[Z_0 \sinh \gamma x + Z_1 \cosh \gamma x] [Z_0 \sinh \gamma (\ell - \xi) + Z_2 \cosh \gamma (\ell - \xi)]\}; x < \xi \quad (35)$$

$$I_2(x,\xi) = \frac{\hat{I}}{\Delta} \{[Z_0 \cosh \gamma x + Z_1 \sinh \gamma x] [Z_0 \sinh \gamma (\ell - \xi) + Z_2 \cosh \gamma (\ell - \xi)]\}; x < \xi \quad (36)$$

$$V_2(x,\xi) = \frac{-\hat{I} Z_0}{\Delta} \{[Z_0 \sinh \gamma \xi + Z_1 \cosh \gamma \xi] [Z_0 \sinh \gamma (\ell - x) + Z_2 \cosh \gamma (\ell - x)]; x > \xi \quad (37)$$

$$I_2(x,\xi) = \frac{-\hat{I}}{\Delta}\{[Z_0 \sinh \gamma\xi + Z_1 \cosh \gamma\xi] [Z_0 \cosh \gamma (\ell - x) + Z_2 \sinh \gamma (\ell - x)]\}; x > \xi \quad (38)$$

The voltage and current distribution at any point $x$ on the line due to an induced voltage and current at $x = \xi$ can now be obtained via superposition. Also, because the main interest here is in the induced signals, let the line be matched, that is let $Z_1 = Z_2 Z_0$.

Substituting $Z_0 = Z_1 = Z_2$ and combining Eqs. (25) to (29) and (35) to (38) while using the relationships $\cosh x = (e^x + e^{-x})/2$ and $\cosh x = (e^x - e^{-x})/2$ and $\cosh x + \sinh x = e^x$, the equations reduce to

$$V(x,\xi) = \frac{-1}{2}(\hat{V} + \hat{I} Z_0) e^{-\gamma(\xi - x)}; x < \xi \quad (39)$$

$$V(x,\xi) = \frac{1}{2}(\hat{V} - \hat{I} Z_0) e^{-\gamma(x-\xi)}; x > \xi \quad (40)$$

$$I(x,\xi) = \frac{1}{2}\left(\frac{\hat{V}}{Z_0} + \hat{I}\right) e^{-\gamma(\xi - x)}; x < \xi \quad (41)$$

$$I(x,\xi) = \frac{1}{2}\left(\frac{\hat{V}}{Z_0} - \hat{I}\right) e^{-\gamma(x-\xi)}; x > \xi \quad (42)$$

Equations (39) to (42) define the voltage and current distribution at any point $x$ on a transmission line as a result of an induced differential voltage and current at $x = \xi$. The sign factors in these equations should be interpreted in terms of the assumed positive direction in Figs. 18 and 19. For example, the minus sign in Eq. (39) indicates that the direction for the positive voltage at $x = 0$ is opposite to that assumed. Again, this was expected from the qualitative analysis. The expressions that hold for $x < \xi$ quantitatively describe the back crosstalk which was qualitatively discussed earlier. In Eqs. (39) and (40) the two terms reinforce each other while for $x > \xi$, the forward crosstalk, the terms are of opposite polarity. These were the results expected in light of the qualitative analysis. If the relationship $\hat{V}/\hat{I} = Z_0$ is true, the forward crosstalk voltage reduces to zero. In this case, for crosstalk considerations, the sense line may be terminated in any impedance at the receiving end.

The total induced voltage and current resulting from a driving pulse propagating along the driver line are obtained by adding the differential voltages and currents expressed in Eqs. (39) to (42) over the region of interaction. Here we follow the method outlined in reference 5. If an applied current signal $I_0$ on the drive line is propagating from left to right, the current and voltage at any point $\xi$ on the drive line may be written as

$$I_a = I_0 e^{-\gamma\xi} \quad (43)$$

$$V_a = Z_0 I_a \quad (44)$$

Defining $C_m$ and $L_m$ as the mutual capacitive and inductive coupling between the driving lines and the sense line, $\hat{I}$ and $\hat{V}$ can be expresszed as

$$\hat{I}(\xi) = s C_m V_a \quad (45)$$

$$\hat{V}(\xi) = s L_m I_a \quad (46)$$

where $s$ is the argument of the Laplace transform.

Substituting Eqs. (43) to (46) in Eqs. (39) and (40) and integrating over the region of interaction we get

$$V(x,s) = -\frac{I_0}{2}\int_0^x [s L_m e^{-\gamma\xi} - s C_m Z_0^2 e^{-\gamma\xi}] e^{-\gamma(x-\xi)} d\xi + \frac{I_0}{2}\int_x^{\ell} [s L_m e^{-\gamma\xi} + s C_m Z_0^2 e^{-\gamma\xi}] e^{-\gamma(\xi - x)} d\xi \quad (47)$$

$$= -I_0\left[\frac{(L_m - C_m Z_0^2}{2} s x e^{-\gamma x} + \frac{(L_m + C_m Z_0^2)}{4\gamma} s(e^{-\gamma(2l-x)} - e^{-\gamma x})\right] \quad (48)$$

For a lossless line the propagation constant $\gamma = \sqrt{LC} = s Td/\ell$ where $T_d$ is the time required to propagate the length of the coupled line $\ell$. Substituting in Eq. (48) and taking the inverse transform, we get

$$V(x,t) = K_f x \frac{d}{dt}\left[V_0\left(t - \frac{T_d}{\ell} x\right)\right] - K_b\left[V_0\left(t - \frac{T_d x}{\ell}\right) - V_0 (t - 2T_d + \frac{T_d x}{\ell}\right]^* \quad (49)$$

where

$V_0 = I_0 Z_0$

$K_f$ = forward crosstalk constant
$= -(1/2)(L_m/Z_0 - C_m Z_0)$

$K_b$ = back crosstalk constant
$= (\ell/4T_d)(L_m/Z_0 + C_m Z_0)$

*All time functions delayed by a constant $T_k$ are modified by the unit step function $u(t - T_k)$ such that

$$u(t - T_k) = \begin{cases} 0 \text{ for } 0 < t < T_k \\ 1 \text{ for } T_k < t \end{cases}$$

# INTEGRATING COMPUTERS INTO BEHAVIORAL SCIENCE RESEARCH

H. Borko
*System Development Corporation*
*Santa Monica, California*

## IMPLICATIONS OF THE COMPUTER FOR THE BEHAVIORAL SCIENTIST

All of us would agree, I believe, that the mathematician should be interested in computers, for computation and mathematics go together. Similarly, it is obvious that the engineer must use computers in his work to perform the calculations involved in designing bridges, in predicting the location of orbiting satellites, or in other engineering tasks. But why should the behavioral scientist, who by definition, is interested in human and animal behavior, be concerned with computers? The answer to this question is that the computer is a very versatile and powerful tool that, when properly used, enhances one's ability to do creative work. The computer provides the behavioral scientist with the capability of studying problems considered impossible as recently as two decades ago.

In the course of this paper I will describe some of this new research and illustrate a few novel uses of computers as well as some of the social implications of this usage. Let me begin by presenting an overview of the effect the computer is having on research design and on the role of the behavioral scientist. As a tool, the computer can contribute to the research task by aiding in the statistical analysis of experimental data, and by modeling, or simulating, experimental variables to find new relationships and test hypotheses.

Research involves the accumulation of data and their eventual analysis and reduction. As our research capabilities expand, more and more data are accumulated and analysis becomes increasingly complex and time-consuming. The statistical techniques are themselves becoming more sophisticated. If behavioral science remains chained to the desk calculator, it will stagnate.

High-speed electronic data processing provides a means for rigorously analyzing masses of data. In addition, special computer programs can search for patterns and relationships that, because of the number and distribution of variables, might remain obscured and undiscovered. From these patterns, as exemplified in the study of electroencephalographs and electrocardiographs, the researcher is seeking to formulate new and significant hypotheses.

Testing of hypotheses can also be facilitated by using the computer to simulate and study human behavior. But why go to all that trouble? Why not study human behavior directly rather than simulating the behavior on a computer and then studying the model? The physiological psychologist will immediately recognize the similarity of this question to the older, more common query: "Why do you study rats if you are primarily interested in human behavior?" And the answer is very much the same.

By simulating behavior on a computer and studying its operation in detail, we can, by analogy, make inferences about the more complex human activities and then test and validate these inferences.

The analogy of the organization of the computer itself to that of the human nervous system is a fascinating model for the behavioral scientist to investigate. It has been suggested that experience leaves a permanent mark on the brain—possibly by changing the configuration of protein molecules in the brain cells. The resulting memory trace permits recall of the experience. Computer memory is somewhat similar. Information is recorded by magnetizing a certain small area of core memory. The computer is then able to recall this information by determining whether the part is, or is not, magnetized. Whether this analogy will lead to a formal theory of memory and thought remains to be seen. But one thing is certain: without the computer it would be impossible to perform the many calculations necessary to develop the theory and to verify it under various operating conditions.

The computer, in addition to providing the behavioral scientist with a new tool for statistical analysis and modeling procedures, is influencing research by changing the very environment in which humans live and behave. One of the areas that the social scientist must study is the effect of computerization in the factory and office. Are workers being replaced by the machine faster than new jobs are being created? What new skills are required by automation, and can a firm's present employees be retrained to qualify for the technically more demanding jobs in the automated plant? Are new training techniques needed and can they be developed? In seeking answers to these questions, management is turning to the behavioral scientist for help.

Thus, we see that the computer is affecting the research efforts of the behavioral scientist in three ways: (1) by providing a more powerful tool for the statistical analysis of his data, (2) by providing a means of simulating a system of complex interacting variables, and (3) by requiring new data on the effects of automation in the changing economic environment.

Let us now examine each of these areas in detail.

## THE COMPUTER AS A DATA PROCESSOR

The study of human behavior is an extremely complex task. One complication, for example, is that there are no "standard subjects" in the sense that there are pure reagents in chemistry. Instead of dealing with typical subjects, the experimenter is forced to work with a sample of a larger population. This raises the question of what is an adequate sample. Providing an answer involves estimating population parameters and calculating tests of significance. The concern over individual differences has led the experimental psychologist to develop sophisticated statistical techniques and complex experimental designs.

Although the uniqueness of behavioral variables was recognized from the very beginning, early experimental work modeled itself after the physical sciences. The behaviorist school of psychology attempted to account for complicated patterns of behavior in terms of elementary unitary sensations. The analogy to the chemist building structures from basic elements is obvious. Early experimentation consisted of accumulating observations of a single variable under varying conditions.

Soon this mode of observation gave way to the "before and after" study, but still the emphasis was on the single variable. These experiments began with the specification of a set of initial conditions in which the dependent and independent variables were neatly separated and identified. The independent variable was then changed in a predetermined manner while all other variables were held constant, and the results, before and after, were measured and compared. This is the classic univariate design. It was used by Pavlov to condition his dogs to salivate at the sound of a bell, and it is used today by teachers who give or withhold rewards to motivate learning under the assumption that, all other things being equal, a child will work harder to earn an *A* in school than he will for a *B*.

In the attempt to understand complex behavior under conditions when all other things are not equal, the univariate design gave way to multivariate statistics, analysis of variance, multiple regression, and factor analysis.

These methods were known and in use prior to the advent of computers. However, their use requires a great many calculations. When computers became available to handle the repetitious and time-consuming tasks of calculation, more experimenters began using multivariate statistics. Even so, in most experiments, and recent ones at that, all regressions are assumed to be linear, and all variables are assumed to be normally distributed. These

assumptions are made in spite of the fact that the real world does not behave this way. Linearity is assumed because it is required by most available statistical models; thus, more realistic/realistic models are needed and are being developed. These are complex models and require the computational capabilities of computers.

Another effect of having a computer available is that the experimenter is able to interact more intimately with his data and to steer the analysis. He can ask the "What if . . .?" question—what would happen if I added another variable, or if I arranged the data differently, or if I used a nonparametric model, etc. With the computer standing by to do the dirty work, the experimenter is willing to ask more questions and do more analyses.

The computer can also be used as an aid in theoretical research on statistical techniques, and thus can help solve some of the problems that computer technology brought into prominence. Some of this research in methodology involves the use of random number generators for studying specialized non-normally distributed samples, new techniques of matrix multiplication, and many other mathematical and statistical problems.

One thing is clear. The computer has had a significant effect on experimental design and data reduction techniques. It is, therefore, especially important for today's scientists to learn to use the computer so that they can bring more powerful research methods to bear in solving the significant problems of our times.

## THE COMPUTER AS A SIMULATION DEVICE

As important as the computer is in data processing, it makes an even greater contribution as a tool for simulation and modeling.

Newell and Simon[1] have pioneered in the use of the computer as a device for simulating the processes of human thinking. In one of their experiments, they presented the subject with a problem requiring the transformation of one set of symbols into another set by following certain transformation rules. They asked the subject to talk out loud as he applied these rules. This verbalization provided the experimenters with a record of the subject's thought processes as he worked toward the solution of the problem.

The research task that Newell and Simon set for themselves was to construct a theory that would identify and integrate the various processes that influence the subject's behavior as he attempts to solve problems. Having constructed the theory, it becomes necessary to test its adequacy by comparing the behavior predicted by the theory with the actual behavior of the subject. This much, I believe, is clear. The point that is unclear, perhaps, is, where does the computer fit in? How can the computer help either in constructing or in testing a theory of human problem-solving behavior?

It will be recalled that the task given the subject involved symbol manipulation—transforming one symbolic expression into another. Obviously humans can solve symbolic problems, but computers are also symbol manipulators.

This similarity suggests that the behavior of both the human and the machine can, at some level of abstraction, be described in the same terms. It is postulated that the human subject's behavior in a problem-solving situation is governed by a program organized from a set of elementary information processes. It is further postulated that a program can be written that will cause the digital computer to execute a procedure corresponding to the human's information-processing or symbol-manipulation behavior. In other words, the computer program will cause the computer to behave in the same way that the subject behaves. Both man and machine will emit substantially the same set of symbols when both are controlled by the same program. Under these circumstances, as Newell and Simon point out, the computer program can then be regarded as a theory of behavior.

This is a most ingenious formulation, requiring no assumptions beyond the easily acceptable notion that men and computers are both general-purpose symbol-manipulating devices, and that the computer can be programmed to manipulate symbols so that the output behavior resembles human information processing. Thus, the use of computer simulation for studying human thinking becomes clear. What Newell and Simon have suggested, and have, in fact, done with their GPS or General Problem-Solver program, was to describe human problem-solving behavior as a set of subroutines that could be written as a computer program. The theory and the program are one and the same. Computer simulation forces the theorist to be complete and precise in his formulations. In addition, the computer program provides a test of the theory by comparing the simulated behavior with the original.

Computer simulation can involve a single individual or a large complex social system. Ithiel de Sola Pool[2] has been interested in studying the so-called "human intangibles" in social systems engineering. He asks, "Can we begin to build into systems planning such factors as the moods of voters, the fads of consumers, the desires of city dwellers or—in the case of attempts to export technology to developing nations—the superstitions of peasants?" And he answers, "To a considerable degree the social sciences can do all of these things." The technique that makes this possible is computer simulation.

Pool is particularly noted for his work in simulating voting behavior and predicting the 1960 presidential election. The source data for this simulation was derived from public opinion polls, which had been conducted between 1952 and 1958. The data were used to construct 480 voter types; one type, for example, was described as "lower income, Protestant, midwestern, urban, white, Republican, female." Along with each type, there was stored in computer memory the number of people of that type in each state and how voters of that type had replied to public opinion poll questions on some 50 topics.

The simulation program utilized this initial information and operated on it by a number of routines expressing social science propositions, especially the cross-pressure theory. In essence, this is a theory that explains and predicts the behavior of a voter when some pressures are pushing him toward one candidate and some toward another. Under these circumstances it is theorized the voter tends to become less interested in politics, makes up his mind later in the campaign, is inclined not to vote at all, but, in the end, tends to vote in accordance with his initial partisanship. For each issue, or cross-pressure, the simulation program looks up the record of the 480 voter types, how each had divided on that issue in the past, how he had voted in the past, how inclined he was to nonvoting, and how steady his voting record was. Each of these values is entered into an equation that, when solved, predicts how that voter type would divide in the 1960 election.

This procedure resulted in a particularly successful simulation of a very complex set of variables and a validation of the cross-pressure theory. As Pool reports, "In August 1960, our operation gave a state-by-state relationship, when we simulated a campaign stressing this issue (i.e., the religious issue), which correlated 0.82 (out of a possible 1.0) with the actual results in November—although no data entered in our simulation were collected any later than 1958!" The record speaks for itself, and provides evidence that simulation is a most important tool for social science research.

In addition to studies of problem-solving behavior and of voting behavior, simulation has been used as a method of studying the cognitive processes, military systems, business organizations, international relations and diplomacy and many other problem areas.[3,4]

## COMPUTER APPLICATIONS IN LANGUAGE AND COMMUNICATION

In the preceding applications, the computer has been used, for the most part, to process numerically coded data, and although the research has been ingenious, the computer has been used in its traditional role. However, the modern digital computer is, as was pointed out, more of a symbol manipulator than a lightning calculator. The computer can be used to process natural language and to aid in human communication. Work in this area has included research on (1) machine translation, (2) the development of question-answering machines, and (3) the retrieval of information in the form of documents or text.

Since machine translation and question-answering machines are being discussed in other sessions of this conference, I will concentrate my few remarks on the application of computers to the library problems of indexing, classifying, storing and retrieving documents. There are a relatively large number of researchers working on these problems as well as an extensive literature. These researchers envision a library of the future in which documents either arrive in machine-readable form or are converted to code by optical scanning equipment. The resulting magnetic tapes are then processed to prepare lists of key terms. These lists, and the documents from which they are generated, are reviewed and used for selecting index terms. At present the final selection can be made more efficiently by skilled humans than by computer programs. The index terms constitute the basis for automatically classifying the documents into subject categories. Although automated classification is consistent, librarians may not always agree that the computer

puts the document into the most reasonable category. This is a problem, and for now one can only respond that since the retrieval will be done by machine, it is more important that the classification seem logical to the computer than to the librarian. This is another way of saying that classification is not an end in itself but serves only to improve the efficiency of retrieval procedures.

Classification has two meanings. The first one, which we have discussed, consists of classifying documents by placing them into existing categories of information. A second meaning of classification is the organization of a collection of items into a reasonable number of subject categories—to derive the categories into which the collection could be logically arranged.

A number of mathematical techniques have been suggested for deriving classification categories; these include clump theory, factor analysis, and multiple regression analysis. In the library of the future, both aspects of classification will be automated; classification categories will be derived by computer programs and documents will be classified into their proper categories by other programs. Automatic classification has the additional advantage that, when new categories are added or existing ones deleted, the entire document collection could be reclassified without additional human effort since the document is left untouched; only the file reference to the document, the computer equivalent of the card catalog, is changed.

A request for material is processed in the same manner as if the request itself were a document. The request is converted to machine code and automatically indexed and classified. It is this request classification category and the index terms that will be used in the document search. The librarian reviews the machine analysis of the request and adds, deletes, or otherwise modifies the terms to improve the system response. The search for appropriate documents is made by the computer, which selects all documents containing the required combination of index terms. If remote inquiry stations and a suitable computer program are available, a user can input a request for documents on a teletypewriter in Los Angeles and receive a reply from information centers in Chicago or Washington. Information retrieval systems of the type described are now being designed,[5,6] and it is reasonable to expect that some operational systems will be available within the next decade.

## IMPLICATIONS OF THE COMPUTER REVOLUTION ON THE HUMAN VALUES

It is obvious from the foregoing discussion that computer technology and automation have many applications beyond numerical analysis and engineering. The computer is beginning to play an important role in our economy and in our lives. The consequences are not always desirable. A significant percentage of Americans have been displaced from their jobs because of automation and computers are now commonplace in business offices and in factories. The changes that are taking place as a result of computer applications can only be described as revolutionary. The effect is comparable to the revolution that took place in our economy and in our lives, when the automobile came into general usage, except that these earlier innovations created new jobs while the cybernetic revolution is capable of producing more goods with fewer people.

Technological unemployment is real and the resulting problems are also real. We are very familiar with one of these problems, namely poverty. I do not wish to dwell on this, except to echo the belief expressed by President Johnson that poverty can be eliminated. We already pay farmers not to plant crops, and I see no reason why we cannot pay a harness maker not to make harnesses or a coal miner not to mine coal. Poverty can be eliminated, but even if everyone had a guaranteed annual wage, there would still be problems resulting from automation. These are psychological problems that have their roots in the conflicting conditions inherent in the changing contemporary American scene. These are conflicts between the desire to work and the lack of jobs and between our reverence for man and the apparently superior capabilities of the machine.

Our society praises work and considers idleness evil. "The devil finds work for idle hands," is an old folk saying. This aspect of our value system was commented on by the economist, Max Weber, who called it the "Protestant ethic." The point is that the Protestant ethic and the values this phrase connotes are in conflict with the realities of a world in which there is not enough work to go around. If it is good to work, and we are not working, then we are not good. This thought is expressed beautifully by Robert Davis in his article "The Computer Revolution and the Spirit of Man."[7] He states, "To question my work is to threaten my value as an individual, to deprive me of work is to take from me

the opportunity to give meaning to my life—the opportunity to achieve and to be recognized by my fellow human beings."[8] In a highly automated society, many jobs that are now being done by man will be done by machine. The machines will, in fact, deprive man of an opportunity to work and to achieve, even though he may still receive an income.

When computers replace man on a job, he cannot help but get a feeling of inferiority. We are told that machines are faster, that they are more accurate, and that they are more dependable than men and, accordingly, we rent large computers at 5 or 6 hundred dollars an hour and pay skilled human labor one one-hundreth of that price.

To quote Dr. Davis again, "It makes very little difference, for the purpose of this discussion, whether the machine is or is not superior to man. In many ways the issue is academic. What really matters is the extent to which the average person is convinced that the machine will replace him because it is better than he is. With the enormous prestige of science standing behind us, I submit that we have unwittingly convinced the average man (who may not in any case hold himself in very high esteem) that he is, in fact, not really worth very much. This, I fear, is a most serious mistake—first, because of the impact it will ultimately have on our culture, and second, because it is not true."[9]

These are some of the problems arising from automation that the behavioral scientist must study and resolve. Poverty is being attacked and will be eliminated. But how does one resolve the conflict between the Protestant ethic, which demands that an individual contribute to society by working and the economic fact that jobs are not available? And how do we make man feel worthy and superior when he is put out of work because the computer can do his job better than he can? Unless these conflicts can be resolved, the men and women who are caught between these inconsistencies will fight their frustrations by adopting some form of antisocial—either aggressive or regressive—behavior to the detriment of our society.

Clearly, solutions are needed, and the social and behavioral scientist must be encouraged to study these aspects of automation. How can we use our resources, both material and human, to build a better life? It is not enough to free man from drudgery; one must free him for something. But how, and for what? How can old values be changed, and even more important, what shall the new values be? We began this talk by examining the effect that computers are having on the research conducted by the behavioral scientist and how the computer was used to simulate relationships and analyze experimental data. We explored the applications of the computer to such diverse disciplines as research, education, and communication. Then, as we examined the effects of automation, we saw how the computer was creating a need for more research and for a better understanding of our social system and its value structure. The computer has provided us not only with more powerful investigating tools but also with more significant problems. It is the task of the behavioral scientist to look beyond the statistics and abstractions of his data and to achieve a sympathetic understanding for the individual, to study the problems of today, and to mold the conditions that will lead to the Great Society of tomorrow.

## REFERENCES

1. A. Newell and H. A. Simon, "Computer Simulation of Human Thinking," RAND Corporation, April 20, 1961, p. 2276.

2. Ithiel de Sola Pool, "Simulating Social Systems," *International Science and Technology,* Mar. 1964, pp. 62-70.

3. H. Borko, *Computer Applications In the Behavioral Sciences,* Prentice-Hall, Englewood Cliffs, N.J., 1962.

4. H. Guetzkow, ed., *Simulation in Social Science: Reading,* Prentice-Hall, Englewood Cliff, N. J., 1962.

5. H. Borko and H. P. Burnaugh, "BOLD: Bibliographic Organization for Library Display," SDC TM-530/008/00, Research and Technology Division Report for 1964, pp. 62-3 (Jan. 1965).

6. G. Salton, "A Document Retrieval System for Man-Machine Interaction," *1964 Proceedings of the Association for Computing Machinery, New* York, pp. I2.3-1-20.

7. R. H. Davis, "The Computer Revolution and the Spirit of Man," *AFIPS Conference Proceedings,* vol. 25, 1964 Spring Joint Computer Conference, Spartan Books, Washington, D.C., pp. 161-167.

8. Ibid., p. 162

9. Ibid., p. 163

# DATA ANALYSIS IN THE SOCIAL SCIENCES: WHAT ABOUT THE DETAILS?

Geoffrey H. Ball
*Stanford Research Institute*
*Menlo Park, California*

## INTRODUCTION

Historically, statistics seems to have been the primary mode of data analysis in the social sciences. It would appear at this time that we are still, to a large extent, using statistical methods developed prior to the advent of the digital computer and that these are now just transposed bodily onto a digital computer to perform the calculations. In this paper we attempt to demonstrate that there exists a class of techniques more suitably oriented toward the capabilities of the digital computer than are conventional analytic statistical techniques. We maintain that these techniques are capable of considering details in social sciences data, that is, relating the individuals described in the data.

The use of some statistical techniques depends on statistical quantities estimated from the data. We show that this extraction from the data can lead to erroneous conclusions because it does not take into account variations in the data that cannot be detected from the quantities estimated. We therefore first show some limitation of some present statistical techniques. We then outline a body of computer-oriented techniques called "cluster-seeking" techniques and summarize the work that has been done so far on these techniques.

We present an improved version of the ISODATA cluster-seeking technique that incorporates aspects of other techniques in ways that appear to overcome certain difficulties that arise in each of the techniques that have been suggested thus far. Finally, we speculate on the ramifications of a widespread use of these cluster-seeking techniques.

## STATISTICAL ANALYSIS OF SOCIAL SCIENCE DATA

In statistical data analysis, much use is made of the covariance and the correlation matrix. For example, the correlation matrix is used in a central way in principal components analysis, in factor analysis, and in canonical correlation analysis.

In Fig. 1 we show three sets of data that, when plotted, appear to be very different. (The second data set consists of samples drawn from a normal distribution. No truncation of the data set should be inferred from the figure.) The interesting fact is that all three of these data sets give rise to the same covariance matrix and hence to the same correlation matrix. If the data points were specified by a different coordinate system, the covariance matrix would be modified in the same way for all the data sets. Hence we see that these data sets are indistin-

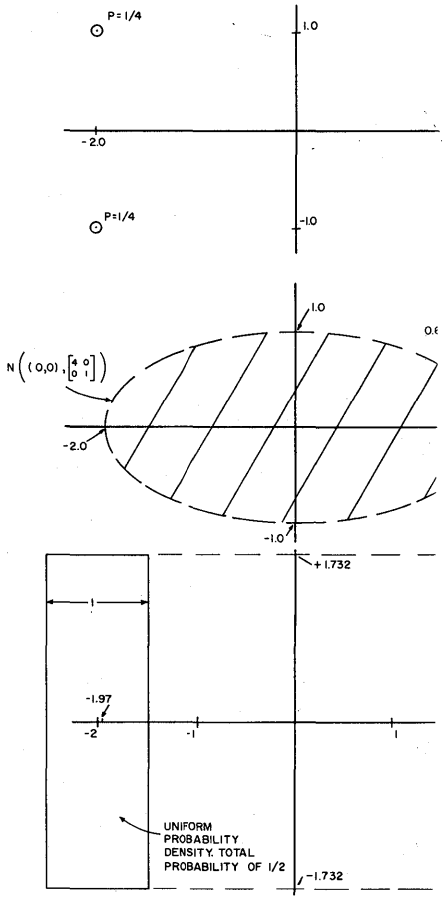



Figure 1 Three distributions of data l covariance matrix

Figure 1. Three distributions of data having the same covariance matrix.

guishable if only the first and second moments and cross-moments are used to describe the data. The fact that these very different data sets lead to the same covariance matrix is rather unnerving when, for example, with the principal components analysis, it is realized that frequently no use is made of the original data except to abstract the means and the covariance matrix. It therefore seems reasonable to ask how the detailed structure of the data might be taken more accurately into account.

Before examining this question in some more detail, let us examine two other aspects of utilizing only the covariance matrix and the mean. Four factors seem particularly significant here:

1. The effect of erroneous data points, caused, for example, by card punching rors, can be rather considerable in a particular covariance matrix. Unless these erroneous points can be determined and removed from the data set, the statistics based on the entire data set will be considerably affected.
2. A second effect arises from the need to estimate the covariance matrix using a set of *sample* points. Dr. David Allais[3] has shown recently that the number of samples should roughly equal 10 times the number of dimensions in order to estimate adequately the covariance matrix.
3. More than one modal point in the data also causes the covariance matrix to be an inadequate description of the data. Thus, if the data is described as being the sum of two Gaussian distributions and a single overall covariance matrix is computed for the entire data set *without* taking the bimodality of the probability distribution into account, then this covariance matrix depends critically on the distance between the means of these two modes. Intuitively this is rather unsatisfying as a description of the data.
4. Predominant subsets in the data can overwhelm subsets that occur less frequently, i.e., the significant but rare event, may not be singled out.

This is not to say that a principal components analysis or factor analysis does not have its place. The point seems rather to be that if these techniques are to be used, then the data ought to have characteristics that in some way satisfy assumptions of Gaussian distributions; that is, the covariance matrix should be a good description of the characteristics of the data.

A useful description of all of the data can be plotted using the axes found by factor analysis or principal components analysis. Even this may not be as meaningful as desired as can be seen by examining Figs. 2a and 2b where we see that two quite different probability distributions give rise to marginal distribution that are uniform in the two primary directions. We can see that this lack of uniqueness can be resolved by using more axes to describe the data. The problem here becomes that of relating events on the different axes, particularly when the data are high-dimensional. The primary concern, then, is that data that is in fact rather dissimilar ap-

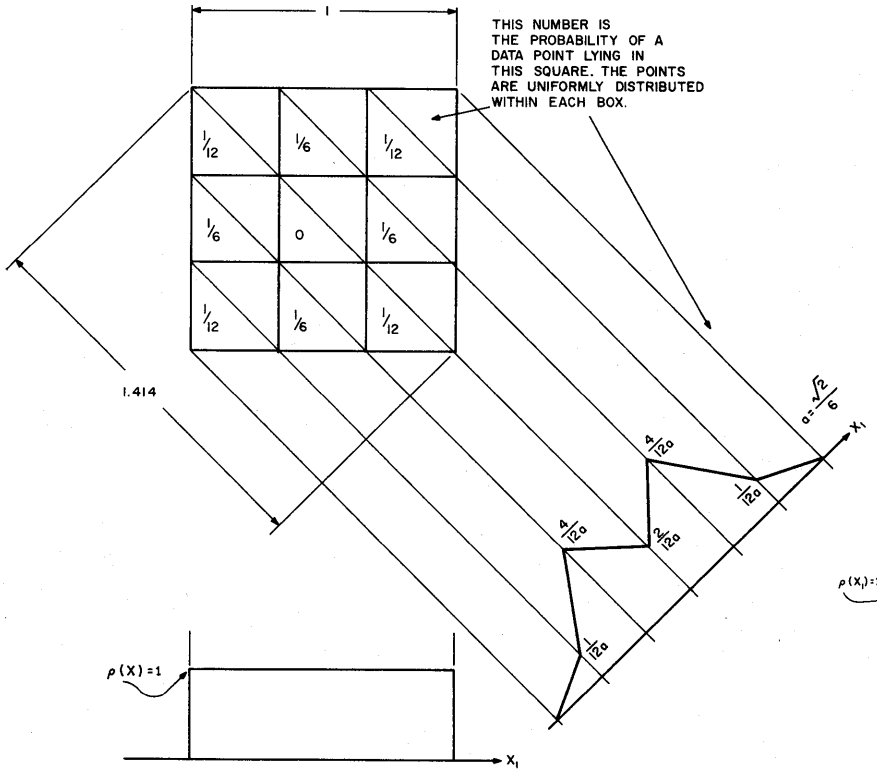



Figure 2a. A nonuniform two-dimensional distribution of points having uniform marginal distributions.

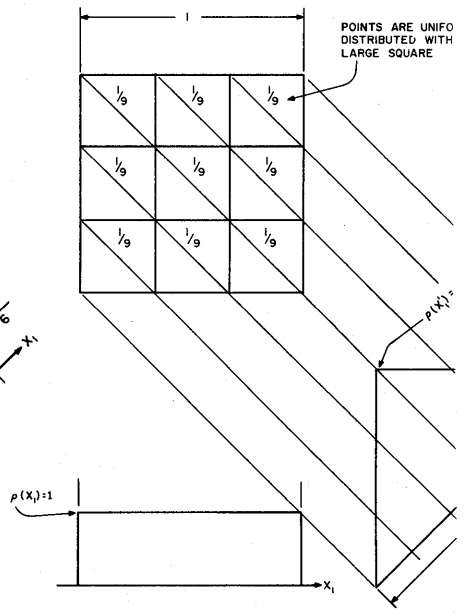



Figure 2b. A uniform two-dimensional distribution of points having uniform marginal distributions.

pear to be quite similar when viewed by data analysis techniques implicitly oriented toward Gaussian distributions.

It therefore appears to us that there is a need to treat local regions in the data space rather than projecting down on to a line or a plane from over the whole space. That is, there is a need to be concerned about the details. We feel that a set of techniques that have been developed primarily over the last five years provides a satisfactory direction for finding an answer to this need to examine the details. Much work remains to be done on these techniques, but it does appear that the particular point of view that they offer can be very helpful.

## CLUSTER-SEEKING TECHNIQUES

The essential characteristics of the techniques that we will be describing is the sorting of the set of data patterns into subsets, such that each subset contains data points that are as much "alike" as pissible. The methods for arriving at the subsets differ in a variety of ways. We will, in the following paragraphs, attempt to describe the known existing techniques and to point out ways in which they are significantly different. We finally attempt to construct a technique that is a composite of the best features of some of these techniques and ISODATA.

### *Historical Background*

In Table 1 we have arranged the papers by type (e.g., probabilistic, signal detection, etc.) and by date. We have made no attempt to determine how much the development of one technique depended on another. They are arranged in chronological order in a way that appears reasonable to us at this time, without attempting to accurately establish priorities. As far as we have been able to determine, nearly all the techniques with the exception of the factor analytic techniques and some lumping techniques originated after 1960. Since most of these techniques require a considerable amount of computation, it seems probable that only the advent of inexpensive digital computation has allowed these techniques to be developed.

Table 1. Cluster-Seeking Techniques.

| Class of Technique | Year First Reported | | | | | | |
|---|---|---|---|---|---|---|---|
| | Before 1960 | 1960 | 1961 | 1962 | 1963 | 1964 | 1965 |
| Probabilistic | | | | Daly | | Fralick | |
| Signal Detection | | Jakowatz et al | Brennan Glaser | Hinnich | Spilker | Turner Smith | |
| Clustering | | | | Okajima et al Sebestyen Hyvarinen | | Ball & Hall | |
| Clumping | Michener & Sokol (1957) | Rogers & Tanimoto Sawrey, Keller, & Conger | Needham Parker-Rhodes Abraham | | | Bonner Fortier & Solomon | |
| Eigenvalue I | | | | Nunnally | | | |
| Eigenvalue II | | | | | Cooper | Mattson & Damman | |
| Minimal Mode-Seeking | | | | | Firschein & Fischler Steinbuch & Piske | | |
| Miscellaneous | | | | Block, Knight & Rosenblatt | Bledsoe | | |

*Definitions*

In order that certain terms to be used over and over again in this paper should not be confused, it seems important to define them:

1. *Measurement* — By measurement we mean a component of the pattern vector; that is, it is one of several numerical values (related to a property of the pattern) used to define a pattern—for example, one out of several answers on a questionnaire.
2. *Pattern* — By pattern we mean the collection of measurements considered to be a single entity in the clustering program for example, the answers to a set of questions on a questionnaire.
3. *Parameter* — By parameter we will *not* mean measurement in this paper. Rather, we will mean a number used to control the operation of one of the cluster-seeking techniques—for example, the threshold used to control lumping in the Ball-Hall technique[4] is a parameter.
4. *Cluster* — A cluster of patterns is, in our mind, a set of patterns contained in a high-dimensional space where the density of patterns is large compared to the density in the surrounding volumes. It is not yet a rigorously defined concept but rather one that depends on the nature of the data. Attempts are being made to define this more exactly.
5. *Mode* — A mode is a cluster of patterns that belong to a *single* class of patterns. (This definition varies from the more precise statistical definition of a mode as the most frequently observed value of a random variable.)

## COMPARE AND RELATE CLUSTER-SEEKING METHODS

In the following paragraphs we describe and dis-

cuss each of the aspects of cluster-seeking techniques that we consider particularly significant.

*The Use of Iterative Techniques*

It generally does appear that iterative techniques allow a more detailed examination of the data than do techniques that require the calculation of a single function of all the data (e.g., the covariance matrix). Iterative techniques can allow the examination of iteratively selected subsets of patterns where the selection of one subset depends on the results obtained from a previous selection. This examination increases the sensitivity of the methods to variations in patterns and the structure of the data without the assumptions that are usually necessary in non-iterative methods.

*The Use of Categorization Information to Cause Mode-Seeking*

It is possible to use classification information to determine the "modes" in the data. When this is done, it then becomes important that all the classes that are to be classified be represented in the data. In Fig. 3 (upper half) we see that it is possible that two modes of Class 1 be sufficiently remote from Class 2 to be considered one mode by these methods. Yet later (as shown in Fig. 3 (lower half)), when other classes are introduced, a new class, Class 3, can lie exactly on top of what was considered the "description" (the average point) of the previously determined mode of Class 2 and yet still be completely separable from Class 1.

The use of classification information can be extremely helpful in cases where the finding of the most economical cluster description is important, and where all the classes are initially available.

*Constraints on the Applicability of Cluster-Seeking Techniques Imposed by Various Factors*

Five factors primarily constrain the applicability of a given technique. These are:

1. Computational complexity
2. Memory requirement
3. Sample size requirement
4. Nature of the data
5. Avilability of categorization information

The computational complexity of iterative methods is usually determined primarily by the computations performed in the inner computational loop of the program. Another factor affecting the utility of the method in terms of computation is the ease with which the required computations can be performed by methods other than the conventional general-purpose digital computer. Distance calculations that require the computation of either a correlation or Euclidean distance can be performed using optical correlation techniques.*

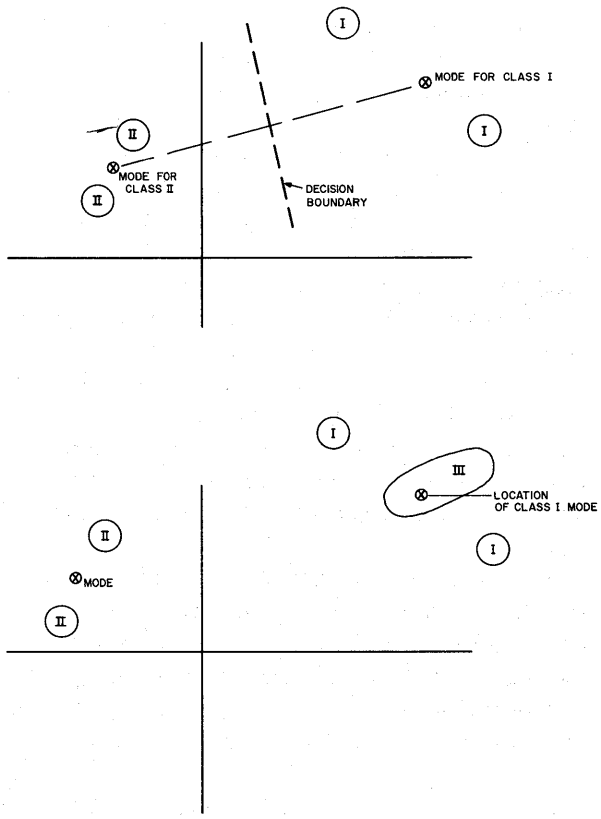



Figure 3. The result of introducing new classes later in mode-seeking techniques.

*The calculation of Euclidean distance between vector $P$ and vector $M$ by using a correlation operation can be shown by the following argument:

The square of the Euclidean distance is $(P - M) \cdot (P - M)$, where $P$ is the pattern and $M$ is the mask, or weight vector. This expression can be expanded into three terms: $P \cdot P$, $M \cdot M$, and $P \cdot M$. If the mask, or weight vector, is not changed after each pattern, then $M \cdot M$ can be considered to be a constant over an entire iteration through all of the patterns. The quantity $P \cdot P$ can be calculated using an optical device that squares a picture in magnitude element by element. And finally the $P \cdot M$ term can be determined by usual optical correlation techniques.

The memory required by a given technique may be so great that the technique is uneconomical. High-speed random access is costly (although disk files make fairly high-speed random access comparable in cost to tape) and it is primarly for this memory requirement that the various iterative techniques should be examined.

The sample size requirement relates to any quantities that must be estimated (in a statistical sense) from the set of sample patterns. Allais[3] showed that this requirement cannot be taken lightly. He indicates that estimating a $D \times D$ covariance matrix with less than $10 \times D$ samples will usually lead to an increase in probability of error if predictions are made based on that covariance matrix. When only small sample sizes are available this seems to require another approach, one in which simpler quanties are estimated—such as the means of clusters, or only the largest eigenvector of the covariance matrix.

The nature of the data is not easily specified. If the data is time-varying it will frequently be more effective (and possibly necessary) to use a different technique from one suitable for stationary patterns. The degree to which the data exists in isolated clusters as opposed to being in "amoebic smears" (i.e., all in one contiguous mass of data) is also important. One of the important research tasks yet to be done is the establishment of better means of classifying data into broad types that suggest the technique that could be most successfully used to further analyze them.

Where categorization information is available *and reliable,* use of it to constrain the clustering seems advised.

### *Measures of Similarity Used by the Techniques*

In relating techniques, consideration should be given to the measure of similarity that was or could be used. Correlation usually requires normalization of the pattern vectors with respect to magnitude. Euclidean distance does not. Euclidean distance is considerably affected by the scale factor associated with each pattern dimension—particularly when these dimensions are not commensurate in units of measurement (e.g., feet vs inches or, worse yet, feet vs seconds).

The measures of similarity used in the papers reported on are given in Table 2. Additional measures of similarity are given on pp. 129-130 of Sokal and Sneath.[40]

Table 2. Comparison of Measures of Similarity.

| Measures of Similarity | | |
|---|---|---|
| Dot Product | $P \cdot W = \sum_{i=1}^{D} p_i w_i = \lvert P\rvert \lvert W\rvert \cos (P,W)$ | Block et al (1962)<br>Steinbuch (1963)<br>Mattson & Damman (1965)<br>Michener & Sokal (1957)<br>Jakowatz et al (1960)<br>Spilker et al (1963)<br>Glaser (1961)<br>Smith (1964) |
| Similarity ratio | where $R_{ij} = P_i \cdot P_j$<br>$S_{ij} = \frac{R_{ij}}{R_{ii} + R_{jj} - R_{ij}}$<br>when $p_{kl} = 0,1.$<br>uses $d_{ij} = -\log S_{ij}$ as "distance." | Rogers & Tanimoto (1960) |
| Weighted Euclidean distance | $D_{\alpha\beta} = \sum_{i=1}^{D} k_i (x_{i\alpha} - x_{i\beta})^2$ | Bonner (1962)<br>Sebestyen (1962) |
| Unweighted Euclidean distance | $D_{\alpha\beta} = \sum_{i=1}^{D} (x_{i\alpha} - x_{i\beta})^2$ | Ball & Hall (1964) |

Table 2. (Continued) Comparison of Measures of Similarity. Measures of Similarity

| | | |
|---|---|---|
| Measure for binary variables taking into account pairwise correlation | $S_{\alpha\beta} = \sum_{i=1}^{D} \sum_{j=1}^{D} r_{ij} [1- \lvert x_{\alpha i}-x_{\beta i}\rvert] \cdot [1- \lvert x_{\alpha j}-x_{\beta j}\rvert] \cdot [1-2\lvert x_{\alpha i}-x_{\alpha j}\rvert]$ <br> $r_{ij}$ is correlation coefficient between measurements $i$ and $j$ | Bonner (1963) |
| Boolean "and" | $S_{ij} = \sum_{i=1}^{D} p_i \cap p_{jk}$ <br> where $\cap$ denotes Boolean "and" | Needham (1961) |
| Weighted Boolean "and" | $S_{kl} = \sum_{j=1}^{D} \begin{cases} r_j, & x_{kj} = x_{lj} \neq 0 \\ 1, & x_{kj} \quad x_{lj} = 0 \\ 0, & \text{otherwise} \end{cases}$ <br> $r_j$ is number of levels of $j^{\text{th}}$ measurement (finite for his data) | Hyvarinen (1962) |
| Normalized correlation | $\dfrac{P_i \cdot P_j}{\sqrt{(P_i \cdot P_i)(P_j \cdot P_j)}}$ <br> where $P_i \cdot P_j = \sum_{i=1}^{D} p_{il}\, p_{jl}$ | Okajima et al (1963) <br> Nunnally (1962) |
| The following six measures of similarity are from Kochen (1963) p. 15, and refer primarily to information retrieval. | $S_{ij} = \dfrac{\log_{10} N(\lvert N \cdot n_{ij} - n_i n_j\rvert - \frac{N}{2})^2}{n_i n_j (N-n_i)(N-n_j)}$ | Stiles (1961) |
| | $S_{ij} = \dfrac{n_{ij}}{n_j}$ | Luhn (1959) |
| | $S_{ij} = \dfrac{n_{ij}}{N}$ | Baxendale (1961) |
| | $S_{ij} = \dfrac{n_{ij}}{n_i + n_j - n_{ij}}$ and $-S_{ij}\log_2 S_{ij}$ | (King-) Tanimoto (1960) |
| | $S_{ij} = N \cdot \dfrac{n_{ij}}{n_i \cdot n_j}$ | Kochen & Wong (1962) |
| | $S_{ij} = -\log \left\{ \dfrac{n_{ij} \cdot N}{(n_i n_j)} \right\}$ and $\dfrac{-n_i n_j}{N^2} \log \left\{ \dfrac{n_{ij} \cdot N}{(n_i n_j)} \right\}$ <br> where $N$ is the total number of measurements and $n_i$ is the number of ones in (binary) pattern $N_{ij}$ is the number of measurements in which pattern $i$ and pattern $j$ are alike. | Abraham (1962) |

*Criteria for Adequacy of Clustering Used by the Techniques*

Some techniques depend critically on the criteria of clustering used. The clumping techniques described below are particularly sensitive to this, since only one iteration is performed and each decision as to the cluster with which a pattern is associated tends to be final.

In iterative techniques this dependence is reduced by allowing the algorithm several iterations in which to associate patterns into clusters. Hence changes can be made based on information obtained in the initial clusterings.

Table 3 gives a list of criteria for clustering used by various authors. In this table $N_C$ represents the number of clusters.

Table 3. Criteria for Clustering.

| | | |
|---|---|---|
| Entropy | $H = -\sum_{i=1}^{r} f(x_j)\log f(x_j)$ where $f(x_j)$ is discrete frequency distribution function. | Hyvarinen (1962) Sebestyen (1964) |
| Average distance from nearest cluster center | $\frac{1}{N}\sum_{i=1}^{N_c} N_i \text{ AVEDST}_i$ where $N_i$ is the number of patterns in the $i^{\text{th}}$ cluster, $\text{AVEDST}_i$ is the average distance of patterns *in* the $i^{\text{th}}$ cluster from the cluster mean, $N_c$ is the number of clusters and $N$ the total number of patterns. | Ball & Hall (1965) |
| Square (used for deviation from single signal) | $\sum_{i=1}^{D} (P_j - \overline{P}_j)^2$ | Brennan (1961) |
| Value for a cluster | $I_{xx} - \frac{1}{N_R}\sum_{y=1}^{N_c} I_{xy}$ where $I_{xj} = \frac{1}{N_x N_y}\sum_{\alpha=1}^{\alpha=N_x}\sum_{\beta=1}^{\beta=N_y} S_{\alpha\beta}$ where $N_x$ is the number of members in cluster $y$, $S_{\alpha\beta}$ is 1 if member $\alpha$ of cluster $x$ is similar to member $\beta$ of cluster $y$; or 0 if they are not similar. $I_{xy}$ is the percentage of possible similarity "links" which are actually present between the members of cluster $x$ and the members of cluster $y$. | Bonner (1964) |
| Coefficient of belongingness | $\frac{\sum_{k=1}^{N_c}\sum_{i,j\,\epsilon C_k}\rho_{ij}\left\{\begin{matrix}\text{correll.}\\\text{coeff.}\end{matrix}\right. \text{ cluster } C_k}{\sum_{k=1}^{N_c}\sum_{i,j\epsilon C_K}\rho_{ij}}$ | Fortier & Solomon (1965) |

Table 3. (Continued) Criteria for Clustering.

| | | |
|---|---|---|
| Total entropy | $E_n[(d_{ij})] = -\frac{1}{2}\sum_{ij} \frac{d_{ij}}{T_n[(d_{ij})]} \log_2 \left[\frac{d_{ij}}{T_n(d_{ij})}\right]$ where $T_n[(d_{ij})] = \frac{1}{2}\sum'_{ij} d_{ij}$ where $\sum'$ means summation only of the finite terms after repeated rows and columns of the distance matrix have been removed and $d_{ij} \equiv -\log_2 S_{ij}$ where $S_{ij}$ is the similarity ratio given by Tanimoto in Table 2. | Rogers & Tanimoto (1960) |
| Probability of error (when defined) | $= 1 - \frac{\Sigma \text{ correct}}{\Sigma \text{ total}}$ | Firschein & Fischler (1963) |

*Lumping of Clusters*

The convergence of cluster seeking techniques is related to the way in which clusters are allowed to form. In particular, if clusters can be created, then some mechanism must be provided for either (1) tightly controlling the formation of new clusters, or (2) "lumping" them together. Without these mechanisms the number of clusters could grow until each pattern was in its own cluster, which would obviously not be very useful clustering. Therefore it is important to examine for each of the clustering techniques the way that the number of clusters is changed. (Note that clusters can be "thrown away" if they become too small, and that this is one additional way of controlling the number of clusters.)

*The Nature of "Convergence" of Cluster-Seeking Techniques*

Considerable experimental evidence exists for the convergence of these techniques. Some analytic work has been done, notably in the probabilistic and signal detection classes of techniques. Nevertheless, we feel it fair to characterize the understanding of "convergence" for cluster-seeking techniques as minimal—particularly with respect to real, non-Gaussian data.

In a general way, it appears that if the data is indeed clustered then the final clusterings will tend to be unique. If, however, the data is "smeared" and "amoebic" then a greater variety of clusterings can exist. Finally, if the data is uniformly placed in data space then no real stable clusters are found—which is to us intuitively satisfying since no clusters really exist in the data.

*Shortcomings of Cluster-Seeking Techniques*

In examining the shortcomings of various clustering techniques, it seems important to ask what affects the way the data is clustered. In Fig. 4 we show that for clustering techniques, the scaling of the various dimensions will undoubtedly affect the way that the patterns are clustered together.

It is, however, possible to normalize the various scales in a way that will lend to a uniqueness of scale with respect to a particular set of patterns. A straightforward way of doing this is to divide each of the measurements by the standard deviation of the marginal distribution for that particular dimension calculated for a given sample of data. Linear transformations of the data, that is, rotations and translations, have different marginal distributions than the same data not transformed. Therefore the scaling on these modified dimensions will be different. It seems probable that in some cases this difference will affect the way that a clustering occurs. Again, most of these statements are qualitative in the sense that if well-defined distinct clusters exist in the data, *moderate* scaling, rotation and translation probably will not affect the clustering greatly. If, however, as is frequently the case, the data is not tightly clustered but has clusters blended into each other, then these effects become more pronounced.

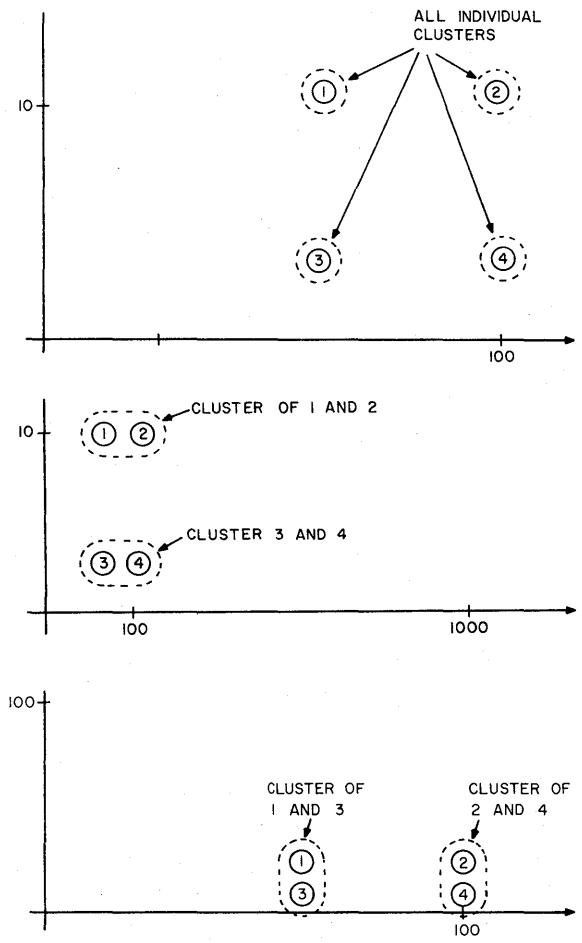



Figure 4. The effect of scaling of measurements on clustering.

Clustering techniques that depend on a correlation measure of distance can be greatly affected by the translation of the data. For example, consider a set of data lying on a hypersphere, surrounding a central point. If the origin is moved outside of the hypersphere then data lying along a given angle from the origin will not consist of a unique part of a hypersphere, but rather of two parts of the hypersphere, and in fact, rather remote portions of the data will be clustered together, if only direction from the origin is used to determine pattern similarity (as is the case with correlation).

The distance measure used to determine similarity or closeness has a considerable effect on the way data is clustered together. For example, if Euclidean distance is modified by having constant multipliers times the various components, where the constant multipliers differ from cluster to cluster, it is then possible to have rather peculiarly shaped volumes assigned to the same cluster. In Fig. 5 we see that it is possible to have a large dispersed cluster surrounding a small, rather compact cluster. The "distance" in Fig. 5 is determined by the value of a Gaussian probability density. If the purpose of clustering is classification, then this peculiar shape of cluster may be acceptable. If, however, the purpose is to describe the data, then it seems that convex clusters would be most useful.

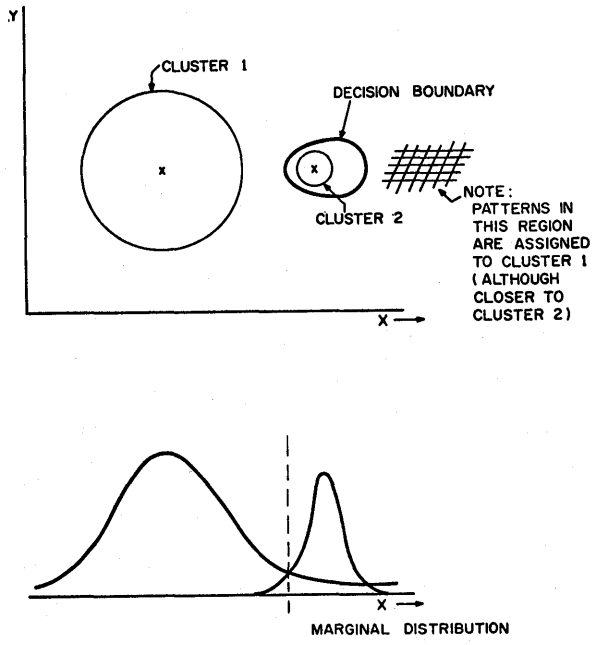



Figure 5. The decision boundaries that result when the same measurement is differently scaled for different clusters.

## MAJOR CLASSIFICATIONS OF CLUSTER-SEEKING TECHNIQUES

The various cluster-seeking techniques have been broken down into seven categories:

1. Probabilistic
2. Signal detection
3. Clustering
4. Clumping
5. Eigenvalue
6. Minimal mode seeking
7. Miscellaneous

The salient characteristics of each of these classes is described in the following paragraphs and summarized in Table 4.

Table 4. Techniques for Finding "Similar" Subsets in Data.

| Type | Proponents | Salient Characteristics |
|---|---|---|
| Probabilistic | Daly 1962)<br>Fralick (1964) | Estimation of probability of occurrence of a pattern using decision theory and then using weighted combinations of patterns to estimate probability distributions. |
| Signal detection | Jakowatz, Shuey & White (1960)<br>Glaser (1961)<br>Spilker, Luby & Lawhorn (1963)<br>Smith (1964) | Detection of presence of signal using energy detection, then estimation of parameters of matched filter (correlator). |
| Clustering | Okajima et al (1962)<br>Sebestyen (1962)<br>Hyvarinen (1962)<br>Ball & Hall (1964) | Finding of minimum distance (or maximum correlation) between pattern and one "cluster center" out of a set of cluster centers. Iterative improvement of the position of these centers. |
| Clumping | Michener (1957)<br>Sokal (1957)<br>Rogers & Tanimoto (1960)<br>Needham (1962)<br>Sawrey, Keller & Conger (1962)<br>Bonner (1964) | Use of closest pair of patterns to form nucleus for a clump of patterns. "Growing" of clump around this nucleus. |
| Eigenvalue I | Cooper (1964)<br>Mattson (1965) | Finding clusters by finding maximum eigenvalue of covariance matrix and splitting patterns on basis of correlation with corresponding eigenvector. |
| Eigenvalue II | Nunnally (1962) | Finding clusters by finding eigenvalues of distance matrix and then examining patterns with respect to this new basis, i.e., by seeking eigenvectors with which many patterns are highly correlated. |
| Minimal mode-Seeking | Firschein & Fischler (1963)<br>Steinbuch (1963) | Use of category information to provide impetus to formation of new modes for a category, i.e., incorrect categorization implies needs for new mode. |
| Miscellaneous | 1) Block, Knight & Rosenblatt (1962) | Uses high probabilities of contiguous patterns (in a time sequence) being in the same class to provide a (somewhat noisy) classification of the patterns. |
| | 2) Bledsoe | Seeks to find the set of hyperplanes passing through "corridors" in the data that have maximal average distance from the patterns. |

*Probabilistic Techniques*

Probabilistic cluster-seeking techniques are primarily analytic studies in the sense that the main results are couched in analytic terms and were derived from decision theoretic considerations. Both papers describe experimental results obtained by implementing the algorithm implied by the decision theoretic mathematics.

Quite possibly this analytical work will provide considerable insight to other techniques in terms of convergence characteristics and imply ways that they might be modified. It also seems probable that the clustering techniques will suggest new formula-

tions of the mathematics in terms of relaxing restrictions or simplifying calculations through the use of approximations.

*Signal Detection*

Signal detection cluster-seeking techniques grew out of a desire to detect unknown signals in noise. The final decision is based on correlation detection. The techniques all suffer from the necessity of making an initial detection on the basis of the energy of the signal and then determining the direction of the signal in signal space. For the most part, these techniques consist of the detection of a single signal of unknown epoch in a noisy environment when the time of occurrence of this unknown signal is unknown.

*Clustering Techniques*

Clustering techniques can be characterized by the sorting of patterns by use of multiple cluster points. Tentative assignments of patterns are made to clusters and these assignments are improved until the means of the clusters "adequately" describe the data. The particular method chosen for modifying the description of this data is the primary distinction between these techniques.

*Clumping Techniques*

In these techniques a single pair of patterns usually the closest pair, is selected as a nucleus for a clump of patterns. Other patterns are assigned to this clump on the basis of their closeness to the pair of patterns, or to the mean of the pair of patterns. Generally speaking, these techniques require the calculation of all pairwise distances between all pairs of points and some of these distances must be recalculated after each new combination. In some of them, these distances must be stored in random access memories. The large amount of calculation caused by the calculation of all pairwise distances, even between pairs that are quite remote, makes these techniques less useful than they might otherwise be.

One application for which such techniques seem particularly valuable, however, is that of developing taxonomies. In these cases, one usually has a limited number of samples (with the possible exception of bacterial species) and the problem is tracing them back along an evolutionary tree, combining branches as clumps become close together. It appears that the same information can be obtained by clustering techniques as is obtained by clumping techniques.

*Eigenvalue Techniques*

Eigenvalue techniques are the only techniques we have characterized as non-iterative. Eigenvalue techniques usually depend on the estimation of the covariance matrix. For this reason these techniques tend to require a relatively large number of samples, particularly as the number of dimensions grows large. Large amount of core storage may be required to store the matrix analyzed. The factor analytic techniques of this sort content themselves with one calculation using all of the patterns and a diagonalization of the distance matrix. The later techniques of Mattson and Damman[27] and Cooper and Cooper[12] use a calculation of only the largest eigenvalue and the corresponding eigenvector, and then subdivide the pattern set in order to proceed further, with a more detailed examination of each of the subsets.

*Minimal Mode Seeking*

These techniques require categorization information to work. A new mode is created only when patterns in one class are nearer to a mode of a different class. Pattern density in space, as such, is not used in cluster seeking.

*Miscellaneous Techniques*

Two techniques do not fit neatly into any of the above categories:

1. The technique of Block et al[7] utilizes a high probability of contiguous runs of patterns in a time sequence being from the same class to adjust the machine to a particular mode. This high probability of runs provides marginal teacher information in a probabilistic sense.
2. Bledsoe[6] seeks corridors in the data.

## INDIVIDUAL TECHNIQUES

In the following portion of the paper we describe briefly the known cluster-seeking techniques. We

will use the following symbols in describing the difficulty of computation or the amount of storage needed.

| | |
|---|---|
| $N$ | The number of patterns. |
| $D$ | The number of dimensions. |
| $NROWS$ | The number of clusters the process finds. |
| $C$ | The number of interval blocks the pattern space is broken up into. |
| $ITER$ | The number of iterations required for the technique to describe the data satisfactorily. |
| $N_T$ | The total number of patterns used when the patterns are repeatedly drawn from the same probability distributions. |

*Probabilistic Techniques*

DALY:[15] Estimates probability of occurrence of all possible binary sequences. Updates estimates after each pattern sample. Computational complexity grows as $e^N \times D$. Convergence has been shown experimentally by computer simulation. Assumption is made that we are looking in a noisy environment, for a single signal that occurs only occasionally. System definitely limited by the exponential growth of the memory of the system as the number of samples increases.

FRALICK:[18] *Recursively* computes a posteriori probability density of distributional parameters by using a sequence of learning samples. Computation goes as $N \times D \times NROWS \times C \times ITER$. Convergence shown analytically for the case of detection of an unknown signal in noise. Appears to assume that the number of pattern classes is known, and appears willing to develop multimodal distributions in order to handle unexpected classes of patterns. Method limited by the need of storing its latest estimate of probability distribution of the samples, that is, the description of the probability density in terms of the frequency of "cells" in high dimensions or by assuming a particular form of distribution. In multivariate problems the storage of these distributions becomes quite serious and quite difficult. The amount of computation required to update all of the distributional parameters can be considerable unless simplifying assumptions are made regarding the nature of the underlying distribution.

*Comments on Probabilistic Technique.* In our opinion these techniques are more useful as guides to the development of practical techniques than as actual formulations of practical techniques themselves. Either the large amount of storage or the large amount of computation seems to imply that the techniques are outside of practical limits for high-dimensional multivariate data.

Fralick[18] points out (P. 13) that in the case where the class a priori probabilities are all the same, the initial selection of the probability distributions for the various classes must be different, or "all computer branches will 'learn' the same thing, and the system as a whole will learn nothing."

A third paper related to this work is that by Patrick and Hancock.[33] This paper presents no new techniques that this author considers as mode-seeking, or learning-without-a-teacher techniques. It does closely examine the efficacy of various estimators that might be used under conditions of partial knowledge concerning probability distributions. The assumptions made in the paper seem to be so strong as to make the paper of relatively little practical significance. Many of the techniques discussed in this paper are for a single-dimensional problem. It does not appear that they generalize usefully to multivariate situations.

A pertinent quote from this paper[33] is as follows: "It is seen from the above discussion that the performance of different learning systems should be compared on the basis of what a priori information is required for their operation."

A second quote[18] relating to the same subject is: "we still require *some* a priori knowledge in order to be able to obtain the very first decision rule."

It does appear that the question of how much a priori knowledge is necessary to provide what might be called "learning" is one that should be looked at quite carefully.

We feel that the examination of this work motivated by decision theory may be very useful if related to the more ad hoc methods of clustering and finding the structure in data that will be discussed below. For example, in the ISODATA procedure if a pattern falls roughly equidistant from two cluster centers it is only added into the closest cluster center. Fralick's work seems to suggest that in this case it might be well to add portions of this pattern into both clusters.

*Signal Detection Techniques*

JAKOWATZ, SHUEY AND WHITE:[21] A sample of the input waveform is stored in the memory of a correlation detection device. When the dot product of the incoming waveform and the stored waveform in the correlator exceeds a threshold the waveform stored in memory is modified by adding in a certain fraction of the waveform as it exists at the time the dot product is maximally greater than the threshold The threshold grows with successful detection and decays with failure to detect. Parameters controlling the algorithm include the threshold level, the memory weighting factor, the threshold setting factor, the threshold decay factor, the filter length, and the bandwidth of the filter. It uses only one correlator. Computation goes as $N_T \times D$. Convergence shown analytically and experimentally.

This technique and those in this section of signal detection techniques are primarily used for signal detection, and as they are presently conceived, their utility outside of this area seems limited (with the possible exception of the proposals by Smith).

GLASER:[19] The incoming waveform is detected on the basis of its energy. When a signal has been detected because of its energy, it is set into the correlation detector. Future decisions are based on a mixed weighting of detection by the energy detector and by the correlation detector. Detection threshold, memory weighing factor, and incoherent vs coherent detection factor are the parameters provided in the program. Computation goes as $N_T \times 2D$. Convergence shown analytically.

SPILKER, LUBY AND LAWHORN:[41] The incoming signal is quantized into three levels. Initially, if the waveform has energy greater than a threshold, then the correlation detector is modified. Future detections then depend on a mixed weighting of the energy detector and the correlation detector. Parameters provided are the detection threshold, the memory weighting factor, and the incoherent vs coherent detection factor. Uses only a single correlator. Computation goes as $N_T \times 2D$. Convergence shown analytically and experimentally.

SMITH:[38] Detects the waveform using energy detection. Energy detection causes the waveform to be stored in the correlator. Gradually correlation detection takes over. If multiple signals are to be detected, then one correlation is trained first and then a second correlator is trained after the first one is trained. Detection threshold, memory weighting factor, incoherent vs coherent detection factor are the parameters provided. A second cluster is formed if the first correlator does not detect a signal when the energy level threshold is exceeded. No mechanism is given for destroying clusters. The computation goes as $N_T \times 2D \times NROWS$. The computations required by Smith's method are more complicated than Glaser's or Spilker's. It is not apparent that a great deal is gained by adding the complexity. The method of adding new clusters does not appear to be very effective in problems having a large number of clusters and it might be damagingly ineffective.

*Comments on Signal Detection Technique.* By using more memory to store possible patterns it might be possible for these techniques to avoid the difficulties caused by using energy detection to start the process going. That energy detection is hindering can be seen in the foregoing simple example, taken from Spilker et al. Spilker[41] (p. iii) mentions a definite thresholding effect on the performance of the filter at a -13 db. Calculations show that the marginal distributions for each component overlap to a very considerable extent (about 45 percent probability of error for decision made on the basis of any one component by itself). It can be shown, however, that if one is able to use a signal with a thousand components in it, then the probability of error when the problem is taken as a multivariate normal decision problem is only 0.0002, which is very small. When energy detection must be performed, the probability of error is considerably increased above this small number. By storing a number of signals a guess as to possible cluster locations can be made without using energy detection.

One point that Glaser[19] (p. 93) raises seems important enough to quote:

> For weak signals the component estimates will tend to be large and induce erratic behavior in the adaptor system. It appears that there may be a minimum signal strength which the signal system can adapt to, and this also may be a function of signal waveform. Signals above this minimum will be successfully adapted to at a rate which decreases as signal strength increases. No calculations on this minimal adaptable signal have been made.

This hypothesis seems worth further investigation. It appears to this author that techniques that avoid the necessity of making threshold decisions based on total signal energy may significantly improve performance.

Brennan[9] (p. 70) quotes three factors as determining convergence for the system of Jakowatz, Shuey and White:

1. False alarm rate less than signal rate.
2. Probability of finding second signal greater than ½.
3. Best estimate of certain coefficients. These coefficients depend entirely on the memory factor $\gamma$.

. . . .

The final optimum which the filter can reach. . .is dictated entirely by the memory factor.

He further (p. 72) comments that it is desirable for the filter to be not quite twice as long as the signal, and that the reduction in signal-to-noise level due to increasing $W$ ($W$ equals filter bandwidth) might then be well compensated by rapid convergence, or even making convergence possible (p. 73). He goes on to point out the importance of keeping the memory weighting factor low when the nature of the signal is not well known so that erroneous signals can be easily erased (p. 75).

Smith[38] uses the squared dot product between the filter and the incoming figure, as well as the dot product unsquared. It is difficult to see what effect this has on performance (except to make calculations more complicated). He does show an interesting example (on p. 21): "It can be seen that the lack of timing information makes it necessary to examine the paths of the signals throughout the space to be sure that there is no ambiguity in the decision regions."

Here he is pointing out that long signals entering into the correlator may appear to be a different short signal due to the limited number of sample points in the filter. Therefore it is necessary to examine translations of a signal to see if this type of ambiguity exists.

Smith has an interesting approach to detecting distinct multiple signals. He suggest, first, detection of one signal until an adequate representation of that signal has been obtained, and then the use of that representation to distiguish the first signal from a second signal. Clustering techniques indicate that this can be avoided by having more than one cluster point, or cluster vector, to adjust at a time.

It would appear that all the techniques are troubles by the epoch (or translation of the time origin) problem. Perhaps this also could be overcome by a greater use of storage (sufficient to store every shift of the signal during the time when the signal occupies a central position within the correlator). This suggests the idea of a double correlator, a correlator within a correlator, to determine when the signal is centrally located in the larger filter.

A paper making some interesting philosophical points, though following Jakowatz' technique, is that by Turner.[44]

*Clustering Techniques*

OKAJIMA, STARK, WHIPPLE AND YASUI:[31] Patterns are selected in random order, weighted, and compared with a set of "typical" patterns, using a normalized correlation measure of similarity. The typical pattern correlation that is maximum is compared with a threshold. If the threshold is exceeded, then the pattern is added into that filter. If the threshold is not exceeded, then a new filter is created. A smoothness of convergence parameter as well as a similarity threshold is used to control the program. Clusters are not destroyed in the process. Cluster points are the average of all of the patterns associated with that cluster point. Computation goes as $D \times N \times NROWD \times ITER$. The clustering is dependent on the order in which the individual patterns are presented to the program. This appears to be quite an excellent technique.

SEBESTYEN[36] and SEBESTYEN AND EDIE:[37] A pattern is selected and compared with the existing cluster centers. The measure of similarity is a weighted Euclidean distance with a weighting by factors that depend on both the particular component and on the particular cluster. The minimum distance of the pattern from a cluster point is compared with two thresholds, one smaller than the other. If the smaller threshold is not exceeded, then the pattern is added into that particular cluster and a new mean computed. If the smaller is exceeded but the larger is not, then the pattern is rejected for the present time, to be used at a later stage in the process. If the pattern distance exceeds the second threshold, a new cluster is created. Parameters controlling the processing include the two thresholds, and a parameter that controls when rejected patterns are forced into the various clusters. Computation goes as $D \times N \times NROWS \times ITER$.

This technique is specifically pointed toward providing as an output a probability distribution based on the sample data and the technique's complexity is increased by this goal.

HYVARINEN:[20] By using a typicality measure, a single pattern is selected as a starting point. Other patterns that are within a certain distance of this pattern are clustered with this pattern. All patterns

that are clustered together are then removed from the total set of patterns. This reduced set of patterns is then examined for another most typical pattern on which to start another cluster. The threshold controlling similarity is a process parameter. Cluster centers are never modified after the initial selection of a typical pattern. Computation goes as $D \times NROWS \times N$. Computation of typicality of patterns may prove expensive for large pattern sets. It would appear that this technique requires data that is more structured than other clustering techniques of this type. The measure of typicality critically determines the effectiveness of this procedure.

BALL AND HALL:[4] Several patterns are selected as trial cluster points. All patterns are then clustered around these trial cluster points and an evaluation of the clustering is made. If the maximum marginal standard deviation for any cluster is too large, and if certain other conditions are satisfied, then the cluster is broken into two clusters by creating a second cluster point out of the first one. A second part of the process combines cluster points that have come closer together than a given threshold. The parameters that control a program are the minimum allowable size of a cluster, the allowable maximum standard deviation in the cluster, and the minimum distance between two cluster points. Clusters are formed by splitting existing clusters. They are destroyed when two cluster points are lumped together or when a cluster gets too small. Trial cluster points are modified only after all patterns in the set have been clustered. The computation goes as $D \times N \times NROWS \times ITER$. It can be generalized to include clustering about line segments and planar sections. Block diagrams of optical implementation have been developed.

*Comments on Clustering Techniques.* Hyvarinen's technique appears to have one major drawback: it is not iterative. If the measure of typicality used is thrown off by some peculiar structure of the data, then it appears possible that clusters could be obtained that would be quite lopsided and not as intuitively pleasing as ones obtained iteratively.

The techniques of Okajima et al and Sebestyen seem very similar in that in both of them distance from a nearest cluster point is compared with a threshold. If this threshold is exceeded, then a new cluster is formed. If it is not (with the exception of the guard ring in Sebestyen), then the pattern is added into that closest cluster. The similarity was stronger in the early forms of Adaptive Sample Set Construction (see Sebestyen[36]), than as modified in Sebestyen and Edie.[37] The later technique has many fine features that now allow it more truly to represent the data as an empirically derived probability distribution.

Sebestyen's technique differs from the other techniques in his use of a metric that is sensitive both to the individual cluster and to the dimension in which a measurement is being made. This should provide greater flexibility in its description of the data. If, however, one of the purposes of the technique is to provide a description of the data, then this may not be an advantage in that it may be more difficult for an individual attempting to visualize the structure of the data to bring into play these factors than it is for him to accept additional custer points. (See discussion under "Shortcomings of Cluster-Seeking Techniques" above.)

The technique of Ball and Hall differs from the other clustering techniques in several ways. First no absolute threshold is used in the measure of similarity criterion of the technique. A pattern is assigned to the closest cluster point regardless of distance. (This has disadvantages in terms of wild shots that lie a great distance from any cluster center. However these rapidly become cluster centers by themselves and then no longer bother the process.) The criteria used to evaluate the "goodness" of the various clusters and in the splitting of the cluster appear to be conceptually different. The lumping in the Ball-Hall technique produces a similar result to that obtained by Sebestyen when he combines sample sets that are not contributing significantly to classification accuracy by their being distinct. In some sense the goals of the two techniques are different, in that Sebesyen's technique is to develop a probability density that can be used in making classifications, whereas the Ball-Hall technique is pointed toward an adequate representation of the data, initially, at least, without regard for the classification to be performed. A further distinction between these techniques is the sorting of all of the patterns by the Ball-Hall technique prior to the modification of any of the sorting criteria. This makes the technique independent of the sequence in which the patterns are presented.

One note of warning should be added regarding the use of metrics weighted with respect to cluster

as well as component. For example, if distance were measured as in ISODATA without using a threshold, one would find that the decision boundaries no longer had the simplicity and intuitive appeal of the perpendicular bisecting plane. Now it would be possible for a small dense cluster to be located in the interior of a larger cluster that was not so dense. Possibly this would be an advantage but it would make interpretation more difficult when the technique is used for data analysis.

Okajima et al provide a contribution in the use of their smoothness of convergence criteria. This is used to make it more and more difficult for a pattern to be added to a cluster as the cluster grows larger.

It is felt by the author that the Ball-Hall technique is so structured that it is better suited to generalizing to new types of clusterings, for example, clustering about line segments or planar sections.

One of the present weaknesses of the Ball-Hall technique is the weakness of the existing criteria for determining the adequacy of clustering. Hyvarinen's suggestion of using an information theoretic measure to determine the most typical pattern in order to choose a best starting place for a clustering is an interesting one.

Stark makes an important point (see Okajima, Stark et al,[31] p. 108) when he discusses how a modification of the metric of the classification space will cause a different grouping of data vectors to occur. This is one of the most vexing and apparently unsolved problems in clustering techniques having no externally supplied guidelines as to the relative importance of the various measurements. It appears possible that some rationale for the selection of scales could be developed by examining the changes in clustering obtained as a result of changing the scales of the various patterns.

Stark (p. 109) points out that modifications in the clustering caused by changes in order of the pattern presentation may provide some indication as to the structure of the data itself.

It is the opinion of the author that the clustering type of cluster-seeking techniques may well be the most profitable direction in which to extend our efforts. Much more analysis of these techniques has to be done. Possibly we will then discover that these techniques are controlled by a form of feedback supplied by the structure of the data itself.

*Clumping Techniques*

MICHENER AND SOKAL:[28] The nucleus of a cluster is established using those two patterns having the highest pairwise coefficient of correlation. Then patterns are added to this nucleus one at a time, always adding first the pattern with the highest average correlation with the members of the group. The limit of the groups could be found by decreases in the level of the average correlation. A "significant" drop was empirically determined. Correlations between small clusters were used to group these small clusters into larger clusters. Computations go as $D \times (N(N-1)/2)$.

This technique and the two next techniques require a number of comparisons to be made and additional computations to be performed. The extent of these computations is difficult to determine without examining specific data sets. The calculation of all pairwise distances between all patterns is considered to be a relatively inefficient way to obtain the structure of the data. This technique is best suited to taxonomic applications.

ROGERS AND TANIMOTO:[34] A function related to information theoretic entropy is computed and minimized. This selects the point pattern nearest the centroid of the system of patterns. This most typical pattern is designated "a prime mode" and a clump of cases very similar to it are clustered around it. Patterns are added to this clump until the inhomogeneity measure suddenly takes a large jump in value, indicating that the last pattern added was not truly a member of this clump. Computations go as $D \times N(N-1)/2$. Random access memory required is $N(N-1)/2$. Patterns considered are binary in value.

NEEDHAM;[29] PARKER-RHODES;[32] ABRAHAM:[1,2] The distance between all pairs of patterns is computed for the pattern set. A threshold related to the average distance between the patterns is computed and the similarity matrix (i.e., the matrix computing all pairwise distances between patterns) is thresholded at some level, reducing the matrix to a matrix of zeros and ones. The $i^{th}$ row thus contains a list of all objects to which the $i^{th}$ object is closer than the threshold. It is possible, working with these lists of objects, to obtain clusters of interrelated objects. One method of doing this is to take a list of patterns related to one pattern, to take the next pattern on the list, and to take the intersection between the two lists of these two patterns, etc., until the number of patterns at the intersection is satisfactory. In this way fundamental clumps can be constructed, which

can then be used to build up other clumps around them; that is, these clumps can serve as nucleus clumps. One parameter of the program is the threshold above which patterns are declared related, and below which, not related. Computations go as $D \times N(N-1)/2$. Random access memory required is approximately $N(N-1)/2$. This technique seems primarily useful for taxonomic problems. The techniques proposed are suggested both intuitively and by the use of graph theory.

BONNER:[8] Two methods are proposed. The first computes a similarity matrix, giving the distance between all pairs of patterns, and then manipulates this similarity matrix. The second, which is the one we shall describe, as it appears to be more satisfactory, takes a "random" center pattern and builds a cluster around this by developing an arbitrary threshold. All members more similar to this center pattern than the threshold are considered to be in the crude cluster. The members of this crude cluster are then examined by more refined criteria to determine whether or not they should be considered as part of this cluster. The yardstick of cluster goodness used is a measure of the rarity of the cluster. The computation goes roughly as $D \times N$. Additional computations are required that depend on the character of the data. Program requires random access to all the set of patterns in order to improve the clusters obtained efficiently. Patterns used were binary. The use of a threshold around an arbitrary pattern seems like a useful way to restrict the number of measurements of distance that must be made between pairs of patterns, if the distance between pairs of patterns is to be measured.

FORTIER AND SOLOMON:[17] Computes the correlations between all pairs of patterns. The absolute value of the difference between this correlation squared and an arbitrary constant is stored in a "distance" matrix. The constant used should be related to the loss expected if two items are clustered that are not strongly related. The matrix is then summed over all clusters of pairs. That is, compute a number that is the sum of the "distance" between all pairs of patterns in a given cluster. Sum this number over all clusters. The attempt would then be made to maximize this final double summed number. Computation goes as $D \times N(N-1)/2$ plus computations necessary to maximize the sum of the differences between these various correlation coefficients and the threshold for each of the clusters formed. The techniques discussed in this paper seem particularly to bear out the combinatorial problems in clustering. They emphasize the necessity for making approximations and for iteratively computing the desired quantities rather than trying to compute it using all the possible information. These techniques seem quite limited with respect to the problems they can attack because of the restrictions on the number of clusters the techniques proposed can consider.

SAWREY, KELLER, AND CONGER:[35] The procedure selects small subsets of nearly identical patterns using the Euclidean distance between patterns. Based initially on a small number of basic profiles, highly homogeneous groups which are also dissimilar to each other are formed. To these homogeneous groups, the remaining profiles in the sample are then compared. As a result of these comparisons, additions are made to one or another of the groups by successively relaxing the criteria of group homogeneity. Profile groups are formed in which the group members are similar to each other but at the same time dissimilar from the members of all other groups. The paper gives quite a thorough analysis of the application of these clustering techniques to psychological data. The computations involved will depend quite definitely on the nature of the data though all pairwise distances must be computed.

*Comments on Clumping Techniques.* The techniques. The techniques of Michener and Sokal[28] were developed for the specific purpose of obtaining a numerical taxonomy of a collection of animals, in this case, bees. The technique itself is highly oriented toward attaining this taxonomy, and as a result seems less useful for the more general problems of clustering than some of the other techniques. The comments found in a later book by Sokal and Sneath, *Principles of Numerical Taxonomy*,[40] are worth reading, however, and for those interested in approaching the formation of subsets by providing a nucleus pattern or patterns and clumping around them, this book is an excellent source of references. One quote from Michener and Sokal follows:

> By using a large number of characters and species (i.e., measurements and patterns) however, we feel that weighting becomes unnecessary because the magnitude of correlation coefficients calculated between species (distances between patterns) would be little affected by extreme weighting.

This seems to indicate they feel that in some sense the most valid general clustering one can obtain is by using a large number of descriptors of the pattern. (With limited sample sizes increasing the number of measurements may, however, lead

to a decrease in "clusterability.") However, in the case of clusterings toward a particular end — for example, clustering weather measurements to obtain indications of high ceiling and low ceiling heights — the weighting of one or another measurement is then tied to the particular task for which the system is intended. For this reason, this particular comment seems most valid only when trying to develop a general taxonomical relationship between the various patterns.

The paper by Rogers and Tanimoto was one of the earliest papers in which a computer was a necessary and integral part of the process used. The program developed seems particularly significant in its attempt to allow the operator to introduce new cases or attributes whenever these are found to be irrelevant to the clustering to be performed. A second interesting aspect of this paper is the use of information theoretic measures to determine the quality of the clustering obtained. If these measures from the clumping techniques were combined with a more efficient method of associating patterns together, such as is found in the clustering techniques, it appears that the resulting algorithms would be particularly powerful for sorting patterns into subsets.

The papers by Parker-Rhodes,[32] Needham,[29] Abraham[1, 2] and Kochen[24] are all concerned primarily with the use of what might be called graph theory for the determination of clusters in a set of data. A common characteristic of these papers is the use of a similarity matrix as a sufficient description of the universe of patterns. As defined in these papers: "Broadly, members of a clump must be more like each other and less like non-members than elements of the universe picked at random" (Parker-Rhodes,[32] p. 9).

Much of the work in these papers is set-theoretic and attempts to determine some rather general concepts concerning what can be considered a clump and what is not a clump. The major problem of these techniques is that they all increase in computation at least as the square of the number of patterns which is a considerable limitation on large problems.

The paper by Bonner[8] contains a large number of insights into the clumping approach to obtaining subsets. Many of these comments are useful regardless of the technique class that is being used. Three algorithms are described in this paper. We have described only the third in any detail. Additional measures are provided in this paper for measuring the strength of internal clustering vs the strength or external interactions. Bonner suggests the following criteria for clustering:

> The clustering problem . . . becomes the problem of finding sets of objects where the attributes are estimated to be independent within a set . . . The philosophy of the clustering procedure to be followed involves finding sets of objects which do not come from this population. If none can be found, the original object set is judged to be one cluster. If any is found, it is removed from this set as a cluster.

Bonner then describes a statistic that can be used to determine the probability that the objects clustered together, if picked at random from a hypothetical population, are statistically significant.

His major conclusion regarding clustering techniques and factor analysis is an interesting one.

> The major point to be made is that clustering methods . . . can be used for problems now done by factor analysis. It is not implied that such a cluster analysis should replace factor analysis, but that both methods applied to the same data should yield a deeper understanding than either method alone.

The paper by Fortier and Solomon[17] is notable in its spelling out of the difficulties involved in handling a clustering of a large number of patterns in high dimensions by direct methods. As they say (pp. 3-4):

> Theoretically speaking, we have a finite problem at hand for which there exists a solution: i) prepare a list of all possible partitions of a set of $N$ points. ii) for each partition (set of clusters) calculate the $B$ function, iii) choose the clustering configuration which corresponds to the optimal (large) $B$ value. However, matters get out of hand rather quickly, for the amount of computation becomes inordinately large, even when $N$ is moderate in size.

The paper contains many tables showing just how out of hand things can get if this point of view is taken, and some approximations are not made.

Two other papers worthy of mention are Cattell[10] and Kaskey.[22] These two papers deal primarily with the analysis related to clustering measurements as opposed to clustering patterns, and for this reason have not been included in detail. The paper by Cattell outlines some of the methods that were used prior to the advent of the electronic computer. The paper by Kaskey makes considerable use of the electornic computer and develops a significance test to be used on the correlations between various measurements. One of the limitations of both of these approaches seems to be the implicit assumption that the set of patterns being analyzed is in some sense

homogeneous. This need not be true; the correlations attained between measurements may well be that the result of two measurements is high, while in another of the groups it may be small. This certainly seems quite probable in cases in which a relatively large number of patterns is being used and in which a priori knowledge is not really adequate to divide these patterns into homogeneous subsets. It is therefore recommended that these procedures be amended to, first, cluster patterns into relatively homogeneous subsets, and secondly, to perform the clustering of the measurements. While the interpretation of the results of such an analysis would probably require new attitudes towards the information contained in the data, it does seem that the results obtained would be much more directly related to the physical phonomena underlying the data.

Cattell also notes in his article that Holzinger's *B* coefficient techniques might be used to relate the density within a cluster to the density of patterns immediately surrounding a cluster. Both Rogers and Tanimoto, and Michener and Sokol use the concept of adding patterns one at a time, until an inhomogeneity measure increases sharply over previous increases. This seems related to the same concept of learning how isolated a cluster is from the other patterns. In analyzing some real data, we have found that while isolated clusters do occur, an interesting and useful description of the data can be obtained by clustering (in the sense of relating together a group of similar patterns) even though each of these "clusters" is not really isolated from the surrounding patterns.

The general comment regarding clumping techniques is that they are in general non-iterative in the sense that there is a specific procedure that determines when a pattern is added to a clump. The use of a once-through, non-iterative procedure requires the selection of a stricter criteria of what constitutes a cluster than would be required if an iterative technique were used. As suggested above, it does appear that the combination of a good criteria of what constitutes a cluster with an iterative technique should provide an extremely powerful synthesis of clustering and clumping techniques.

*Comments on Eigenvalue-Type Techniques*

Nunnally[30] is in some sense intermediate between the clumping technique and the eigenvalue techniques. His technique is similar to the clumping techniques in that he uses the distance matrix, giving the pairwise distances between all of the patterns, and is similar to the eigenvalue techniques in that he uses the principal directions in the distance space to define the clusters, i.e., he uses the eigenvectors of the distance matrix.

This technique is particularly interesting in that it ties together the more classical approach of statistics, e.g., factor analysis, and the principal components analysis, to the techniques of clustering and clumping. Nunnally has several useful comments on the importance of considering what he calls level (signal amplitude), shape (which corresponds to direction in our pattern space) and dispersion (which is related to the variance of the pattern around its mean level); these comments are more directly related to psychology than to other types of patterns. Nunnally's work is in some ways reminiscent of the work of Bonner.

One quote from Nunnally seems particularly relevant:

> The decision to use profile (cluster) analysis is determined in part by preferences for methodologies, which are, in essence, wagers about the likely research payoff in the long run, from choosing one method of investigation rather than another. The reader can judge for himself whether the studies using comparisons of profiles (e.g., measures of "assumed similarity" in interpersonal perception) have borne the expected fruit.

With the availability of computers increasing and the cost decreasing, it seems very worthwhile to use both the more classical factor analytic or eigenvector methods and the methods of profile or clustering analysis on any given set of data. We feel that the two view points are sufficiently different that they can each supply important information regarding the data.

The paper by Cooper and Cooper[13] appears to depend rather heavily on a number of assumptions regarding the nature of the data. While these assumptions are necessary for their analytic examination, and while they have suggested ways of getting around some of the assumptions that they have made, we feel that much work remains to be done before this technique will be of value when applied to problems in which little is known about the data. At this point it is worth noting the relationship between the technique of Mattson[27] and that of Cooper and Cooper. In some sense it appears that Mattson has taken the underlying philosophy of the Cooper and Cooper technique and extended it to problems where little is known about the structure

of the data. This relaxation of assumptions make the Mattson technique particularly useful. The most prevalent assumption in the Cooper and Cooper paper is the one of the data beging formed from two Gaussian distributions. In this respect it is reminiscent of the approach of Patrick and Hancock.[33]

An interesting quote from Cooper and Cooper[13] (p. 419) is the following:

> As is to be expected, non-supervised adaption cannot be uniquely achieved for abitrary distributions. But where there is adequate probability structure of the problem, the partition can be unique. There are many cases for which this is possible. Of special importance is the two-category case where the distributions are translates of one another, but have general functional form, and further interest centers on the cases where the distributions are finitely parameterized.

While it is true that some of the clustering techniques that have been discussed do not arrive necessarily at a unique solution in a mathematically rigorous sense, it is nonetheless intuitively clear that in well-structured data the partitions achieved will be what might be called "epsilon-unique," that is, the partitions achieved will vary only slightly (for the same parameter settings of the clustering program). The most pertinent variable in determining the size of epsilon in this case is the inherent structure of the data. If the data exists in clusters that are compact and well isolated from each other, then the partitioning will be unique. If the data, on the other hand was drawn from a uniform distribution of random numbers, then in all probability the clustering will vary with every attempt at clustering. It has been our experience that most data lies somewhere between these two extremes. Perhaps one of the most useful aspects of the clustering techniques is their ability to indicate which extreme a particular set of data lies nearest.

The paper by Mattson and Damman[27] is in the author's opinion an excellent example of combination of analytical and intuitive approaches. While the technique follows well-defined lines in terms of what is computed, nevertheless the concatination of these various well defined mathematical functions arrives at a technique that is considerably more useful than any one of these techniques applied separately. It is worth noting that this technique aims toward synthesizing efficient threshold networks. The technique is, however, useful for much more than that, in that it makes possible an examination by the researcher of the nature of his data in a relatively limited region of the space. Perhaps the most important aspect of thie technique is its use of direction in data, with the breaking up of the data into smaller subsets preventing heterogeneous subsets of the data from confusing the analysis.

Both Mattson and Damman[27] and Cooper and Cooper[13] are in effect searching for valleys between high density regions in the data. A technique which will be described below (Bledsoe[6]), is a third technique that has at its core the projection of data onto a line and the searching for low density spots in the distribution of the patterns along this line.

### Minimal Mode-Seeking Techniques

FIRSCHEIN AND FISCHLERS[16] Computes dot product of patterns with cluster centers (classes are partitioned into subclasses so that each member of a particular class is closer, in the sense of high correlation score, to the centroid of its one subclass than to the centroid of any other subclass; to classify an unknown vector, we determine the closest subclass centroid to the vector and assign the unknown vector to that class. Patterns are divided into three classes:

1. Patterns having highest dot product with centroid of its own subclass
2. — with a centroid of another subclass which is in the same overall class as the pattern
3. — with the centroid of another subclass which is in another class than the one given.

New subclasses are formed by having that pattern having the lowest dot product with its own subclass centroid chosen to form a new subclass centroid. The procedure is iterated until allowable error rates are reached or some arbitrarily chosen number of system iterations has been made. Computation goes as $N \times NROWS \times D \times ITER$. This technique and the following technique depend on the use of classification information in determining the subclasses. The technique may run into difficulty in cases where the pattern classes are overlapping unless some method is found for recognizing patterns that lie in the overlapping areas.

STEINBUCH AND PISKE:[42] Subclasses are formed if the distance between a pattern and a mode of its particular class is greater than a fixed threshold amount. The procedure is iterated until adequate separation is achieved, or other constraints are satisfied. Computation goes as $D \times N \times NROWS \times ITER$.

*Comments on Minimal Mode Seeking.* Firschein and Fischler's technique[16] appears to be quite useful

in cases in which pattern subclasses are linearly separable. However, from our experience at SRI it appears that modifications of the technique will be necessary when pattern classes are badly overlapped and intermixed in one another. That is, some method of storing patterns that are consistently causing difficulty should be used so that new subclasses would not be created for these patterns, if, for example, they lie right in the central part of another class.

A significant quote from this paper[16] (p. 138) is:

> Unlike previous procedures for subclass formation, this method does not require the specification of an arbitrary fixed distance as a criterion of membership in a subclass. Another advantage of the present technique is that it is not necessary to specify the required number of subclasses beforehand.

We consider these both to be significant aspects of this technique.

The paper by Steinbuch and Piske[42] we found difficult to read insofar as the recognition and clustering characteristics of the learning matrix are concerned. The article appears to be primarily a description of the learning matrix itself, rather than a description of the ways in which it operates, and its limitations. From our other experience it does appear that this technique would be useful, though some of the methods of normalization reduce its general applicability. It seems important to note that category information is required for this and for the Firschein and Fischler technique.

*Miscellaneous Technique*

BLOCK, KNIGHT AND ROSENBLATT:[7] The computational unit consists of two layers of summing networks followed by thresholds. The summing units on the first layer are in one-to-one correspondence with the summing units of the second layer and have a threshold of $\theta$. The weightings on the connections between the first and second layer are incremented if, when the threshold of the first layer unit is exceeded at time $T$, and the threshold of the second layer unit is exceeding at time $T + 1$. All connections are decremented each unit of time. The size of this decrement in the thresholds is a control parameter of the program. The computation goes as $D \times N \times NROW \times ITER$. Classification information is inferred from the sequence in which patterns are shown the machine. It is assumed that there is a high probability of continuous runs of patterns coming from the same class occur in contiguous runs.

BLEDSOE:[6] An arbitrary plane passing through the patterns is selected. Distances from this plane are computed for all of the patterns. The average distance of the pattern from this plane is maximized by a series of iterative adjustments of the plane. This procedure is tried for several different initial starting points. The plane having the maximum average starting distance is selected as the best plane. All patterns are projected onto this plane and a second plane in D-1 dimensions that maximizes distance from all the patterns is sought. (The procedure is constrained to pass the plane in such a way that all the subsets formed by the plane are of approximately equal size.) Computations go approximately as $N \times D \times ITER$. The results given in this paper were not really adequate to determine the effectiveness of the technique.

*Comments on Miscellaneous Techniques.* The method of Block, Knight and Rosenblatt is an ingenious method of taking advantage of high probability of runs. It is commented in that paper that the technique also has generalizing properties with regard to particular types of transformations. The paper referenced has a detailed analysis of the technique and its performance. The technique appears limited by the contiguous-run requirement. It is stated in the article that patterns need not necessarily look alike if they are continguous in sequence for them to be classified together. In other words, this is in some sense not a clustering technique, but it is a self-organizing technique that does not require an *explicit* external teacher.

The procedure Bledsoe follows in his paper is essentially one of trying to learn the "best" corridor (in the sense of a hyper-plane) to the learning set. A method appears adequate to do this but it is difficult from the description to tell how generally useful this technique would be. It might provide useful preprocessing, for, say, alphnumeric character recognition, in that it would in some sense "optimally" divide up the pattern set.

It seems worthwhile to note that a new calculation is required each time the dividing plane is adjusted into a new position. This new calculation determines whether the plane is appropriately adjusted or not. It is not apparent from the paper that a hill-climbing technique is being employed except in a probabilistic way.

## COMPOSITE CLUSTER-SEEKING ALGORITHM

The composite technique presented in this section is an attempt to improve the ISODATA algorithm by including the good points of other cluster-seeking techniques described. Though it seems unlikely that this one technique will be suitable for all types of data, we feel that it contains no obvious shortcomings.

### *Composite Technique (1965)*

Compute the average of patterns and the average distances of all patterns from this average. Set a threshold $\theta_D = k\ x$ (average distance of all patterns from this overall average) where $0 \leq k \leq 1$. All dimensions would be scaled at this time to cause each dimension to have a standard deviation of 1 for all patterns taken together. Until 3rd iteration minimum allowable cluster size is 1.

START: Compute dimensions of all patterns from all existing cluster points. If minimum distance (measured using Euclidean distance) from a pattern to any existing cluster point exceeds $\theta_D$, then create a new cluster point at the location of that pattern. (This should rapidly locate small isolated clusters.) Evaluate the goodness of the cluster, perhaps using a criterion similar to that used in the clumping techniques.

SPLIT: If clustering is not satisfactory, then "SPLIT" either those clusters having unsatisfactory cluster properties or those clusters having maximum standard deviations greater than a threshold (if the cluster also is of sufficient size and has sufficiently large average pattern distance from the cluster center). Return to (START) and repeat process of computing pattern distances from the new cluster centers.

LUMP: Combine (LUMP) cluster points that are closer than a given threshold. Return to (START) and repeat process using (SPLIT). At the end of this iteration all cluster points having fewer than a given number of patterns associated with them would be removed from the list of patterns (this should remove all wild shots). These patterns would be printed out.

Recompute new "standard deviations" for each dimension for reduced pattern set and rescale all patterns so that the normalized standard deviation in each dimension equals 1. These standard deviations could be the average using all patterns and computed about an overall minimum, or they could be the average standard deviation of patterns in each cluster about its own cluster center. This standard deviation might be further modified by taking into account the pairwise distances between cluster centers at each distance.

Lump and split until a criterion of adequacy of clustering is satisfied. The relative increase in measure of clustering from iteration to iteration could be used to determine when lumping and splitting should stop.

Computational difficulty is proportional to $D \times NROWS \times N \times ITER$. *Note*: If all multiplications with elements in each cluster center can be done in parallel (optically) then this reduces to $N \times ITER$.

Parameters that control the program are related to:

1. minimum allowable number of patterns in a cluster,
2. the allowable maximum standard deviation (or spread) of a cluster,
3. the minimum distances between two cluster points,
4. the fraction of the standard deviation used in the first iteration to control the initial creation of clusters and the discarding of wild shots,
5. the iteration-to-iteration difference required to imply convergence of clustering.

This technique could be generalized and probably implemented optically.

*Comments on Composite Technique.* It seems unlikely that any one algorithm will be ideal for all types of data and for all situations. It does seem possible, however, to describe types of data, and to determine the type of cluster-seeking algorithm that seems most useful for handling each type of data. A need exists for a preliminary, exploratory technique to tell which of the more specialized types of techniques to apply. We need to be able to broadly categorize types of data so as to decide how to process them further in more detail.

This suggested technique is an attempt to include in ISODA the best (noncontradictory) aspects of other techniques. The following are improvements over the original ISODATA program:

1. The modification relating to the initial cluster-creating procedure should cause more rapid convergence.
2. Discarding and identification of wild shots from pattern sets should improve efficiency considerably.
3. The use of good clustering criteria would allow realistic termination of the iterative procedure.
4. Scaling of dimensions so that some (yet to be determined) criterion is satisfied would remove from clustering some of the arbitrariness arising from the choice of the unit of measurement used in the various dimensions.

## CONCLUSIONS

The main point we wish to make is that cluster-seeking techniques exist that can be used to organize data from the social sciences and other disciplines in a way that allows examination of the details, i.e., the individuals, in the data. Using these techniques a single data point can be related to an adequate description of the rest of the data. Cluster-seeking techniques can be effectively utilized by persons having relatively little formal mathematical and staistical training. These techniques have the further advantage that the effect of changing scale or measurements can be directly and easily related to its effect on the data. In other words, the relative arbitrariness of calling certain patterns "similar" becomes more apparent.

An important consideration in determining the value of the clustering obtained is the use to which the clustering is to be put. That is, is the clustering obtained from the data to be used to determine the significance of the data in a statistical sense, or is it to be used primarily as a descriptive organizing of the data for the researcher? It is important in criticizing or commenting on clustering techniques to keep this distinction clear. Some clustering criteria are not adequate to determine significance of the data; nevertheless, the clustering may provide a completely adequate description of the data that is very suggestive of new experiments to be performed or new interpretations of the data that then can be tested by more rigorous methods.

With respect to the specific cluster-seeking techniques, we regard the "clustering" techniques as providing the most efficient and easily interpreted clustering, except in taxonomic problems where clumping techniques (modified to be more efficient) appear best. The addition of more adequate cluster criteria to the "clustering" techniques seems a natural next step.

Much more work needs to be done on convergence of these techniques and on methods for interpreting and examining the resulting clusters.

It appears important to us not to rely entirely on some function of the data such as the covariance matrix. Rather it seems that for data analyses, *particularly where the data base is small,* working directly with the patterns themselves is required. Naturally the patterns must be organized into some comprehensible form. Clustering the data is one way to organize them.

Finally, the distinction to us, between classical statistics and clustering with regard to data analysis is the distinction between the point of view (statistical) that immediately generalizes from the specific (the patterns) to the general (the estimated distribution), and the point of view (clustering) that remains at the specific (the patterns) and has the capability of analyzing the individual patterns for local as well as global relationships.

### *Speculation Regarding the Effect on the World Around Us*

We feel that computer-oriented techniques that can quickly organize data in a way that allows rapid analysis of the data will profoundly affect experimental science. Starting with existing clustering techniques and using proposed peripheral computer programs, it will be possible for the experimental scientist to see on a display the data he is gathering *as he gathers it.* The potential value in such rapid feedback seems enormous when we think how rapidly we forget all of the details of an experimental situation. We at SRI consider ourselves to be working toward this eventuality which may have considerable effect on the world around us.

## ACKNOWLEDGMENTS

I would like to acknowledge the support of Rome Air Development Center under Contract No. AF 30(602)-3448 and the continuing encouragement and guidance of Dean F. Babcock, my laboratory manager. I would like to express my considerable gratitude to David J. Hall of SRI. Conversations

with him have greatly aided in the formulations in this paper. My thanks also go to Wanda Fedurek and Jane Bond for their speed and accuracy in typing this report.

## ADDENDUM

Since this paper was written, additional references have come to our attention. These papers are listed in the Supplementary Bibliography.

The author would *greatly* appreciate receiving copies of past and future papers or references to past papers dealing with cluster-seeking methods and their use that are not included in the References or the Supplementary Bibliography.

## REFERENCES

1. C. T. Abraham, "Evaluation of Clusters on the Basis of Random Graph Theory," unpublished report, IBM, Yorktown Heights, N.Y.

2. ———,"A Note on a Measure of Similarity Used in the DICO Experiment", Appendix I, Quarterly Report 3, vol. 1, Contract AF 19(626)-10.

3. D. C. Allais, "The Selection of Measurements for Prediction," Stanford Elec. Lab., System Theory Div. Report, TR6103-9 (AD 456 770), Stanford, Calif. (Nov. 1964).

4. G. H. Ball and D. J. Hall, "ISODATA, A Novel Method of Data Analysis and Pattern Classification," Stanford Research Institute, Menlo Park, Calif. (Apr. 1965).

5. P. Baxendale, "An Empirical Model for Computer Indexing," *Machine Indexings Progress and Problems,* American University, Washington, D.C., Feb. 13-17, 1961, p. 267.

6. W. W. Bledsoe, "A Corridor-Projection Method for Determining Orthogonal Hyperplanes for Pattern Recognition," unpublished report, Panoramic Research Cor., Palo Alto, Calif. (1963).

7. H. D. Block, B. W. Knight and F. Rosenblatt, "The Perceptron: A Model for Brain Functioning, II," *Reviews of Modern Physics,* vol. 34, no. 1, pp. 135-142 (Jan. 1962).

8. R. E. Bonner, "On Some Clustering Techniques," *IBM Journal of Res. and Dev.,* Jan. 1964.

9. E. J. Brennan, "An Analysis of the Adaptive Filter," General Electric Elec. Lab. Tech. Information Series Report R61 ELS-20, Syracuse, N.Y. (1961).

10. R. Cattell, "A Note on Correlation Clusters and Cluster Search Methods," *Psychometrika,* vol. 9, no. 3 (Sept. 1944).

11. D. B. Cooper, "Nonsupervised Adaptive Signal Detection and Pattern Recognition," Raytheon Report, October 22, 1963.

12. ——— and P. W. Cooper, "Adaptive Pattern Recognition and Signal Detection without Supervision," *IEEE International Convention Record,* pt. 1, 1964, pp. 246-256.

13. ———and———, "Nonsupervised Adaptive Signal Detection and Pattern Recognition," *Information and Control,* vol. 7, No. 3 (Sept. 1964).

14. R. F. Daly, "Adaptive Binary Detection," Stanford Elec. Lab. Tech. Report No. 2003-2, Stanford, Calif. (June 26, 1961).

15. ———, "The Adaptive Binary-Detection Problem on the Real Line," Stanford Elec. Lab. Report SEL-62-030, Stanford, Calif. (Feb. 1962).

16. O. Firschein and M. Fischler, "Automatic Subclass Determination for Pattern Recognition Applications," *Trans. PGEC,* EC-12, no. 2 (Apr. 1963).

17. J. J. Fortier and H. Solomon, "Clustering Procedures," Tech. Report 7, Dept. of Statistics, Stanford University (Mar. 20, 1964).

18. S. C. Fralick, "The Synthesis of Machines Which Learn Without a Teacher," Tech. Report No. 6103-8, Stanford University (Apr. 1964).

19. E. M. Glaser, "Signal Detection by Adaptive Filters," *IRE Trans. On Info. Theory,* vol. IT-7, no. 2 (Apr. 1961).

20. L. Hyvarinen, "Classification of Qualitative Data," *British Info. Theory J.,* 1962 pp. 83-89.

21. C. V. Jakowatz, R. L. Shuey and G. M. White, "Adaptive Waveform Recognition," *Information Theory,* C. Cherry, ed., Butterworths, Washington, D. C., 1961.

22. G. Kaskey et al, "Cluster Formation and Diagnostic Significance in Psychiatric Symptom Evaluation.", *Proc. Fall Jt. Computer Conf.,* 1962, p. 285.

23. H. Kazmierczak and K. Steinbuch, "Adaptive Systems in Pattern Recognition," *IEEE Trans. on Electronic Computers,* vol. EC-12, no. 6 (Dec. 1963).

24. M. Kochen, "Techniques for Information Retrieval Research: State of the Art," presented at IBM World Trade Corporation Information Retrieval Symposium at Blaricum, Holland, Nov. 1962;

to be published in the proceedings of the symposium.

25. ———and E. Wong, "Concerning the Possibility of a Cooperative Information Exchange," *IBM Journal of Research and Development,* vol. 6, No. 2, pp. 270-271 (Apr. 1962).

26. H. P. Luhn, "Auto-Encoding of Documents for Information Retrieval System," *Modern Trends in Documentation,* M. Boaz, ed., Pergamon Press, New York 1959, pp. 45-58.

27. R. L. Mattson and J. E. Damman, "A Technique for Determining and Coding Subclasses in Pattern Recognition Problems," submitted for Publication to *IBM J. of Res. and Dev.,* Mar. 1965.

28. C. D. Michener and R. R. Sokal, "A Quantitative Approach to a Problem in Classification," *Evolution,* vol. 11, pp. 130-162 (June 1957).

29. R. M. Needham, "The Theory of Clumps, II," Report M. L. 139, Cambridge Language Research Unit, Cambridge, Eng. (Mar. 1961).

30. J. Nunnally, "The Analysis of Profile Data," *Psychological Bulletin,* vol. 59, no. 4, pp 311-319, 1962.

31. M. Okajima, L. Stark, G. Whipple and S. Yasui, "Computer Pattern Recognition Techniques: Some Results with Real Electrocardiographic Data," *IEEE Trans. on Bio-Medical Electronics,* vol. BME-10, no. 3 (July 1963).

32. A. F. Parker-Rhodes, "Contributions to the Theory of Clumps," I.M.L. 138, Cambridge Language Research Unit, Cambridge, England (Mar. 1961).

33. E. A. Patrick and J. C. Hancock, "The Non-Supervised Learning of Probability Spaces and Recognition of Patterns," Tech. Report, Purdue Univ., Lafayette, Ind. (1965).

34. D. J. Rogers and T. T. Tanimoto, "A Computer Program for Classifying Plants," *Science,* vol. 132, Oct. 21, 1960.

35. W. L. Sawrey, L. Keller and J. J. Conger, "An Objective Method of Grouping Profiles by Distance Functions and its Relation to Factor Analysis," *Educational and Psychological Measurement,* vol. 20, no. 4 (1960).

36. G. S. Sebestyen, "Pattern Recognition by an Adaptive Process of Sample Set Construction," *IRE Trans. on Info. Theory,* vol IT-8, Sept. 1962.

37. ———and J. Edie, "Pattern Recognition Research," Air Force Cambridge Res. Lab. Report 64-821 (AD 608 692), Bedford, Mass (June 14, 1964).

38. J. W. Smith, "The Analysis of Multiple Signal Data," *IEEE Trans. On Information Theory,* vol. IT-10, no. 3 (July 1964).

39. R. R. Sokal and C. D. Michener, "A Statistical Method for Evaluating Systematic Relationships," *University of Kansas Science Bulletin,* Mar. 20, 1958.

40. ———and P. H. A. Sneath, *Principles of Numerical Taxonomy,* W. H. Freeman and Co. San Francisco, 1963.

41. J. J. Spilker, Jr., D. D. Luby and R. D. Lawhorn, "Progress Report—Adaptive Binary Waveform Detection," Tech. Report 75, Communication Sciences Department, Philco Corp., Palo Alto, Calif. (Dec. 1963).

42. K. Steinbuch and U. A. W. Piske, "Learning Matrices and Their Applications," *IEEE Trans. on Electronic Computers,* vol. EC-12, no. 6 (Dec. 1963).

43. H. E. Stiles, "The Association Factor in Information Retrieval", *Communications of the ACM,* vol. 8, no. 2, pp. 271-279 (Apr. 1961).

44. R. D. Turner, "First-Order Experimental Concept Formation," *Biological Prototypes and Synthetic Systems,* E. E. Bernard and M. Kare, eds., Bionics Symposium 2, Ithaca, N. Y., Plenum Co., 1961.

## SUPPLEMENTARY BIBLIOGRAPHY

B. M. Bass, "Iterative Inverse Factor Analysis—A Rapid Method for Clustering Persons," *Psychometrika,* vol. 22, no. 1, pp. 105-107 (Mar. 1957).

Eigenvalue I—Uses iterative method to factor analyze test scores in order to cluster people.

W. D. Fisher, "On Grouping for Maximum Homogeneity," *Journal of American Stat. Assn.,* vol. $53^2$, pp. 789-798 (Dec. 1958).

Clumping—On real line examines all possible partitions and selects that partition minimizing the weighted square distance from the cluster average point.

E. W. Forgy, "Detecting 'Natural' Clusters of Individuals," *Western Psychological Association Meetings,* Apr. 19, 1963, Santa Monica, Calif. (Can be obtained from author at Center for Health Sciences, U.C.L.A., Los Angeles, Calif.)

Miscellaneous.

E. W. Forgy, "Evaluation of Several Methods for Detecting Sample Mixtures from Different N-Dimensional Populations," *American Psychology As-*

*sociation Meetings,* Los Angeles, Calif., Sept. 9, 1964. (Available from author at Center for Health Sciences, U.C.L.A., Los Angeles, Calif.)

Gives results of five methods for clustering on several artificial problems. The five methods are:

1. Using the frequency histogram of interpoint distances.
2. Sokal and Sneath (1957).
3. Ward (1963)—See below for reference.
4. Forgy (1963).
5. Factor analysis—Q sort.

J. A. Gengerelli, "A Method for Detecting Subgroups in a Population and Specifying their Membership," *Journal of Psychology,* vol. 55, pp. 457-468 (1963).

Miscellaneous—Analysis of distribution of pairwise distances between patterns.

L. I. McQuitty, "Typal Analysis," *Educational Psychological Meas.,* vol. 21, pp. 677-696 (1961).

Clumping.

P. Medgyessy, *Decompositions of Superpositions of Distribution Functions,* Publishing House of the Hungarian Academy of Sciences, Budapest, 1961.

Discussed how a "sum distribution" that is the sum of a known number of elementary distributions of known form can be used to find the parameters of the elementary distributions. Though not directly providing a clustering method, it *may* provide some analytical insight into the analysis of clustering methods.

R. C. Tryon, *Psychometrika,* vol. 22, no. 3, pp. 241-260 (Sept. 1957).

Uses clustering to indicate factor analytic communalities. Gives references to Tyron's earlier work on clustering.

—,"Domain Sampling Formulation of Cluster and Factor Analysis," *Psychometrika,* vol. 24, no. 2, pp. 113-135 (June 1959).

Eigenvalue I—Discusses "Domain Sampling" as a viewpoint underlying his custer analysis *and* factor analysis. Tends to look at cluster analysis as related to correlations between the measurements found over the entire data base.

J. H. Ward., Jr., "Hierarchical Grouping to Optimize an Objective Function," *Journal of American Statistical Association,* vol. 58, no. 301 (Mar. 1963).

Clumping-combines those two patterns that maximally increase an objective function. Primarily related to Taxonomic structures.

—and Marion E. Hook, "Application of an Heirarchical Grouping Procedure to a Problem of Grouping Profiles," *Educational and Psychological Measurement,* vol. 23, no. 1 (1963).

See Ward above.

J. H. Wolfe, "A Computer Program for the Maximum Likelihood Analysis of Types," Tech. Bull., 65-15, U. S. Naval Personnel Research Activitiy, San Diego, Calif. 92152 (May 1965).

Miscellaneous—Using an initial rough clustering procedure, the clustering is refined using maximum likelihood methods and an assumed underlying mixture of multivariate normal distributions.

G. Young, "Factor Anaylsis and the Index of Clustering," *Psychometrika,* vol. 4, no. 3 (Sept. 1939).

Proposes the dispersion of the eigenvalues of the covariance matrix as an "index of clustering."

# NONLINEAR REGRESSION MODELS IN BIOLOGY

Joseph A. Steinborn
*Department of Preventive Medicine*
*University of California, Los Angeles*

## MATHEMATICAL MODELS IN BIOLOGY

Biological processes can be considered in the abstract as a response to a set of input quantities where input and output are measured in analogy with the real numbers.

Many processes are not really understood unless they can be described with numbers. Basically, a mathematical-biological model is an isomorphism between a biological process and the real number system. That is, a model is a function $f$ defined on a set of (possibly) independent real variables $x$ and having values corresponding to what is expected to be the value of the process. Nature, however, has her own function $Y$, which perhaps cannot be known, and whose domain hopefully corresponds to $x$, so that we may write:

$$Y(x) = f(x) + \mathbf{e}(x)$$

where $\mathbf{e}(x)$ is error term consisting of observational error and/or failure of "isomorphism."

Mathematical models are in a sense a fiction and we do expect the error due to a failure of the isomorphism to be small in comparison to the observation as well as the error of observation.

The mathematical model is a formalized quantitative working hypothesis. The conventional operation of scientific method requires four steps.

1. The formulation of the working hypothesis from experimental data.
2. The experiment to test the hypothesis.
3. Numerical fitting of the data.
4. Examination and possible reformulation of the hypothesis to fit observations.

The purpose of model building is twofold: to gain insight into a phenomenon and to be able to predict a response to a given set of input. These notions can be extended and idealized into a procedure as in Fig. 1, from which it can be seen that the construction of mathematical models is a synthesis of several disciplines.

As the frontier of knowledge of biology is advanced, demands are made for the construction and solution of more critical and complex models. The success is dependent upon the ability to correctly select one model from several *a priori* acceptable ones, none of which are equivalent.

There are three problems inherent in nature that affect mathematical models. First, nature in general is nonlinear. Second, biological systems cannot be measured without frequently large error. Third, biological phenomena generally have a stochastic character. All affect the formulation, fitting, and verification of the model in different ways. At times, the effect is cumulative and may give rise to hidden but gross errors, wherein a seemingly good

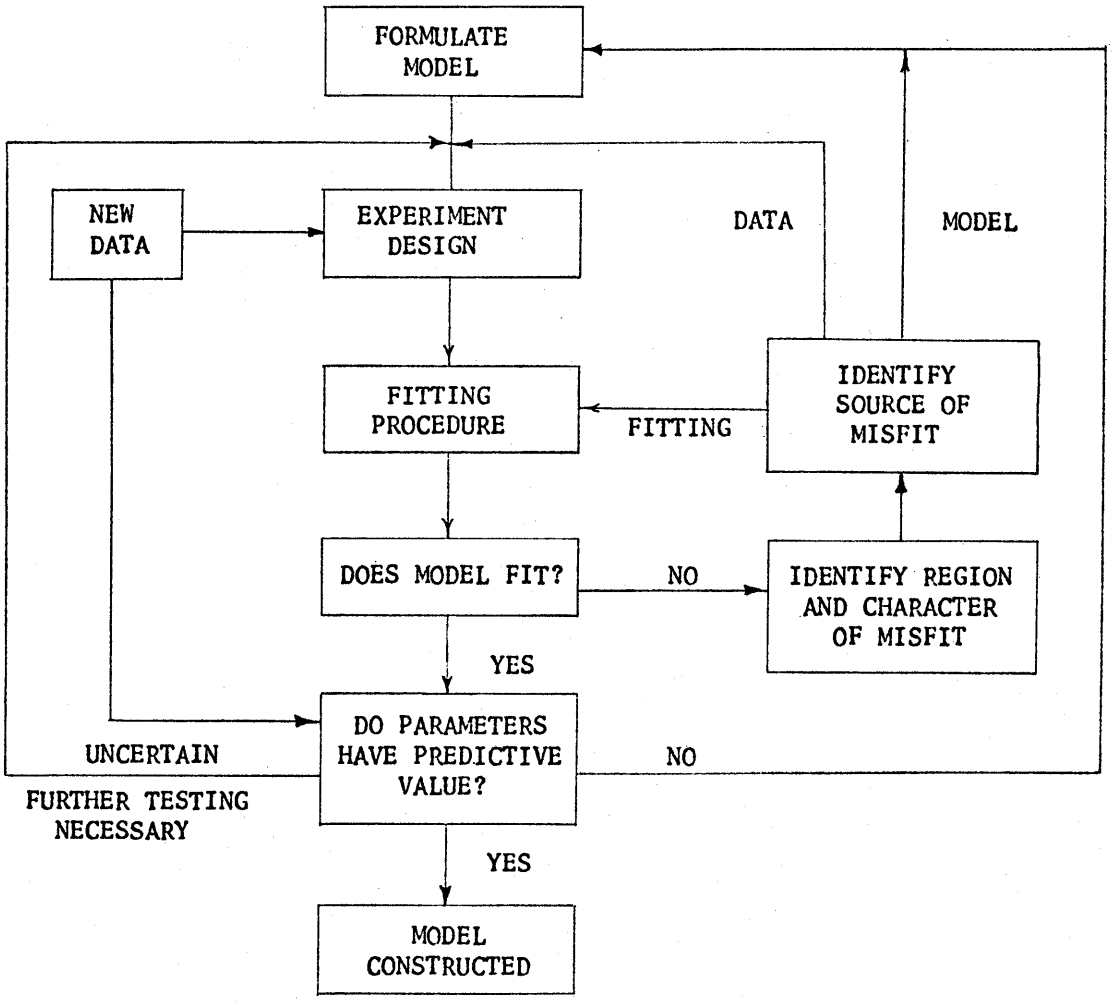



Figure 1. Morphology of model building.

fit may be an artifact of the number of parameters alone.

Once a mathematical model is constructed, it is often taken for granted that there is no information loss in the fitting procedure or any other computational aspect of the process. Unfortunately numerical problems present a difficulty for every aspect of the procedure and should be considered by the biologist even in the choice of the experimental design.

The problem from the point of view of information processing is to identify sources of numerical difficulty throughout the process and in particular to numerically fit the data to the model as faithfully as possible with the finite digit capacity of a computer. The central problem from our viewpoint is how reliably we can identify the numerical solution to the model fitting problem.

## DEFINITION OF NONLINEAR PROBLEM

Let $F$ be a function defined on real vector spaces $A$ and $X$ and having values in $R$. Then $F$ is linear in $A$ provided that

$$F(\lambda a,x) = \lambda F(a,x)$$
$$F(a+b,x) = F(a,x) + F(b,x)$$

where $a, b \in A$ (parameter),
$\lambda \in R$ (real numbers), and
$x \in X$ (independent variable).

A similar definition applies to linearity in $X$.

We can consider a mathematical model as being a path $(x, F(a,x)): x \to R^2$ in $R \times R$ where the fitting procedure is a technique of finding the point $a \in A$ that satisfies some criterion of best fit such as mini-

mum sums of square deviation of the model from the data.

The model function $F$ can be linear or nonlinear in either or both of $A$ and $X$. The discussion concerns itself with the problem of fitting a model to a set of data when the function is not necessarily linear in the parameters.

However highly nonlinear $F$ may be in $X$, if $F$ is linear in $A$, then the computational aspects of the fitting procedure are relatively simple and are usually done in one step. However when $F$ is nonlinear in $A$, then the computational problem is more complicated.

## METHOD OF FITTING NONLINEAR MODELS

For simple nonlinear models, there frequently exists a choice of two basic methods of solution of the fitting problem.

1. Linearization by transformation.
2. Successive approximation function $S$ (gradient and nongradient) of the form $a^{\nu+1} = S(a^{\nu})$ for which we seek a fixed point $a = S(a)$.

Linearization has the decided advantage of not requiring starting values of the parameters although it usually requires estimation of a weighting function to correct for changes in the error structure induced by the transformation. Iterative methods require starting values and are in fact a function of the starting values.

The methods will be compared by numerical experiment. The Gauss-Newton method will now be discussed to illustrate certain of the computational difficulties it presents.

The numerical problem in nonlinear regression is simply to obtain the correct parameter values. Unfortunately, there exists factors that cause failure of this, which include:

1. Large errors in data or errors not having a uniform random deviation from the model.
2. Insufficiently good starting values for the iterative technique.
3. Overparametrized model.
4. Structure of the numerical technique which is used to obtain the fit
   (a) parameterization,
   (b) choice of experimental range of data,
   (c) numerical behavior in the various computational phases of the regression procedure.

Each Gauss-Newton iteration is a least squares fit of the function $G = Y\text{-}F$ as a function of the differential

$$\left(\frac{\partial F}{\partial \alpha_1}, \frac{\partial F}{\partial \alpha_2}, \ldots, \frac{\partial F}{\partial \alpha_n}\right)$$

whose solution is a vector

$$\tau = (\triangle\alpha_1, \triangle\alpha_2, \ldots, \triangle\alpha_n)$$

The nonlinear fitting problem is reduced to a sequence of linear fittings in $n$ variables. We list the following structural problems that disturb the process numerically:

1. The differential of $F$ may be highly nonlinear in some of the components. Hence the differential represents the function for a small neighborhood at each $x$ and the differential may vary in its ability to approximate $G$ to a considerable extent as a function of $x$.
2. The intra-variable variation of the differential as a function of $x$ as well as $a$ may be too great for the digit capacity of the machine and may therefore bias the fit.
3. The inter-variable variation of the components of the differential as a function of $x$ may also be too large.
4. The components of the differential may be numerically dependent over the range of $x$.

Various methods can be applied to reduce the effect of each of these difficulties when using the Gauss-Newton method:

1. Modify the magnitude of the Gauss-Newton vector by some sort of search pattern to minimize residual sum of squares.
2. Use weighted regression.
3. Reparameterize the function or rescale the independent variable.
4. Apply stepwise regression techniques (although, the information used by the stepwise procedure is again based on linearization and the wrong components may be left out of the regression because of this).

## MICHAELIS-MENTEN KINETICS

The following is a pair of chemical equations for the formation of a produce P from a substrate S via an enzyme-substrate intermediary complex ES:

$$\mathrm{E} + \mathrm{S} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} \mathrm{ES}$$

$$\mathrm{ES} \xrightarrow{k_3} \mathrm{E} + \mathrm{P}$$

where $k_i$ are rate constants. Using brackets to indicate concentrations for the reactants, and letting the initial concentration of the enzyme be denoted as $[\mathrm{E}]_0$, the following differential equations describe the reaction kinetics:

$$\frac{d[\mathrm{ES}]}{dt} = k_1([\mathrm{E}]_0 - [\mathrm{ES}])[\mathrm{S}] - (k_2 + k_3)[\mathrm{ES}]$$

$$\frac{d[\mathrm{S}]}{dt} = -k_1[\mathrm{E}][\mathrm{S}] + k_2[\mathrm{ES}]$$

The reaction has a steady state when $[\mathrm{S}]_0$ is large and [ES] maintains a maximum value $[\mathrm{ES}]_m$ for an extended period of time. For $[\mathrm{ES}] = [\mathrm{ES}]_m$ we have

$$\left.\frac{d[\mathrm{ES}]}{dt}\right|_{[\mathrm{ES}]_m} = 0$$

Of particular interest is the rate $V$ of reformation of the product P

$$V = \frac{-d[\mathrm{S}]}{dt} = k_1[\mathrm{E}][\mathrm{S}] - k_2[\mathrm{ES}]$$

In brief we have a system of differential equations of the form

$$\frac{dY}{dt} = (as + b)\,Y + ex$$

$$\frac{dx}{dt} = (ax + d)\,Y - ex$$

where $a, b, d, e$ are constants.

An enzyme is said to satisfy the Michaelis-Menten kinetics when steady state approximation to the system is satisfied:

$$\frac{dY}{dt} = 0$$

$$V = \frac{-dx}{dt} = \frac{V_m x}{K + x}$$

where $K = (k_2 + k_3)/k_1$ is the Michaelis constant.

It is often important to know if an enzyme obeys th Michaelis-Menten kinetics or whether a more complex reaction is occurring. The model can be extended to more complex ones where such things as enzyme inhibition occur which give rise to models that are rational in bilinear forms:

$$V = \frac{\Sigma\, a_i x_i}{\Sigma\, b_J y_J}$$

The usual method employed by researchers to fit the above model is:

1. Linearize the function by transformation.
2. Plot the transformed data.
3. Approximate the parameters from the graph.

Graphical solutions are not statistical statements, since they are obtained qualitatively, are frequently biased and they do not provide estimates of variances of the parameters. Moreover, recent studies have been made which point out the pitfalls attending choosing a graphical method such as Lineweaver-Burke plot.

The model equation:

$$V = \frac{V_m x}{K + x} + e = f(x) + e$$

where $x \geq 0$, and
$e \in N(f_{(x)}, \sigma_x^2)$,

will be fitted by the following four methods:

I. Linearization by transformation

$$\frac{1}{V} = \frac{K}{V_m} \cdot \frac{1}{x} + \frac{1}{V_m} - \frac{e}{f^2}\left(1 - \frac{e}{f} + \cdots\right)$$

Weighted linear regression is used (a) once (b) iteratively

II. Successive approximation

$$V = V_m + K\left(\frac{-f(x)}{x}\right) + e$$

III. Linearization, iteration of weighting function

$$V = \frac{V_m}{K} x + \frac{1}{K}(-V_x) + \left(\frac{K + x}{K}\right) e$$

IV. Gauss-Newton iteration

$$V - f(x; V_m^0, K^0) = (V_m - V_m^0)\left(\frac{x}{K^0 + x}\right) + (K - K^0)\left(\frac{-V_m^0 x}{(K^0 + x)^2}\right)$$

The Michaelis-Menten model will be studied here from the computational point of view since it is the simplest of a whole class of nonlinear regression

problems, and it has the advantage of the fact that there are these four methods of obtaining the solution which can be compared readily. From this, one should be able to gain insight as to the reliability of the various methods for more complex problems.

Naturally, such considerations as starting values for iterative methods present a problem. There are not very many models that can be so conveniently linearized as in I. Method III seems quite tempting since it is applicable to rational functions and does not require starting values. In the Gauss-Newton method convergence is guaranteed provided that we are in a sufficiently close neighborhood of the solution and we do not have the numerical difficulties mentioned before.

These methods will be compared using the following criteria:

1. Ability to predict the parameters.
2. Predictive power for the estimates of the variances of the parameters.
3. Stability of iteration.
4. Effect of range of independent variable on results.
5. Effect of magnitude of error on results.
6. Minimize the sums of squares.

Note that the solutions for $K$ and $V$ are equivalent for methods II and IV, theoretically. It remains to be seen if they are equally robust computationally.

## SIMULATION OF EXPERIMENTS

Three overlapping ranges of independent variables were used.

$$R_1 = \{ x \mid 0.2 \cdot 10^{-3} \leq x \leq 2.0 \cdot 10^{-3} \}$$
$$R_2 = \{ x \mid 0.08 \cdot 10^{-3} \leq x \leq 0.8 \cdot 10^{-3} \}$$
$$R_3 = \{ x \mid 1.5 \cdot 10^{-3} \leq x \leq 15.0 \cdot 10^{-3} \}$$

Four error levels are used for each range $R_j$ of $x$

$$E_i: \sigma = p_i f(x) \qquad x \in R_J$$
$$P_i = 0.05,\ 0.1,\ 0.15,\ 0.2$$

Note that for each experiment the variance is uniform and that there are 12 different experiments.

For each range, 100 sets of data were generated with random normal deviates at each of 4 error levels. Each of the fitting procedures attempted to determine the parameters that generated the data, which in all cases were K = 0.003, $V_m = 1$. For each method, the following information was collected:

$$(V_m,\ \sigma_V,\ K,\ \sigma_K,\ \Sigma(V-f)^2)$$

After 100 "experiments," the following statistics were computed for each of the methods:

$$(V_m,\ \underset{m}{\sigma_V},\ K,\ \sigma_K,\ \Sigma(V-f)^2)$$
$$(s_V,\ s_{\sigma V},\ s_K,\ s_{\sigma K},\ s_\Sigma)$$

For the Gauss-Newton methods, two choices of starting values were made and compared, the one being the results of weighted linear regression, the other being the true values of the parameters.

## CONCLUSIONS

Preliminary studies seem to indicate the following:

1. Methods II and III were least reliable. In the first place they always required many more iterations than method IV. Secondly, when method II converged, ($R_4$) the estimates of the standard deviation were off by a factor of two. Method II gives consistently better results than III. Method II behaved at its best for $R_3$, yielding estimates practically identical with IV but only after many iterations. For $R_1$ and $R_2$ the iterations were either prohibitively slow or diverged. There is no compelling reason to believe that the situation would improve for higher dimensionality. Moreover, the sums of squares look reasonable when the estimates are far off and the iteration is not moving.

2. Methods I and IV have the most consistent agreement between sampling variance of estimates and estimated variance of parameters for error levels $E_1$, $E_2$ on all three ranges of $x$ and for $E_3$ on $R_3$.

3. When the weights are iterated in an attempt to improve the results, the residual sum of squares (unweighted) is reduced considerably, but the estimates of the variances of the parameters are unreasonable except for low error range. Also, the computed estimates of the parameters themselves are very poor except for the error level $E_1$. The situation is again slightly improved when the range $R_3$ of $x$ is used. The addition of another term on the weighting function slightly improves the situation numerically.

4. The Gauss-Newton iteration definitely improves the situation using starting values from I with a few unsettling execeptions. Method I tends to underestimate the parameters slightly while IV seems to overestimate the parameters although the residual sums of squares are less (see Table 1). For $R_1$ the Gauss-Newton iteration continually decreases the sum

of square deviation while diverging even for small errors. This is possibly the result of a few outliers since using a search technique to modify the Gauss-Newton iteration and starting values equal to the expected value of the parameters does not seem to prevent divergence. Note that method IV diverges at those points where the correspondence between $S_k$ and $\alpha_k$ in method I deteriorates. Finally, the divergence on the average noted in Table 1 is due to a relatively few experiments which have outliers. The detection of such outliers and their removal is outside the scope of this present paper.

Table 1. Comparison of Methods I and IV for Estim ating $K$.

| | | Method I | | | | | Method IV | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $x\ 10^{-1}$ | | | | | $x\ 10^{-2}$ | | | | |
| | | $K$ | $S_k$ | $\sigma_k$ | $S_\sigma$ | *RMS* | $K$ | $S_k$ | $\sigma_k$ | $S_\sigma$ | *RMS* |
| $R_1$ | $E_1$ | .289 | .131 | .114 | .18 | .0056 | .337 | .145 | .151 | .20 | .0051 |
| | $E_2$ | .253 | .286 | .283 | .82 | .0012 | Diverges on average | | | | .0011 |
| | $E_3$ | .229 | 1.22 | 2.4 | 10. | .018 | | | | | .016 |
| $R_2$ | $E_1$ | .281 | .059 | .054 | .02 | .011 | .303 | .065 | .063 | .03 | .011 |
| | $E_2$ | .253 | .102 | .088 | .05 | .024 | .324 | .117 | .139 | .08 | .022 |
| | $E_3$ | .268 | .351 | .190 | .6 | .038 | Diverges on average | | | | .033 |
| | $E_4$ | .186 | .353 | .244 | .7 | .052 | | | | | .044 |
| $R_3$ | $E_1$ | .293 | .043 | .041 | .01 | .030 | .300 | .043 | .042 | .01 | .030 |
| | $E_2$ | .301 | .086 | .081 | .03 | .062 | .328 | .094 | .091 | .03 | .059 |
| | $E_3$ | .254 | .121 | .105 | .06 | .098 | .308 | .133 | .131 | .07 | .088 |
| | $E_4$ | .263 | .243 | .146 | .14 | .14 | .333 | .268 | .196 | .20 | .12 |

One would conclude that the Gauss-Newton method when it converges is the most faithful in recovering the information in the system hidden by noise; and that a high price is paid for the inability to obtain sufficiently good estimates to initiate the iteration. It might be hoped that method III would yield sufficiently good starting values for the G-N method in the case of a rational polynomial of higher degree when this information is not known.

## REFERENCES

1. J. E. Dowd and D. S. Riggs, *Journal of Biological Chemistry,* vol. 240, no. 2, p. 863.

2. E. Ackerman, *Biophysical Science,* Prentice-Hall, 1962.

3. H. O. Hartley, *Technometrics,* vol. 3, no. 2, p. 269.

## ACKNOWLEDGMENTS

The computations were performed at the Health Sciences Computing Facility, University of California at Los Angeles. The author wishes to express thanks to Dr. Donald Jenden at the Department of Pharmacology for his suggestions and to Sandra James for her assistance.

# COMPUTER CORRELATION ANALYSIS OF INTRACELLULAR NEURONAL RESPONSES*

F. F. Hiltz
*Applied Physics Laboratory*
*The Johns Hopkins University*
*Silver Spring, Maryland*

## INTRODUCTION

The contribution of large general-purpose digital computers to the bio-medical discipline has been considerable. However, as great as the contribution has been, the full potential has yet to be realized.

Increasingly less is the bio-medical investigator confined to spend large segments of his efforts reducing raw data manually. Increased instrumentation capabilities and the utilization of both analog and digital computers are being profitably employed to semiautomatically or automatically reduce large portions of raw data.

These procedures have their Pandora's boxes, however. Not infrequently, the large amount of raw data confronting an experimenter has been replaced by an equally large, or larger, amount of reduced data; leaving the experimenter with the dilemma that he knows less about a good deal more.

In evolving a digital computer program[1,2] to recognize intracellular neuronal responses in a time scale of approximately six man-months to one machine-hour, the Applied Physics Laboratory, The Johns Hopkins University was instrumental in replacing the first stack of data with the second for investigators of neuronal responses. Having accomplished this replacement, the problem arose of substituting a third amount of data for the second; the third being an informative reduction, through various forms of analyses, of the second.

This paper is a description of a digital computer program which will receive the data (of the second form) from the neuronal response recognition program, analyze the data in terms of various statistical procedures, and present the results in a condensed, and hopefully profitable, form.

The questions asked by the analysis program are simple and basic. Whether the answers will be adequate remains for the future to determine. If they are not, other perhaps more complex questions and techniques will have to be evolved. The questions asked by this program are: (*a*) What are the time interval histograms of the recognized excitatory postsynaptic potentials (EPSP's), the recognized inhibitory postsynaptic potentials (IPSP's), the action potentials (spikes),? (*b*) What is the likelihood that given one particular type of event response out of the previous three, that a second particular type of event response will occur within a given delay-

*This work was supported by the Bureau of Naval Weapons, Department of the Navy, under Contract NOw 62-0604-c.

time class interval, independent of whether the event responses are contiguous, as with the histograms?, (*c*) What are the amplitude histograms of the recognized EPSP's and the IPSP's? and (*d*) What is the probability that given a particular type of event response of a given amplitude class, a particular type of event response of the same, or different, amplitude class will occur within a given delay-time class interval?

## EVENT RECOGNITION

In order to appreciate better the results of the computer program to be discussed, some background in the preceding collection and preliminary processing of the neuronal data is necessary. Figure 1 illustrates in block diagram form the data handling from initial collection through the event recognition program, and finally, analysis.

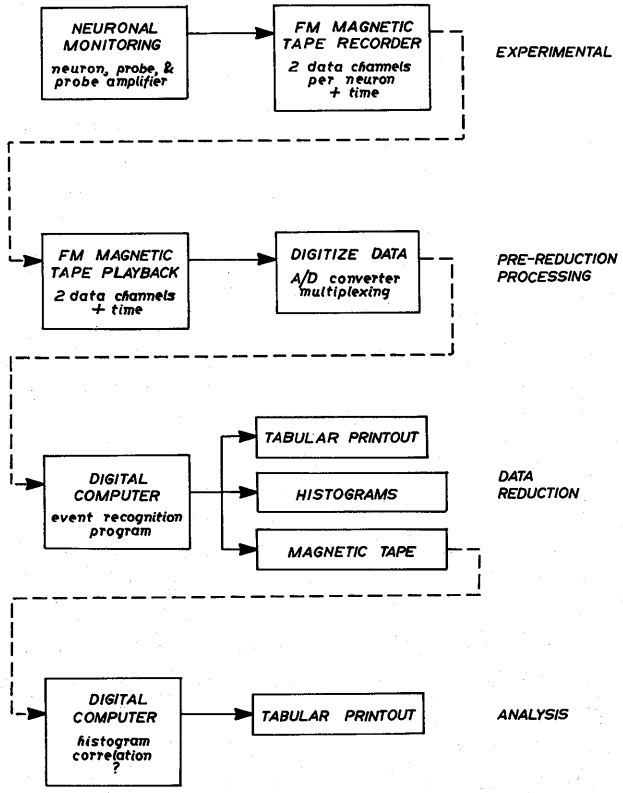



Figure 1. Block diagram of neuronal data flow.

The experimenter monitors the transmembrane electrical response of a neuron via his probe. The electrical signal is amplified and bifurcated into high and low gain channels. The high gain channel is devoted to subthreshold responses comprised of excitatory and inhibitory postsynaptic potentials (EPSP's and IPSP's respectively). Suprathreshold activity (spikes) are clipped in this channel. The low gain channel, however, contains the entire signal. Both channels of information are placed on magnetic tape in the form of an FM signal. This separation of the signal is important for signal-to-noise reasons, as well as ease and economy of automatic reduction.

In the second area, the preprocessing data reduction is in the form of a conversion of the analog FM signal to digital form. The two channels, high and low gain, are retained in the digitizing process through a multiplexing or interlacing process. The digital form of the neuronal response is in turn placed on magnetic tape. In addition, a time code is placed on the digital magnetic tape in order to ascertain the time of occurrence of recognized responses.

The third step in the process is the insertion of the digital information into the digital computer under control of the program for recognizing the neuronal events viz., the EPSP's, IPSP's and spikes. Figure 2 represents a segment of typical neuronal data which has been processed through the first three techniques outlined in Fig. 1. In this illustration, the symbol E on the graphical portion denotes the initiation of an EPSP. The * in the rise time (DELTA T) column signifies an event which was interrupted in its rising phase by a second event, in this case, another EPSP. The event recognition program is able to recognize these responses with an accuracy of approximately 95 percent. One form of output from this program is a tabular listing—as shown in Fig. 2—listing the type of event, the time of its initiation, the rise time and its amplitude excursion. If the recognized event was a spike, not shown in this illustration, the tabular printout listing also includes its duration.

To date, the minimum event amplitude which may be reliably detected is 250 microvolts. The dynamic range of measurable activity between sub- and suprathreshold events may be in excess of 100 millivolts. To operate over so large a dynamic range in a recognition mode is one basis for the previously mentioned split of the data into two channels.

One additional output from the event recognition program is recorded on digital magnetic tape. This is the same data contained in the tabular printout. The purpose of this output is to enable statistical

SUBTHRESHOLD (* FOR INCOMPLETED EVENT)
SUBTHRESHOLD ( () FOR EVENT BELOW AMP.)

| TYPE | INIT. TIME | DELTA T | DELTA AMP. |
|---|---|---|---|
| EPSP | 0.2435126 2E+04 | * | 0.555758E+00 |
| EPSP | 0.2435127 2E+04 | 0.671387E-03 | 0.635152E+00 |
| EPSP | 0.2435128 2E+04 | 0.167847E-02 | 0.269940E+01 |
| EPSP | 0.2435147 2E 04 | 0.100708E-02 | 0.158788E+01 |
| EPSP | 0.2435156 2E 04 | 0.131226E-02 | 0.166727E+01 |
| EPSP | 0.2435161 8E 04 | 0.100708E-02 | 0.793940E+00 |
| EPSP | 0.2435170 9E 04 | 0.335693E-03 | 0.793940E+00 |
| EPSP | 0.2435175 9E 04 | 0.671387E-03 | 0.952728E+00 |
| EPSP | 0.2435179 5E 04 | 0.671387E-03 | 0.635152E+00 |
| EPSP | 0.2435188 1E 04 | 0.167847E-02 | 0.111152E+01 |
| EPSP | 0.2435194 9E 04 | 0.131226E-02 | 0.873334E+00 |

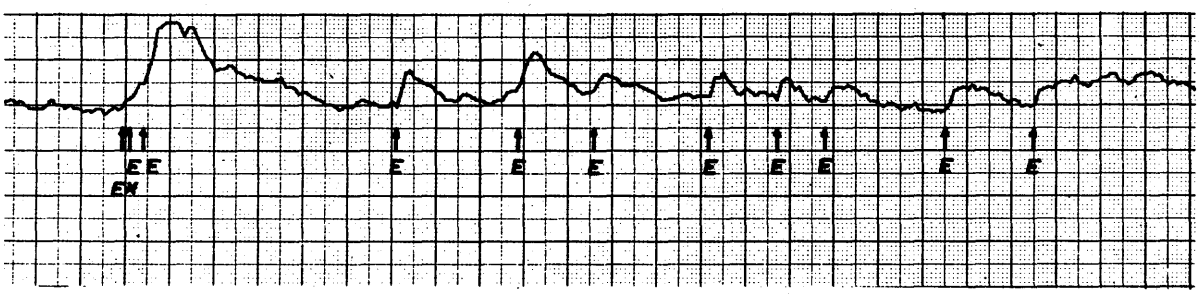



Figure 2. Sample of neural-electric data (bottom) and the results of the event recognition program. E±. . . denotes the multiplying power of ten's exponent. Time is in seconds, amplitude in millivolts.

analyses to be performed on the digital computer, rather than manually.

## HISTOGRAM GENERATION

A common form of preliminary statistical analysis, associated with intracellular neuronal response data, is to count the number of times a given variation occurs within a sample. These discrete probability distributions, or histograms, are divided into class increments of the variation. For neuronal data, two types of variation are of interest. In one type, the variable is amplitude, thereby generating an amplitude histogram. For variations with time, there is the second type, a time interval histogram.

The production of either or both of these two main types of histograms may be performed during the event recognition program, or from the reduced data contained on magnetic tape.

### *Amplitude Histogram*

In examining the neuronal data, only two types of events are considered for inclusion in the amplitude histogram category. They are the EPSP's and the IPSP's. Spikes are not included in this form of histogram. In addition, only those events which have definitely attained their response peak prior to initiation of another event are included. The events which are interrupted—those marked with an * in Fig. 2—by the onset of another event are not included since no accurate determination of their peak value may be made.

The procedure governing assignment of an event to a particular amplitude class interval, or amplitude 'box,' is depicted in a very simplified flow chart shown in Fig. 3. There are separate amplitude histograms for the EPSP's and the IPSP's.

In Fig. 3, the symbol $\Delta A$ represents the basic amplitude class interval. To avoid truncation errors, which may be quite misleading, $\Delta A$ (an input variable to the program in millivolts) must be an intger multiple of the digitizing amplitude resolution. N is an integer representing the assigned class interval and ranges from 1 to 200.

### *Time Interval Histogram*

There are three separate time interval histograms: one each for the EPSP's, the IPSP's and the spikes. The requirement on the EPSP's and IPSP's that they display a discernible peak (i.e., be unin-

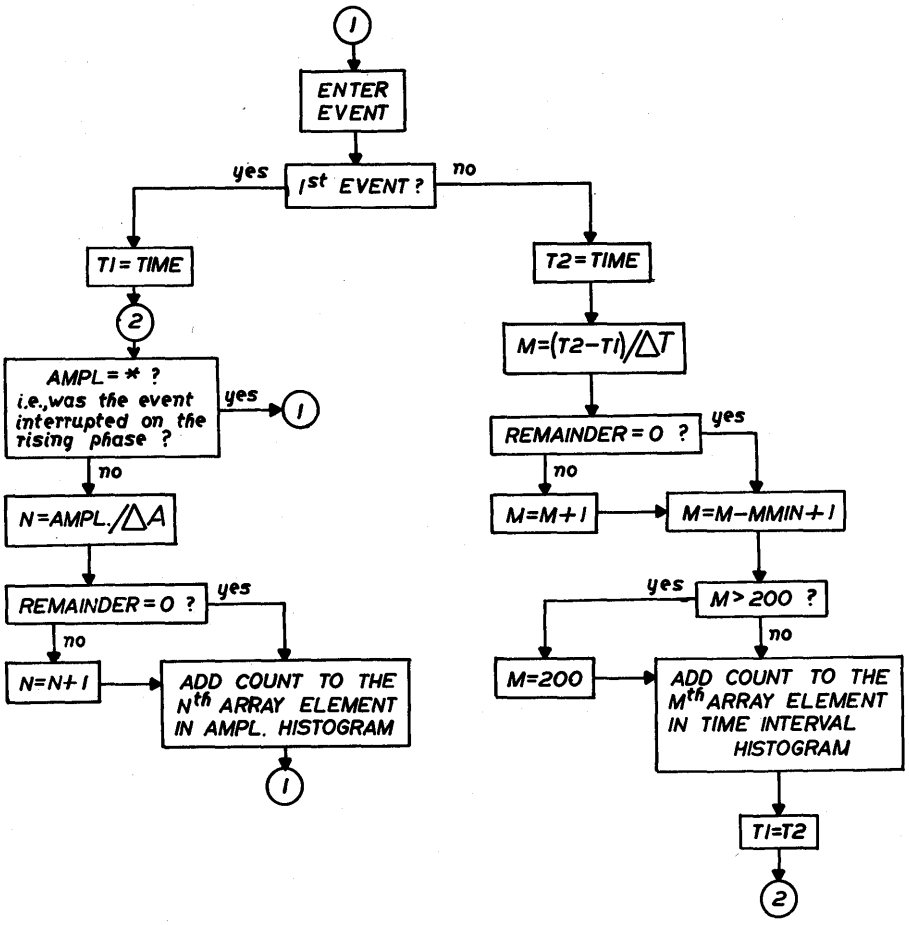



Figure 3. Simplified flow diagram for time interval and amplitude histogram generation.

terrupted on the rising phase), as in the case for the amplitude histograms, is absent. All that is required in the form of an amplitude criterion is that their amplitude be at least as large as the minimum acceptable at the time of the peak, or of their interruption by a supervening event.

The class interval for these histograms is $\Delta T$ seconds, an input variable to the program. The measured times between contiguous like events is taken as the time difference between initiation of one event and initiation of the next like event. The first is termed the reference event, while the second is the crossreference event. Their times are labeled as T1 and T2 respectively. Similar to the amplitude class width N, M is an integer denoting the assigned time class interval or 'box,' and its value ranges from a present minimum value MMIN to MMIN + 200. Since the time intervals between adjacent like events may exceed $200\Delta T$, counts indicating that two events separated by $200\Delta T$ or more are all placed in the $200^{th}$ box.

*Histogram Readout*

As was mentioned previously, the histogram may be formed concurrently with the operation of the event recognition program, or in subsequent operations involving the reduced data on magnetic tape. In either case, selection of particular histograms is an input variable. That is, a choice may be made, prior to program operation, of which histograms are to be printed out. Another program input variable is the periodicity of histogram printout. This option will supply the investigator with a time history of the histogram formation if he so desires every $\Delta T_J$ seconds. The time history is supplied in two forms: (*a*)

the total of the $\Delta T_J^{th}$ time of printout and (*b*) the difference between the histograms at the times associated with $\Delta T_J$ and $\Delta T_{J-1}$.

Printout of all histograms may be in either of two forms, tabular or graphical, or both. Collateral with the tabular printout is the total number of counts entered into each particular histogram at the time of printout. Figures 4 and 5 are representative samples of the graphical plots of an EPSP amplitude histogram and time interval histogram respectively. The original data was obtained from a motoneuron in the spinal cord of a cat.*

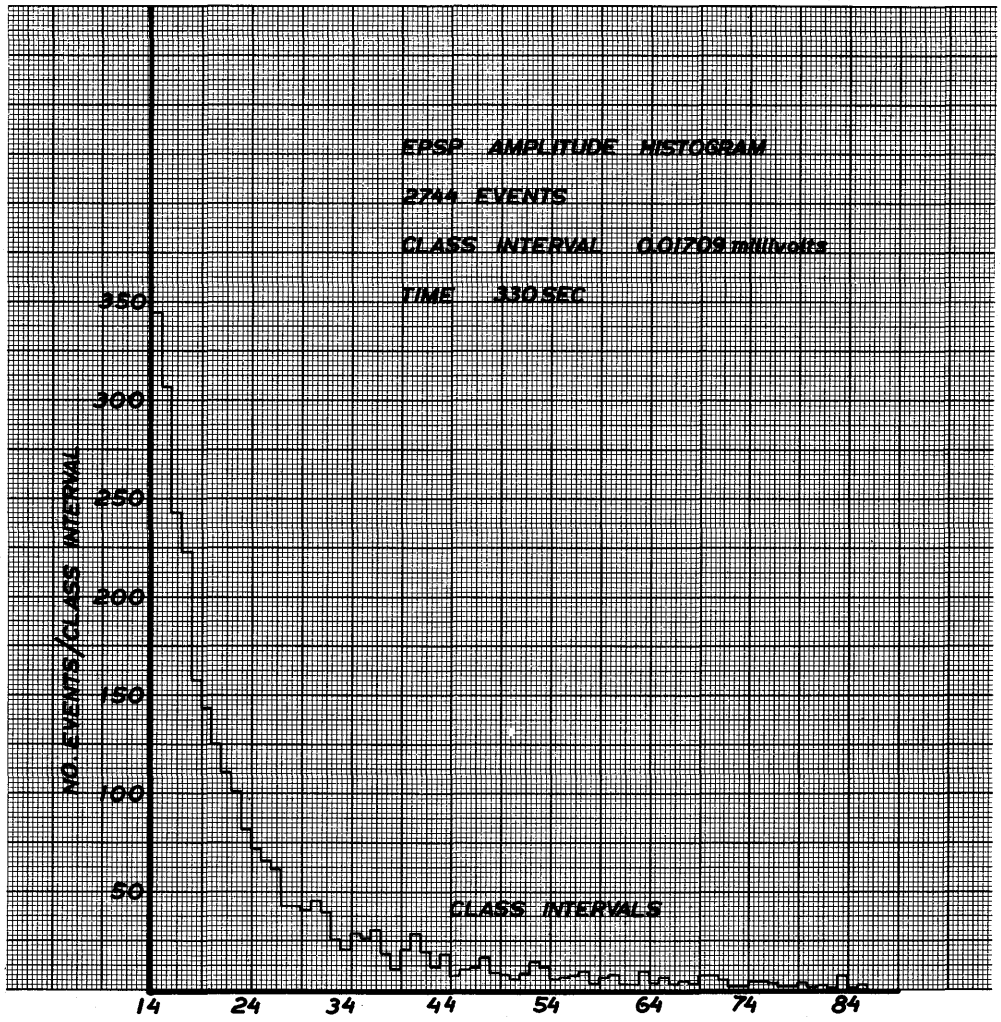



Figure 4. Amplitude histogram for excitatory postsynaptic potentials (EPSP's) obtained from a motoneuron in a cat spinal cord. Only EPSP's which were uninterrupted during the rising phase are included. Minimum acceptance amplitude was 250 microvolts.

## TIME CORRELATION OF EVENTS

The program described here is employed primarily with the data reduced by the event recognition program and stored on magnetic tape. However, it may be used on other data in card form in the correct format. Due to storage limitations of the digital computer, it cannot be performed concurrently with the event recognition program.

In essence, the time correlation of neuronal events is an extension of the frequency distribution

*Obtained from experimental data supplied by Dr. T. G. Smith, Spinal Cord Section, Laboratory of Neurophysiology, NINDB, National Institutes of Health, Bethesda, Md.

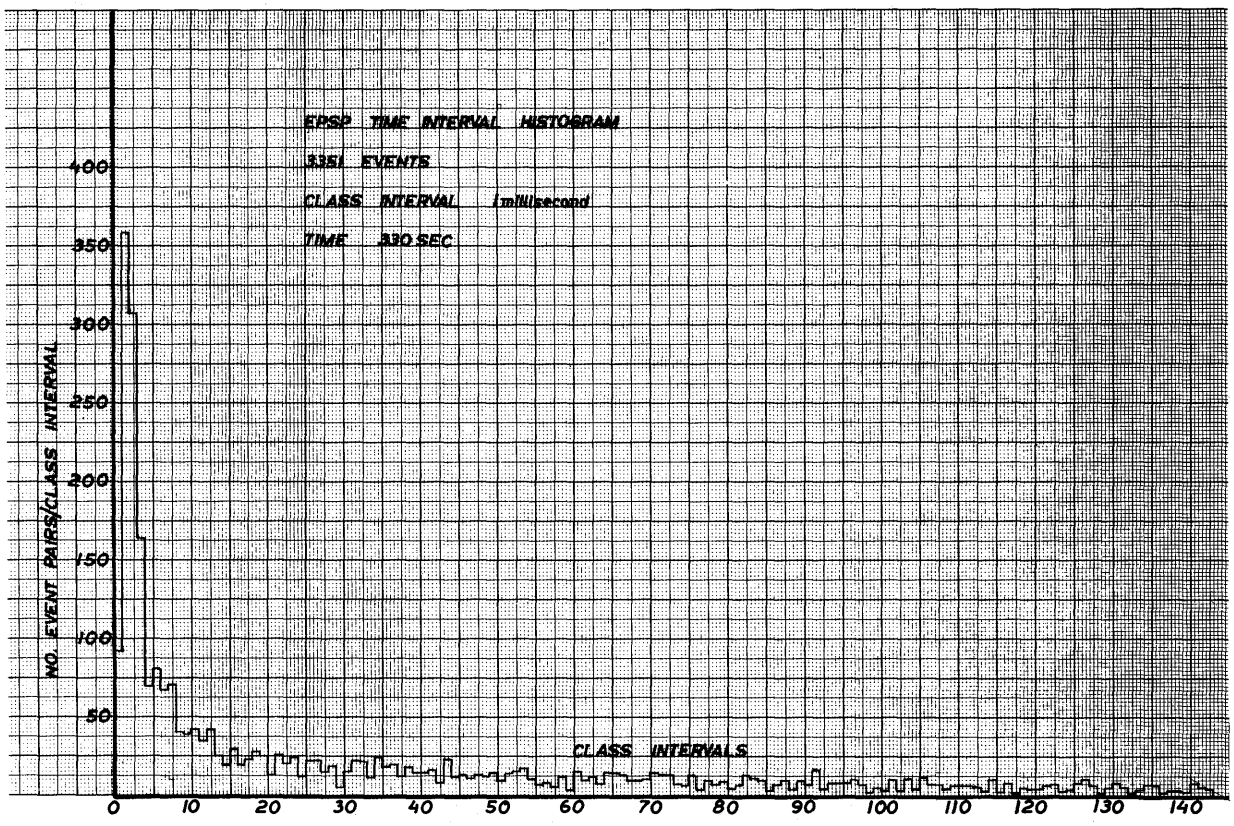



Figure 5. Time interval histogram for the same data as in Fig. 4. Here, however, all recognized EPSP's are included as long as their measured amplitude difference from initiation-to-peak or initiation-to-interruption was at least 250 microvolts.

analysis just described. Whereas the time interval histograms were involved with contiguous and like events, this form of analysis is not so limited. This program, referred to as the *correlation program,* measures the probability (or correlates) that, given a particular type of event (EPSP, IPSP or spike), there will be another event of a particular type within a given delay time class interval. The two events, reference and cross-reference, need not be adjacent; nor need they be of the same type.[3]

This type of analysis is felt to be much more meaningful than the normal frequency distribution, or histogram, form. The observed EPSP's and IPSP's are usually each from multiple input pathways to the single neuron being monitored. Because the various signals on the multiple pathways may be asynchronous, indications of underlying signal processing or event relationships in time by histogram may be undiscernible.

There are two main types of time interval correlations performed. One type, designated type-A, is independent of the amplitude of either the reference or the cross-reference event. All events included in the time interval histograms are included in this type. The second type of correlation, designated type-B, is performed with a condition of amplitude imposed. Only subthreshold events (EPSP's and IPSP's) which meet the requirements for inclusion in the amplitude histograms are eligible for this type of correlation. All spikes may be included in either type.

### *Type-A Correlation*

The type-A time interval correlations are in turn divided into six categories as a function of the reference and cross-reference event types. These six categories are referred to as the IPAIR's. Figure 6 lists the various A-types and their collateral ref-

TYPE-A CORRELATIONS

| IPAIR | REF. EVENT | CROSSREF. EVENT |
|---|---|---|
| 1 | EPSP | EPSP |
| 2 | IPSP | IPSP |
| 3 | SPIKE | SPIKE |
| 4 | IPSP | EPSP |
| 5 | EPSP | IPSP |
| 6 | EPSP | SPIKE |

TYPE-B CORRELATIONS

| TYPE | AMPL. CONDITIONS REF. | AMPL. CONDITIONS CROSSREF. | MAX. NO. DELAY CLASS INTERVALS |
|---|---|---|---|
| B(MD,I,L) | $(1 \le L \le 16)\Delta A$ | $(1 \le L \le 16)\Delta A; NI = N2$ | MD=200 |
| B(MC,N,L) | $(1 \le L \le 16)\Delta A$ | $(1 \le L+N)\Delta A$ for $N=0$<br>$(L+N-1 \le 16)\Delta A$ for $N>0$ | MC=50 |

Figure 6. Classification of time interval correlation types. Type-A includes all events meeting the minimum amplitude criterion. Type-B performed for a particular type-A, and in addition, has conditions of peak amplitude imposed.

erence and cross-reference event types, as well as the conditions of amplitude included in the various type-B correlations.

Associated with each type-A correlation are M class intervals of time. The minimum (MMIN) and maximum (MMAX) values of the class intervals are input parameters to the program. The maximum difference allowed between MMAX and MMIN for any single processing run is 200. The values of M must be positive and consecutive integers. The width, $\Delta T$ in milliseconds, associated with each delay time class interval is also an input parameter to the program.

Figure 7 is a simplified program flow chart for the correlation program for the type-A. Data from the magnetic tape is read into a buffer storage. Contained in the storage is data designating the type of event, its initiation time and its amplitude between initiation and peak; or if the event was interrupted, an equivalent of the * (Fig. 2) indicating an interruption in place of amplitude.

The data is examined until an event initiation time is found which corresponds to an input *start* time. The time of the event is set equal to $t_j$. This time is checked against the input *stop* time and the print time criterion. If $t_j$ is equal to, or greater than, either of these, the PRINT routine is entered. This will be discussed later.

The type of event (I) is determined (i.e., EPSP, IPSP or spike) and a count is entered in the appropriate reference event counter {KE(I)} for that type of event. Next, the amplitude is checked to see if it was a completed event or not. If the event was completed (no *), its type is checked against the type of reference event to be used for the type-B correlation; that is, an ICLASS input corresponding to one of the IPAIRS. Assume that the reference event is the type to be used for the type-B correlations.

The event amplitude class interval is determined, LT. The value of LT is checked against 16 input amplitude classes. If the event's value of LT corresponds to one of these classes, a count is entered into the appropriate type-B reference event counter {KEB(L)}.

After such preliminary processing of the reference event, the computer advances to the next event in

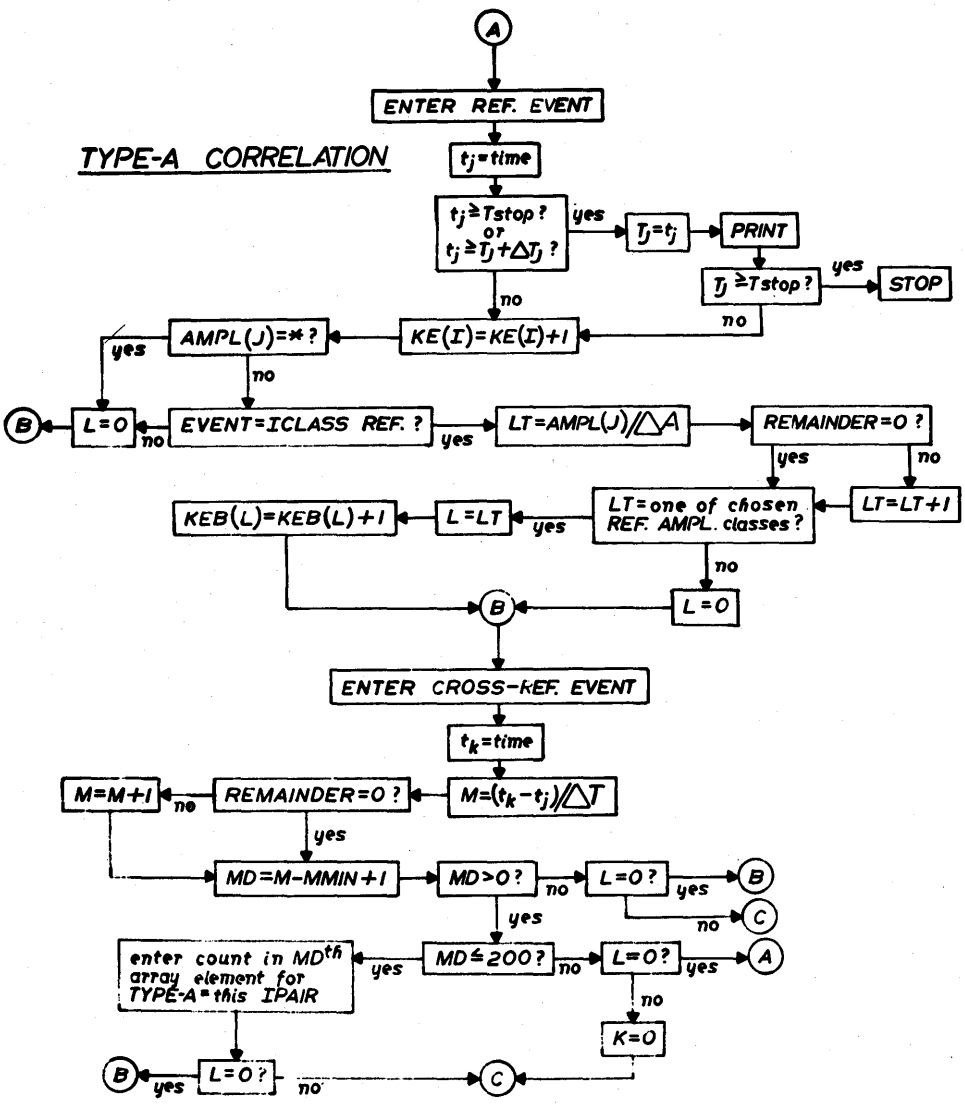



Figure 7. Simplified program flow chart for the type-A correlation mode.

storage. This event is the first cross-reference event. A check is made (compute the M-class interval) to ascertain if the difference in the initiation times of the reference and cross-reference events lies between (MMAX) × (ΔT) and (MMIN) × (ΔT). If the time difference is in excess of (MMAX) × (ΔT) and the reference event was interrupted in amplitude, a new reference event is chosen. A time difference less than (MMIN) × (ΔT) calls for a new cross-reference event. Assume that neither of these conditions was encountered.

At this point, a determination of the cross-reference event type is made (IPAIR determined), and a count in the appropriate M-class interval for the corresponding type-A correlation is made. The reference event is checked again at this point to see if it is a completed or interrupted event. If it is an interrupted event, no entry is made into the type-B correlation routine, and the program then examines the next data point as a cross-reference event, and makes the previously mentioned time checks, etc. If, in this portion of the loop, the determined class interval falls outside of either the minimum or maximum values of class interval, entry into the type-B correlation is made if the reference event is uninterrupted, and is the proper amplitude class for a type-B reference event ($L \neq 0$).

*Type-B Correlations*

Collaterally with the six type-A correlations, time analysis on conditions of amplitude may be performed to a limited extent. Due to storage limitations in the digital computer, only a limited number of combinatorial conditions of event type, conditional amplitude class interval and delay time class intervals may be imposed. For any one processing run, the type-B correlations may correspond to only one particular type-A IPAIR; e.g., EPSP's correlated with EPSP's, where both the reference and cross-reference events have an amplitude condition imposed. The type chosen is an input parameter labeled ICLASS, where the ICLASS equals one of the six allowed IPAIRS. Referring to Fig. 8, the processing procedure for the type-B is given in the following (it will be assumed here that the reference event had a measurable amplitude so that entry into this portion of the program was made).

The first check in this portion of the routine is to determine if the combination of reference—cross-reference events is the chosen ICLASS.

Following this, a check is made upon the amplitude of the cross-reference event to determine if it is an uninterrupted event or not. Assume for discussion purposes that the ICLASS = IPAIR and that the cross-reference event had a measurable amplitude. The amplitude class corresponding to the cross-reference event (NT) is then determined. This class is adjusted to the class interval of the reference event by forming N = NT – LT.

The allowable class intervals for the cross-reference events are designated by the symbol N, which is also an input to the program. N may vary positively or negatively, and is an integer. The maximum num-

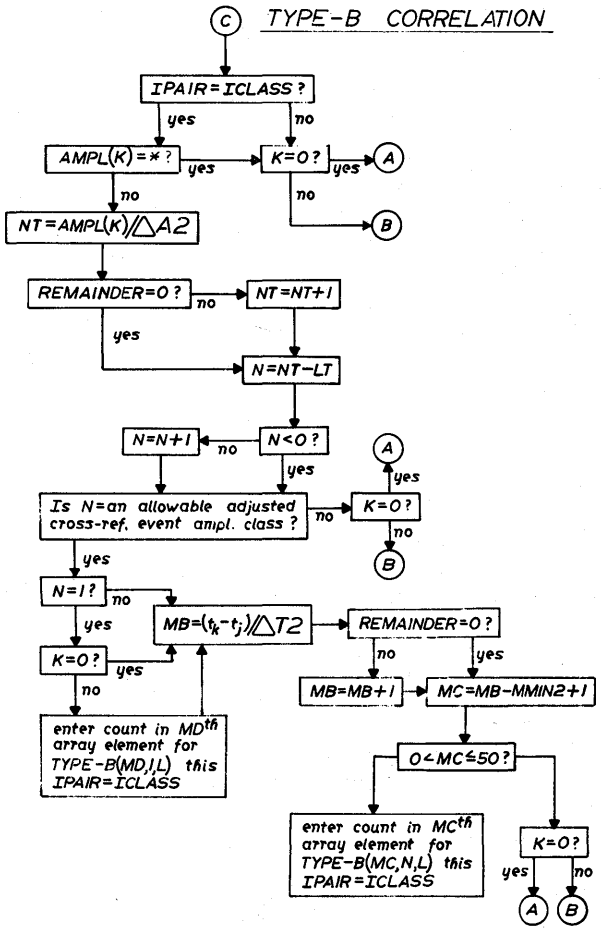



Figure 8. Simplified program flow chart for the type-B correlation mode.

ber of values for N for a particular processing is ten. For each value of L, a conditional correlation is possible for each value of N. The function of N is to adjust LΔA for the cross-reference events. For *positive* values of N, the maximum allowed cross-reference amplitude class for *each* value of L is L(ΔA)–(N-1)ΔA. For negative values of N, the minimum allowed class is L(ΔA)–|N|ΔA.

The actual allowed cross-reference amplitude classes are determined by imposing the condition that the determined value of N must equal one of the L values. This condition is imposed in order to obtain conveniently the number of cross-reference events employed in any amplitude-conditional correlation. The L inputs may number 16, and the N inputs 10.

Assume that the value of N corresponds to one of the input values, and that this value was N = 1. A check is made (K = 0?) to determine if the time difference already made is equal to an allowable MD class interval. If it is, a count is made into the appropriate array element for type-B (MD, 1, L), where this array has the same 200 delay class intervals as the type-A arrays.

The next step is to determine if the time difference is one of the 50 class intervals of the type-B array, where these 50 class intervals do not necessarily have to correspond to any of the 200 class intervals of the type-A's or the B(MD, 1, L)'s. The new class interval is determined using the input delay class width ΔT2, and a check is made to determine if this class interval, MC, is allowed. If it is, a count is made in the appropriate array element for the type-B (MC, N, L). Following this, either a new reference event or cross-reference event is obtained, and the processing continued until a reference event's time is found which equals or exceeds the input stop time.

*Printout*

The results of the correlation processing may be printed as often as desired. A $\Delta T_k$ input to the program determines the frequency of print in real (experimental) time, not machine time. The time of each reference event is examined. If its time is $\geqq$ to the start time plus $\Delta T_k$ (k = 1, 2, 3, . . . . ), the results of the correlation as of the time of that event are printed, contingent upon meeting certain print criteria.

Since the processing is slowed every time the digital computer has to print results, it is economical to print only when it is felt the results may have meaning. Hence, there are print criteria included in the program. At present, there are three print options from which one may choose. Fig. 9 lists these in the form of a simplified flow chart.

Upon entry into the print routine, certain values are computed for each array. These values are the number of counts, or correlations, expected if one assumes that the events are Poisson-distributed in time. The expected number of correlation counts per class interval is found by forming the product of the number of reference events for the particular array being examined (N1) and the number of cross-reference events possible for that array (N2; N2 may equal N1 for certain array types as shown in Fig. 9). This product in turn is multiplied by the individual delay class interval width of the array being examined. This product is divided by the time difference between the time of the first reference event employed in the correlation and the time of the last reference event corresponding to the print time. The value so determined, X, is the average number of correlation counts expected if the process being examined has a Poisson distribution of event times. One standard deviation is determined by calculating the square root of X.

In the first option, in order to determine if a particular correlation array is to be printed, the counts in all of the class intervals of that array are summed. The sum is divided by the square root of the product of the number of reference events and the number of cross-reference events. This quotient, $P_c$, is compared to an input criteria number. If it is larger, the array is printed out.

The second option employs the previously computed value of X. X is multiplied by the number of delay class intervals to form a value Y. Y is the expected average number of counts for the entire array, and $\sqrt{Y}$ the standard deviation. If the sum of the correlation counts in the array exceeds $P_c = Y \pm z\sqrt{Y}$, where z is an input parameter, then the array is printed out.

The third option consecutively examines each element in the array. As soon as an array element (where the count $\neq 0$) is found which exceeds $X \pm z\sqrt{X}$, the array is printed. Again, z is an input parameter.

In all options, if the print criteria is not met, the array type, the value of X and $\sqrt{X}$ for that array, and the value of N1 (the number of reference events for that array) are printed. In addition: for options

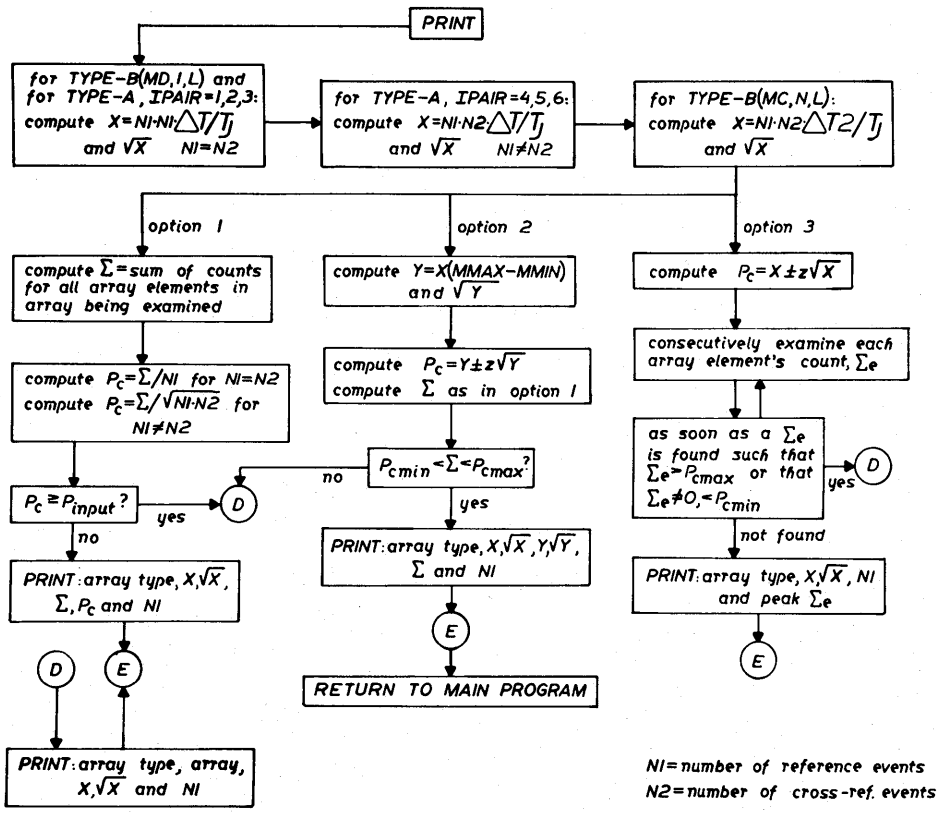



Figure 9. Simplified program flow chart for the Correlation Program's *print* routine.

1 and 2, the sum of the array counts and the value of $P_c$ are printed; for the third option, the peak count of an array element is printed. When the print criteria is met, the following is printed: array type, array, X, $\sqrt{X}$ and N1. Following the type-A's, the time of the last reference event is given.

Printout to date is in tabular form only. To obtain a graphical plot, an auxiliary program is employed where the count values (to a scale normalized to 1000 maximum) are consecutively entered onto cards as a function of the delay class interval.

## RESULTS

In addition to control test runs, experimental data from several motoneurons in the spinal cord of cats has been processed through the entire set of programs herein described. The results have been quite fruitful.* Complete histograms and type-A and B correlations have been obtained for a total of over 55,000 events. Computation time for event recognition, histogram generation and the various correlations (e.g., type-A's from 0 to 2000 ms in 1 ms delay class intervals) totaled less than 20 hours. The reduction in time compared to processing the same data by hand is fairly obvious.

Fig. 10 illustrates one particular result obtained from the just mentioned processing. The plot is correlation counts on the ordinate and delay class intervals along the abscissa. The reference and cross-reference events were both EPSP's. The plot indicates that there is a preference for EPSP's to occur fairly close together. This was borne out by recalling the EPSP time interval histogram previously seen (Fig. 5). The remaining portion of the correlogram would indicate that the activity is more or less a Poisson process. The next illustration (Fig. 11) is a time interval histogram from the same experiment but at a later time. The input ac-

*These results will be detailed and the possible anatomical and physiological mechanisms underlying them will be discussed in a subsequent paper.

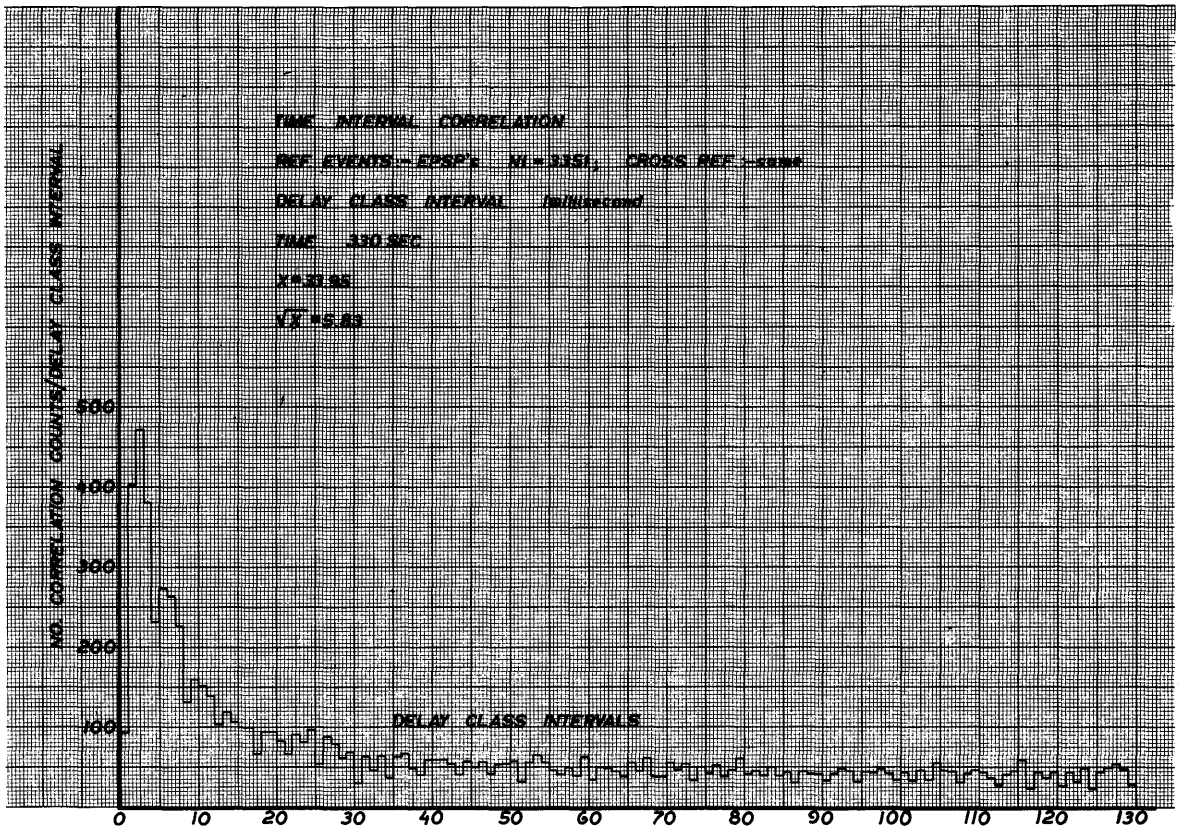



Figure 10. Time interval correlation (auto-) results for the EPSP's used for the generation of Fig. 5. The time listed in the legend is the duration of the experimental data analyzed. X is the expected average number of correlation counts assuming a Poisson distribution.

tivity to the monitored neuron had increased considerably. This time interval histogram shows the same preference of EPSP's to occur close together. In addition, there is a suggestion of a preferred spacing in the vicinity of 40 to 50 ms, which might be overlooked.

The next figure (Fig. 12) is a correlation plot similar to that in Fig. 10. It encompasses the same data as the immediately preceding histogram. The preference of EPSP's to occur at spacings of 50 ms is not obscured in this plot as it was in the histogram. In the histogram, the peak occurs around 40 ms; whereas in the correlogram, the peak is at 50 ms. The difference between these two peaks is accounted for by the high preference of closely spaced EPSP's pairs obscuring the real peak in the histogram, but not in the correlogram. Furthermore, examination of the next illustration, Fig. 13, shows a relationship between the EPSP's and IPSP's, which occurred between the times encompassed by Fig. 10 and Fig. 12, and which would not be discernible from examination of histograms.

The correlograms processed to date have uncovered activity relationships between neuronal responses which were not anticipated by monitoring during the original experiment or by histograms. They were not anticipated for two main reasons. First, standard histograms are not overly sensitive to relationships of responses due to numerous sources mixed together, and prior to this, the majority of analyses of the time relationships between neuronal events has been in the form of time interval histograms. Secondly, detailed analysis of several thousand events obtained in a single experiment has been a long time-consuming process which few experimenters have been willing or able to undertake.

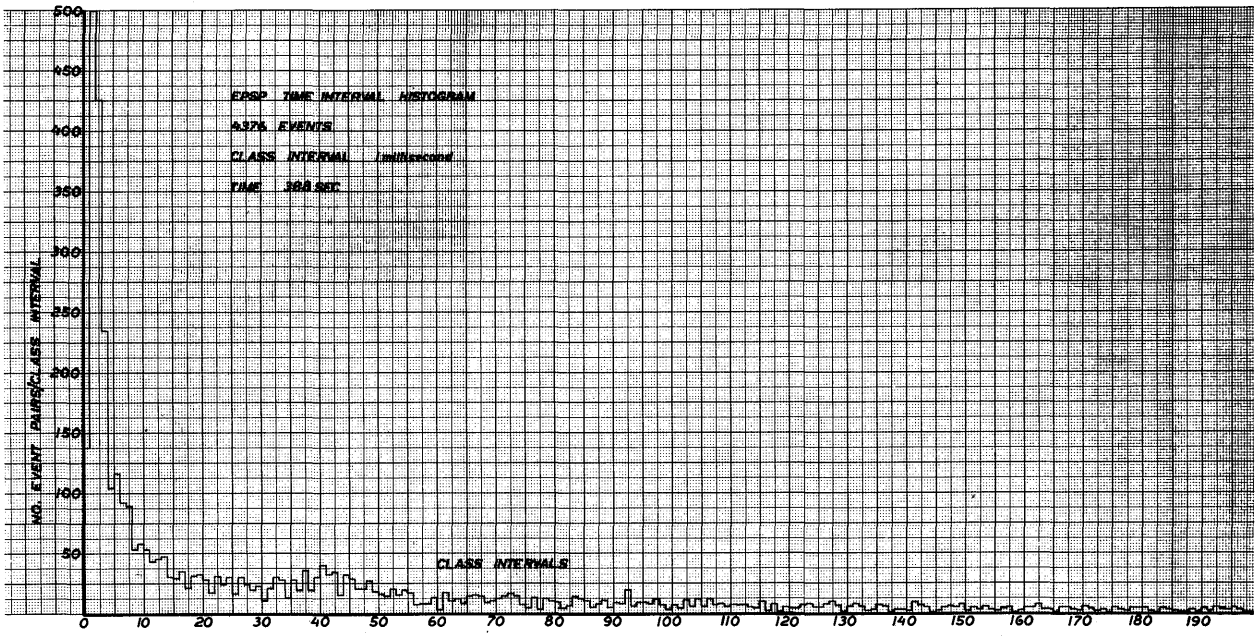



Figure 11. Time interval histogram of data from the same experiment as the previous figures. However, this histogram is for a longer duration. Note the suggestion of a periodicity at 40 ms.

## REFERENCES

1. F. F. Hiltz, "A Method for Computer Recognition of Intracellularly Recorded Neuronal Events," *IEEE Trans. on Bio-Medical Engineering,* vol. BME-12, no. 2, pp. 63-72 (April 1965).

2. N. R. Lakey, "A Fortran Program for Intracellular Event Recognition," *IEEE Trans. on Bio-Medical Engineering,* vol. BME-12, no. 2, pp. 73-87 (April 1965).

3. F. F. Hiltz and C. T. Pardoe, "A Correlator of Time Intervals Between Pulses," *IEEE Trans. on Bio-Medical Engineering,* vol BME-12, no. 2, pp. 113-120 (April 1965).

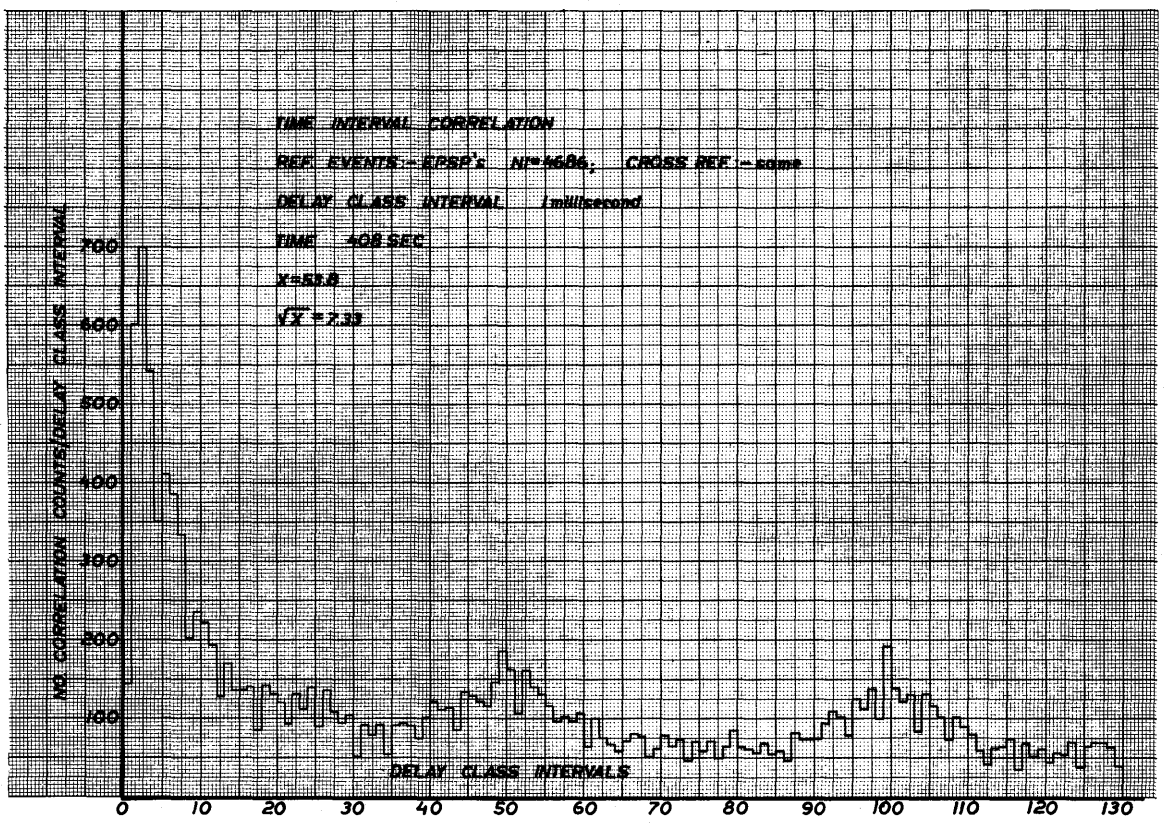



Figure 12. Time interval correlation (auto-) results for the EPSP's in the same experiment. The data was for a slightly longer duration than that of Figure 11. Note the pronounced periodicities, and at a different period than indicated by the histogram in Fig. 11.

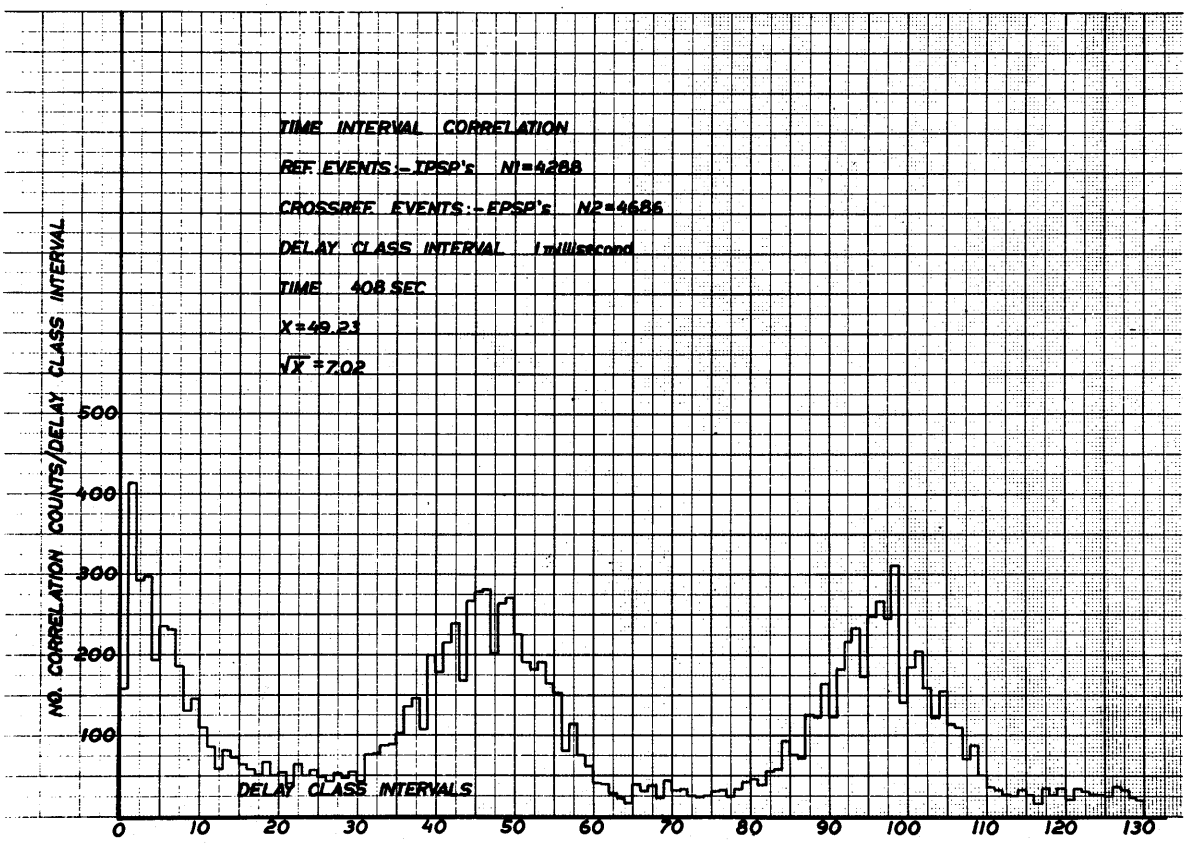



Figure 13. Time interval correlation (cross-) results for the EPSP's *and* the IPSP's occurring during the same experimental time encompassed by Fig. 12. Separation of the correlation peaks is nominally 50 ms, with the first peak occurring at 45 ms, compared to the 50 ms peak in Fig. 12.

# INFORMATION PROCESSING OF CANCER CHEMOTHERAPY DATA

Alice R. Holmes and Robert K. Ausman
*Health Research, Incorporated*
*Roswell Park Memorial Institute*
*Buffalo, New York*

## INTRODUCTION

Roswell Park Memorial Institute, one of the largest cancer research and treatment hospitals in the world, conducts an extensive program of cancer chemotheraphy, the use of chemical agents in the treatment of cancer. Roswell Park, acting as the statistical center for participating hospitals throughout the United States, is responsible for implementation, control, and follow-up of several chemotherapy studies. To facilitate the handling of the diverse and extensive amount of data made available by the participants, the architects of these chemotherapy studies found it desirable and necessary to automate as many facets as was feasible. This paper describes the procedures employed in a system which extensively treats medical data.

## GENERAL PRINCIPLES

For each study the participating hospitals throughout the United States follow the same procedures, thus insuring essentially identical treatment of all patients. This procedure guarantees uniformity which facilitates statistical analyses. Prior to the initiation of any given study, guidelines are determined and a protocol to be followed is established, stipulating qualifications a patient must meet prior to entry in the study such as age, medical history, blood picture, and previous therapy. Also specified is the conduct of the study, the method of drug administration, dosage regimen, frequency of blood counts, and follow-up policies. Investigators are expected to conform as closely as possible to the protocol.

In conjunction with the writing of the protocol is the design of the forms that enables the physician to collect the information requested. The proper design of these forms is extremely important, for it determines the quantity and order of the material to be stored in the computer for later use in several types of statistical analyses. To facilitate the handling of the data collected, most questions are worded for an objective answer. An important aspect of the system is the return of these forms to the statistical unit as soon as the pertinent patient information is available to the physician. In this manner there is continuing assurance that the protocol is being followed correctly. In addition, the immediate response allows for current reports on the progress of the study.

A patient is entered in the study by a referring physician who telephones the statistical unit. At

this point, he receives information as to the specific drug, dosage, and frequency of administration. Subsequently, the physician receives written confirmation of the entry and procedures of therapy.

During the exchange of information between Roswell Park and the participants there are several checks on the completeness and accuracy of the protocol procedures. If the forms are received as incomplete, inaccurate, or if they contain discrepancies, physicians from the statistical center staff note these errors. The corrections are indicated in memos which are prepared by a separate semiautomated system and recorded on the IBM Magnetic Tape Selectric Typewriter and sent to the participants. If corrections have not been received at the close of a 30-day period, names of delinquents are run through the Magnetic Tape Selectric Typewriter, which prepares a reminder form.

Results of the 3-month follow-up examinations, perhaps the most important phase of the studies, are submitted to determine the effect, if any, of the chemotherapy. In this manner it is possible to tell if there has been a recurrence of cancer, the time lapse between therapy and recurrence, or most significantly, if there has been no recurrence at all.

The methods of handling the data received, previous to automation, were time-consuming, clumsy, and inaccurate. The information on each patient chart was transferred manually onto three separate code cards. Because of the limited available space on the cards, only a summary of the data could be coded. For example, the total amount of drug given was recorded, but the individual doses and dates of administration were eliminated. In addition, only the lowest blood count was punched in the card, eliminating the daily record. This system made it impossible to analyze the effectiveness of the drugs accurately and completely.

As the number of studies grew, the staff became aware of the absolute necessity for a method which would allow more intricate and sophisticated recording and analysis of patient information. A means was devised by which a greater amount of data could be recorded with less manual labor. With the introduction of an automated system, initiated in 1963 and utilizing an IBM 1401, the chemotherapy studies have produced more reliable information, stricter adherence to the protocol, and less misinterpretation of data.

## METHODS AND PROCEDURES

The individual patient charts were chosen as the initial input documents since they afforded the most logical and practical method of handling raw data. Because of the format of the charts, it was determined that most of the data could be punched directly from the charts with an absolute minimum amount of coding.

The first stage of automation included the design of the punched card layout. Since the cards are punched directly from the patient forms, the card layout follows these forms as closely as possible. The number of columns to be allowed for an item provides for the highest possible value of the particular data. Two major controls incorporated on each punched card are the card type number and patient study number. The card type number indicates a particular card format and the information recorded on that card. As an example, the patient's former medical history may be recorded on one card, while daily observations may be found on another. The unique number assigned to each card reveals at a glance which category of information is contained on the card.

The patient study number, different for each patient, is punched on all cards for that patient. In the event that one card is separated from the others belonging to a patient, the information is not incorporated incorrectly with the data recorded for another patient.

The assigning of codes to that data which have not been written previously in numerical form must be established in conjunction with the design of the card layout. Examples are the indication of sex, to be coded with a 1 or 2, or any type of yes or no question which can be coded with a 1 for yes and 2 for no. This translation can be accomplished easily by keypunchers.

Magnetic tape is employed as the storage device for the records. The tape format can be designed in several different ways. Each tape record may be made identical to the initial input card, resulting in a card image on magnetic tape. In this case, card type and study number are retained on each tape record, insuring proper identity. The second method of formatting the tape is to combine several input cards into one tape record. Utilizing this approach, type and study number appear only one time. A third way is to combine all cards for a patient into one record; unless there is a fixed number of rec-

ords for every patient, this method is not advisable.

The present application creates a tape record for every input card. The number of records per patient may vary from 5 to as many as 100, depending on the number of daily observations recorded for a patient. If there were one record per patient, the record size would range from 400 characters to 8000 which would be cumbersome to program and analyze.

## SYSTEM ORGANIZATION

The first program in the system writes the data cards on tape. A standard card-to-tape routine is used providing there are no changes or modifications in the data at the time it is written on tape.

When the tape is not formatted in the same manner as the cards, it is converted to the proper layout. This conversion may include combining two card records into one tape record, or possibly rearranging the data so that it can be manipulated more effectively. What may be efficient for keypunching may not be efficient when working with the massive file of patient records on tape.

Once the tape has been created and sorted, a complete edit is performed by the computer in order to check all the data for accuracy, completeness, and logic or validity. If the individual who reviewed the patient form overlooked a discrepancy, or the data was keypunched erroneously, the program detects and prints the error.

There are several different types of checks. Most data are screened for validity. For example, months must be numbered 1-12, days 1-31. Some data can be coded only with certain codes (male=1, female=2). A code of 3 for sex would indicate an error.

A portion of the data is recorded on several different records as they occur; the dates are cross-checked in each location to insure compatibility. A case in point is the date of surgery which appears on three types of records. These dates must agree since there is only one possible date of surgery in the study. Calculating the patient's age from date of birth and comparing it to the given age is another important check. For these studies age is a principal factor in determining the type of therapy the patient is to receive because an error may exclude the patient from the study.

In some instances the given information determines the nature of other information that should be present. If the patient died during the course of the study, there must be a date of death, or in the event that a patient had a disease recurrence, the site and system of the recurrence must be recorded.

The amount of drug given is reviewed to insure that the dosage was in accord with the protocol. Extremely low or high blood counts are listed and checked later. Checks on all dates that must be later than the date of the surgery are made to detect any errors. All parameters specified in the protocol are examined to insure that the patient has met the qualifications for the study.

When the tape has been edited, the list of errors is sent back to the statistical unit for correction. This function may involve writing a letter to the investigator for clarification, or it may represent re-coding and repunching. After editing, the tape is merged with the master tape, thus creating a new master.

The update program demands a variety of routines. First, it allows for the inclusion of new patients on the master tape in the proper sequence. Second, it permits the insertion of additional records for a patient and also the deletion of extraneous records. Third, this program has the flexibility necessary if data on a particular record demand alteration.

After the master tape has been updated, it is ready for statistical analysis. A duplicate of the master tape, denoted as a "frozen tape," is created for the statistical work. This tape is not updated as frequently because statisticians run correlations, frequency distributions, and analyses which do not allow for the daily change of the data. On the other hand, administrators require up-to-date information to prepare reports, or to provide the investigators with current data relative to their patients. Requests for information concerning certain facets of the study are made at frequent and irregular intervals.

As soon as a patient is entered in the study, the initial entry form is sent to be keypunched. This information consists of the patient's study number, name, type of therapy, and date of surgery. The card is written on tape. The master tape is updated and contains at least one record for every patient entered to date. Reports must be prepared monthly, giving a distribution of the number of patients entered in each different drug category by hospital. Having an automated record of each patient entered allows for the production of reports that are current. Prompt fransit of the forms to the statistical

unit is of utmost importance. If forms are overdue, a letter is sent to each investigator. As a form is received, it is recorded and the patient's tape record is updated daily. Each month a delinquency program is run against the master tape to determine which patients have outstanding forms. A personal letter with a list of the missing forms is written to the investigator on the computer printer. These letters have been extremely effective in reminding the study participants that they must send information promptly. They continue to receive a letter each month until all deliquent forms for a patient have come into the office.

Another control is the determination of which patients are overdue on the follow-up section of the study. Since the patients must be examined every three months, forms summarizing the results of the examination are submitted. A list of overdue patients by hospital, as well as those patients who must be seen during the current month, is prepared by the computer. This list gives the patient's study number, name and date that the patient should have been seen.

These two systems are the most effective way of reminding the study participants. They have provided more effective control and obtain requested information with more accuracy and less time.

Specific and individual patient or drug information which may be of extreme importance to a particular physician is easily available. He can receive a printout for each patient which might include daily drug dose, any toxicity, and other data. In another case, a doctor may phone to inquire about response and/or toxicity of a particular drug to determine if he wishes to administer this drug to a patient. If results show that patients having a tumor and receiving a certain drug experience extreme toxic reactions, a doctor may not wish to administer the agent or may wish to give a smaller dose.

Frequently, requests are made to submit reports or punched cards to the National Institutes or Health, coordinators for these chemotherapy studies. Because all conceivable patient information is on tape, it is possible to provide any requested data in whatever format is desired. With the former method or present methods at statistical centers other than Roswell Park, much of this data would have to be obtained from the patients' charts and either coded, if punched cards were requested, or typed, if a report were requested. This process is long and tedious, especially if each patient must be recorded separately.

The ability to punch certain information from the master tape and make graphs with a plotter that is connected to the IBM1620 is important, as it eliminates the expenditure of many man hours. Graphs are plotted for individual patients to compare the response of their tumors (e.g., did one tumor decrease, while another increased during study, or did the tumors decrease while the patient was receiving drug, and then increase with suspension of the drug). Applications of this kind are extremely useful when developing reports or presentations concerning a particular study.

## SUMMARY

As a result of this system, more patient information is recorded with less summarizing and coding of the data prior to punching and writing on tape. The data, which always is current, facilitates the control of delinquent forms and produces reliable and representative reports.

The ability to perform more intricate and sophisticated statistical analyses has been enhanced by the system because of the improved condition and increased amount of data recorded.

The accurate and close scrutiny on the progress of these chemotherapy studies offers physicians information which previously was available to them only through the "trial and error" method.

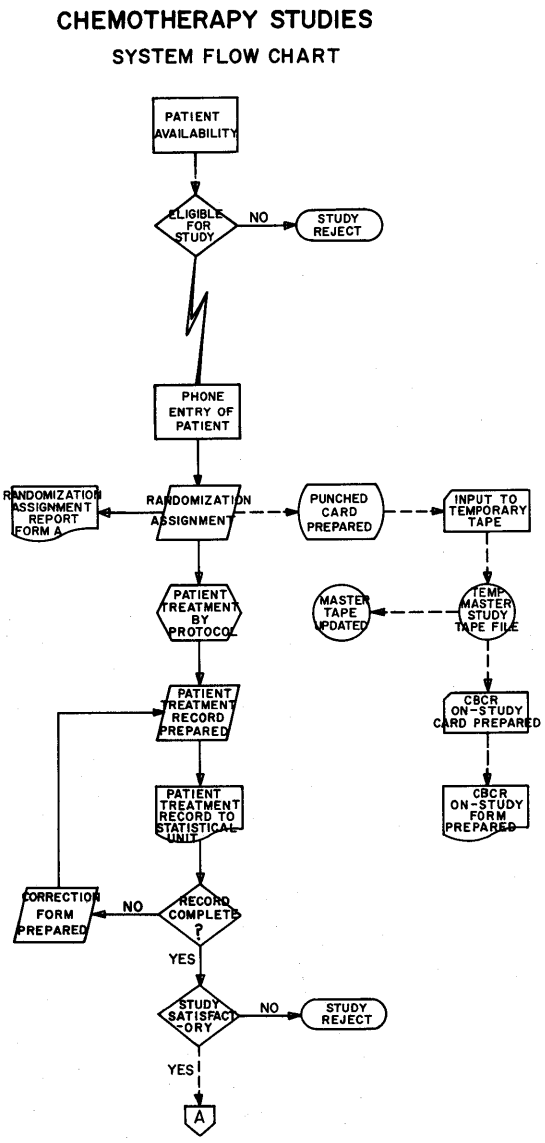



Figure 1. Chemotherapy studies — system flow chart.

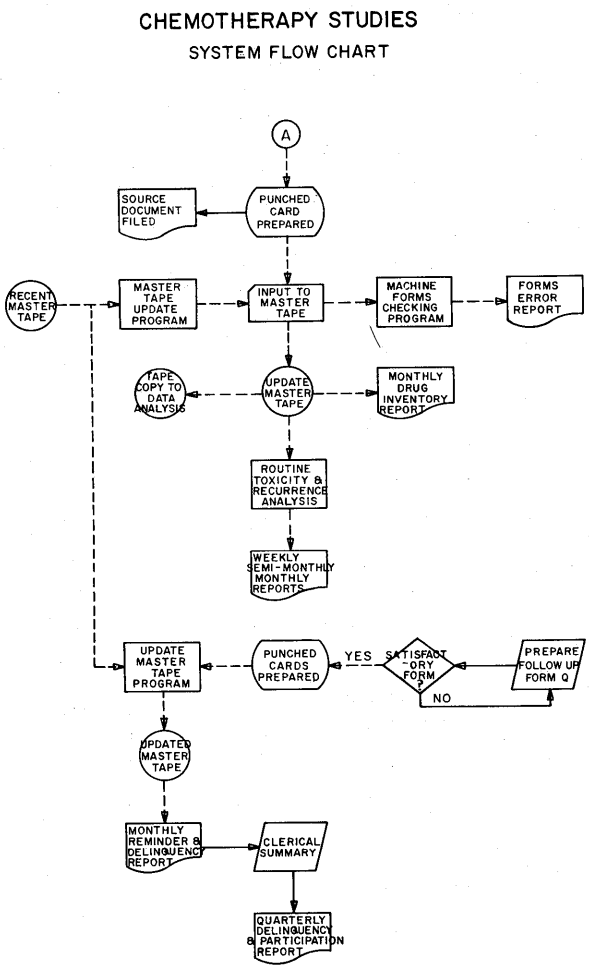



Figure 2. Chemotherapy studies — system flow chart.

# A FACILITY FOR EXPERIMENTATION IN MAN-MACHINE INTERACTION*

W. W. Lichtenberger
*Computer Center and Department of Electrical Engineering*
*University of California, Berkeley*

and

M. W. Pirtle
*Computer Center*
*University of California, Berkeley*

## INTRODUCTION

The broad objective of the project of which the work reported below is a part is to explore and develop techniques in man-machine interaction. The situation involving a person interacting with a machine in the performance of a task generally requires that the person be on-line with the machine. The amount of machine time wasted while the person is carrying out his part of a task may require, even in an experimental situation, a time-sharing system.

### *The Role of Time-Sharing in the Project*

The time-sharing facility described below was constructed

(a) to develop and test some ideas in time-sharing a digital computer and
(b) to develop a useful facility for a variety of experimenters in man-machine interactive areas.

It should be emphasized that the time-sharing system, although general in nature, is an experimental system intended to give great flexibility and fast response to a limited number of users. (In particular, it is not designed to serve a large number of users over a broad spectrum of problems as a utility approach[1,2] to time-sharing.)

### *Design Philosophy of the Time-Sharing System*

Because of the variety of tasks to be performed by the users of the system it was felt that each user should be given, in effect, a machine of his own with all the flexibility, but onerousness, inherent in a "bare" machine. It was also felt that additional features should be provided to enable the user to reduce the onerousness, perhaps at the cost of flexibility, to the extent desired. Thus each user is given a "copy" of a slightly modified SDS 930 with 16K of fast memory. This "copy" differs from the normal Scientific Data Systems (SDS) 930 only in (1) the obvious impairment of certain real-time capabilities which result from the necessity of running programs for short, nonregular intervals, (2) the substitution of a set of instructions which initiate system-controlled input/output for the standard I/O instructions, and (3) the addition of many new (software-interpreted) instructions along with various system routines and a number of large-scale subsystems.

In order to cut down response time it was felt desirable to have more than one program present in memory and to swap with auxiliary memory only

*The work reported in this paper was supported by the Advanced Research Projects Agency, Department of Defense, under Contract SD-185.

when necessary. In practice, swapping occurs relatively frequently.

For further economy of both active and auxiliary storage, it was felt desirable to provide for *common programs* — single copies of programs shared by more than one user. Common programs are pure procedures with a unique copy of temporary storage assigned to each user. Many system routines are written as common programs, as are some large-scale systems.

The major part of the system executive, for example, is a common program. Since this part of the executive was written as it were dealing with one user it was simpler to write and to debug. Furthermore, the same part of the executive would require no changes if more central processors were added to the system. Simplicity, small size, and flexibility were among the goals of the system executive, and all of these goals have been to some degree achieved.

The project objectives made it desirable to base all input/output around the remote consoles as much as possible and to minimize the role of more standard I/O equipment (cards, line printers, magnetic tapes, etc.). The user is given mass storage in the form of either word-addressable or sequential files and a generalized file-handling capability. Files are independent of any peripheral I/O device or storage medium and are addressed homogeneously regardless of their current position in the storage hierarchy. The file-handling facilities augmented by comprehensive editing programs provide the users at remote consoles the ability to manipulate information conveniently within the system.

## SYSTEM DESCRIPTION

### *Local Units*

As shown in Fig. 1, the system is built around a modified SDS 930 central processor[3] and a main memory consisting of two 16K modules of core storage. The main memory is augmented by a large capacity drum which is in turn augmented (it is anticipated) by a mass storage unit. Filling out the list of local components are the teletype, multiplexor, the I/O processor, a 45KC magnetic tape unit, and a 200-cpm card reader.

A cursory description of the SDS 930 central processor and its modifications for time-sharing is the subject of several of the following sections of this paper; therefore, the various memory devices will be given first attention.

The main memory consists of two modules of 16,384 words. The words are 25 bits in length (including a parity bit), and the memory cycle time is approximately 1.9 microseconds. Each of the modules is connected to three memory buses. These buses have fixed priorities, with the drum I/O processor, the general I/O processor, and the CPU connected to buses having progressively lower priority. To accommodate the high data transfer rate of the drum ($525 \times 10^3$ words per second), the timing for the main memory units and for the CPU are derived from a timing track on the drum, and the memory addresses are interleaved between the two modules. By having the drum I/O processor reference the modules alternately, the CPU can operate at approximatly 65 percent capacity during drum transfer operations. This assures that interrupt processing capability is preserved and that a significant amount of processing accompany the data transfer operations.

The next level of storage is in the form of a magnetic drum* having a capacity of 1,376,256 words and a data transfer rate of approximately $525 \times 10^3$ words per second. The drum is word-addressable to facilitate handling files and has a storage format commensurate with its function of swapping programs, or parts thereof, between the drum and the main store. This format provides 84 bands of 16K words, with each band divided into 8 segments of 2K. Each of the segments is separated by a gap of sufficient length to allow the drum I/O processor to accept an instruction between segments. This feature facilitates the scatter reading and writing necessitated by the memory page technique employed (cf. Memory Relabeling and Protection, below).

The next level of storage will be in the form of a mass storage unit having a capacity in excess of $10^8$ words and an access time not greater than 0.5 second. This unit will have some type of interchangeable cartridge, thus providing yet another level of storage having a still greater capacity and access time.

The teletype multiplexor consists of 16 input and 16 output buffers along with control logic to notify the computer of buffer conditions requiring service. The general I/O processor is another device having

*The drum is being produced to local specifications[4] by Vermont Research, North Springfield, Vermont.

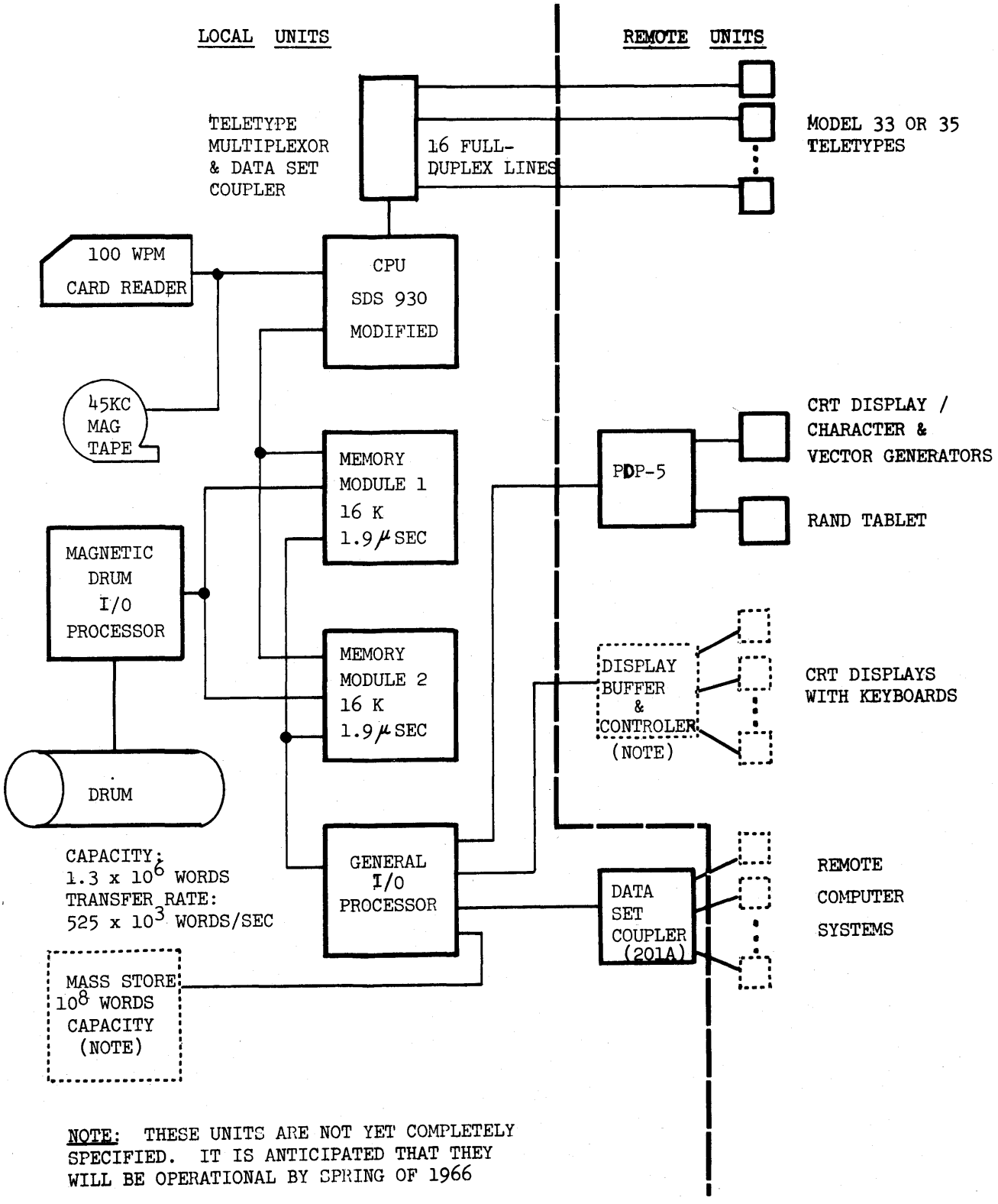



Figure 1. Configuration of equipment.

very little complexity. It is made up of a central control channel and several independent subchannels. The subchannels operate concurrently and may retain their requisite information (word count and current address) internally or in main memory.

The final two local units, the 45KC magnetic tape unit and the 200-cpm card reader are used by the occasional user who desires to transmit data between this and some other system via one of the media processed by these units. In addition, magnetic tape is being used temporarily as a secondary storage medium. This function, of course, will be assumed by the mass store upon its acquisition.

### *Remote Units*

The remote units include 10 model 33 and model 35 teletypes, 2 different types of CRT display-keyboard units, and a PDP-5 with a CRT display and Rand Tablet.

The teletypes are operated in the full-duplex mode with each character being individually processed by the CPU. That is, the teletype keyboard and printer are treated as independent units by the system I/O programs, and, for further flexibility, the input characters to the CPU are processed on a character-by-character basis rather than on a message basis. This procedure consumes processor time (approximately 300 microseconds to input and echo a character, and 200 microseconds to output a character), but experience indicates that the capabilities thus obtained justify the processing expenditure.*

In fact, the full-duplex, character-by-character I/O philosophy will probably be carried over to the alphanumeric CRT display-keyboard stations. These stations will consist of CRT display units driven by a central buffer and control unit located in the vicinity of the stations and keyboards which will communicate with the CPU via the central control unit.

The second type of CRT display-keyboard stations will be similar to the Culler-Fried console.[5] These consoles employ a storage tube display with a script generator, i.e., a generator which produces short vectors of length and angle specified by the input character.

For research efforts requiring a more capable CRT display system, a remote unit is provided which consists of (1) a PDP-5 processor with 4K words of memory, (2) a CRT-display unit[6] with character, vector, and script generators,* and (3) a RAND tablet.[7] These three major components are integrated into a unit which provides the researcher with a large amount of flexibility with regard to both the use of the present system and the addition of supplemental equipment to provide still greater capabilities.

In the present system, the PDP-5 functions as a buffer and controller for the CRT display and the RAND Tablet and performs some elementary operations such as smoothing the data input from the Tablet. All computations are performed in the central computer.

## FEATURES OF THE MODIFIED 930 USED FOR MULTIPROGRAMMING

### *Modes*

The role of the system monitor is unique among programs residing in the machine. Reflecting this fact, the 930 has been modified to operate either in monitor or in user mode. Monitor mode permits the monitor the use of privileged instructions and unrestricted memory. The function of user mode is the subject of the following sections.

### *Protection of the System from User Action*

It is necessary to protect the system and all other users from certain actions of any user. Such actions include the execution of instructions which: (a) affect peripheral equipment, (b) halt computation, (c) interfere with rapid response to interrupt requests, or (d) access unassigned memory locations.

User actions of type (a) and (b) are handled merely by trapping the offending instruction (the term *trap* means an interrupt of highest priority). Instructions which fall into this category are called *privileged instructions*. When the 930 is in user mode an attempt by a user to execute a privileged instruction will result in the execution of a no-op followed by a transition to monitor mode and a transfer of control to a memory location unique to the illegal instruction trap.

*It is calculated that 16 teletypes executing independent input and output simultaneously at full speed consume 8 percent of processor time.

*The display is being produced to local specifications by the Burroughs Corporation, Ann Arbor Laboratory, Ann Arbor, Mich.

Type (c) actions are treated by permitting interrupt requests to preempt execute and indirect address operations. If either of these operations are in process when an interrupt request occurs, the operation is aborted and the interrupt request acknowledged. Upon return to the interrupted program the aborted intruction is begun anew. In this way, infinite indirect address and execute loops in a user's program cannot halt the system.

The solution to user actions of type (d) is related to memory relabeling and is discussed in the following section.

*Memory Relabeling and Protection*

The memory relabeling or paging technique[8] adopted provides both for dynamic program relocation and memory protection with no increase in memory access time. The technique was adopted initially because it eliminates the need to move information around within the fast memory in order to provide space for incoming programs and because it easily provides memory protection. Other important uses (discussed later) became apparent as the system progressed. The implementation consists of 8 relabeling registers of 6 bits each laid out in 2 registers as shown in Fig. 2.

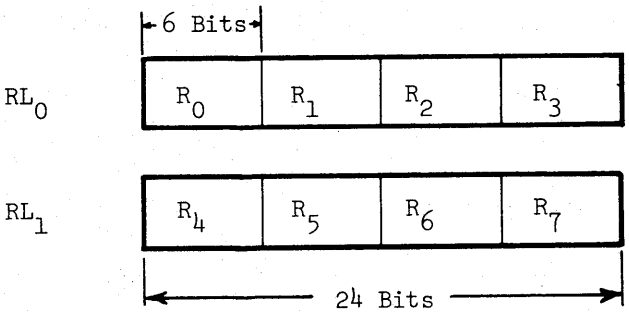



Figure 2. Physical arrangement of relabeling registers.

For purposes of relabeling, the memory is divided into 16 pages or blocks. Calls are addressed by block number and location within a block as specified by subfields of the address.

The block or page size is fixed at 2K by the 11 bits of the least significant part of the address (which may be thought of as an address within a page or a *page address*). Because the address field of the 930 contains 14 bits, only 16K or 8 pages are permitted each user. The upper 3 bits of the address constitute the *page number*. Relabeling hardware replaces the user's page number $i$ with an actual page number $R_i$ which may be different from time to time as the program is moved in memory (cf. Fig. 3). Because of the spatial relationship of the page number and page address, the user is not conscious of page structure. Note that relabeling permits user's storage to be located in noncontiguous blocks while appearing to the user and to the machine to be connected.

Of the 6 bits in a relabeling register, the lower 5 are used for the actual page numbers. Addresses after relabeling are therefore of length 16 bits, permitting as much as 64K of fast memory in the system. The sixth bit in a relabeling register designates a read-only block. The facility to have read-only storage enables users to share subsystems directly without interference and without the necessity of calling the monitor constantly to change relabeling.

Absolute memory protection (i.e., protection against any reference) is accomplished by using $R_i = 0$ to mean that no memory is assigned to the page $i$. Any reference to a cell whose relabeling register contains zero is trapped.

Figure 4 shows a 6K memory allotment distributed in 2K blocks at 24000, 64000, and 14000. The block at 14000 is read-only. It may be seen that references to any location greater than 13777 will point to one of the relabeling registers 3 through 7, causing an out-of-bounds trap. The choice of the combination $R_i=0$ prevents absolute memory locations 00000 through 03777 from being used for user programs, but this is of no consequence since that area is part of the monitor. Note that the user may transfer control, for example, to his locations 10000 through 13777, but an attempt to store information there will cause a trap.

Relabeling is always performed in user mode. It is also possible to invoke relabeling for individual instructions in monitor mode. In accessing memory to obtain the effective address of an instruction, any word encountered with Bit 0 set causes relabeling to apply immediately and for the duration of the instruction. Thus an instruction with Bit 0 set causes relabeling of its address, while an instruction with a chain of indirect addresses produces relabeling at the first instance of Bit 0 set. In the latter case subsequent references come from relabeling memory.

*Mode Transitions*

During the design of the modifications to the 930 it was felt desirable to make the transitions between modes as simple and as natural as possible. In par-

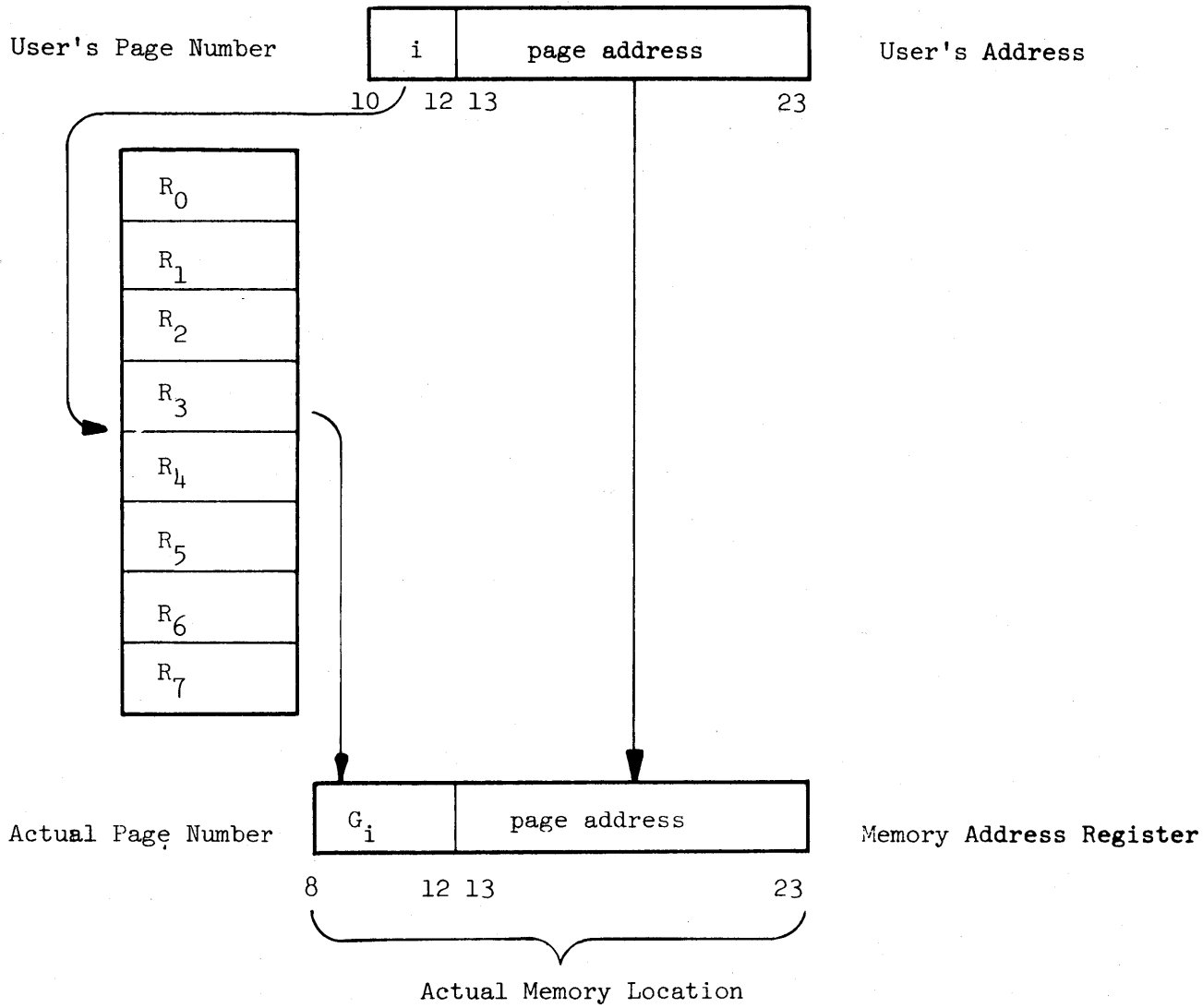



Figure 3. Action of relabeling.

ticular, it was felt that there should be provided sufficient hardware capability to insure that the interrupt routines could be independent of the mode of the machine at the time of the interrupt and that the system routines explicitly called by the various programs should not require *software interpretation* of

- the source of the calling program (monitor or user),
- the location of the call,
- the location of the arguments, and
- the specific action requested.

Readers who have some experience in implementing high-speed interrupt or I/O routines will perhaps appreciate the spirit of the above objectives.

Transitions from user to monitor mode occur only upon (1) an interrupt or trap, or (2) execution of a system programmed operator (cf. the following section). The user has no interest or direct concern over category (1) and does not think explicitly of the instructions in category (2) as changes of mode.

The system initiates the transition from monitor to user mode by transferring control to a user program. Specifically, a control transfer calling for relabeling causes a transition to user mode.

In order to provide closure in the above scheme, the previous mode of the machine is stored as a single bit in the subroutine link of both interrupt and system programmed operator routines. The bit used is the same as that used to designate relabel-

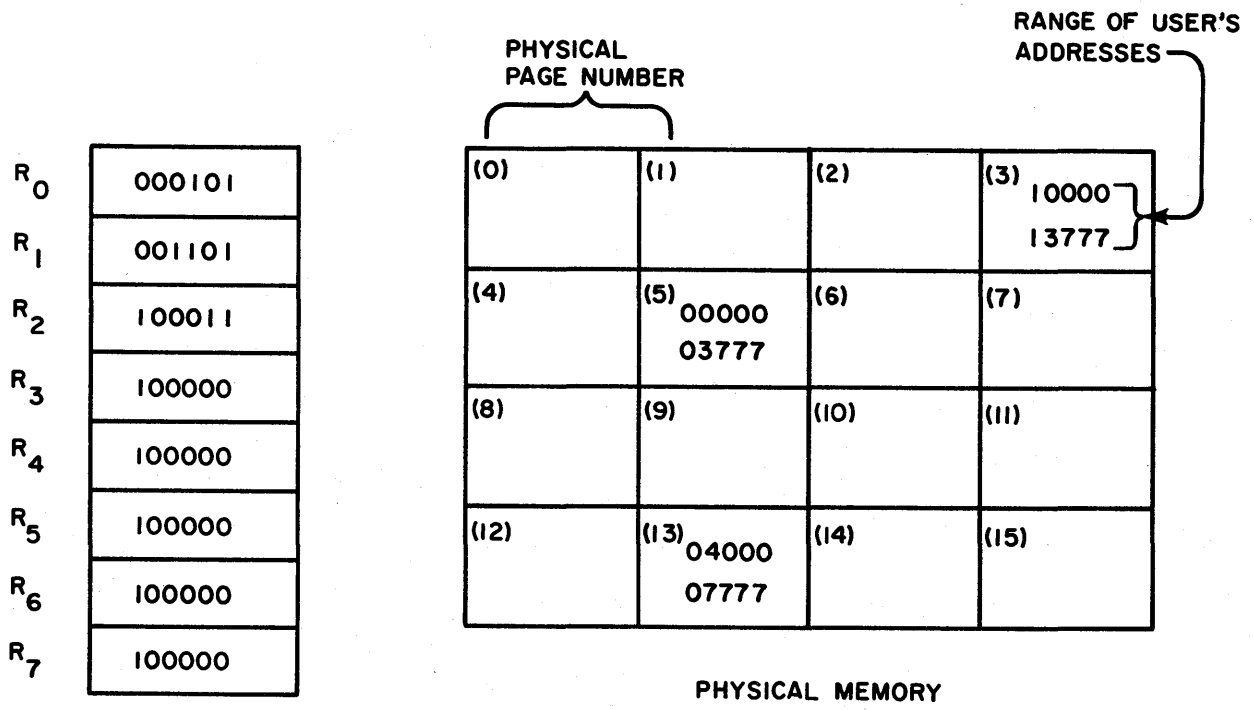



Figure 4. Example of a 6K memory allocation.

ing. Thus at the end of the routine, when the return instruction is executed, the mode will automatically revert. If arguments are accessed through the link (cf. the following section), relabeling is or is not applied, depending on the mode storage bit.

Table 1 summarizes the primary functions of the modes.

*System Programmed Operators*

Input/output instructions are among the privileged instructions not allowed in the user's machine. The system must do all I/O for the user, and he must therefore be able to call the system for such services. Also the system executive requires many complex services, some of which are potentially useful to a user. Such services should be provided by system calls. The *system programmed operator* (SYSPOP) is the device by which such calls are accomplished.

The SYSPOP is an extension of a normal 930 feature—the *programmed operator* (POP). POP's are invoked by setting a bit in the instruction word, and they function as a special kind of subroutine call. In the execution of a POP the op code bits are not decoded in the usual way. Instead, they are taken to be the relative address in a transfer vector beginning at $0100_8$ to which control is transferred. At the same time the contents of the program counter and status of the overflow indicator are stored as a subroutine link in location 00000. The indirect address bit of this link is set as well. Single arguments

Table 1. Summary of 930 Modes and Their Effects.

| Monitor Mode | User Mode |
|---|---|
| *All* 930 instructions may be executed. | All 930 instructions *except* privileged instructions (I/O, halts, etc.) may be executed. |
| Normal addressing applies to references without Bit 0. References with Bit 0 set call for relabeling. | Relabeling applies to all memory references. |
| Transition to user mode occurs upon executing a transfer whose effective address is relabeled (i.e., Bit 0 is set). | Transition to routine mode occurs on all interrupts or executions of SYSPOP's. Bit 0 is set at the formation of the first subsequent subroutine link so that memory access to data will be relabeled and so that control will revert to user mode. |

or the location of a list of arguments can thus be transmitted to the body of the POP indirectly through the link in 00000. The format of a POP is the same as that of a normal routine instruction, hence the POP is a convenient way of simulating nonexistent machine instructions.

The 930 was modified so that a POP executed in user mode with Bit 0 set causes transition to monitor mode. The user thus has the facility to jump to a standard transfer vector in the system. Note that the user may still implement his own POP's. The SYSPOP's, however, give the user 64 new "machine instructions" which do not require his memory allocation or other attention.

The reader should note that by having the mode stored in the relabeling bit of the link, all four objectives for system routines listed in the preceding section are accomplished; modes are completely invisible to interrupt and system programmed operator routines. Most importantly, interrupt routines *take no more time* and in fact are no different from similar routines in a non-time-sharing system. Furthermore, the overhead associated with calls to the system (SYSPOP's) is only 4 memory cycles.

## GENERAL DISCUSSION OF SYSTEM FEATURES

The features of the system described above came into being through a compromise between that which is desirable and that which is feasible, to implement time-sharing on a machine basically not designed for time-sharing. Some of the features have been shown to be surprisingly compact and effective, however. For example, the SYSPOP provides a simple but versatile system call. Also, relabeling is not only useful for dynamic storage allocation but provides the basic means by which common programs can be constructed.

### *Method of Writing Common (Re-Entrant) Routines*

By its very nature, a common routine consists of (1) *a pure procedure*—a body of code which is not self-modifying and in which there is no temporary storage—and (2) one or more copies of all temporary storage associated with the routine. To implement a common routine, one allocates all temporary storage—the *data block*—to a unique block or blocks of memory different from those blocks of the procedure body. Because the procedure is pure the *state* of a computation at any time is determined by the contents of the data block and the active registers. Thus to interrupt computation for one user and continue computation for another is merely a matter of saving and restoring active registers and changing relabeling for the data block.

The only programming conventions which must be followed in writing the procedure body are those of avoiding self-modification. Avoiding direct self-modification is especially easy in a machine like the 930 which permits combinations of indexing and indirect addressing and which has an executive instruction. The programmer simply avoids storing information within the program. Assemblers

can check programs for such storage automatically.

In addition to the constraints mentioned above, the programmer must avoid the use of the normal SDS 930 subroutine transfer (BRM) since it stores the subroutine link at the head of the subroutine and thus within the procedure body. At the moment, a SYSPOP is provided to steer the link storage indirectly one level into the data block. This is done by placing the address of the subroutine link at the head of the subroutine, where the link itself would normally reside. Thus the SYSPOP SBRM $y$ at location $p$ first looks in $y$ to find a link address $z$. The value $p$ is then stored in $z$, which is outside of the pure procedure. Control is transferred to $y + 1$. Return is accomplished with the normal subroutine return instruction using indirect addressing (BRR*$y$). The indirect address sequence causes the machine to look first in $y$ to find $z$ and then in $z$ to find $p$. Control goes to $p + 1$. It is anticipated that SBRM will be incorporated as a new 930 instruction.

In cases of the occurrence of POP's within a program, it is usually desirable to have the data block at least start within the user's block 0. The use of a POP places a return link in the user's location 0, and this, of course, must be in the data block.

It should be noted that the utility of relabeling in implementing common routines was not fully realized at the outset. It should also be observed that using relabeling for such purposes restricts the common routines virtually to subsystems (compilers, debuggers, interpreters, etc.) since an entire page is reserved for temporary storage. Routines of this magnitude, however, do have the most need to be single-copy.

In our system many functions which might otherwise force users to have copies of the same little routines in their programs are taken over by SYSPOP's. Finally, for those routines that lie halfway between (e.g., packages of mathematical subroutines—SIN, COS, etc.) the read-only facility allows users to share procedure storage. The only abridgment of the users' freedom here is that such routines must be located absolutely in user memory so that they may address themselves properly without asking the system to change relabeling upon entry and exit.

### *The Structure of SYSPOP's*

The SYSPOP mechanism is basic to the overall system and is used extensively by programs at all levels. SYSPOP routines run in monitor modes. If called by a user the execution of a SYSPOP is part of his program, and this is the *only* instance of a user-controlled program running in monitor mode. The absence of normal protections during such intervals imposes constraints on the program structure of SYSPOP's as follows:

1. SYSPOP's must be written so as not to cause disaster if erroneously called. This feature calls for a certain amount of software interpretation, but it is on a different level from the interpretation spoken of in the section on Mode Transitions.
2. SYSPOP's are normally small and of short duration. Because they share the same link it is difficult to make SYSPOP's re-entrant without time-consuming maneuvers. The SYSPOP's are therefore not re-entrant and contain their own temporary storage.
3. Since SYSPOP's are not re-entrant and since they are shared by all users and by most parts of the system itself, program interruption is handled by allowing a SYSPOP in process to go to completion. This is done by having all SYSPOP's return control through a common (1-instruction) routine.

The reader should note the merit of the mode-changing scheme discussed in the section on Mode Transitions, as reflected in resulting simplicities in SYSPOP's. Recall that the mode of the calling program is stored in Bit 0 of the link, and that the SYSPOP accesses the calling parameters indirectly through the link. If a user calls a SYSPOP, relabeling will be applied to accesses of calling parameters, otherwise not. When returning control the SYSPOP executes a return instruction. If Bit 0 of the link is set, relabeling is applied and the mode is set back to users'. Thus modes are completely invisible to SYSPOP's.

## SUMMARY

The project goals discussed at the beginning of this paper have been set. The time-sharing system involving memory relabeling, common routines, and duplex teletype operation has been in operation since April, 1965. The system is highly flexible and

can provide, for users who require it, a response time of less than one second.

It should be noted that memory relabeling* is accomplished with no increase in access time. The number of processor modes is small (two), and mode transitions are done in such a way as to enable interrupt and user-called system routines to be independent of mode.

The user machine is clean and well defined. Input/output is simpler, more foolproof, and device-independent. The user is given a variety of other services ranging from generalized file-handling capability to string processing to assemblers, compilers, debuggers, and editors.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the direction and valuable advice of Professors Harry D. Huskey and David C. Evans, co-principal investigators. Mr. W. J. Sanders made many valuable contributions to the early system design and SDS 930 modifications. The time-sharing executive system[9] and most of the subsystems were written by Mr. Peter Deutsch and Mr. Butler Lampson, who suffered through seasons of balky, fidgety hardware and primitive input/output to produce an excellent result.

## REFERENCES

1. R. M. Fano, "The MAC System: The Computer Utility Approach," *IEEE Spectrum,* vol. 2, no. 1, pp. 56-64 (Jan. 1965).

*The technique of relabeling was developed by M. Pirtle in April 1964 and was implemented in two weeks on the 930 upon delivery the following November.

2. J. I. Schwartz, "A General Purpose Time-Sharing System," Proc. AFIPS Conf., 1964, vol. 25, pp. 397-411.

3. W. W. Lichtenberger, M. W. Pirtle and W. J. Sanders, "Modifications to the SDS 930 Computer for the Implementation of Time-Sharing," Document no. 20.10.10, Project GENIE, University of California, Berkeley (Jan. 1965).

4. M. C. Hurley, "Drum Storage System Preliminary Reference Manual," Document no. 20.70.20, Project GENIE, University of California, Berkeley (Mar. 1965).

5. G. J. Cullen et al, "TWR Two-Station On-Line Scientific Computer," vols. II and IV, TWR Space Technology Laboratory (July 1964).

6. G. D. Hornbuckle, "Display System Specifications," Document no. 20.60.10, Project GENIE, University of California, Berkeley (Jan. 1965).

7. M. R. Davis and T. O. Ellis, "The RAND Tablet: A Man-Machine Graphical Communication Device," *Proc. AFIPS Conf.,* 1964, vol. 26, part I, pp. 325-331.

8. J. B. Dennis, "Program Structure in a Multi-Access Computer," Technical Report MAC-TR-11, Project MAC, Massachusetts Institute of Technology (1964).

9. L. P. Deutsch and B. W. Lampson, "SDS 930 Time-sharing System Preliminary Reference Manual," Document no. 30.10.10, Project GENIE, University of California, Berkeley (Apr. 1965).

# A TIME- AND MEMORY-SHARING EXECUTIVE PROGRAM FOR QUICK-RESPONSE ON-LINE APPLICATIONS

James W. Forgie
*Lincoln Laboratory**
*Massachusetts Institute of Technology*
*Lexington, Massachusetts*

## INTRODUCTION

APEX is an experimental operation-oriented on-line data analysis system being developed for the TX-2 Computer at MIT Lincoln Laboratory. This paper describes the executive program which has been designed to satisfy the needs of that system as well as the other activities currently soaking up the computational energies of TX-2. These needs are developed into a set of requirements for a fast-response time-sharing system. The requirements, in turn, lead to a series of design decisions which involve both the hardware and software parts of the system. A memory and program-sharing system, with the hardware to make such a system efficient, takes the form of a complex of apparent computers, one for each console, which share some common hardware (TX-2). The salient characteristics of these computers are described as well as the executive program structure which gives them apparent reality.

## BACKGROUND

The TX-2 Computer, an experimental facility at MIT Lincoln Laboratory has been in operation since 1960. Never a service facility, the computer has been used principally in a number of long-term research projects which have taken advantage of the special input/output capabilities and direct accessibility of the machine. These projects have included such areas as graphics, waveform processing, and pattern recognition. Most of the work on the computer has involved real-time inputs, interaction with output displays, or both. The computer has always been used as an on-line facility with the bulk of the computer time being used in sessions of several hours duration. Programming has been carried out in machine language augmented in the past by a number of personal macro languages and recently by a more general macro language for list processing (CORAL). An on-line macro assembler, MK 4, has been used both as an assembly program and an on-line operating system by most users. In the fall of 1963 it was decided to realize on TX-2 on experimental operation-oriented on-line system in order to study man-machine interaction in problem solving. This system (APEX) is designed to allow the scientist or engineer to make use of the computer throughout his work on a data analysis problem without having to be concerned with many of the details ordinarily involved in programming a computer. The system is

*Operated with support from the U. S. Air Force.

based on a library of computational and display routines which may be called directly by the user in an appropriate problem-oriented language. For a problem area in which library routines exist, it is expected that individual library routines or short combinations of routines will suffice for a high percentage of the total operations needed. In order to handle the few remaining cases, it is desirable to have within the system special and/or general compilers which the user can utilize to create the occasional pieces of program which he may need to complete the solution to his problem. It is likely that a person using a system of this sort will probably spend much more time looking at his displays and thinking about what to do next, than he will spend actually doing computations. Economics then dictate that multiple consoles should be provided for the computer, and the computer facilities shared among these consoles. An executive program which will handle such sharing of the computer facilities and related problems of storage allocation and communication becomes desirable to complete the system.

In January of 1964 a firm commitment was made to undertake the realization of such an operation-oriented on-line system on TX-2. Since experimentation with the new system would put pressure on the already full schedule of TX-2, a further requirement was placed on the design of the new system, namely, that its executive program should allow for the use of TX-2 in something approaching its accustomed style at the same time that the system was running. Thus the advantages of time-sharing could be made available to the existing users of the machine. This paper discusses the design of the executive system which has grown out of those decisions.

## SYSTEM REQUIREMENTS

The design requirements for the APEX executive system may be briefly stated as follows:

1. *Time-Sharing.* The system should provide for time-sharing essential computing facilities among a small number of consoles (perhaps half a dozen), most of which would have oscilloscope displays as well as the usual keyboard and typewriter.

2. *Fast Response.* The on-going activities in graphics, waveform processing, and pattern recognition all involved the use of interactive displays. It appeared that response times in excess of one second would noticeably degrade the performance of already existing programs in these areas. In addition, the proposed experiments with the operation-oriented on-line system called for the ability to degrade response time in order to measure its effect on the user. Thus, all proposed applications of the system called for fast response under most circumstances.

3. *Retention of Results.* The executive should assume responsibility for the implicit retention of all program and data files whose destruction was not specifically ordered by the user or his program.

4. *Subroutine Autonomy.* The executive should allow any program, written as a closed subroutine and following certain conventions, to be run as an independent progam making full use of core storage addresses and index registers. Routines to be run in this fashion should be precompiled and stored in absolute binary form. They should be completely independent of the routines which call them and may thus be called recursively. The executive should provide isolation and protection for such routines and facilitate the passing of parameters to them. This requirement for subroutine autonomy was intended to allow both fast operation of library routines by eliminating compilation or relocation time and relative simplicity of programming by minimizing the number of conventions such routines must follow.

5. *Flexible Input/Output Services.* The executive should handle the details of all input/output operations. It should provide continuity for displays and keyboard-typewriter conversation. It should provide for sharing of common I/O devices such as printers and magnetic tape. Insofar as possible, it should leave formats and the interpretation of data to user programs.

## DESIGN DECISIONS

Early in the design phase of the executive program a number of policy decisions were made which had considerable effect on the character of the final program. Among these were the following:

1. *Memory-Sharing.* It appeared from the outset that if the requirements for fast response were to be met, it would be necessary to keep some part of each user's program in core at all times. The size of the TX-2 memory (97K) made this feasible for a

small number of users. In order to facilitate memory-sharing it was decided to add relocation and bounding hardware to the computer and to provide some executive services which would make it easier for programmers to break large program structures into pieces of manageable size.

2. *Program-Sharing.* If memory sharing was to operate efficiently, it was obvious that large public routines such as compilers should be written in re-entrant form, so that they could be shared by all current users. The TX-2 order code allows this kind of program to be written without any special difficulty. It was decided to add certain features to the executive to facilitate the operation of re-entrant programs and to add the necessary hardware to protect them.

3. *The executive should simulate an apparent computer for each console.* The requirements of the operation-oriented on-line system could have been met by a highly specialized executive program, but such a design would not have satisfied the needs of the on-going research projects already using TX-2. Their needs would, perhaps, have best been served by a time-sharing system which provided the entire facilities of the computer for each user in turn. The multiple sequence design of the TX-2 input/output system made the realization of the latter design appear unreasonably complex. The realization of a simulated computer similar, but not identical, to TX-2 seemed a reasonable compromise between these two requirements.

4. *There should be no direct communication between the user and the executive.* All user commands should be passed through the executive to programs operating within the simulated computer for that console. Communications involving the executive are then passed back from such a program to the executive. It appeared both unnecessary and undesirable to tie the system to any language conventions by building these into the executive.

5. *Insofar as possible, software features would be realized in programs operating in the simulated computers.* This decision allowed the executive to be as simple as possible and permitted expansion of the overall software structure without having to modify the executive program.

6. *Compatibility between former TX-2 programs and programs which would operate within the simulated computer was not to be a requirement.* The design of the simulated computer should be made to correspond to TX-2 whenever possible and reasonable. But it was expected that some change would have to be made in all programs to accommodate the different input/output characteristics of the executive and to take advantage of the storage allocation facilities provided by the executive.

7. *Hardware changes to the TX-2 system were to be considered as legitimate variables in the design work.* The computer engineering group was prepared to make reasonable and compatible modifications to the computer when such changes appeared to be the desirable and economical solution to the design problem. Throughout the development period of the executive program there was considerable interaction between hardware and software designs and designers, and major changes have been made in the TX-2 computer to facilitate the APEX executive system. These include the addition of a file memory (a UNIVAC Fastrand Drum), hardware to trap the attempted execution of privileged instructions, and four memory-snatch channels to increase the efficiency of high speed I/O operations. The most significant change was the addition of a hardware system called SPAT (an acronym for Symbolic Page Address Transformation). SPAT, which has been in operation since January 1964, utilizes a 1024-word thin film memory and high-speed transistor circuitry to realize a 3-level address transformation within a single TX-2 clock pulse time (0.4 microseconds). This transformation makes available to the executive the advantages of paging, segmentation, and complete memory protection. It greatly reduces the overhead involved in memory- and program-sharing.

## CHARACTERISTICS OF THE APPARENT COMPUTER

The APEX Executive Program simulates an apparent computer for each console. These apparent computers may be viewed as being somewhat restricted replicas of TX-2 augmented by features provided through the executive program. The core storage for each apparent computer is bounded and segmented and limited in total extent to approximately two-thirds of the TX-2 core capacity. The order code for the apparent computer is that obtained by eliminating input/output and multiple sequencing instructions from the TX-2 order code, and then adding some executive calls to handle input/output, file maintenance, and storage allocation in the apparent computer. The number of

index registers is reduced to 15, and some restrictions are placed on the choice of machine configurations available. The apparent computer is a single-sequence (program counter) computer in the current version of the system, but the hardware allows for future expansion to three sequences. In general, programs written for TX-2 will not operate in the apparent computer, and vice versa. However, programs which do not involve I/O operations may often be transferred with little or no change.

The storage structure of the apparent computer takes advantage of the SPAT address transformation hardware in TX-2. The SPAT hardware breaks up core storage into pages of 256 registers. These are organized into books (segments) of up to 32 pages (8,192 registers). The 17-bit addressing capability of TX-2 allows 16 such books to be selected by the 4 highest order address bits. Since the apparent addressing capability of the machine exceeds the real core (currently 97K), some of the books must always be incomplete or empty. In the apparent console computers realized by the APEX executive program, the user's programs and data are organized into files. A file is a contiguous group of registers which must be some integral number of pages in length. A file always has a name which is known to the APEX file directory. Files may exceed one book in length, but they must begin at the start of the book, and no more than one file may occupy any one book. Executive calls in the user's program control which files are to appear in core at any one time. A file may be set up in a book specified by the directory, as is usually the case for program files, or it may be set up in an arbitrary book according to the requirements of a program which is to process it. All files begin as working storage files with ephemeral names. When a program has placed information in such a file, an executive call may be given to assign a permanent name to the file. The naming call may also specify that the file is to be a read-only file in future appearances in core, that it may be operated as a program file, and that it must be set up in a particular book when called as a program. After the file has been given a permanent name, it will remain in the file memory until it has been discarded by a call from the user's program. Thus, all data files which the user has had occasion to name will be retained from one session to the next. We refer to the complex of files set up in the user's apparent memory at any one time as a MAP. A MAP may be thought of pictorially as we see in Fig. 1 for a typical setup of matrix routines. However, a MAP may be equally well described as simply a list of names of files together with the book numbers in which each is to appear in the MAP. In Fig. 1, the dashed lines indicate the potential capacity of each book, while the solid lines indicate the actual core occupied by the file named within the block. More will be said about the contents of this MAP after a short discussion of the way the APEX system handles library routines.

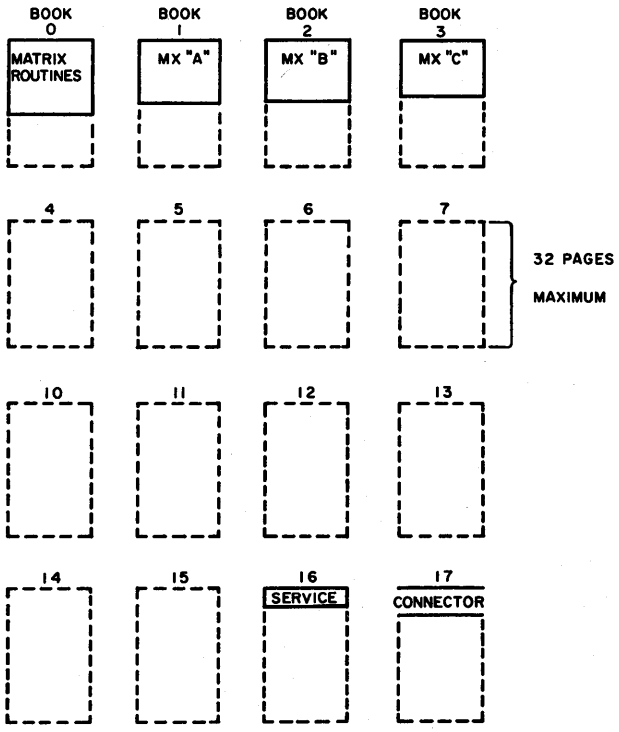



Figure 1. An APEX core MAP for the matrix operation AxB-C.

One of the principal design requirements for the APEX system was to provide a means whereby library routines (or arbitrary user subroutines) could be called into core and operated without any conflict between the core addressing requirements of the routine and the program which called it. This requirement is met by providing a fresh MAP for the called routine. When a program wishes to call a library routine to be operated in a new MAP, it does so by issuing a GO UP call to the executive, passing along the name of the library routine as a parameter of the call. This process is called "going up" because the new MAP is thought of as being

placed on top of the MAP which contained the calling program. Since normal operation of the APEX system involves multiple MAPS, the typical situation at some instant of time will be a stack of MAPS such as illustrated in Fig. 2. This figure

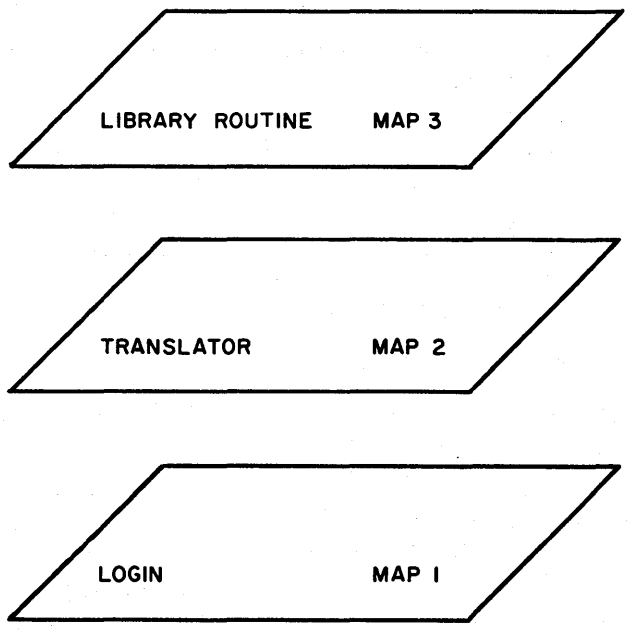



Figure 2. Typical MAP stack for a simple library routine operation.

shows the stack which will result in the simple case of a user having logged-in and used a translator to call a library routine. At the instant of time represented by Fig. 2, the library routine is presumed to be in operation. When it finishes, the stack will return to two MAPS in depth as control returns to the translator.

Let us look in a little more detail at the operation of going up to a new MAP. The GO UP call will cause the executive to produce an entirely new MAP containing the library routine, which it sets up according to the directory specifications for the routine. In addition to the library routine, the new MAP will contain two other standard files. One, called the CONNECTOR file, is common to all MAPS and is used to pass parameter information and to provide small amounts of working storage for the library routines. The first register of the CONNECTOR file is used to indicate the first free register in the file. The contents of this register are noted by the executive on going up and are restored on "peeling back," as the operation of returning to a lower MAP is called. The other standard file, called the SERVICE file, contains a number of often needed small routines such as those for floating-point arithmetic and the manipulation of calling parameters. As shown in Fig. 1, these two standard files are located in the two highest numbered books in the MAP. If, as is usually the case, parameters are to be passed to the library routine, they are placed in the CONNECTOR file in a standard format by the calling program. The location of these parameters is passed to the executive as a second parameter on the GO UP call. The executive then writes a PEEL BACK call into the CONNECTOR file at the proper location and passes control to the library routine as if it had been entered by a standard subroutine call instruction. The library routine then finds its parameters in a standard calling-sequence relationship to the index register contents which specify its return point. It inspects the parameters and calls such other files as it may need to carry out its mission. Upon completion of its operations, it makes a standard subroutine exit, which transfers control to the PEEL BACK call which the executive had placed in the CONNECTOR file. The PEEL BACK call causes the executive to discard the MAP containing the library routine and return control to the calling program. This procedure assumes that the output produced by the library routine has either been placed in a file whose name was supplied in its calling parameters, or that its output was placed in the connector file at a location which was supplied in the calling parameters.

The way in which the APEX system handles library routines has a number of advantages. First, the library routine is written as an ordinary closed subroutine and is not in itself concerned with going up or peeling back, unless it needs to call another routine in the course of its operation. It may therefore be operated either by going up to a new MAP or by being brought into core as a part of the MAP containing the calling program and called as a subroutine. The latter mode of operation has speed advantages but is limited to situations where core assignments and index register usage are compatible. Checking for compatibility is left to the programmer in this case. A second advantage comes about because the MAP changing facility is not limited to library routines but is available to arbitrary user programs. The MAP stack then aids the programmer in putting together large complicated program structures which may exceed both the real core and the core addressing capacities of the ma-

chine. However, he must keep in mind that changing MAPS involves a bookkeeping overhead for the executive and may involve file memory swapping (at disc speeds) if real core capacity is exceeded.

The APEX system also uses MAPS to handle interrupts. The user's program may define special MAPS called GHOST MAPS which may be individually associated with all sources of interrupts. When an interrupt occurs, the associated GHOST MAP is placed on top of the MAP stack, and control is passed to the program there. For alarming situations such as illegal instructions, boundary violations, and I/O troubles, a special HELP GHOST MAP is provided, which automatically takes the user to a fixed public routine which straightens out his I/O problems of the moment, if any, and then sets up a basic translator which allows him to call debugging or other routines to his aid. Note that in this situation, the HELP GHOST MAP has suspended the operation of his program and has given him the full use of his apparent computer to work on his trouble. In addition, the MAP stack has preserved all that was known about his program structure at the time this interrupt occurred. He may be able to fix the trouble and continue, discarding only the HELP structure at the top of his stack, or he may elect to start again at the bottom, forgetting everything about his old structure. GHOST MAPS may be defined which will intercept all interrupts according to arbitrary priorities, but the HELP GHOST MAP may not be overridden and is always there should some other GHOST MAP get into trouble itself.

## STORAGE AND RETRIEVAL FACILITIES FOR THE APPARENT COMPUTER

One of the major tasks of the executive system is to remember the user's data and program files as well as other quantities which he may find useful in maintaining his continuity of operation from one session to another. A portion of the executive called the Librarian maintains a private directory for each user as well as a public directory which is shared by all users. A number of calls are available to the simulated console computer to allow a user's program to insert items into his private directory and to inquire about these items and others in the public directory. Items remembered through a directory are identified by names. A name is a string of up to 50 characters. The characters which may be used are restricted to Roman capital letters, Arabic numerals, and period. The directory itself is a list structure arranged in the form of a tree to give a logarithmic search for names. Once a name has been entered into the directory, a unique name block is created within the list structure, and the pointer to that name block is used as a compact and more efficient substitute for the original string of characters. Remembering an item in the directory involves two calls. The first call asks the Librarian to accept a string of characters and to return the related name pointer. The second call uses that name pointer together with the necessary defining information to request the Librarian to establish an association between the name and the item to be remembered. The director can keep track of the following kinds of items, either directly or by way of the file memory.

1. *Files.* A file is any contiguous group of memory registers. As discussed above in connection with the storage structure of the apparent computer, the directory contains information concerning the protection status and origin (if any) for each file, and knows whether or not the file contains program or data. In the case of a data file the type and kind of data which the file contains is known only by the internal format of the file and not by information in the directory.

2. *Scalars.* A scalar is a single-register quantity remembered directly within the directory. The directory scalar is useful for allowing the user to remember, from one session to another, quantities which are not part of some fixed program. It is also useful in allowing re-entrant public routines to remember certain parameters from one usage to the next.

3. *Entrances.* An entrance is a number associated with a file. A program file may contain a number of related routines which perform different functions. Entrances can then be used to call these different routines by entering the program file at different locations. If a GO UP to a new MAP call is given to the executive and the parameter on that call specifies an entrance, the specified file will be set up (if it is a program file), and control will be transferred to the location specified by the entrance. Entrances may also be used in connection with data files. For example, an entrance may identify the start of a particular ring in a list structure.

4. *References to Public Names.* A reference to a public name is a device for allowing a user to use a name of his own choosing for a public name which is unsatisfactory to him.

5. *File Groups.* The file group, as the name implies, is merely a related collection of files. Its existence in the directory allows a related group of files set up in memory by means of a single call. For example, consider the case of using a general translator to translate a particular problem-oriented language. In addition to the file containing the translator itself, a file of definitions for the particular language and a suitable working storage file must be set up before any translating can begin. Treating them as a file group allows the executive to get them all into core and set up before any attempt is made to run the program.

The directory not only maintains relationships between names and things but it also maintains relationships between names. A synonym call is available which allows the user program to indicate that a particular item in the user's directory is to have a second synonymous name. A name may have any number of synonyms. These are added one at a time by the SYNONYM call and may be removed one at a time by the UNDEFINE call. If all of the names have been removed by the UNDEFINE call, the original named entity will be forgotten by the directory and destroyed. A DROP call is available which will cause all the names to be undefined and the entity destroyed with a single call. Synonyms are useful for abbreviation and parameter substitution. They are handled by the executive rather than left to particular translators because they are felt to be language-independent relationships which should be closely tied to the items remembered via the directory.

## INPUT/OUTPUT FACILITIES IN THE APPARENT COMPUTER

The input/output devices available to the console computers may be split into two categories. The first contains those which must be shared among the consoles because there are not as many devices as consoles, and the second is made up of those devices located at each console. The first class of devices may be more important in this time-sharing system than in many others because the majority of the consoles will be located in the computer room, and the users at those consoles will have easy access to the common shared devices. The shared devices may be again split into two classes. The first of these includes those devices which are assigned to the individual consoles on a first-come first-served basis. If the user on one console wishes to use such a device and finds that it is assigned to another console, he must go and negotiate with the user at the other console for its release. Devices in this class include magnetic tape, the photoelectric paper tape reader and the analog-to-digital input. The second class of shared device includes the xerographic high-speed printer and the paper tape punch. For these devices the executive maintains a pseudo input/output device which accepts user calls to build files of characters to be printed or punched at some opportune time. These buffer files are saved in the file memory until they can be processed. These devices appear to be always available to the console computers even though the actual physical output may appear at some later time. These shared input/output devices can produce a considerable load on the executive program and their presence in the system posed a number of detailed problems to the designers of the executive, but the solutions to these problems are too specialized to the nature of TX-2 to warrant further discussion of them here.

The input/output devices available at the individual consoles were of particular interest in this system because they are the means by which the user interacts with the system. There are, of course, differences in the makeup of individual console equipment. But the basic console is made up of a keyboard, a typewriter, an oscilloscope display with light pen, and a few push-button switches. A RAND tablet will be available on one console and some of the consoles will have a connector with 36 output and 36 input wires to which a user can connect special equipment of his own. The keyboard and typewriter are basically the same equipment that has been used for TX-2 on-line communication and paper tape preparation in the past. The color-coded keyboard has a double set of keys, eliminating the need for case shifting. The typewriter has a platen rotator which allows super and subscripting. The keyboard has the full Roman alphabet only in capitals, thus allowing more than the usual number of special symbols. The character set allows for the very nice typing of mathematical expressions, but English text is singularly poor because of the lack of a full set of lower case Roman

letters. The keyboard has seven so-called "function keys" for which there is no direct typewriter key. These keys are available for particular interpretation by programs operating within the console computer. The keyboard sends its output signals only to the computer, and the computer must send signals to the typewriter to type back what the user has keyed in. The executive handles this typing directly. When a key is struck, the executive places the character code for that key directly into keyboard buffer file for the appropriate console computer. At the same time the executive examines each character to see if it is one of a set of terminating characters specified by the user's program. If the character is a terminator, and if the user's program has gone into an inactive state waiting for further typing to be complete, then the executive will put the user's console program into an active state. In order for the user's program to gain access to the keyboard input, the keyboard buffer file must be set up as a part of the user's MAP. Both the user's program and the executive may write in this file, and it is the responsibility of the user's program to write empty marks in those registers from which information has been extracted. If a key is struck and the executive finds that the next location in the buffer file is not empty, it will interrupt the user's program and switch to the HELP GHOST MAP structure. This method of handling keyboard inputs is well suited to the ordinary translator form of user program which completes an operation before returning to look at the input for the next command. In the case where it is desired to use the keyboard to interrupt an on-going program, a GHOST MAP type of interruption mode is available for the keyboard. In this situation, the occurrence of a terminator character will cause the prespecified keyboard Ghost Map to be put into operation.

The typewriter has three functions in this system. It normally types back the keys struck by the user. This mode may be disabled in situations where the keyboard is to be used as an input to the oscilloscope. The typewriter will also type messages from the user program as well as messages from the executive itself. The latter have the highest priority, with messages from the user's program second, and typing back of the user's input as the lowest priority task of the typewriter. In the case where the typing back of input has been interrupted because of a system message or a user program message, the user's keyboard is locked until the messages are complete. If, in this locked keyboard situation, he wishes to interrupt a message from his program, he may push the HELP REQUEST button on the console and go to the HELP GHOST MAP for corrective action. He may not under any circumstance interrupt a message from the executive.

Cathode ray tube displays and associated software techniques are areas in which much work has been done on TX-2 in the past. It is expected that there will be considerable future development in these areas. As a consequence, the organization of the display portions of the executive program have received considerable attention. The present design of this part of the system appears to be a reasonable compromise between the various requirements placed upon it, but it may well have to be changed in the not too distant future as new requirements and techniques develop. The displays themselves are slave-type units. Some have Charactron tubes, and some have ordinary cathode ray tubes. They are driven from a shared vector and curve generator which gets its inputs directly from the computer memory. Analog integrators are used to generate lines, circles and parabolas from digital information obtained from the computer. The analog deflection signals are sent to all scopes, but only the scope for which the display information is intended receives an intensification signal. The display information is stored in a ring-type list structure, which reflects not only the order in which parts of the display are to be produced, but also the associations which may exist between parts of the picture. A display routine within the executive threads its way through this ring structure transmitting the data it finds in the structure to the display generator. Characters are stored in the display file in the form of packed keyboard character codes. If the particular console for which the display is intended has a Charactron tube, the display routine will recode the character code appropriately and transmit that information to the display scope itself. If the console does not have a Charactron tube, the display routine will generate from a stored list the appropriate vector and curve segments necessary to make up the character. A single pass around the ring representing the display for an individual console defines a frame for that display. The display generator is time-shared on a frame-by-frame basis. Control bits in the list structure allow selected portions of a display to be darkened, either until further command from the user pro-

gram, or on an alternate-frame basis so that a flashing effect can be obtained. The display list structure is built in CORAL language format by executive calls given in the user's program. The CORAL language builds a block-format list structure. Thus an entire drawing may appear as a single block in the structure if the user so chooses. The structure also allows for associations in a hierarchical sense between the blocks in the structure. Thus the display file is not just a sequential list of lines, points, curves and characters which make up the picture which appears on the display, but it can also represent some of the relationships which parts of this picture may have to each other. The display routines maintain the display on the user's console even though his program is inactive either because of time-sharing or because it is waiting for an input. A "push-to-see" button is used by some types of display programs to keep down the load which display maintenance places on the system.

The light pen is the principal graphical input device available in the APEX system. The light pen may be used in two modes—pointing and tracking. In both modes the executive maintains a complete record of all light pen returns in a buffer within the display file. A light pen return while in the pointing mode causes the executive to place information in the buffer from which the user's program can calculate both the pen position and the picture element seen by the pen. In many light pen applications it is necessary to associate the pushing of a button or the striking of a key with the pointing operation. This association is handled by the executive, and the Associated character code is placed in the light pen buffer. In the tracking mode, the executive displays a tracking cross every 10 milliseconds. If the light pen sees any part of the cross, the tracking routine moves the cross to center it in the field of the light pen. The location of the center of the cross is placed in the buffer. Smoothing and extrapolation are done in the tracking program to achieve good "writing" characteristics for the pen. The processing of light pen signals is a high priority task for the executive since response time is a critical parameter in graphical input operations.

## ORGANIZATION OF THE EXECUTIVE PROGRAM

The SPAT address transformation is a three-level transformation. The first level is unique to TX-2 and comes about because TX-2 is a multiple-sequence computer. TX-2 has 33 program counters. Most of these are associated with I/O devices and must be used to operate these devices. Some are used to handle interrupts, one is used to start programs from console controls, and the remainder are used for computational purposes. Most ordinary programs in TX-2 use but a small subset of the available program counters at any one time. A priority relationship between these program counters determines at any instant which of them will provide the address for the next machine instruction cycle. Figure 3 shows how the first level of the SPAT transformation treats this multiple sequence structure. The SPAT hardware provides for a total of 64 books or segments. These are divided into 4 shelves of 16 books each. Three of the shelves are tied to single program counters. One of these is used for user programs in the current system. The other two are treated as user shelves by the hardware (i.e., privileged instructions are prohibited) but are treated as unused spare shelves by the current executive program. The fourth shelf is shared by all the other program counters except a special one which is used for starting the computer and is not subject to the SPAT transformation. This fourth or executive shelf contains all of the executive program. The executive is thus itself transformed by the hardware it controls. The application of SPAT to the executive makes the advantages of segmentation available to the executive program as well as to the user programs. SPAT allows all private directories to appear in the same block of addresses in the executive MAP. Switching from one to another requires only two instructions. Similarly, I/O buffers can be quickly switched, and drum and tape transfers can be carried out while a file is being relocated. The registers which control the SPAT transformation appear as part of addressable memory and are themselves transformed by SPAT. They may thus be easily protected from user program interference by allowing them to appear only in the executive shelf. The problem of allowing the executive to examine or change a register in the user's shelf is solved by setting up the appropriate file in a spare book of the executive shelf. Switching between users' core MAPS is handled by changing the 16 SPAT registers which control the user shelf.

The five major parts of the executive are as follows:

1. *The Maestro* is the part of the executive that determines which user, interrupt, or alarm is to be handled next. It implements the time sharing scheduling algorithm, which is a simple round-robin of

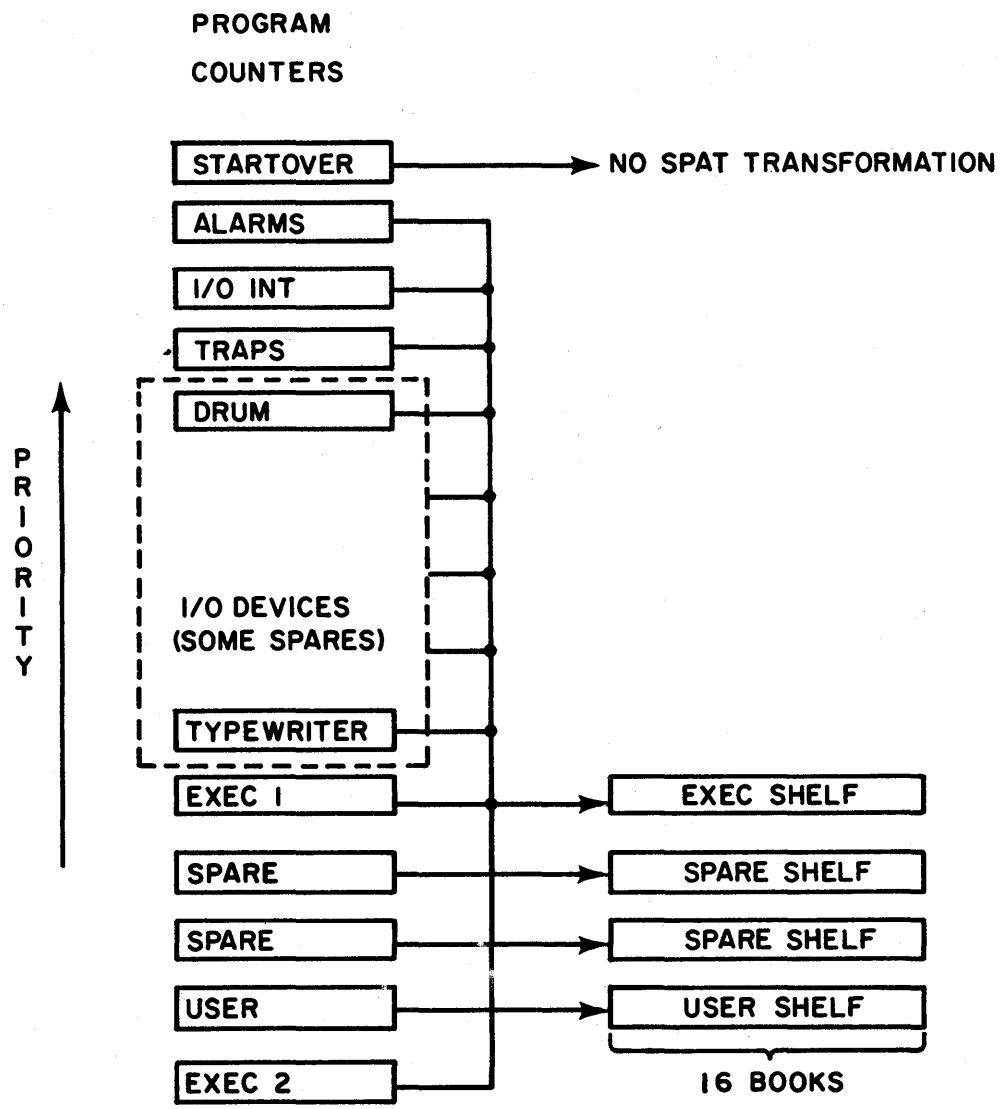



Figure 3. First level of the SPAT transformation in TX-2.

all active users. The requirement for fast response eliminates most other potentially more efficient schemes. Fortunately, the small number of users in the system complements the fast-response requirement.

2. *The Csar* (Core Storage Allocation Routine) handles the bookkeeping required to maintain the users' MAP structures. The SPAT hardware with its limited page address memory (corresponding to twice the limit of addressable core) reduces but does not eliminate the storage allocation problem. Paging removes the need to move registers in core, and segmentation reduces the number of consecutive registers that must be found in the page address memory (PAM), but it is still occasionally necessary to move files in PAM. The Csar handles the allocation of PAM space, maintains a supply of free core pages, and calls for files to be transferred to and from file memory or discarded. It is by far the largest of the five parts of the executive.

3. *The Librarian* builds and maintains the public and private directories. It provides a source of

ephemeral names for temporary quantities and supervises their destruction when requested.

4. *The Mover* transfers information to and from the file memory (a Univac FASTRAND drum). It keeps track of free drum space and maintains bounds on each user's share of drum space. Actual transfers are made on a page-by-page basis, but bookkeeping is done on a file basis. The pages within a file are tied together on the drum by a list structure. Free drum space is not tied together on the drum but is found from a bit table in core. Files are stored at random on the drum and users are limited by a quota rather than a fixed drum area.

5. *The Secretary* handles all input/output communications and interrupts. It is made up of a central program and a number of routines which are specific to individual I/O devices. These device routines are optional parts of the executive and do not require core space if the devices are not in use.

## COMMENTS AND A LOOK TO THE FUTURE

The APEX executive program was designed for fast response. Its reponse, as must always be the case, is not as fast as some users would desire There are three major areas in which work is being done to improve the response characteristics. One is the area of bookkeeping overhead. The present program uses list structures built in CORAL language format for all bookkeeping. CORAL was chosen to simplify the programming of the system, but its use exacts a price in storage and time which could be reduced by the use of a more specialized list structure. However, the major reprogramming of an experimental system sucy as this is unlikely. The only noticeable improvement in bookkeeping overhead will probably come from the addition of a list-processing instruction to TX-2. A second area in which changes can improve response time is I/O. The contemplated addition of an I/O memory bus would make a substantial increase in the number of memory cycles available to the CPU during periods of high I/O load. The third area involves the auxiliary memory facilities on TX-2. The access time and transfer rate of the present drum system are such as to cause a serious degradation in response when memory allocation exceeds available core. With a faster auxiliary memory, this degradation could be substantially reduced. If such a memory existed on TX-2, the SPAT hardware would be very useful in implementing a page-turning scheme which would allow an individual user to address 390K of virtual core without excessive overhead costs.

# A DESIGN FOR A MULTIPLE USER MULTIPROCESSING SYSTEM

James D. McCullough
Kermith H. Speierman
and
Frank W. Zurcher
*Burroughs Corporation*
*Paoli, Pennsylvania*

## INTRODUCTION

The B8500 system is designed to deal with the following situation. A large number of active programs requiring various services are present in the system and their current status and required service are recorded. When some component of the system becomes available, e.g., processor, memory space, peripheral device, it is assigned to the active job of highest priority that requires this service. The important concept is that no component of the system belongs to any program but rather provides a service and then goes on to service another program. The main function of the executive scheduling program is to keep track of the services required by programs and to schedule the services when equipment becomes available.

This mode of operation requires that the system have the following properties:

1. The equipment should consist of independent modules that can function concurrently; e.g., processors, memories, I/O, etc.
2. A bulk memory system that is a logical extension of main memory.
3. Segmentation of data and programs to make more effective use of memory and permit a large number of active programs to be present in the system.
4. Dynamic allocation of memory.
5. Memory Protection to prevent interference between programs.
6. An Executive Scheduling Program (ESP) that controls and schedules the entire system.

The Burroughs B8500 is a modular processing system designed for a multiprocessing and multiprogramming mode of operation. In addition to the concept of multiple central processors, the B8500 also functions with multiple input/output processors which operate nearly independently of the central processors. A high-speed fast-access disk storage unit is provided as an extension to the main memory and is used concurrently by the input/output modules for storing input from external communications and retrieving required programs and data for the central processor.

## STRUCTURE OF PROGRAMS

Program segmentation is a basic requirement of a multiple user system to provide effective use of memory by permitting a large number of active programs to be present in the system. A B8500 program may be considered as the output of one compilation consisting of program segments, data segments, an operand stack segment, a working storage segment, and a program reference table (PRT). The minimum structure that a program must retain in main memory to remain active is one program segment, an operand stack, a working storage segment, and a program reference table. A large program may also require additional program and data segments at various periods of program execution. The allocation of required segments is provided at the time the segments are referenced through descriptors in the program reference table. The descriptors define the segments as they appear on disk storage and when the segments are allocated in main memory the descriptors provide communication between the separately allocated segments of the program.

### *Tag Bits*

A memory word contains 52 bits, 48 data bits and 4 tag bits that may only be modified in a protected control mode. In addition to a parity bit, three tag bits are provided to construct and control memory words used as descriptors. One of these tag bits is a presence bit that is used by ESP to define the presence in main memory of the segment that is represented by the descriptor. A reference through a descriptor to a segment not yet allocated in main memory causes a presence bit interrupt and invokes ESP control of allocation of the required segment. The two remaining tag bits are encoded as program descriptor, data descriptor and indirect memory reference. A typical program is shown in Fig. 1.

### *Program Segments*

Program Segments contain instructions and constants and may have a maximum length of 4096 words. Programs are location-independent and all internal addressing of constants and jump instructions is relative to the Base Program Register (BPR) which contains the absolute address of the segment base. Jump instructions may reference any syllable within a word. Syllables are six bits long and instructions contain from one to four syllables. The Program Counter Register (PCR) is a 15-bit register relative to the BPR: 12 bits are required to address the relative word in the segment and 3 bits to address one of the 7 syllables contained in the word. The PCR was designed to enhance dynamic memory reallocation and allows the ESP to move a program segment which has been partially executed, simply by changing BPR to the new base location of the segment.

Program segments can only be read and are protected from accidental modification since they are allocated outside the area bounded by the memory bounds registers. Since program segments are referenced by program descriptors which have the appropriate tag bit configuration, they may never be accidentally read as data segments. Program segments may contain internal subroutines which are referenced via a program descriptor in the PRT. While individual program segments are restricted to the 4096-word limit, a large program may contain many program segments. Transfers between segments are directed by program descriptors in the PRT.

### *Program Reference Table*

The program reference table is a read-only segment and contains descriptors for program communication with data segments and other program segments. Descriptors are addressed relative to the PRT base register and addressing beyond the limits of the table is prohibited. A program is filed on the disk with its PRT and program and data segments.

The filed PRT contains the information required by ESP to construct the descriptors which must be present in the PRT when the program is placed in active status. For each descriptor, this information includes the name of the object to which it refers, the type of object (procedure, simple subroutine, data, etc.), the mode of use (global, own, read-only, etc.) and the descriptor required for its input from the disk. When the PRT is input to memory, ESP decodes this information and sets the necessary tag bit configuration required for processor recognition. One tag bit configuration is set for descriptors which refer to program segments, procedures, and subroutines and another configuration is set for

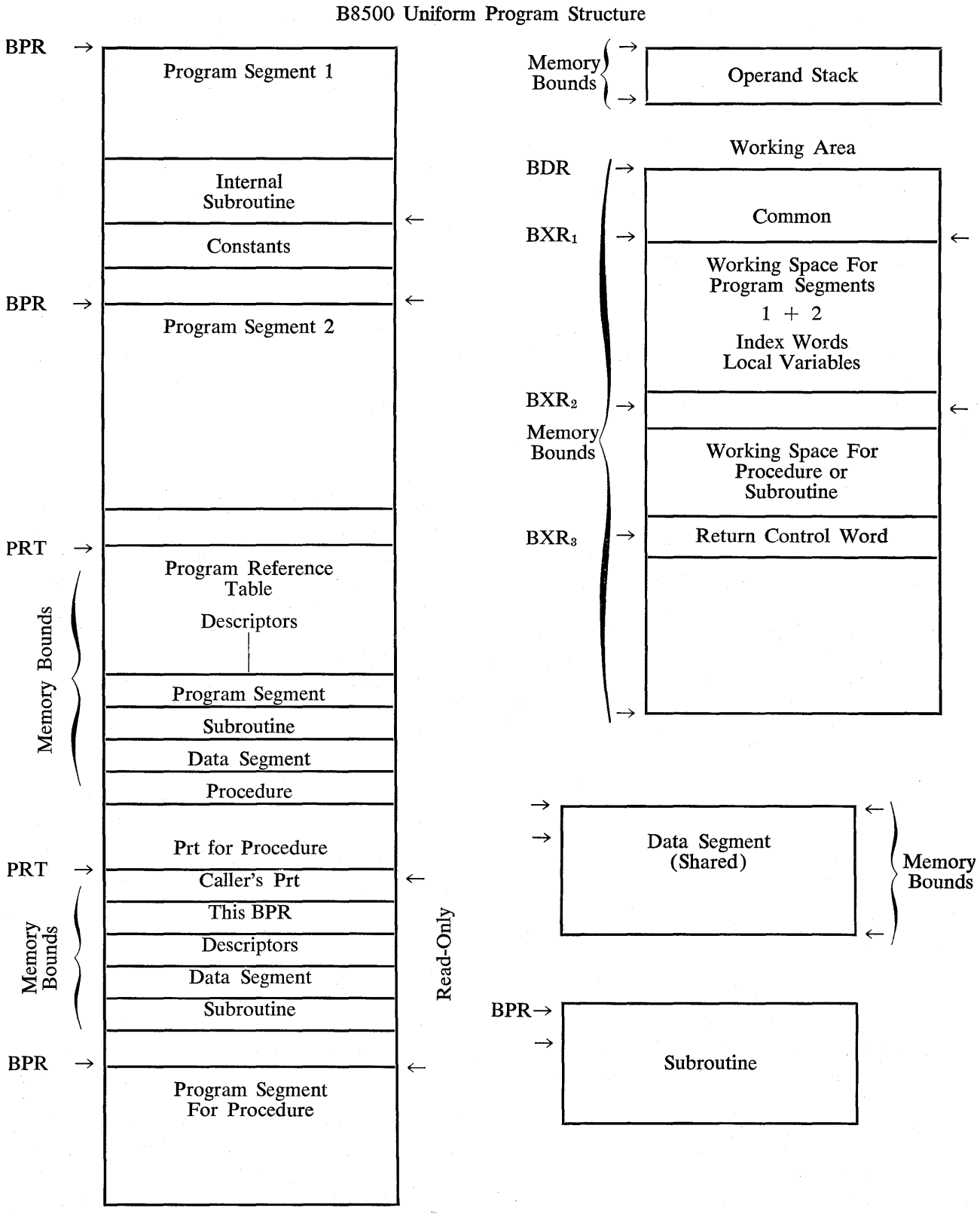



Figure 1. Typical Program Structure.

those descriptors which reference data sets or memory space.

Words zero and one relative to the PRT base register are reserved for special use on a procedure call. A procedure is a program which requires its own PRT, and is provided to permit calls on programs which have been compiled separately from the calling program. A procedure call is executed by a reference to a program descriptor in the caller's PRT. This program descriptor contains the address of the procedure's PRT instead of the program segment of the procedure. Word zero relative to the new PRT is used to save the contents of the caller's PRT base register such that it may be restored to its correct value when the procedure returns to the caller. Word one relative to the new PRT contains the BPR value of the procedure's initial program segment.

*Data Segments*

Data segments are addressed relative to data descriptors which contain the absolute addresses of the segments. The tag bits of the descriptor determine the memory bounds. The next instruction which executes a memory fetch or store is compared with these memory bounds, providing both read and write memory protection. Any reference to a data descriptor relative to the PRT base register causes that descriptor to be placed in the processor's local high-speed memory such that subsequent references to that descriptor will not require a main memory fetch if it is among the last 16 descriptors referred to.

*Working Memory Segment*

The working memory area comprises two logical segments, common, and the subroutine control stack, allocated in a contiguous memory block and bounded by the processor memory bounds registers. The common, or global data area, is addressed relative to the Base Data Register (BDR). It should be noted that we have not provided any direct method of setting or saving the BDR on subroutine jumps or procedure calls because of its intended use for common data.

The subroutine control stack provides dynamically allocated space for subroutines and procedures and is used to contain local variables and index words and for passing parameters between procedures and subroutines. Addressing in the control stack area is relative to the Base Index Register (BXR). When a subroutine or procedure is called, the BXR is incremented beyond the calling program's control stack area; parameters and return information are stored relative to the new BXR value; and the called program addresses relative to the new BXR. Word zero relative to BXR is used to save the relative BPR, PCR, BXR increment, and jump control bits of the calling program. The subroutine return instruction refers to this location for its information. This structure provides an automatic mechanism for subroutine nesting and recursion. Any of the 4096 directly addressable words relative to BXR may be used as index words; the most recently used are kept in the processor's local high speed memory.

*Operand Stack Segment*

The operand stack segment is used by the processor to hold operands and results for the arithmetic and logical instructions. Programs for expression evaluation are Polish strings which are directly executed by the arithmetic unit using a push down stack implemented in the processors hardware. The stack segment discussed here is in main memory and is a logical extension of the processor's stack.

*Memory Protection*

Memory space is allocated by the ESP and given to the user program by setting bounds registers in the processor and descriptors in the PRT. These registers and descriptors can only be set by ESP (in the control mode) which prohibits the user from having any control over the assignment of memory.

The working segment and operand stack segment are read-write areas and each are defined by memory bounds registers. The program segments are read-only objects and are not contained within the limits of bounds registers. The PRT, which is a directory of all program and data segments used by a program is a read-only object and is contained within the limits of bounds registers which prevent using any descriptors that are not in this PRT. If a user tries to change his PRT he will be interrupted and ESP given control. Data segments are referred to by data descriptors in the PRT. Each time such a data reference is made the descriptor sets up bounds around the data segment being referenced for the

duration of the data reference. Any attempt by the user to read or write any other areas of memory will cause an interrupt and entry to ESP.

It is possible to branch outside of a program segment without detection but the program is still restricted to its own data and working storage segments so it can't effect another user by accidentally branching to his program. I/O operations are controlled by the ESP to prevent a user from interfering with another's space. This combination of ESP and hardware conventions allows any number of user programs to be executed together in a multiprocessing-multiprogramming mode without the danger of accidental interference.

## THE EXECUTIVE SCHEDULING PROGRAM

The Executive Scheduling Program (ESP) schedules both hardware and software services for all programming tasks or jobs that are present in the system. Many of the services of the ESP are themselves programs that are structured as normal user programs and therefore may also be scheduled in the normal manner. The intent is an organization of the ESP consisting of many subprograms which are separately constructed and therefore may be executed concurrently. Each of these subprograms is extended system privileges according to the functions it is to perform, e.g., initiate I/O, manipulate tag bits, etc. The ultimate requirement of the ESP is to efficiently schedule all services, both hardware and software, to effectively establish maximum throughput of the system.

### *Scheduling*

Jobs may be introduced to the system from various sources. Prestored production tasks are entered by the ESP without any external request; requests may be entered from external remote stations; input streams from peripheral devices are interpreted for batched requests; or a program or job may request the execution of another job during execution. All jobs presented to the system are retained on bulk storage and descriptions of these candidates for scheduling are kept in a Cold Job Table which is also kept on bulk storage. A Cold Job entry remains in the system from the time it is introduced for execution until its final outputs are delivered. Each Cold Job entry contains information required by the scheduling program to efficiently introduce tasks to the system. Information required in each Cold Job entry includes class and priority of the task, estimated processor time, memory requirements, input files required, dependence upon other jobs and accounting information.

Prior to a job's introduction to the scheduling program, the availability of program and data files must be established. This fundamental requirement is established to insure that once a program is entered for execution, its completion will not be deterred by the unavailability of a program or data file when required. Therefore, prior to successful scheduling, a collection program is invoked to accumulate the job's external files on bulk storage and present to the system the required information concerning those files.

When a job is acceptable for execution, the scheduling program generates a Hot Job Table entry in main memory and requests the allocation and readying of the program's initial requirements. Initial requirements for all programs are the program reference table, the working storage area, the operand stack, and at least one initial program segment. Other required program and data segments are allocated and readied when they are first referenced through the descriptor which describes them. These Hot Job Table entries establish a path of control which the processor is to execute, and contain the processor state information (hardware register values) recorded at the program's last suspension of execution. The entries also contain the program status and are linked in priority order. The program status may be ready to execute, awaiting I/O, awaiting memory, being executed, awaiting time, or being terminated, and is used by the internal scheduling program for determining the next useful task to assign to the processor.

### *Classes and Priorities*

Class is defined as a mode of operation; e.g., real time, batch, conversational, etc. Priority is defined as relative importance within a class. Classes possess a relative priority to each other. It is a desirable feature that the system be self-regulating to prevent a group of users in one mode from monopolizing the system's resources. The philosophy employed is to get done what must be done in a timely fashion, but always to maintain a lower limit of re-

sources below which the sum of users in a class cannot fall. A good example is the conversational mode in which remote terminal users may experience a decrease in system response time but will not experience a complete blackout due to higher priority requirements.

*Memory Allocation*

Memory allocation is controlled entirely by ESP since no hardware directed technique is attempted. The memory allocation program is responsible for the maintenance of all main memory. Its basic functions include obtaining a block of available space to satisfy a request and to assume responsibility for space being relinquished by its prior owner. Allocation performs its function through the mechanism of linked tables which include all blocks of memory. All blocks of memory, whether available or assigned, are linked by address in a memory map.

All blocks of memory which are available are linked by size in available space map. An attempt to allocate space for a caller is governed by the priority and class of the caller and the amount of space which has previously been committed to callers of that class.

The allocation routine will first try to allocate by scanning the available space map to find the smaller block which is large enough. If a block of sufficient size cannot be found, the overlay program is called.

The function of the overlay program is to find a currently committed block of memory which can be reassigned to the caller of allocation. Using the priority and class of the requestor, the overlay program will scan the Memory Map for a block of memory to be reassigned. The considerations to be applied at each block will be:

1. Does the block belong to a running program?
2. Is I/O going on in the block?
3. Size of block and surrounding blocks.
4. Is the block program or data?
5. Priority and class of block.
6. The number of disk operations required.
7. Number of users.
8. Size required.

If a program segment is overlaid, the appropriate program reference tables are updated to cause interrupt on access by the callers of the segment. If a data segment is overlaid, the data segment is saved in bulk storage before the space is reassigned. User PRT segments and stored register values are updated appropriately.

In the event that a request for space cannot be granted by these means, a deferred space request will be put into an unallocated request chain. This chain will be scanned periodically to allocate the deferred request.

In the event that the system finds itself bound by having too many things to do, and not enough space to do it in, it will (based on priority, class and percent completion) choose a job which has been introduced into the system and terminate processing on that job. The Cold Job Table will be restored to a previous state so that the job can be rescheduled at a later time.

The PRT contains the descriptors through which programs address separately allocated segments without knowledge or regard to the absolute memory allocation. Location independent addressing allows ESP to dynamically change the contents of memory by releasing segments not currently in use and replacing them with other required segments. The presence bit of a descriptor is used to indicate the presence of a segment in main memory and a reference to a descriptor representing an absent segment causes a processor interrupt invoking ESP. ESP must then ready the required segment referenced by that descriptor. The descriptor is interpreted for type and when a global type is required, the memory map is scanned to attempt to locate the desired segment in active memory.

If the segment is not in memory or a local copy had been requested, the memory requirements and disk address for the segment are available from the PRT, and ESP places the requesting program in a suspended status, initiates an input request for the segment, and assigns the processor to some other useful task. When the requested input is completed, the descriptor(s) which addresses it are marked present and those programs may then be scheduled for the processor.

REFERENCES

1. "Burroughs B5500 Information Processing

System Reference Manual," Burroughs Corp., 1964.

2. "A Narrative Description of the Burroughs B5500 Disk File Master Control Program," Burroughs Corp., 1965.

3. "D825—A Multiple Computer System for Computer System for Command and Control," *Proceedings of FJCC,* 1962.

4. R. N. Thompson and J. A. Wilkinson, "The D825 Automatic Operating and Scheduling Program," *Proceedings of FJCC,* 1963.

# A COMPUTING SYSTEM DESIGN FOR USER SERVICE

Webb T. Comfort
*IBM Corporation*
*Yorktown Heights, New York*

## INTRODUCTION

After a long period of study and experimentation with various forms of user/terminal/system interaction, IBM is developing a general purpose time-sharing system. This is the recently announced System/360 Model 67 and the associated programming support package.

The basic technical objective of such a system is to provide a user at a console with what appears to him to be immediate response; i.e., when he asks for something relatively simple to be done, it should be done within 1 to 2 seconds. When he asks for difficult and complicated things to be done, there should not be an unreasonable amount of delay before they are in fact done. (This response time concept is very closely related to the current concern in batch processing systems over turn-around time.) Superimposed upon the response time requirement is the necessity to provide a broad scope of selectable procedures which allow a user at a console to simply and conveniently create, debug and execute his programs. More properly, he needs the necessary facilities at his fingertips to solve his problems.

However, from a marketing point of view, this is not sufficient. For those customers who have a requirement for a major facility of that type, the system must be able to support large numbers of such users simultaneously without an unreasonable amount of system overhead. Indeed, there are some computing installations in the country today who are prepared to take just such a step. On the other hand, there are also a good number of installations —in fact, probably the majority as of today—which have a requirement for some immediate access of this type, but not at the expense of crippling their normal batch processing operation. Consequently, the system design objective has to be to provide a flexible system which can provide either type of service (immediate access or standard batch) as the demand fluctuates.

One other point, which has been made increasingly clear in most of the pioneering time-sharing systems across the country, is that in the "hands-on" type of system operation being discussed here, long and arbitrary periods of system down time are simply unacceptable. As a result, the system design must include procedures for automatically handling as many hardware error situations as possible and avoiding a total system shutdown as long as possible.

Now, it is not the intent of this paper to describe in detail the Model 67 hardware, the programming system problems and solutions, or the specific user interface. Rather, some of the basic system charac-

teristics will be discussed, particularly as they relate to the objectives pointed out above. Then the major hardware and software characteristics will be described. In this way, the reader should be able to get a feel for the overall system operation without getting bogged down in a myriad of details. Throughout the software discussions, the emphasis will be on the control program area rather than the language area, since it is the control program and its associated routines which determine how the system will operate.

It should also be noted that the system design has been influenced in a multitude of ways by various previous internal IBM efforts, including in particular the following:

TSM[1]
QUIKTRAN[2, 3]
IBM's recently announced Administrative Terminal System
Studies of One Level Store
Studies of Polymorphic Multiprocessing

It has been aided by a number of discussions with various individuals from General Motors Corporation, Lincoln Laboratories, and particularly the University of Michigan.

## SYSTEM CHARACTERISTICS

There are several basic characteristics of system operation which dictate how the hardware operates and what the design approach to the programming system must be.

The basic mode of operation is multiprogramming, wherein a multiplicity of tasks (represented by programs) reside in core at the same time and have access to common devices. However, the normal multiprogramming technique has been to run one task until it had to wait for some reason, such as for completion of some I/O. At that point, the CPU could be switched over to another task, to return to the first one later when its I/O was complete. Time-sharing goes a step beyond that, in that it is known that a certain (dynamic) number of tasks *must* be serviced within a reasonable period of time; namely, the response time mentioned in the Introduction. To this end, an algorithm is used to determine dynamically how much of the system's resources a task ought to be allowed to consume. If this threshold level is exceeded, the task is forced to stop and wait while other tasks have a chance; in other words, *forced* multiprogramming of sorts.* (This has been generally termed as *time-slicing.*) In this way, a multiplicity of tasks—and therefore a multiplicity of users at consoles—can be accommodated.

The system is designed to operate with multiple CPU's and with multiple CPU-independent selection paths for I/O devices. This is necessary to provide the desired increase in system availability (as distinguished from reliability). It is also necessary in order to maintain orderly growth, particularly as it will probably not be possible to specify optimal system configurations until after the system has been in operation for a while. In addition, this allows for partitioning within the system, to provide smaller but independent subsystems, if desired. (The same system will, of course, operate with only one CPU.)

Dynamic relocation is implemented (as described in the next section) and applied. This allows a task to operate as if it had a full addressable memory (called a *virtual memory*) of $2^{24}$ ( or, in the larger CPU, $2^{32}$) bytes, almost independent of the amount of real core provided in the system. Dynamic relocation can be used to simplify the traditional relocation techniques. This is particularly important in time-sharing, where it often becomes necessary to throw a task out of real core before it is finished and bring it back later (the forced multiprogramming indicated above). Dynamic relocation completely eliminates the problem of having to return the task to the identical area of core it occupied during its previous period of residency.

There are two characteristics of program execution which have heretofore been unexploitable. The first is that in general a program must claim an amount of core large enough for its worst case operation, even though in many cases, either during debugging or based upon parameter values or program structure, the actual core requirement for a given execution is significantly less than that maximal amount. The second is that in many cases program activity is localized for significant periods of time, i.e., during its execution, a certain set of routines will run for a while, and then another set, while the first

*In many places in this paper, for purposes of avoiding unwarranted complexities, general statements are made. Many of them, if interpreted literally and assumed applicable in all cases, would be clearly objectionable. It must be emphasized that such generalities are for expository purposes only, but do represent the basic system action.

are not used. To capitalize on these situations, the *paging* concept is used, wherein through the dynamic relocation facility the control program can recognize on a dynamic basis which parts of a program or its data are now required, and which are no longer required. Virtual memory is then broken up into blocks called pages, and only those pages actually referenced by the task will be brought into real core. This will provide a better utilization of real core storage, as well as allowing an equivalent level of multiprogramming operation (when compared with always moving a complete package of a program and its data) but within a significantly smaller amount of real core.

In the past, it has generally been necessary to declare at least by load time all subroutines which might possibly be needed during execution, and to load them all into memory before execution can begin. Dynamic linkage (or, more properly, dynamic loading) is a facility whereby a routine need not be declared or loaded until, at execution time, it is actually called. (At call time, virtual memory can be allocated for the called routine and external symbol definitions resolved. Actual relocation modification of addresses—or, in System/360 terms, address constants—need not occur until paging time.) This is particularly important for a user at a console, for he may elect in the middle of an execution to change his mind about what subroutine he wants the program to call—and he ought not to be required to stop and reassemble or reload. This also contributes to more effective utilization of real core.

A general purpose terminal oriented system would not be complete without a "warehouse" of previously stored programs and data sets, maintained and cataloged by the system, and callable by the user or his program.

The scope of user facilities is defined by the set of languages at his command. The basic Model 67 program package will include a Command Language (with a set of Program Checkout techniques), a FORTRAN compiler, and a macro assembler. Later extensions will include COBOL and PL/1.

## HARDWARE CHARACTERISTICS

Several extensions and modifications have been made to System/360 hardware in order to facilitate the system characteristics summarized in the Systems Characteristics section. The most important of these will be discussed briefly here. (It is assumed that the reader is familiar with the basic System/360.[4]) In those situations where the CPU is altered, there is generally a switch of some sort (physical or programmable) which will disable the feature, thus assuring that Model 67 will still run programs prepared for any other model of System/360.

Standard System/360 has a 32-bit word, and 24-bit addressability; i.e., $2^{24}-1$ is the largest memory address recognizable by the hardware. On the Model 67, an option is provided for full 32-bit addressing. Associated with this facility is a new instruction, Branch And Store, which is essentially a 32-bit version of the Branch And Link instruction.

In System/360, an effective address is formed in the general case by forming the sum of the contents of a base register, the contents of an index register and the contents of the displacement field specified in the instruction. The Model 67 is provided with a program-controlled relocation mode, which imposes a translation function between the time the logical (effective) address is generated and the time the address is sent to the appropriate memory box. The total addressing space (virtual memory) is broken up into pages of 4096 bytes apiece. Thus a logical address can be considered to be a 12-bit page number (optionally 20-bits with 32-bit addressability) concatenated with a 12-bit byte address within the page. The basic mechanism is to provide a relocation table for a direct look-up of the logical page number, and to fetch a new physical page number from the table. Because of its potential size, this relocation table is kept in main core storage rather than in its own hardware implementation. However, since the translation must occur on the fetch of every instruction and operand from core storage, a set of associative registers are provided to reduce the number of additional memory references to an acceptable level.

Figure 1 shows a simplified data flow for the relocation hardware. It will be noted that the page number translation is a two-step process. That is, pages are divided into groups of 256, called *segments*. The logical segment number is looked up in a segment table, and determines the location of a page table, which is then used to translate the logical page number into a physical page number.

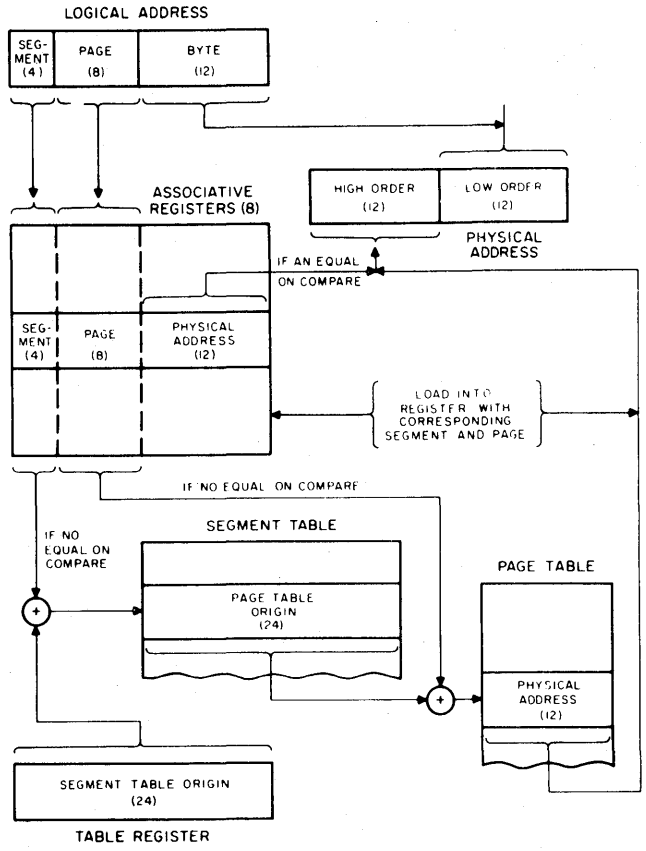



Figure 1. Simplified data flow of dynamic relocation.

Availability bits are provided at both levels, and are used to cause CPU interrupts when references are made during execution to selected segments or pages. (This is how the paging technique is implemented.) This two-level relocation technique provides the following:

1. A convenient way to allocate a data area of unknown length, by allocating it at the beginning of a segment, and allowing it to fill the segment.
2. A way of compressing page tables, since actual table entries are not required for unused pages at the end of a segment.
3. A way of reducing the amount of real core required to contain page tables at execution time, since segments can be marked unavailable, and the associated referencing interrupt used as a signal to bring that particular page table into core.
4. A very convenient way of sharing programs or data sets among tasks, since a one-page table could be pointed to by segment table entries for several different tasks.
5. An effective mechanism for read/write protection of areas of virtual memory. The reason for a segment being marked unavailable is open to interpretation by the control program.
6. A simple and efficient overlay mechanism, through substitution of segment table entries.

Associated with the relocation feature is a new privileged instruction, Load Real Address, which allows the control program to find the translated form of any virtual memory (logical) address.

In System/360, basic CPU control is contained with a Program Status Word (PSW). In Model 67,

this PSW control has been extended by a set of control registers, whose contents include the table register (which defines where the active segment table is), extended masking facilities, and program-readable status indicators for various system switches. Associated with this capability is a pair of new instructions, Load Multiple Control and Store Multiple Control, which allow for manipulation of the control register contents.

In order to allow for multiple-CPU operation, multiple memory bus connections are provided. Two-channel switches and channel controllers serve to provide CPU-independent selection and control of I/O. Extended Direct Control provides for inter-CPU communication. The instruction Test And Set allows guaranteed interlocking where required.

Partitioning switches are included at critical communication points within the hardware to allow for dynamic manipulation of the hardware system configuration. New devices can be added, others removed for testing, or a full subsystem partitioned out for independent operation. This facility must be carefully controlled in actual operation to prevent unintentional system operator slips from disrupting the system. On the other hand, ultimate authority must rest with a human being, to safeguard against undebugged or recalcitrant programs.

## SOFTWARE CHARACTERISTICS

The basic unit of control in the system is a *task*. For a user at a console, one task is defined to provide services for his complete session at the terminal (i.e., from logging on to logging off). For a non-conversational (batch) job, a task controls the reading and interpretation of the job control cards and requested services.

Each task operates within its own Virtual memory; i.e., there is a set of relocation tables for each task. When multiprogramming among tasks, simply changing the table register causes a new set of relocation tables to be brought into use.

When looked at from the point of view of one task, the software can be thought of as having three levels as pictured in Fig. 2. There is a basic system Supervisor, which has the following characteristics:

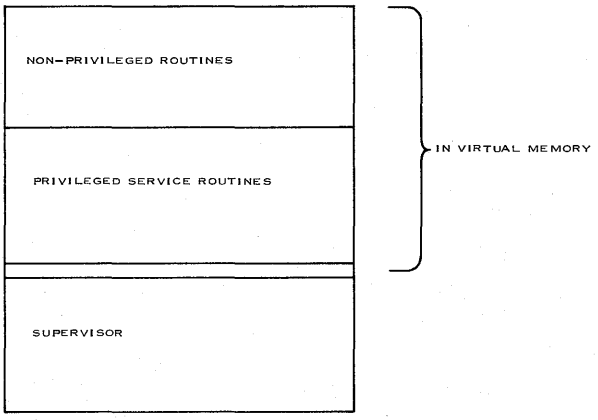



Figure 2. The software world, as seen by one task.

Permanently resident in real core
Runs non-relocated
Not addressable in Virtual Memory
Runs in Supervisor State
Not time-sliced or paged

This is the Supervisor program which is common to all tasks and all CPU's. It handles all the details associated with interrupts, I/O, paging and scheduling. Figure 3 shows the major pieces of the Supervisor, as it is now defined. Without discussing each one, a few comments will clarify the operation:

1. The Supervisor is basically an interrupt handler. System 360 hardware automatically sorts out five types of interrupts (called I/O, Program, SVC, External and Machine Check). The Supervisor must sort each of these types into finer categories and cause the appropriate actions. As a general rule of operation, interrupts are stacked by the Supervisor; i.e., if the Supervisor is pro-

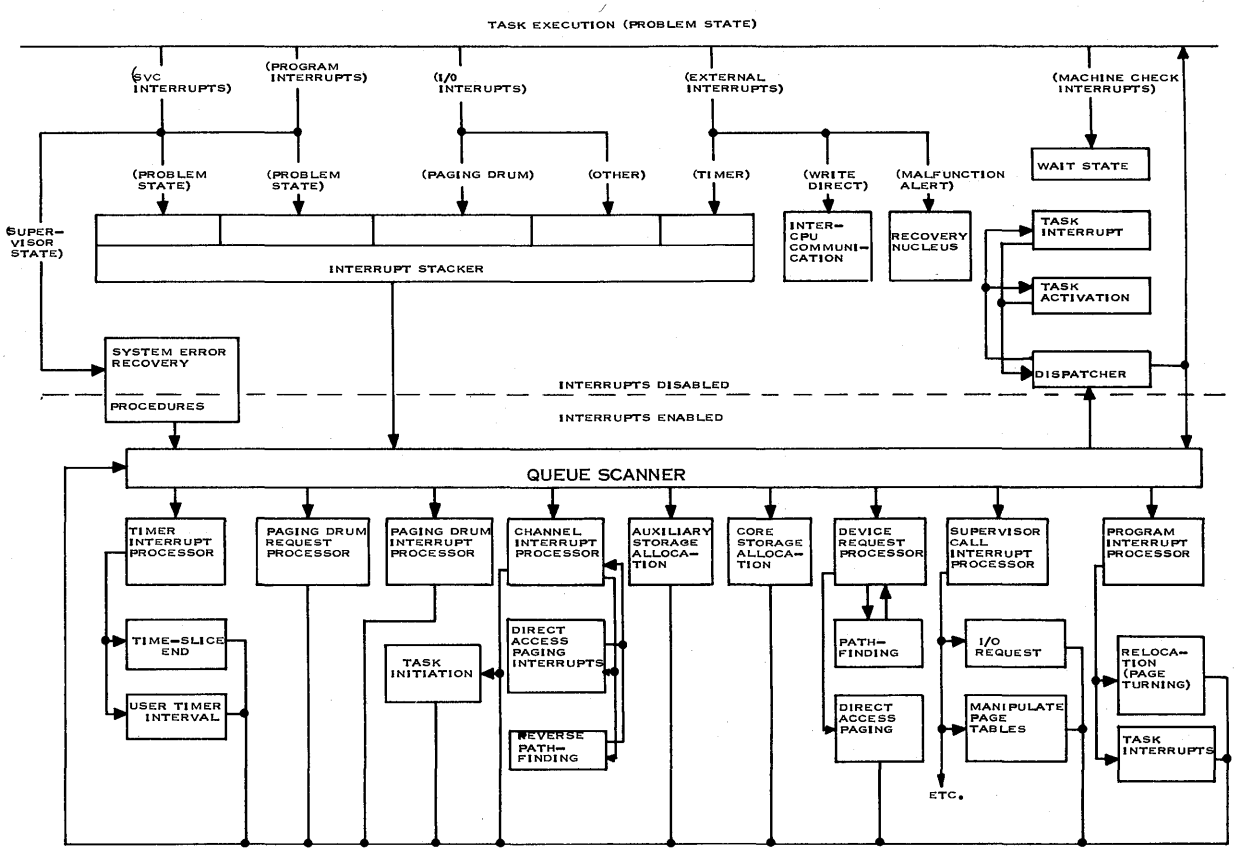



Figure 3. Supervisor block diagram.

cessing one interrupt when a second one occurs, the second one will be put on a queue until processing of the first is complete. All such pending interrupts will be processed (if possible) before returning to one of the tasts in Virtual Memory.

2. Unfortunately, there are many situations within the Supervisor where requests are made for services or facilities which are busy on other things. Whenever this situation occurs, it is necessary to form a queue of some sort. In order to handle queued requests in a reasonably uniform way, all such queues are controlled at a central point. There is a queue provided for each type of hardware interrupt (which is how interrupts are stacked), as well as a queue for each facility (such as real core storage, drum space, and I/O device use) which might be busy when a request is made. A generalized queue scanner is then provided to see to it that whenever any CPU is in the Supervisor, it will handle all outstanding serviceable queued items.
3. The major portion of what is generally called the scheduling algorithm is built into the Dispatcher. Since everyone seems to have his own idea about what is a good scheduling algorithm, its most important characteristic in this system is to make it as modular as possible so as to be easily changeable, or even replaceable. Rather than outline a specific technique, it will suffice here to list the conflicting objectives which such an algorithm must attempt to satisfy.

(a) Provide "reasonable" response at a terminal to a request for a "reasonable" amount of work

(b) Provide good throughput on batch-type, non-conversational tasks.

(c) Provide some degree of balance between (a) and (b), based upon relative loads.
(d) Utilize the paging concept as efficiently as possible.
(e) Implement multiprogramming of time-shared tasks, as required by (d).
(f) Provide dynamic variation of (d) and (e) to match CPU speed with drum and disk rates.
(g) Provide some correlation with a user-oriented priority scheme.
(h) Be simple and fast.

4. When a CPU recognizes a hardware error (i.e., a Machine Check Interrupt occurs), that CPU goes into Wait State, and another CPU is alerted (via Malfunction Alert Interrupt through Direct Control). The alerted CPU then invokes a recovery procedure.
5. Because the system is designed to operate by paging, such paging I/O functions are handled separately, to make them as efficient as possible.
6. Pathfinding is the general control mechanism for I/O selection and utilization.

Within a task's Virtual Memory are a set of Service Routines, which are a part of the programming support, and which provide in effect a one-task supervisor to control the services and communications for the user with whom this task is associated. These routines will in general be reentrant, and shared among all tasks. Some of these service routines are defined to operate with a privileged action capability (which is not the same as the Supervisor State recognized by the System/360 hardware). Such routines are allowed to request special actions by the Supervisor (such as changing relocation tables), and are protected from the rest of the programs (including those of the user) within Virtual Memory. Such privileged service routines include the following:

1. The program which interprets terminal commands.
2. Some of the subprograms which carry out terminal commands.
3. All services relative to the catalog (searching, interrogating, changing, etc.)
4. Virtual Memory allocation.
5. Private device allocation.
6. Allocation of external (catalogable) space.
7. Access methods, which provide the task with GET/PUT capability to terminals, sequential, and direct access devices.
8. Dynamic loader, and associated tables.
9. Mechanism for handling task interrupts (as distinguished from hardware interrupts).

The nonprivileged area of Virtual Memory (which is by far the bulk of it) is available for the language processors (FORTRAN, PL/1, etc.) and user-generated routines.

The basis behind the three levels depicted in Fig. 2 is protection. The Supervisor is protected from accidental random damage by virtue of the fact that it is not addressable in anyone's Virtual Memory. (What is addressable in a task's Virtual Memory is determined by what the Supervisor puts into the page table for the task. Note that this also provides complete protection between different tasks.) Communication with the Supervisor is limited to a specific set of SVC's (Supervisor Calls, a System/360 interrupt mechanism for that specific purpose). However, the majority of such possible requests are of a very sensitive nature, i.e., if misused, they could seriously affect the operation of the whole system, and this is highly undesirable. To help control this problem, the set of privileged service routines was defined, and execution of the sensitive requests is limited to such service routines. In turn, access to these privileged service routines is limited to legitimate entry points, and they are protected (via System/360 protection keys) from access by non-privileged routines. In this way, it is hoped to eliminate the possibility that an undebugged or irresponsible task could hurt anyone but itself.

## SUMMARY

This report attempts to give an over-all picture of the System/360 Model 67 Time Sharing System, its system design, and major hardware and control program characteristics. The unique combination of hardware and software objectives makes a very complex problem, for which a simple and efficient solution is desired—a difficult task at best. It is further complicated by the fact that there is no recognized consistent way to measure such a system, either in how it performs or how well specific technical problems have been solved.

It is the author's opinion that one very significant concept made available in this system is the large addressable Virtual Memory. It should force a complete reevaluation of how programs should be written, and has the potentiality of making obsolete the traditional I/O techniques. However, it will require a good deal of experience and experimentation to know how best to exploit this new facility.

## REFERENCES

1. H. A. Kinslow, "The Time-Sharing Monitor System," *Proc. FJCC 1964,* Spartan Books, Inc., Washington, D.C., 1964.

2. T. M. Dunn and J. H. Morrissey, "Remote Computing—An Experimental System. Part 1: External Specifications," *Proc. SJCC 1964,* Spartan Books, Inc., Washington, D.C., 1964.

3. J. M. Keller, E. C. Strum and G. H. Yang, "Remote Computing—An Experimental System. Part 2: Internal Design," *Proc. SJCC 1964,* Spartan Books, Inc., Washington, D.C., 1964.

4. "IBM System/360 Principles of Operation," IBM Document, Form A22-6821-1.

# DESIGN CONSIDERATIONS FOR A 25-NANOSECOND TUNNEL DIODE MEMORY

D. J. Crawford, R. L. Moore, J. A. Parisi,
J. K. Picciano, and W. D. Pricer
*International Business Machines Corporation*
*Systems Development Division*
*Poughkeepsie, New York*

## INTRODUCTION

About two years ago a tunnel diode memory system was described which employed substantially different techniques than those previously used.[1] Although earlier systems had tended towards array arrangements that had the storage cells connected in parallel on one or more axes, the new system employed series connections along two axes. This new arrangement has several design and performance advantages compared to previous systems. The original paper described the basic approach and some of the earlier work which included the design of array cross sections and the associated driving and sensing circuits. Since that time one version of the system has been operational in two IBM 7030 systems,[2] and a 16-word, fully-populated, higher-speed laboratory model was built and reported.[3] The present paper describes the engineering considerations used in the design of a larger and faster memory employing the basic techniques.

The new memory system contains 64 words of 48 bits each, and test results from a partially-populated cross-sectional model indicate a complete READ/RESTORE or a CLEAR/WRITE cycle time of less then 25 nanoseconds. A fully-populated complete memory system is in the final stages of construction and assembly.

## CELL OPERATION

The basic storage cell is simple, as shown in the dashed-line box in Fig. 1. It consists of a tunnel diode shunted by a series-connected load resistor and the secondary winding of a transformer. A biasing current is introduced to the tunnel diode

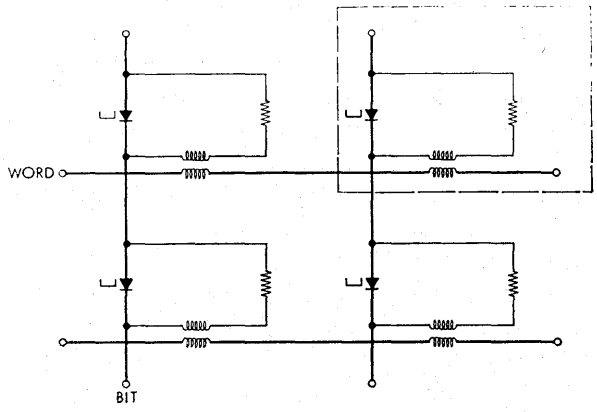



Figure 1. Array configuration.

storage cell along the bit axis. During the writing portion of a memory cycle, the biasing current can be increased by the addition of a bit current. The word driver introduces voltage excursions within the storage cell loop by means of the transformer. The normal bias current through the tunnel diode cell is such that the tunnel diode can be either in a high-voltage region or in a low-voltage region.

The cells are series-connected along two axes (Fig. 1) in order to form the basic memory array configuration. Figure 2 shows the load-line diagram for the basic storage cell. The current bias and the load resistor are such that the cell normally has two stable states. During the READ operation, if the cell has been storing a ONE, the voltage induced into the secondary winding of the cell transformer

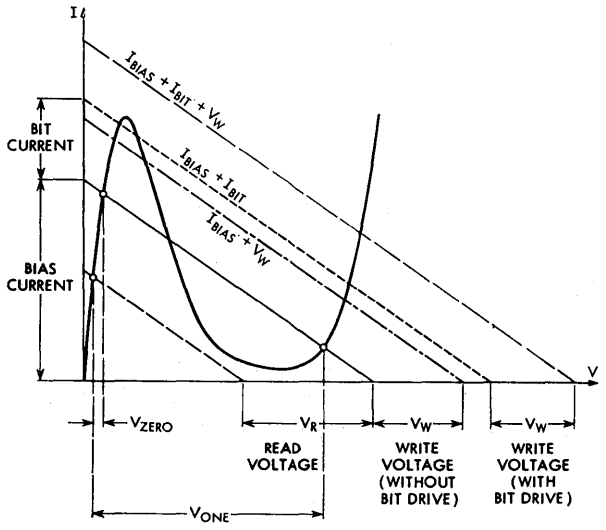



Figure 2. Tunnel diode load-line diagram.

shifts the load-line and clears the cell from the high-voltage state to the low-voltage state. The net voltage drop when the cell changes its operating point from the high-voltage to the low-voltage state is transmitted through the series connection of the diodes on the bit line to the end of the bit line, where it can be detected.

On the other hand, if the cell was storing a ZERO, it initially would be in the low-voltage state. The shifting of the load line in this case by the READ voltage, $V_r$, produces only a very small voltage as the ZERO response. At the conclusion of the READ cycle, all the cells associated with the particular word will have had their information read out and will be left in the low-voltage state.

## ARRAY DESIGN CONSIDERATIONS

One of the key items in the design of the system is the cell transformer. The objectives are to achieve a low impedance when looking into the primary loops, and to have reasonable inductive coupling but a minimum of capacitive coupling from primary to the secondary loop. In the earliest work, etched circuit construction was tried but the state-of-the-art of fine-line etching and the difficulty of making good minature connections made the approach impractical at that time. The succeeding systems employed transformers with secondary windings made of wire. When the work on this new system was started, it was decided to again attempt to solve the various problems of the etched circuit transformer for the inherent advantages of reproducibility it offered.

The first step in the design process was to study electric field patterns of likely configurations using resistance paper analog techniques. From this work, reasonably accurate predictions of the transformer parameters such as mutual coupling and secondary self-inductance were made. This procedure permitted a quick review and optimization of different transformer patterns. A typical field plot used is shown in Fig. 3. As a result of this work, preliminary decisions were made as to the desired shape and dimensions of the transformer.

At this time different constructional methods and arrangements of the storage cell into arrays were considered from both a mechanical and electrical viewpoint. It was finally decided that an approach which had several cells on a module would simplify manufacturing problems and improve serviceability. Although it offered certain mechanical constraints, the SLT (Solid Logic Technology) type of module employed by IBM was chosen as a starting point for the design.

An alumina ceramic wafer about one-half-inch square serves as the main mechanical structure for the module. The ceremic wafer has 16 swaged pins on 0.125-inch centers for external connections plus 6 pins in the opposite direction to serve as mounting points for welding the tunnel diodes. One surface of the ceramic has 4 screened resistors and a solder-coated circuit pattern. A view of this subassembly is shown in Fig. 4. The transformers are contained in a separate multilayer etched copper wafer assembly that is slipped over the 16 pins and dip-soldered. The final assembly operation con-

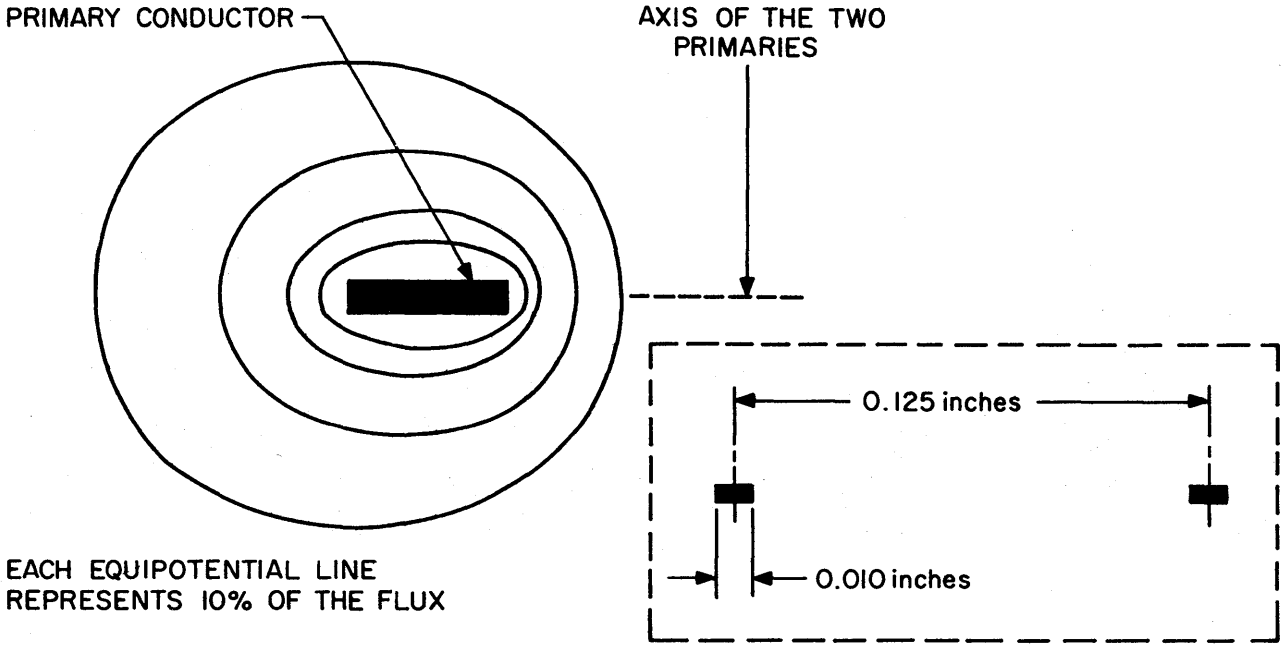



Figure 3. Magnetic field pattern around primary conductor.

sists of applying a ferrite powder coating in an organic binder on the primary side of the transformer wafer to increase the mutual coupling from primary to secondary. The assembled modules are shown in Fig. 5.

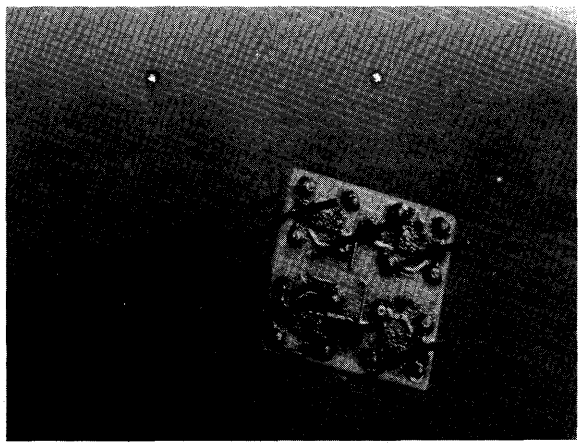
Early in the project, it was recognized that a large, accurately scaled mockup of the module assembly was needed for making electrical measurements. These measurements were needed to assist in the transformer design, and were also vital to the determination of the overall array parameters. The normal-size modules were so small that it was virtually impossible to make electrical measurements with any degree of accuracy of precision. Because both capacitance and inductance scale are directly proportional to linear dimensions, it was decided to make a module assembly 20 times normal size. One problem was to find a suitable dielectric material substitute for the ceramic substrate because large ceramic pieces were not available. The solution was found by loading an epxy resin mixture with titanium dioxide powder in a ratio of approximately 1:1 by weight to achieve a dielectric constant of 9.4. Analog field plots showed that thick copper wiring patterns could be simulated by using two thinner patterns appropriately spaced and connected in parallel. Figure 6 shows photographs of the large module and some of the transformer patterns.

The final transformer patterns are shown in Fig. 7. The solid squares are lands used for connecting points. To cancel capacitive coupling effects from the primary to the secondary wiring, each transformer uses two primary wires which are driven push-pull with respect to ground. The primary wires are routed in a manner to maximize the mutual coupling to the secondary wires, and the secondary wires are arranged in a mirror image configuration to match the land pattern to obtain uniform characteristics. Capacitance stub "fingers" are included to balance the capacitances between the primary lands and the secondary windings.

A cross section of a transformer wafer is shown in Fig. 8. To minimize the capacitance, the primary and the secondary wires are separated by Teflon® (registered trademark of E. I. Du Pont de Nemours and Co.). The outer layers are used as supports for a land pattern for soldering to the pins, and also to protect the transformer pattern. Through-hole plating techniques are used to connect the inner land patterns with the outer lands.

## BIT-LINE CHARACTERISTICS

The bit line used in the tunnel diode memory consists essentially of a number of tunnel diodes connected in series. The inductance of the interconnections is the series inductance of the line, whereas the transformer and module capacitance is the major component of the shunt capacitance. However, the tunnel diodes add a nonlinear series



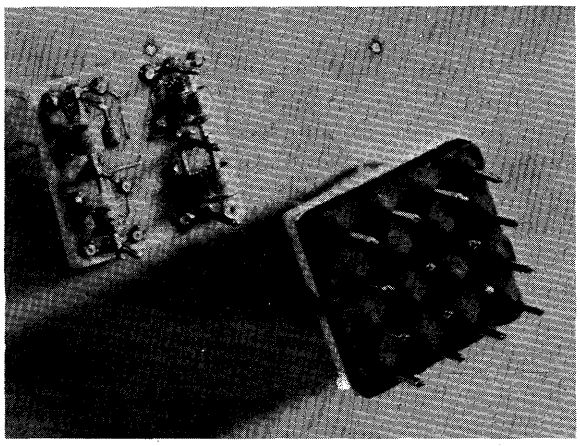


Figure 4. Array modules: (*a*) module with resistors and pins; (*b*) top and bottom views of array modules.

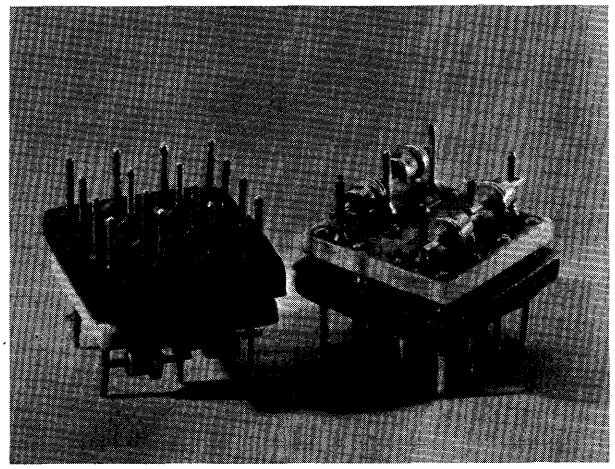
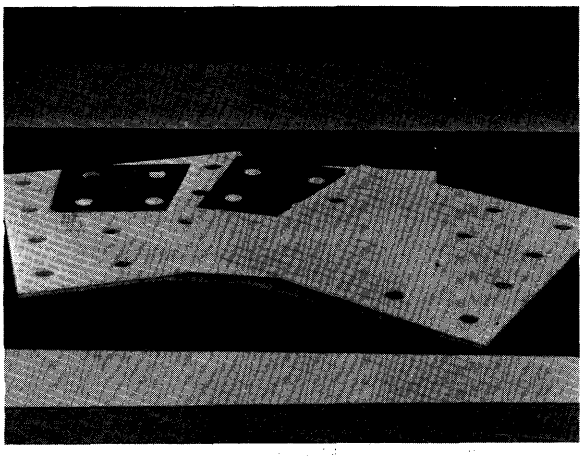


Figure 5. Assembled memory array modules.



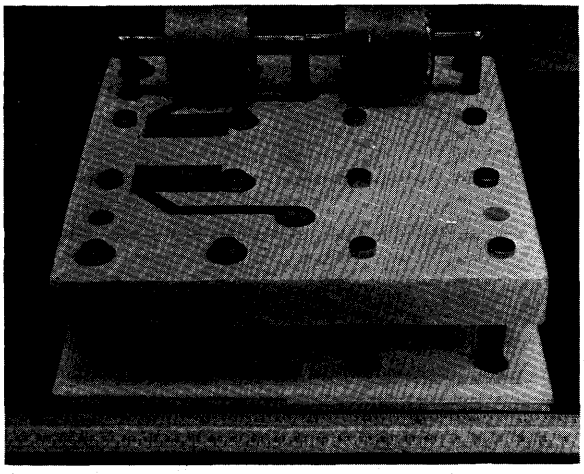


Figure 6. (*a*) Transformer wafer sheets, 20 times normal size; (*b*) memory module, 20 times normal size.

resistance component that predominantly affects the response of the line. An equivalent circuit for the line is shown in Fig. 9 where each section represents one bit of the line. $L_S$ is the series inductance; $C_d$ is the diode junction capacitance; $R_d$ is the nonlinear diode resistance; and $C_S$ is the total shunt capacitance. Because each of these parameters is very small and the current rise times desired are very fast, accurate measurements necessary to optimize the line design could not be made.

A program was written to calculate the response of the bit line that utilized this equivalent circuit and took the nonlinearities of the tunnel diode characteristics into account. By using this program, the

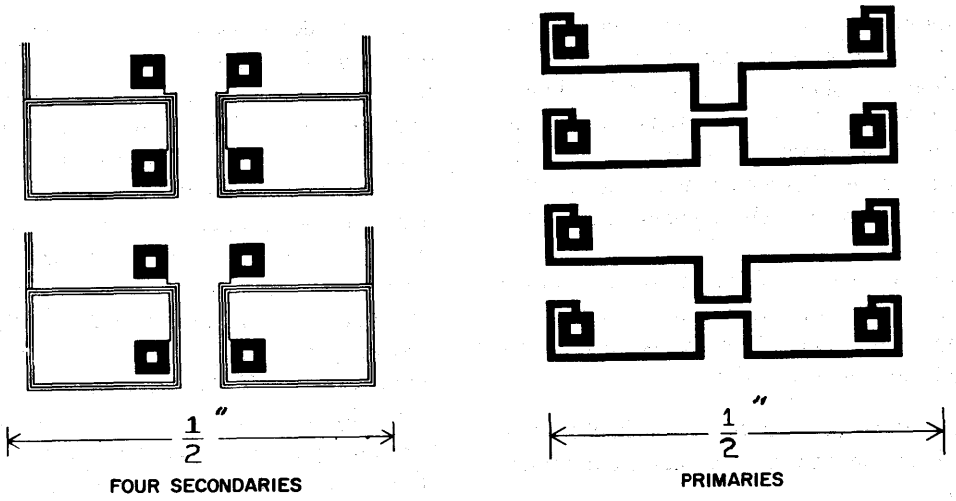



Figure 7. Transformer pattern.

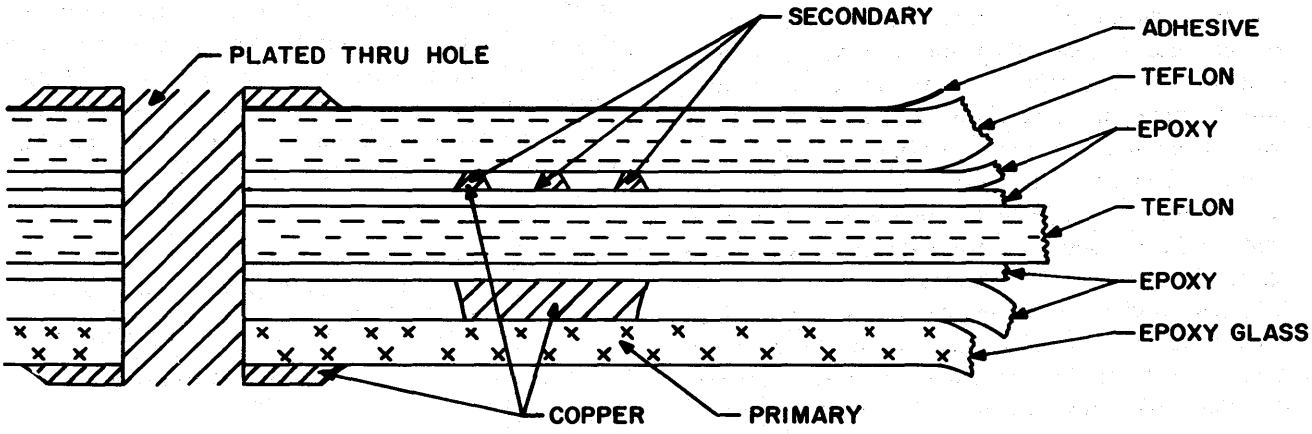



Figure 8. Transformer wafer cross section.

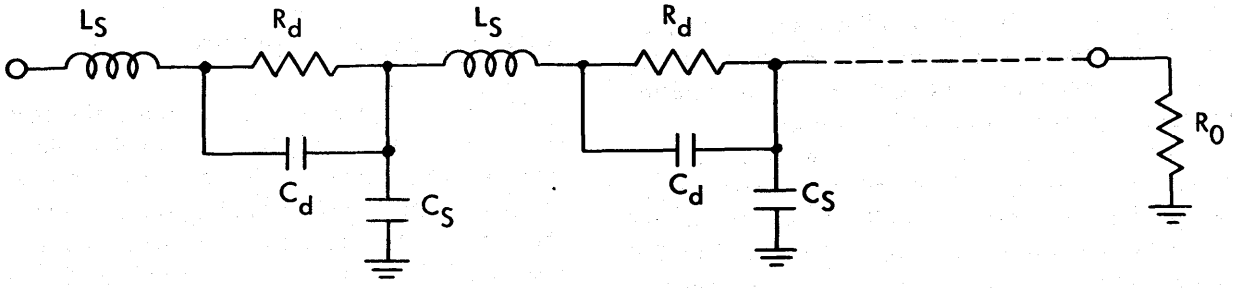



Figure 9. Bit-line equivalent circuit.

parameters of the line were varied to find their effect on the response and to find an optimum termination. The range of parameters used was determined by measurements of the 20-times module model. The results of the investigation led to the following conclusions:

- Increasing the series inductance of the line does not improve the response of the line; that is, in spite of the fact that the line becomes less lossy as the series inductance is increased, the sum of the rise time plus the delay time increases with increasing inductance.
- The parameters that most strongly affect the response of the line are the diode resistance, the shunt capacitance, and the termination resistance, in this order.

- The optimum termination resistance, chosen with the criterion of minimum rise time with negligible overshoot, is considerably below the $\sqrt{L/C}$ for the line. Reflections due to this lower terminating resistance tend to increase the bit drive current at the far end of the array and thereby compensate for the attenuation.
- Because of the nonlinearity of the diode resistance, the fastest response times are obtained when the bit drive is toward increased current in the diode.

Actual measurements verified these predictions but could not provide the accurate resolution obtained in calculations.

Two types of crosstalk were investigated. One was the inducement of a false sense signal on a bit line because of a switching tunnel diode on an adjacent bit line. The other type was accidental writing into a tunnel diode receiving full word current by way of mutual inductive coupling between transformer secondaries. The primary source of a false sense signal is the capacitance between cells on adjacent bit lines; these transformers are only 50 mils apart. The value of the capacitance predicted by the 20-times module (0.31 picofarad) is slightly high because no ground plane was used in the measurement. In order to find the worst case, it was assumed that the capacitance of eight cells are all lumped together at a point and that the voltage rises in one L/R time constant. Under these conditions, it is predicted that a maximum of 96 millivolts would be produced from the two adjacent bit lines. However, the actual noise should be much smaller and the detector rejection level is safely above 96 millivolts.

Calculations were made of the worst-case noise current induced by mutual inductive coupling between adjacent transformer coils. They showed that the peak induced current would be less than 2.5 percent of the diode peak current. This value is small enough to have a negligible effect on operating tolerances.

## WORD LINE STUDY

The ferrite material on the primary winding of the transformer increases the characteristic impedance and delay of the line compared to a similar line without the ferrite. In addition, the ferrite increases the high-frequency losses and results in rise-time deterioration as the word pulse travels down the line. To maintain a reasonable variation in rise time, the word lines are split into 24-bit segments, with 2 segments being driven by the 2 output stages of a word-driver circuit.

A number of methods of maintaining independence of the secondary output current from primary current rise-time variations were investigated. These involved shorting the word line or inserting a rise-time pad in the output of the word-driver circuit. An investigation of the response of the transformer itself showed that secondary inductance and load resistance caused it to act as a reasonable rise-time pad to a fast word-drive pulse. Fig. 10 shows the method of drive presently employed. The line is terminated in its characteristic impedance, and the pulse is clamped at the driver to control the pulse amplitude. The transformer secondary circuit is used as a rise-time pad, with the result that a 20 percent change in the primary current rise time result in only a 10 percent change in the secondary output voltage amplitude.

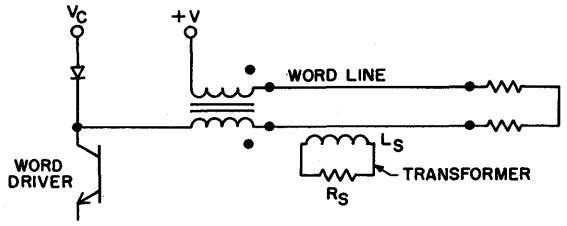



Figure 10. Word drive system.

## ARRAY CHARACTERISTICS

The array is formed by mounting the memory cell modules on a pluggable card which contains the interconnection pattern. The card has etched wiring on the two outer surfaces and a ground plane inside the laminate In addition to the memory cell modules, the array card also contains all the word-driver circuits, word line termination networks, bit line termination networks, and miscellaneous power supply decoupling networks associated with the modules. Each card contains 8 words, 48 bits per word, of storage. An assembled array card is shown in Fig. 11.

One of the advantages of the series-type array is that the driving requirements are readily met with high-speed transistors. The array described utilizes a unidirectional current pulse of 88 milliamps into the word line and presents a load of

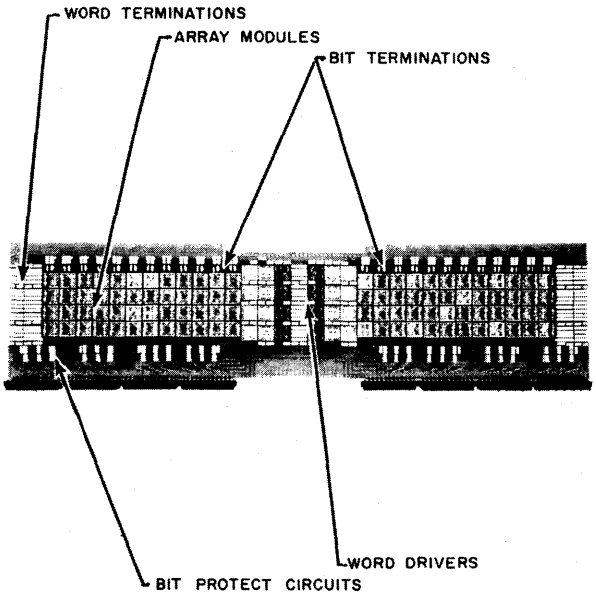



Figure 11. Array card layout.

136 ohms to the word driver. As previously explained, each word line is split into 24-bit segments, with a pair of segments driven by one word-driver circuit. The propagation delay from the word line input terminals to the far end of each line is about 3 nanoseconds. In contrast to the word line, the bit line is much more like a distributed RC-line than a conventional low-loss transmission line and, therefore, exhibits high phase and frequency distortion. To minimize delay, the bit line is broken into 8-bit segments, with a pair of segments handled by each information control (regeneration) circuit. The bit drive required is only 8.27 milliamps and is also a unidirectional pulse into a load impedance of about 50 ohms. Output signals for sensing are in the range of 300 to 400 millivolts, although the detector sensing level is usually set lower to save cycle time. The maximum quiescent power dissipated in a memory cell is 4.5 milliwatts which leads to an overall maximum standby power dissipation in the 64 by 48 array (exclusive of associated circuits) of about 14 watts.

## CIRCUIT DESIGN

### *Information Control Circuit*

Figure 12 shows the information control circuit diagram. The sense amplifier, detector, information control logic, bit-line bias current, and bit driver were included in this single circuit to shorten regeneration time and to simplify the circuitry.

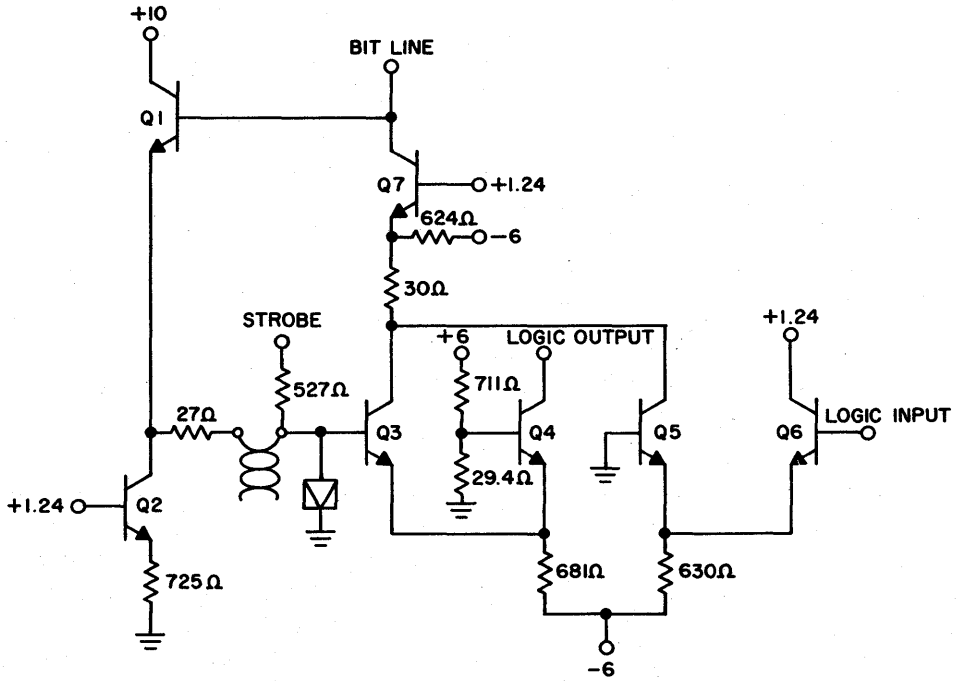



Figure 12. Information control circuit.

The operation of the circuit is as follows: The first transistor, $Q_1$, connected as an emitter follower, receives a sense signal at its base and drives a transmission line that is open-circuited at the far end.

This passes a pulse of a width of approximately 3 nanoseconds. If the strobe input is positive at this time, the tunnel diode switches on the first 200 millivolts of the signal. The detector diode remains in the high state as long as the strobe pulse is present. This drives the current switch ($Q_3 + Q_4$) that supplies the bit current pulse to the bit line. The bit drive may also be actuated by pulsing the logic input terminal negatively ($Q_6$). The bit-driver collector current increase of the output transistor rises in 2 nanoseconds or less, and the entire line assumes the full drive amplitude within 7 nanoseconds of bit-driver turn-on. No overshoot has been observed. In addition to furnishing the bit pulse, the output transistor also supplies the DC biasing current for the bit line. Throughput time of the ICC (Information Control Circuit) from sense-amplifier input to bit-driver turn-on is 2 to 3 nanoseconds. The total time elapsed from the resetting of the tunnel diode to its zero state until the restoration of that diode to its ONE state is 10 to 11 nanoseconds. During the write noise, the first transistor cuts off, thereby preventing the write noise from influencing the state of the tunnel diode detector.

## WORD DRIVER

The word driver circuit is shown in Fig. 13. The operation of the circuit is as follows: The current switch formed by $Q_1$, $Q_2$, and $Q_3$ performs the logical AND function and provides regenerative feedback through $C_1$ to improve the rise time of the input pulse; $Q_4$ provides the necessary current gain to drive

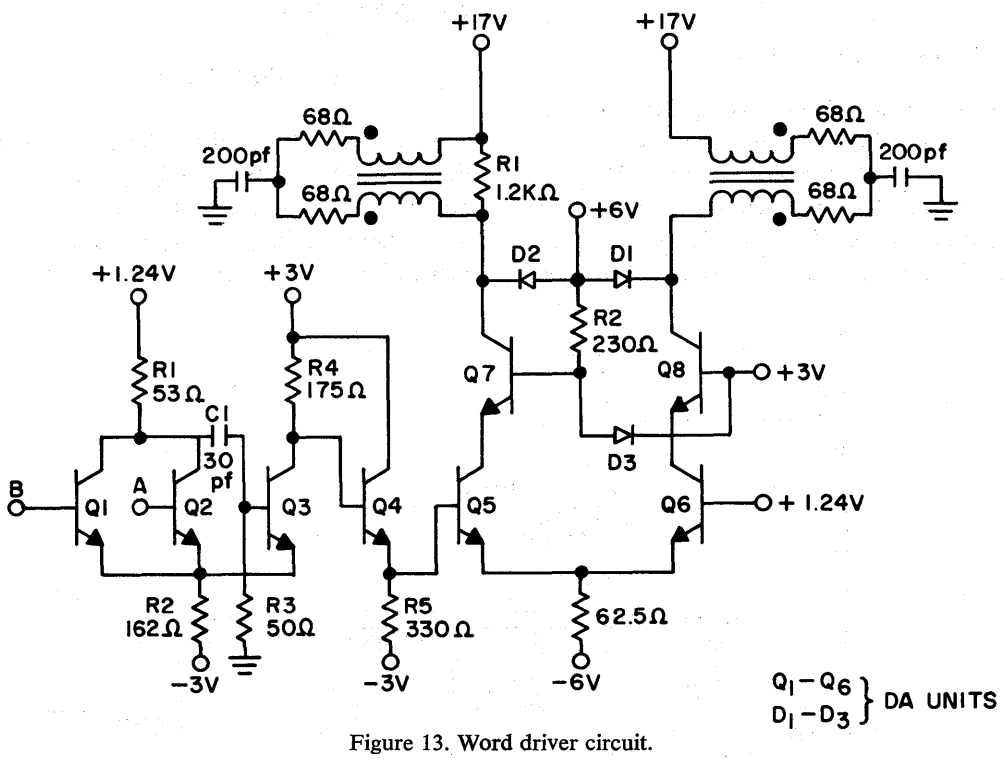
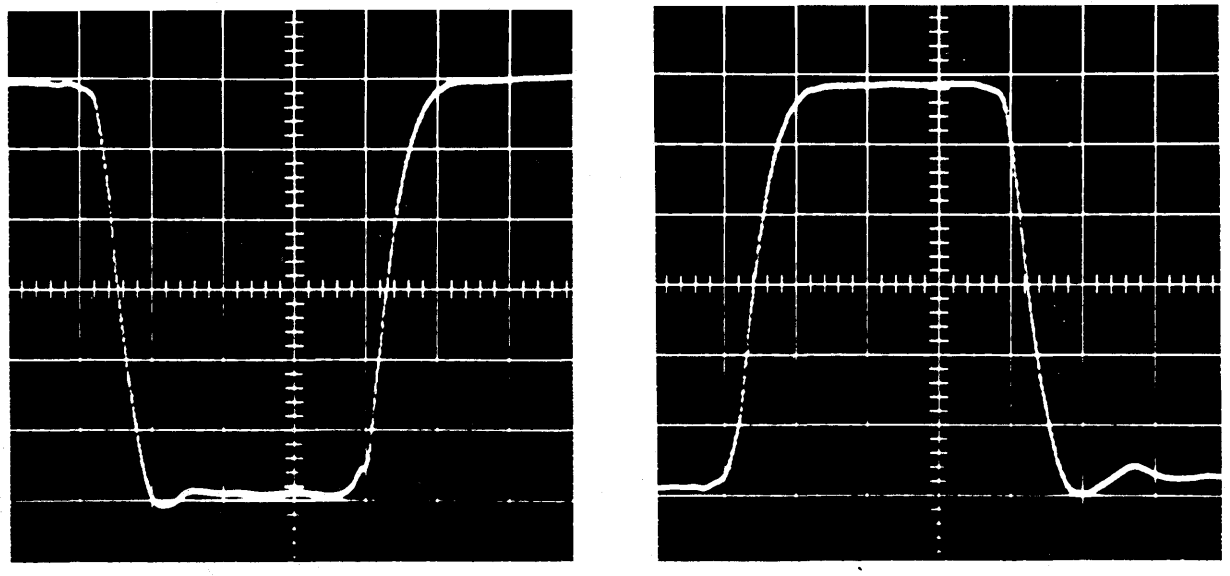



Figure 13. Word driver circuit.

the output stages. In this circuit, both outputs from the current switch are utilized to drive the word line in two segments. The output stage consists of the current switch formed by $Q_5$ and $Q_6$, which drives two 2N2369A ($Q_7 + Q_8$) transistors for increased voltage gain. The output levels are clamped by the diodes $D_1$ and $D_2$ to provide a well-controlled output level. The two wires of each word line segment are driven push-pull with respect to ground by a balun transformer. Because of the relatively light drive power requirements, high-speed current switch techniques can be used. It should be noted that half the drive is left ON all the time because only the changes in drive line current can activate the tunnel diode cells through the transformers.

The word driver produces an output pulse of 88 milliamps into a 136-ohm load. The current amplitude stays within a tolerance of ±10 percent under worst-case conditions. The rise and fall times of the pulse are nominally 3 nanoseconds, 10 to 90 percent.

The worst-case tolerance on the write transition is within ±15 percent of nominal value and the read transition is at least as fast. The waveforms of the two outputs of the word driver are shown in Fig. 14.



a. In phase output.

b. Out of phase output.

(Horizontal = 5 ns/div and vertical = 2v/div. $R_L$ = 136 ohms.)

Figure 14. Final word driver output waveforms.

## OVERALL SYSTEM DESCRIPTION

In addition to the memory array and the associated driving and sensing circuits, the complete system includes a binary address register with decoding circuits, data input gating circuits, a data output register, and a clock with timing pulse distribution circuits. An address counter, comparing circuits, and manual data input switches permit exercising the memory system without using any external connections. Either the internal clock, or an external source of clock pulses may be used. The logic circuits are constructed of an advanced form of IBM solid logic technology, referred to as ACPX in papers presented previously.[4,5]

The system interwiring and voltage distribution is contained in 2 multiple-layer boards, each about 10 by 14 inches, mounted side by side. These boards have male pins protruding from them which serve as connecting points for external wiring and also receive the sockets mounted on the bottom of each circuit card. The memory array cards are about 18 inches long and plug into both board assemblies, bridging across the gap between them. Except for one narrower card, all the other circuit cards are about 3 inches wide and 5 inches high. About two-thirds of the volume is used by the memory array and the information control circuits; the remainder is used by the various logic circuits. A sketch of the layout is shown in Fig. 15. The card and board assemblies, blower system, power supplies, control panel, and I/O connection panel are contained in a cabinet.

Tests on a cross-sectional model containing 2 populated word lines and 4 populated bit lines indicated good operating margins with a cycle time in the range of 22 to 24 nanoseconds. Assembled array cards, shown in Fig. 11, have also been tested, with excellent operating margins verifying the validity of the cross-section test data. Fig. 16 shows the actual voltage waveforms from the cross-section model. At the time this is being written, the complete system is under construction; a photograph of the partially completed system is shown in Fig. 17.

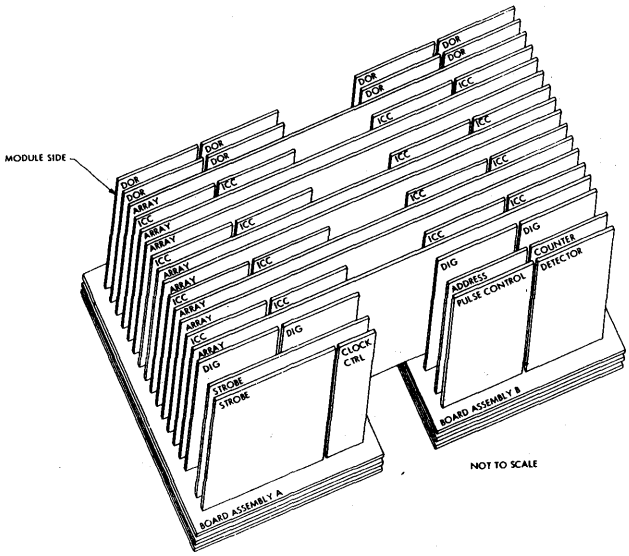



Figure 15. System layout sketch.

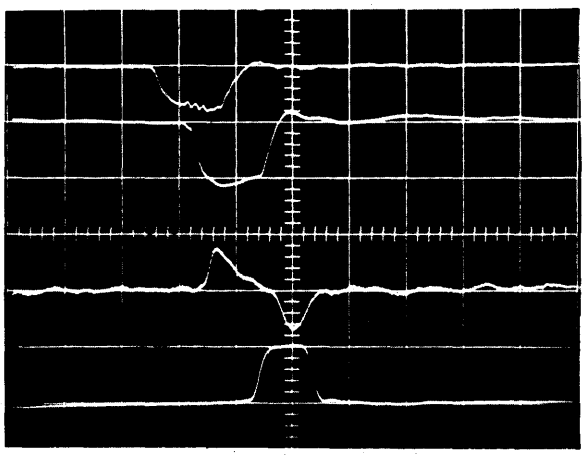


Figure 16. Cross-sectional model memory circuit timing waveforms (10ns/cm). The traces from top to bottom are as follows: (*1*) Input to word driver. (*2*) Driven end of word line. (*3*) Transformer output of the cell at the far end of the word line. (*4*) ICC output.

This represents the first complete memory system using any type of technology reported in this size and speed range.

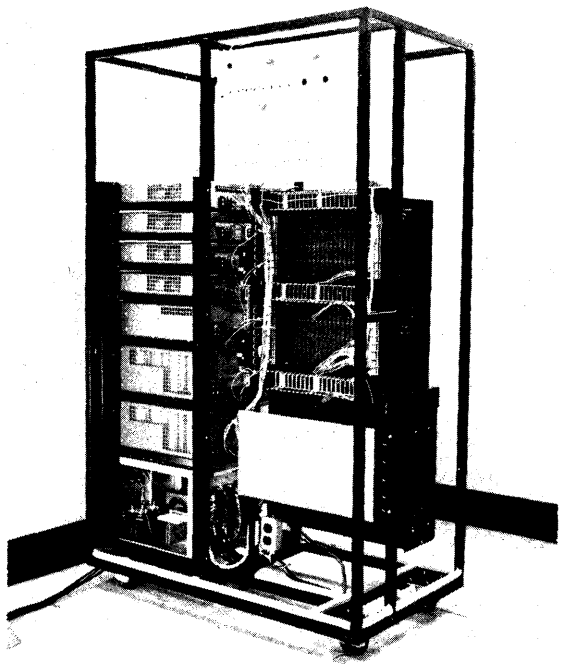


Figure 17. Partially assembled memory system.

## REFERENCES

1. D. J. Crawford, W. D. Pricer and J. J. Zasio, "An Improved Tunnel Diode Memory System," *IBM Journal of Research and Development,* vol. 7, pp. 199-206 (July 1963).

2. "Series-Coupled Tunnel Diode Memory," *Computer Design,* vol. 2, no. 10, p. 8 (Nov. 1963).

3. G. Feth, "Scratch-pad Tunnel Diode Memory," presented at the IEEE Workshop on Computer Memories at UCLA Conference Center, Lake Arrowhead, September 10-12, 1964.

4. G. M. Amdahl, T. C. Chen and C. J. Conti, "IBM System/360 Model 92," *AFIPS Conference Proceedings, Fall Joint Computer Conference, 1964,* Spartan Books, Inc., Washington, D.C., 1965, vol. 26, part II, pp. 69-96.

5. M. J. Flynn, "Engineering Aspects of Large High-Speed Computer Design," presented at the ONR Symposium on High-Speed Computer Hardware, Washington, D.C., November, 1964.

# A SILICON MONOLITHIC MEMORY UTILIZING A NEW STORAGE ELEMENT

Richard Shively
*Litton Industries*
*Woodland Hills, California*

## INTRODUCTION

The advances that have been made recently in monolithic chip semiconductor logic circuits have significantly contributed toward the development of high-speed, low-power computers. These advances also emphasize the need for marked improvements in storage techniques if the operating speeds, the weight, and the electrical power of future computers, especially airborne or spaceborne computers, are not to be adversely affected by the computer memory.

A machine organization that utilizes several small "scratchpad" memories can be used to advantage in providing the required random access storage. This technique combines a number of small-capacity, high-speed memories with a large-capacity, slow-speed working store. A significant increase in overall machine speed can be realized by the proper application of this combined memory technique.

If the high-speed memory is to operate at a cycle time in the 100-nanosecond region, the class of storage elements that can be used is somewhat limited. Storage elements capable of switching speeds compatible with 100-nanosecond cycle times include (*a*) thin magnetic films of several types, (*b*) some forms of laminated ferrites, (*c*) tunnel diodes, and (*d*) semiconductor flip-flop type devices.

Most forms of thin film and laminated ferrite storage devices exhibit adequately fast switching times, but the drive current requirements are large and the readout signals small. The current amplitudes and driver power levels usually preclude the use of integrated circuit drivers for the thin film and laminated ferrite memory systems. Recent development work indicates that integrated circuit sense amplifiers suitable for sensing low-level signals at high speed will soon be available. However, it is felt that the cost of these amplifiers may adversely effect the cost per bit of a small memory system for some time to come.

Tunnel diode memory elements are capable of fast switching speeds, and the drive and sense circuit requirements are amenable to mechanization via the use of integrated circuit techniques. The main stumbling block in fabricating a small, low-power tunnel diode memory is found in the tunnel diode itself. Although the tunnel diode is a semiconductor device, the unique characteristics of the element have made it quite difficult to fabricate in monolithic form. Therefore, the prospects of a low-cost, physically small tunnel diode memory are not good.

The proper application of integrated circuit techniques to the fabrication of a bistable multivibrator

or flip-flop appears to be, at this time, the most practical method for building a high-speed, low-power, physically small and potentially low-cost scratchpad memory. Most standard integrated circuit flip-flops can be used as random access memory devices; however, the power per bit is high, being in the order of 40 milliwatts and up, and the number of bits per chip that can be obtained by using commercially available flip-flops even with new metalization masks is felt to be no larger than 3 bits per chip.

Efforts have been made to tailor the flip-flop topology to the needs of a random access memory device. By incorporating this philosophy, improvements in power/bit and packing density have been reported. A device power of 30 milliwatts per bit and 8 bits per 14-lead flat package has been reported.[1]

The storage device that is presented in this paper represents a combination of advanced semiconductor technology and a unique flip-flop type of topology that has been optimized for random access memory usage. Because of the fact that the storage device is constructed using monolithic integrated circuit technology, the name Semiconductor Memory Integrated Device, or SMID, has been used. It is felt that the SMID represents a significant addition to existing storage devices. The basic element can be used at read cycle times of less than 100 nanoseconds over the temperature range of –55°C to +125°C, and the power/bit is less than 0.55 milliwatt.

Nine bits of SMID storage has been fabricated on a monolithic chip. Figure 1 shows a 9-bit memory array. When the fact that the 9-bit array and all of the memory drive electronics are batch fabricated is coupled with other element characteristics, there is the indication that the SMID will provide a vehicle for the construction of fast, low-power and low-cost scratchpad memories.

## THE BASIC STORAGE ELEMENT

A fundamental requirement of any binary memory element is that the element exhibit two distinct stable states. It is possible to form a bistable element by interconnecting a PNP and NPN transistor in the proper manner. By utilizing this fact, the semiconductor industry has been manufacturing, for a number of years, bistable elements that are known by various names.[2,3] In addition, several families of monolithic integrated circuits, usually linear circuits by design, have exhibited, much to the dismay of the manufacturer and the user, a four layer (PNPN) latching mode of operation. However, the proper utilization of this four-layer latching mode of operation has afforded a vehicle for the construction of the SMID element.

The basic SMID element is shown in Figure 2(*a*). The conductance or nonconductance of the transistor latching pair $Q_1$ and $Q_2$ constitutes the fundamental storage mechanism. When the latching pair is in the conducting state the SMID element is said to be storing a "one."

The method by which the storage of data is accomplished can best be understood by considering the static operation of transistors $Q_1$ and $Q_2$. Transistors $Q_1$ and $Q_2$ are interconnected to form a positive feedback loop. The gain of this positive feedback loop is approximately $\beta_{Q1} \times \beta_{Q2}$, the product of the current gains of the two transistors. The stable states of the transistor latching pair are represented in Fig. 3 as point *A*, the "zero" state, point *C* the "one" state, and point *B*. Point *B* is a stable state that will exist only during a read "one" or write "one" operation. If a "one" is to be written into a storage element, base current must flow into transistor $Q_3$. $Q_3$ will conduct into saturation causing the latching transistor pair to be switched to point *B*. The load line shown in Fig. 3 is determined by resistor $R_2$. Once a SMID element has been switched to point *B*, the external drive can be removed. The voltage across the SMID element will now be $+V_H$ and the transistor latching pair will be operating at point *C* of Fig. 3 due to the self-regenerative action in the transistor latching pair.

If the current that conducts into the latching pair is equal to or greater than the value given by Eq. (1), the memory element will remain in the low impedance or "one" state. The maximum hold-on current as a function of temperature has been plotted in Fig. 4 from empirical data and agrees with the hold-on current as calculated from Eq. (1):

$$I_{eQ1} = \frac{\beta_{2\,\min}\,(1 + \beta_{1\,\min})}{\beta_{2\,\min}\,\beta_{1\,\min} - 1} \times \frac{V_{BEQ2\,\max} + V_{CE\,sat\,\text{WORD DRIVER}} - (-V_{\max})}{R_{1\,\min}} \quad (1)$$

where

$\beta_{2\,\min}$ = the minimum beta of transistor $Q_2$,
$\beta_{1\,\min}$ = the minimum beta of transistor $Q_1$,
$V_{BEQ2\,\max}$ = the maximum base to emitter drop of transistor $Q_2$,

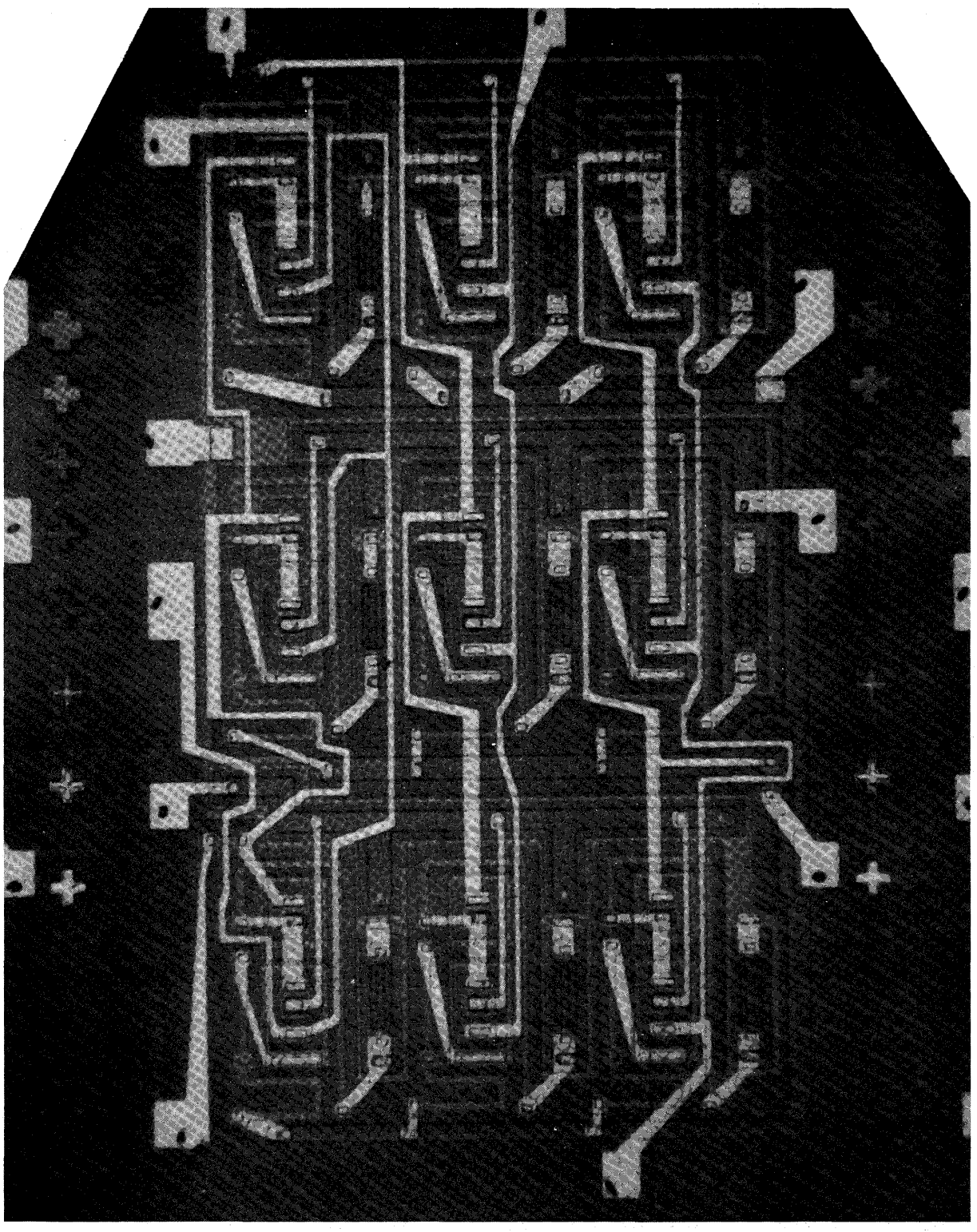


Figure 1. Nine-bit memory array.

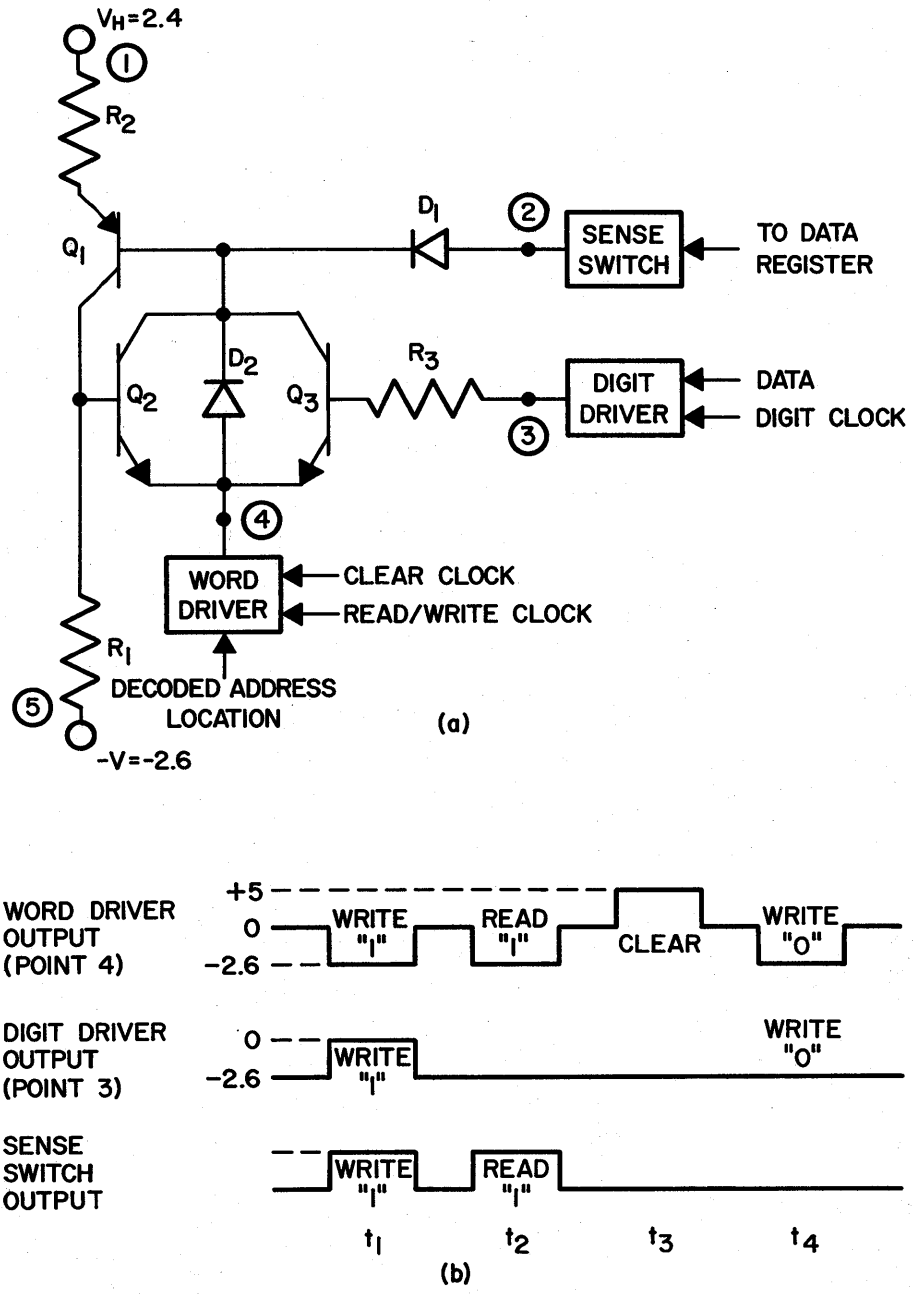



Figure 2. Basic SMID element and typical timing waveforms.

$V_{CE\ sat\ \text{WORD DRIVER}}$ = the collector-to-emitter saturation voltage of the word driver output transistor,

$-V_{max}$ = the maximum negative value of $-V$ volts (2.86 volts), and

$R_{1\ min}$ = the minimum resistance of resistor $R_1$.

The SMID element will remain in the "one" state until the current flowing in the latching pair is reduced to a value that is less than that given by Eq. (1).

The following discussion is included so as to indi-

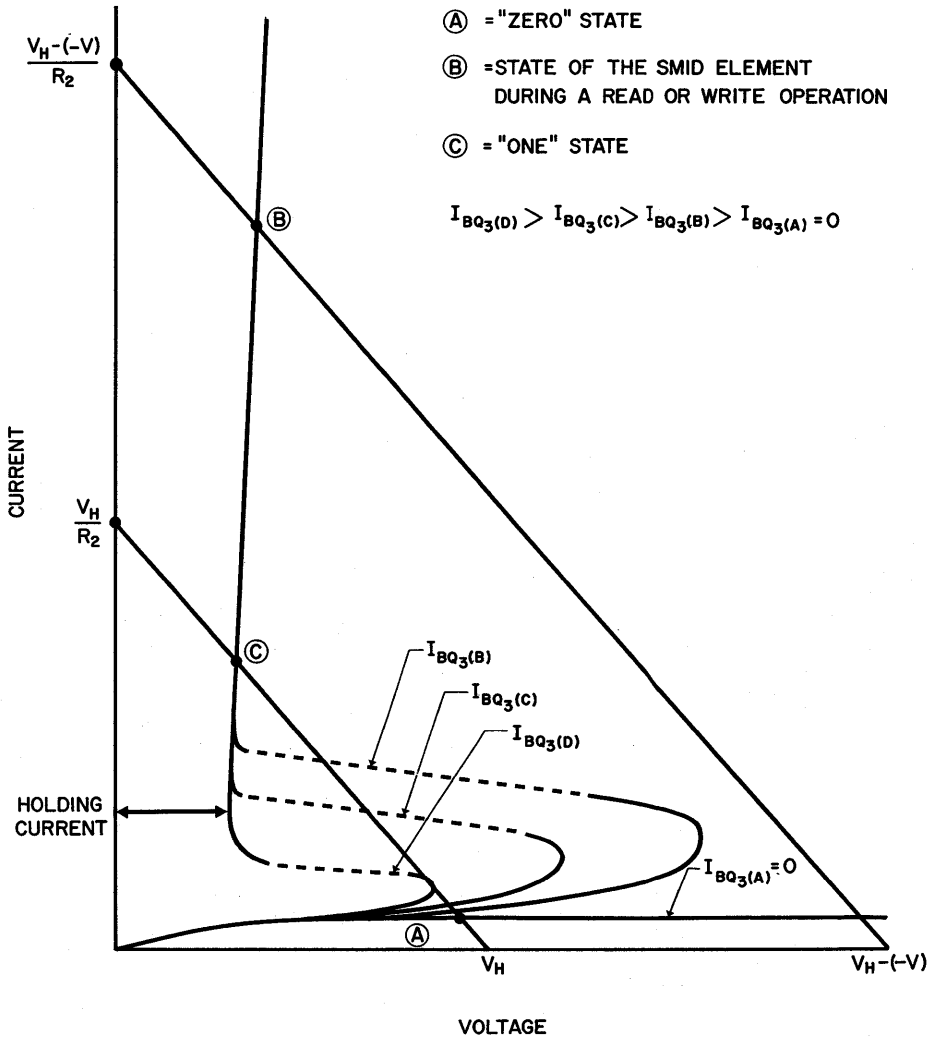



Figure 3. Characteristics of the transistor latching pair.

cate the manner in which the SMID is used in a random access memory configuration. It will be assumed, for example's sake, that the SMID element of Fig. 2a is initially in the high impedance or "zero" state. A "one" is written into the element by the simultaneous switching of point 4 from ground to –2.6 volts and point 3 from –2.6 volts to ground. This corresponds to time $t_1$ of Fig. 2b. The simultaneous switching of points 3 and 4 will cause base current to flow into transistor $Q_3$ thereby causing current to flow into the transistor latching pair. The minimum values of $-V$ as a function of temperature that will guarantee that transistor $Q_3$ will be switched into saturation during a write "one" operation are shown in Fig. 5.

Once the latching transistor pair has latched on, point 3 of Fig. 2b is returned to –2.6 volts and point 4 is returned to ground. Hold-on current will now flow from +2.4 volts through $R_2$ and the transistor latching pair into ground. The output of a word driver will always be at ground except when data are being written into, read out of, or cleared from the particular word of memory associated with that word driver. A more detailed discussion of the operation of the word driver is included in the section on Memory Electronics.

Reading out the data that are stored in an element is accomplished by switching point 4 to –2.6 volts. This corresponds to time $t_2$ of Fig. 2b. If the element is storing a "zero," the voltage level at point 2 of Fig. 2b will not change. If the element is storing a "one," and is therefore in the low-impedance state, the voltage level at point 2 will tend to follow point 4. This means that the voltage level at point 2 will drop from ground to approximately –2 volts. When the input to the sense switch is switched to a negative

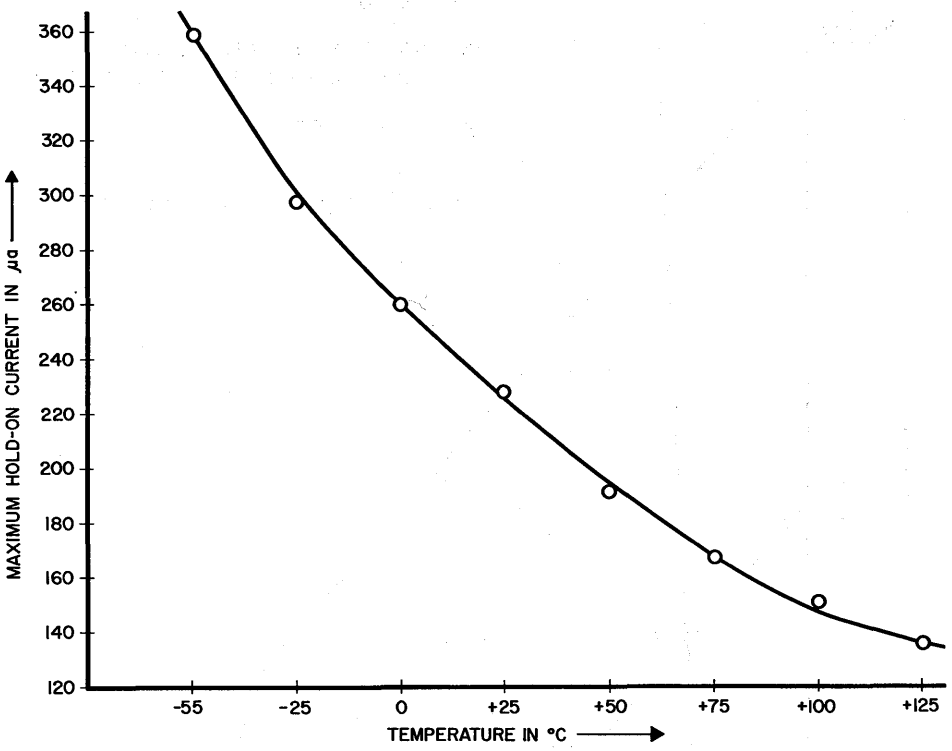



Figure 4. Maximum hold-on current versus temperature.

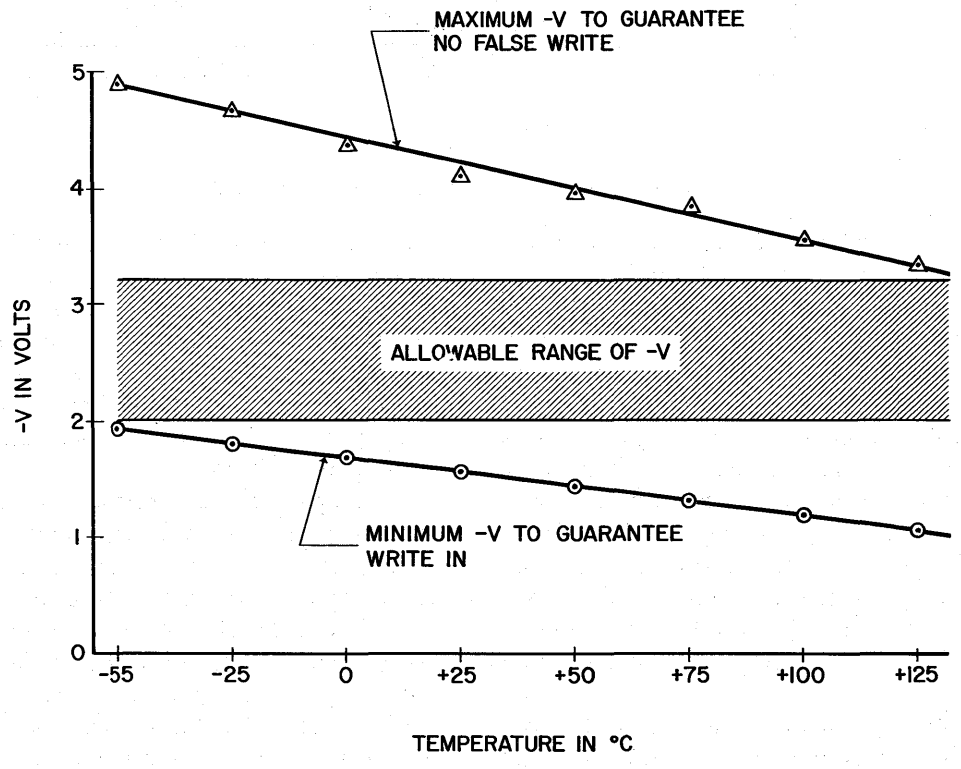



Figure 5. Maximum and minimum values of -$V$ as a function of temperature.

potential the output of the sense switch will go from ground to +5 volts. A more detailed discussion of the operation of the sense switch is included in the next section.

If the SMID element is storing a "one," hold-on current will continue to flow through the element after a read operation has been completed and the word driver output is switched back to ground. If the SMID element is storing a "zero," the read operation will not affect the state of the transistor latching pair in any way. Therefore, the read operation is nondestructive in nature.

The amount of time required to read out a "one," or access time, is effectively determined by the external electronics and not the SMID. This is because the transistor latching pair is already in conduction when storing a "one," and the time that is required to switch on diode $D_1$ of Fig. 2b is in the order of a few nanoseconds. The cycle time of a read operation may also be limited by external electronics and not the SMID. This is especially true if the number of bits on a given sense line is large. This can be seen if one considers the fact that the anode of diode $D_1$ is at a negative potential at the end of a read operation; the capacitance of this point must be charged to a potential that is equal to the threshold of the sense switch. Once the threshold of the sense switch has been reached, the output of the sense switch will go to ground, and a new read operation can begin. As the size of memory increases, or the cycle time requirements of the memory system decrease, the number of bits that can be sensed by a common sense switch must be decreased. This means that several sense switches must then be ORed together to form a total sense line output.

When data are to be cleared out of a given word of memory, the output of the word driver is caused to switch from ground to +5 volts. The clear operation corresponds to time $t_3$ of Fig. 2b. Since +5 volts is greater than $+V_H$, the flow of holding current is stopped and the base to emitter junctions of transistors $Q_1$ and $Q_2$ will be back biased. The SMID element will be switched to point $A$ of Fig. 3. Resistor $R_1$ of Fig. 2a has been sized so that the base of $Q_2$ will recover to $-V$ volts approximately 50 nanoseconds after a clear operation has ended. Figure 6 shows the time that is required to clear data out of a SMID as a function of temperature. As can be seen from Fig. 6, a faster clear operation is obtained if $+V_H$ is compensated as a function of temperature.

The last waveform of Figure 2b indicates a write

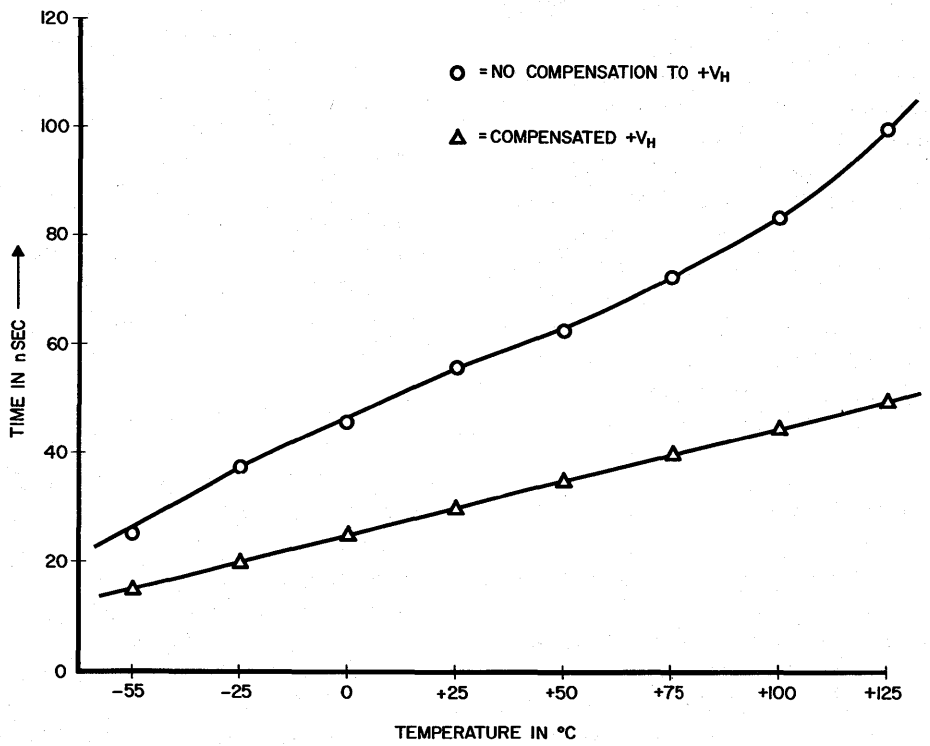



Figure 6. Clear time versus temperature.

"zero" operation. The output of the word driver is switched from ground to –2.6 volts, but since the datum to be written into the SMID element is a "zero," the output of the digital driver remains at –2.6 volts. The base to emitter junction of transistor $Q_3$ of Fig. 2a will be back biased by a collector-to-emitter saturation voltage; therefore, base current cannot flow into transistor $Q_3$. The transistor latching pair will remain in the high impedance or "zero" state.

Writing a "zero" into a SMID is a straightforward operation; however, the latching transistor pair is susceptible to a rate effect type of turn on.[4] The rate effect type of turn on in a SMID element can best be understood by considering the capacitive voltage divider that is formed by the parallel combination of the base-to-collector capacities of $Q_1$ and $Q_2$. When the voltage swing at point 4 of Fig. 2a is large enough, the base to emitter junction $Q_1$ will become forward biased (due to capacitive coupling of the voltage pulse) and the transistor latching pair will begin to conduct. Figure 5 shows the maximum value that $-V$ volts can have as a function of temperature in order to guarantee that a write pulse alone (i.e., no digital pulse in coincidence with write) will not cause the latching transistor pair to conduct. From Fig. 5 it can be seen that $-V$ volts (–2.6) can be varied in excess of $\pm 13$ percent from –55°C to +125°C.

Although Fig. 2a represents the basic SMID topology, it should be noted that it is possible to vary this topology slightly and yet obtain the same basic type of operation. Transistor $Q_3$ can be eliminated and replaced by a diode. The cathode of this diode would be connected to the base of transistor $Q_2$, and resistor $R_3$ would be in series with point 3 and the anode of this new diode. The diode would be used to isolate the transistor latching pairs that are common to a digit line. Nine-bit SMID arrays have been fabricated by the utilization of both topologies. Each type of SMID array operates satisfactorily.

## MEMORY ELECTRONICS

The low-power levels of a SMID memory are quite compatible with the power handling capability of monolithic integrated circuits. Two special integrated circuit drivers have been designed and fabricated, and a commercially available $T^2L$ gate has been remetalized. These three special integrated circuits in conjunction with standard $T^2L$ gates have been used to construct a completely integrated circuit memory system.

The drive electronics that are used to write, read, hold, or clear data in a given word of memory are shown in Fig. 7. Pins 1 and 2 are used as the second level of memory address decoding. The first level of address decoding is performed in an $X$ $Y$ type of

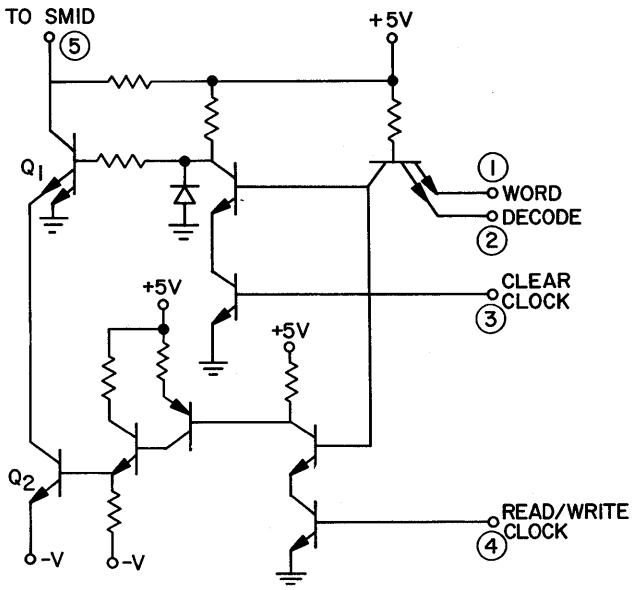



Figure 7. Word driver.

matrix consisting of $T^2L$ logic gates. Pins 3 and 4 are clock inputs. The input terms to a SMID word driver are normally false (i.e., at ground). When pin 1 or 2 or 3 is at ground level, the output transistor, transistor $Q_1$ of Fig. 7, will receive base drive and therefore be driven into saturation. This will allow hold-on current to flow from the SMID elements of a given word and into ground. If pins 1, 2, and 3 are all true (i.e., +2.5 volts or greater) at the same time, the base drive to transistor $Q_1$ will be zero. This will cause the voltage level at pin 5 to be +5 volts. This is the state of a word driver during a clear operation. If pins 1, 2, and 3 are true at the same time, transistor $Q_2$ as well as $Q_1$ will conduct into saturation. The voltage level at pin 5 will be approximately –2.6 volts. This is the state of a word driver during a read or write operation.

Four word driver circuits are included on a single chip and are packaged in a standard 14-lead flat package. With one flat package of word electronics it is then possible to perform all necessary word functions on four different words of memory. The

word-drive electronics has been designated so that word lengths up to 40 bits can be accommodated.

The driver that is used for switching a SMID memory system's information line from –2.6 volts to approximately ground during the writing of a "one" into a given bit is shown in Fig. 8. The input terms to the digit driver are normally false. Under these conditions transistor $Q_2$, of Fig. 8, will receive base drive and the output of the driver will be at approximately –2.6 volts. When both the clock and data

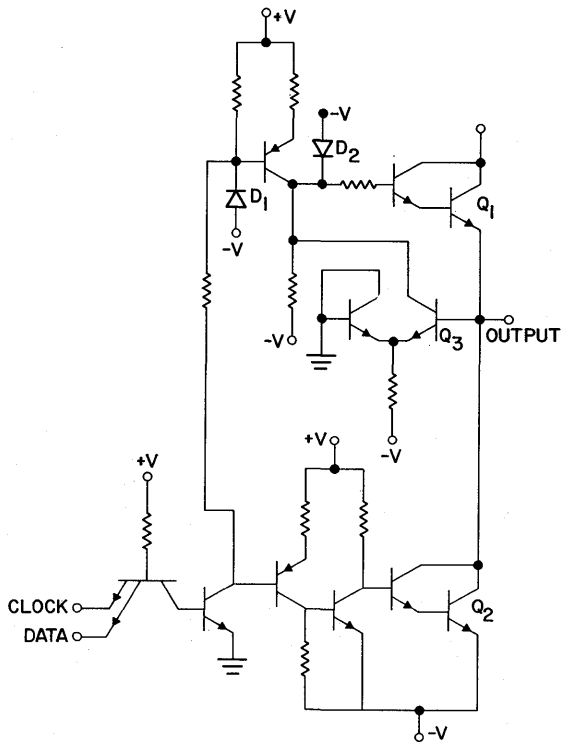



Figure 8. Digit driver.

input terms are true the base drive to transistor $Q_2$ is "zero" and transistor $Q_1$ will receive base drive. The junction capacity of diodes $D_1$ and $D_2$ delay the turn on of $Q_1$ thereby guaranteeing that $Q_2$ will be turned off before $Q_1$ is turned on. The voltage level on the output terminal will rise from –2.6 volts toward +5 volts. Once the voltage level of the output terminal reaches ground, $Q_3$ will conduct and thereby take base current away from $Q_1$.

There are four circuits of the type shown in Fig. 8 on each digit driver chip. Each chip is packaged in a 14-lead flat package. Each digit driver flat package can therefore drive four SMID information lines. The push-pull output of the digit driver makes it possible to drive long digit lines at high speeds with low standby power requirements.

The sense switch is shown in Fig. 9. A standard $T^2L$ logic gate was remetalized so that the isolation region could be tied to $-V$ volts rather than to ground. This guarantees that the isolation regions of the sense chip will not become forward biased during a read or write operation.

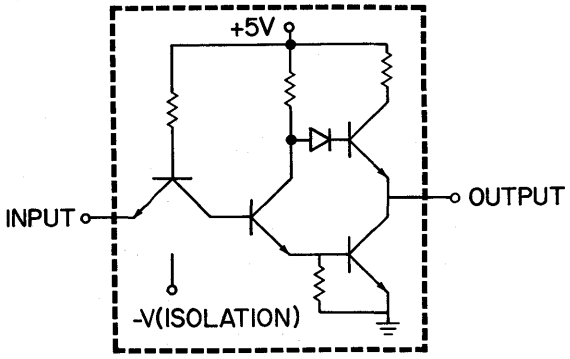



Figure 9. Sense switch.

The output of a sense switch is normally at ground. If the input to the sense switch should go to ground or lower (as is the case during a write "one" or read "one" operation), the voltage level at the output of the sense switch will rise towards +5 volts. There are 4 sense switches in a 14-lead flat package.

Figure 10 shows the manner in which the SMID elements and the drive electronics are interconnected, and a block diagram of the complete 64-word-by-21-bit system is included.

## RESULTS AND PROJECTIONS

A SMID memory system of 64 words, 21 bits per word, has been constructed and is now undergoing tests. All of the selection and drive electronics are monolithic integrated circuits.

The total memory system, including all peripheral electronics, requires 194 14-lead flat packages. The capability of operating at 100-nanosecond read-only cycle times with access times of less than 80 nanoseconds has been demonstrated. The total system power has been measured to be less than 3.0 watts. Pertinent waveforms of a sequential clear/write operation are shown in Fig. 11a. Waveforms of the read-only operating mode are shown in Fig. 11b.

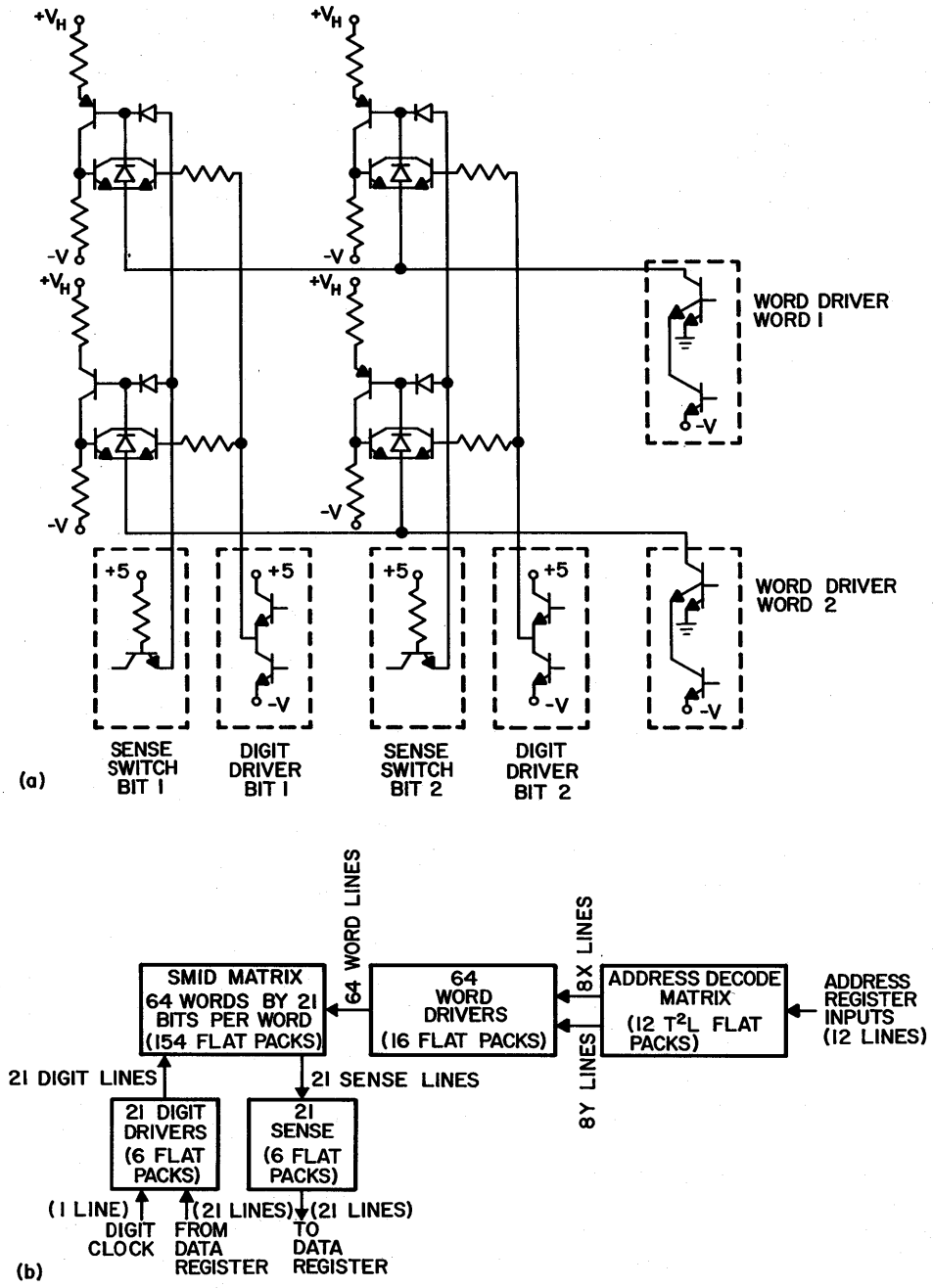
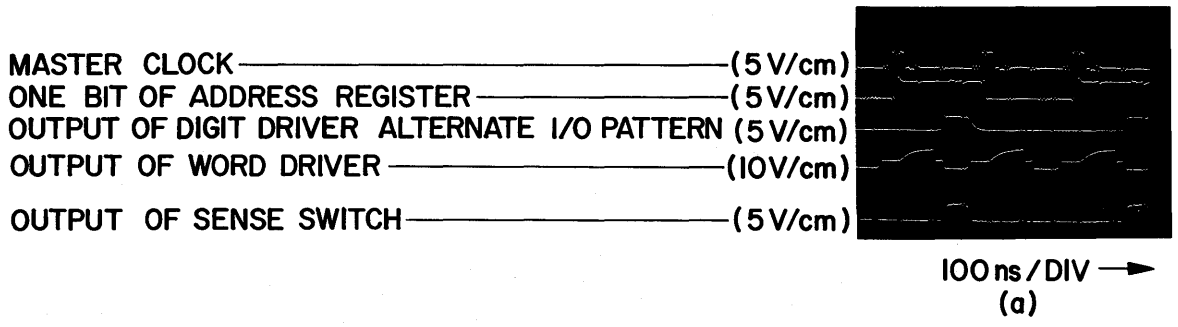



Figure 10. Interconnections between SMID array and driver electronics.

The cost of a small scratchpad memory system in which the SMID and its associated drive electronics are utilized has been projected to be under $2.00 per bit in 1966. It has also been projected that 100 64-word-by-21-bit SMID memories, including all the necessary electronics, assembly, and checkout charges, will cost less than $1.00 per bit in late 1967. A 16-bit SMID chip packaged in a standard 14-lead flat pack is now being investigated. A chip such as this should reduce the price per bit and certainly will improve the memory system packing density.

Several types of flip-chip interconnection schemes are now being investigated. The proper chip interconnect technique, in conjunction with a package that has many more leads than 14, should



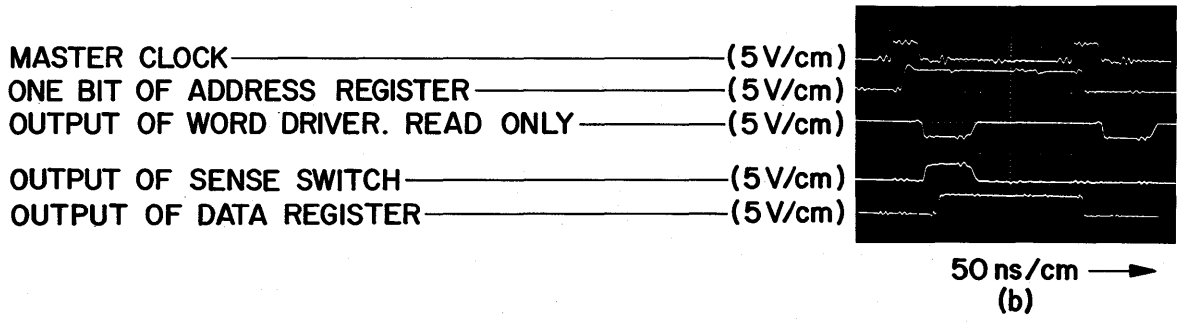




Figure 11. SMID memory operating waveforms.

make it possible to package the 64-word-by-21-bit memory on a circuit board area of less than 3 square inches.

## CONCLUSION

A transistor latching switch that has been optimized for random access memory usage has been fabricated as 9 bits of memory on a single integrated circuit chip.

When compared to other semiconductor memory devices, the element described in this document represents at least an order of magnitude reduction in power without significantly sacrificing switching speed. The low-power requirements of the storage element have made it possible to build a completely integrated circuit memory system.

## ACKNOWLEDGMENTS

Thanks are due to A. Ronald Roth and James Payton for their advice and encouragement. The author would also like to acknowledge his indebtedness to Yosh Murakami for his help in obtaining test data, and for the work he has done on the analysis of the SMID element and driver circuits.

## REFERENCES

1. R. H. Cole and P. Smitha, "An Integrated Circuit Memory," presented at a Los Angeles IEEE Section Symposium on Computer Technology, December 1964.

2. *Silicon Controlled Rectifier Manual,* General Electric Co., Rectifier Components Dept., Auburn, N. Y. 1964.

3. T. A. Longo et al, "Planar Epitaxial PNPN Switch with Gate Turn Off Gain," presented at the 1962 WESCON.

4. R. A. Stasior, "How to Suppress Rate Effect in PNPN Devices," *Electronics,* Jan. 10, 1964.

# AN EXPERIMENTAL 65-NANOSECOND THIN FILM SCRATCHPAD MEMORY SYSTEM

G. J. Ammon and Carl Neitzert
*Radio Corporation of America*
*Computer Advanced Product Research*
*Camden, New Jersey*

## INTRODUCTION

As computers become larger and more complex, the need for a high-speed scratchpad type memory becomes greater. Furthermore, the size and speed requirements increase. An early memory, suitable for use as a scratchpad, was described by H. Amemiya, R. L. Pryor, and T. R. Mayhew.[1] This memory used two ferrite cores per bit and had a read/regenerate cycle time of 200 nanoseconds. More recently, a 64-word by 20-bit thin magnetic film memory was described by G. J. Ammon and C. Neitzert.[2] This memory had a read/regenerate cycle time of 125 nanoseconds and its speed was limited primarily by the electronic circuitry. A number of other memories and memory designs, suitable for use as scratchpads, have been reported in the literature.[3,4,5] Future need for a 256 to 1,024 word memory having a cycle time of 50 nanoseconds has been indicated. A project was therefore initiated to study the feasibility of such a system.

At their present state of development, ferrite cores do not appear capable of such speeds. Thin magnetic film memories, on the other hand, are believed to be capable of this speed using transistor circuitry, provided the best circuit and packaging techniques are used. The goal for the project was therefore set as a 256-word by 25-bit thin film memory having a read/write cycle time of 50 nanoseconds.

## MEMORY ARRAY

The memory array consists of two polished aluminum plates coated on one side with a 1000-angstrom continuous film of 80-17-3 NiFe Co. These are mounted back to back. A set of 128 word lines, etched from ½-oz copper backed with Mylar is placed over each film surface. The word lines are 10 mils wide and are on 20-mil centers and are grounded to a gold-plated portion of the film plate at the far end by means of a pressure contact. They have a characteristic impedance of 24 ohms and a delay of approximately 1.5 nanoseconds per line.

A set of digit and sense lines, also etched from ½-oz copper backed with Mylar, is placed over and orthogonal to the word lines. They extend completely around the two film plates and are spaced from the word lines by a sheet of ½-mil Mylar. The digit and sense lines use an interdigitated arrangement in which the digit line consists of two parallel conductors 6.5 mils wide and 11.5 mils apart. The digit lines are driven at their center, from a 15-ohm line, and are terminated at their free ends with 27-ohm resistors. The resistors are trimmed to give a specified differential digit noise of not more than 2.5 millivolts, at the sense amplifier input, when the digit rise and fall times are 5 nanoseconds. The sense lines have a characteristic impedance of 54 ohms and are connected to their respective sense

amplifiers by a pair of 50-ohm terminated coaxial cables. The ratio of signal to noise at read time is specified to be greater than 5 to 1.

## SYSTEM ORGANIZATION AND PACKAGING

A block diagram of the system organization is shown in Fig. 1. The address is decoded in two steps. The second step consists of a word driver matrix with a driver for each word line and a 2-input AND gate at the input to each driver. The first step of decoding is provided by AND gates at the inputs of the matrix drivers.

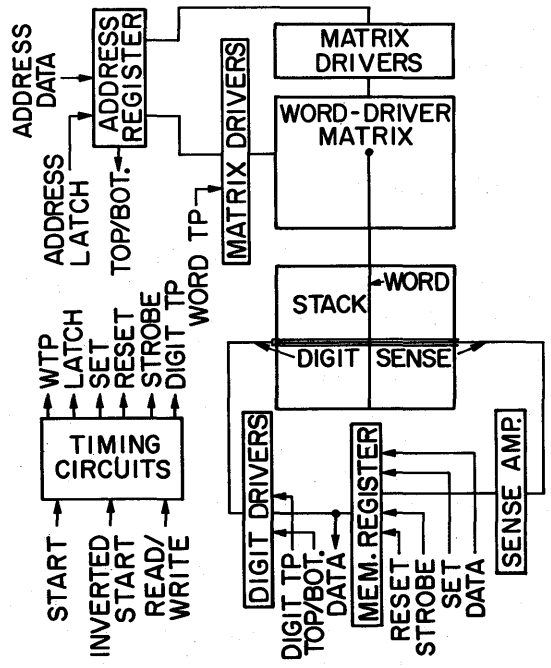



Figure 1. System block diagram.

A regeneration loop consists of a 4-stage sense amplifer, a gated flip-flop memory register and a bipolar digit driver. The polarity of the digit current is controlled by the information to be written and the location of the word being written. A timing circuit generates all timing pulses required for the system operation when initiated by a start pulse. Details of all circuits are given later.

All electronic circuits are packaged on small plug-in modules. Figure 2 shows a two-view photograph of a half digit driver.

It was decided that the feasibility of the system could be demonstrated if a minimum of 8 words and 6 bits could be operated at one time. Therefore, 6 regeneration loops and the associated portion of the timing circuit were constructed on one mother board. Top and bottom views of this board are shown in Figs. 3 and 4.

The complete system was mounted in an aluminum angle frame shown in Figs. 5 and 6. The bit board appears in the upper portion and the memory array in the lower portion of Fig. 5. Any 6 of the 25 digit and sense lines can be connected to the board by means of coaxial cables and connectors. The electronic circuits required for 8 words are mounted on a movable word board shown in the upper left hand portion of Fig. 6. All 256 word lines are brought out to connectors, visible in this figure, by means of miniature 15-ohm solid-wall coaxial cable, and any 8 lines can be connected to the word drivers by means of short pieces of 24-ohm flexible cable.

## ELECTRONIC CIRCUITS

Except for the sense amplifiers and drivers, the electronics is based on a logic circuit shown in Fig. 7a. This circuit is very fast and can be adapted to perform a large number of functions in a very simple manner. This circuit performs the logical operations shown in Fig. 7b, but it can be varied to provide other numbers of AND-gate and OR-gate inputs. It can also provide the functions shown in Fig. 7c. By returning the direct output to one of the inputs, a set/reset flip-flop is obtained and its use as a one-shot multivibrator is explained later.

When driving a 70-ohm terminated line the output rise time is 3 nanoseconds and the circuit delay, at the 50 percent points, is 3 nanoseconds. Its speed can be increased slightly by the use of more recent transistors. The input impedance is roughly approximated by a resistance of 7,500 ohms in parallel with a capacitance of 5 picofarads. The output impedance appears to be 5.5 ohms but this value can be effectively decreased for large fanouts by splitting the load and using 2 or more output emitter followers. By employing 3 emitter follower outputs, a fanout of 24 has been obtained. In this case the rise and fall times were 8 nanoseconds and the circuit delay was 5 nanoseconds.

For the circuit shown in Fig. 7, the normal logic levels are +0.2 and +1.0 volts at both the input

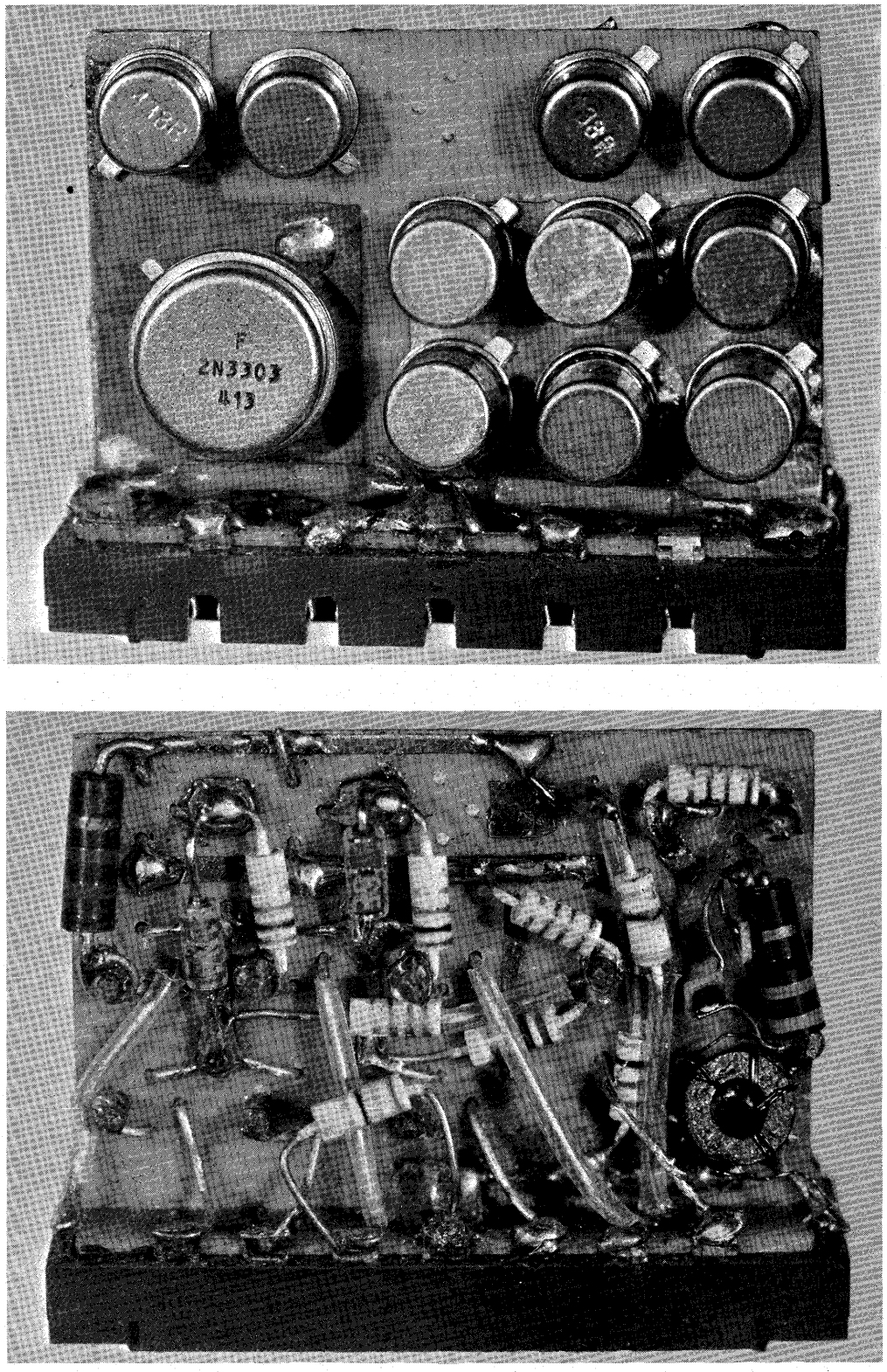



Figure 2. Front and back views of a plug-in module.

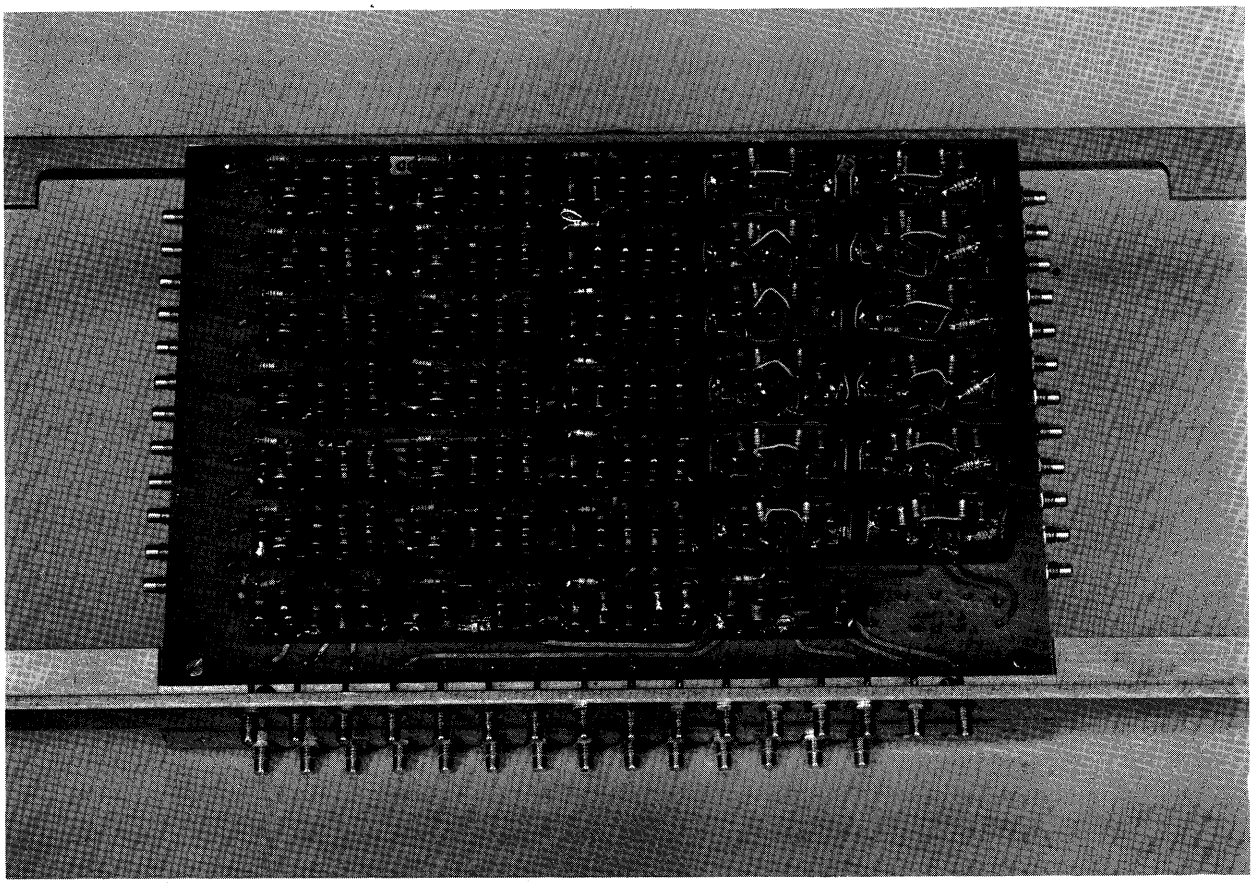


Figure 3. Top view of the bit board.

and output. It is shown later how these can be changed at the output when special loads must be driven.

*Timing Circuit*

A logic diagram of the timing circuits is shown in Fig. 8. These circuits are made from delay lines and variations of the basic logic circuit shown in Fig. 7. The timing unit receives a high level on the read/write input during a read cycle and a low level during a write cycle. The start and inverted start pulses are received simultaneously. The inverted start pulse is used to reset the memory register. The start pulse sets the word TP flip-flop at the top of the figure. This starts the word pulse and latches the address register. The start pulse is then inverted and delayed to end the word pulse and unlatch the address register. A small delay is inserted in one of the latch lines to prevent a race condition in the address register.

The start pulse is also delayed and split into its direct and inverted forms. These are combined with the read/write level to give either a set pulse or a strobe pulse. During a write cycle the inverted start pulse is ANDed with the low read/write level to give a set pulse while during a read cycle the direct start pulse is ANDed with the high read/write level to trigger a one-shot multivibrator which in turn generates the strobe pulse. The one shot consists of a flip-flop in which the inverted output is delayed and returned to an input to reset the flip-flop and terminate the pulse.

The direct start pulse is further delayed and used to trigger another one-shot multivibrator which generates the digit timing pulse.

*Address Register*

The logic diagram of one bit of the address register is shown in Fig. 9. Circuit details and parameter values are shown in Fig. 7. When input *a* is high and input *b* is low the register is latched, and changes in the level of input *c* do not change the state of the flip-flop. This is the case for the duration of the word current timing pulse. When the

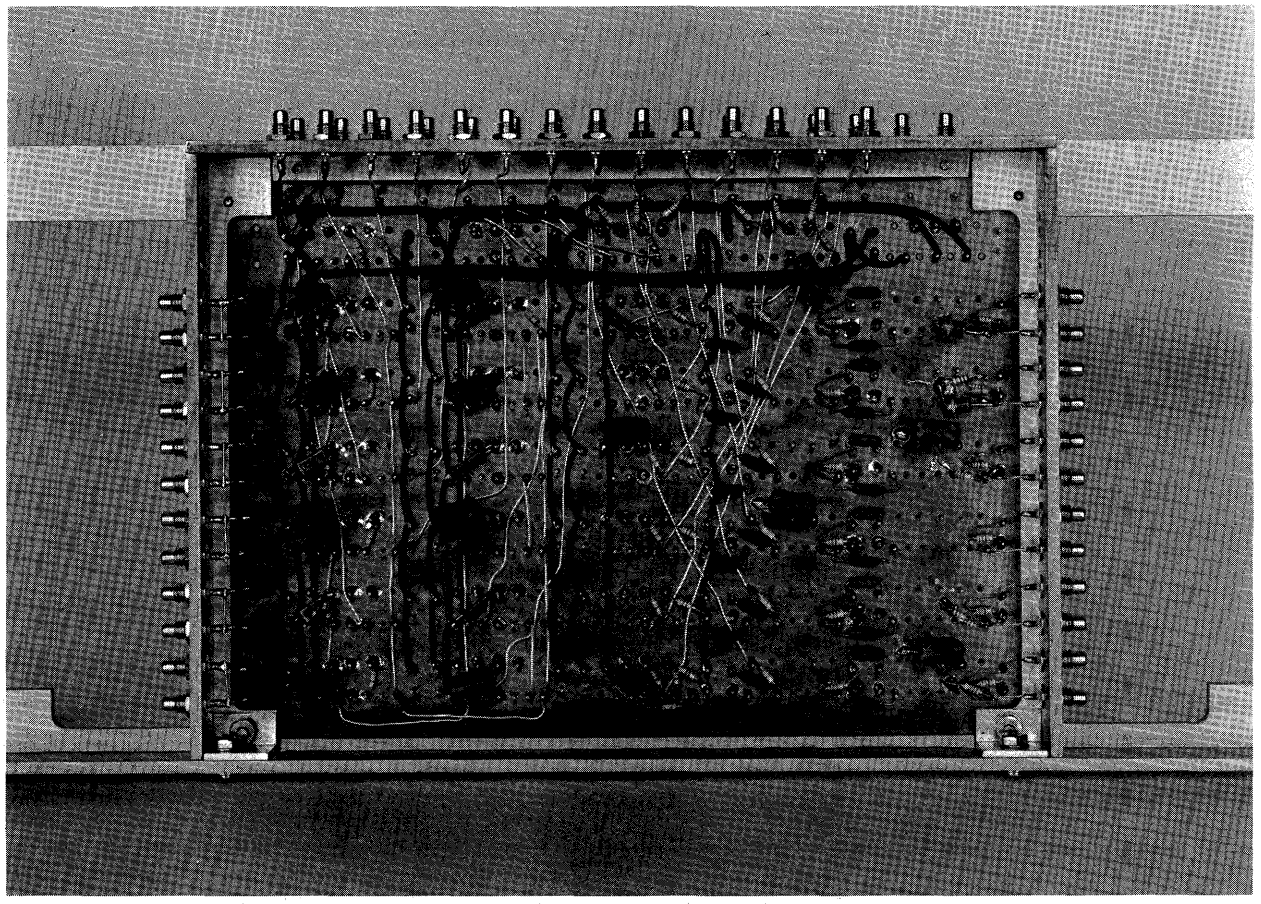


Figure 4. Bottom view of the bit board.

word pulse ends the address is no longer needed and input *a* goes low and input *b* goes high. The state of the flip-flop is then under the direct control of input *c*. If *c* is high the flip-flop is set, and if *c* is low it is reset. Thus, an address is held in the register only as long as it is needed after which the register is free to accept a new address if it becomes available before the end of the cycle.

*Matrix Drivers*

The elements of the word matrix, which are individual word-line drivers, require input levels of +0.2 and +2.5 volts in order to give fast turn-on and turn-off. In order to supply these levels, a slight modification of the basic logic circuit, as shown in the schematic circuit diagram of Fig. 10, is required.

*Word-Line Drivers*

In order to attain a minimum sense voltage of 1.5 millivolts, it was predicted that a word current 0.5 ampere with a rise time of 6 nanoseconds would be required. The total delay of a word line and its connecting cable is 3.2 nanoseconds so that with a rise time of 6 nanoseconds the condition for the line to act like an inductance ($t_r > 2t_d$) is poorly satisfied. However, this condition was assumed in the driver design and the resulting schematic circuit diagram is shown in Fig. 11. Two drivers are packaged in one module.

The inductance of the word line and its connecting cable is approximately 65 nanohenrys and the supply voltage is limited to about 15 volts by the transistor breakdown voltage. If the transistor is an ideal switch and capacitor *C* is selected to give critical damping on the word current turn-on, the time required to rise 0.45 ampere is 3.6 nanoseconds. However, the turn-on time of the transistor increases this to more than 6 nanoseconds. As a result the capacitance must be increased beyond the critical value and the circuit becomes oscillatory. The computed value of *C* for critical damping is 18

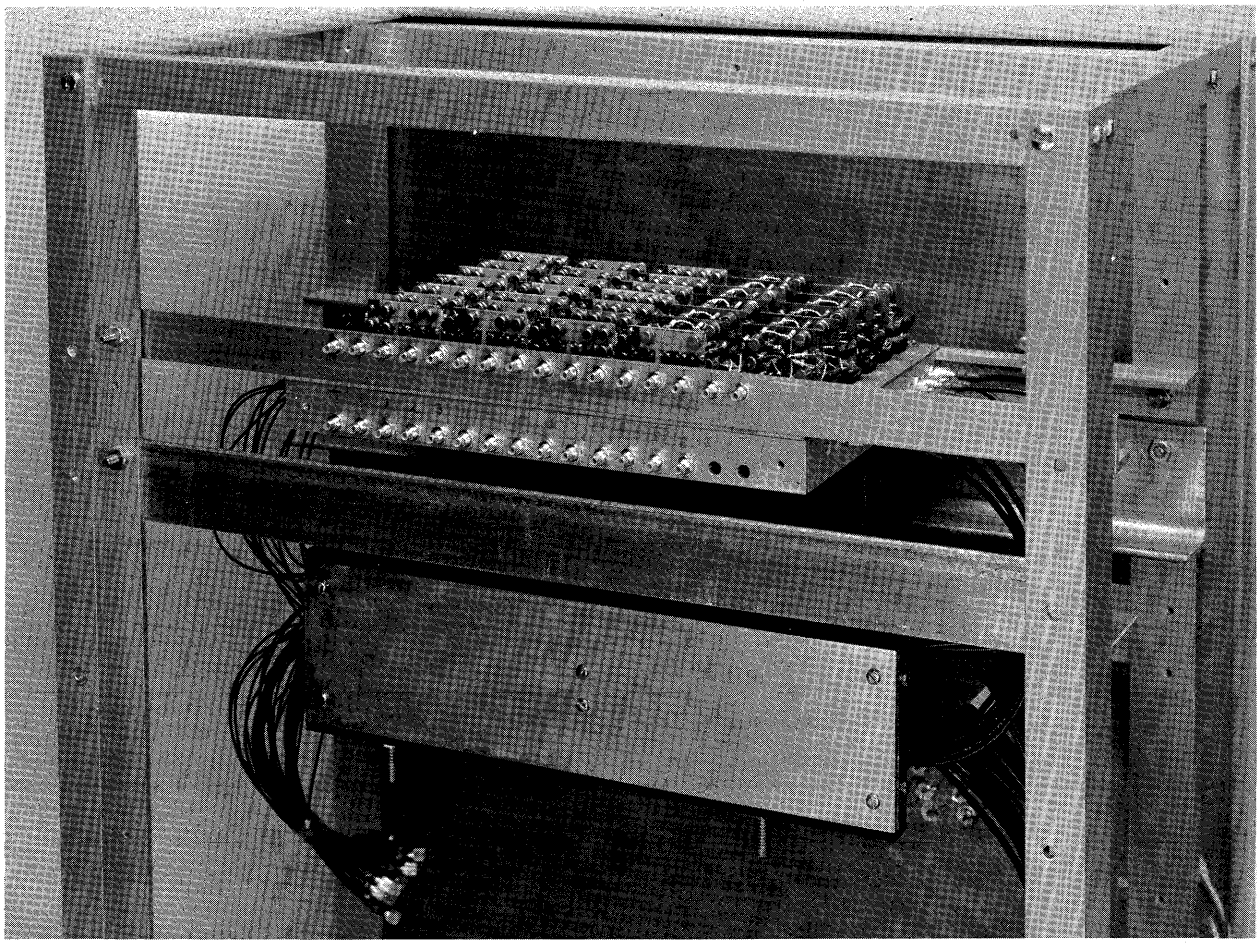


Figure 5. Bit side of complete memory system less timing and connecting cables.

picofarads. The value used was determined experimentally.

The fall time of the current is determined by the line inductance and resistor $R$. If the diode has a zero threshold voltage and zero resistance, the computed time for the current to fall to 0.05 ampere is 6.7 nanoseconds. This fall time has an effect upon domain wall creep in the magnetic film and will be discussed later.

The first two transistors of Fig. 11 serve as an AND gate and the third drives the transformer. The 150-picofarad speed-up capacitor, in the emitter circuit of the third transistor, gives a large initial transformer voltage to turn the output transistor on fast. The field discharge of the transformer gives a large reverse base voltage to turn it off. The transformer also provides a d-c level shift making it convenient to ground the far end of the word line. The $RC$ circuit between the power supply and the collector of the output transistor is made common to all 8 drivers. This reduces the space required for resistors and is permissible since only one driver is used at a time.

*Sense Amplifiers*

Figure 12 shows the schematic circuit diagram of a sense amplifier. Capacitance coupling is required between stages to eliminate drift in the d-c levels due to temperature changes. Each amplifier is packaged in two modules with the coupling capacitors connected between modules. One side of the output drives the sense input to the information register and the other side provides an inverted output for test purposes.

This amplifier gives a single-ended output of 1 volt for a differential input of 2.4 millivolt. With a very fast rising input, the delay through the amplifier is 5.6 nanoseconds, and the output rises in 4 nanoseconds. When a common mode pulse—having

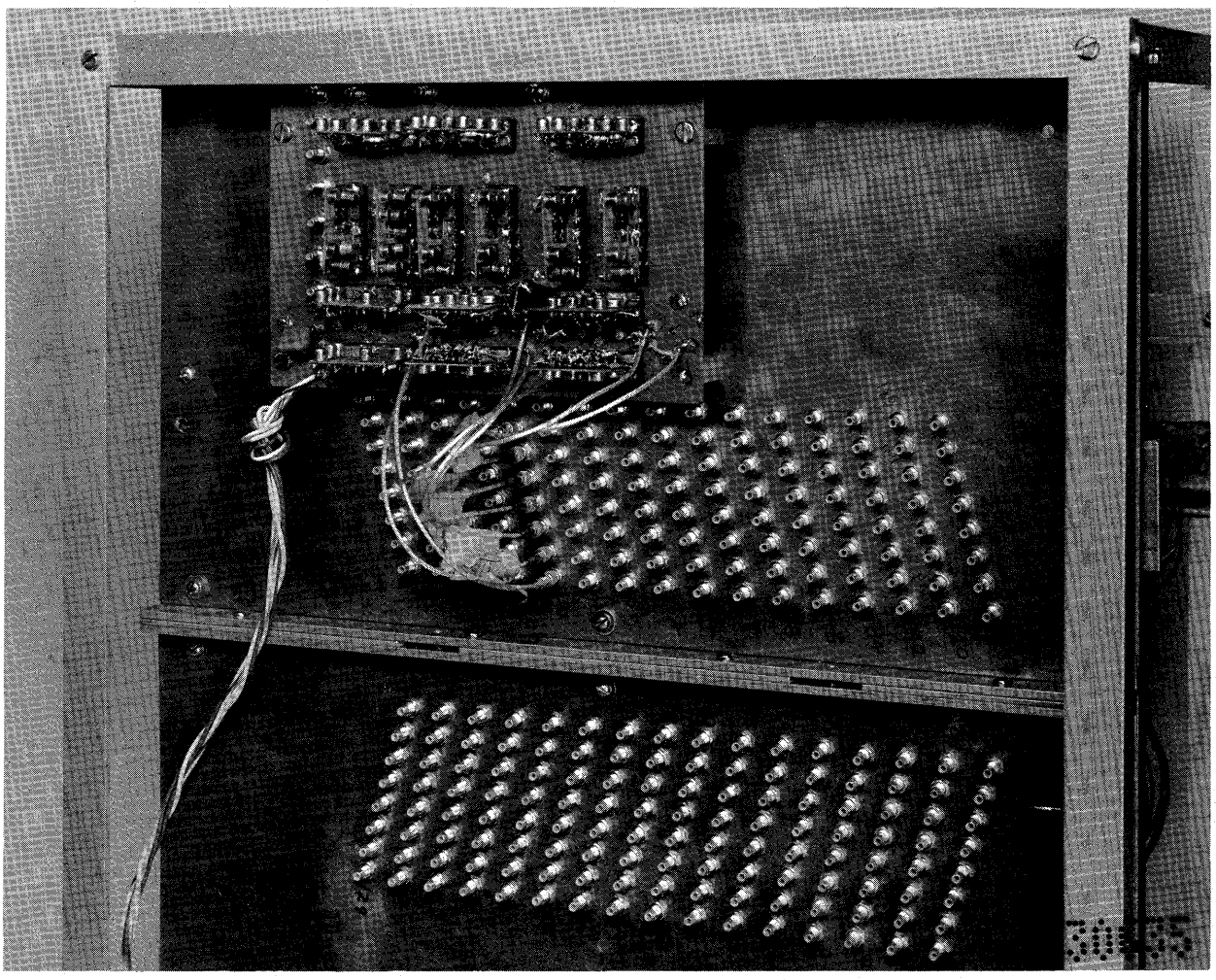


Figure 6. Word side of complete memory system.

an amplitude of 250 millivolts, rise and fall times of 4 nanoseconds, and a base width of 10 nanoseconds—is applied to the input the common mode output is not measurable. However, the unbalance in the amplifier produces a single-ended output of up to 1.9 volts, and the time required for amplifier recovery is about 25 nanoseconds. Waveforms under operating conditions are given later.

*Memory Register*

Each bit of the information register consists of a flip-flop made from one logic module as shown in Fig. 13. The schematic circuit diagram is the same as that shown in Fig. 7a except that a third AND gate is used and a feedback connection is provided to form a flip-flop. Strobing of the sense signal is unusual in that it is performed at the register input rather than in the sense amplifier.

*Digit Drivers*

A hybrid diagram of a complete digit driver is shown in Fig. 14. It consists of a positive and a negative driver in separate modules with their outputs paralleled at the digit-cable inputs. The logic on the input is necessary because a ONE is written with a positive digit current on the top plate and a negative current on the bottom plate. A ZERO, of course, requires the opposite polarities. Transformer coupling to the output transistors permits the use of the same type high-speed transistor for each current polarity. In order to properly drive the transformers, the logic circuits supply a high voltage output similar to that shown in Fig. 10.

This driver delivers a 200-milliampere current pulse to the digit line cable with a rise time of 6 and a fall time of 4 nanoseconds. The delay be-

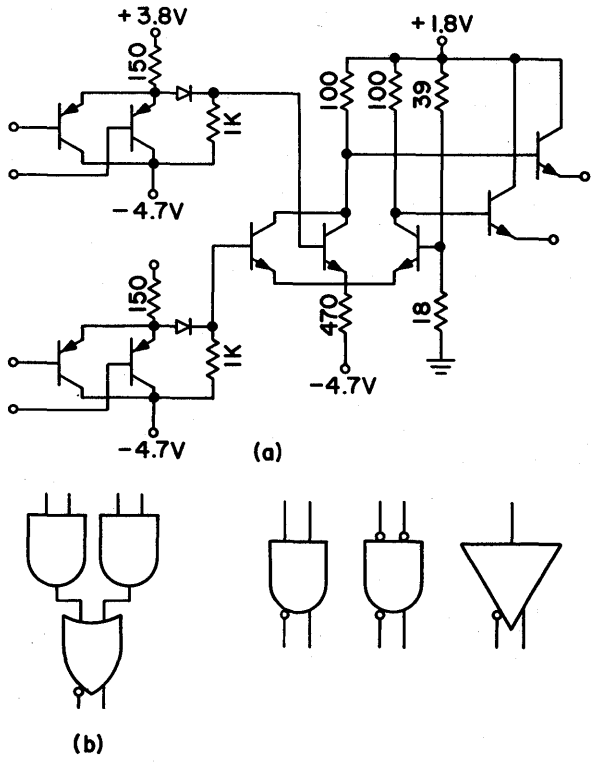



Figure 7. Basic logic circuit: (*a*) Schematic circuit diagram; (*b*) logic function performed by Fig. 7a; (*c*) other logic functions obtainable.

tween the leading edges of the input timing pulse and the output current pulse is 8 nanoseconds when measured between the 50 percent points.

## EXPERIMENTAL RESULTS

### *Temperature Effects*

The input and output of a sense amplifier were monitored while the ambient temperature was varied from –45°C to +60°C. The variation in delay through the amplifier was less than 0.5 nanosecond and the variation in gain was less than 5 percent.

In another test two inverters, a flip-flop connected as a one-shot, and a matrix driver were connected in a cascade to drive all four inputs of two word drivers simultaneously. The word driver outputs were connected to short-circuited cables to simulate word lines and their input voltage and output currents were monitored while the ambient temperature was varied from –45°C to +80°C. In the range from 0°C to 55°C the input pulse to the word driver increased in width by 2.5 nanoseconds and the output pulse width increased by 5.5 nanoseconds. No

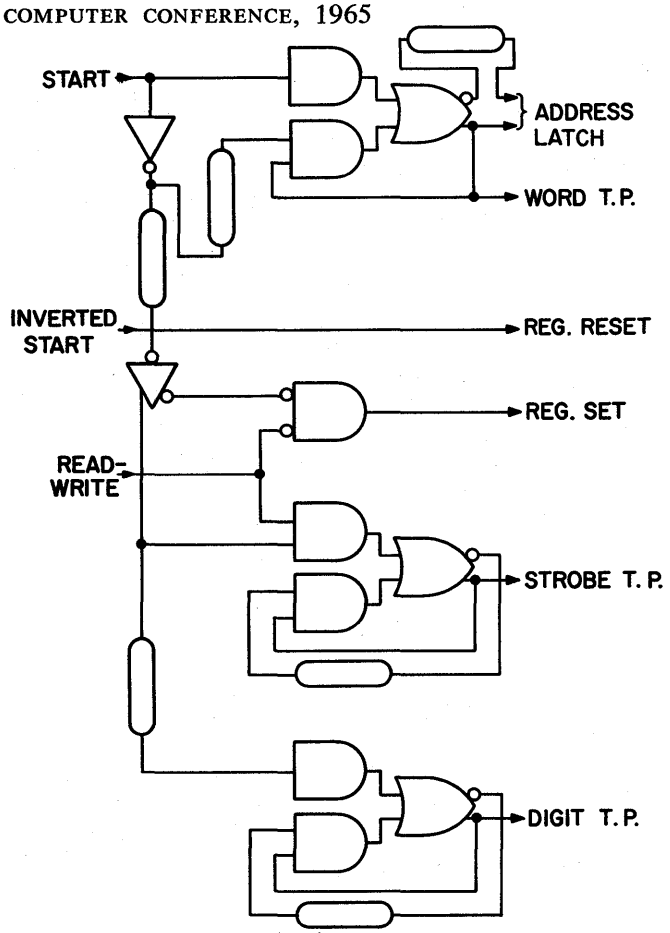



Figure 8. Logic diagram of the timing circuits.

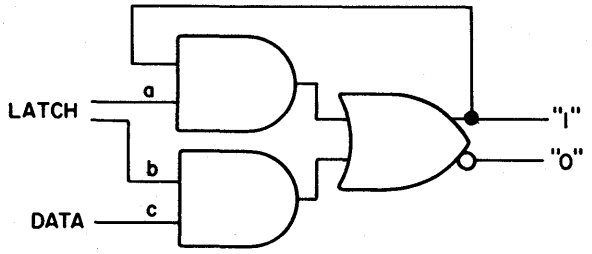



Figure 9. One bit of the address register.

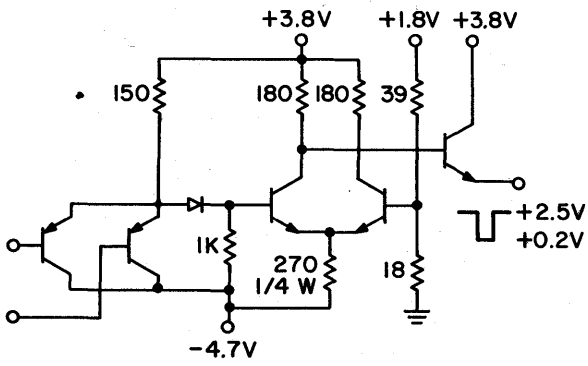



Figure 10. Schematic circuit diagram of a matrix driver.

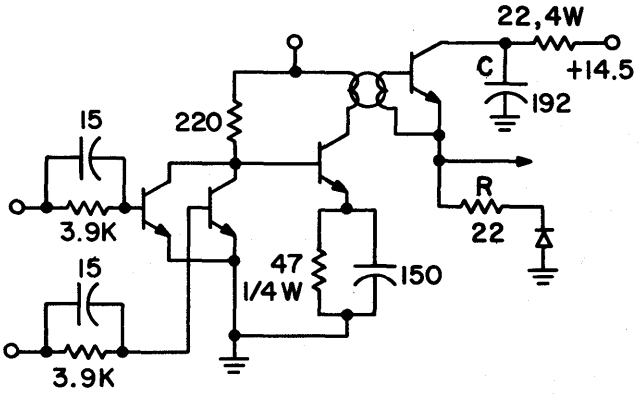



Figure 11. Schematic circuit diagram of a word driver.

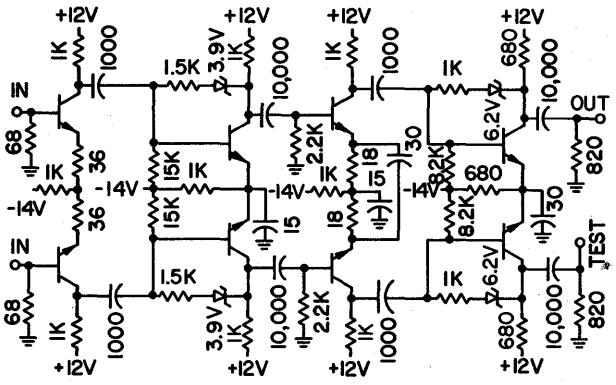



Figure 12. Schematic circuit diagram of a sense amplifier.

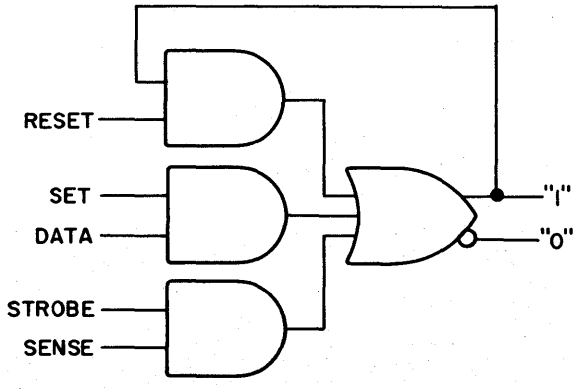



Figure 13. Logic diagram of an information-register bit.

change in delay of the leading edge through the word driver could be detected for the full temperature range.

*Cycle Time*

Figure 15 shows selected waveforms taken while reading and regenerating a ONE at a repetition rate of 10 megacycles. In curve (*d*), the sense voltage is too small to observe and the noise shown is substantially all common mode. In curve (*e*), the

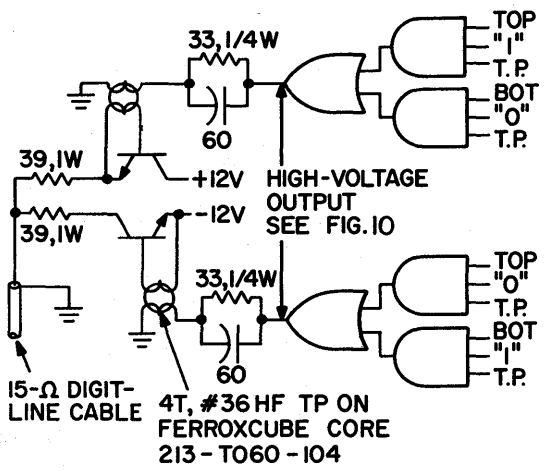



Figure 14. Digit driver diagram.

sense voltage is the first negative pulse and the remainder is digit noise plus the rewrite sense voltage. Curves (*a*) and (*c*) show 60 nanoseconds from the beginning of the start pulse to the end of the digit current. Curve (*e*) shows that the amplifier recovers from the digit noise 65 nanoseconds after the beginning of the sense voltage. Finally, curves (*a*) and (*f*) show an access time of 30 nanoseconds. Fig. 16 shows the amplifier output with a higher repetition rate. In this case the circuit is arranged to write a ONE after reading a ZERO and to write a ZERO after reading a ONE. Two sense ONE signals appear as negative pulses at 2 and 8 divisions from the left and a sense ZERO appears as a positive pulse midway between the ONEs.

*Increased Number of Words*

Although the memory plane has a total of 256 words, electronics was built for only 8. However, physical path lengths and loading, for a full set of electronics, were closely approximated except for the loads on the matrix drivers. In order to show the effect of a 16-by-16 word-driver matrix, a matrix bus was loaded with an additional capacitance of 200 picofarads in parallel with 300 ohms. There was no appreciable change in either the leading edge of the word current or the amplitude or timing of the amplified sense voltage. However, there was an appreciable increase in the fall time of the word current. As will be shown later, this is not objectionable.

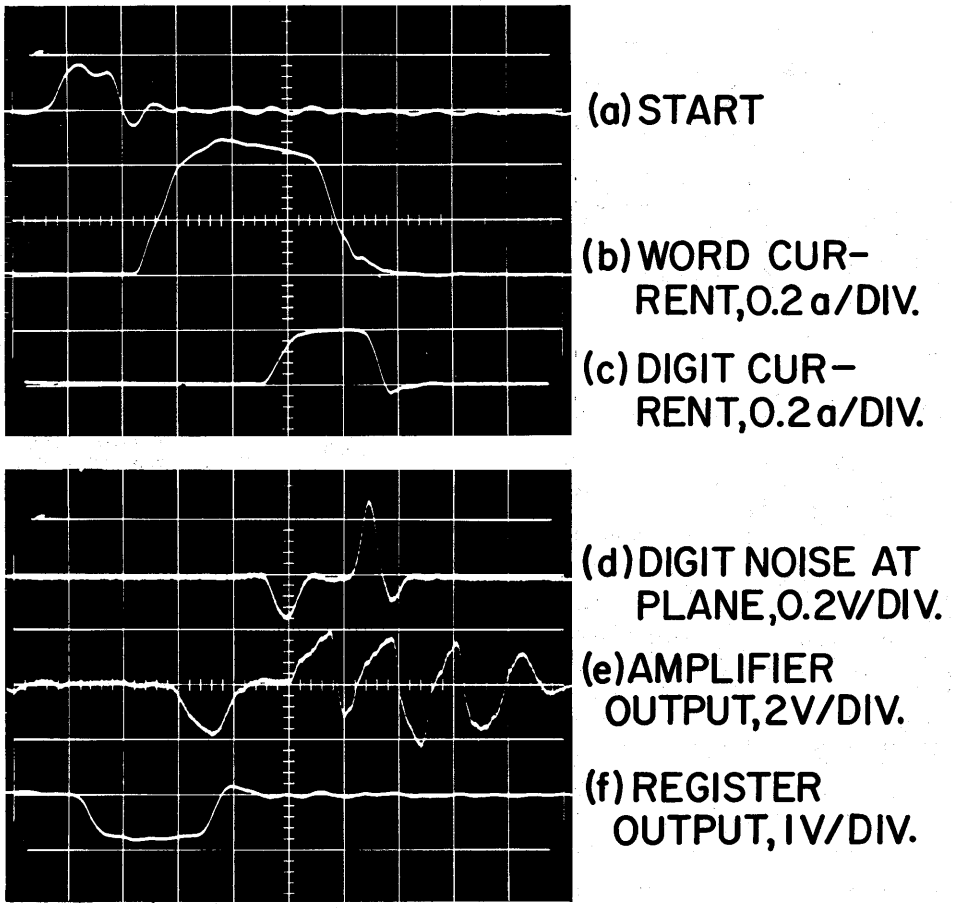



Figure 15. Selected waveforms during regeneration of a ONE.

*Creep Tests*

Adjacent-word disturb tests were made in all four corners of both plates and in the central portion of the top plate. The test consisted of writing alternate ONEs and ZEROs once in a word after having predisturbed the word by writing the opposite information 10 times. The word was then disturbed by writing the opposite information 25 million times in each adjacent word. Finally, the original word was read out and checked.

Initially, when every word was used with a word-current fall time of 4 nanoseconds, information was lost in a large number of bits. Improvements were made by using only odd numbered words and short circuiting unused word lines and by increasing the word-current fall time to the value shown in Fig. 15(*b*). Under these conditions a few bits failed in one corner of each plate. A total of 9 bits, distributed among 3 words, failed.* In every case the information lost was a ZERO. Apparently, loss of information was due to easy-axis skew in the magnetic film. The presence of skew is indicated by the difference in the amplitude of the ZERO and ONE signal in Fig. 16.

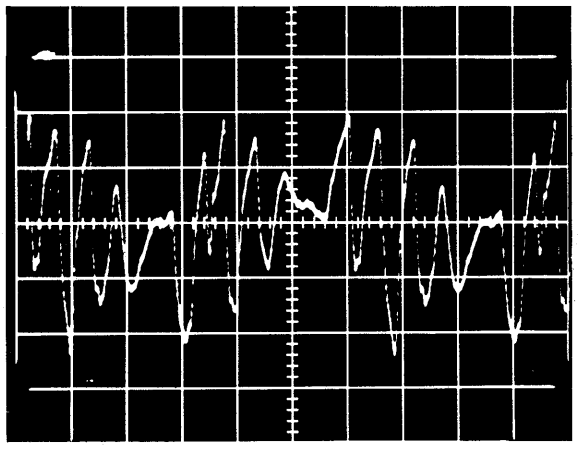


Figure 16. Amplifier output while reading ZEROs and ONEs alternately with a 60-nanosecond cycle time.

*It later developed that a partial short between two word lines caused part of the failures.

## CONCLUSIONS

These experiments show the feasibility of a 256-word scratchpad memory with an access time of 30 nanoseconds. By using faster transistors, now available, this value should be reducible to 25 nanoseconds. The read/write cycle time, however, will still be limited by the amplifier recovery so that with the best transistors available it appears that 60 nanoseconds are required.

## REFERENCES

1. H. Amemiya et al, "High Speed Ferrite Memories," 1962 Fall Joint Computer Conference Proceedings.
2. G. J. Ammon and C. Neitzert, "A 125-Nanosecond Thin Magnetic Film Scratchpad Memory System," Internal RCA Report, Sept., 1964.
3. M. M. Kaufman, L. Dillon and G. J. Ammon, "Tunnel Diode Memory," 1964 IEEE International Convention Record.
4. A. M. Bates and F. P. D'Ambra, "Thin Film Memory Drive and Sense Techniques For Realizing a 167-Nsec Read/Write Cycle," 1964 Solid-State Circuit Conference Digest.
5. D. Seitzer, "Amplifier and Drive Circuits for Thin Film Memories With 15-Nsec. Read/Cycle Time," *IEEE Transactions on Electronic Computers,* Dec. 1964.

# IMPACT OF SCRATCHPADS IN DESIGN: MULTIFUNCTIONAL SCRATCHPAD MEMORIES IN THE BURROUGHS B8500

Simon E. Gluck
*Burroughs Corporation*
*Great Valley Laboratory*
*Paoli, Pennsylvania*

The B8500 Modular Data Processing system is the latest design in the rapidly growing family of Burroughs Modular Computers. As in previous modular systems,[1,2] the Burroughs Corporation has found it expedient and efficient to utilize scratchpad memories to enhance the performance of the computer and other modules. This paper will describe in detail the application of multifunctional scratchpad memories in the computer module of the B8500 system.

The overall system design of the Burroughs B8500 is described in a separate paper presented at this conference.[3] It will suffice to review only the basic characteristics of the system as background for this paper.

The B8500 is an advanced design, high-performance modular data processing system. Three modules comprise the major building blocks of the system: the computer module, the input/output module, and the memory module.

## COMPUTER MODULE

52-bit words: 48 data, 3 control and 1 parity bit
Push-down (Polish) stack of operands to help implement arithmetic operations
Absolute, indirect and five forms of relative addressing:
- Base Index Register
- Base Data Register
- Base Program Register
- Self Relative
- Program Reference Table

Unlimited Index Registers
Binary arithmetic: 200-nanosecond addition (single precision integer)
Multiple multifunctional scratchpad thin film memories
Associative memory
Full repertoire of arithmetic, logical and character handling function
Memory protection registers

## INPUT/OUTPUT MODULE

512 independent simplex channels
Multiple 52-bit words of thin film buffering per channel
One 104-bit descriptor word per channel

Linked descriptors provide I/O processing with minimum computer intervention
Parallel byte transfers—variable from 1 bit to 51 bits
Maximum throughput of 590,000 bytes per second
Data transfer cycle time per byte (any size less than full word), 1.7 microseconds
On-demand servicing of input and output devices

## MEMORY MODULE

Thin film, destructive readout
4096 words of 208 bits: each memory word contains 452-bit computer words.
Cycle time 500 nanoseconds: access time 300 nanoseconds for 208-bit word
16 module system capability:
65,536 — 208-bit words
262,144 — 52-bit words
Capability of executing lengthy memory-oriented descriptors independently of, and concurrently with, computer module processing
Computer and I/O modules may total 16: at least one of each type must be included. Memory modules may be added up to 16 modules: (262,144 — 52-bit words)

The remainder of this paper will be confined to a discussion of the scratchpad memories that are part of the B8500 computer modules. The buffer memory of the I/O module could also be considered scratchpad, but due to the limitations of time and space, it will not be discussed.

## INFLUENCE OF MEMORY MODULE DESIGN UPON SYSTEM

The B8500 memory module concept, with its efficient "four-fetch" organization, provides the rationale for much of the scratchpad utilization in the computer module. One memory address descriptor, transmitted to all memory modules along a common communications bus, will select the addressed module, and within that module will select one of 4096 208-bit words. This composite memory word, containing 4 computer words, data or instructions, is available for transmission to the computer module 300 nanoseconds after the arrival of the address. Stored in the read-write register, the 4 52-bit words are sequentially transferred (four-fetched) to the requesting module at the rate of one full parallel 52-bit word every 100 nanoseconds. Including transmission time of the address descriptors and the returning information, 4 52-bit words can be requested from a memory module and received at a computer module in a total of 1.0 microsecond, or an average of 250 nanoseconds per word. Similar speeds are available for writing; "four-store" is also possible in the B8500 memory modules.

This four-fetch, four-store capability influences to a great extent the use, size and organization of scratchpad memories for the B8500. In a gross sense, it may be stated that one of the uses of scratchpad is to buffer the four-fetches until they are needed, or, in the case of four-store, to buffer words of data until they are assembled into four-word blocks for transmission to a memory module for storage.

## SCRATCHPAD CONCEPTS

At the risk of being pedantic, let us discuss for a moment the definition of scratchpad memories. Universally accepted definitions are difficult to find, but the following statement will represent the scratchpad concept to the majority of system designers:

> Scratchpads are small uniform access memories, with access and cycle times matched to the clock of the logic, and which are closely coupled to the source and/or sink of the data.

This definition studiously avoids the inclusion of the functions to which the scratchpads are applied. It also does not mention the form of implementation. These variables are left to the ingenuity of the system and circuit designers. In the latter case, implementations may vary from discrete component flip-flop registers to thin film uniform access memories.

Historically, core memories were used as scratchpads in computers where the bulk memories were magnetic drums. IN the case of the Burroughs D825 system, the scratchpad is thin film; the bulk memory is core. Implementation may eventually travel a full circle; with the advent of inexpensive integrated circuit flip-flop registers, the cost per bit of this implementation may become comparable with high-speed thin film memories.

## SCRATCHPAD APPLICATIONS

Functionally, scratchpads have been used for a variety of purposes. A major application is the buffering of information flow among the main memory of computers, computational elements, and input/output elements. This use also provides a speed conversion between data source and sink. Look-ahead designs, in which block transfers of instructions and data minimize memory accesses and transfers, have used scratchpad memories.

A major utilization has been the storage of intermediate arithmetic or logical results from a computational unit, minimizing the time and program steps required to transfer information to main memory and retrieve it when later needed. An outstanding example of this application is the use of a thin film scratchpad for the "last-in, first-out" stack in the Burroughs D825.

Finally, in many cases, economy dictates the use of scratchpad memories. Many registers, formerly implemented with flip-flops, are now stored as words in scratchpad memories. Typically, such registers are utilized for index words, base registers, real-time clocks and similar relatively infrequent usages. While it is true that these registers could be stored in main memory, the lower access time of the scratchpad makes the information available more quickly and doesn't tie up main memory address logic and communications lines.

## USE OF SCRATCHPAD MEMORIES IN THE B8500

Now to specifics: the use of scratchpad memories in the Burroughs B8500 computer module. Figure 1 represents a block diagram of the computer module. The major elements of the computer module are self-evident from the drawing but special attention should be given two blocks: the Local Thin Film Memories #1 and #2. These are two scratchpad memories; multifunctional in application, identical in physical construction and speed, different in word size and word length. Tied in with scratchpad #2 is a small 28-word associative memory (19 bits per word) whose use enhances the utilization of the scratchpad memory by providing content addressing as well as the conventional binary coded word addressing capability.

It must be emphasized that each of the two scratchpad memories contains its own independent addressing logic, sense amplifiers, and read/write registers. Each of the memories is available to the two processing elments of the computer module: the Arithmetic and Logic Unit, and the Address Arithmetic Unit.

The rationale behind the inclusion of local scratchpad memories in the B8500 computer module encompasses many of the reasons previously stated. Foremost among them, however, is the need for buffering of four-fetches of instructions and data in advance of their use, i.e., look-ahead. Also important are its uses as storage for intermediate results, as an economical implementation for registers and counters, and for the extension of the push-down stack.

The use of two scratchpad memories, rather than one common unit, is necessitated by the concurrency of operations in the computer module. Areas of each memory are assigned in a manner which permits simultaneous operations of the most frequently used functions. Ideally, it would be desirable to have a multiplicity of scratchpad memories; one for each function. Cost, space and power considerations prohibit such extravagance now. Future adaptations, to be discussed in later papers, may offer some hope in this direction.

## SCRATCHPAD PERFORMANCE CHARACTERISTICS

The B8500 scratchpads are implemented by magnetic thin film techniques developed and organized into linear-select memory arrays at the Burroughs Defense and Space Group at Paoli. To realize the high-speed access requirement of 45 nanoseconds, the reading function is nondestructive, eliminating the need for a restoring write cycle when data are to be retained unchanged.

Insertion of new data into the local memories (writing) can be accomplished within the 100-nanosecond clock period of the computer module. For both read and write, the memory cycle time is 100 nanoseconds, permitting synchronous operation with the logic of the computer module operating at 10 megacycles per second clock rate.

Addressing of all words of each of the two memories is performed through explicit binary coded addresses, which may be generated by the subcommand control hardware or included in a field of data or instruction word. Twenty-eight of the 44

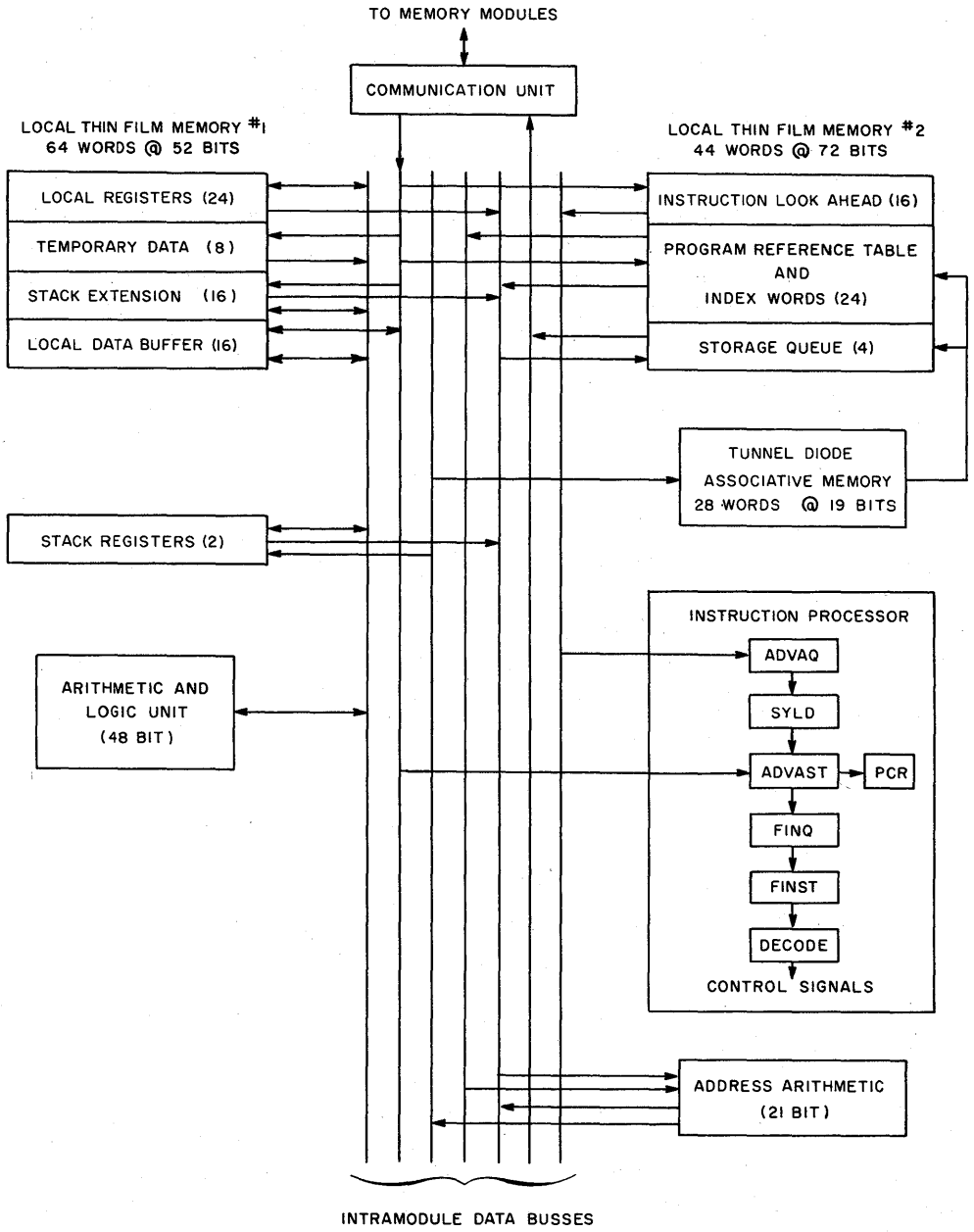



Figure 1. B8500 computer module block diagram.

words in scratchpad #2 can also be addressed by discrete output lines from the associative memory.

## SCRATCHPAD CONFIGURATIONS

### *Scratchpad #1*

Local Scratchpad Memory #1 contains 64 words of 52 bits each. This memory is utilized for four major functions:

1. Locally Used Registers and Counters
2. Temporary Data Storage
3. Stack Extension
4. Local Data Buffer

*Registers and Counters.* In this 24-word portion of #1 memory are stored a multiplicity of registers and counters that do not require frequent access by either the program or the hardware, yet which must be readily available within a clock cycle when addressed. Economy is the rationale in this

case; cost, power and space savings over flip-flop implementation are evident. Typical among the words stored in this memory area are: Interrupt Return Register, Base Interrupt Address Register, and Interval Timer.

*Temporary Storage.* This area is used to store literals and portions of multisyllable instructions. The latter usage stores syllables for the period of time from their detection and preliminary processing at the Advanced Station (ADVAST) until they are required at the Final Station (FINST). Eight 52-bit words are reserved for this function.

*Stack Extension.* Sixteen words in #1 scratchpad memory provide additional depth for the Polish stack above that available from the two hard stack registers associated with the arithmetic unit. Additional stack depth is automatically accomplished by automatic storing and fetching from the stack extension in #1 to an area in the memory modules; the depth of the stack is limited only by the total capacity of the memory and the permissible area assigned to it by the Executive and Scheduling Program (ESP). Four-fetch and four-store are used in stack transfers to minimize the traffic on the inter-module communication buses.

*Local Data Buffer.* This 16-word section of #1 is unspecialized scratchpad. The area is not reserved for a specific category of information but can be utilized under program control for storage of any data word or field. It is, however, the only scratchpad area that is capable of buffering four-fetch and four-store of data. Specific instructions are included in the machine repertoire to permit manipulation of data to and from the Local Data Buffer.

### *Scratchpad #2*

Local thin film memory #2 possesses the same performance characteristics as #1 but contains 44 words of 72 bits each (Fig. 2). The additional word length is required so that it can be utilized with the associative memory; 52 bits hold a normal computer word, while the remaining 20 bits contain an absolute memory address to which it may eventually be sent for storage (two bits are control bits).

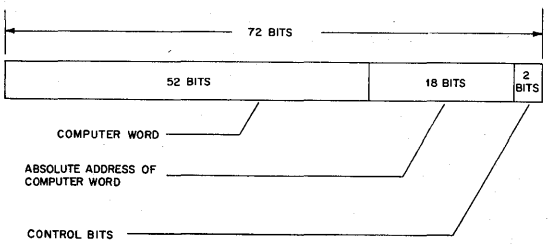



Figure 2. Scratchpad #2 word format.

Three functional areas are contained within this memory:

1. Instruction Look-Ahead
2. Program Reference Table and Index Words
3. Storage Queue

*Instruction Look-Ahead.* Sixteen words of #2 scratchpad are used to store four-fetches of instruction words transmitted from the memory module in advance of their use in the instruction processing section of the computer module. This area, called Instruction Look-Ahead (ILA), can hold up to four such four-fetches of "packed" instructions. (Instruction words of 52 bits in the B8500 contain 1, 2, or 4 12-bit syllables.) Only 52 bits of the 72-bit words available are utilized in ILA.

*Program Reference Table and Index Words.* The 24 words in #2 scratchpad devoted to the storage of Program Reference Table (PRT) lines and index words utilize all 72 bits; 52 for the normal word and an additional 20 bits for the absolute memory address and 2 control bits. A word stored in this area can be addressed in either of two ways; by explicit addressing or by selection by an output of the associative memory. The PRT entries (copies of PRT lines in main memory) and index words are stored interchangeably within this 24-word area.

Index words provide an increment to an address accumulator to point to a specific memory location. Index words stored in #2 contain an index value, an increment and a limit field withing the first 48 bits. The PRT word may contain, within its first 48 bits, the starting address and an upper limit for a data area main memory.

The PRT word may also point to the starting address or the entry point of a procedure. (These are called Data Descriptors and Program Descriptors, respectively, in the B5500 nomenclature.) Appended to each type of word is an 18-bit field representing the absolute address in a memory module where it has been or will be stored.

When a word is inserted into this 24-word area a copy of its absolute address is placed in the 20-bit word field described above, and also into one of the 19-bit words of associative memory. As instruction steps are decoded in the instruction proc-

essor, those instructions calling for an index word or a PRT reference send the calculated fetch address (absolute address) both to the communications unit and to the comparison register of the associative memory. If the associative memory contains the identical address, a word drive line from the associative memory will read out the proper word in the related 24-word portion of #2 memory. The selected word (and its 18-bit address) will be read out nondestructively, and the request to the communications unit is canceled. Such a sequence takes only 100 nanoseconds compared with the 600 nanoseconds (minimum) required if the words were to be fetched from a memory module, or an even longer time if the 24 words were searched and compared sequentially.

The 18-bit address field of the 72-bit #2 memory is required when the words referenced by the associative memory must be returned to a memory module. The associative memory used in the B8500 cannot have its contents read out like a conventional memory. When the logic requires storing of an Index or a PRT word in main memory following its access from #2, the 18-bit field is used to provide the communications unit an absolute address for the store function.

*Storage Queue.* The 4-word portion of the #2 designated for use by the storage queue is similar to the Index and PRT area. In both cases 72 bits are used and reference is achieved either by explicit addressing or by selection via the associative memory. The storage queue contains words destined for storage and their absolute address. Since the storage function has the lowest priority in the communications unit, words are retained in this area until service time is available. In a manner identical to that described for the Index and PRT area, data being fetched are checked against the contents of the storage queue by the use of the associative memory. This use is not included primarily to save time (although it does) but, more importantly, to ensure that the "newest" data are fetched to the computer. Fetching of data from a main memory location about to be updated by a word awaiting service in the storage queue would provide incorrect information to the program.

## SUMMARY

This paper has presented an example of the application of scratchpads to the computer module of a large processing system. The utilization of multifunctional scratchpad memories in the Burroughs B8500 system has enhanced the performance of the system and has resulted in significant savings of space, power and hardware.

## ACKNOWLEDGMENTS

Recognition and thanks must be directed to the many systems designers involved in the B8500 evolution. Paramount among these are Richard Bradley, George Barnes, Albert Sankin and Richard Stokes.

## REFERENCES

1. James P. Anderson et al., "The D825-A Multiple-Computer System for Command and Control," *AFIPS Proceedings, Fall Joint Computer Conference,* 1962.

2. R. V. Bock, "An Interrupt Control for the B5000 Data Processor System," *AFIPS Proceedings, Fall Joint Computer Conference,* 1963.

3. James D. McCullough, Kermith H. Speirman and Frank W. Zurcher, "A Design for a Multiple User Multiprocessing System," this volume.

# SCRATCHPAD-ORIENTED DESIGNS IN THE RCA SPECTRA 70

A. T. Ling
*Radio Corporation of America*
*Camden, New Jersey*

## INTRODUCTION

As data processing applications become more sophisticated, so do the computers that carry them out. Real-time, time-sharing, data communications, mass random access, multiprocessing, and other advanced computer concepts depend largely on a new level of program and data control within the processor.

One of the primary design objectives of the RCA Spectra 70 family was to provide the many capabilities needed to handle these complex new applications while also simplifying the user programs needed to keep track of data and the condition and state of programs in the machine. The family concept itself added to processor complexity. To achieve program compatibility within a family, a total system specification—complete and complex in its functional requirements—must be adopted.

To accomplish these objectives, the capabilities of Spectra 70 processors and their software systems were greatly expanded over second generation RCA computers, especially in the areas of data, program, and input/output control. The ultimate effect of this increased complexity was that the total system threatened to become a greater burden than could be handled at current logic speeds.

During the system concept and basic design phases of the Spectra 70 program, advanced technology provided solutions to many or the problems encountered. Research and development efforts and new manufacturing techniques greatly improved magnetic memory cost, speed, packaging and reliability. These factors made it possible to use small, extremely fast, scratchpad memories as integral parts of computer organization.

General-purpose commercial processors with scratchpad memories did not appear on the computer market until 1959. The reasons for this were both technological and conceptual. Technologically, the small, fast memories needed for effective scratchpad applications were too costly and not reliable enough. Conceptually, the computer was not complex enough to justify the large number of machine registers required. Memories used in computers, in addition to the main memories, were small magnetic core stacks. The purpose of these auxiliary memories was to buffer data between peripheral devices and the central processors to accomplish speed, compatibility and to translate code or format. These moderate-speed, small-capacity memories were equal to or slower than the central processor's main memory.

System designers, however, proposed or implemented systems, on an experimental basis, using small fast memory as auxiliary storage. The PILOT

system of the National Bureau of Standards[1] used a diode-capacitance memory as a secondary storage. An early development using the list processing technique had been proposed[2], using a small memory for list address scanning. The speed of such a memory device was to be 10 to 20 times faster than the main memory.

Since 1959, a number of general-purpose commercial computers have been offered, featuring small fast auxiliary memories. Honeywell 800, UNIVAC 1107, Burroughs D825, and RCA 3301 are typical examples. Generally, the auxiliary memories in these machines have been used for address registers, operand stack, index registers, index increments, and program interrupt storage. Although this did extend machine capabilities, it had not fully exploited the scratchpad potential. Computer instructions normally do not operate directly on the scratchpad registers without transferring the contents to the execution unit. Further, program manipulations of scratchpad contents require transfers into the main memory through special move instructions.

## GENERAL PROCESSOR ORGANIZATION

The general organization of the Spectra 70/45 and 70/55 processors (Fig. 1) is divided into three major functional units—memory, input/output, and central processing unit. The central processing unit

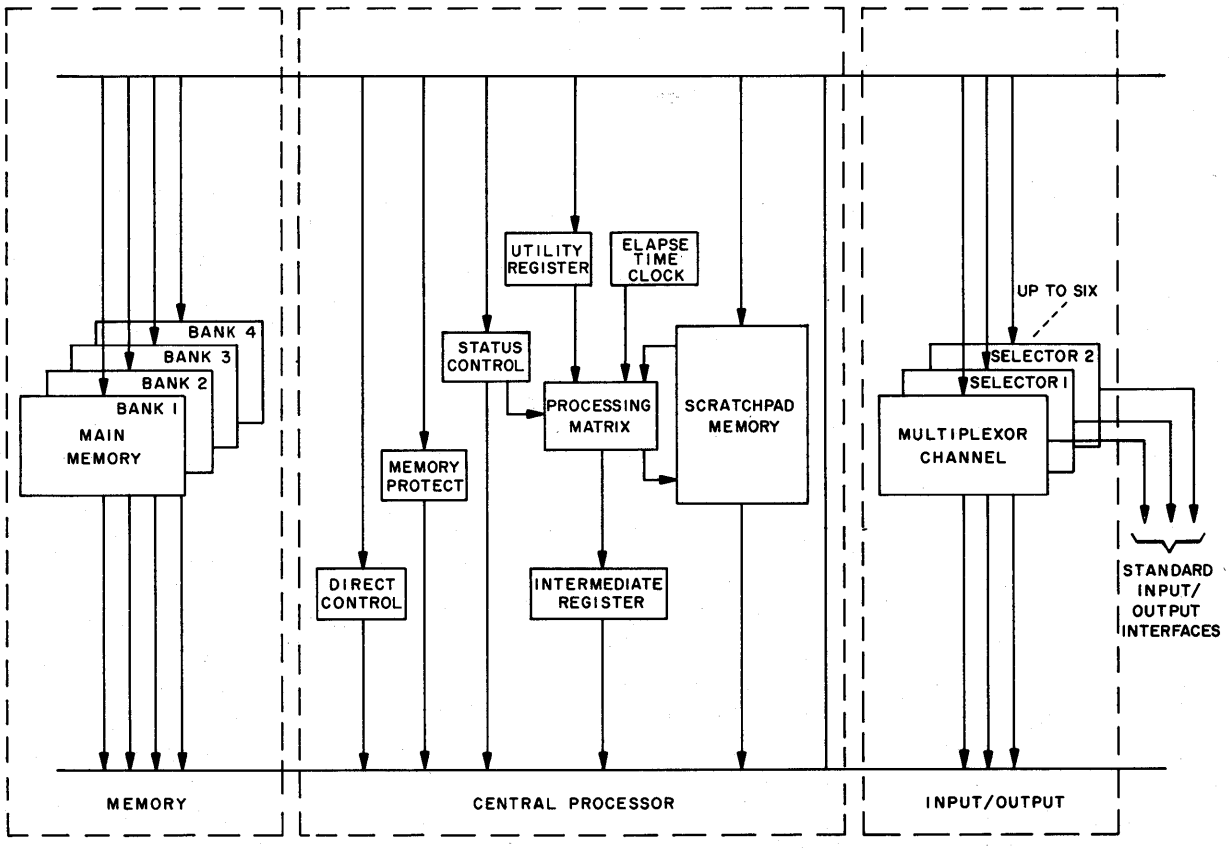
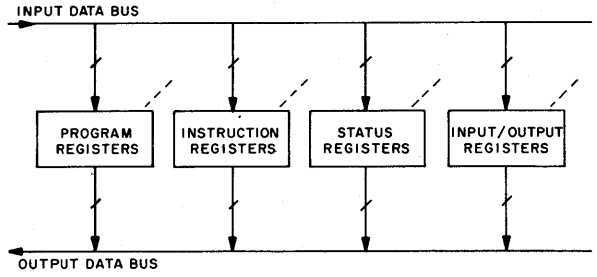



Figure 1. General processor organization.

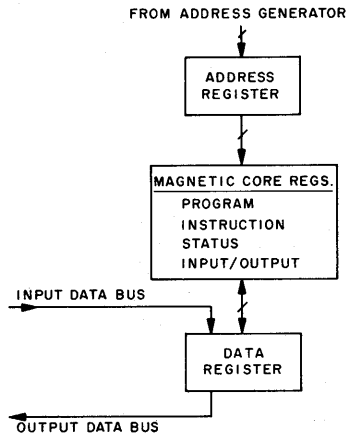
can be connected to a maximum of four main memory banks. Each memory bank is a complete unit, capable of operating independently. The input/output unit consists of a multiplexor channel that can control up to 256 simultaneous input/output operations and a variable number of selector channels, each controlling one input/output operation. These input/output channels—multiplexor and selectors—interface the peripheral devices through a standard input/output interface. This standard input/output interface allows interchangeable connectivity of any peripheral device with the input/output channels of all models within the Spectra 70 family.

## SCRATCHPAD-ORIENTED DESIGN

The design of the RCA Spectra 70/45 and 70/55 processing unit is based on a unique concept that makes the scratchpad an integral part of the computer organization, not an adjunct. In essence, the scratchpad design is a was of arranging the machine registers with the logic structure built around it. As shown in Fig. 2a, all the machine registers are arranged into arrays which interact with the rest of the processor logic through an input and output bus. The assembly is implemented by a small magnetic core scratchpad memory as shown in Fig. 2b.




(a) FLIPFLOP IMPLEMENTED REGISTER SCRATCHPAD




(b) MAGNETIC CORE IMPLEMENTED SCRATCHPAD

Figure 2. Block diagrams showing various scratchpad implementations.

By adding a processing matrix to the scratchpad memory (Fig. 3), data from registers in the memory are circulated through the processing matrix before it is regenerated back into the registers. Logic functions such as data incrementing, extraction, movement, table look-up, and status recognition can be executed with this circulation path. In fact, these single-operand logic functions are sufficient for processing interrupt servicing of a multiplicity of simultaneous input/output operations. This circulation path is passive and self-sufficient and therefore does not disturb any other register content in the machine.

The normal compute mode processing, however, requires the complete repertoire of single-operand and double-operand logical and arithmetic functions. These are achieved by adding two registers to the data structure. The Utility Register (UR) serves as a buffer for the second operand input to the processing matrix, while the Intermediate Register (IR) acts as an output intermediate storage. These two registers, in effect, supply a second and parallel circulation path through the data bus in the course of instruction execution. Of course, besides the basic data structure, there are a number of miscellaneous control registers and counters necessary to maintain an adequate performance level.

In the single-operand mode of operation, the scratchpad memory operates typically in a read/wait/regenerate cycle. The scratchpad memory register location is addressed by the address register. The scratchpad cycle waits while the content of the addressed register is read out onto the data register. The output of the data register is modified by the processing matrix, independent of the contents of the Utility Register. The processing matrix output passes through the regeneration matrix into the regeneration input of the scratchpad memory to start the regenerate cycle. The wait between the scratchpad readout and regenerate cycle varies from zero to some finite time, depending on the logic function to be performed in the processing matrix. For example, the wait time will be zero, if the data register output is transferred to the output data bus only.

In the case of double-operand operations, the second operand is either previously set into the Utility Register by a prior scratchpad memory register transfer cycle, or concurrently set into the Utility Register from the input data bus, with the data source from the main memory unit, the Intermediate Register, or other processor logic, while the first operand is being read out from the scratchpad.

## SCRATCHPAD MEMORY FUNCTIONS

Machine registers can be classified into four groups according to their functional assignments. A

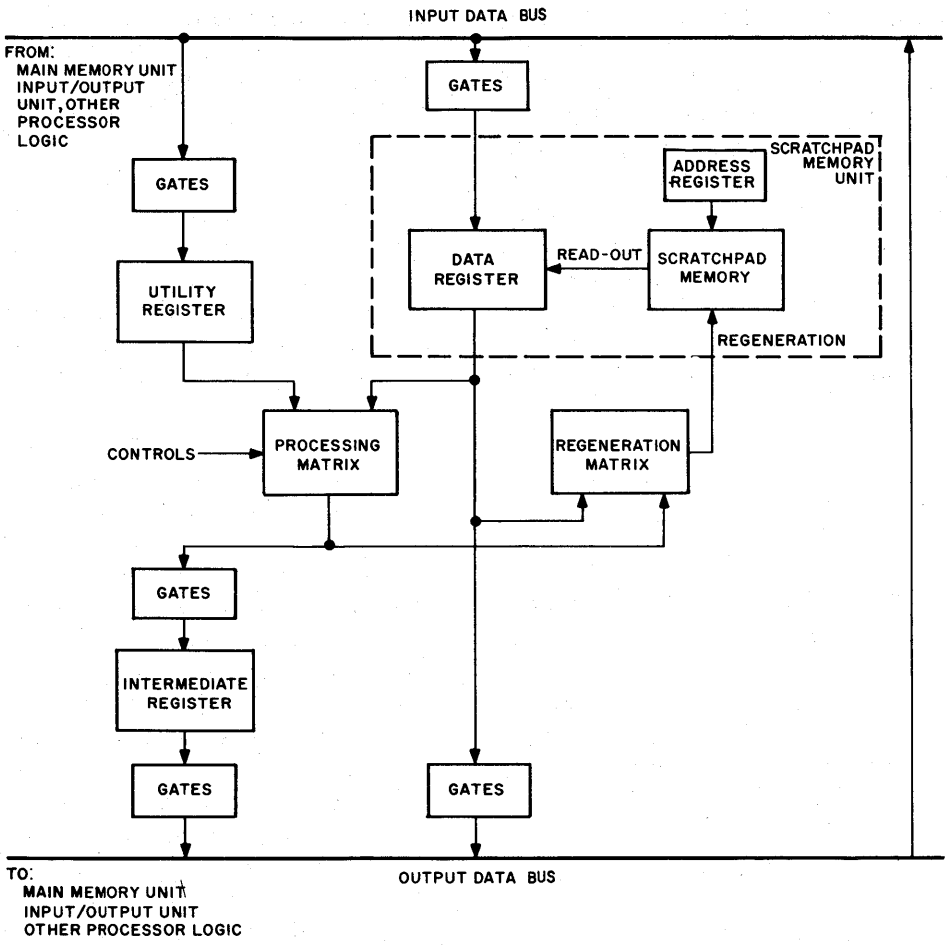



Figure 3. Scratchpad-oriented processor data structure.

machine register is a 32-data-bit register used either as an address register or data register, depending on the specific assignment.

1. *Program Registers* hold information regarding the complete and current status of a program they are assigned. These registers are required for execution of a program, from instruction to instruction. *One set of program registers in the Spectra 70 family is called a program state.* A full-program state consists of 16 General Registers, 3 program control registers—namely, the Program Counter (PC), Interrupt Status (IS), and Interrupt Mask (IM)—and 4 double-length Floating Point General Registers. The General Registers are for assignment, within the current user program, as indexes, base addresses, data accumulators, or intermediate storage registers.
2. *Instruction Registers* are used in the execution algorithm of an instruction as defined by an operation register. They are not held over or used by a subsequent instruction, but are treated as utility registers without permanently assigned functions and are used in various ways as needed in the course of algorithm execution.
3. *Status Registers* hold the current status of the processing unit. They are used by the control structure to indicate the program interrupt status and the current program connection of the processing unit.
4. *Input/Output Registers* are used for the simultaneous operation of peripheral devices. In the Spectra 70 system, each operation requires a subchannel consisting of a set of three registers. A fourth register is used for final standard interface status, reporting at the termination of an operation.

In addition, utility registers facilitate data assembly and chaining operation execution. These registers are preset at the initiation of an input/output operation in the normal processing mode and are sufficient to execute a sequence of input/output operations with branching and chaining ability according to a control word list in the main memory.

## SPECTRA 70 SCRATCHPAD

These four classes of machine registers are arranged into a scratchpad memory array as shown in Fig. 4. The scratchpad memory is large enough to provide one set of input/output registers for the multiplexor and up to six selector channels. In addition, a significant feature is that there are four program states, namely four sets of program registers,

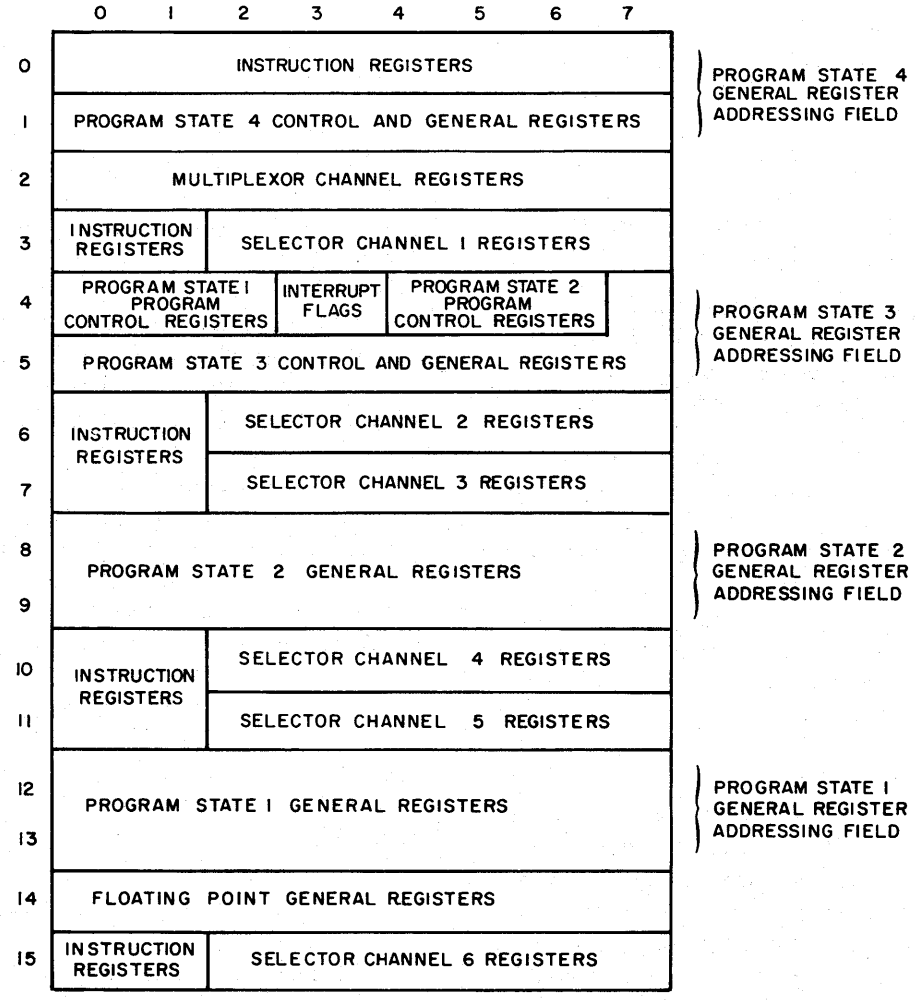



Figure 4. Spectra 70/45 and 70/55 processor general scratchpad layout.

provided in the scratchpad. The four program states —two executive program states and two problem program states—are not full program states as defined earlier but are tailored to the software system assignment described below.

One set of the four program states is selected at a time according to the setting of a Program State Register to be operated on by the processing matrix. The addressing of the program registers by the processing control logic is automatically modified by the Program State Register at the scratchpad address generation to select the proper register according to the rule of the scratchpad layout. In this way, the normal instruction execution algorithms

are concerned with only one program state, namely the one determined by the setting of the Program State Register. By switching the setting of the Program State Register between instructions, the processing logic automatically operates on a different program state, and consequently a different program. However, the status of the previous program state is preserved. Since normal instruction execution is confined to registers within a program state, each program state is protected from others in the computer. However, two privileged instructions, called "Load Scratchpad" and "Store Scratchpad," are provided for priviliged program access to any or all of the registers in the scratchpad memory. These instructions can only be executed when the program is assigned a privileged operation status, as in the case of the executive program.

The spectra 70 software system assigns the four program states for the following program functions:

Program 1—Processing State
Program 2—Interrupt Response State
Program 3—Interrupt Control State
Program 4—Machine Condition State

Essentially program states 1 and 2 are alternate object programming states for multiplexed program ming. Program state 2 is for interrupt processing or input/output programs and does not have a separate set of Floating Point General Registers. When a floating point instruction is issued in program state 2, it is executed with the Floating Point General Registers of program state 1. In fact, floating point instructions in either state operate with the same set of Floating Point General Registers. This is also true for program states 3 and 4.

Program states 3 and 4 are privileged executive program states. Because of their functions, either program state requires less than 16 General Registers. On the other hand, program state 3 must have the ability to manipulate the three Program Control Registers that are not normally directly addressable. The scratchpad arrangement takes advantage of the extra general register designation capability in the instruction format to allow program state 3 to address the Program Control Registers of states 1 and 2 as if they were General Registers. This allows the executive program to use the full power of the instruction complement in manipulating these registers.

Similarly, program state 4 can address the first six instruction registers used for instruction execution algorithms and the two operand address registers directly. Thus, the program register capacities of the four program states are:

| | Program State | | | |
|---|---|---|---|---|
| | *1* | *2* | *3* | *4* |
| General Register | 16 | 16 | 6 | 5 |
| Program Count Register | 1 | 1 | 1 | 1 |
| Interrupt Status Register | 1 | 1 | 1 | 1 |
| Interrupt Mask Register | 1 | 1 | 1 | 1 |
| Floating Point General Register | 8 | — | — | — |
| | 27 | 19 | 9 | 8 |

TOTAL CAPACITY: 63 registers

Standard input/output operation control in the Spectra 70/45 and 70/55 processors requires a set of four registers—a Control Address Register, two Channel Control Word Registers, and a Data/Status Register. In the case of the multiplexor channel, the first three registers for each of up to 256 subchannels are stored in a hidden portion of the first main memory bank. A specific set is transferred into the multiplexor channel set of four input/output registers in the scratchpad for servicing. These registers are restored before servicing the next subchannel. In addition, a duplicate set of four input/output registers is used for termination reporting to the software system. In the case of the selector channel, two additional registers are used for prefetching of chain controls or other utility functions. The Input/Output Register capacities are:

| | Channel Number | | | | | | |
|---|---|---|---|---|---|---|---|
| | *0* | *1* | *2* | *3* | *4* | *5* | *6* |
| Basic Set | 4 | 4 | 4 | 4 | 4 | 4 | 4 |
| Termination Set | 4 | — | — | — | — | — | — |
| Utility Set | — | 2 | 2 | 2 | 2 | 2 | 2 |
| | 8 | 6 | 6 | 6 | 6 | 6 | 6 |

TOTAL: 44 I/O Channel Registers

The balance of the scratchpad registers are used for instruction and status registers. Since the result of every instruction is stored in program registers, the instruction registers normally convey no additional information needed for subsequent instructions. The instruction registers are truly utility registers. Both the 70/45 and 70/55 processors are free to use these registers for instruction algorithms in the most efficient way.

Because of their performance requirements, most status registers are in flip-flop registers outside

the scratchpad memory. But pertinent information from these registers is also stored in the Interrupt Status Registers in the program states. The Interrupt Flag Register, however, is in the scratchpad, addressable by program state 3 as a General Register.

## PROGRAM STATE SWITCHING AND THE INTERRUPT SYSTEM

The above-described processor design strives to minimize the necessity of storing connective program information outside the scratchpad at the completion of each instruction. The program state can be switched by merely changing the Program State Register setting. Program state switching can be caused by an occurrence of one of the 32 program interrupt requests, by a privileged instruction called "Program Control," or as a result of an operation on the Interrupt Mask Register (IM) or the Interrupt Flag Register (IF).

The interrupt system is connected to program states 3 and 4. When one or more of the program state switching reasons occur in the system, the processor normal mode automatically switches into the program interrupt processing procedure at the termination of the current instruction. This procedure involves the following hardware steps:

1. Program interrupt requests, if any, are entered into the Interrupt Flag Register (IF) in the scratchpad memory for permanent recording until they are individually processed.
2. The current program state's Interrupt Mask Register (IM) determines the allowability of the outstanding interrupt requests in the Interrupt Flag Register (IF). This permits complete programming control of the interrupt request processing. It is possible that none of these requests will be allowed by the current program state. When this happens, no program interrupt occurs but the interrupt requests remain outstanding in the Interrupt Flag Register.
3. The allowable interrupt requests are processed by priority positions. Then a priority weight is generated and stored in an assigned general register in program state 3 or program state 4, according to the interrupt request.
4. The corresponding interrupt request bit in the Interrupt Flag Register (IF) is then reset. Only one interrupt request, the highest priority of the allowed requests, is processed at one time.
5. According to the interrupt request, certain additional preparatory functions are performed and the interrupt state registers of the outgoing and the incoming program states are updated.
6. The setting of the program state register is changed to program state 4 for system error interrupt request or to program state 3 for all others. Instruction execution proceeds next with the new program state register setting.

## THE SCRATCHPAD MEMORY UNIT

To implement the Spectra 70 scratchpad oriented design, a single 128-word magnetic core memory unit was selected for both the 70/45 and 70/55 processors. The memory unit uses magnetic cores of 30 mils outside diameter and 10 mils inside diameter, wired in a 2-core-per-bit fashion. A three-wire core threading format is used—two for access currents and one for a common sense/digit signal. The word oriented linear selection system is accomplished by a transistor-diode cross-bar matrix.

Each memory cycle accesses 32 bits in parallel plus 2 parity bits. The access time is 120 nanoseconds with a complete read-regenerate cycle time of 300 nanoseconds. The scratchpad memory unit can operate in one of two modes—straight read/regenerate mode or read/wait/regenerate mode. The "wait" in the latter case is for the processing matrix operation completion.

Figure 5 is a simplified block diagram showing the addressing, the sense, and the regenerate systems. The addressing system uses two drivers per word, each driving into a voltage switch circuit. A discharge circuit helps to speed up the discharging of the voltage switch line at the end of the regenerate cycle. The sense system consists of 34 difference amplifiers, with strobing. Regeneration information does not come automatically from the data register but is supplied, at the regenerate inputs with two digit drivers per bit, by a regenerate matrix in the processor logic as shown in Fig. 3. The data register is under processing unit control, independent of the memory cycle timing. The data register is used

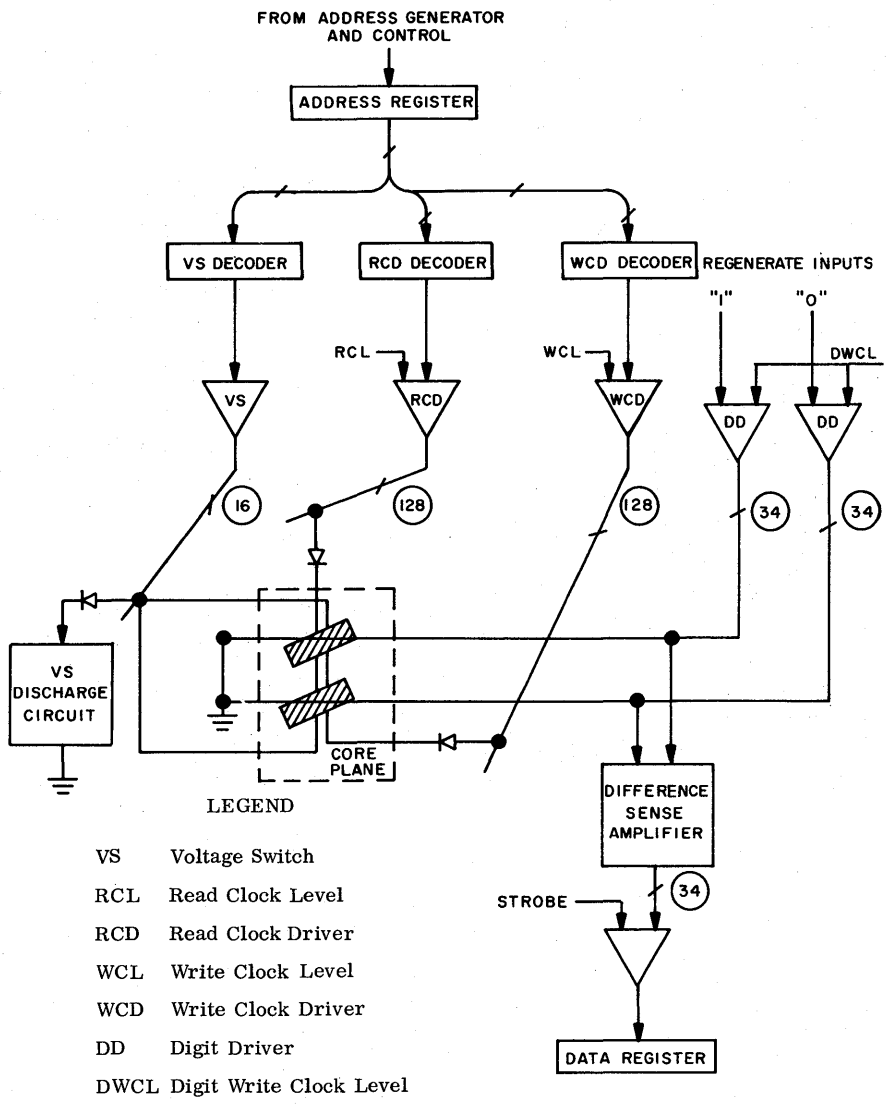



Figure 5. Spectra 70/45 and 70/55 scratchpad memory block diagram.

for logic operations without regard to the scratchpad memory operation.

The actual scratchpad memory unit is packaged in a standard Spectra 70 printed circuit logic platter (Fig. 6). The unit itself occupies approximately two-thirds of the 18 x 18-inch platter; the rest of the platter contains processor logic.

## DESIGN CONSIDERATIONS

The design considerations in selecting the method of scratchpad implementation are cost, packaging and size, accessibility, speed of operation, and reliability. The choice of implementation consists of an integrated semiconductor flip-flop register array or a magnetic core device. Integrated semiconductor register banks are not available for design consideration at this time.

### *Cost*

A simplified cost comparison between flip-flop implemented scratchpad and magnetic core implemented scratchpad is illustrated in Fig. 7a. The cost ordinate is in normalized cost unit for comparison purpose. The number ordinate is in number of 32-

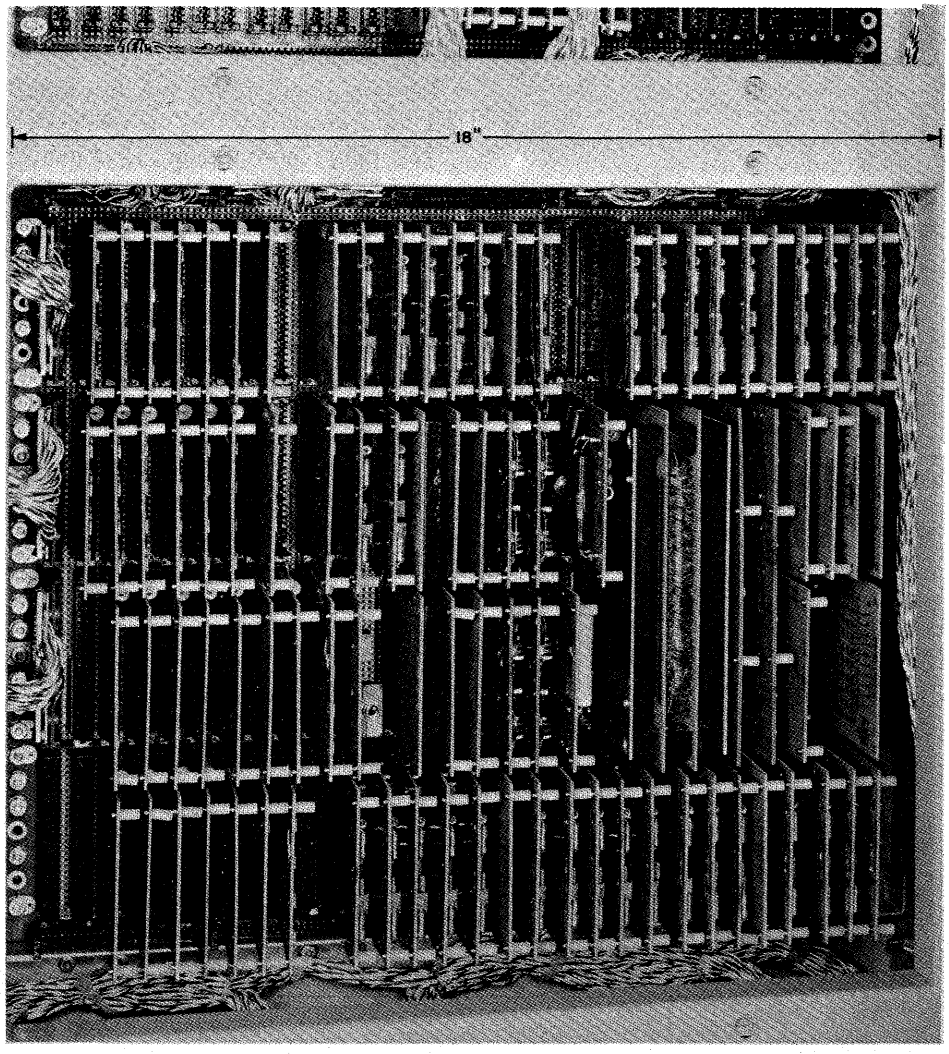
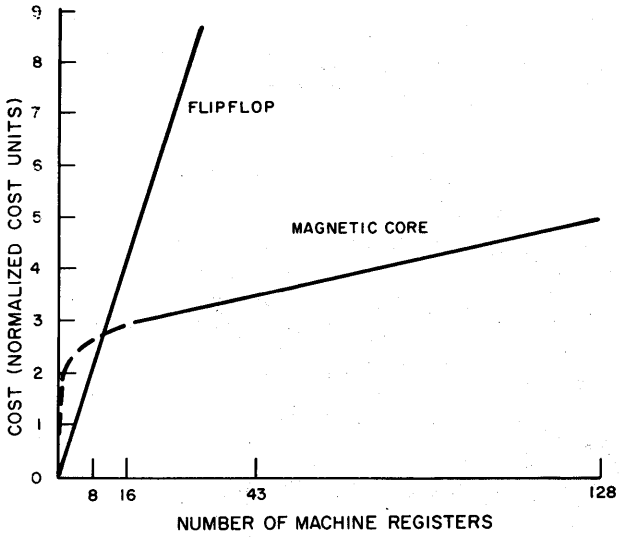



Figure 6. Scratchpad memory in the 70/45 processor.

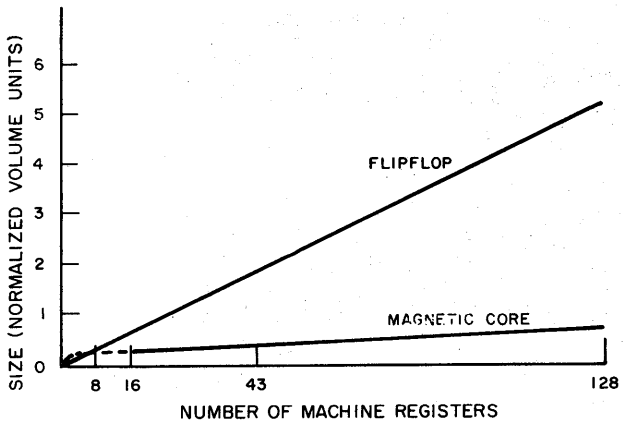
data-bit registers. Each of the two curves is really a family of curves as a function of operation speed. The exact relative positions would vary accordingly. In this illustration, the magnetic core device has an access time of 120 nanoseconds and a cycle time of 300 nanoseconds. The flip-flop scratchpad is the Spectra 70 integrated circuit packages with 3-pair access, 4½-pair cycle. Here the crossover point is in the range of 8 to 16 registers. Thus, when a computer cost and performance specification requires 16 or more registers, as in the case of Spectra 70, a magnetic core device becomes increasingly advantageous. The minimum number of registers for a single program state and 3 input/output channels for the 70/45 processor is greater than 40. By using a magnetic core scratchpad, it is possible to provide 128 registers for a small cost increase. The extra capacity makes it economically possible to provide four program states and a multiplicity of simultaneous input/output controls. Furthermore, extra utility registers with almost logic circuit speed access, overlapping with the comparatively slower main memory, have permitted use of algorithms to achieve higher instruction execution speed than otherwise would be possible.

*Speed*

The scratchpad memory access and cycle speed must be economically achievable by current state-of-the-art standards, that is, in the 100-nanosecond access speed range. For the logic circuit speed of 25




(a) COST COMPARISON




(b) PHYSICAL SIZE COMPARISON

Figure 7. Flip-flop scratchpad vs magnetic core scratchpad.

nanosecond worst-case pair delay, the flip-flop scratchpad has an access speed advantage by a factor of 2 and, more importantly, by a factor of more than 3 for cycle time. The practical number of flip-flop scratchpad registers is limited, however, due to packaging size and interconnection degradation. The choice of scratchpad speed for a cost range and the required system performance level is effected by the proper balance of the main memory unit, the control logic structure, and the scratchpad register speeds.

*Accessibility*

Accessibility considered here is the number of registers that can be accessed simultaneously. The magnetic core scratchpad has an access limitation of one register per cycle. The flip-flop scratchpad can allow multiple access via additional data paths of more than one register at a time, thus, faster execution possibilities by parallel operations. This accessibility, however, is accompanied by packaging and loading complications due to irregularity of the register array and extra interconnection paths that ultimately reflect on a higher packaging cost. The processor designs of the 70/45 and 70/55 have achieved the performance requirement, even with the accessibility limitation of the magnetic core scratchpad. Cost and an orderly structure to minimize interconnection problems are more important considerations for these two processors.

*Packaging and Size*

Since the scratchpad is packaged in standard size logic plug-in modules on a fixed depth printed circuit platter, size comparison is the same whether in terms of volume or platter area. The relative physical size comparison is shown in Fig. 7b. It is important to remember that the size comparison is a function of the system packaging design. For the Spectra 70 platters and integrated circuit packages (14-pin dual 4-input gate flat packs), the magnetic core scratchpad has significant size advantage for these two processors, despite the fact that the special memory circuits are not integrated packages. The data structure is thus kept physically smaller. The interconnection degradation and platter layout problems of such large numbers of registers are lessened. The regularity of the logic structure is conceptually suitable for a higher degree of integrated hardware and batch fabrication techniques.

*Reliability*

The orderly array of magnetic cores, replacing a large number of individual logic gates that implement the flip-flop scratchpad, offers advantages in reliability and maintainability by way of interconnection efficiencies. Like a main memory unit, the magnetic core scratchpad is a nonvolatile storage. Power failure or power turn-off has little effect on the information contents in the scratchpad.

## SUMMATION

The magnetic core scratchpad provides an abundance of machine registers. These registers are or-

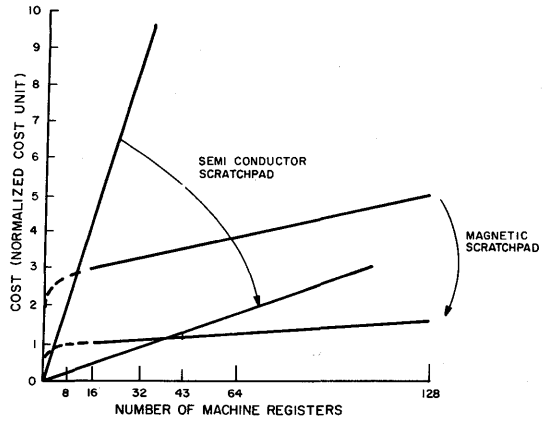



Figure 8. Batch fabrication influence on flip-flop scratchpad vs memory device scratchpad.

ganized in a scratchpad-oriented design to provide the cost and performance level requirements of the Spectra 70/45 and 70/55 processors. The multiplicity of registers and the unique feature of four program states offer design advantages in both hardware and software.

For the hardware design, the many instruction registers allow the implementation of fast but more complex execution algorithms than otherwise would be possible. The seven sets of input/output registers provide controls for a high degree of input/output simultaneity. The orderly data structure eases the high-speed packaging requirement.

For the software design, the four program states make possible new programming horizons, particularly in executive software and in multiprogramming. The multiple sets of general registers more effectively yield usable registers for the individual program for minimizing register reassignment.

For computer emulation, the four program states make possible effective and efficient cooperation between hardware and software. It allows one program state to operate in an emulator mode while the other states can operate in the more efficient Spectra 70 mode or any special mode advantageous to the specific emulator. The abundance of high-speed registers provides emulator algorithm speed. For instance, frequently used translation table and controls can be held in the scratchpad, instead of in the comparatively slower main memory.

## A LOOK AHEAD

The magnetic core scratchpad is not a design panacea and is not necessarily applicable to all performance ranges. In low-performance processors, the scratchpad can be implemented more economically as part of the main memory while still providing adequate execution speed. In higher performance processors, the designer needs to investigate the tradeoffs in the scratchpad arrangement, particularly the speed consideration. The scratchpad is likely to be arranged into small modular units with faster access and cycle times. The multiprogram state concept may be further expanded for multiprogramming, having one object program state per active program in the machine. Hierarchical memory control and addressing and program stacks might be added to the scratchpad functions.

Integrated monolithic circuits are here now, but the computer manufacturing field is at the threshold of the batch fabrication[3] technological breakthrough which will affect machine design significantly in years to come. On the horizon are low-cost, miniature, high-speed semiconductor circuits which provide hundreds[4] and even thousands[5] of circuit elements integrated and interconnected within a single chip. A similar batch fabrication breakthrough is likewise expected on magnetic devices.[6] Choices for future scratchpad implementation may include more than the two mentioned in this paper.

## REFERENCES

1. A. L. Leiner et al, "PILOT — The NBS Multicomputer System," *Proceedings, Eastern Joint Computer Conference,* 1959.

2. R. L. Wingington, "General Purpose List Processor," *IEEE Transaction,* June 1963.

3. *Proceedings of the National Symposium on the "Impact of Batch Fabrication on Future Computers," Los Angeles, April 1965.*

4. E. A. Sack, R. C. Lyman and G. Y. Chang, "Evolution of the Concept of a Computer on a Slice," *IEEE Proceedings,* Dec. 1964.

5. E. Keonjian (ed.), *Microelectronics — Theory, Design, and Fabrication,* McGraw-Hill, New York, 1963, chap. 2, pp. 18-19.

6. R. J. Petschauer and H. S. Kukuk, "Batch Fabrication of Magnetic Computer Memories," *Proceedings of the National Symposium on the "Impact of Batch Fabrication on Future Computers," Los Angeles, April 1965.*

# SCRATCHPAD MEMORIES AT HONEYWELL: PAST, PRESENT, AND FUTURE

N. Nisenoff
*Honeywell Incorporated*
*Electronic Data Processing Division*
*Wellesley Hills, Massachusetts*

## INTRODUCTION

The computer industry, during its short life of approximately 20 years, has seen many innovations, evolutionary trends and developments, and has even accommodated several major revolutionary technological breakthroughs. One of these revolutionary-evolutionary developments has been the increasing use of small memories as scratchpads.

A working definition of scratchpad memories is given in this paper. From that definition three specific, nonoverlapping applications are derived. One of these, the use of scratchpad memories for control purposes, has been developed and employed at Honeywell for some time. Other manufacturers have also employed scratchpad memories—for example, Burroughs in their D825, UNIVAC in their 1107, and RCA in the larger members of their SPECTRA 70 systems. Discussions of this particular aspect of scratchpad, or control, memory forms the major portion of this paper. In separate sections, the specific uses of control memory in the Series 800, the Series 200 and the Honeywell 8200 are described.

In the succeeding section some prognostications concerning the state-of-the-art in control memories, based upon the results of extrapolations to 1970, are presented. A simplified tradeoff analysis is also developed and discussed.

## DEFINITION — AN EXPANDED OR EXTENDED VIEW

Scratchpad memories have been with us for a long time. Let us first define the term scratchpad memory. A scratchpad memory is that fast access portion of a data processor or a computer in which modifiable information is temporarily stored. The key words to be noted in this definition are "modifiable information," and "temporarily stored."

In particular, the general functions performed by scratchpad memories can be categorized as:

1. Manipulative storage for "overhead" processes.
2. Faster access to a limited (although not necessarily a contiguous) portion of main memory.
3. Extended storage facilities in the processing element itself for minimizing the restoring of temporary or intermediate results.

This paper will discuss in some depth the use of this form of memory for "overhead" processes—control memory. This particular use of scratchpad memories has been extensively used in the Honeywell 800, the Series 200 family, and in the Honeywell 8200.

## THE HONEYWELL SERIES 800 FAMILY OF DATA PROCESSING SYSTEMS[1]

The Honeywell 800 was announced in early 1958. At the grossest level, the system appears to be a very conventional system of that time period. However, this is misleading. It was a dynamic and revolutionary jump beyond the then current state-of-the-art in computer system organization. A few of the H-800's contemporaries and competitors are characterized in Table 1.

Table 1. H-800 Competitor Analysis.

| Machine | First Installation Date | Monthly Rental *thousands of dollars* | MCT *μsec* | Number of Addresses | Add Time *μsec* | Word Size *B* |
|---|---|---|---|---|---|---|
| IBM 7070 | 6/60 | 11–31 | 6 | 1 | 72 | 50 |
| H-800 | 12/60 | 19–35 | 6 | 3 | 24 | 48 |
| GE 225 | 1/61 | 2.5–25 | 18 | 1 | 36 | 20 |
| RCA 301 | 2/61 | 3.5–25 | 7 | 2 | 210 | 6 |
| G-20 | 4/61 | 14–22 | 6 | 1 | 27 | 32 |

The Honeywell 800 differs from machines of its own era, and indeed from most currently available systems which have been designed and built during the interim. Figure 1 presents a diagram of the Honeywell 800 system organization. In most respects it resembles a "conventional" computer. The difference is in the crosshatched area in the control unit which is referred to as the "multiprogram control."

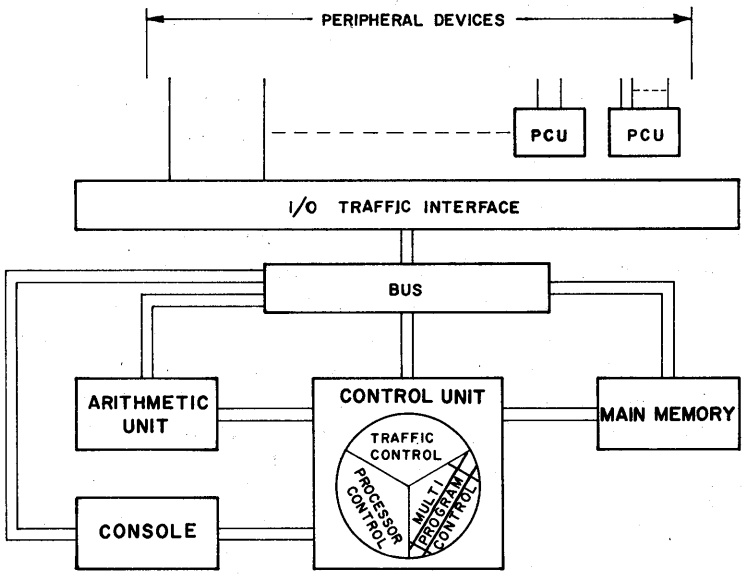



Figure 1. H-800 system organization.

The multiprogram control permits the total system to operate the I/O devices with a maximum efficiency as well as to time-share the arithmetic circuits among a maximum of eight separate and distinct programs.[2]

The physical implementation of the multiprogram control uses a 256-word scratchpad core memory.[3,4] This memory—a control memory—was designed with a memory cycle time equal in duration to the memory cycle time of the main high-speed core memory. However, to facilitate the interleaving of register accesses and to minimize dead time, the control memory timing was designed to be 180 degrees out of phase with the main memory.

Within this context let us consider the functions of the control memory. Table 2 presents a list of functions, each associated with a unique register or location, for a total of 32 locations. This set is referred to as a control group. Each control group of 32 registers operates and controls the entire processing system for a single program. Each control group provides the programmer with the powers and facilities of a conventional computer. Since there are eight such groups, the Honeywell 800 is eight com-

Table 2. List of Registers Contained in a Single H-800 Control Group.

| Function | Number of Registers |
|---|---|
| Sequence registers | 2 |
| Sequence history registers | 2 |
| Index registers | 8 |
| Unprogrammed transfer registers | 1 |
| Mask index register | 1 |
| Arithmetic control counters | 2 |
| General-purpose registers | 12 |
| Read/write address counters | 4 |
| TOTAL | 32 |

puters in one—certainly a unique design when it was introduced in 1958.

Each control group contains two sequence counters which permits the program to select and/or return to alternate sequences or paths of processing on an instruction-by-instruction basis. For each sequence counter in the system, there is a corresponding history register. These registers are used to store the contents of a sequence counter whenever a branch is taken by an instruction. This permits a first level of return from a subroutine and also assists in program debugging.

Most of the registers listed in Table 2 are self-explanatory; however, two items merit some further comment. The general-purpose registers, like the index registers, are used primarily for address modification. Their use differs from that of index registers in several important respects. For example, the contents of these registers can be incremented or decremented (to an upper limit of 31) following their use. These registers are employed primarily to address an operand or a result location in any bank of memory, but they may also be used as programmed counters, as temporary storage for the contents of other special registers, and for any other purpose the programmer may devise.

The other set of locations in the control memory which require additional comment are the read/write address counters. To understand their use we must first examine the input/output interface and the traffic control system.

The traffic control directs the time-sharing of memory by the peripheral units and the central processor. Its main object is the efficient use of the entire system according to a set of priorities which are derived directly from the nature of the peripheral equipment and are independent of the program.

The creation of a demand signal by any device is represented in Fig. 2 by the closing of the switch shown in the corresponding control stage. When any

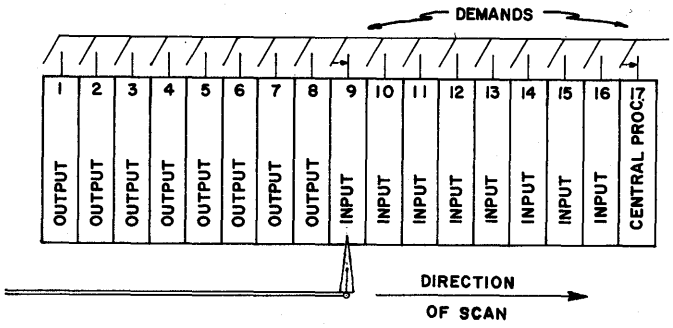



Figure 2. H-800 traffic control.

program has been turned on in the central processor, the switch corresponding to the central processor stage is continuously closed. Traffic control begins each scan at the left end of the band. It proceeds to the right, ignoring all stages which show no demand signal, until a demand stage is reached. This stage is allowed access to the main memory for one memory cycle only. Traffic control then returns to the left end of the band to begin the next scan. Since the control search is anticipatory, no system time is consumed in bypassing stages in which no demand exists.

Execution of a peripheral instruction automatically loads the data address into the proper read or write counter and initiates activity on the addressed peripheral device. The central processor is then free to continue instruction execution. Whenever the device is ready to transmit a word the stage switch is closed causing traffic control to allocate a memory cycle. The data word is transmitted using the address in the read/write counter and the counter is incremented by one, thus requiring no program reference beyond the original peripheral instruction.

If no peripheral device requires a memory cycle, it is given to the multiprogram control. The multiprogram control, in turn, allocates the memory cycle to the appropriate control group.

If only one control group (one program) is active, all cycles allowed the central processor are used in executing instructions from the active program. Since traffic control allows the central processor all available cycles except those needed to honor the intermittent demands of peripheral devices, this case represents that of the conventional single-program computing machine with the ability to implement input/output operations simultaneously with computing.

When more than one control group is active, central processor cycles must be shared among the sev-

eral programs. For example, if three programs are active, then one instruction is performed in turn from each. With the initiation of an instruction, all succeeding cycles must be devoted to the execution of this instruction until it is completed.

The true power of multiprogram control appears when an active program attempts to execute an instruction which cannot be implemented because of the unavailability of a system component. Sensing that the instruction cannot be executed immediately, multiprogram control places the control group in a "stall" condition. This condition indicates to multiprogram control that (*a*) this program, although still active, shall not be allowed any central processor cycles as long as the stall indication remains, and (*b*) when the channel and/or the device involved completes its present task, the stall condition shall be automatically removed and the program restored to its full active status.

Thus, when an instruction cannot proceed because of input/output conflicts with either the same or another program, the central processor cycles which it would have used are made available to the other active programs, enabling processing to proceed faster. The result of the operation of multiprogram control is that there is never an idle memory cycle as long as there is any active program in which an instruction can be executed.

In this respect the 800 automatically subdivides itself into as many computers as there are active programs.

## THE HONEYWELL SERIES 200[5,6,7]

The Series 200 has been designed primarily for business applications and for the mixed environment of business data, communication and scientific processing. In this environment the key performance dimension is throughput. High throughput, in turn, requires that the proper consideration or weights be given to three operations:

1. Inputting of data
2. Data manipulation
3. Outputting of data

The Honeywell Series 200 line of data processors is composed of five members; at the lower end of the spectrum is the H-120; this is successively followed by the H-200, the H-1200, the H-2200, and the H-4200, each more powerful in turn. Key attributes of each processor are shown in Table 3.

Table 3. Series 200 Members — Key Attributes.

| Processor Model | Main Memory Speed Cycle Time *per character*) | Memory Capacity *(thousands of characters)* | Number of Input/Output Trunks | Number of I/O Operations Simultaneous with Computing | Advanced Programming Instructions | Financial Edit Instruction | Multiply and Divide Instructions | Scientific Processing Instructions | Memory Protect Facility |
|---|---|---|---|---|---|---|---|---|---|
| 120 | 3 μsec | 2–32 | 4 controls in processor; 4 I/O trunks available | 2–3 | Op | Op | NA | NA | NA |
| 200 | 2 μsec | 4–65 | 8–16 | 3–4 | Op | Op | S | NA | NA |
| 1200 | 1.5 μsec | 8–131 | 16 | 4 | S | S | S | Op | Op |
| 2200 | 1 μsec | 16–262 | 16–32 | 4–8 | S | S | S | Op | Op |
| 4200 | 188 nanosec | 32–524 | 32–64 | 8–16 | S | S | S | S | Op |

S = Standard Op = Option NA = Not applicable

In the Series 200 the scratchpad memory, or control memory, is organized in a different manner than it is in the H-800. This is due, in part, to the fact that the Series 200 members are character-oriented rather than word-organized processors. Further, the Series 200 processors do not possess capability for parallel processing—i.e., the control memory is organized as a single control group (in H-800 terminology).

Physically, the control memory has a maximum of 64 locations, the number found in any one machine depending upon the model, options and features. The number of bits per position depends upon the size of the main core memory. Unlike the scratchpad memory in the H-800, the control memory in the various members of the Series 200 operates at a faster rate than the main core memory. This permits multiple scratchpad memory ac-

cesses during a main memory cycle. The speed ratio for each member of the series is shown in Table 4.

Table 4. Relative Speed Ratio of Control Memory to Main Memory for the Members of the Series 200.

| Processor | Main Memory Cycle Time / Control Memory Cycle Time |
|---|---|
| H–120 | 6 |
| H–200 | 4 |
| H–1200 | 3 |
| H–2200 | 2 |
| H-4200 | 3 |

As noted above, the Series 200 processors are character-oriented. This fact had a direct influence in the specification of the functions and number of registers per function which were designed into the scratchpad memory. These functions are listed in Table 5.

Table 5. Control Memory Function in the Series 200 Processor.

| Descriptive Functions | Number of Locations |
|---|---|
| A & B address registers | 2 |
| Sequence register | 1 |
| Change sequence register | 1 |
| R/W channel current location counter | 3–16 |
| R/W channel starting location counter | 3–16 |
| Work registers | 3 |
| External interrupt registers | 1 |
| Internal interrupt register | 1 |
| Floating point accumulation | 12 |

The Series 200 processors are not only character-oriented machines but they operate with instruction formats which may contain zero, one, or two addresses—thus the requirement for two address registers. Further, the input/output interface has been designed to work in conjunction with the control memory. The read/write channels (RWC's) are not scanned as the traffic control was scanned in the H-800, but in a fixed cyclic manner. The first memory cycle is offered to RWC 1. If this RWC cannot use the cycle it is utilized by the central processor. The next memory cycle is offered to RWC 2, etc. The I/O access is accomplished via two registers in the control memory which are directly associated with each RWC.

The remaining counters and registers listed in Table 5 are self-explanatory with the possible exception of the floating point accumulators. These registers are provided in the larger members of the Series 200 where the Scientific Option is available. The availability of four floating point accumulators permits the temporary storage of intermediate results which, if stored in and recalled from main core memory on a character-by-character basis, would be cumbersome and time-consuming. With the results in scratchpad, they are available for manipulation on a register-to-register basis.

The use of the scratchpad memory has allowed the designer to economically implement by hardware many conventional software functions with significant improvements in machine performance. We believe that the proper balance has been struck between cost and capability at a reasonable and marketable price.

## THE HONEYWELL 8200[8]

The recently announced H-8200 represents the intelligent integration of the Series 800 and the Series 200 into one system. It has carried the concept of a control memory to its next logical level of utilization—the delegation to a structured hardware element of those systems-control functions which can best be performed by hardware for the greatest speed increase per dollar of cost.

The H-8200 main memory has a cycle time of 750 nanoseconds and a scratchpad which will provide a speed ratio of control memory to main memory of from 4 to 6:1.

The H-8200 system's processor (see Fig. 3) contains 10 programming groups. Nine of these groups control active running programs, while the tenth is a master control group.

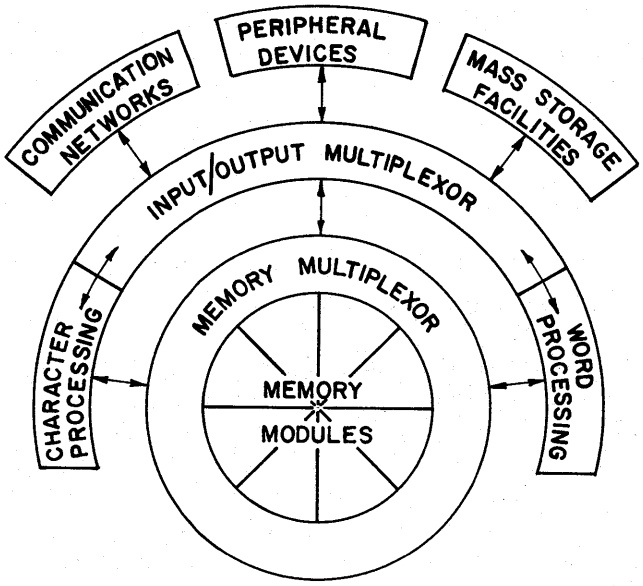



Figure 3. H-8200 system organization.

The function of the master control group is to provide intercommunication among the active programming groups—and thus, in fact, to monitor the entire system. Of the nine remaining groups, eight handle data and instructions in the H-800 fixed-word mode while the ninth group operates in the H-200 variable-length character mode.

A further function of the master control group is to manage the input/output facility as well. In particular:

1. It maintains identification information regarding memory partitioning.
2. It diagnoses program, peripheral and memory usage violations.
3. It responds to privileged operating system instructions.

The 10 program groups are physically realized in 3 distinct scratchpad memories. This distribution permits overlapping and concurrent operation. Two other scratchpad memories have been employed, one in the memory multiplexor and the other in the input/output multiplexor.

The scratchpad memory in the input/output multiplexor provides the following functions:

1. Read/write channel capability for both the character and word processors.
2. Reservation facilities on a read/write channel basis.
3. Peripheral barricading.

By utilizing scratchpad memories to a greater extent than ever before, the H-8200 design has minimized the memory storage requirement for overhead functions and at the same time produced a faster reacting system. This provides greater throughput, which in turn provides greater economy to the user.

## CONTROL MEMORY IMPLEMENTATION TRADEOFF ANALYSIS

The utilization of scratchpad memories at Honeywell has developed over an extended period of time. Their continued use has been reviewed and evaluated for each new product in which they have, or could have, been employed.

The decision has turned, and must always turn, upon one key point: Does the use of a scratchpad memory produce a quantitative increase in the overall performance of the system for the dollars involved? The correct decision involves the answers to many questions. For example, what set of functions will the scratchpad memory perform? Is the set of functions to be performed optimal? What effect will the inclusion of a scratchpad memory (with a given set of functions) have upon the software design? Does the cost of the scratchpad actually reduce the cost of the overall system, or does it increase it?

The approach followed by any group in answering these questions will depend upon backgrounds, personalities, availability of funds, etc.

Let us assume that the decision to employ a scratchpad memory for control purposes has been made and that the decision was based upon a preliminary systems analysis which delineated the functions which the unit would be required to perform. As a result of the specification, the size of the memory (and therefore its capacity in bits) was established.

The next step is to determine the speed of the unit, and the implementation technology.

Recall from Table 4 that the speed ratio between the control memory and the main memory for several members of the Series 200 ranged from 2 to 6. It is probable that this ratio in the future will tend toward the high end of the range, and possibly even extend beyond it. For discussion purposes, let us assume 4 as a minimum and 8 as a maximum. We now need a speed estimate for future main memory systems.

Several studies have been performed recently[9,10] which provide a reasonable engineering estimate of the speed of future main memory systems. One of these studies was performed by Honeywell's Electronic Data Processing Division under the sponsorship of the U.S. Army Electronics Command and produced a set of extrapolated computer characteristics. For one of these extrapolations, concerning the memory cycle time of future data processing systems, the data file has been expanded and updated, and a current extrapolation has been produced. It is shown in Fig. 4.

The data base for Fig. 4 represents 208 distinct machines introduced over a period of nearly 20 years. The machines contained in the data base represent, as far as memory technology is concerned, the full gamut of main memory implementation techniques from mercury delay lines to magnetic thin films.

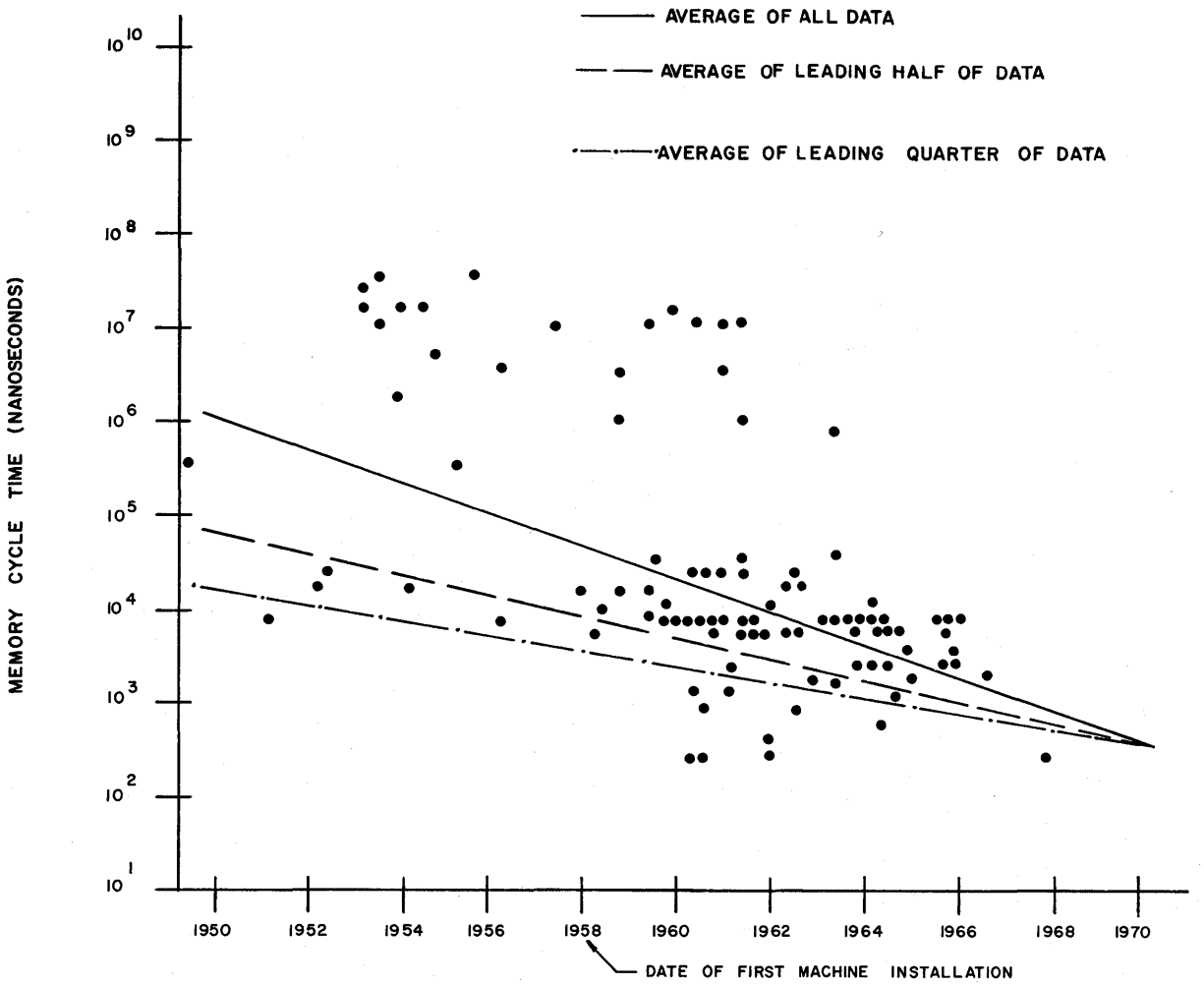



Figure 4. Memory cycle time extrapolation based on an expanded MILDATA data base.

The upper curve in Fig. 4 represents the least squares fit to the total data base. Once plotted, all points above that curve were eliminated from the data base and a second curve fitted; finally the points above the second curve were dropped and a third curve was generated. The three curves thus represent:

1. The average for the total data base.
2. The average of the leading or, in this case, the lower half of the data base.
3. The average of the leading quarter of the data base.

If a specific datum is selected and the three extrapolations of Fig. 4 are plotted for this data, as in Fig. 5, a trend indicating the leading edge of the technology for the particular time period should emerge.

This hoped-for trend in memory cycle time does not appear to be emerging. This implies, if the extrapolation technique is valid, that a saturation or slowing down of technological developments in this area is occurring.

This type of extrapolation has been made by the writer three times during the past six years. Figure 11 summarizes the three results for the extrapolated memory cycle time obtained for the year 1970. The number of machines considered in each extrapolation is listed in the second column of Table 6.

Returning to the detailed specification of the control memory, we can estimate a main memory cycle time of approximately 300 nanoseconds for 1970. Based upon this figure and the speed ratio

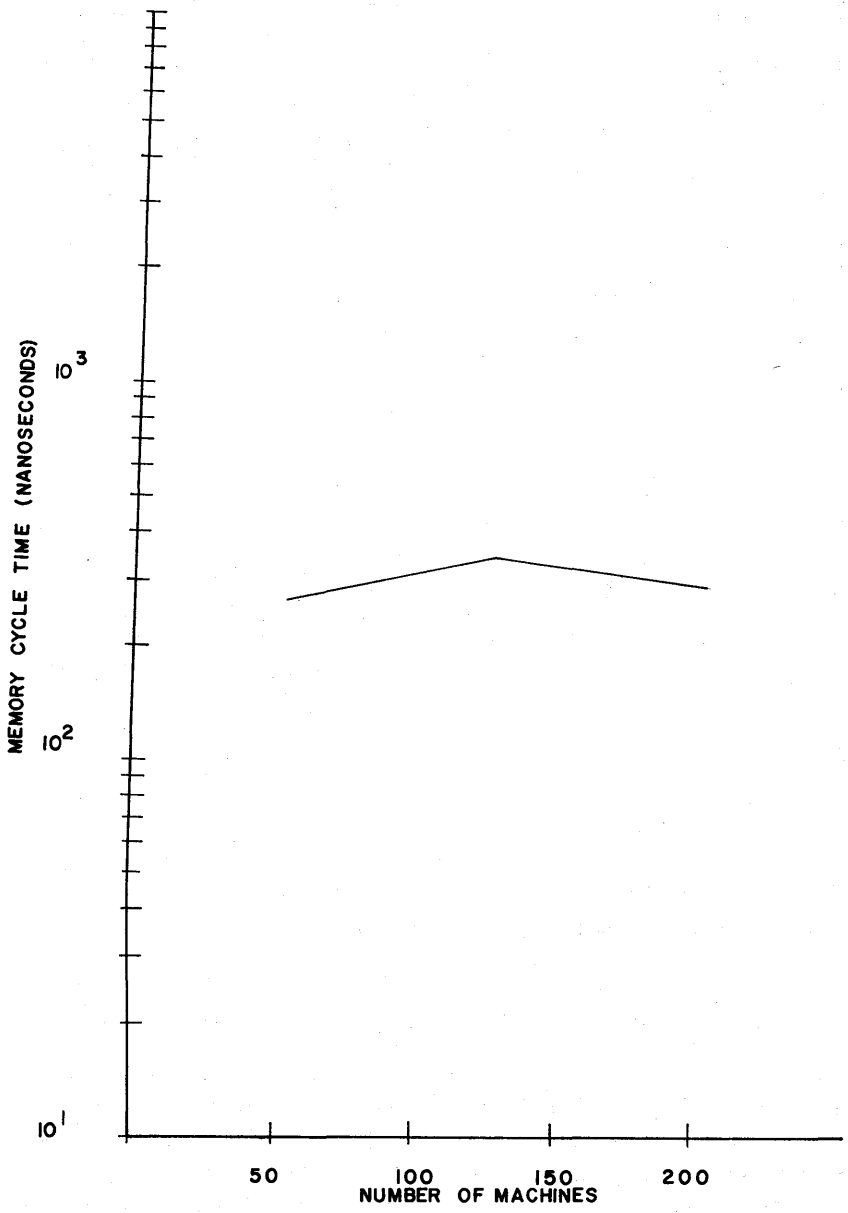



Figure 5. Extrapolated memory cycle time for machines introduced in 1970 (based upon an expanded MILDATA data base).

Table 6. Comparison of Three Extrapolations Concerning Memory Speed in 1970.

| Date | Data Base *(number of machines)* | Average Memory Cycle Time 1970 *μsec* |
|---|---|---|
| 1959 | 30 | 0.25–0.50 |
| 1963 | 154 | 0.30 |
| 1965 | 208 | 0.330 |

developed above (control memory cycle time should be 4 to 8 times as fast as the main memory cycle time), the memory cycle for a control memory in 1970 should range from approximately 40 to 75 nanoseconds.

The capacity and the speed of the unit has been established. The final question to be considered in this paper—the selection of the control memory implementation technology—can now be examined.

The control memory could be implemented via one of several different physical implementations—for example, semiconductive flip-flop elements or magnetic thin films.

The general cost (in dollars) to produce (on a production basis) a control memory of a given capacity and speed can be defined as:

$$C = I + D \cdot B \tag{1}$$

where

$C$ = total product cost in dollars,
$I$ = initial "investment" cost per machine for the particular technological approach (the sum of the costs for the drivers, sense amplifiers, address and data registers, etc.) in dollars,
$D$ = cost per bit of storage (in dollars), and
$B$ = number of bits.

For two distinctly different methods of implementation the crossover point occurs when the total costs of the control memory, in production for both, are equal.

Therefore:

$$I_A + D_A B = I_B + D_B B \tag{2}$$

Solving for $B$, we obtain:

$$B = \frac{I_A - I_B}{D_B - D_A} = \frac{\Delta I}{\Delta D} \tag{3}$$

$\Delta I$ and $\Delta D$ represent the differential in "investment" and cost per bit respectively.

Figure 6 presents Eq. (3) in a convenient form. A single example will suffice to explain its use. Assume we are comparing a magnetic thin film memory with a set of flip-flop registers for the control memory in a machine. Here $I_A$ will represent the peripheral cost to support a film memory for this purpose. In practical terms, $I_A$ is the sum of the costs for the memory drivers, sense amplifiers, address and data registers, etc. Hypothetically $I_B$ could be considered to be equal to zero. $D_A$ and $D_B$ would be the cost per bit for each approach. Summarizing:

$$I_A = \$10^3$$
$$I_B = 0$$
$$D_A = \$0.50$$
$$D_B = \$2.50$$

Therefore:

$$\Delta I = 10^3 \qquad \text{and} \qquad \Delta D = 2.00 \tag{4}$$

Using Fig. 6, we see for this example that the break-even point is at 500 bits. For larger capacities a core memory is more economical; for less capacity the active flip-flop implementation is more economical.

## SUMMARY AND CONCLUSIONS

Historically, Honeywell was the first to develop and utilize the concept of scratchpad memories for control purposes within its data processing systems. The H-800 and the Series 200 systems each employed different embodiments of the basic concept. Increasing utilization of the fundamental concepts both by other computer manufacturers and Honeywell (for example the expanded and extended control memory system of the H-8200) substantiates the growing recognition by systems designers of the usefulness of such devices.

Extrapolations of current trends with an eye toward future developments in scratchpad memories leads one to the definite conclusion that scratchpad memories will be a significant factor in the design of future systems. In substantiation, three facts can be noted: (1) the rapid and significant decrease in the cost of small high-speed memories; (2) the definite trend toward the design and development of more sophisticated and complex systems; and (3) the increasing desire of the systems designer to emancipate the programmer from the individual machine characteristics to the greatest extent possible.

## ACKNOWLEDGMENTS

The author wishes to express his appreciation to the members of Honeywell Electronic Data Processing Division who made the submission and presentation of this paper possible. In particular, Mr. John Gilson, Miss Sonya Shapiro and Mr. Robert Zinn showed a great deal of patience in reviewing and assisting in clarifying my thoughts during the preparation of this paper.

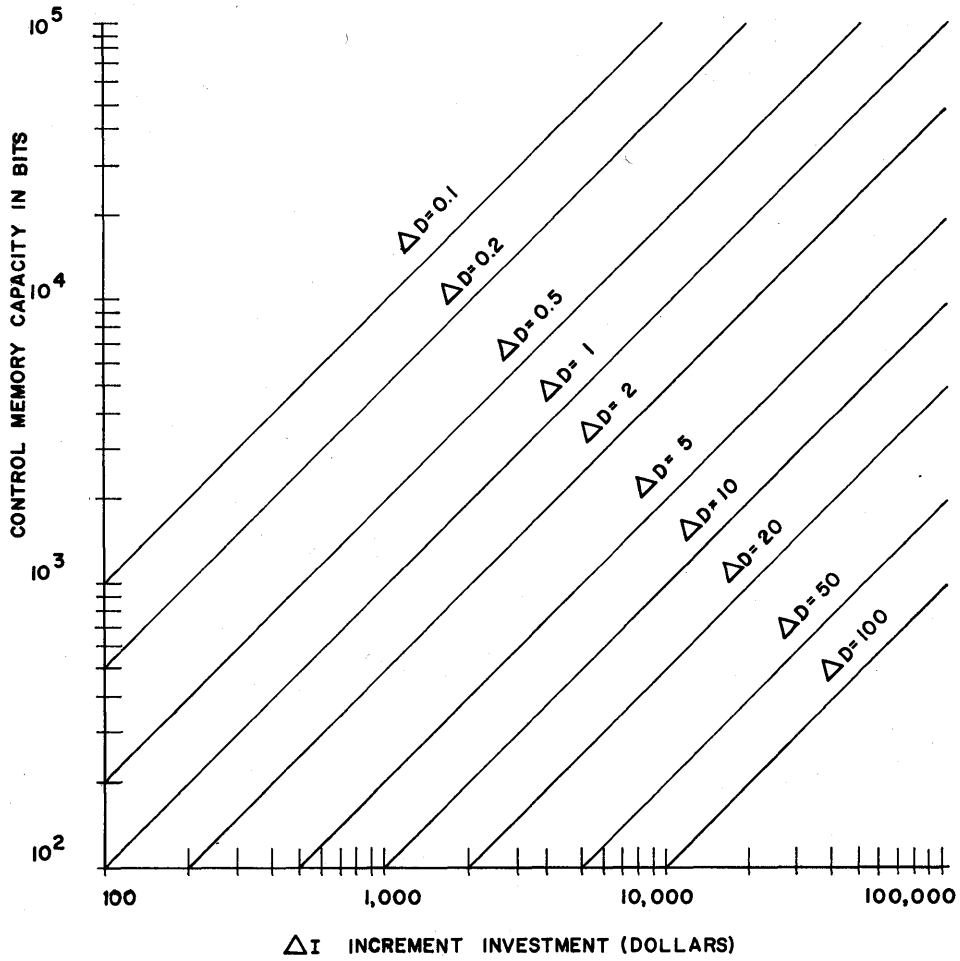



Figure 6. Differential cost analysis, comparing two distinctly different implementation techniques.

## REFERENCES

1. *Honeywell 800 Programmers' Reference Manual,* Honeywell Inc., 1964.

2. S. D. Harper, "Automatic Parallel Processing," *Proceedings of the Computer Data Processing Society of Canada,* 1960, pp. 321-331.

3. N. Lourie et al, "Arithmetic and Control Techniques in a Multi-Program Computer," *Proceedings Eastern Joint Computer Conference,* 1959.

4. H. W. Schrimpf, "Information Handling Apparatus for Distributing Data in a Storage Apparatus," U.S. Patent *3,142,043,* July 21, 1964.

5. *Series 200 Programmers' Reference Manual—Models 200/1200/2200,* Honeywell Inc., 1965.

6. *Series 200 Programmers' Reference Manual—Model 120,* Honeywell Inc., 1965.

7. *Series 200 Programmers' Reference Manual—Model 4200,* Honeywell Inc., 1965.

8. *Series 200 Summary Description—Model 8200,* Honeywell Inc., 1965.

9. N. Nisenoff, "MILDATA, An Optimizing Study of a Modular Digital Computer System," Final Report, Contract DA 36-039-AMC-03275(E), vol. II, 1965 (AD 462 043).

10. L. C. Hobbs, private communications.

# A BOUNDED CARRY INSPECTION ADDER FOR FAST PARALLEL ARITHMETIC

Emanuel Katell
*Electronic Associates, Inc.*
*West Long Branch, New Jersey*

## INTRODUCTION

This paper suggests a new mechanism for parallel, high-speed arithmetic for digital computers. It is based on a *bounded carry inspection adder* (BCIA) that operates on ternary coded data words. The recoding circuitry is of the type currently in use in computers that perform high-speed multiplication by the modified short cut (MSC) technique of shifting over ones and zeros.[1, 2] The uniqueness of the BCIA lies in the application of this recording to addition, and to an even greater speed-up of the multiplication technique that fostered it. In the process of multiplication, repeated additions/subtractions are required. The BCIA speeds up the process by providing an addition technique that yields the sum *in parallel in one step* through the elimination (bounding) of carry propagation.

Previous attempts at multiplication speed-up required separate adders for addition and multiplication arithmetic,[2, 3] or employed adders that were not truly parallel in the one-step sense,[4, 5] or retained difficulties in sign and overflow detection.[6, 7] In the method described, the same adder is used for addition and multiplication arithmetic. Further, the nature of the recoding technique provides for one-step summation: since there are never adjacent digits in the recoded summands, there can never be any carry propagation. Any question of sign or overflow is resolved by providing an N + 2 bit register for summation, where N is the conventional signed word length.

## RECODED-BINARY ARITHMETIC

The increased prominence of scientific computation has focussed attention on the need for increased arithmetic speed since, in general, scientific problems require a higher ratio of arithmetic-to-housekeeping operations. The search for increased speed has centered on faster circuitry and faster algorithms. The latter approach has led to recoded-binary arithmetic.

### *High Speed Multiplication*

The method most currently used for binary multiplication is based on the decimal short-cut multiplication method used on desk calculators. The technique centers on the ability to replace a string of ones between positions $a$ and $m$ by $\sum_{a}^{m} 2^i = 2^{m+1} - 2^a$. For example, the binary number 00111110 = $+2^5 + 2^4 + 2^3 + 2^2 + 2^1$ can be recoded as: $+ 2^6 - 2^1$. Thus, in multiplying, five additions and four shifts

can be replaced by one subtraction, a single shift of five, and one addition. Lehman[1] extended this basic recoding technique and demonstrated that isolated ones and zeros could be treated as part of a longer string of digits. An isolated one calls for an addition of the multiplicand, while an isolated zero calls for a subtraction. For example, 00110110, which would ordinarily be treated as two sequences, can be recoded as $+2^6 -2^3 -2^1$. The justification for this treatment of isolated units stems from the two self-obvious identities $+2^N -2^{N-1} = +2^{N-1}$ and $-2^N +2^{N-1} = -2^{N-1}$. The result is that there are never two sequential addition/subtractions.

The preceding techniques may be given a more formal representation by considering a ternary coding of binary numbers such that:

$$2^{-n} \sum_{o}^{m} B_t 2^t \equiv 2^{-n} \sum_{o}^{m+1} (-1)^{S_t} C_t 2^t$$

where $B_t$, $S_t$, and $C_t$ are binary variables and $t$ is the positional indicator ($t = 0, \ldots, m$). An operation is performed in any $t$th position in which $C_t = 1$, with the sign of the operation controlled by $S_t$. In the above, $S_t = 0$ commands an addition. The entire recoding logic consists of an implementation such that:

$$C_t = (B_t \neq B_{t-1})\ (C_{t-1}) \qquad (1)$$

$$S_t = (C_t B_{t-1}) + (C_t S_{t-1}) \qquad (2)$$

Note, from Eq. (1), that $C_t C_{t-1} = 1$, and thus operations can *never* be required in two successive cycles.

*Ternary Coded Addition for BCIA*

The Bounded Carry Inspection Adder operates by using the *code commands directly* as the representation of the summands, the only additional requirement being that both data words are recoded. Addition takes place in parallel in exact one clock pulse, hence the use of the "I" in BCIA to indicate addition by mere inspection of the summands. The rules for addition are stated in the following equations, using the notation above and with the subscripts $a$ and $b$ representing terms in the first and second summands, respectively:

$$(S_t C_t)_a (S_t C_t)_b = S_{t+1} C_{t+1} C_t \qquad (3)$$

$$(S_t C_t)_a (S_t C_t)_b = S_t C_t \qquad (4)$$

$$(S_t C_t)_a (dC_t)_b = S_t C_t \qquad (5)$$

$$(S_t C_t)_a (S_t C_t)_b = S_{t+1} C_{t+1} C_t \qquad (6)$$

$$(S_t C_t)_a (dC_t)_b = S_t C_t \qquad (7)$$

In the preceding equations, the $d$'s represent "don't cares" and the $a$ and $b$ subscripts may be interchanged since the commutative laws must obviously apply.

Since the mathematical representation may tend to obscure rather than explain, a simple example is shown in Fig. 1.

| | | |
|---|---|---|
| Exponent of $2^t$ | $t =$ 7 6 5 4 3 2 1 0 | |
| | + − + | |
| Augend | $A =$ 0 0 1 1 1 0 1 0 = | 58 |
| | + − − | |
| Addend | $B =$ 0 0 1 1 0 1 1 0 = | 54 |
| | | 112 |
| Augend in BCIA notation | $A =$ 0 1 0 0 0 0 1 0 | |
| Addend in BCIA notation | $B =$ 0 1 0 0 0 0 0 0 | |
| | 1 0 0 0 0 0 0 0 = | $2^7 - 2^4$ |

Note: 0 is a non-value space zero. (See Appendix A.)

Figure 1. Addition example.

In the figure the plus-minus notations above the conventional-binary summands indicate that an operation is to be performed ($C_t = 1$), and give the sign of that operation, ($+ \equiv S_t = 1$). The example illustrates the modified Boolean operations to be performed. The complete rules for addition (positive numbers) are summarized in Table 1.

| *Summands* | | | *Position Entry* | |
|---|---|---|---|---|
| A | B | | t + 1 | t |
| 1 | 1 | = | 1 | 0 |
| 1 | 0 | = | 0 | 0 |
| 0 | 0 | = | 0 | 0 |
| 0 | 0 | = | 0 | 0 |
| 1 | 0 | = | 0 | 1 |

Table 1. Addition Table for the BCIA.

The rules for addition with both of the summands negative are identical. It is necessary only to keep track of the sign, as in conventional sign and magnitude addition. Addition of oppositely signed numbers is readily implemented. The numbers would be recoded normally, independent of sign. Then the complementary output of the register containing the recoded-negative summand would be used for the addition. The sign of the sum would be obtained from the sign decoder, as before.

Overflow detection is readily obtained in the BCIA. Conventional notation for an N-bit computer usually implies an N – 1 number and a sign bit in the Nth position. For addition, an N + 1 sum register

is used to provide for overflow. In BCIA addition, an N + 2 addition register is employed. The sign is in position N + 2, while position N is used for a "non-overflow overflow," and position N + 1 used for actual overflow. The "non-overflow" must be provided for since the recoding of an N – 1 bit word can produce a 1 in the Nth position. This would normally be reabsorbed into the standard word length when the sum is formed. If both summands have a 1 in this position, a conventional overflow is formed in position N + 1, and is treated in a normal manner.

Subtraction is performed in the manner previously indicated for opposite-sign addition. The complementary output of the recoded subtrahend register is always used in the process. The choice of the true or complementary output of the recoded minuend depends on its sign, with the complement chosen for negative sign. Addition then proceeds as before. Note that there is no need to sense the overflow bit, as in conventional subtraction, to determine whether subtraction has proceeded in the right direction (i.e. has a larger number been subtracted from a smaller number, necessitating a correction cycle?). This stems from the fact that the representation of the most significant bit in the answer *is* the sign of the answer, while the *magnitude* of the answer is already correct. In this sense, as well as in the fact that borrow propagation is elimited, the BCIA produces faster subtraction than previous techniques.

### *Fast Multiplication*

McSorley,[2] in his description of Stretch arithmetic, pointed to the speed advantage in using special purpose carry-save adders in the multiplication process, while using carry-look-ahead for ordinary high-speed addition. With BCIA arithmetic, the same adder is used for both multiplication and addition. Further, multiplication speed is considerably increased.

In the carry-save technique, all intermediate additions can be effectively performed without carry propagation, while the final addition required conventional carry ripple. With BCIA representation, one-step addition/subtraction is used for all intermediate operations, and for the final one as well. In multiplication, both multiplier and multiplicand are used in recoded form. It should be noted that the full multiplication *operation code* is contained in the format of the multiplier word. Thus: Multiplier $54 = 1\ 0\ 0\ 0\ 0\ 0\ 0$ (BCIA ternary coding) calls for a shift, subtraction of the multiplicand, another shift and subtraction, then a double shift and an addition of the multiplicand. It seems apparent, therefore, that a further savings in overall hardware is obtained with BCIA arithmetic, since the multiplication decoder can be considerably simplified. It is also seen that a further speed-up can be obtained, since multiplication sequence decoding, which normally proceeds serially by groups, is already "stored" in the multiplier word. And finally, it should be noted that the elimination of carry due to the BCIA permits formation of the most significant bits first, for both serial and parallel machines. This is obtained by merely using left shifts instead of right shifts and initiating the multiplication from the left. The first significant answer bit becomes available in one step, and the *i*th bit becomes available in the *i*th step. It thus becomes possible to implement a rounded-multiply operation which terminates after the minimum commanded number of digits has been formed. Such an operation would find use in variable byte arithmetic as well as in significant-digit analysis of floating point calculations.[8]

### *Division*

An analysis of a ternary division algorithm using BCIA notation is lengthy and will not be treated at this time. It may be briefly stated, however, that the method employs a shifting over space zeros. That is similar to the method of shifting over ones and zeros presently employed for high-speed division.[2,12,13] The justification is similar since, in both cases, the digits shifted over are merely position indicators and not value digits. Preliminary analysis of the shift average, a figure of merit proposed by Robertson, provides a figure greater than three. This is obtained without the need for generating and storing multiples of the divisor, typically ¾, 1, and 3/2.[14] Penalty is paid, however, in going to a two-step addition process, since intermediate partial remainders will now sometimes have adjacent digits. This will result in a transfer digit,[5] which is absorbed in the second step.

## IMPLEMENTATION

The full description of implementation of the concepts suggested by this paper can only be con-

sidered by a lengthy treatise on the design of a digital computer in which the use of BCIA arithmetic is extended to the entire machine. If Robertson's definition of redundancy[9] is extended to include the BCIA form of ternary coding, then such a computer is fully possible. There is a hierarchy of approaches which could be implemented. Briefly stated, these are:

1. *Total.* In this approach, at a cost of program assembly time, but with a saving in output decoding, the entire machine, including instructions, data and addresses, is structured for BCIA ternary representation.
2. *Arithmetic only.* Here, only data words are converted, as inputted. Address modification would require local recoding.
3. *Arithmetic, as needed.* In this approach, all words are in conventional binary form and are converted as required.

Certainly the Total approach would be the most desirable. The problem of a ternary representation can be solved on a multiregister basis or can be implemented with three-state devices, such as the tunnel diode, or appropriately wound magnetic cores. Thus, for example, the ability to implement 0, the *non-value* space zero, presently exists. The multiregister technique represents a brute-force approach that would provide two N-bit registers to distinguish among the three states. The way in which the registers are filled and the use in BCIA addition are shown in Appendix A. These are but two approaches: other, more elegant, schemes can be developed.

The recoding of a binary word into ternary form has been treated elsewhere.[2,12] For completeness, Appendix B shows a possible logical implementation for a four-bit word. Extension to longer word lengths is treated in the references. Essentially, these reduce down to logic diagram descriptions of the equations presented earlier in this paper.

## CONCLUSION

An extension of ternary coding to permit the elimination of carry propagation in addition and subtraction has been presented, and a new parallel Bounded Carry Inspection Adder has been proposed. The speed-up of fast multiplication by virtue of adder speed-up and the ability to use the same adder for both multiplication and addition has also been indicated. The technique described provides for simple over-flow detection and for round-off techniques that suggest significant-digit-arithmetic capability.

While the discussion has centered on a ternary representation, and electronic means for such a representation have been mentioned, it is noted that normal binary hardware could be used for an equivalent representation. The possibility of redundant representation in an entire computer has been opened through the use of redundancy for both summands in the adder, thus extending such representation to the entire arithmetic unit. It is felt this can provide the means for faster parallel execution in a digital computer.

## APPENDIX A

In handling BCIA operations, a means of representation of the non-value space zero, 0, must be provided. As is seen in Appendix B, the determination of the space-zeros and their location in the recoded word is determined from the type of recoding used for multipliers. The ones of the BCIA word represent the *addition* commands of the multiplier, the zeros represent the subtractions, and the *shift* commands represent space-zeros. The *length* of shift represents the number of sequentially registered 0's. In this manner, the decimal number 54 is recoded as: "Shift one (0), subtract (0), shift one (0), subtract (0), shift two (00), and add (1). If an eight-bit word is considered, an additional shift (0) must be entered. Written in BCIA form, this becomes

$$54 = 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0$$

To represent this using the brute force approach of two N-bit registers, the procedure would be as follows:

1. Load register A with ones in the positions where an add command had occurred. The remainder of the register is filled with zeros.
2. Load register 0 with zeros in the position where a subtract command had occurred. The remainder of the register is filled with ones.

For the decimal number 54, the register would appear as below:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| A Register | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 Register | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

The columns in which the same digit notation occurs represent the existence of that digit in the

BCIA number. The columns containing differing entries in the two registers represent the space-zeros, 0. This can be seen when it is remembered that a 1 represents an *add* command and a 0 represents a *subtract command.* A space-zero then is a simultaneous command to add and subtract, i.e. a no-op statement, in effect.

Addition of two numbers in this format is given in the example below for 54 + 57 = 111

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 54 | $A_a$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 57 | $A_b$ | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| 54 | $0_a$ | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 57 | $0_b$ | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| SUM | $A_s$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| | $0_s$ | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| | | ↓ | | | ↓ | | | ↓ | ↓ |
| | | $+2^7$ | | | $-2^4$ | | | $-2^1$ | $+2^0$ |

The entries in the sum registers are seen to be determined as follows:

Terms in Summands

| | | | |
|---|---|---|---|
| $A_s$ | 1 | 0 | |
| $0_s$ | 1 | 1 | $= A_a A_b 0_a 0_b$ |
| $A_s$ | 0 | 0 | |
| $0_s$ | 0 | 1 | $= A_a A_b 0_a 0_b$ |
| $A_s$ | $0\,L_{t+1}$ | 0 | |
| $0_s$ | $1\,L_{t+1}$ | 0 | $= (0_a 0_b + 0_a 0_b)\ A_a A_b \equiv K_t$ |
| $A_s$ | $0\,K_{t+1}$ | 1 | |
| $0_s$ | $1\,K_{t+1}$ | 1 | $= (A_a A_b + A_a A_b)\ 0_a 0_b \equiv L_t$ |
| | $t+1$ | $t$ | |

A magnetic core logic implementation using "leaster" and "moster" symmetric circuits can be employed to provide the sum.

The example given serves to demonstrate an additional requirement of a system employing BCIA arithmetic: while the addition of two BCIA coded words will always take place without carry (since there can be no adjacencies), the sum may not always be generated in the proper recoded form. This can, on occasion, hold for the results of other arithmetic operations as well. An approach in resolving this is to recode before the next operation, using look-ahead recoding equations of the form of Eqs. (1) and (2). Since the equations contain terms subscripted with $t-1$, they are expandable to $t-n$, thus suggesting a one-step recoding that can be accomplished during the *fetch* portion of the next instruction.

## APPENDIX B

Fast Multiply. Shifting Over Ones and Zeros, Modified.

Determination of Length of Shift.

A maximum shift length of 4 is analyzed.
$H$ = Previous history, 1 = Add, 0 = Subtract.
$R$ = 1 = Perform an operation, then shift. $R$ = 0 = No ops.
$yzAB$ = 4 least bits of multiplier presently being examined.
Dash (–) represents a "don't care."

| $y$ | $z$ | $A$ | $B$ | $H$ | $R$ | No. of Shifts | Remarks |
|---|---|---|---|---|---|---|---|
| – | – | 1 | 0 | 1 | 1 | 1 | Shift 1 when |
| – | – | 0 | 1 | 0 | 1 | 1 | $R = 1$ *and* $A \neq B$ |
| – | 1 | 0 | 1 | 1 | 0 | | |
| – | 0 | 1 | 0 | 0 | 0 | | Shift 2 when |
| – | 1 | 0 | 0 | 0 | 0 | 2 | $z \neq A$ *and* |
| – | 1 | 0 | 0 | 1 | 1 | | shift of 1 is |
| – | 0 | 1 | 1 | 1 | 0 | | *not* called for |
| – | 0 | 1 | 1 | 0 | 0 | | |
| 1 | 0 | 0 | 1 | 1 | 0 | | Shift 3 when $z$ |
| 0 | 1 | 1 | 0 | 0 | 0 | | $\neq A$ *and* $y \neq$ |
| 1 | 0 | 0 | 0 | 0 | 0 | 3 | $z$ *and* a shift |
| 1 | 0 | 0 | 0 | 1 | 1 | | of 1 is not |
| 0 | 1 | 1 | 1 | 1 | 0 | | called for |
| 0 | 1 | 1 | 1 | 0 | 1 | | |
| 0 | 0 | 0 | 1 | 1 | 0 | | Shift 4 when |
| 1 | 1 | 1 | 1 | 0 | 0 | | $Z = y = z$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 4 | *and* a shift of |
| 0 | 0 | 0 | 0 | 1 | 1 | | 1 is not called |
| 1 | 1 | 1 | 1 | 1 | 0 | | for |

From the remarks, the shift equations follow: ($Sh$ = Shift)

$$Sh\,1 = R\,(AB + AB)$$
$$Sh\,2 = (Az + Az)\,Sh\,1$$
$$Sh\,3 = (yzA + yzA)\,Sh\,1$$
$$Sh\,4 = (yzA + yzA)\,Sh\,1$$

From the section below, it is seen that the term for $R$ is generated in the process of determining the operation (add/subtract) to be performed. In the Karnaugh maps and logic equations that follow, $H$ is a history flip-flop that indicates whether the last operation was addition (1) or subtraction (0). $B$ is the L. S. B. of a pair, $A$ is the next digit, $S$ means add, $D$ means subtract, and $R$ means shift right (with respect to $B$ ).

| A | B | H | S | D | R |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 |

S
ADD MAP

| H \ A | 0 | 0 | B |
|---|---|---|---|
| 1 | 1 | 0 | |
| 0 | 0 | 0 | |
| 1 | 1 | 0 | |

D
SUBTRACT MAP

| H \ A | 0 | 0 | B |
|---|---|---|---|
| 0 | 0 | 1 | |
| 1 | 0 | 0 | |
| 0 | 0 | 1 | |

$S = ABH + ABH$
$D = A[BH + BH]$
$R = (S + D) = SD$
$(BH + BH) = (B + H)\ (B + A)$

## REFERENCES


1. M. Lehman, "High Speed Digital Multiplication," *IRE Transactions on Electronic Computers,* vol. EC-6, pp. 204-205 (1957).

2. O. L. McSorley, "High Speed Arithmetic in Binary Computers," *Proceedings of the IRE,* vol. 49, pp. 67-91 (1961).

3. B. Esterin, B. Gilchrist, and J. H. Pomerene, "A Note on High Speed Digital Multiplication," *IRE Transactions on Electronic Computers,* vol. EC-5, p. 140 (1956).

4. A. Avizienis, "Signed Digit Representations for Fast Parallel Arithmetic," *IRE Transactions on Electronic Computers,* vol. EC-10, pp. 389-399 (1961).

5. A. Avizienis, "Binary-Compatible Signed-Digit Arithmetic," *AFIPS Proceedings,* vol. 26, pp. 663-672 (1964).

6. H. L. Garner, "The Residue Number System," *IRE Transactions on Electronic Computers,* vol. EC-8, pp. 140-147 (1959).

7. R. D. Merrill, Jr., "Improving Digital Computer Performance Using Residue Number Theory," *IEEE Transactions on Electronic Computers,* vol. EC-13, pp. 93-101 (1964).

8. N. Metropolis and R. L. Ashenhurst, "Significant Digit Computer Arithmetic," *IRE Transactions on Electronic Computers,* vol. EC-7, pp. 265-267 (1958).

9. J. E. Robertson, "Introduction to Digital Computer Arithmetic," presented at University of Michigan Engineering Summer Conference on Introduction to Digital Computer Engineering (June 1964).

10. F. Salter, "A Ternary Memory Element Using a Tunnel Diode," *IEEE Transactions on Electronic Computers,* vol. EC-13, pp. 155-156 (1964).

11. J. Santos, H. Arango, and M. Pascual, "A Ternary Storage Element Using a Conventional Ferrite Core," *IEEE Transactions on Electronic Computers,* vol. EC-14, no. 2 (1965).

12. Ivan Flores, *The Logic of Computer Arithmetic,* Prentice-Hall Inc., Englewood Cliffs, N. J., 1963.

13. R. S. Ledley, and J. B. Wilson, "An Algorithm for Rapid Binary Division" *IRE Transactions on Electronic Computers,* vol. EC-10, pp. 662-670 (1961).

14. C. V. Freiman, "Statistical Analysis of Certain Binary Division Algorithms," *Proc. IRE,* vol. 49, pp. 91-103 (1961).

# A FAST CONDITIONAL SUM ADDER USING CARRY BYPASS LOGIC

Joseph F. Kruy
*Honeywell — EDP Division*
*Waltham, Massachusetts*

## INTRODUCTION

The higher speeds obtainable with present day logic circuits of various integrated circuit types increase the need for faster adders. The speed of addition can be increased primarily in two ways: (1) by more efficient logic organization, (2) by using faster logical elements.

Among the different binary full adders the best known perhaps is the iterated type or ripple carry adder.[1,2] In this adder, the carry is propagated between adjacent ordered stages through relatively fast carry circuits, and the sum is generated after the carry propagation has been completed. It is recognized that in this type of adder the major portion of the required addition time is due to carry propagation time. There are several ways of speeding up carry propagation. Two of the most frequently used methods are the lookahead and the carry skip techniques.[3,4,5] In each of these techniques the sum generation is subsequent to the speeded up carry propagation.

In the conditional sum adder, the generation of the sum is simultaneous with that of the carries. The conditional sum formation was first described by Sklansky in 1960[6] and later used by Bedrij in a parallel adder utilizing a parallel pyramidal type of logical organization.[7] In the conditional sum adder described here the carry signal is propagated serially in two logically different paths. One of these paths is for the generation of the two possible carries of each binary stage and the other is to perform the carry bypass and selection logic. Because of the serial propagation of signals, fast circuitry is required.

The fastest available digital circuits today still use tunnel diodes and are fabricated by hybrid integrated circuit methods. Speeds of about an order of magnitude higher can be obtained with circuits incorporating tunnel diodes than with those using transistors only. The controversy about the future of the tunnel diode in digital computers stems from the fact that they do not lend themselves to the most promising type cf integrated circuit technology, i.e., fabrication using monolithic techniques. In hybrid integrated circuits, however, the speed advantage of the tunnel diode can be efficiently exploited due to the decreased stray reactances of packaging and construction. Therefore, in this writer's opinion the future use of the tunnel diodes in digital computers will depend to a large extent on the acceptance of some sort of hybrid construction technology.

In the design of the adder, an approach useful for functional units appears to be the most suitable one and is the approach followed here.[8] This design phil-

osophy makes use of the fact that optimization for the maximum performance per cost ratio is an easier task if the logic circuits have to perform only a predetermined, limited number of logic functions. In this case a better coordination between logic and circuit design is also possible.

The logic organization presented here observes the capabilities and limitations of practical circuitry and topology. The circuit design conforms to the characteristics of the logic functions and takes into account the potential advantages of the microminiaturization techniques.

## LOGICAL ORGANIZATION OF ADDER

The principle of the conditional sum adder is based on the computation of conditional sums and carries that result from all possible distribution of carries.[6] Since for a binary full adder stage the carry can be either a one or a zero, only two conditional carries, and consequently two conditional sums, need be generated.

The illustration of the principle is shown in Fig. 1. Let us assume that two operands of $N$ bits each are to be added. Both operands, and therefore the

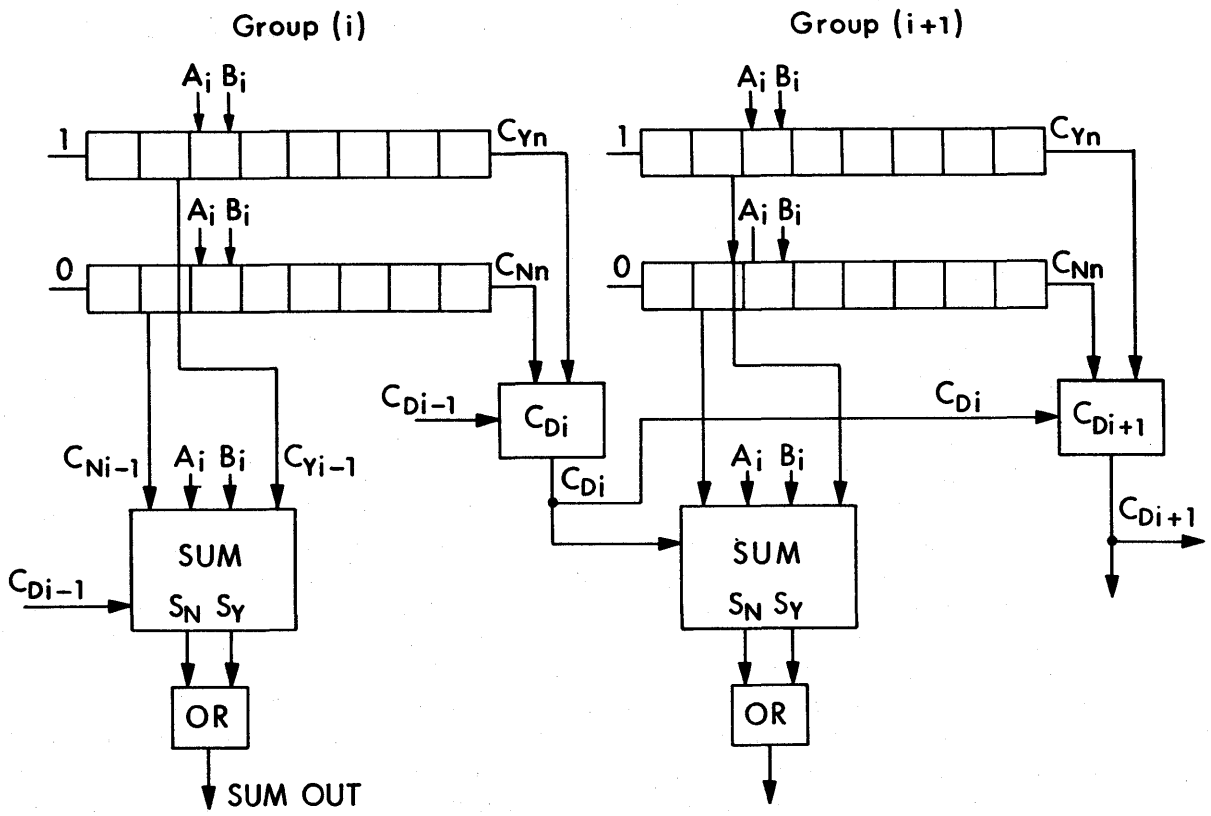



Figure 1. Principle of the conditional sum-carry bypass adder.

adder, are divided into $k$ groups of $n$ bits each ($n$ does not necessarily have to be the same for every group). In each group the carry and a portion of the sum circuits are duplicated in order to generate the two carries and two sums corresponding to a possible one and zero carry input to the group. Having two sums available, a decision is then made as to which sum is the correct one and this in turn becomes the sum output.

The carry-bypass decision logic circuit determines the correct carries using the carry signals of the last stage of the group and the output of the previous bypass circuit. The decision logic is then propagated to higher order groups. The total worst-case carry propagation time for two operands of $N$ bits is

$$t_p = n_1 t_c + ktg \qquad (1)$$

where $t_p$ is the total carry propagation time, $t_c$ is the propagation time of the carry circuit within the group, $n_1$ is the number of carry stages in the first group, $k$ is

the number of groups, and $t_g$ is the intergroup propagation time. This relationship assumes that the output signal of the last stage of a group is available by the time the decision signal reaches the corresponding decision stage.

The derivation of the decision logic either by the method of Karnaugh map or total induction is straightforward, thus only the end results are given here. The bypass decision function ($C_{Di}$) of the $i$th group of an $n$ bit grouping is as follows:

$$C_{Di} = C_{D(i-1)}C_{Yin} + C_{Nin} \tag{2}$$

where $C_y$ and $C_n$ are the carry output as shown in Fig. 1.

Unfortunately, the logical expression of Eq. (2) cannot be readily realized with one tunnel diode unit-delay. Also the AND circuit would require tighter component tolerances than required by an OR Circuit, thus contradicting the design goals. For these reasons, a different logic function whose generation is practical with a tunnel diode transistor circuit using the tunnel diode in the propagation path was needed. This circuit should have operation margins at least as good as a "one out of three" analog threshold circuit. By employing logic which is not of the minimal form, a convenient function was found, which is given in Eq. (3).

$$C_{Di} = C_{Yin}\,(C_{D(i-1)} + C_{Nin}) \tag{3}$$

This logic statement can be realized by an emitter coupled transistor-tunnel diode bypass circuit having the properties described above. More will be said about this circuit in the following section.

The sum is produced by two cascaded logical equivalence functions ($A\nabla B = AB + \overline{A}\overline{B}$) for both $S_y$ and $S_n$. The two sums are given in Eq. (4).

$$\begin{aligned} S_y &= (A \nabla B) \nabla C_Y \\ S_n &= (A \nabla B) \nabla C_N \end{aligned} \tag{4}$$

From this we can see that the first subsum can be combined for $S_y$ and $S_n$ and the circuit producing A $\nabla$ B does not have to be duplicated. Therefore, the expression for the sum of group $i$ stage $j$, including selection, is given in Eq. (5).

$$\begin{aligned} S_{ij} = (A_{ij}\nabla B_{ij})\nabla[&C_{Yi(j-1)} \cdot C_{D(i-1)} \\ &+ C_{Ni(j-1)} \cdot C_{D(i-1)}] \end{aligned} \tag{5}$$

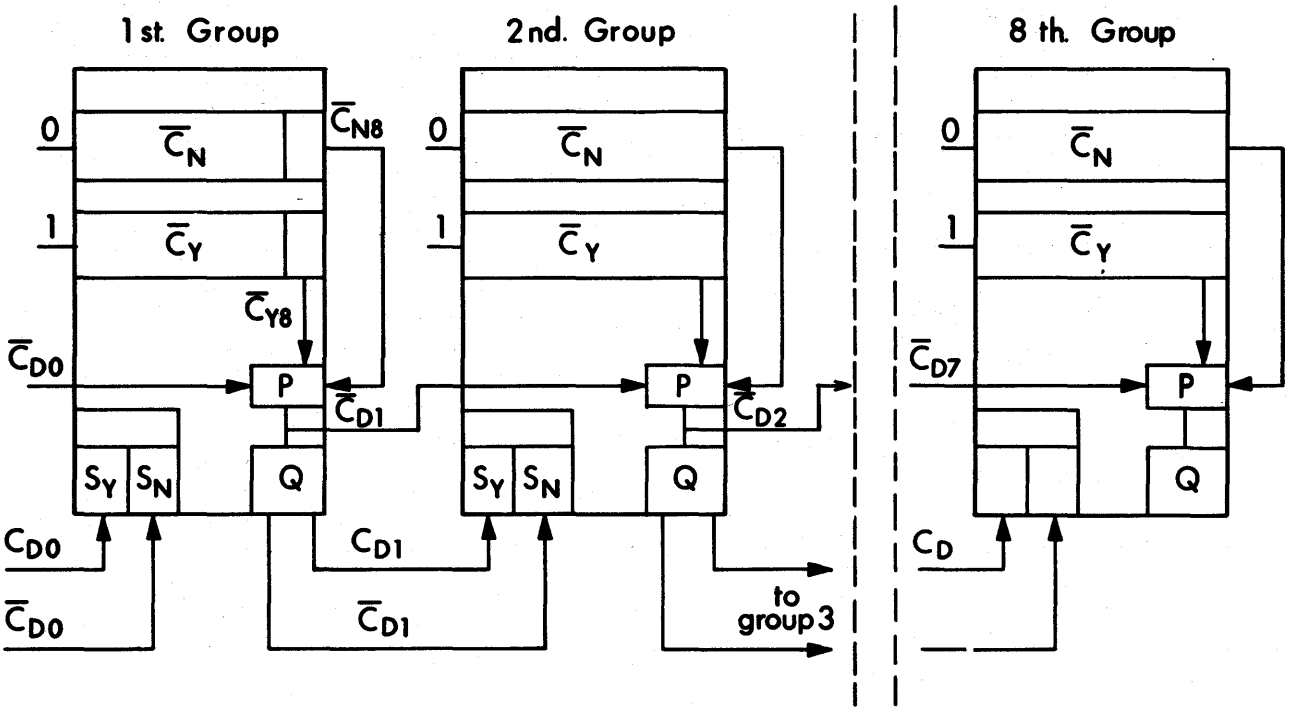



Figure 2. Block diagram of adder, eight groups — eight bits each.

In order to eliminate the relatively slow logical equivalence circuits from the sum selection path the following logically identical expression was implemented in the actual adder.

$$S_{ij} = [(A_{ij} \bigtriangledown B_{ij}) \bigtriangledown C_{\gamma i(j-1)}] \cdot C_{D(i-1)} + [(A_{ij} \bigtriangledown B_{ij}) \bigtriangledown C_{Ni(j-1)}] \cdot C_{D(i-1)} \quad (6)$$

The complete logical organization of the adder is given by the block diagram of Fig. 2. For reasons of speed and circuit simplicity, the input signals to the individual adder stages are the complements of the operand bits. Consequently, the complement of the carry rather than the carry is propagated. This way only the complements of the input signals are needed as inputs; greatly reducing packaging and interconnection problems. Each square on the diagram represents $n$ adder stages in accordance with logical grouping and packaging. The $Cn$ and $Cy$ functions are generated by the carry circuits according to Eq. (8). Box $P$ designates the decision logic circuit ($C_{Di}$) and $Q$ stands for the amplifier of the decision function for the selection of the proper sums. These are either $S_{\gamma ij}$'s or $S_{Nij}$'s of Eq. (4). The iterative logical organization is clearly seen from the block diagram.

If we assume that each group contains the same number of stages (this is a practical assumption for reasons of modularity and cost considerations, but does not give the highest obtainable speed), and that $tg = \alpha \cdot tc$, then Eq. (1) becomes

$$t_p = ntc + \frac{\alpha N}{n} \cdot tc \quad (7)$$

where $N$ is the number of operand bits.

If this expression is minimized as a function of $n$, we obtain for the optimal number of stages within a group:

$n = \sqrt{\alpha N}$

By taking $\alpha = 1$ (i.e., the intergroup stage delay is equal to the intragroup stage delay) we obtain: $n = \sqrt{N}$. For $N = 64$ this yields $n = 8$ and $k = N/m = 8$.

The timing diagram for a 64-bit adder is given in Fig. 3. The times shown represent worst-case design values with packaging and interconnection parameters taken into consideration.

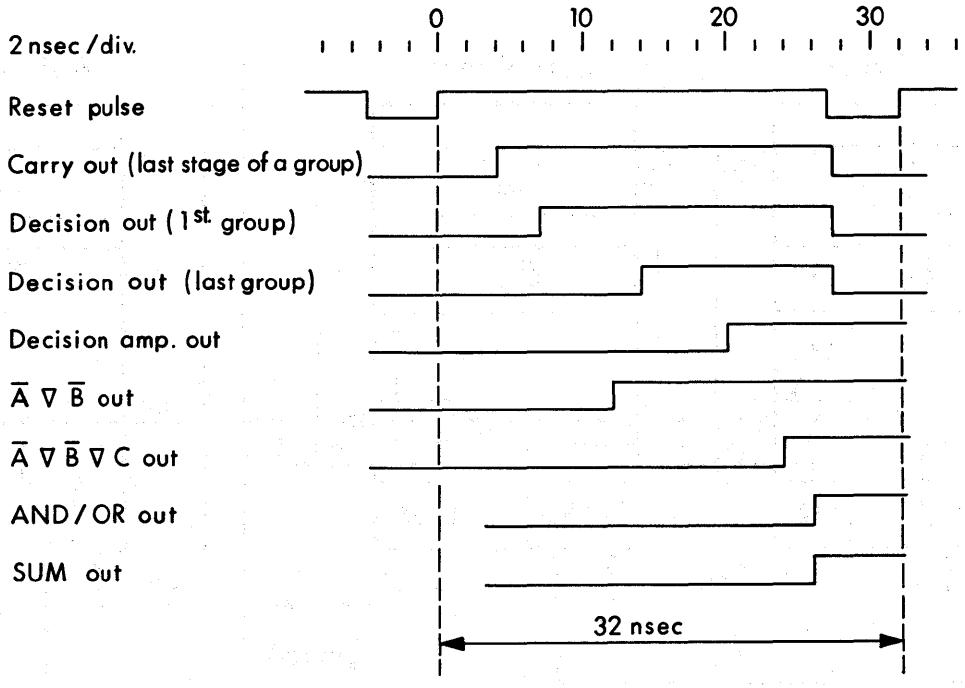



Figure 3. Timing diagram of 64-bit adder. A complete cycle is shown.

## CIRCUITS

The circuitry used in the conditional sum adder is designed to satisfy the constraint that the logic be built using not more than three different modular microcircuits. These circuits must be flexible enough to meet different requirements—either logical or gain—for the various applications. Each of these microcircuits is built on a separate ceramic substrate using hybrid microminiaturization techniques. Two of these circuits do not lend themselves to any other technique (unless the tunnel diodes are attached separately to monolithic circuits). The third circuit could be built with monolithic construction, however, at the time of the design the required speeds were not obtainable using the commercially available monolithic circuits. A snap-off diode pulse generator circuit was used for the generation of the clock pulses. This was built by miniature discrete components to be amenable to future modifications. However, in the construction of this circuit hybrid techniques could be used as well.

## THE CARRY CIRCUIT

In order to obtain the required speed and simplicity in the adder logic, the complement of the carry signal rather than the carry is generated by the carry tunnel diode (Fig. 4). However, for the sake of easy reference, the circuit will be referred to as the carry circuit.

The Boolean equation for the complement of the carry function of the adder of $j$th binary order is given below:

$$C_j = A_jB_j + A_jC_{j-1} + B_jC_{j-1} \qquad (8)$$

where $A_j$ and $B_j$ are the two operands of the $j$th stage; $C_{j-1}$ is the carry function of the stage of the next lower binary order; $C_j$ is the carry function of the $j$th stage. The barred symbols represent the complements of the logic variables defined above.

In the circuits of Fig. 4, the $A_j$ and $B_j$ signals are applied to terminals 7 and 8 respectively. $C_{j-1}$ is applied to terminal 5. $X$ and $Y$ terminals are for the power supply voltages ($X = +6\text{v} \pm 1\%$, $Y = +3\text{v}$ $\pm$ 5%). The tunnel diode, *TD*, performs "2 out of 2" analog threshold logic, i.e., the output of the tunnel diode is a one, if two or more of the inputs are ones; otherwise, it is zero. D-c bias is applied to the tunnel diode through the $R1$ resistor, and conditional bias through the series resistor network of $R2$ and $R3$.

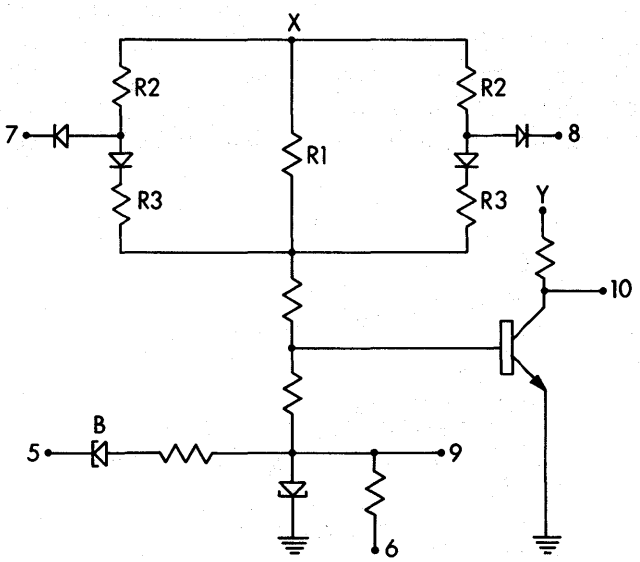



Figure 4. Circuit diagram of the carry circuit.

The tunnel diode is operated in the bistable mode using unconditional reset, with the reset pulse applied to terminal 6. Unilaterization between tunnel diodes is performed by a low peak current tunnel diode, $B$. The carry function $C_j$ appears at output terminal 10 and $C_j$ appears at output terminal 9.

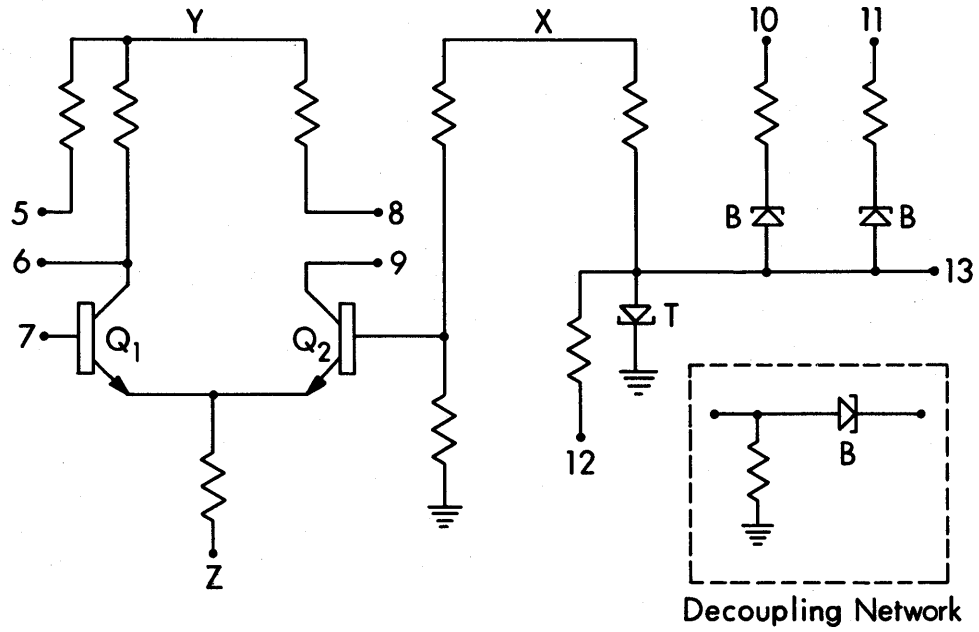



Figure 5. Decision-carry bypass circuit.

## THE BYPASS-DECISION CIRCUIT

The decision circuit, Fig. 5, is used for two functions: (1) to perform the carry bypass-decision logic and (2) to provide high-speed amplification. The circuit consists of two parts: (a) an emitter coupled transistor circuit and (b) a tunnel diode regenerative pulse amplifier or logical circuit.

When the circuit is used for the generation of the decision logic, the following logic function is performed: $C_{Dk} = C_Y (C_N + C_{Dk-1})$. In this application terminal 5 is connected to terminal 6. Terminal

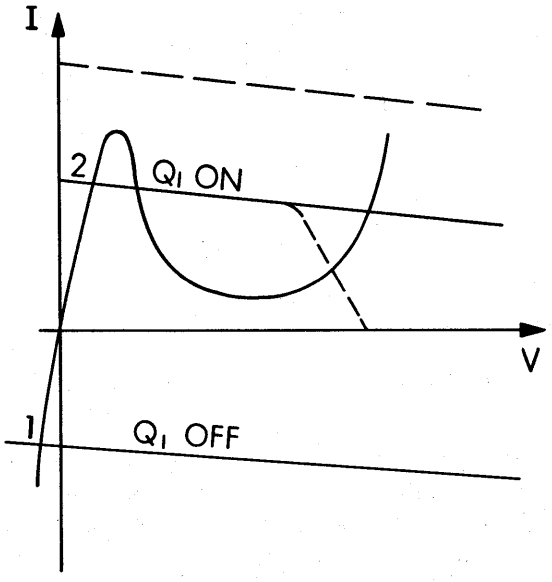



Figure 6. Characteristic curve and load lines of the tunnel diode.

9 is connected to 8, and to 13 through a tunnel diode-resistor decoupling network shown in Fig. 5. The power supply voltages are $X = +6v \pm 1\%$; $Y = +3v \pm 5\%$; $Z = -6v \pm 5\%$. $C_y$ is applied to terminal 7; $C_N$ and $C_{Dk-1}$ are applied to terminal 10 and 11 respectively. The output is obtained at terminal 13. The tunnel diode characteristic with its load line is shown in Fig. 6. When $C_Y$ is low (40 millivolts), the transistor on the righthand side conducts, and the tunnel diode is biased to point 1. There is not sufficient current to switch the tunnel diode even when $C_N$ and $C_{Dk-1}$ are both present. However, when $C_Y$ is high (500 millivolts), the transistor on the righthand side is cut off and the tunnel diode is biased to point 2. Now, either of the two signals, $C_N$ or $C_{Dk-1}$, is capable of setting the tunnel diode to the high voltage state. The above logic function is thus generated. The tunnel diode is reset by an unconditional reset pulse through terminal 12.

In the second application, the tunnel diode is used as a buffer amplifier and the transistor circuit as a complementary current switching amplifier. Then terminal 8 is connected to terminal 9 and terminal 7 to 13. The input is applied to terminal 13 and the complementary outputs are obtained at terminals 6 and 8.

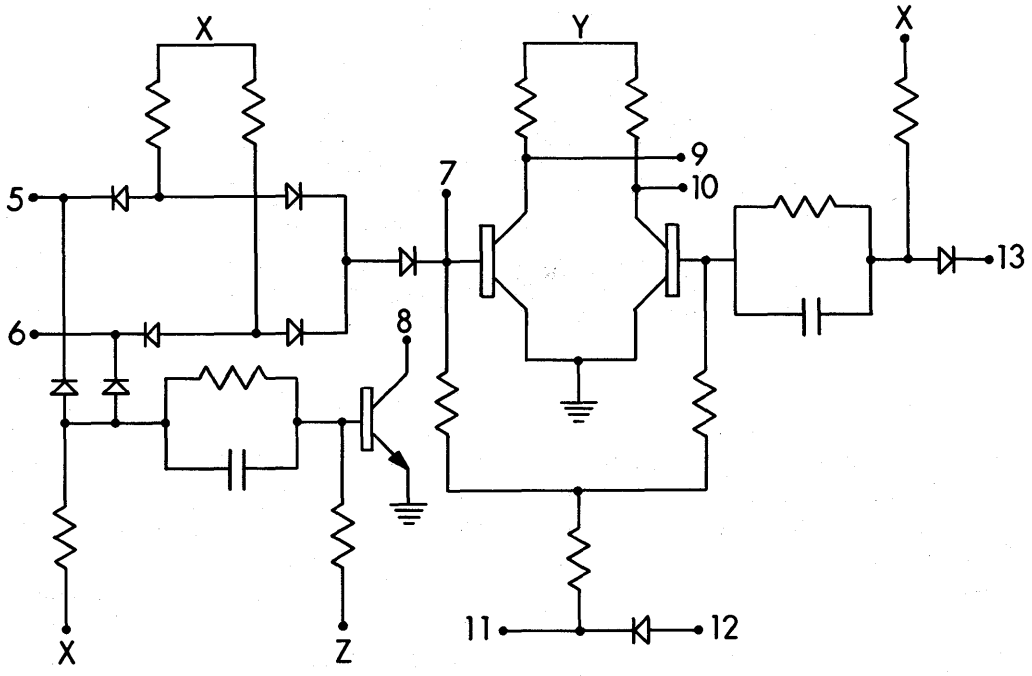



Figure 7. Circuit diagram of gated logical equivalence circuit.

## SUM CIRCUIT

As described previously the sum function is generated by logical equivalence circuits. The sum substrate contains a NOR, a NAND, and an inverter circuit (Fig. 7). These circuits can be readily interconnected to provide the desired functions. In the first level of sum logic the inputs $A$ and $B$ are applied to terminals 5 and 6. Terminal 7 is connected to terminal 8. $X = +6v$; $Y = +3v$; and $Z = -6v$. The output $F = AB + \overline{A}\overline{B}$ is obtained at terminal 9. The operation of the circuit is straightforward, therefore only a short explanation is given here. When $A$ and $B$ are both low, neither $T1$ nor $T2$ conducts, therefore the output is high. When $A$ is low but $B$ is high, or vice versa, $T2$ is cut off but $T1$ conducts. Consequently, the output is low. When $A$ and $B$ are both high, $T2$ conducts making $T1$ cut off. As a result the output is high.

In the second level of sum logic, terminal 7 is connected to 8 and terminals 9, 10 and 12 are connected together. The input signals $AB$ and $C_D$ are applied to terminals 5, 6, and 13, respectively. $X$ and $Y$ are both connected to $+6v$ and $Z$ to $-6v$. The output func-

tion $F = (AB + AB)\ C_D$ appears at terminal 11. The operation of the circuit is the same as described for the first level circuit. In addition, the transistor amplifier circuit which is connected to terminal 9 manifests itself as an inverted AND output at terminals 9 and 10. The resistor diode network between terminals 12 and $Z$ is part of a logical OR network required to obtain the sum output on a single terminal.

## CONSTRUCTION AND PACKAGING

The circuit modules are fabricated on a 0.5 × 0.75-inch alumina substrate. The photograph of the assembled substrate without the cover is shown in Fig. 8. The resistors and conductive paths are printed by silk-screening process. The transistors and rectifier diodes are of chip form and are mounted onto

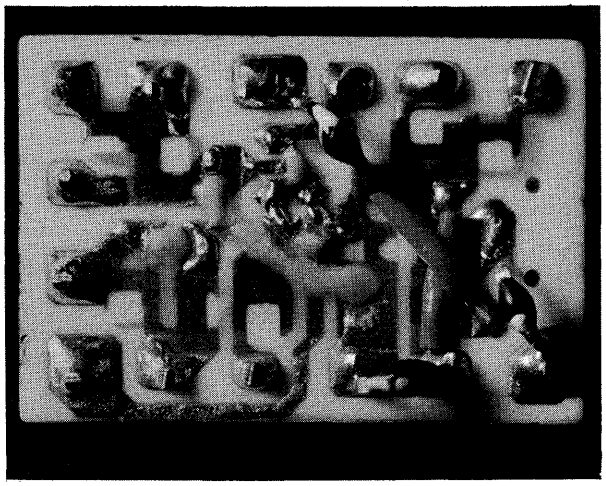


Figure 8. Photograph of the assembled substrate.

the substrate by high-compression bonding and solder flow techniques. The germanium tunnel diodes are encapsulated units in micro-epoxy packages. They are attached to the substrate by soldering. Fourteen pins of 20 mils in diameter are perpendicular to the substrate with center spacing of 125 mils. In the feasibility model, 4 adder stages are packaged on a 4 × 4-inch mother board. A mother board contains 22 hybrid integrated circuits and a reset pulse generator, which generates a negative pulse of 5 nanoseconds duration and of –4.5 volts amplitude. The board itself is a 5-layer board of 3/32-inch thickness using two layers of signal interconnections and a power supply plane between 2 ground planes as inner layers. The reset pulse is distributed via tapped, terminated transmission lines. The power distribution is accomplished by low-impedance transmission lines.

## CONCLUSION

A complete adder motherboard was assembled and tested. Average stage delays of 0.3 nanoseconds and voltage margins better than ± 8% were obtained using tunnel diodes of $I_p$ = 4.7 ma ± 3% and minimum $I_p/C$ ratio of 2.5 ma/pf. The high speed and good operating voltage margins are due to the isolation of the transistor capacitances from the tunnel diodes, the low stray capacitance construction and packaging techniques, and the large overdrive available for the tunnel diodes. From the experimental data obtained, the estimated add time of two 64-bit operands is 26 nanoseconds. This is approximately 20 percent faster than the maximal time of 32 nanoseconds. Due to its iterated logic organization, the adder is readily expansible. The add time of a 100-bit adder consisting of 10 modules of 10 stages is estimated to be approximately 30 nanoseconds.

With the exception of the tunnel diode, all other circuit elements can be fabricated by monolithic technology. However, since in both circuits the tunnel diodes are connected to input or output terminals, they can be attached separately to the monolithic circuits as part of the packaging procedure. With this approach, the speed of monolithic logic could be improved considerably where fast serial signal propagation is needed.

## ACKNOWLEDGMENTS

The author wishes to thank Dr. R. B. Lawrance and Dr. B. Rabinovici for their support and valuable suggestions and F. Duben for his assistance in the circuit design, packaging and evaluation.

## REFERENCES

1. R. K. Richards, *Arithmetic Operations in Digital Computers,* Van Nostrand, 1955.

2. W. G. Daly and J. F. Kruy, "A High-Speed Arithmetic Unit Using Tunnel Diodes," *IEEE Trans. on Electronic Computers,* vol. ED-12, pp. 503-511 (Oct. 1963).

3. A. Weinberger and J. L. Smith, "One Microsecond Adder Using One Megacycle Circuitry," *IRE Trans. on Electronic Computers,* vol. EC-5, pp. 65-73 (June 1956).

4. M. Lehman and N. Burla, "Skip Techniques for High-Speed Carry-Propagation in Binary Arithmetic Units," *IRE Trans. on Electronic Computers,* vol. EC-10, pp. 691-698 (Dec. 1961).

5. O. L. MacSorley, "High-Speed Arithmetic in Binary Computers," *Proceedings of IRE,* vol. 49 (Jan. 1961).

6. J. Sklansky, "Conditional Sum Addition Logic," *IRE Trans. on Electronic Computers,* vol. EC-9, pp. 226-231 (June 1960).

7. O. J. Bedrij, "Carry-Select Adder," *IRE Trans. on Electronic Computers,* vol. EC-11, pp. 340-346 (June 1962).

8. J. F. Kruy and C. V. Ramamoorthy, "Composite Synchronous-Asynchronous Logic Circuits for Nanosecond Computing," *Proc. National Electronics Conference,* October 1963, pp. 173-182.

# A CHECKING ARITHMETIC UNIT

Richard A. Davis
*Mesa Scientific Corporation*
*Subsidiary of Planning Research Corporation*
*Santa Ana, California*

## INTRODUCTION

In conjunction with studies of mechanizations for a 30-bit digital computer arithmetic unit, an investigation of techniques for checking arithmetic unit operations was conducted. The objectives were to provide maximum protection against undetected errors, while holding the cost of checking to a minimum. In addition, no loss of arithmetic unit speed should result. Finally, the checking process should give protection against memory-unit-induced errors.

In view of the constraints established, incorporation of checking through extension of existing arithmetic, control and memory unit hardware, rather than through extensive modifications, was decided upon. The feasibility of incorporating a Checking Arithmetic Unit (CAU) was selected as a major target for study.

In considering the problem, a number of basic questions are immediately apparent:

1. Are all errors of equal significance?
2. Are multiple-bit errors more significant than single-bit errors?
3. Is the computer used in an application where larger (magnitude) errors are obvious, due either to the nature of the problem or to the feasibility of programmed checks?
4. Is the computer used in an application where small errors are insignificant? Or conversely, do they tend to become accumulative?

A review of the requirements of the intended applications (primarily real-time data acquisition, telemetry decommutation, and test control) suggested the following set of ground rules to cover the design required:

(a) Single-bit errors are far more likely than double-bit errors with conventional logic elements.
(b) Additional protection against errors of large magnitude, even though highly improbable, is usually justified.
(c) Undetected errors in algebraic signs, or indications of singularities (overflow, underflow, division by zero, etc.) are catastrophic.
(d) Control element errors or failures are usually worse than others, and potentially less susceptible to programming checks.
(e) The detection equipment should be at least as reliable as the units being checked.

Provision of separable checking controls to allow independent trouble-shooting is a necessity.

## OPERAND RECODING VS. AUGMENTATION

A major consideration is the desirability of recoding operands. The utilization of checking codes such as the $An + B$ code proposed by Brown[1] requires provisions for encoding and decoding in the input/output units, as well as provisions for compatibility with other arithmetic and logical instructions. Such provisions are far from straightforward. Consider, for example, the problems of masking or extraction from an operand in $(37_n + 13842)$ code.

Further, such codes require extensive modifications to the basic arithmetic elements, rather than the (desired) addition of a relatively independent CAU. Interpretation of recoded displays or binary memory dumps would be extremely difficult, seriously hampering program and hardware debugging. Considering all these factors, it was decided to limit consideration to what might be termed "outboard" codes; i.e., error detecting codes which are appended as additional digits beyond the normal data word bits.

## ADDER CHECKING BY RESIDUES

Peterson has shown[2] that all adder checking systems utilizing an independent checking element are fundamentally the same as some residue-class check. In the limit, as the magnitude of the residue approaches the magnitude of the numbers being added, the checker becomes a duplicate adder. (See Fig. 1 and reference 3.)

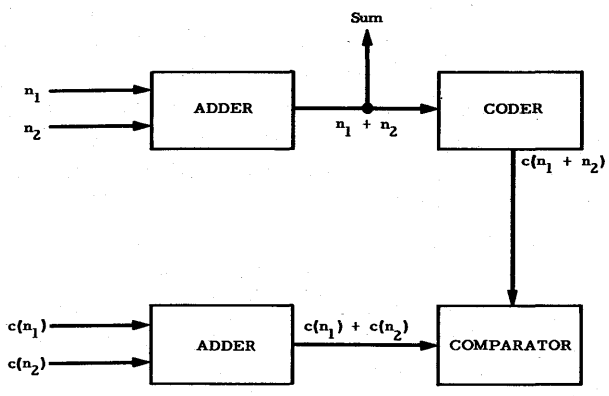



Figure 1. Parallel checking of adder operations.

An example of a contemporary machine which employs residue checking is the Univac III computer. This machine uses residues modulo three, calculating them by a process which is essentially identical to Brown's Theorem 4.[1] (Note that the Univac III checks only addition, subtraction, storage, and data transfer operations. Note also that the residue of the data word is calculated, and the added code bits are generated so as to round out the residue to zero—a sort of residue-three, even-parity system.) Residues modulo three have the properties of detecting all single errors and 50 percent of all double errors.

An additional property of residue checking, also pointed out by Peterson,[2] is the relative ease of handling overflow and other singularities. Possibly the only major disadvantage of residue checking is that it is fundamentally a "distribution-free" check; i.e., indication of error at a validity check gives no indication of the magnitude of the error, or even which group of bits is at fault (odd or even in the case of mod three checking). Further, undetected double errors (which are less probable as larger check moduli are employed) are uniformly distributed over the data word. Hence, undetected errors may be as easily large in magnitude as small. These are, however, minor shortcomings, far outweighed by the advantages of checking adder operation by residues.

Next consider the problem of checking and storage registers. The simple addition of extensions to each register to hold check (residue) codes, coupled with a post transfer validity check, will check registers and transfers adequately. However, two other operations involving registers must be considered: shifting and complementation.

## RESIDUE CHECKING SHIFT OPERATIONS

Basically, shifting may be considered in two different ways: as a mechanical relocation of data bits; as a scaling operation, accompanied by truncation or roundoff of the scaled operand as bits are shifted off. Due to the cyclic properties of residue codes, cyclic shifting of the code corresponds exactly to cyclic shifting of the corresponding data word. Thus, no correction for cyclic shifts (often called rotation operations) is required.

Arithmetic (or so-called "end-off") data shifts are not quite so simple to handle. However, the required corrections are relatively easy to implement.

The required correction rules may be summarized in terms of the four possible cases for dropping a bit off the end of the data or check code registers.

1. Neither drop a bit; i.e., the bit shifted off is a zero: no correction is required.
2. Both drop a one: no correction required.
3. Data bit dropped is a zero, but code bit dropped is a one: cycle (or add) a one into the other end position of the (shifted) check code.
4. Data bit dropped is a one, but code bit dropped is a zero: borrow one from the other end position of the shifted check code. Since this position will contain a zero, this operation is essentially a subtraction operation from the entire check code. It may be implemented simply as follows (assuming that a residue adder is available):

   (a.) Left shift: assuming that an $n$-bit check code is carried, add the quantity $2^n-2$ to the shifted (but uncorrected) check code. This constant may be generated readily by gating. It is essentially $n-1$ one with a zero in the next-to-least significant position.

   (b.) Right shift: add the quantity $(2^{n-1}-1)$ to the shifted check code. This constant may also be generated readily. It comprises a zero in the most significant bit and ones in all other positions.

Examples, illustrating single-place left shifts and residue check codes, for the four cases above:

| | Before Shift | | After Shift | |
|---|---|---|---|---|
| | Data | Check | Data | Check |
| 1.) | 0 1 0 1 1 0 | 0 1 | 1 0 1 1 0 0 | 1 0 |
| 2.) | 1 0 1 0 0 1 | 1 0 | 0 1 0 0 1 0 | 0 0 |
| 3.) | 0 1 0 1 0 0 | 1 0 | 1 0 1 0 0 0 | 0 1 |
| 4.) | 1 0 0 1 0 0 | 0 0 | 0 0 1 0 0 0 | 1 0 |

Double-register shifts, both cyclic and arithmetic, require further corrections. For cyclic shifts, the code registers must be linked to provide both single-register and double-register cycling. The corrections required are (in terms of *each* of the code registers):

1. Bit cycled from both data and code registers is a zero: no correction required.
2. Bit cycled from both data and code registers is a one: cycling is performed on normal double-register basis; i.e., no correction is required.
3. Data bit cycled is a one and code bit is a zero: add the data bit to the other data register and add one to the code register in the corresponding position; borrow one from the vacated position in the code register. (Note that the one cycled into the other code register must be added, since the corresponding position in the other code register may contain a one which was cycled in due to the characteristics of its own data word.)
4. Data bit cycled is a zero and code bit is a one: inhibit cycling of the one into the other code register; cycle the one end-around into the vacated position in its own code register.

These corrections are quite simple to implement, thus making residue checking of arithmetic operations including shifting (such as multiply and divide) quite feasible.

## OTHER CONSIDERATIONS IN RESIDUE CHECKING

Sign correction of residue check codes is quite simple. The check code (which is assumed to be carried with its sign in complement form) is complemented accordingly. Since the residue system employs addition modulo $2^n-1$, it is inherently like the conventional "ones" or digitwise complement system. (For example, zero is represented by either all zeroes or all ones—the equivalent of plus and minus zero in ones complement notation.) Thus digitwise complementation of a residue code gives a result which corresponds to the residue of the negation of the corresponding data quantity which is in twos complement form.

The choice of the residue modulus is of considerable importance. The primary considerations are checking efficacy, ease of implementation, and economy of added check bits. The checking capabilities of a residue code are directly related to the size of the residue employed. The larger the residue, relative to the largest data magnitude, the lower the probability of undetected double errors. Use of residues having simple cyclic properties (such as the mod three residues calculated, in the Univac III,

by shifting and counting) facilitates implementation.

Finally, the residue should be encoded efficiently, i.e., with the largest modulus consistent with the number of check bits carried. Obviously, the largest residue which can be encoded with n bits is $2^n$. However, effective error detection requires that the modulus be relatively prime to the number base or integral powers thereof.[1,4] Hence $2^n$ is not a suitable modulus. The next largest possible modulus is $2^n-1$. This choice, which is always prime to a binary base system, has no disadvantages (in fact, as noted previously it offers advantages when complementation is required). Hence, consideration should be given to residue moduli such as 3, 7, 15, 31, 63, etc.

## CHECKING USING BINARY LOGARITHMS

A unique checking system may be constructed using binary logarithms. While not affording the exact bit-error checks afforded by residue checking, logarithmic checks offer several complementary advantages including:

1. Very good indication of the magnitude of the error; this is an area where residue checking was found to be lacking.
2. Ease of mechanization independently from the arithmetic unit controls; this facilitates checking macro-operations and the arithmetic controls—one of the original objectives.

Logarithms are particularly effective in checking multiply, divide, and square root operations. Since these involve long sequences of micro-operations and are hence most susceptible to both arithmetic and control errors, logarithmic checking appears most attractive. So far as the writer is aware, this is the first proposal for such a system. The idea is based on an article by Mitchell of NCR on multiplication and division using approximate binary logarithms.[4]

Mitchell generates the characteristic of the log by shifting the number left until the leading one is shifted off; the shift count is the characteristic. (This is an exact process. However, it is not suitable for use directly with numbers in complement form.) The mantissa was generated by using the remaining low order bits (after shifting off the leading one). As discussed in reference 4, this corresponds to linear interpolation (Fig. 2) and provides for exceedingly simple implementation.

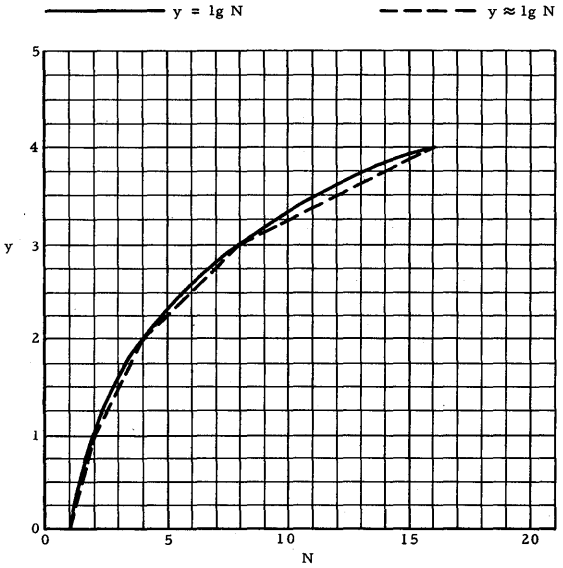



Figure 2. Binary logarithmic curve vs. straight-line approximations.

The price for the simple mechanization of binary log generation is accuracy. Table 1 compares exact vs. approximate binary logarithms. The error analysis in reference 4 shows that multiplication errors range from zero up to —11.1 percent below the correct product. Division results in quotients ranging up to + 12.5 percent high. Considering the low

Table 1. Errors in Approximate Binary Logarithms.

| $N$ | $N$ (binary) | Approx. lg $N$ | lg $N$ | Error |
|---|---|---|---|---|
| 1 | 00001 | 0,000 | 0,000 | 0 |
| 2 | 00010 | 1,000 | 1,000 | 0 |
| 3 | 00011 | 1,500 | 1,585 | 0.085 |
| 4 | 00100 | 2,000 | 2,000 | 0 |
| 5 | 00101 | 2,250 | 2,322 | 0.072 |
| 6 | 00110 | 2,500 | 2,585 | 0.085 |
| 7 | 00111 | 2,750 | 2,807 | 0.057 |
| 8 | 01000 | 3,000 | 3,000 | 0. |
| 9 | 01001 | 3,125 | 3,170 | 0.045 |
| 10 | 01010 | 3,250 | 3,322 | 0.072 |
| 11 | 01011 | 3,375 | 3,459 | 0.084 |
| 12 | 01100 | 3,500 | 3,585 | 0.085 |
| 13 | 01101 | 3,625 | 3,700 | 0.075 |
| 14 | 01110 | 3,750 | 3,807 | 0.057 |
| 15 | 01111 | 3,875 | 3,907 | 0.032 |
| 16 | 10000 | 4,000 | 4,000 | 0 |

cost of mechanization, the accuracies are quite creditable. However, they are somewhat excessive for the supplementary checking operations desired.

Among the possible ways for improving accuracy, the most obvious one is generation of more accurate mantissas using a stored table. A small table —say 64 values—could be stored in a read-only memory at a modest cost. If desired, these values could be subjected to linear interpolation for greater resolution using Mitchell's original scheme applied to the lower bits. This method should produce products and quotients accurate within 2 or 3 percent.

Consideration of the interpolation procedure used by Mitchell indicates that the error is almost a direct function of the distance of the argument from the integral (i.e., zero-mantissa) points. A preliminary estimate might be made using Mitchell's basic system and stored temporarily. The remainder of the data word (now shifted until the leading bit is off) is once again shifted left until the next "one" shifts off, counting the shifts. The resulting shift count is used as the argument to enter a small table of correction constants (say 28 values for a 30-bit data word). The correction constant is then added to the estimated log to generate a more accurate estimate. This process could be repeated a second time, at the expense of time and a larger table of correction constants. No accurate error analysis of this method has been made. However, even the single correction system should produce results good to 1 percent or better, and at a very modest cost.

## DETECTION CAPABILITIES

In calculating the expected error incidence and corresponding detection probabilities, the question arises: "Should the computer word be treated as 30 bits, or as 29 magnitude bits with the sign treated separately?" Since sign errors are of such significance, signed residues should be employed to produce independent sign bit checking. Hence, the analysis which follows will consider only the 29 magnitude bits.

Using the usual notation (see for example reference 5, page 50), the number of possible $m$-bit errors in an $n$-bit word is:

$$C_m^n = \frac{n!}{(n-m)\,!\,m!}$$

Thus the number of double errors possible in the 29-bit word is:

$$C^{29} = \frac{29!}{(29-2)\,!\,2!} = \frac{29!}{27!\,2!} = \frac{29 \times 28}{2} = 406$$

Algebraic or combinatorial formulas for calculating the error-detecting capabilities of residue codes may be formulated; however, the process is quite involved. A simple, heuristic analysis can be accomplished more rapidly (at least for the word lengths and residue magnitudes considered for this problem). Further, this gives some insight into the error distribution—something which is lost when the more formal methods are used. For example, consider residues modulo three (which detect *all* single errors). Brown has given a simple theorem[1] for calculating residues for this case.

*Given*: the digital number

$$X = x_n 2^n + x_{n-1} 2^{n-1} + \ldots + x_1 2 + x_0$$

*Then*: the residue mod 3 of $X$, designated $R)X(3$ is given by:

$$R)X(3 = R_n\,(W_{\text{even}} - W_{\text{odd}})$$

where $W_{\text{even}}$ and $W_{\text{odd}}$ are the number of ones in the even- and odd-numbered digit positions respectively, and $R_n(Y)$ signifies the least nonnegative remainder resulting from dividing $Y$ by $n$.

Consider first double errors in either the even or odd digit positions alone. The error pair may involve unlike digits (i.e., a zero and a one), in which case it is undetected. It may involve like digits, in which case it is detected. Thus 50 percent of the double errors occurring in like-numbered (even or odd) digit positions alone are detected.

Next consider double errors split between even and odd digits. For the error to be undetected, it must give a resulting residue error which is an integral multiple of three—the residue base. Consider the possible bit combinations before and after the occurrence of the double error, given in Table 2.

Table 2.

| Original | | Erroneous | | |
|---|---|---|---|---|
| Odd | Even | Odd | Even | Error Residue |
| 0 | 0 | 1 | 1 | +3 |
| 0 | 1 | 1 | 0 | +1 |
| 1 | 0 | 0 | 1 | −1 |
| 1 | 1 | 0 | 0 | −3 |

Obviously, 50 percent of the errors are undetected, since they result in error residues of plus or minus 3. Now an error pair may occur in like- or unlike-numbered (even or odd) digit positions with equal probability. Hence, the overall probability of detecting a double error using residue checking modulo 3 is 50 percent.

The extension of this analysis to other residue moduli is straightforward and will not be treated here. The double-error detection properties of residue codes of the form $2^n-1$, up to a modulus of 255, are tabulated in Table 3. (Note that the table is based on the problem assigned: checking 29-bit magnitudes.) Also tabulated is a figure of merit which indexes the relative efficiency of each modulus; this is derived as:

$$\text{Figure of merit} = \frac{\text{Detection probability}}{\text{Relative redundancy}} = \frac{\text{Detection probability}}{\text{Check bits/Data bits}}$$

For example, the figure of merit for the residue modulus 31 is:

$$\frac{0.91}{5/29} = \frac{0.91 \times 29}{5} = 5.22$$

Table 3. Double-Error Detection Properties of Residue-Class Checking Codes.

| Residue Modulus | Code Bits | Errors Detected* | Detection Probability | Figure of Merit |
|---|---|---|---|---|
| 3 | 2 | 203 | .50 | 7.14 |
| 7 | 3 | 343 | .84 | 8.10 |
| 15 | 4 | 360 | .89 | 6.37 |
| 31 | 5 | 371 | .91 | 5.22 |
| 63 | 6 | 378 | .93 | 4.49 |
| 127 | 7 | 383 | .94 | 3.89 |
| 255 | 8 | 386 | .95 | 3.45 |

*Out of 406 possible in a 29-bit word.

For a typical constraint—detecting at least 90 percent of all double errors—the residue modulus must be at least 31. Note that the highest figure of merit (for the 29-bit case) is exhibited by residues modulo 7. The modulus 3 is quite simply mechanized (as in the Univac III). A dual-residue system, comprising checks modulo 3 and 7, would have a detection probability of 0.92—slightly greater than that for mod 31 checking. However, the increased check code bit and logic costs of the dual system are strong deterrents to serious consideration over a single residue check system.

## RESIDUE CAU MECHANIZATION

To provide a basis for a trial mechanization of a Residue CAU, a basic arithmetic unit design was selected. (See Fig. 3.) The design is based on the H-3330 Digital Computer developed by the Hughes Aircraft Corporation.[6] The characteristics of the H-3330 may be summarized briefly as:

- Parallel, synchronous, single-address
- 30-bit words, sign and binary magnitude
- Add time: 1.8 microseconds; multiply: 13.5
- Logic: diode-transistor-inverter; 2.2 megacycle clock

The properties of the arithmetic unit registers are:

*A* — Accumulator; double-rank; right- or left-shifting; can be added to *B* or *Q*; can be transferred to *B* or *Q* or Memory; holds most significant half of product after multiply; holds most significant half of dividend before division, and remainder afterwards.

*B* — Double-rank; complement gating; can be added to *A*; can be transferred to *Q*; holds operands read from memory (addend, subtrahend, multiplicand, divisor).

*Q* — Double-rank; right- or left-shifting, alone or linked with *A*; can be added to *A*; can be transferred to *A* or Memory; holds multiplier before multiply, and least significant half of product afterwards; holds least significant half of dividend before division, and quotient afterwards.

*J* — Single-rank; holds overflow, divide check, etc.

The preliminary CAU design is based on using residues modulo 31. The check codes are stored in memory as six-bit, sign and magnitude numbers, together with a parity bit on the check code. Thus, the memory word length must be extended by seven bits. In the arithmetic section, the six-bit residue code is carried in two's complement form. (The arithmetic operands are converted to complement form in the *B* register upon reading from Memory.) While this increases the number of microsteps required, it simplifies the micro-operation control

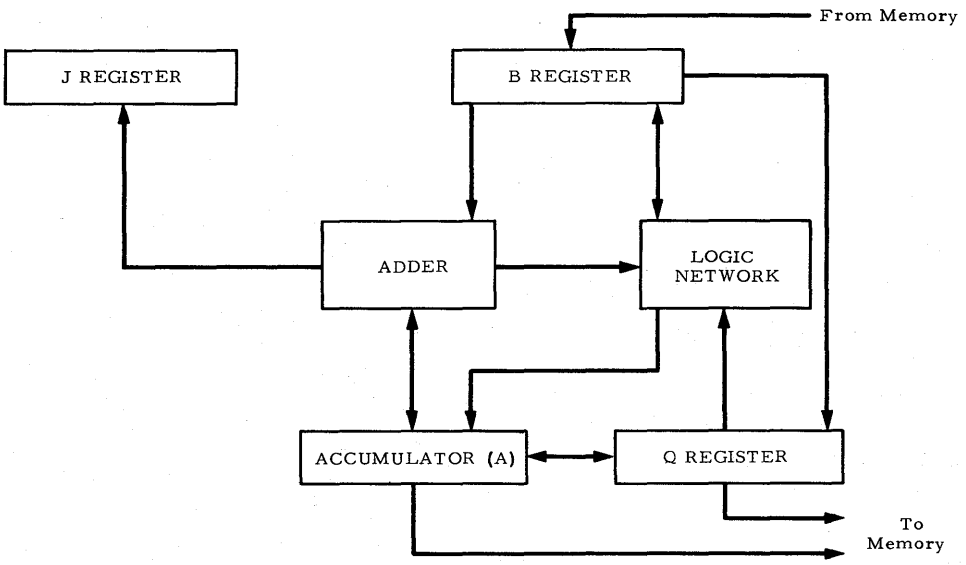



Figure 3. Basic arithmetic unit — block diagram.

since both operands and check codes are subjected to the same sequences of operations, including complementation.

The organization of the Checking Arithmetic Unit is shown in block diagram form in Fig. 4. It may be noted that the general organization is simi-

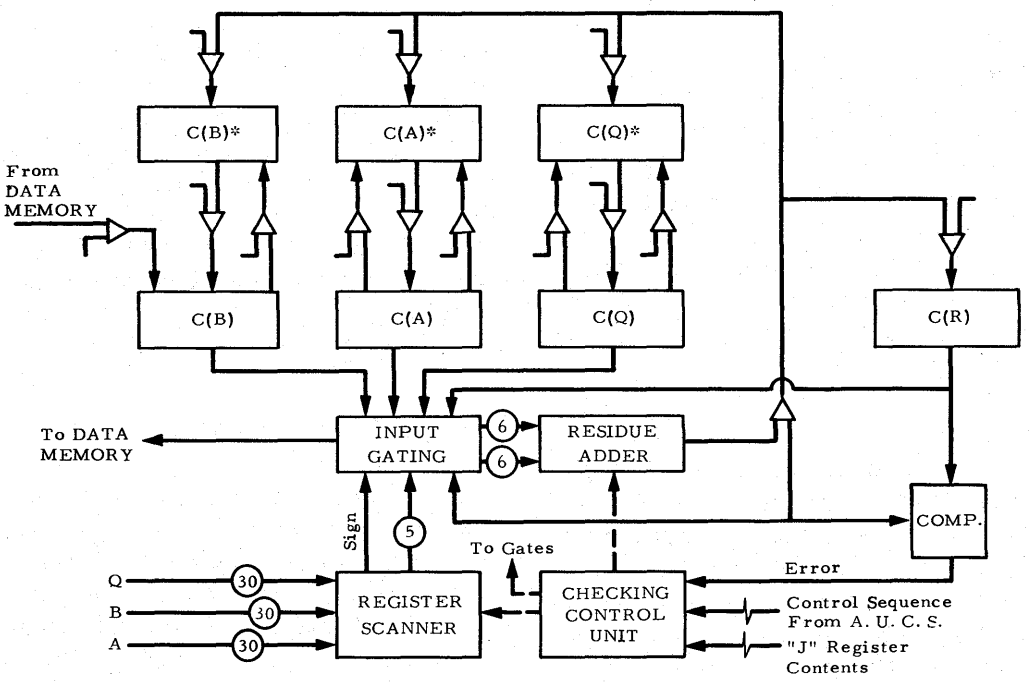



Figure 4. Residue checking arithmetic unit — block diagram.

lar to that of the basic arithmetic unit. The registers are designated according to the corresponding data register; i.e., $C(A)$ holds the check code for the data in the Accumulator. The starred (*) registers are so-called "slave" registers, used for temporary storage during transfers and shifting. The characteristics of the registers (together with the gating logic) are:

- $C(A)$ can be an input to the adder or comparator; can be transferred to $C(B)$, $C(Q)$, or Memory; can shift right or left.
- $C(B)$ can be an input to the adder; gating for digitwise complement; receives check codes from Memory; can be transferred to $C(A)$ and $C(Q)$.
- $C(Q)$ can be an input to the adder or comparator; can be transferred to $C(A)$, $C(R)$, or MEMORY.
- $C(R)$ temporary storage register for residue results during generation of new residue code by scanning data word registers.

The Residue Adder accepts two six-bit inputs plus control signals (end-around carry inhibit and initial carry preset) and performs addition Modulo 31. Inputs and results are in two's complement form, and singularities are signaled to the Checking Control Unit in the usual manner. The Residue Adder is a completely parallel type, based on a network of logic gates connected in a "brute-force" configuration.

The functions of the other units are relatively obvious. The Register Scanner sequentially gates the sign + five-bit increments of a data register (usually one containing the results of an arithmetic operation) into the Residue Adder to generate the mod 31 residue check code. This is compared with the residue calculated during the sequence of arithmetic operations (and temporarily stored in $C(R)$) in the Comparator. In case of error, the Checking Control Unit is signaled.

The Checking Control Unit performs two main functions. During the data arithmetic cycle, it accepts the micro-operation control sequence from the Arithmetic Unit Control Sequencer and generates appropriate gating signals to accomplish the corresponding residue arithmetic operations. Upon completion of the residue arithmetic cycle, the Control Unit then initiates the checking cycle, as described in the preceding paragraph. In case an error is detected, a flag may be set into the *J* Register (which indicates arithmetic singularities) and a program interrupt requested. The Checking Control Unit also initiates residue corrections for data singularities (overflow, illegal division, etc.) based on signals from the *J* Register.

## LOGARITHMIC CHECKING UNIT

Time did not permit a detailed mechanization of the Logarithmic Checking Unit (LCU). However, several pertinent comments may be made. The independence of such a unit from the data arithmetic controls (and even the instruction decoder if desired) offers much greater protection than the Residue CAU. The sensitivity of the LCU to errors of relatively significant magnitude (greater than 1 percent) may facilitate making decisions when the CAU has also detected an error. Note that the residue checking system gives no indication of the magnitude of the error. If the LCU indicates the error is small, it may be possible to continue operation under some conditions.

The operation of the LCU in checking multiplication, division, shifting, and square root are relatively obvious and straightforward. The application to add and subtract is not so simple, since logarithms give only order-of-magnitude checks on addition or subtraction. This may be accomplished by comparing the characteristics of the two operands to give an indication (generally within one or two bit positions) of the magnitude of the result, in terms of the position of the most significant bit. While only a very rough check, it may still be of value in detecting control unit errors.

The LCU also provides for gross checks on many of the decisions (i.e., conditional transfer) operations. The checking of branch on zero, nonzero, plus, or minus is obvious. Branching on magnitude comparison can only be checked for reasonableness if the binary logs differ. If the logs are equal, the magnitudes are equal within a 2:1 or 1:2 range.

## RESIDUES VS. HAMMING CODES

It is illuminating to compare the capabilities of the residue checker with other schemes, such as Hamming codes. Hamming defines the bound $B(n,d)$ of the maximum number of unique code points in a unit cube of dimension $n$, with minimum distance $d$ between them.[7] For the given problem, we need a bound of at least $2^{29}$ points (for the 29-bit magnitude portion of the computer word) and a $d$ equal to three (i.e., double-error detection). Hamming's bound for this case is:

$$B(n,3) = 2^m \leq \frac{2^n}{n+1}$$

For the residue 31 code considered, $n$ is 34, giving a bound of:

$$B(34,3) \leq \frac{2^{34}}{35} = \frac{17,179,869,184}{35}$$

$$= 1.83 \times 2^{28}$$

Thus the use of a 34-bit Hamming code (i.e., one having the same number of redundant bits as the residue 31 code) would fail to give perfect double-error detection for the 29-bit words used. It should also be pointed out that the lower bound defined by Hamming is only valid for so-called close-packed codes (i.e., codes where the factor $2^n/(n+1)$ is an integer). Since the nearest larger close-packed code (Hamming) requires an $n$ of 63, even less double-error detection efficiency can be expected from the 34-bit Hamming code. This comparison with Hamming codes on an efficiency basis, coupled with the ease of implementing residue codes, makes residues the prime choice for checking arithmetic operations.

## CONCLUSIONS

It seems appropriate here to consider the economics of the proposed checking system. A typical H-3330 configuration (central processor and memory only) sells in the one-million dollar price region. Preliminary estimates for the Residue CAU indicated a cost of $50,000. The cost of the LCU was estimated at $30,000. A reasonable price for the dual checking facility (assuming 25 percent gross profit) might be $100,000. Thus, for an additional 10 percent, one can have a system which enjoys extensive protection from undetected errors, both bit and magnitude, as well as provisions for implementing sophisticated operations depending upon the type and magnitude of errors detected.

The proposed checking system includes a proposal for a Logarithmic Checking Unit. This unit offers significant advantages at a low cost. It is believed to be a significant and unique innovation. The discussion of the checking of shift operations using residues is also believed to be new.

## ACKNOWLEDGMENT

The guidance and encouragement of Professor A. Avizienis of UCLA, who posed the original problem and suggested the use of the Checking Arithmetic Unit, is gratefully acknowledged.

Most of the work discussed here was done while the author was a member of the Research Staff at Beckman Instruments, Systems Division, Fullerton, California. The support of the Research Staff is gratefully acknowledged.

## REFERENCES

1. D. T. Brown, "Error Detecting and Correcting Codes for Arithmetic Operations," *IRE Trans. PGEC,* vol. EC-9, no. 3, pp. 333-337 (Sept. 1960).

2. W. W. Peterson, "On Checking an Adder," *IBM Journal,* vol. 2, pp. 166-168 (Apr. 1958).

3. D. S. Henderson, "Residue Class Error Checking Codes," preprint, 16th National Meeting ACM, Los Angeles, Sept. 5-8, 1961.

4. J. N. Mitchell, "Computer Multiplication and Division Using Binary Logarithms," *IRE Trans. PGEC,* vol. EC-11, no. 4, pp. 512-517 (Aug. 1962).

5. H. H. Goode and R. E. Machol, *System Engineering,* McGraw-Hill, New York, 1957.

6. Hughes Aircraft Co., "H-3330 Programming Manual," FR 62-16-123 (1962).

7. R. W. Hamming, "Error Detecting and Error Correcting Codes," *BSTJ,* vol. XXIX, no. 2, pp. 147-160 (Apr. 1950).

# SERIAL ARITHMETIC TECHNIQUES

M. Lehman, D. Senzig and J. Lee
*IBM Watson Research Center*
*Yorktown Heights, New York*

## SERIAL MODE MULTIPROCESSING

It has recently been suggested[1] that the association of serial-mode functional (arithmetic) units with multi-instruction counter, multiprocessing computing systems may result in highly efficient processing complexes. This follows from the fact of life that in going from parallel to serial realizations of various algorithms the hardware requirements fall much more rapidly than does the speed. Furthermore the speeds of the slower, serial, arithmetic units may be more closely matched to those of memory and to other parts of the system. Thus the need for extra control circuitry, for high-speed registers, queueing circuits and look-ahead control, for example, is reduced and the system's overall cost/performance ratio improved. For appropriate serial configurations then, performance may be improved relative to a parallel system when system costs are to be held constant. Reconfiguration of a fast parallel circuit, for example, can yield a number of slower serial devices which, when kept busy, will increase throughput of the system.

The crux of the problem of optimizing throughput lies, of course, in the extent to which the large fast unit on the one hand and the several associated, slower, smaller units on the other can be kept busy; that is, their relative efficiency within an operating computing environment. As previously demonstrated,[1] this consideration leads quite naturally to the multiprocessing configuration.

Recognizing that serial-mode operation may once again assume significance in the field of high-speed processing, we now wish to outline some appropriate algorithms. Thereby we hope to draw attention to techniques other than those currently considered during architectural, system and logical design activities. We shall not, however, in general, include detailed assessment of performance (speed and/or hardware) nor engage here in comparisons of serial and parallel devices. Such general comparative studies are meaningless when divorced from technologies and actual designs. Extrapolations from them to actual physical design situations are in general not valid and their usefulness is restricted to helping towards a deeper understanding of the various circuits and techniques discussed and compared.

## BASIC CONCEPTS

### *Serial Operation*

Traditionally a serial machine has been one operating on words, numbers, in a bit-by-bit fashion. We now intend to use the term to include bit-parallel, byte-serial, operation. Thus with a $k$-bit byte we may

conveniently regard the machine as operating radix $2^k$. We expect that the cost/performance ratio for the multiprocessing system considered, will be optimized for such serio-parallel operation. For the present discussion we consider $1 \leq k \leq 5$. This range arises from timing and speed considerations which suggest, for example, that the basic radix $2^k$ adder should be able to add two bytes in one bit-time as defined in the next section. Optimum $k$ will then be determined mainly by the fan-in and fan-out of the logical elements being used.

*Time*

In conventional high-speed machines the absolute measure of speed is generally established in terms of the average, effective time delay $t$ (nowadays in nanoseconds) experienced by a signal or signal pattern in passing through a logical element or level. Naturally $t$ will depend on the particular technology being used, that is, on the logical complexity of the element and on its fan-in and fan-out characteristics. Further, for a given product line $t$ will spread over a range of values and the in-place delay of a circuit will depend on its loading. However, we may use $t$ symbolically to indicate speed, giving the duration of a specified operation to a first order of accuracy in terms of the number of logical levels its implementation requires.

In designing synchronous serial circuits it is not necessary or indeed desirable to synchronize signals at each logical level. Thus there emerges a new concept, the bit-time, $t_b$, which represents the time interval between clock signals. $\frac{t_b}{t}$ then represents the maximum number of logical circuits or levels between points at which clock signals are applied. A lower limit for $t_b$ arises from the maximum operating rate of a flip-flop (latch) assembled from, or at least equivalent to, two or more logical elements. In general, $t_b$ will exceed $2t$ and a practical operating range will be $2t \leq t_b \leq 4t$. Thus typically we may consider elements with a logical structure of complexity comparable to that of an AND/OR complex, with fan-in and fan-out in the range 5 to 10, an average delay of order 1ns, and operating at a rate of order $3 \times 10^8$ bits per second, that is a bit time of order 3ns.

When discussing serial techniques we also make frequent use of the term "word-time" which in an $n$-bit binary machine generally refers to n bit-times. In an $\frac{n}{k}$ -byte, radix $2^k$ system, processing the $k$-bit bytes in parallel, the word-time will be $\frac{n}{k}$ bit-times.

*Registers*

The serial shift-register is a component whose ready availability is fundamental to the concept of serial operation. In the past these registers have usually taken a distributed, dynamic, transient form; delay lines (ultrasonic, magnetostrictive, electric) and magnetic drum circulating tracks all having been used. Discrete, static, permanent registers using tubes or transistors have, however, proved economical, serving as both storage registers and shifting devices. With the second type of device it is simple to arrange for fully parallel input or output so that the register may also serve as a serial/parallel converter. This function is of importance in the general type of system where information is processed serially, in parallel, or in some more general mode, performance being optimized at each point in the system according to appropriate parameters of the algorithm being implemented. Typically we might wish to read from memory in parallel but perform arithmetic on the extracted number byte-serially.

The discrete, static shift-register appears to be the ideal candidate for realization in integrated or monolithic form when configurations larger than simple gates or adders are considered. Thus extensive use of its properties in systems design should prove rewarding. Its expected ready availability in the technologies of the future will in fact strongly reinforce the trend to increased use of bit-by-bit processing techniques.

## SHIFTING

Shifting is a basic operation in its own right and is also an essential part of the synthesis of various other functions. In particular it may be required for alignment and fast normalization in floating-point operations and as an integral part of multiplication and division algorithms. Information stored in a shift-register may be immediately shifted any number of places at a rate or order $t_b$ $n$'s per bit or byte, a rate comparable to that of the parallel, cyclic shifter. In the most general circuit this shifting can occur in either direction. For more economical devices shift-

ing, say, to the right only, a left $p$-place shift will require $\left(\frac{n}{k}-p\right)$ right shifts with end-around feed. For long shifts, the simple circuit is slower than a full pyramid-shifting circuit but by using radix $2^k$ shifting registers with multiple path interconnections, as in Fig. 1, the maximum serial shift time for any number of places shifted can be kept below some $2k$-bit times. This arrangement permits shifts of any size to be obtained. However if shifting can be restricted

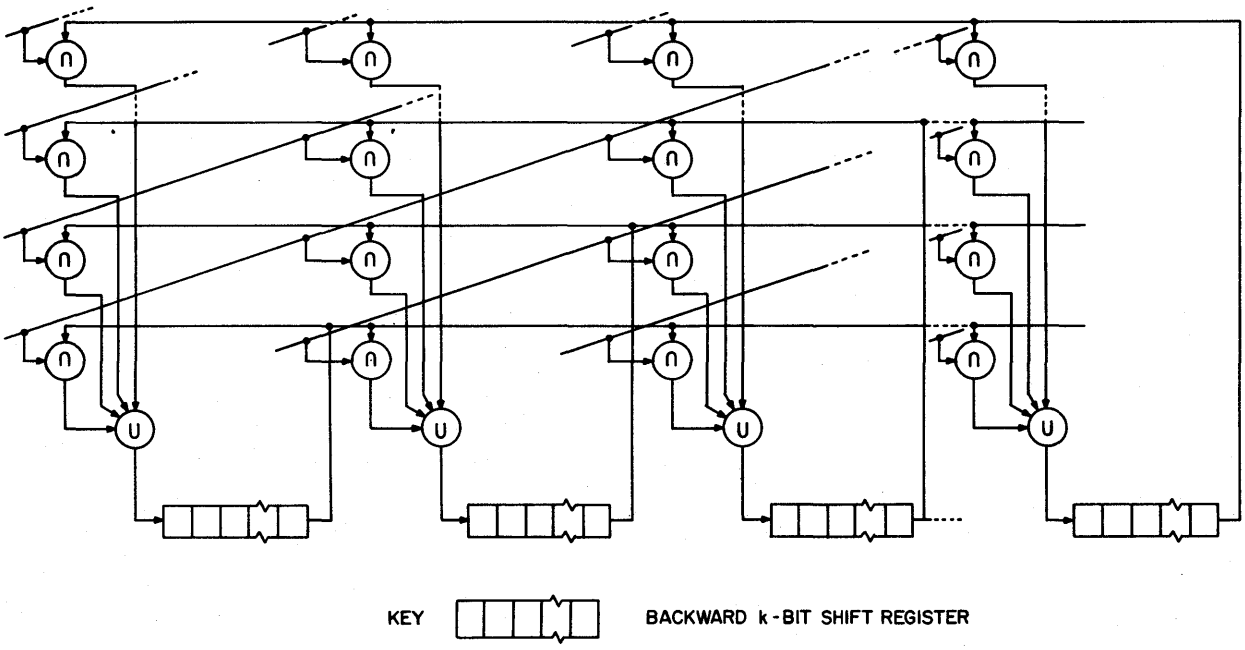



Figure 1. A bidirectional shift register.

to be by byte, that is, radix $2^k$, it becomes practical to consider a very simple register structure as in Fig. 2.

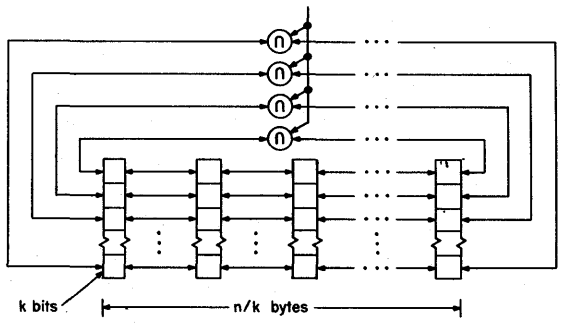



Figure 2. Byte-shifting register.

The requirement for shifts in floating-point operation is well known. The particular techniques appropriate to serial-mode operation are perhaps not so familiar. In addition, for example, it is normally necessary to obtain a relative shift between two numbers and the shift register then has properties like those of the serial matrix store.[2] That is, the contents of one register are "shifted" relative to those of another simply by delaying the application of shift-control pulses to one or other of the two registers. An alternative technique, faster but requiring more hardware, would use shifting registers with several output points (a tapped "delay-line"). Relative shifts are then obtained by choosing the appropriate output point.

When dynamic storage devices are being used, tapped delay lines with elements of delay $t_b$, $2t_b$, $4t_b$, etc., have also been used[3] to permit the simple control of from one to $n$ place shifts directly from a shift control register.

## ADDITION

Techniques for serial, fixed point addition in a full adder, with dynamic or flip-flop storage of the carry, are well known. For radix-$2^k$ operation, the byte sum should be available in one bit time and this can be done in a $k$-bit parallel adder using a direct implementation of the radix sum-logic or binary full-adders with a carry speed-up network.[4] It is largely the practical implementation of this one bit-time parallel

adder that determines an upper bound for $k$. The fixed point addition time will be $\frac{n}{k}$ bit times or $\frac{nt}{k} \cdot \frac{t_b}{t}$ where $\frac{t_b}{t}$ is the bit time to logic-level-delay ratio. This time must be compared with the time $(1 + \log_2 n) \cdot t$, a lower bound on the speed of any adder which follows from the result of Winograd.[5] That is, the ratio of serial to parallel addition time will be less than $\frac{n}{k(1+\log 2^n)} \cdot \frac{t_b}{t}$. For example, if $n$, $k$ and $t_b$ equal 50, 4 and 3 respectively, the serial by byte adder will be less than 6.6 times as slow as the *best possible* parallel adder. The serial adder unit will require relatively less carry speed-up hardware (if any) than a single stage of the parallel unit so that the hardware ratio between parallel and serial devices will be in excess of $\frac{n.}{k}$. Thus the cost/performance ratio of the serial unit is considerably better than that of the parallel device, an improvement which is maintained or even increased when floating-point adders, multipliers and dividers are considered.

Floating-point addition algorithms require the addition to be preceded by an exponent comparison and, where the exponents are unequal, a shift. Typical ranges for the exponents will require them to be represented by some eight bits or from one to three bytes. It may prove advantageous to process the exponents in a small parallel unit though the time penalty for processing serially will be small, only some two bit times, in the case of three byte exponents. The alignment shift will be obtained using one of the techniques described in the previous section. As shown there, no physical shifting of the operands need be performed and the size of the alignment will not significantly affect operation time.

Normalization of the sum obtained from a floating-point adder appears to be standard programming practice and in many machines normalized addition only is provided. This additional (micro) operation is time-consuming and appears to serve no useful purpose whatsoever. The present discussion will not pursue this point.[6] The authors could not, however, have reviewed techniques for serial normalization in connection with the addition operation without at least questioning its utility.

The most common technique for normalization shifts the sum argument one bit or byte at a time until an indication is obtained that standard form has been achieved. This cyclic procedure, widely used in both serial and parallel arithmetic units, is economical but slow and makes the duration of additions operand dependent. Its use is, however, difficult to avoid in parallel units except through the provision of a large amount of extra hardware which may double the cost of the adder complex and also slow it down significantly. In serial units, however, a simple algorithm, used already in the Manchester Mark I computer,[7] permits determination of the normalizing shift and hence the exponent increment during the actual addition process. A counter is required which is reset initially and every time two neighboring sum bits are found to be unequal. Otherwise it is incremented by one during each bit time of the addition process, yielding immediately on its completion the shift number/exponent increment. Use of the fast shifter then permits rapid completion of the floating-point addition.

An alternative, faster technique requires the provision of an additional register. As the sum output appears from the adder it is received in the conventional fashion by the sum register. A second register is connected by a parallel transfer mechanism to the sum register and receives its contents whenever the exponent increment counter is reset to zero, that is, whenever two succeeding sum bits are unequal. Thus on completion of the addition the normalized sum will appear in the second register and no further shifting is required.

## MULTIPLICATION

Exponent arithmetic has been adequately covered in the preceding section and we may, in our discussion of multiplication, restrict ourselves to the fixed point variety, considering first the purely bit-serial device. The basic cyclic multiplier, requiring of order $n^2$ bit times, that is $n^2 . t_b$, is well known. By using a number, $a$, of adders the multiplication time may be reduced to be of order $\frac{n^2}{a} \cdot t_b$. Thus in the Manchester Mark I machine 20 adders were used (with a 40-bit word) to yield a two basic-cycle multiplier.[7] In the limit by using $n$ adders (or fewer than $n$ adders with a recoding scheme), as in Fig. 3, a two word time multiplier is obtained. The least significant half of the double length product is here available after one word time as is the most significant half, but the latter is in redundant (pseudo-sum and carry) form. Thus serial multiplication may be further speeded

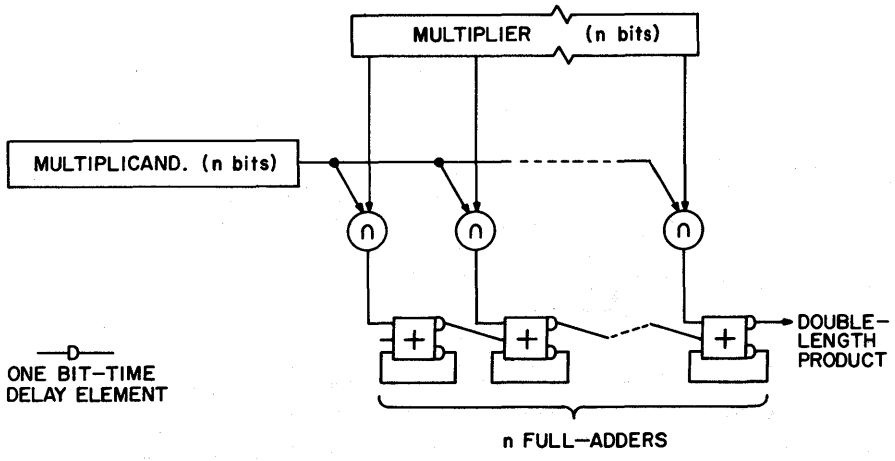



Figure 3. The two word-time shift multiplier.

up by attaching a parallel carry-assimilation circuit as in Fig. 4 to yield a multiplication time of order $(n+\log_2 n)\ t_b$. It is interesting to note that the scheme that now results is almost identical to the parallel stored-carry multiplier scheme first advocated by Burkes, Goldstine and von Neuman[8] and implemented in the Illiac II.[9]

The circuit may be further improved by recoding the multiplier bits in $k_2$-bit groups, using the output of the recoder to control the selection of appropriate multiples of the multiplicand in the selector circuits. Also we may revert to $k_1$-bit byte operations to obtain, as in Fig. 5, a multiplier array still containing only $n$ full-adders. The development of the circuit may also be taken one stage further as in Fig. 6 by providing $q$ such arrays each processing, say, $\frac{n}{q}$-bit portions of the multiplicand.

In all these arrangements we clearly recognize the inadequacies of the classical terms *serial* and *parallel* for classification purposes, nor is it terribly important to attach such a name to the circuit. What is

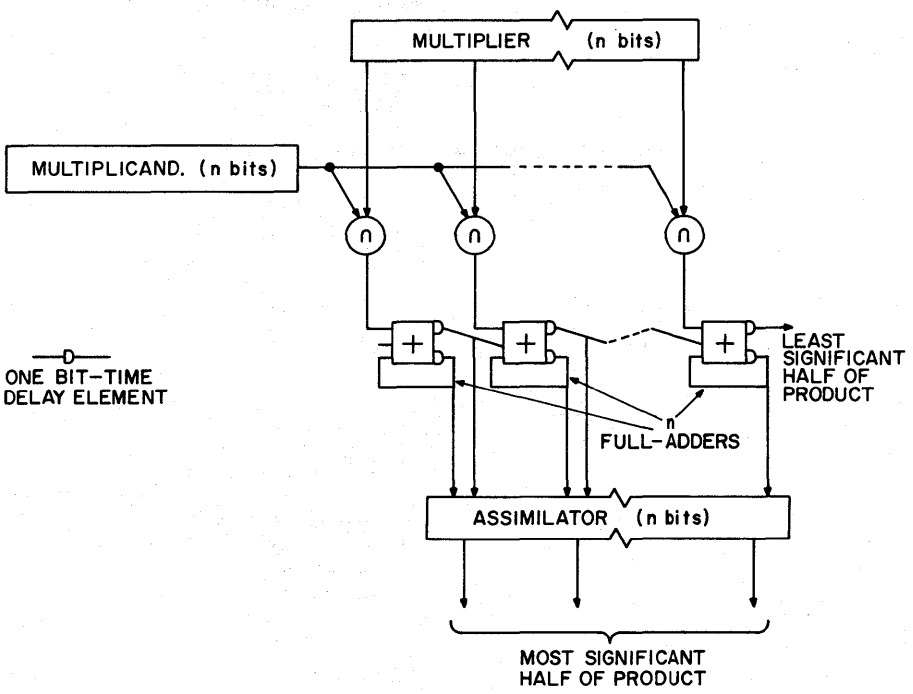



Figure 4. The one word-time shift multiplier.

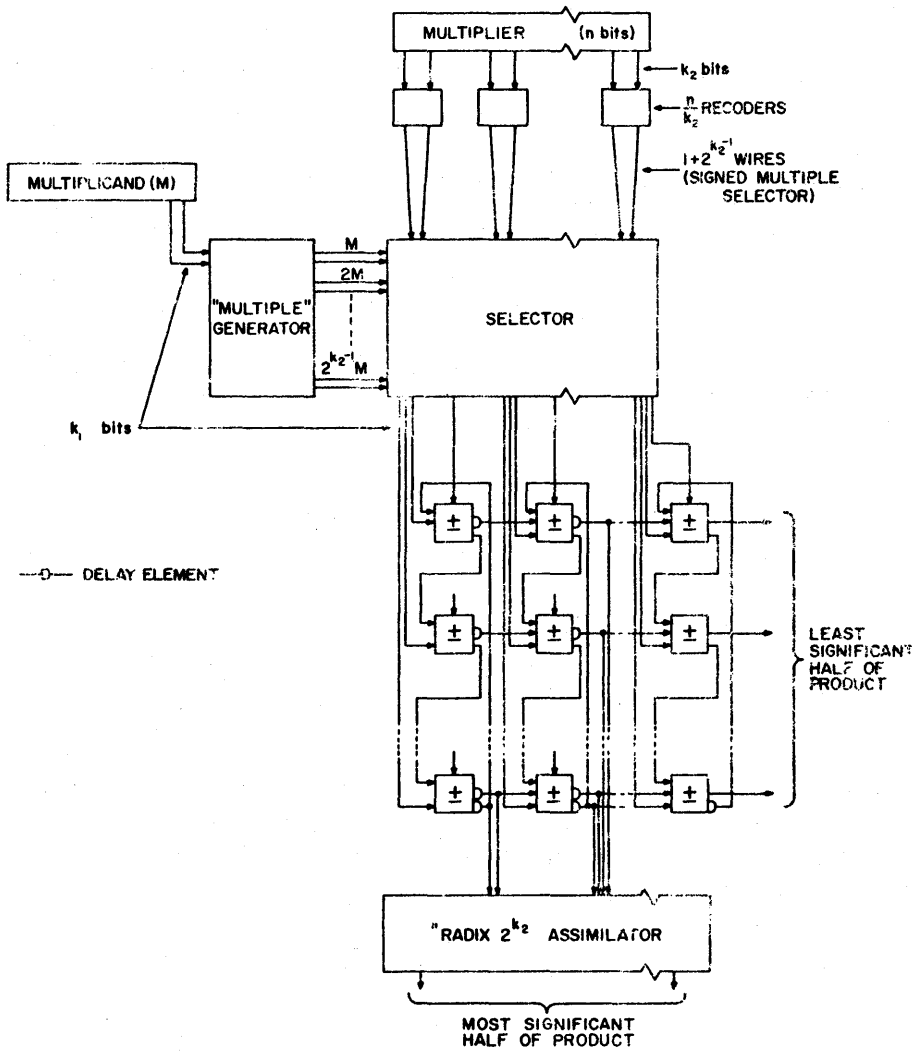



Figure 5. Partially parallel shift multiplier.

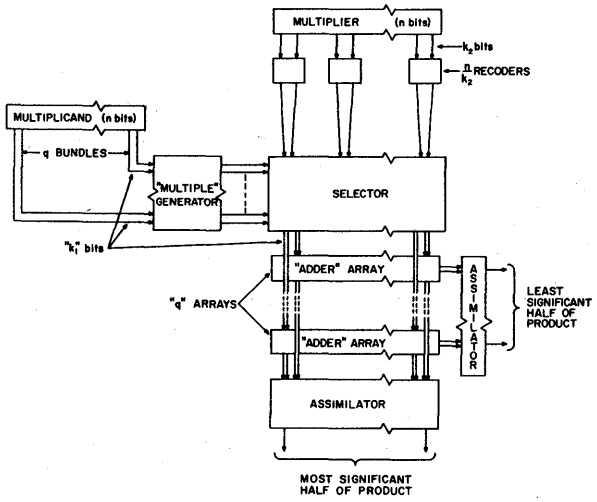



Figure 6. Multiarray shift multiplier.

important is that by starting with a serial approach to multiplier design and by making use of the properties of synchronous, sequential operation in and with a shifting register, there has emerged a circuit whose performance may, at much lower cost, approach that of the fully parallel inverted-pyramid multipliers.[10,11,12] In the Appendix we quote some preliminary results for a multiplier design corresponding to the configuration of Fig. 5. These are compared with an iterative parallel multiplier designed to be of the same speed. These and other results substantiate a more general assessment that the cost/performance of the new configurations exceeds that of the parallel structures by a factor of between two and three.

## DIVISION

The basic restoring and nonrestoring division techniques are inherently slow since they require a minimum of order $n$ subtraction and/or addition cycles. They may be supplemented by one or more of a number of speed-up techniques,[13,14,15] but Freiman's analysis[16] indicates that in practice these alone cannot yield a speed-up factor greater than about four.

We restrict ourselves therefore to a brief discussion of a technique previously[17] outlined for decimal dividers. The technique is illustrative of a class of division methods which integrate readily with fast multiplier circuits to form efficient arithmetic units.

Consider a required remainderless division:

$$Q = \frac{a}{b} \qquad 1 > |b| \geq \frac{1}{2}$$

Suppose a factor $m$ could be determined such that $mb = 1$, then $Q = ma$. The algorithm to be described defines an iterative procedure for determining a sequence $m_i$ such that

$$b_i \, \pi \, m_i \rightarrow 1$$

hence

$$a_i \, \pi \, m_i \rightarrow Q$$

both quadratically. Let

$$\begin{aligned} b_0 &= b, & b > 0 \\ &= 2 + b, & b < 0 \end{aligned}$$

We may write

$$b_0 = 1 + \delta \cdot 2^{-t_0} + \epsilon 2^{-t_0}$$
$$1 > |\delta| \geq 2^{-t_0} \text{ or } 0.1 > \epsilon \geq 0.$$

Define

$$\begin{aligned} m_0 &= 1 - \delta \cdot 2^{-t_0} \\ &= 2 - 2^{-t_0} [b_0 \cdot 2^{2t_0}] \end{aligned}$$

Consider

$$\begin{aligned} b_1 &= m_0 b_0 \\ &= 1 - (\delta^2 - \epsilon) 2^{-2t_0} - \delta \epsilon 2^{-3t_0} \end{aligned}$$

$b_1$ deviates from one at least quadratically less than $b_0$. Generalizing we may thus define an iterative, convergent procedure

$$\begin{aligned} b_{i+1} &= m_i \, b_i \\ t_{i+1} &= 2t_i \\ m_{i+1} &= 2 - 2^{-t_{i+1}} [b_i \cdot 2^{t_{i+1}}] \\ a_{i+1} &= m_i \, a_i \\ t_0 &= 1 \\ m_0 &= 3/2 \qquad b > 0 \\ &= 1/2 \qquad b < 0 \end{aligned}$$

This yields $b_i \rightarrow 1$, $a_i \rightarrow Q$ to the accuracy of $a$, $b$ in some $2 + [\log_2 n]$ iterations.

Convergence can be speeded up by determining an initial value for $m_0$ (that is, some $m_i$) from a wired-in table look-up procedure — as a function of the leading bits of $b$. A 16-entry table based on four bits of $b$, as in Table 1, is practical and would save two iteration cycles, producing a 32-bit quotient in 4 cycles or 33 to 64 bits in 5 cycles.

Table 1. A Typical $m_0$ Coding Table.

| Five Most Significant Bits of Normalized Positive Divisor | $m_0$ |
|---|---|
| 0.10000 | 1.1110 |
| 0.10001 | 1.1110 |
| 0.10010 | 1.1100 |
| 0.10011 | 1.1010 |
| 0.10100 | 1.1000 |
| 0.10101 | 1.1000 |
| 0.10110 | 1.0110 |
| 0.10111 | 1.0110 |
| 0.11000 | 1.0100 |
| 0.11001 | 1.0100 |
| 0.11010 | 1.0100 |
| 0.11011 | 1.0010 |
| 0.11100 | 1.0010 |
| 0.11101 | 1.0010 |
| 0.11110 | 1.0001 |
| 0.11111 | 1.0001 |

The algorithm leads to a divider faster than the shortcut technique only if a fast multiplier is available. Figure 7, drawn for simplicity for bit-by-bit operation, indicates that a circuit realization integrates well with the fast multiplier discussed in the previous section. For $k$-bit byte operation a total division time of order $\frac{n}{k}$ $[\log_2 n]$ bit times should be attainable. This speed can be obtained using a single, split, multiplier circuit; since the length of the multiplier $m_i$ can always be held less than $\frac{n}{2}$ bits except in the last iteration cycle. In that cycle, however, only one product $a_i m_i$ has to be determined.

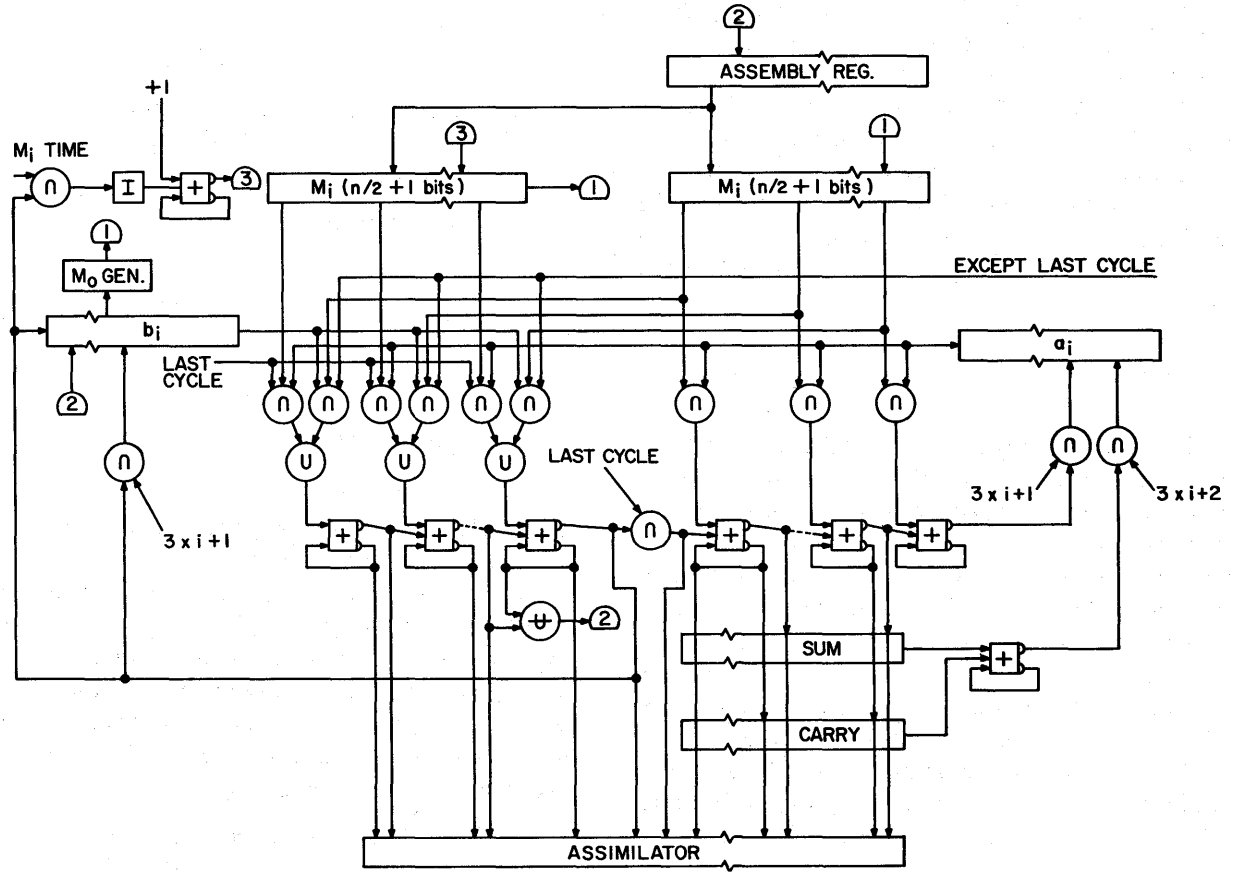



Figure 7. Integrated multiplier-divider.

A first estimate of hardware requirements for the integrated multiplier-divider and its control has been made. The results are presented in the Appendix.

## CONCLUSION

The paper has presented a review of some techniques for the realization of the common arithmetic function. It is considered that generalized performance figures for the various circuits are not of direct interest if based on a particular technology or system design. Care has therefore been taken to avoid repeated comparisons or claims. The results presented in the Appendix enable initial evaluation of the multiplier techniques discussed and their comparison, in terms of speed, cost and cost/performance ratio, with more conventional units. They are included so as to indicate that some of the intuitive notions presented can be substantiated in fact.

Despite the title of this paper, the discussion has not been limited to techniques of a purely bit-by-bit or byte-by-byte nature. The aim has rather been to indicate that with serial processing as a starting point, algorithms and circuits can be developed that will bear comparison with the classical high-speed configurations. In general they will yield speeds slower than the latter. However there exists an ultimate barrier to ever higher processing speeds using parallel structures due to the saturation effects of circuit speeds, cost and system complexity. Thus

the use of techniques such as those described in a multiprocessing, multi-instruction counter configuration is believed capable of yielding performance and throughput superior to that of presently envisaged conventional systems.

In the past designers have, on the whole, pursued an evolutionary path. With the very rapid development that has taken place in computer applications, in systems concepts and in technology it is, however, necessary from time to time to reexamine some of the more fundamental notions of machine structure. How, for example, should the fact that the cost of a monolithic, serial shift-register is likely to approach that of an individual flip-flop, with its input and output stages, affect systems structure? Decisions which were valid in 1947 are not necessarily valid in 1965 and concepts should not be rejected merely because they are unconventional or because they have been rejected or abandoned in the past. In essence therefore this paper suggests and illustrates, with one example, what may be gained by fundamental rethinking in approaching the problems of system and logical design.

## APPENDIX

A circuit design for the serio-parallel multiplier has been completed. The implementation is straightforward and represents only the first round of the normal iterative, optimizing design procedure. Hence it is expected that the hardware requirements summarized below could be somewhat reduced. A matching design for a more conventional, iterative, pyramid multiplier as in Fig. 8 was undertaken so as to enable an objective comparison of the two approaches. Since the designer who undertook both designs (J. Lee) had had previous experience on the design of conventional multipliers, it is expected that any bias in the degree of optimization of the two designs would operate in favor of the latter.

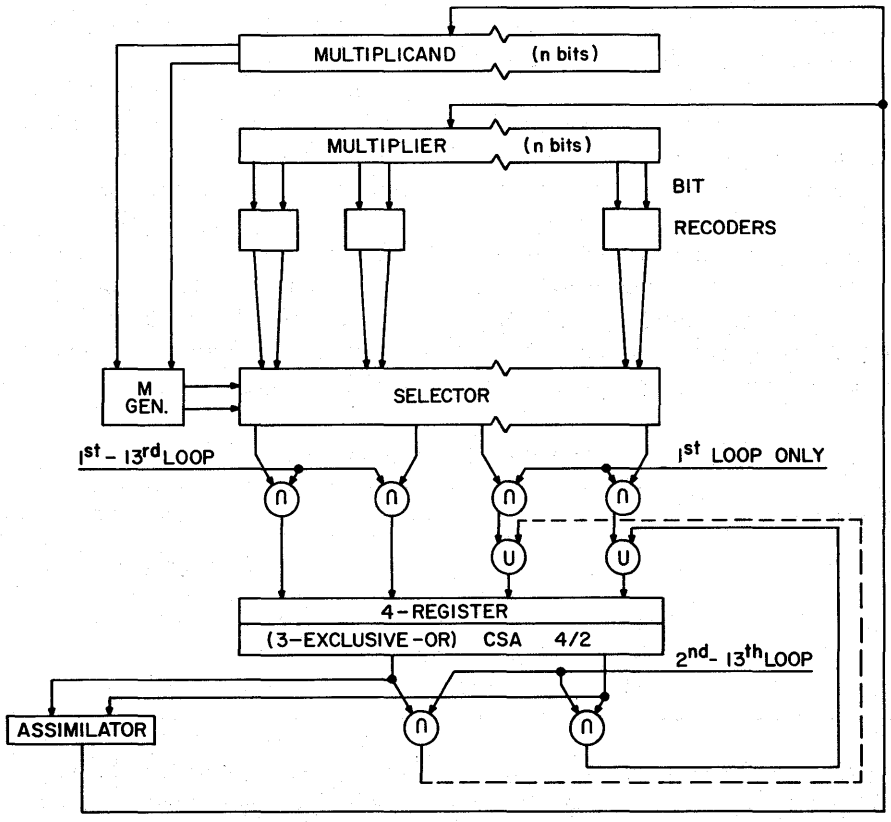



Figure 8. Conventional iterative pyramid multiplier.

All designs were undertaken for 24, 48 and 56 bit operands with a 3-bit byte. They included all logic, registers and powering elements required to produce a double-length product. Circuits used were of two basic types. The first: a current-switching

NOR/OR circuit with a fan-in of up to 4, a fan-out of 10 on each of the 2 outputs and a direct connection (dotting) capability of 4 for either an additional AND or OR function. The second circuit, termed a spread-gate, yields functions, each of 3 out of 6 input variables, on 4 outputs. Fan-out and dotting capabilities remain at 10 and 4 respectively.

The comparative results of the designs are given in Table 2.

It should be noted that (in this age of integrated and monolithic circuits) numbers of circuits rather than transistor counts are the really significant cost characteristic. The latter has, however, been recorded to provide a standard of comparison with earlier designs and also reliability may ultimately be related to that figure.

Table 2. Multiplier Circuits.

| Word-Length | Serio-Parallel | | | Parallel, Iterative Pyramid | | |
|---|---|---|---|---|---|---|
| Bits | Circuits | Transistors | Logic Levels | Circuits | Transistors | Logic Levels |
| 24 | 1,085 | 5,412 | 46 | 2,403 | 8,605 | 45 |
| 48 | 2,165 | 8,355 | 78 | 4,429 | 17,435 | 78 |
| 56 | 2,419 | 9,869 | 88 | 5,633 | 20,157 | 85 |

This investigation was based on the serio-parallel design. That is, having obtained the results for that circuit, iterative multipliers were designed to require an approximately equal number of logic levels, that is, they were designed to be of approximately the same speed. An alternative would have required determination of the speed (logic-levels) of parallel multipliers using a like number of circuits. This approach proved impractical since the number of circuits permitted would have been too small to make a true iterative pyramid multiplier, with carry assimilator, possible.

The conclusions from the results of the table are clear.

The serio-parallel circuit is more than twice as "efficient" as the parallel scheme. This result is substantiated by a comparison with still another design for which estimates were available. This design was for a 48 bit multiplier yielding a single length product in some 31 levels and required about 10,000 circuits or 33,000 transistors.

A circuit estimate for a 24-bit integrated, bit-by-bit multiplier divider as in Fig. 7 has also been undertaken. Results are quoted in Table 3.

Table 3. Multiplier-Divider

| | $m_0$ Length | Circuits | Transistors | Logic Levels | |
|---|---|---|---|---|---|
| | Bits | | | Multiplication | Division |
| Multiplier | — | 951 | 3,893 | 102 | — |
| Multiplier- | 3 | 1,331 | 5,196 | 102 | 406 |
| divider | 6 | 1,581 | 6,696 | 102 | 310 |

The table indicates that the cost of a multiplier-divider is some 50 percent greater than that of the multiplier alone. Moreover, division time is some three or four times as great as multiplication time. Enlarging the $m_0$ table from 3 to 6 bits costs some 250 circuits and yields a 25 percent increase in speed and would normally be well worthwhile. Finally, comparing Tables 2 and 3, we see the effect of changing from bit serial to byte serial operation. In the examples given the change to the 3-bit byte has resulted in more than doubling the speed for an additional hardware investment of only some 14 percent.

REFERENCES

1. M. Lehman, "Serial Mode Operation and High Speed Parallel Processing," *Information Proc-*

*essing 1965, Proceedings of IFIP Congress 65,* part 2, New York (in press).

2. ———, "Serial Matrix Storage Systems," *IRE Trans. on Electronic Computers,* vol. EC-10, pp. 247-252 (June 1961).

3. K. Lonsdale and E. T. Warburton, "Mercury, A High-Speed Digital Computer," *Proc. IEEE,* suppl. no. 2 (*Digital Computer Techniques*), vol. 103, part B, pp. 174-183 (Apr. 1956).

4. M. Lehman, "A Comparative Study of Propagative Speed-Up Circuits in Binary Arithmetic Units," *Information Processing 1962, Proceedings of IFIP Congress 62,* North Holland Publishing Co., Amsterdam, 1963, pp. 671-677.

5. S. Winograd, "On the Time Required to Perform Addition," *J. Assoc. Comp. Mach.,* vol. 12, no. 1, pp. 277-285 (Apr. 1965).

6. R. L. Ashenhurst and N. Metropolis, "Unnormalized Floating Point Arithmetic," *J. Assoc. Comp. Mach.*; vol. 6, no. 3, pp. 415-428 (July 1959).

7. T. Kilburn et al, "Digital Computers at Manchester University," *Proc. IEEE,* part 2, vol. 100, pp. 487-500 (1953).

8. A. W. Burkes, H. H. Goldstine and J. von Neuman, "Preliminary Discussion of the Logical Design of an Electronic Computing Instrument," part 1, Institute of Advanced Study, Princeton, N. J., 1947.

9. D. J. Wheeler, "The Arithmetic Unit," University of Illinois, Digital Computer Laboratory, Report No. 92 (1959).

10. E. Bloch, "The Central Processing Unit," *Planning a Computer System,* W. Bucholz, ed., McGraw-Hill, New York, 1962, chap. 14.

11. C. S. Wallace, "A Suggestion for a Fast Multiplier," *IEEE Trans. on Electronic Computers,* vol. EC-13, pp. 14-17 (Feb. 1964).

12. T. Lamdan, "Some Aspects of the Design of a Simultaneous Multiplier for a Parallel Binary Digital Computer," Ph.D. Thesis, University of Manchester, 1963.

13. M. Lehman, "Parallel Arithmetic Units and Their Control," Ph.D. Thesis, London University, Feb. 1957.

14. K. D. Tocher, "Techniques of Multiplication and Division for Automatic Binary Computers," *Journ. Mech. Appl. Math.,* Aug. 1958, pp. 364-384.

15. J. E. Robertson, "A New Class of Digital Division Methods," *IRE Trans. on Electronic Computers,* vol. EC-7, pp. 218-222 (Sept. 1958).

16. C. V. Freiman, "Statistical Analysis of Certain Binary Division Techniques," *Proc. IRE,* vol. 49, no. 1, pp. 91-103 (Jan. 1961).

17. R. K. Richards, "Arithmetic Operations in Digital Computers," Van Nostrand, Princeton, N.J., 1955, pp. 279-282.

# SIMULATION MODELS FOR PSYCHOMETRIC THEORIES*

Carl E. Helm
*Department of Psychology, Princeton University*
*Princeton, New Jersey*

## INTRODUCTION

My topic concerns another instance of a new methodology for the behavioral sciences made feasible by the availability of large fast computers. The strategy which characterizes this methodology is the attempt to simulate by means of computer programs, the paradigms which constrain and direct the behavior of an experienced behavioral scientist as he attempts to deal with the problems and data of interest to him. The significance for psychology of this computer-based methodology is similar to that of psychometric methodology to the extent that it provides formal systems for the explication and publication of the intuitions of the experienced psychologist.

By way of background for those of you who may not be psychometricians, let me briefly review the problem area. The formal measurement of individual differences, both psychological and physical, is a significant feature of almost all processes associated with an individual's movement across the boundaries which separate structures and organizations in our society. We can characterize research in the measurement of individual differences as directed to providing simple, reliable, operationally useful tests which are designed to compare the behavior of two or more persons.

The procedures for developing scoring rules for such tests are usually based on both clinical and actuarial considerations. For example, the items which make up a personality test may be selected on the basis of clinical judgment by psychiatrists and psychologists. Such a test will then be given on an experimental basis to groups of normal subjects and to groups of subjects with a variety of mental disorders. The scoring procedures finally developed for such a test will be based on the differences in the patterns of responding of the different groups. Thus items which fail to discriminate the two groups will usually be discarded. In the case of essay examinations designed to measure writing ability, scores are typically determined on the basis of clinical judgments. It is of particular interest that there are significant differences between the pattern of scoring of essays by college English teachers and the pattern of scoring of the same essays by newspaper editors.[1]

Given some standard method for scoring a test, it is necessary to develop procedures to optimize the effective use of the test. With only one test and a single measure of the criterion, the problem is most

*This work made use of computer facilities supported in part by National Science Foundation Grant NSF-GP579. These studies were aided by contract Nonr-1858(43), between the Office of Naval Research, Department of the Navy, and Princeton University.

accurately solved by a straightforward regression analysis. However, in actual practice the recognition of the complexity of the sources of human behavior leads the users of mental tests to obtain a complex pattern of scores covering a broad spectrum of human performance measures. Again, a number of investigations have been carried out which demonstrate that decisions based on purely statistical procedures are more accurate on the average than clinically based decisions (for example, reference 2). However, in those cases where considerable weight is placed on accurate placement and selection of individuals, it is typical to make the final judgment clinically using the expert opinion of people familiar with the requirements of the organization and also with the properties of measuring instruments. Perhaps the most common instance of this is the college admissions selection process.

## BACKGROUND

The present project is directed to the exploration of the procedures used by psychologists in evaluating a person's performance on a battery of psychological tests. A central strategy in the research is to develop computer programs, based on interviews with psychologists, which will produce written evaluations of test profiles comparable to those produced by psychologists for the same profiles. It is reasonable to have the computer produce written text as output if it is assumed that the empirical referent of the psychological processes being investigated is the verbal productions of a psychologist who is interpreting a profile. Let me stress that the intention is to simulate the behavior of the clinical psychologist by incorporating his theoretical ideas and his practical experiences, in so far as he is able to express them, into a computer program. It is *not* the intention to simulate the behavior of the persons taking the tests.

The source of data for the analysis was the personnel files of a large organization whose needs include people with a wide variety of skills at a highly professional level. In view of these needs, an extensive testing and selection program is conducted to screen incoming applicants. Data collected from the applicants include a comprehensive biographical inventory, several aptitude and ability measures, a personality test providing seven different temperament scales, a work attitudes inventory which provides 14 different personality and attitude measures, an interest inventory and a written essay examination. This material is reviewed by one of the staff psychologists who produces a summary paragraph which is the basis for further action on each applicant.

Data which included 77 scores as well as the summary paragraphs based on these scores were obtained from the files for approximately 200 applicants. This material together with information obtained in interviews with the psychologists provided the basic data for the analysis. It should be noted that no attempt was made to process the written essay or the biographical material.

In the first stage of the analysis, we attempted to classify the content of the paragraphs. This was done by first coding the paragraph content into numerical form and subsequently converting the numerical form back into written text by means of a simple computer program. The discrepancies between the input and output verbal texts provided an effective source of information for improving the coding. Ultimately 65 classes of sentences were evolved which covered about 90 percent of the information in these paragraphs. Two general types of sentence classes were observed. Simple sentences, which reflected a simple translation of a person's score on one or more tests, predominated. For example, "He has above average general ability" or "He has very poor general verbal ability" were two instances of a class of simple sentences. Compound sentences were observed as well, although less frequently, and these in general reflected a contrast between two or more sets of scores on a profile. An example of this type was "His performance on a test of verbal fluency is very good although he has very poor general verbal ability."

Having developed a reasonable classification scheme for the information content of these summary paragraphs, we were then able to effectively discuss with a psychologist who had written a paragraph precisely what rules he felt he had been using when he had produced the paragraph. The sorts of rules cited were of the following. In the profile of scores we are given a score on a reading comprehension test and a score on a vocabulary test, each with values in the range of zero to nine. If the average of the vocabulary score and the comprehension score is two or less or six or more, then produce the sentence "He had 'X' general verbal ability", where 'X' takes a value from 'very poor' to 'very superior'

depending on the value of the average score. This is one member of a broad class of decision rules, leading in general to simple sentences based on weighted linear combinations of scores. It was convenient to subject the value thus derived to one of a small number of arbitrary transformations to produce a final value, in order to handle the rather common occurrence in which a sentence was not to be produced if a subject had a score in the middle range of values. A second class of decision rules was based on "Boolean" combinations of scores. For example, if a score in the range of six to nine on a measure of social responsibility was obtained, *and* a score in the range of five to seven on a measure of interest in supervisory tasks was obtained, then the sentence "He appears to be willing to assume supervisory responsibility" was produced. It is important to note that the psychologists considered averaging and "Boolean" combination as different, in their treatment of the data.

It was observed that compound sentences were typically produced by decision rules which operated on values derived for the production of simple sentences. For example, given a score on a test which is assumed to be a measure of verbal fluency, and another score which is the average of the scores on the verbal comprehension and vocabulary tests, one or both of the simple sentences based on these tests might be produced. However, if a difference of three or more between the score on the fluency test and the average of the scores on the comprehension and vocabulary tests was obtained, the separate sentences were not produced and the following compound sentence was produced: "His performance on a test of verbal fluency is 'X', although he has 'Y' general verbal ability." 'X' and 'Y' take values from 'very poor' to 'very superior' depending on the values computed separately for the component simple sentence.

In the next stage of the analysis we proceeded to develop a set of programs representing these "decision rules" and these routines were then incorporated into the earlier program which had reproduced the paragraphs. We then processed profiles for the subjects whose paragraphs had previously been encoded and reproduced by the first system, but in this phase we had the computer produce paragraphs directly from subjects' profiles, using the rules which were evolved on the basis of interviews with the psychologists. Again we looked at the discrepancies. Again, a reasonably good correspondence between the computer-produced output and the psychologist-produced output was observed and indeed in a number of cases discrepancies were resolved in favor of computer-produced output. Some examples of output at this stage may be seen in Figs. 7c, 8c and 9c.

At this point an interesting fact began to emerge. Statements made by psychologists, which appeared to be the most significant for diagnostic purposes, typically tended to be based upon rather complex sets of decision rules. These more complex decision rules could usually be represented as combinations of many different simple rules, although they were customarily not perceived this way by the psychologists. In some cases it was possible to resolve the complex decision process into a tree-like network of decisions which were made sequentially. In other cases it seemed more appropriate to consider the complex decision rule as consisting of a hierarchy of subroutines nested within one another. An additional complication also became apparent in that it seemed more and more important to be able to generate more complex linguistic structures, i.e. such things as the conditional inclusion of phrases in compound sentences.

## THE LANGUAGE SYSTEM

### *Introduction*

After considering the additional sentence generating operations needed, along with changes in the program logic required to make the system easier to use, it was decided to start from scratch and develop an entirely new system to facilitate this exploration of the processes used by the clinical psychologist in interpreting mental test data. This system, designated Protran II (for *pro*file *trans*later) is programmed in Fortran II for the 7094 and consists of some 58 routines, about 1000 lines of Fortran in all. A users' manual is available and in the following exposition we will present only enough detail to indicate how we can use the system to facilitate our research.

From the users' point of view the key element of this system is the "sentence generating operator." These operators serve as the fundamental units out of which a system of interpretation is built up by the user, and as such they have been designed to coincide as nearly as possible both in format and in

function with the conceptual elements that characteristically turn up in our interviews with psychologists. Externally a sentence generating operator has four main sections: a name; the name of a "combination rule"; a sentence section; and an argument section. Figure 1 gives an example of an operator in exactly the form in which it is punched into cards by the user for input to the system.

```
NVRA BAND HIS /REASONING ABILITY WITH
MATERIAL INVOLVING NON-VERBAL /OPER NVRA
1 ELEMENTS /IS /VAR(9/NVSK/5)*NVSK/6/9/CON
/0/2*
```

The name of the operator is 'NVRA'. Cols. 1-5 of all operator cards contain names.

The combination rule name is 'BAND' (Boolean and). Cols. 7-11 of first card only.

The sentence section is 'HIS /REASONING ABILITY WITH MATERIAL INVOLVING NON-VERBAL /ELEMENTS /IS /VAR (9/NVSK/5)'. Starts after col. 12 and extends to first asterisk.

The arguments of the combination rule BAND are 'NVSK/6/9/CON/O/2'. Starts after the first asterisk and extends to the second asterisk.

Segments are delimited by slashes.

Continuation numbers are located in col. 6.

Operator cards are identified by 'OPER' in cols. 73-76.

The segment 'VAR(9/NVSK/5)' is used by the system to find an appropriate adverbial phrase such as 'very superior'.

Figure 1. The first two lines show the contents of the cards required for a typical sentence generating operator.

Let me make some general comments about the form and function of sentence generating operators. The system builds up a dictionary of phrases out of the sentence section of the operators as they are input to the system. A phrase is delimited by slashes and any phrase which has previously occurred in some previous operator will not be stored in this phrase dictionary a second time. A segment is considered to begin with the first nonblank character following the slash, comma or asterisk. Up to ten cards may be used for any one operator. Several of the combination rules allow an indefinite number of arguments (within space limitations). The total number of tests and operators may not exceed 1,000. Either test names or the names of operators may occur as arguments within any operator. Associated with each operator will be a score which, unless otherwise specified, will be the average of the scores of tests and operators named as arguments for the given operator. A point to be emphasized is that, since operators can be named as arguments of other operators, it is possible to build up complex hierarchies of decision rules.

A basic principle of operation of the system is that if the conditions required by the combination rule of an operator are met, then the operator is said to be 'satisfied', and in this event one or more of the following will occur.

1. The sentence will be printed as it stands;
2. The sentence will be printed as a phrase in the sentence of some other operator;
3. The value computed for the operator will be used as the value of an argument of some other operator.

### *Combination Rules*

The subroutines which represent the combination rules have been developed more or less as the need has arisen and some overlap in function will be apparent in the twelve routines so far developed. For example, the combination rule named NBOOL is identical in function to the combination rule named BOOR (Boolean or), if K, the criterion, is set equal to one. NBOOL is identical in function to BAND (Boolean and) if K is set equal to the number of arguments. However, the simpler rules have been maintained because of their mnemonic value and simpler format.

Figure 2 lists the available combination rules, giving the name and a brief description of the function of each.

1. Logical Rules
   (a) BAND Boolean and
   (b) BOOR Boolean or
   (c) NBOOL n or more 'ands'
2. Arithmetic Rules
   (a) EQWT equal weights
   (b) ARWT arbitrary weights
   (c) AGTRB a greater than b
   (d) KDIFR difference of k or more
3. Matching Rules
   (a) LTAG 1 or more tags
   (b) BSIGN Boolean signs
4. Miscellaneous Rules
   (a) SUPP suppress
   (b) NOOP no op

Figure 2. Combination rules presently available in PROTRAN II.

### *Sentence Generation*

Let me next discuss some of the characteristics of the sentence generation process. There are three

categories of segments allowed in the sentence section.

1. Literal verbal phrases such as "HIS/". These literal phrases are entered in a phrase dictionary if they have not been entered by a previous operator. There is some merit to segmenting phrases so as to minimize storage, but this must be weighted against convenience in terms of the verbal habits of the user.

1. (NO ARGUMENT FOUND)/
2. VERY POOR/
3. POOR/
4. BELOW AVERAGE/
5. AVERAGE/
6. GOOD/
7. VERY GOOD/
8. EXCELLENT/
9. LIKES/
10. DISLIKES/

. .
. .
. .

Figure 3. The variable dictionary VAR is a (200, 12) array. It is necessary to store an error message in the first row of all variable dictionaries. The first set of phrases is in rows 2-8.

2. References to variable dictionaries such as "VAR (9/NVSK/5)". There are four components. In this example the first, VAR, is the name of the appropriate variable dictionary. The second component, 9, specifies which row of VAR contains the first phrase of the desired set. The third component, NVSK, gives the name of the test or operator whose score is to be used as the entry to the first scale vector in order to determine which element of the desired set of phrases is to be printed. The fourth component, 5, specifies which scaling vector is to be used to transform the score used as the entry to the first scale vector. (In this example the score for operator NVSK will be transformed by the fifth scaling vector and the resulting value will be added to 9 to get the row of VAR containing the desired phrase).

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 1. | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 |
| 2. | 01 | 01 | 01 | 00 | 00 | 00 | 00 | 02 | 02 | 02 |
| 3. | 01 | 01 | 02 | 03 | 04 | 04 | 05 | 06 | 07 | 07 |

.
.
.

Figure 4. The first row shows the first scaling vector which is used to define the input score scale. If a score of 4.3 was converted by scaling vector 2, the output value would be zero. Up to 20 such scaling vectors may be used. Scale is a (20, 20) array.

3. Indirect references to other operators such as '$PVF/' or '-PVF'. When the processes which compose an output sentence for an operator encounter such a reference, they executed (pseudo) recursively on the named operator.* This makes it possible to develop compound sentences of considerable complexity. A simple example of this may be shown using the following set of four operators.The sentence produced by the operator ANCON would consist of the component sentences joined by 'although', for example, HIS ARITHMETIC REASONING ABILITY IS WELL ABOVE AVERAGE ALTHOUGH HE HAS LITTLE SKILL IN HANDLING NUMERICAL DE-

```
ARA EQWT HIS/ARITHMETIC REASONING
ABILITY /IS/VAR (2/AP/2)*2/AP*                    OPER
NUDE EQWT HE/HAS/VAR(15/NO/6)SKILL
IN HANDLING NUMERICAL DETAIL*                     OPER
NUDE 1 6/NO*                                      OPER
ANCON KDIFR $ARA/ALTHOUGH/$NUDE*
5/AP/NO*                                          OPER
SUPNV SUPP *ANCON/ARA/NUDE*                       OPER
```

The sentence produced by the operator ANCON would consist of the two component sentence joined by 'although'. For example—HIS ARITHMETIC REASONING ABILITY IS WELL ABOVE AVERAGE ALTHOUGH HE HAS LITTLE SKILL IN HANDLING NUMERICAL DETAIL —would be produced if a subject had a high score on AP and a score 5 points lower on NO. It may be noted that the operator SUPNV would suppress the printing of sentences from ARA and NUDE if ANCON were printed.

Figure 5. A set of four operators which show the use of '$' to indicate indirect referencing.

*In Fortran "recursions" are conveniently done with a reproducer. In the present case the "recursion" terminates with a diagnostic after two levels.

TAIL, would be produced if a subject had a high score on AP and a score five points lower on No. It may be noted that the operator SUPNV would suppress the printing of sentences from ARA and NUDE if ANCON were printed.

We note that '$' and '—' are used to indicate that an operator is being indirectly referenced. These marks have the following meaning. A '$' forces the inclusion conditional upon whether or not the named operator has been 'satisfied'. In the following discussion a number of examples are given of the way conditional inclusion may be used. It may be kept in mind that the format of the argument section of an operator depends on the combination rule of the operator, and this format is different for different combination rules.

The system, in analyzing the data for a subject vis-a-vis a particular operator, associates a "tag" with the operator by executing the combination rule on the arguments of the operator. If the subject's scores (i.e. the current values of the arguments) are such as to 'satisfy' the combination rule of the operator, the operator is satisfied. It is convenient to amplify this idea at this point. Consider the set of operators in Fig. 6a.

```
PACE BOOR A RATHER/VAR(39/PACE/14) PACED PERSON* QUICK/0/2/QUICK/7/9*

VIGOR BOOR VAR(41/VIGOR/14)/VIGOROUS PHYSICAL ACTIVITY* TPHYS/0/2/TPHYS/7/9*

VIPAC LTAG HE DESCRIBES HIMSELF AS /-PACE/ AND/-VIGOR*1/PACE/VIGOR*

SUPPV SUPP*VIPAC/PACE/VIGOR*
```

Figure 6a. A set of four operators which show the use of '—' to indicate conditional indirect referencing.

The operator named PACE has the combination rule BOOR (Boolean or). The arguments of the operator are QUICK/0/2/QUICK/7/9. This combination rule will be satisfied if a subject has a score of 0 to 2 on the item named QUICK *or* a score of 7 to 9 on the item named QUICK. If this occurs the operator named PACE will be satisfied. Otherwise it will be tagged with a -1. VIGOR is executed in a similar fashion. Finally VIPAC will be satisfied, resulting in a sentence being printed if its combination rule, LTAG, finds that one or more of its arguments PACE and VIGOR have been satisfied.

The various sentences that can be printed by VIPAC, depending on a subject's scores, are tabled in Fig. 6b (remembering that the adverbs such as QUICK, SLOW, ENJOYING, and DISLIKING are

| SCORES ON | | RESULTING SENTENCE |
|---|---|---|
| QUICK | TPHYS | |
| 0-2 | 3-6 | HE DESCRIBES HIMSELF AS A RATHER SLOW PACED PERSON. |
| 3-6 | 0-2 | HE DESCRIBES HIMSELF AS DISLIKING VIGOROUS PHYSICAL ACTIVITY. |
| 7-9 | 3-6 | HE DESCRIBES HIMSELF AS A RATHER FAST PACED PERSON. |
| 3-6 | 7-9 | HE DESCRIBES HIMSELF AS ENJOYING VIGOROUS PHYSICAL ACTIVITY. |
| 0-2 | 0-2 | HE DESCRIBES HIMSELF AS A RATHER SLOW PACED PERSON AND DISLIKING VIGOROUS PHYSICAL ACTIVITY. |
| 0-2 | 7-9 | HE DESCRIBES HIMSELF AS A RATHER SLOW PACED PERSON AND ENJOYING VIGOROUS PHYSICAL ACTIVITY. |
| . | . | . |
| . | . | . |
| . | . | . |
| 3-6 | 3-6 | (No sentence will be printed) |

Figure 6b. Various sentences produced by VIPAC.

taken from the array named VAR under control of the segments VAR (39/QUICK/14) and VAR (41/TPHYS/14)).

In the first example VIGOR is not satisfied since a score in the range 3-6 is encountered for TPHYS. Therefore, the phrase produced by -PACE in VIPAC is suppressed (along with the following AND). In the last example no sentence is printed since LTAG, the combination rule for VIPAC, did not find one or more of VIGOR and PACE satisfied. Also it may be noted that SUPPV suppressed the printing of the component operators PACE and VIGOR (of course they occur in VIPAC).

*Conditional Branching*

The operator name 'JUMP+' is reserved for a special function, that of branching conditionally. There is an array in the system which contains the names of all the tests followed by the names of all the operators. The system, in evaluating the operators vis-a-vis a particular subject's scores, processes the operators in sequence from first to last. However, if JUMP+ is encountered, and the combination rule is satisfied, the system will transfer to a specified operator and continue in sequence from that point. For example, the operator will cause the system to transfer to the

JUMP+ BAND LOSER* FM*/9/9/RV/1/
1/RC/1/1*

operator named 'LOSER' if the subject has a score of 9 on FM, and a score of 1 on both RV and RC. Otherwise it will continue in normal sequence.

The convention is that the segment to the left of the first '*' must contain the name of the operator to which control is to be transferred.

It must be emphasized that any operators which are named as arguments of other operators must have been executed themselves before being utilized. Therefore, a certain amount of caution must be exercised in designing conditionally executed sequences.

## DISCUSSION

Given this system, it was possible to build up within a few days a set of approximately 100 operators which constituted a very comprehensive system of analysis of the battery of tests with which we had started. The system is being used by psychologists with no direct experience with computer programming or the use of computers and has proven to be a highly flexible research tool. Figures 7, 8 and 9 show examples of paragraphs produced by this system, together with psychologist-produced paragraphs for the same data.

In the first case the correspondence is very good. In the second case the correspondence is poor. However, the psychologists agreed that the computer produced output was useful and accurate.

About two seconds of machine time are required

His performance on various ability measures ranged from poor to high average. His abstract reasoning ability is only average, his arithmetic reasoning ability is high average, and his skill in the rapid handling of numerical computations is average. His vocabulary level is poor, his comprehension of narrative material is fair, and a 30-minute essay was fair in quality. He demonstrates poor ability in making interpretations from data presented in chart and graph form. He has poor knowledge of contemporary world affairs. His interests appear strongest in areas which involve attention to details, face-to-face persuasion, and a variety of essentially 'business management' activities. He is a physically active person who appears eager in his willingness to adapt to a wide variety of job situations.

Figure 7a. Psychologist generated interpretation for subject A.

TEST RUN 3/29/65

COMPUTER GENERATED INTERPRETATION FOR SUBJECT A.

HE HAS RATHER POOR GENERAL VERBAL ABILITY.

HIS ARITHMETIC REASONING ABILITY IS AVERAGE AS IS HIS SKILL IN HANDLING NUMERICAL DETAIL.

HIS ABILITY TO INTERPRET GRAPHS IS MUCH WORSE THAN USUAL.

HE DISPLAYS LITTLE KNOWLEDGE OF CURRENT WORLD EVENTS.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN POSITIONS OF WELL DEFINED AUTHORITY OR STATUS.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN JOBS INVOLVING CLOSE ATTENTION TO DETAIL.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN POSITIONS INVOLVING FACE TO FACE PERSUASION AT THE VERBAL LEVEL.

HE APPEARS TO BE WILLING TO ASSUME SUPERVISORY RESPONSIBILITY.

HE EXPRESSES AN INTEREST IN A JOB WITH CONSIDERABLE VARIETY.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN MANAGERIAL AND ADMINISTRATIVE POSITIONS.

HE DESCRIBES HIMSELF AS ENJOYING AND PARTICIPATING IN VIGOROUS PHYSICAL ACTIVITY.

AN UNUSUALLY LARGE NUMBER OF LOW SCORES ON THE WORK ATTITUDES INVENTORY SUGGESTS THE OPERATION OF A RESPONSE SET, MAKING INTERPRETATION DIFFICULT.

HE APPEARS TO HAVE AN UNUSUAL LACK OF NEED FOR RECOGNITION OR REWARD.

Figure 7b. Computer generated output.

CODED DESCRIPTION OF SUBJECT A.

HE HAS A RELATIVELY POOR COMPREHENSION OF WRITTEN MATERIAL.

HE HAS A RELATIVELY POOR ABILITY TO WRITE AN ESSAY.

HE HAS AN AVERAGE ABSTRACT REASONING ABILITY.

HE HAS AN AVERAGE ARITHMETIC REASONING ABILITY.

HE HAS AN AVERAGE COMPUTATIONAL ABILITY.

HE HAS A POOR ABILITY TO INTERPRET GRAPHS.

HE HAS A POOR KNOWLEDGE OF CURRENT WORLD EVENTS.

HE HAS A POOR COMMAND OF VOCABULARY.

HE HAS A HIGH DEGREE OF INTEREST IN DETAIL WORK.

HE HAS A HIGH DEGREE OF INTEREST IN PERSUASION OF OTHERS.

HE HAS A HIGH DEGREE OF INTEREST IN PUBLIC, SOCIAL, AND OTHER SERVICES.

HE HAS A HIGH DEGREE OF INTEREST IN BUSINESS.

HE WOULD ADAPT VERY WELL TO A VARIETY OF POSITIONS.

HE IS PHYSICALLY ACTIVE.

Figure 7c. An early version of computer generated output.

His level of intellectual ability would indicate that he would have a fair amount of difficulty competing with other college graduates. This is especially so if comprehension of complex ideas and precise analysis of graphs and charts is required. He appears to be more in need of purpose, resourcefulness, and self-confidence than the average employee his age. While he would be more satisfied than most with a routine, procedural type of activity he does need an opportunity to move about and a situation where people talk to him; such as in a library or a mail room.

Figure 8a. Psychologist generated interpretation for subject B.

TEST RUN 3/29/65

COMPUTER GENERATED INTERPRETATION FOR SUBJECT B.

HE HAS RATHER POOR GENERAL VERBAL ABILITY. HIS ARITHMETIC REASONING ABILITY IS BELOW AVERAGE ALTHOUGH HE HAS A HIGH DEGREE OF SKILL IN HANDLING NUMERICAL DETAIL.

HIS ABILITY TO INTERPRET GRAPHS IS MUCH WORSE THAN USUAL.

HE DISPLAYS LITTLE KNOWLEDGE OF CURRENT WORLD EVENTS.

HIS SCORES ON TESTS OF ABILITY AND INTELLIGENCE TENDED TO VARY WIDELY.

HE APPEARS TO BE UNWILLING TO ASSUME SUPERVISORY RESPONSIBILITY.

HE DESCRIBES HIMSELF AS ENJOYING AND PARTICIPATING IN VIGOROUS PHYSICAL ACTIVITY.

HE DESCRIBES HIMSELF AS SOCIALLY RETIRING.

HIGH SCORES ON THE WAI SUGGEST THAT HE IS DEFINITELY NOT EAGER TO ACCEPT SUPERVISORY ASSIGNMENT, OR TO TOLERATE INEFFICIENT OR UNDESIRABLE WORK COMPANIONS, OR TO ACCEPT ASSIGNMENT REQUIRING INITIATIVE OR ADAPTABILITY.

LOW SCORES ON THE WAI SUGGEST THAT HE IS EXCEPTIONALLY EAGER TO ACCEPT ASSIGNMENT REQUIRING PHYSICAL STRENGTH, AND TO WORK UNDER SUPERVISION.

Figure 8b. Computer generated output.

CODED DESCRIPTION OF SUBJECT B.

HE HAS A RELATIVELY POOR LEVEL OF GENERAL ABILITY.

HE HAS A RELATIVELY POOR COMPREHENSION OF WRITTEN MATERIAL.

HE HAS A RELATIVELY POOR ABILITY TO INTERPRET GRAPHS.

HE HAS A STRONG NEED FOR A WELL STRUCTURED JOB.

HE HAS A STRONG NEED FOR CONTACT WITH PEOPLE ON JOB.

HE HAS LITTLE ORIGINALITY, RESOURCEFULNESS, AND CREATIVITY.

HE IS SELF ASSURED.

Figure 8c. An early version of computer generated output.

His comprehension of narrative material is excellent, his vocabulary level is average, and a 30-minute essay was fairly long and moderately well written. His abstract reasoning ability is excellent, his arithmetic reasoning ability is superior, and his skill in basic computations is average. He demonstrates above average ability in the interpretation of data presented in graph and chart form. He has a high average knowledge of contemporary world events. His measured interests are focused in technical and scientific areas. He prefers a job where there will be evidence of appreciation for his efforts. He likes a job which requires the use of mechanical skills and equipment.

Figure 9a. Psychologist generated output for subject C.

TEST RUN 3/29/65

COMPUTER GENERATED INTERPRETATION FOR SUBJECT C.

HE HAS ABOVE AVERAGE GENERAL VERBAL ABILITY.

HIS ARITHMETIC REASONING ABILITY IS WELL ABOVE AVERAGE.

HIS PERFORMANCE ON A TEST OF VERBAL FLUENCY IS ABOVE AVERAGE.

HIS ABSTRACT REASONING ABILITY IS SUPERIOR.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN JOBS INVOLVING TECHNICAL AND MECHANICAL TASKS.

HE HAS MEASURED INTERESTS LIKE THOSE OF PERSONS IN APPLIED SCIENCE.

HE DESCRIBES HIMSELF AS A RATHER SLOW PACED PERSON AND DISLIKING VIGOROUS PHYSICAL ACTIVITY.

HIGH SCORES ON THE WAI SUGGEST THAT HE IS DEFINITELY NOT EAGER TO ACCEPT A JOB INVOLVING SECURITY RESTRICTIONS.

LOW SCORES ON THE WAI SUGGEST THAT HE IS EXCEPTIONALLY EAGER TO ACCEPT HAZARDOUS ASSIGNMENTS, AND TO ACCEPT ASSIGNMENTS INVOLVING MECHANICS.

HE APPEARS TO HAVE AN UNUSUALLY STRONG NEED FOR RECOGNITION AND REWARD.

Figure 9b. Computer generated output.

CODED DESCRIPTION OF SUBJECT C.

HE HAS A VERY SUPERIOR COMPREHENSION OF WRITTEN MATERIAL.

HE HAS AN ABOVE AVERAGE ABILITY TO WRITE AN ESSAY.

HE HAS AN EXCELLENT GENERAL VERBAL ABILITY.

HE HAS A VERY SUPERIOR ABSTRACT REASONING ABILITY.

HE HAS AN EXCELLENT ARITHMETIC REASONING ABILITY.

HE HAS AN AVERAGE COMPUTATIONAL ABILITY.

HE HAS AN ABOVE AVERAGE ABILITY TO INTERPRET GRAPHS.

HE HAS AN AVERAGE KNOWLEDGE OF CURRENT WORLD EVENTS.

HE HAS AN AVERAGE COMMAND OF VOCABULARY.

HE HAS A HIGH DEGREE OF INTEREST IN TECHNICAL AND MECHANICAL TASKS.

HE HAS A HIGH DEGREE OF INTEREST IN THE SCIENCES.
HE HAS A STRONG DESIRE FOR APPRECIATION ON THE JOB.

Figure 9c. An early version of computer generated output.

to process each profile and develop a paragraph based on 100 operators.

Detailed analyses of the application of the system to a variety of situations will be presented in a paper now in preparation.[3] There are, however, some points of possible confusion regarding the problem of the validity of the productions resulting from such applications, which should be addressed in this paper. It was stated earlier that the fundamental strategy is to simulate the implicit theory underlying the behavior of the experienced clinical psychologist. I share Roger Shepard's position[4] that "each of us may already be carrying within us a kind of *implicit* theory that is capable of at least qualitative predictions of a far greater range of human behavior than any of the theories that so far have been formulated *explicitly*." While it is true that most people must be able to predict a range of human behavior in order to survive in society, we have found it a particularly useful heuristic to work with experts in the domain. The computer system which has been devised makes it possible for the clinician to easily develop representations of his theory in a form which has rather direct empirical consequences. The paragraphs produced by the sequence of operators put into the system by a clinical psychologist are quantitative only in the general sense of the term. However, they represent explicitly the implications of his theory in a form which is as useful as a quantitative form, for investigating problems of validity. In this context, then, the computer system is relevant to the issue of validity only to the extent that its availability encourages the clinical psychologist to carry out more detailed and complex scientific analyses of his problem domain.

We have, therefore, the observation that the structure of the computer system has nothing directly to do with the validity of theories which happen to be representable using the system. On the other hand, the system evolved out of considerable experience with clinical psychologists and detailed observation of the consequences of their thinking as they analyzed profiles of test scores. Certain "structural" characteristics of the system were designed to conform with "structural" characteristics of the thought processes which were deduced from observations. The fact that operators can occur as arguments within other operators makes it possible to build up complex sets of decision rules in a natural way. Moreover, the fact that operators can be indirectly referenced in the sentence sections of other operators makes it possible to build up commonly occurring linguistic structures readily. These characteristics are consistent with the psychologist's inclination to first consider the simple translation of quantitative data into verbal form. These simple sentences then become the conceptual elements out of which he builds up more complex interpretation schemes and linguistic structures.

We may, if we choose, direct our attention to the validity of these "structural" aspects of the system as representations of the thought processes of the experts who use the program, but this is an entirely different matter from the validity of the paragraphs produced by the system. I will not address this problem now, although I am convinced that the attempt to simulate the thinking of experts working with problems in their domain of specialization will provide valuable insights into the mechanisms of human thought.

The relationship of the current project to other projects directed to the automation of the interpretation of mental test data is a matter of some interest. (See, for example, references 5, 6 and 7.) The system of interpretation which was developed for the original battery of tests using PROTRAN II is certainly an example of the same kind of research as that cited. However, the development of a special

language system to facilitate this kind of research appears to be somewhat novel. It appears that, in most cases, the systems of interpretation such as those reported in the references could be represented in PROTRAN II with no difficulty. It is apparent of course that PROTRAN II provides a fixed system of interpretation with no provision for self-modification within the system. Such features, if and when they are incorporated into the system, will necessarily be based on much more detailed knowledge of human thought processes.

## ACKNOWLEDGMENTS

I am greatly in debt to Dr. M. J. McKee, who collaborated on much of the research and whose astute clinical insights were invaluable. Among the many others who assisted in many ways I must single out Mrs. Jocelyn Helm, who made numerous substantive suggestions and who spent long tedious hours analyzing paragraphs, tabulating data, keypunching, cooking.

## REFERENCES

1. P. Diederich, J. French, and S. Carlton, *Factors in the Judgment of Writing Ability*. ETS Research Bulletin RB 61-15, Educational Testing Service, Princeton, N. J. (1961).

2. P. Meehl, *Clinical vs. Statistical Prediction*. U. of Minn. Press, Minneapolis, 1954.

3. C. Helm and M. McKee, "Computer Simulation of a System of Interpretation for the California Personality Inventory" (in preparation).

4. R. Shepard, "Review of *Computers and Thought*," *Behavioral Science,* vol. 9, no. 1, pp. 57-65 (Jan. 1964).

5. W. Swenson, J. Pearson, and H. Rome, "Automation Techniques in Personality Assessment: A Fusion of Three Professions," *Proceedings of the Conference on Data Acquisition and Processing in Biology and Medicine, New York, 1962*. Pergamon Press, New York, 1963.

6. B. Kleinmuntz, "Personality Test Interpretation by Digital Computer," *Science,* no. 3553 (Feb. 1, 1963).

7. Z. Piotrowski, "Digital Computer Percept-analytic (Rohrschach) Diagnoses in Neuropsychiatric and Personality Description." Paper given at E.P.A. meetings, New York, April, 1963.

# HUMAN DECISION MAKING UNDER UNCERTAINTY AND RISK: COMPUTER-BASED EXPERIMENTS AND A HEURISTIC SIMULATION PROGRAM*

Nicholas V. Findler
*University of Kentucky*

## INTRODUCTION

Every organism of higher order has to make decisions of varying importance regularly and frequently in order to survive and to survive efficiently. While the outcome of decision making has been studied extensively by a wide range of different disciplines, the decision making processes themselves have been neglected in comparison. Emphasis has been placed on the normative aspects of human behavior, i.e., how a rational† person or a group of rational persons *ought* to behave, as distinct from descriptive theories which are to explain and predict *actual* human behavior. An overwhelming majority of techniques and methods of attack in operations research, management science, industrial mathematics, etc. could be given the label: Normative Decision Theory. The students of mathematical psychology, on the other hand, beginning probably with Lady Lovelace, Bernoulli and Laplace, have been concerned with the behavioral aspects of decision making, i.e., what is being done in certain situations, and why.[1]

In recent years another approach to understanding human behavior has emerged through the pioneering work of Newell, Simon and their followers.‡ The Complex Information Processing method (CIP) dissects, for example, problem solving activity into elementary building blocks, such as different phases of planning, building of subgoals, utilization of certain heuristics, etc. These information processes and an executive routine are programmed for a digital computer. Comparing the trace of the program with the verbal reports and protocols elicited from subjects may indicate the level of completeness of the theory.

*This terminology is not quite uniform. Most authors in statistical decision theory and economics have adopted the convention that in a situation of *uncertainty* there is no a priori information about the system at hand, consequently probability distributions of various alternative outcomes cannot be objectively specified. In cases involving *risk*, however, it is assumed that, relying on prior events of identical character, these frequency distributions are available. In the present work we have not distinguished between these two cases and described the psychological state of the subjects as that under uncertainty. The element of risk is represented by the subjects' financial involvement in the outcome of the experiments. The adopted terminology and the distinction made here may be rather useful with experiments which aim at discovering correlation between subjects' behavior and varying payoff matrices while the stochastic components of the environment stay stationary.

The work reported here was done at Carnegie Institute of Technology and was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense: Contract SD-146.

†A "rational" person is (a) completely informed, (b) infinitely sensitive, and (c) can formulate his problem in an optimum manner.

‡The basic philosophy of this approach has been described in detail, e.g., in reference 2.

Simon and Newell[3] make the following three underlying propositions in this regard:

(1) A science of information processing can be constructed that is substantially independent of the specific properties of particular information-processing mechanisms [meaning, for example, *which* particular language is used on *which* particular machine—N.V.F.].
(2) Human thinking can be explained in information-processing terms without waiting for a theory of the underlying neurological mechanisms.
(3) Information-processing theories of human thinking can be formulated in computer programming languages, and can be tested by simulating the predicted behavior with computers.

Special problem-oriented list-processing languages, such as IPL-V, LISP, COMIT, etc., have been developed to enhance the symbol-manipulating capability of computers. Projects ranging from game-playing[4] through theorem-proving[5-7] and concept learning[8] to verbal learning[9] have proved that it is possible to find a *sufficient* theory, embedded in a computer program, that gives satisfactory explanation for different cognitive activities. Perhaps the most significant, but as yet incomplete, achievement is the discovery of universal features in human problem solving, that are independent of the particular task at hand.[10] Continuing this line of reasoning, it remains to be seen how a successful chess playing program can make use of its skill in learning, e.g., bridge, or how a Russian-into-English mechanical translator would be in a superior position in adapting itself to, say, French-into-English translating, as compared with a linguistically naive machine.

Information processing models exist for certain decision making tasks, such as Feldman's program,[11] which simulates human behavior in the binary choice situation and Clarkson's program[12] which makes trust fund investments.

The present work aims at a particular type of decision making. The project has two phases. First, a series of computer based experiments were designed and carried out. In these, human subjects were placed in quantitative task environments and their verbal behavior was observed and recorded after they had been asked to think aloud. The first part of the paper lists the research objectives, outlines the task environment and discusses the "design principles" resulting in the particular organization of the experiments.

The other phase of the project, dealt with in the second half of the paper, describes the subjects' behavior and attempts to establish a theory for the above behavior. Following the lines of the CIP approach, we formalized in terms of flow charts the subjects' internal representation of the environment, and the rules and methods they have employed in their organized, problem-oriented activity. The theory was then tested. The particular mechanisms, the elementary information processes, specified by the flow charts can be either hand simulated or, in view of their highly complex nature, programmed for a digital computer and the subjects' behavior compared with the computer performance. It has been found that the theory under discussion not only is a satisfactory explanation of *average* human behavior, which achievement is the usual goal of most endeavors in psychology,[13] but can also describe certain characteristic features of individual behavior. In order to be able to do so we have departed to some extent from the totally deterministic viewpoint of the logic theorist or the general problem solver and have incorporated certain stochastic components of limited importance in the framework of this theory to bridge over the gap of our ignorance about intraperson inconsistencies and interperson differences. This fact and the manner in which an individual is initially characterized for the program (cf. executive program) render the theory solely phenomenological. This, however, has been the contention of researchers at this level of explanation.[14]

## THE RESEARCH OBJECTIVES

The project has three basic research objectives of methodological and theoretical implications:

1. To develop techniques for organizing complex decision making areas in terms of their internal structure so as to make subjects exhibit specific aspects of behavior.
2. To test the applicability to control tasks of the CIP method and to see the limits of respective theories embodied in the form of computer programs.
3. To further our understanding on human decision making under uncertainty and risk.

The following research objectives of secondary importance also seem to be appropriate:

4. To assess the capability of man confronted with a complex quantitative problem situation;
5. To describe the cognitive and affective behavior of humans trying to gain understanding of partially confounded causal relationships;
6. To explore the effects on search behavior of task environments of different internal structure;
7. To investigate the extent and, possibly, the motivational background of individual differences in decision making behavior;
8. To study the efficacy of different subjective representational languages in problem solving;
9. To identify patterns of response to 'regular' and 'unexpected' changes in the environment; and so on.

## THE ORGANIZATION OF THE EXPERIMENTS

The psychological study of decision making is often handicapped either by the restricted task environment or by the significant but irrelevant factors possibly affecting the outcome of the experiments which are carried out under "nonlaboratory", i.e., not fully controlled, conditions. These difficulties are hoped to have been at least partially overcome by the simulated realism of computer-based experimentation.

Computer-based experiments may aid at man-machine relationships per se or use different modes of interaction as flexible tools to study the behavior of humans in a variety of well-determined environments. The present work was intended to cover both areas with a definite emphasis on the latter.

The experimental setup was as follows:

The subject was situated in front of a computer console which represented his input/output medium to a complex continuous technological process. The nature of and the laws regulating the process were not known to the subject. At his disposal there was a set of control variables, $\{x\} = x_1, x_2, \ldots x_m$; further he could observe a set of state variables, $\{y\} = y_1, y_2, \ldots . y_n$. Each of these state variables was associated with a cost variable through a cost function, $c_i = f_i(y_i)$, in a simple and well-defined way. (This relationship was, however, not given explicitly to the subject.) The task of the subject was to run the system at minimum overall cost, $\sum_{(i)} c_i \rightarrow \text{Min!}$, over some period of time. (See Fig. 1)

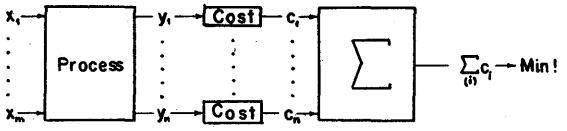



Figure 1. Block diagram of the organization of the experiments.

After this had been explained to the subject, he sat down to the computer console together with the experimenter, who typed in the loading instructions (to transfer the program from a magnetic tape to the magnetic core memory), a number specifying a task environment, and another number that determines the initial frequency of outputs. Then the subject named a set of $\{x\}$ values. Following this first input, at regular intervals, regardless of whether he changed the above values or not, he received an output consisting of the $\{y\}$ and $\{c\}$ values, and also $C = \sum_{(i)} c_i$. Due to the fact that the $\{y\} = \pm(\{x\})$ relationship is a combination of a set of mathematical functions and different kinds of noise, of which we shall speak shortly, the output values varied even if there was no change in the input values.

It was thought that interference between the subject's verbal behavior and decision making processes could be best avoided if there was no time pressure imposed on the subject. He was, therefore, allowed to increase or decrease the output frequency within reasonable limits. (For instance, the same subject asked for an output every 5 seconds when he was concerned with the noise components, and every 60 seconds when he wanted to concentrate on "structural information.") The median time interval between two subsequent outputs was 20 seconds.

Initially, there was a "free" period of search during which the subject was not concerned financially with the outcome. Five minutes' learning of the task proved ample with all our subjects. Subsequent to this the concept of risk was introduced. At this time a certain amount of money, that was considered to have been initially given to the subject, was involved. (The subjects were, however, told that they would not lose their own money.) During this latter period, which usually lasted 90-120 minutes,

the subjects were asked to think aloud. Their verbal behavior was tape-recorded and the transcripts of these protocols, together with the computer input/output, constitute the experimental data.

From a utilitarian point of view the advantages of this setup are manifold: the experimenter can vary the internal structure of the "black box" process and the cost functions at will to suit his particular needs; information-processing errors are virtually excluded; the quantitative nature of the environment renders the stimuli and the responses direct and unambiguous; experimental replications are easy and inexpensive to perform. Further, a whole gamut of experiments can be conducted, and are in fact planned for future studies, on game learning, game playing, coalition formation if certain subsets of control variables are assigned to different players. Experience can also be gained on how human operators adapt themselves to different real-life control tasks, etc.

## PRINCIPLES OF THE PROCESS DESIGN

The experimenter in psychology naturally would like his subjects to exhibit as wide a spectrum of relevant behavior as the physical and conceptual limitations of the experiment allow. He also wants to see what correlations there are between certain changes in the task enrivonment and the subject's behavior so that his previous goal can be achieved.

Unfortunately there is nothing like the well-formulated statistical theory of experimental design to be applied to meet these problems. The designer has to resort to "analog computations," i.e., he either imagines himself into the subjects' role or tries out different tentative tasks with subjects to see whether their gross behavior in fact tends to be of the type he had anticipated. As we have mentioned before, the process is represented by a set of mathematical relations with some noise superimposed upon it. Unless we want the subject to come to a high degree of confusion, the average noise level should apparently be adjusted to be about 1/10—1/20 of the average state variable level. The periodicity of noise should also be low enough not to be noticed. Instead of pseudo-random numbers we have used algebraic type noise generators, examples of which are as follows:

1. $Ai \pmod{B}$, where $A$ and $B$ are relative primes and their magnitude is roughly determined by the required level and tolerated periodicity of noise, respectively; $i$ is the number of times the subject has given input values.
2. $C \sin k\frac{\pi}{D}$, where $C$ and $D$ are constants the values of which are determined again by the required level and tolerated periodicity of noise, respectively; $k$ is the number of times the subject has received outputs.
3. $E\,\delta_{k,\,0(\mathrm{mod}\ F)}$, where $E$ and $F$ are constants the values of which are determined again by the required level and tolerated periodicity of noise, and $\delta$ is Kronecker's delta function. This type of noise is represented by a sudden peak of size $E$ every time $k$ is a multiple of $F$.
4. Some simple functions of $|\Delta x_l^{(i)}| = |x_l^{(i)} - x_l^{(i-1)}|$, i.e., of the absolute value of the change in the $l$th control variable between the $(i-1)$th and $i$th input (step size).
5. Some simple functions of $|x_j^{(i)} - x_l^{(i)}|$, i.e., of the absolute value of the difference between two control variables input concurrently, and different variations and combinations of all the above types.

Types (1) and (2) provide "background noise," with otherwise no specific role. Type (3) can be used in studying response patterns to sudden "unexpected" changes as distinct from "regular" fluctuations. The last two types of noise can be used for penalizing or rewarding the subject if he is making too big or too small steps or if he deviates from certain regions in the control variable space. Type (4) is equivalent to a short term memory in the system, while type (5) represents a reasonably simple interaction between two control variables.

We can provide the subject with state variables of high variability in the low-cost region or with variables of low variability in the high-cost region. (He can achieve subgoals in comprehension at a certain cost or remain partially in the dark inexpensively.) This could induce the subject to make a distinction between short-term and long-term objectives, which tendency may be important if the payoff is effected according to time-average achievements. Sophisticated persons sometimes would like to suboptimize detached parts of the system even at a certain cost. This can be realized with carefully thought out interaction terms. Other task environments can help us find out the upper limit of the

number of state variables the sum of which is approximately constant, regardless of the control variable levels, and the subject can still detect this fact within some reasonable time. Or, ascertain how long it takes a subject to find out that certain state variables are purely noise functions, which control variables interact, what the most reasonable search behavior is in terms of step size, etc.

In the course of setting up the experiment we found a few golden rules of thumb, following which a fairly systematic approach developed and assumed much of the role of a fool-proof theory of "experimental design." These are: it is good to have a few but not too many local minima; the absolute minimum should not be either in the middle or at the limits of the control variable ranges; interaction between control variables should be noticeable but not predominant. In general, it is advantageous to limit the range of all control variables identically. This range should be big enough to allow sufficient exploration but not too big so that subjects can examine and possibly even memorize important features of the whole range. In this study $0 \leq x \leq 50$ and $0 < y < 1000$ for all $x$'s and $y$'s.

In order to avoid the possibility of the subject's drawing sketches of two or three-dimensional response patterns on the one hand, and overcomplicated multidimensional searches on the other, task environments with three control and three state variables were uniformly selected.

For the sake of simplicity, in this series of experiments linear cost functions of the type $c_i = \alpha_i + \beta_i y_i$ were used.

Four exemplary task environments are described in Appendix I.

## THE SUBJECTS' BEHAVIOR

In view of the relative complexity of the task environment one would not expect every subject to behave identically, nor even the same person to take identical courses of action in the same situation at different times. The crucial question that has to be answered by students in behavioral sciences is whether there are significant characteristics of demonstrable generality that are common either for everybody or, at least, for sizable groups of human beings. If we are able to discover such characteristics of all the major facets of human behavior then a causal psychological theory of great explicatory and predictive power is, figuratively speaking, just "around the corner." As has been mentioned before, the solving of fairly well-defined problems by humans is a type of activity that does reveal certain general characteristics and the General Problem Solver, which simulates human problem solving on a computer, is a successful vehicle for its theory.

A theory of decision making can claim a higher degree of validity if it incorporates, besides cognitive processes, also some phenomena of affective behavior, such as aspiration, interest, curiosity, etc.[15] Many, although far from all, subjects can be induced to express in words their motives, even the ones they, themselves, regard as childish or nonsensical.* As a result of our continuous effort, subjects did stay fairly communicative. Consequently, a detailed analysis of the verbal reports, the extraction of essential elements and the systematic ordering and classification of the motivated actions were not negligibly small tasks.

So far 16 subjects' behavior has been studied. Some characteristics of the findings are summarized in Appendix II. Appendix III contains fairly detailed excerpts from a specimen protocol. Although a sample of this size would not allow us to draw minute conclusions of high statistical significance, a great amount of information was collected on a certain kind of human behavior. It is believed that the following, nonquantitative discussion, relying on experimental evidence, represents facts of reasonable generality.

### *The Initial Search*

The initial search behavior of subjects can be divided into two broad categories. Some of them tend to explore the task domain, i.e., the range of control variables, in an allegedly 'random' fashion. They try to generate as many different combinations of numerical values as possible. However, these attempts obviously reveal some sort of basic pattern, which the subjects, behaps unconsciously, use as a guide in the search. The fundamental human discomfort in experiencing or producing irregularity can be easily detected in these cases. The net result of this type of search is not much different from that of the other category of subjects who take pride in being

*As one subject (of what sex?) said: "I can see small particles swirling around in a big blue room, hitting each other and the walls. The faster they fly the higher values these $C$'s take."

"systematic." The search for structure, so inherent in human behavior, is modulated by preformulated ideas, such as

"Let me first see all the corners," or
"I'll explore the center thoroughly," or
"I want to go along this line at first," or
"Everything is somewhat symmetric," etc.

Subjects of varying backgound and apparent capabilities have shown such Einstellung effects. As Luchins[16] in a different context has noticed, persons of superior intelligence and education develop habituation effects to as great as or even greater degree than do persons of low educational level and IQ. As possible motives he lists for example the educated man's desire to generalize and also his contempt of childish or ostensibly repetitive tasks. In contrast with Luchins' findings, our subjects did not exhibit a mechanistic blind attitude toward the problem for any significant length of time, particularly not so after the initial search just described. The information feedback from the outcome of individual trials seemed to have an effective influence on the search behavior.

The initial search goes on with both types of subjects until they form the first hypothesis about the organization of the task. This event can happen quite suddenly, "out of the blue," or may be preceded by a long and thoughtful meditation. The subject "cannot help but discover it" or else he may force himself to visualize some sort of model for his problem. As one would expect, not only the type but also the length of the initial search varies from subject to subject within certain limits.

*The Hypotheses on, or the Internal Representation of, the Task Environment.*

The subjects' approach to the task varied considerably. Each of them understood the problem apparently equally well but interpreted it rather differently. The differences in personality, educational background and, maybe, temporary disposition resulted in somewhat different sets of "intellectual tools." We shall call this set of tools a subject's language of representation and the model built with its help the mental image of the task environment.

Before we discuss our findings in this regard, let us digress briefly and investigate the basis and implications of the above-mentioned concepts.

Man's information-handling capability is, in general, enhanced by thinking in terms of symbolized concepts whether these be elements of a spoken language, pictographs, logical constructs or mathematical relationships. In meeting man's changing needs in problem solving and decision making, sophisticated and powerful methods of attack can coexist with hunches, "intuitions," "insight," *déjà vu* processes. An important part of the learning process consists of improving and replacing conceptual hypotheses, which are formed after processing the received and self-generated information complex. These hypotheses are mentally combined and represent the image of the outside world (Weltanschauung). The combination of hypotheses is based on elementary conscious processes, mentioned before, such as remembering, comparing, recognizing, abstracting, logically interpolating and extrapolating, etc.

Man's problem-solving methodology starts from and depends on the language of representation he has formed to comprehend the environment. Pattern perception, verbal learning and concept formation are examples of the interaction between the language or representation and the resulting cognitive behavior.

In trying to perceive the relations of the stimulus environment, humans also try to relate their own behavior to the variations in it.[17] This fundamental tendency connects the so-called internal and the external structures, and renders the environment "meaningful." If these two structures are fairly homomorphic with a particular language of representation and with a particular task environment, the building of a mental image will be comparatively easy and successful. Similarly, with a problem-oriented computer language the data representations and the information processing instructions are such that we can solve with it a family of problems efficiently. A good problem solver finds good languages of representation, just like a good (multilingual) programmer is able to select the best computer language for his task.

The origins of the hypotheses about the environment can only be partly explained by the Gestaltist argument that the behavior is an inevitable result of some stimulus configuration and that the stimulus forces a specific response. It is also true that responsive behavior relies on the individual's interpretation of the environment. Any new situation to a subject is not a confused, meaningless conglomera-

tion of sensory impressions to which he makes confused, meaningless, uncoordinated and unrelated responses. He brings to the new situation a whole history of experience which he is ready to apply. Each response of the subject we observe is not a *ding an sich* but a meaningful part of his total behavior, which has casual roots both outside *and* inside.

Consequently, the decision making behavior (a) is systematic and purposive (displaying an "if-then" character), (b) involves some degree of abstraction, and (c) does not depend entirely upon the external environment for its initiation and performance.

Let us now see the essential elements of the major types of representation the subjects have revealed.

*The "Mathematical" Language of Representation.* This is the most frequently occurring type (NB: this was also the "true" language of representation), which fact is probably due to the biased nature of our sample. Not every college student subject of ours has, however, had mathematical training at tertiary level. Some were undergraduates in psychology and in fine arts (cf Appendix II).

The state and control variables are thought to be connected by a set of mathematical relations, which are somehow perturbed by extraneous noise. The search behavior can be divided into two, rather distinct stages. First, the subject is only concerned with the overall cost, $\Sigma_{(i)} c_i = C$, and its relation to different levels of control variables. In specifying new points in the $\{x\}$ space he follows certain rules, which will be described in detail in the next section dealing with the simulation of the behavior. The subject discovers the presence of noise fairly soon and tries to overcome its disturbing effect by averaging operations. Intermediate successes and failures raise and lower his aspiration levels, respectively, and eventually cause him to enter the second stage.

The second stage is a more sophisticated continuation of the first one. The subject tries to discover relations between individual state and control variables. How he does this depends largely on the extent of his mathematical training. We were able to discover more or less distinct versions of hill-climbing, univariate, factorial and random methods. The most frequent approach followed a pattern similar to the one described as direct search by Hooke and Jeeves[18] and employed by, e.g., Flood.[19] This fact is quite reasonable, there being no universal analytical technique or computational scheme that will always work in case of highly irregular functions with ridges, temporary plateaus, saddle points, etc.; Hooke and Jeeves themselves incorporated in their method heuristic procedures that "do not guarantee . . . a correct solution . . ."

These optimizing procedures were seasoned with occasional jumps of 'sudden insight' at times when a new idea has occurred to the subject or when a response pattern started to resemble one already known to him. (This could be a *déjà vu* pattern in the first few minutes.)

The techniques associated with the mathematical language of representation were the most successful ones and the language itself was usually maintained throughout the session unless a sufficiently high level of frustration removed the subject from it into the "patternless" representation to be described later.

Finally, we wish to point out that all these human "on-line" optimization methods are inferior in accuracy and efficiency to the above-referenced techniques, which the subjects were in fact trying to simulate. Man's gross computational and data processing ability cannot match well-planned and well-performed calculations in these instances. On the other hand, due to the remarkable human flexibility, man excels in adapting himself to a novel problem by changing an old method or finding a new one, which feature is almost never incorporated in a ready-made computer library routine. A statistical justification of the above statements is beyond the scope of the present work.

*The "Analog" Language of Representation.* The subject in this case conceives of the process as being a real analog of some physical or chemical plant, and the control and state variables represent actual physical or chemical quantities.* While the broad behavior of the subjects with this hypothesis was found to be similar to the behavior of those who adopt the mathematical language of representation, the search behavior was more cautious, smaller steps were made and every sudden change in the process output was liable to impose a more conservative attitude on the subject. This fact may be related to the subjects' experience that in real life one often does not have control over all relevant factors and even if he does, the control may be only partial, indirect or inefficient. Some subjects believed that if

there had been a larger number of control variables the noise level could have been significantly reduced. Unexpected, i.e., larger than usually experienced, changes in the response pattern often led to revisions of plans and "let's start again around here" statements, as if some mechanical breakdown could be avoided by a more careful approach.

This hypothesis results in a more detailed and, on the average, less efficient search behavior than the mathematical language of representation does. Experimentalists, people interested in technological areas are the best specimens of this class, as distinct from the theoretically inclined representatives of the previous hypothesis. (See Appendix II.)

*The "Counteracting" Language of Representation.* The subject is convinced that the experimenter, directly or indirectly, is involved in the system, and the responses he gets as outputs are somehow related to his own behavior. This underlying assumption may prove to hinder the searching activity and result in a statement:

"Well, whatever I do you can counteract it," or

"Now you have upset my plans again."

The way in which counteracting takes place was thought to be from either outside, on the console (although subjects could well see that only input data, requests for changes in the output frequency and certain initial loading instructions were typed in) or from inside, through prearranged tables.

None of our subjects started out with this attitude and neither did they sustain it for long periods of time. Some, particularly the one who had not seen a computer before, were more likely to adopt this viewpoint than others, after a sudden dose of frustration. Every time the "counteracting" idea struck a subject, his aspiration level dropped, he lost his drive and became uninterested. These facts revealed themselves through the subsequent forcedly random search, which completely disregarded all the previous results, and also there were visible signs of the subject's losing interest.

We cannot of course speak in this instance about the efficiency of the language of representation, it is an episode-type event which contributes negative aspects of insecurity and uninterestedness to the search behavior.

*One subject, disregarding the fact that there were three independent variables instead of two, considered the environment as some terrain and the $C$ values representing the height of mountains, hills, and valleys. His "hill-climbing" was a tourist's excursion.

*The "Causal Network" Language of Representation.* The subject may build up the image of a casual network of some sort. The control and state variables are in direct interaction with each other and the upper limit of complexity of the causal connections is only determined by the subject's cognitive capabilities. The plans and strategies can become very complicated. The difficulties may discourage the subject and lower his level of aspiration. Intermediate successes would change this situation again. The net result of the search behavior is similar to the univariate method mentioned under the mathematical language of representation. The time-dependence of the background noise is considered to be due to some mechanical vibrations, and sudden peaks due to resonance effects.

In general, it can be said that a rather inefficient and unstable behavior originated from this hypothesis. It occurred rather rarely and could be deduced from recent educational experience. (See Appendix II.)

*The "Personifying" Language of Representation.* A language of representation, in some respects akin to the above mechanistic causal networks, can develop in the subject's trying to personify individual variables. He describes the "behavior" of the variables in anthropomorphic terms, such as:

> "Well, $y_2$ seems to run away from $y_1$," or "Now $y_3$ tends to approach $y_2$," or
>
> "You see, $y_1$, is not concerned with $x_1$, it has a mind of its own", or
>
> "Perhaps $y_1$ is the leader, maybe because it comes up first and then $y_2$ has to take some number because $y_1$ took one particular kind and then $y_3$ comes along and he doesn't have to . . . he can choose anyone that he wants," etc.

This attitude, however naive it appears, can be quite common among people with specific education backgrounds, such as drama students, or among people who spend much time in Nature. (These inferences, too, are based on small samples and their statistical validation is not possible.)

The search is rather haphazard and nonsystematic. The subject picks some issue of no apparent importance ("I wonder what happens if $x_1$ jumps from 0 to 50 and back, all the time") and sticks to it quite some time. The experiment seems to him a purposeless but enjoyable game void of risk and responsibility.

*The "Patternless" Language of Representation.* Finally, there can be subjects who are unable to construct any model of the process and believe that the output values, independently of their selecting the levels of the control variables, are "completely random" and do not represent any patternlike phenomenon. This mode of behavior often resolves itself into one of the above languages of representation and also it may happen that, after sufficient disillusionment about one, the subject "gives up" and resorts to this nihilistic viewpoint.

This language of representation, like the "counteracting" type, is only an episode with most subjects. It reduces their drive and interest to an extent that may depend on the individual, his expectations, and the length of time spent with this hypothesis.

The above-listed representations have occurred with varying frequency. The subjects occasionally replace one with another, sometimes without trying to explicitly justify the changeover, sometimes after obvious dissatisfaction over its failure. These hypotheses serve as frameworks for the subjects who want to find relevance and meaning in the structure being explored.* There was an interaction not only between the representation and the search behavior but also between certain features of the task environment and the adopted hypotheses. We have purposely selected task environments that might induce the subjects to exhibit particular types of associations. For example if the process functions were fairly simple, smooth and not highly periodic, the mathematical language of representation was to be expected. The "analog" representation required also some discontinuities, a small number of steep rises and falls, and moderate interactions between control variables. The "counteracting" hypothesis is more likely to occur if the sum of two or all the three cost functions remains approximately constant, regardless of the changing control variable values. (The subject observes no effect of his attempts.) The "casual network" representation was selected when the state variables were some functions of type (5) noise. (The interaction between control variables is rather obvious.) Similar relations should lead to the "personifying" language of representation. Finally, the "patternless' hypothesis is more often encountered if significant or exclusive noise terms appear in the cost functions.

*Compare the "strategies of decision making" as described in reference 20, or the TOTE units described in reference 21.

## THE SIMULATION PROGRAM

The aim of the simulation program is, in making use of the characteristic features of the verbal records of the subjects' rules of action, methods, procedures and strategies, to make the computer behave in a way that is essentially similar to the human behavior. This classical Turing's test demands that not only the desicions be the right ones but also the *reasons* that lead to the decisions be the right ones. With a successful simulation program we can investigate all the possible implications of the theory inexpensively and effortlessly by running many computer experiments in different task environments and with "different" subjects.

Flow charts have been constructed and hand-simulated for every language of representation with the exception of the "personifying" type. The executive routine, the initial search and the mathematical representation have reached the running program stage.

In the following we discuss some aspects of the program only. A more detailed discussion would exceed the scope of this paper.

As mentioned in the Introduction, the program is not a completely deterministic model, it also incorporates stochastic elements. The introduction of the concept of probability into the description of human cognition is considered here a quantitative means of getting around our quasi-ignorance about many minute causal effects. It is, for example, not essential to know which subject selected 8 and which selected 12 trial points during their initial search in the same task environment. It is sufficient to make certain that this number is not correlated in any obvious way with the subject's personality, if we may use such a broad term for the totality of known character descriptives, and that neither does it depend apparently on the outcome of the individual initial trials. We determine the range of this quantity (in this case between 6 and 15) and generate a random number within it for the extent of the simulated initial search.

In general, certain parameters are provided as input for the executive routine. These describe probability distribution functions of which samples are taken to serve as program switches (e.g. type of initial search) or event counters (e.g., extent of initial search).

In a similar fashion, the initial aspiration level and another reasonable-looking parameter, the level of cautiousness are determined. The respective lower bounds are specified for the executive routine, which then adds small random numbers to these to obtain the actual values.

The adjustment and testing of the aspiration level take place in the following way. The current aspiration level, at first equal to the initial aspiration level, is increased by one every time the program finds a new point with a better "total cost," i.e., with a value lower than the highest one in the group of best points so far.* Also, the current aspiration level is decreased by 1 after a cycle of 3-7 failures of finding a better point. (Compare this with the readiness of humans to strive for higher goals, as contrasted with the reluctance in regard to lowing ambitions.)

Whenever the absolute value of the difference between the initial and the current aspiration levels exceeds a certain constant, the program quits the first stage of the mathematical language of representation, in which the subject is only concerned with the total cost, and enters the second stage. In this, he tries to establish relations between individual control and state variables. This changeover may, therefore, be due either to having reached a certain level of success ("let's see what more could be done") or to some feeling of frustration ("I ought to try some more sophisticated approach now").

An "abstracting subroutine" makes the program disregard some process outputs between two subsequent input operations at certain times. On other occasions it calculates rough averages of process outputs over small localities of the $\{x\}$ space.

The executive routine provides a "public memory," which the different languages of representation can make use of. It also has monitoring and interrupting power at certain decision junctures. It may happen at one of these interruptions that the program of another representation takes over the control of the hierarchy of elementary information processes.

## DIRECTIONS OF FURTHER STUDIES

Firstly, there are several possible refinements within the present framework of the project. For example consideration is being given to providing the simulation program with a sort of "learn to learn" feature. In this a record of the relative successes of different representations could influence the hypothesis-selecting mechanism. A much more significant achievement would be if the character of individual task environments could be evaluated and used by the hypothesis-selecting mechanism to find the most efficient representation for the problem at hand. Also, with more experimental data, the probability distributions of the employed parameters would become less tentative.

A possible extension of the present work, which has employed linear and highly regular cost functions, would be to introduce discontinuous, saturated or hysteresis-type cost functions and compare search behaviors in systems with different cost structures while the process itself remains the same.

Another interesting comparison could be made between the decision making processes of two groups of engineering students. The same task environment would be used but one group of students would be told explicitly that the variables of the experiment represent actual, highly critical physical quantities, say $x_1$ is temperature, $x_2$ is pressure, etc.

A different area of research could be pursued on game playing, game learning, coalition formation, etc. with similar experimental apparatus if certain subsets of control variables are assigned to different players. The participants can be situated in separate rooms, in front of remote consoles.

## CONCLUSIONS

Computers have been found flexible and efficient tools for providing task environments in psychological experiments. With the development of multiconsole time-sharing systems, this endeavor is becoming economically quite feasible on large machines as well. The subjects' total behavior can be reasonably faithfully observed by combining the computer's data processing abilities, for the overt patterns of behavior, with tape-recording the verbalized thinking processes.

The CIP approach has proved to be able to give a causal description of an interesting and rather important facet of human behavior, decision making under uncertainty and risk. It is realized that the particular task environments employed may have had significant effects on the details of the behavior under study, but it is contended that the general as-

*One must not become confused by the directions of change: the aspiration level *increases* if a better, i.e., *lower*, $C$ value has been found, and vice versa.

sumptions and consequences of the theory are independent of the characteristics of different task environments. On the other hand, to force every aspect of a certain type of human behavior into the Procrustean bed of a running program is obviously farfetched. It is the intention of the present author to generalize this work to cover other phenomena of decision making while remaining as parsimonious as possible.

## ACKNOWLEDGMENTS

The author is indebted to Professor Allen Newell for suggesting this research area, and to him and Professor Herbert A. Simon for many stimulating discussions.

## REFERENCES

1. The interested reader's attention is drawn to the following, far from exhaustive, list of papers and books, and to the extensive bibliographies included in them:

A. G. Hart, "Risk, Uncertainty and the Unprofitability of Compounding Probabilities," *Studies in Mathematical Economics and Econometrics,* O. Lange, F. McIntyre and T. O. Yntema, eds., University of Chicago Press, Chicago, 1942.

A. Wald, *Statistical Decision Functions,* Wiley, New York, 1950.

W. Edwards, "The Theory of Decision Making," *Psych. Bull.* vol. 51, pp. 380-417 (1954).

R. M. Thrall, C. H. Coombs and R. L. Davis, eds., *Decision Processes,* Wiley, New York, 1954.

D. Blackwell and M. A. Girschik, *Theory of Games and Statistical Decision,* Wiley, New York, 1954.

P. Wasserman and F. S. Silander, "Decision Making, An Annotated Bibliography," Grad. School of Business and Public Administration, Cornell Univ., 1958.

R. D. Luce and H. Raiffa, *Games and Decisions, Introduction and Critical Survey,* Wiley, New York, 1958.

L. Weiss, *Statistical Decision Theory,* McGraw-Hill, New York, 1961.

R. E. Machol and P. Gray, eds., *Recent Developments in Information and Decision Processes,* Macmillan, New York, 1962.

2. A. Newell and H. A. Simon, "Computers in Psychology," *Handbook of Mathematical Psychology,* Luce, Bush and Galanter, eds., Wiley, New York, 1963, vol. 1.

3.——— and ———, "Information Processing in Computer and Man," CIP Paper No. 67, Carnegie Institute of Technology, (Mar. 31, 1964).

4. A. Newell, J. C. Shaw and H. A. Simon, "Chess-Playing Programs and Problem of Complexity," *IBM J. Res. and Dev.,* vol. 2, pp. 320-335 (1958).

5. ———, ——— and ———, "Empirical Explorations of the Logic Theory Machine," *Proc. WJCC,* Los Angeles, Calif., 1957, pp. 218-230.

6. H. Gelernter, "Realization of a Geometry-Theorem Proving Machine," *Proc. Internat. Conf. on Information Processing,* Paris, 1959, pp. 273-282.

7. J. Robinson, "Theorem-Proving on the Computer," *J ACM,* vol. 10, pp. 163-174 (1963).

8. E. B. Hunt and C. I. Hovland, "Programming a Model of Human Concept Formulation," *Proc. WJCC,* Los Angeles, Calif., 1961, pp. 145-155.

9. E. A. Feigenbaum, "The Simulation of Verbal Learning Behavior," *Computer on Thought,* Feigenbaum and Feldman, eds., McGraw-Hill, New York, 1963.

10. A. Newell and H. A. Simon, "GPS, A Program that Simulates Human Thought," *Lernende Automaten,* H. Billing, ed., Oldenbourg, Munich, 1961, pp. 109-124.

11. J. Feldman, "Simulation of Behavior in the Binary Choice Experiment," *Proc. WJCC,* Los Angeles, Calif., 1961, pp. 133-166.

12. C. P. E. Clarkson, "A Model of the Trust Investment Process," *Computer and Thought,* Feigenbaum and Feldman, eds., McGraw-Hill, New York, 1963.

13. See, e.g., C. L. Hull, *Principles of Behavior,* Appleton-Century-Crofts, New York, 1943, or R. R. Bush and W. K. Estes, *Studies in Mathematics Learning Theory,* Stanford University Press, Stanford, 1959.

14. See, e.g., H. A. Simon, "An Information Processing Theory of Intellectual Development," *Thought in the Young Child,* W. Kessen and C. Kuhlman, eds., (Monogr. Soc. Res. Child Dev., 27), pp. 137-161 (1962).

15. See e.g., H. A. Simon, "A Theory of Emotional Behavior," CIP Working Paper No. 55, Carnegie Institute of Technology (June 1, 1963).

16. A. S. Luchins, "Mechanization in Problem Solving: The Effect of Einstellung," *Psych. Monographs,* vol. 54, no. 6, whole no. 248, pp. 1-95 (1942).

17. W. R. Garner, *Uncertainty and Structure as Psychological Concepts,* Wiley, New York, 1962.

18. R. Hooke and T. A. Jeeves, " 'Direct Search' Solution of Numerical and Statistical Problems," *J ACM,* vol. 8, pp. 212-229 (1961).

19. M. M. Flood, "Stochastic Learning Theory Applied to Choice Experiments with Rats, Dogs and Men," *Behavioral Science,* vol. 7, pp. 289-314 (1962).

20. J. S. Bruner, J. J. Goodnow and G. A. Austin, *A Study of Thinking,* Wiley, New York, 1956.

21. G. A. Miller, E. Galanter and K. H. Pribram, *Plans and Structure of Behavior,* Holt, New York, 1960.

## APPENDIX I

## MATHEMATICAL DESCRIPTION OF THE TASK ENVIRONMENTS USED

In the following, four task environments are specified. The meaning of some symbols is given first.

$i$ = the number of times the subject has named input values;

$k$ = the number of times the subject has received output values;

$\delta_{d,b} = \begin{cases} 0 & \text{if } a \neq b \\ 1 & \text{if } a = b \end{cases}$ :Kronecker's delta function;

$a=b \pmod m$: $a$ is said to be congruent to $b$ modulo $m$, if $a-b=km$, where $a$, $b$, $k$, and $m$ are integers. We used the shorthand notation $b \pmod m$, representing the remainder of the division $b/m$;

$\Delta x_l = x_l^{(i)} - x_l^{(i-1)}$: is the difference between the values of control variable $x_l$ specified currently and on the previous occasion. In all other instances only the current values of variables are considered.

Task Environment 1

$$y_1 = 80 \sin \frac{x_3}{100}\pi - 60 \sin^3 \frac{x_3}{100}\pi + 10 + 5x_2 + 8i \pmod{21}$$

$$y_2 = 0.008\, x_1^3 - 0.6\, x_1^2 + 10\, x_1 + 60 + 8 \sin k$$

$$\frac{\pi}{4} + 10.\ \delta_{k,\,0 \pmod 7},$$

$$y_3 = 0.1\, x_2^2 - 1.8\, x_2 + 30 + 10\, x_1 + 5\,(|\Delta x_1| + |\Delta x_2| + |\Delta x_3|).$$

Task Environment 2

$$y_1 = 60 \sin \frac{x_1}{10}\pi + 0.04\, x_1^2 - 1.2\, x_1 + 70 + \frac{20}{|x_1 - x_2| + 1},$$

$$y_2 = 250 - 200\ \delta_{x_{21},\,9} + 20 \cos \frac{i}{5}\pi + 3k \pmod{16},$$

$$y_3 = 200 - 2\,(|x_1 - x_2| + |x_2 - x_3| + |x_3 - x_1|).$$

Task Environment 3

$$y_1 = 15 \sin \frac{i}{5}\pi + 3 \sin \frac{k}{2}\pi + 60,$$

$$y_2 = \frac{300}{x_3 + 2} + 4x_3,$$

$$y_3 = 258 - \frac{300}{x_3 + 2} - 4x_3.$$

Task Environment 4

$$y_1 = 150 \sin \frac{x_1}{10}\pi - 0.05x_1^2 + 6x_1 + 100 + \frac{1}{100}|x_2 - 8| \cdot |x_2 - 27| \cdot |x_2 - 36|,$$

$$y_2 = \frac{1}{300}|x_1 - 13| \cdot |x_2 - 41| \cdot |x_3 - 24|,$$

$$y_3 = 430 - y_1 - y_2 + 71\, i \pmod{52}.$$

## APPENDIX II

### Summary of Experimental Results

The following table contains the summarized results of experiments with 16 subjects. Although this number may appear rather small, the amount of information collected was considerable. The transcript of tape-recorded protocols exceeds 100 typewritten pages.

The Educational Background column of the table refers to the major subject of the person. The last column indicates a score (poor, far, good or excellent) that roughly reflects the quality of the subjects' search and the level of the minimum he was able to achieve by the end of the experiment.

Table 1. Summarized Results of the Experiments.

| Subject No. | Sex | Educational Background | Task Env. No. | Adopted Language(s) of Representation | Total Number of Trials/Task Env. | Score |
|---|---|---|---|---|---|---|
| 1 | M | Grad. stud. in Eng. | 2 and 3 | Analog | 48 + 33 | Good |
| 2 | M | Grad. stud. in math. | 1 and 2 | Mathematical | 59 + 47 | Exc. |
| 3 | F | Undergrad. in psych. | 1 and 3 | Mathematical and counteracting | 38 + 36 | Fair |
| 4 | M | Undergrad. in drama | 4 | Patternless | 39 | Poor |
| 5 | F | Undergrad. in psych. | 1 | Mathematical and patternless | 87 | Fair |
| 6 | F | Undergrad. in design | 2 | Personifying and patternless | 48 | Poor |
| 7 | F | Underg. in bus.&social | 3 and 4 | Analog | 41 + 54 | Poor |
| 8 | M | Undergrad. in physics | 2 | Causal network | 45 | Good |
| 9 | M | Undergrad. in chem. | 2 and 4 | Analog and patternless | 35 + 31 | Fair |
| 10 | F | Undergrad. in drama | 3 | Personifying | 33 | Poor |
| 11 | M | Grad. stud. in math. | 1 and 4 | Mathematical and patternless | 52 + 49 | Fair |
| 12 | F | Undergrad. in nat. sc. | 2 | Analog and patternless | 40 | Fair |
| 13 | M | Undergrad. in psych. | 1 | Mathematical | 56 | Fair |
| 14 | F | Undergrad. in nat. sc. | 3 | Causal network and counteracting | 68 | Poor |
| 15 | F | Undergrad. in psych. | 4 | Patternless | 37 | Poor |
| 16 | M | Undergrad. in math. | 4 | Mathematical | 64 | Exc. |

## *Appendix III*

## COMPARISON BETWEEN EXCERPTS FROM A REPRESENTATIVE PROTOCOL AND COMPUTER PERFORMANCE

In both instances, in the experiment and in its computer simulation, Task Environment 1 was used with the simplest cost function, in which $c_i \equiv y_i$. Editorial remarks are put in square brackets.

The subject's initial search here could be classified as systematic with slight random components. He adopted the Mathematical Language of Representation without any hesitation. In Stage (A) he followed a scheme fairly thoroughly but was not satisfied with the rate of improvement and entered Stage (B). In this, he employed the so-called univariate method which consists of changing only one control variable at a time. The optimization takes place in terms of single control variables and at the end an attempt is made to specify a global optimum.

* * *

*E*: Can we start now?

*S*: O.K., let's try around zero, say, 2, 3, and 4. [i.e., $x_1=2$, $x_2=3$, and $x_3=4$; the first output followed: $c_1=42$, $c_2=77$, $c_3=90$, $\Sigma c=209$.] Well, let us try another type . . . [interrupted by a new output: $c_1=42$, $c_2=83$, $c_3=90$, $\Sigma c=215$] . . . hmm, it's increased . . . O.K., let them be 45, 5, 4. [i.e., $x_1=45$, $x_2=5$, $x_3=4$; output $c_1=60$, $c_2=31$, $c_3=698$, $\Sigma c=789$.]

In describing the computer's actions, instead of articulated English sentences, we can only refer to brief statements, such as EXTRAPOLATION, INTER-

POLATION, ASPIRATION LEVEL NOW:, COUNTER NOW:, NEW POINT:, etc. These had been planted in the program to indicate at appropriate times what action the machine is taking. The following excerpts from the trace of the program [P] are approximately equivalent to the segments of human behavior at the left, almost paragraph by paragraph.

* * *

INITIAL SEARCH TYPE 1.
NEW POINT: XI= 3,X2=47,X3= 49.
OUTPUT: C1=273,C2=84,C3=196, SUM=553.
OUTPUT: C1=273,C2=90,C3=196, SUM=559.

[The noise appears here.]

NEW POINT: X1= 4,X2= 2,X3= 0.
OUTPUT: C1= 36,C2= 98,C3=541, SUM=675.
NEW POINT: X1= 5,X2= 48,X3= 3.
OUTPUT: C1=260,C2=102,C3=474, SUM=836.

Whoops, it's up . . . [interrupted by a new output: $c_1=60$, $c_2=29$, $c_3=698$, $\Sigma c=787$]. Now, let them be 4, 48, 48. [Output: $c_1=273$, $c_2=90$, $c_3=754$, $\Sigma c=1117$].

Oh, that's bad. 0, 50, 0. [Output: $c_1=271$, $c_2=52$, $c_3=460$, $\Sigma c=783$.]

Still no good, let's mix them further . . . [interrupted by output: $c_1=271$, $c_2=50$, $c_3=460$, $\Sigma c=781$] . . . Well, $x_1=45$, $x_2=48$, $x_3=4$. [Output: $c_1=278$, $c_2=27$, $c_3=879$, $\Sigma c=1184$.] This is the worst so far.

Now how about the center? Say, 20, 25 and 30. [Output: $c_1=173$, $c_2=83$, $c_3=617$, $\Sigma c=873$.]

. . .
. . .
. . .

*E*: How did you pick these values? Can you explain it now?

*S*: Well, I thought I would first try all these corners . . . a sort of local search at the two ends of every $x$ range, and also in its middle. This might give me some idea of how that sum behaves . . . It could, of course, go up and down in between . . . but that may not be important. I hope these functions [the hypothesis is formed!] do not oscillate too much . . .

. . .
. . .
. . .

OUTPUT: C1=260, C2=104, C3=474, SUM=838.
NEW POINT: X1= 24, X2= 27, X3= 22.
OUTPUT: C1=191, C2= 64, C3=589, SUM=844.

. . .
. . .
. . .

[P names nine points with coordinates either at the two ends of the ranges, $0 \leq x \leq 5$ and $45 \leq x \leq 50$, or in the middle, $20 \leq x \leq 30$.]

. . .
. . .
. . .
. . .
. . .

THE MATHEMATICAL LANGUAGE OF REPRESENTATION IS ACCEPTED.
INITIAL ASPIRATION LEVEL: 8.

. . .
. . .
. . .
. . .
. . .
. . .

*S*: Now I have . . . how many . . . four, five reasonable points. The sums here are no worse than say, 800. Let us be careful now . . . I want to give you a good $x_1$ . . . It's better if I don't care about these continuous outputs now . . . Well, when $x_1$ was around zero, the sum was about 210; when it was 28, we had almost 600. So, how about . . . how about, say, if $x_1$ equals 10 . . . that might hit the minimum . . . This kind of thing can give us, shall we say, 15 for $x_2$, and . . . well . . . I'm doing the same for $x_3$ . . . O.K., let it be 20. [Output: $c_1=134$, $c_2=108$, $c_3=736$, $\Sigma c=778$.] It didn't do much good . . . O.K., let us include this point as well . . .

. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .
. . .

SELECTED SUBSET OF POINTS:
X1= 3, X2=47, X3=49, SUM=553;
X1= 4, X2= 2, X3= 0, SUM=675;
X1= 4, X2= 3, X3=50, SUM =680;
X1=46, X2= 4, X3= 1, SUM=779.

[P selects four points out of the nine "non-noisy" ones, with the lowest total cost values.]

INTERPOLATION: X1=21;
[=3+(46–3). 553/(553+779)]
INTERPOLATION: X2=22;
[=2+(47–2). 675/(675+553)]
INTERPOLATION: X3=25.
[=0+(50–0). 675/(675+680)]
NEW POINT X1= 21, X2= 22, X3= 25.
OUTPUT: C1=130, C2= 97, C3=563, SUM=780.
OUTPUT: C1=130, C2=103, C3=563, SUM=786.
COUNTER NOW: 1.

[P fails to hit upon a sum better than at least 779, by interpolation. It is counting the number of failures.]

SELECTED SUBSET OF POINTS:
X1= 3, X2=47, X3=49, SUM=553;
X1= 4, X2= 2, X3= 0, SUM=675;
X1= 4, X2= 3, X3=50, SUM=680;
X1=46, X2= 4, X3= 1, SUM=779;
X1=21, X2=22, X3=25, SUM=780;
X1= 5, X2=48, X3= 5, SUM=804.

[P selects 6 points out of the 10 "non-noisy" ones so far, with the lowest total cost values.]

. . .
. . .
. . .

*S*: I just can't get a better point . . . This is silly. Why don't I move off? Suppose, we just try 35, 40, 45 for a change.

. . .
. . .
. . .
. . .
. . .

*S*: [After hitting a point with $\Sigma c=581$]. Here we are, that's reasonable . . . I see it now . . . This thing seems to move downwards, toward the smaller $x_2$'s. Let us choose . . . well, $x_1$ was all right, let it stay at 8, . . . $x_2$ has to be down . . . down . . . say, 15 and $x_3$ stays in the middle, 25.

. . .
. . .
. . .

*E*: What is wrong?

*S*: I just don't get that sum any better. I somehow feel, I have to do something about $c$'s . . . I mean, about $c_1$, $c_2$ and $c_3$ separately. Maybe I should minimize $c_1$ only, at first. Let $c_2$ and $c_3$ be whatever they are, and then come back to them . . . Yes, that's what I am going to do. Minimize them, one by one, and then get them together again. Where was $c_1$ the best?

COUNTER NOW: 5.
RANDOM BREAK-OUT.
ASPIRATION LEVEL NOW: 7.
NEW POINT: X1=17, X2=38, X3=9.

[After five failures the aspiration level drops and the next point is specified at random.]

. . .
. . .
. . .

EXTRAPOLATION: X1=0;
EXTRAPOLATION: X2=48;
INTERPOLATION: X3=23.
NEW POINT: X1=0, X2=48, X3=23.

. . .
. . .
. . .
. . .
. . .

ASPIRATION LEVEL NOW: 4.
ENTER STAGE (B).
POINTS WITH BES C1:
X1= 4, X2= 2, X3= 0, C1= 36;
X1=45, X2= 4, X3=46, C1= 59;
X1=32, X2=11, X3=17, C1=106;
X1=21, X2=22, X3=25, C1=130.

[The difference between the initial and current aspiration levels is four, which fact makes P quit Stage (A) and enter Stage (B). P selects four points with the lowest $c_1$ values so far.]

. . .
. . .
. . .

*S*: Well, $c_1$ depends on $x_2$ and $x_3$, and a little bit, maybe a little bit, on $x_1$. And it also varies with time . . . Yes, low values of $x_2$ are all right . . .

. . .
. . .
. . .
. . .
. . .

*S*: You see, this was easy, to minimize $c_2$ . . . I now know what it is like . . .

. . .
. . .
. . .
. . .

*S*: That $c_3$ is tough, I can't see much reason behind it. Somehow, when I don't move with the $x$'s, it's

smaller, and when I change them a little, it jumps up . . . It is a funny variable . . .

. . .
. . .
. . .
. . .
. . .
. . .
. . .

POINTS WITH BEST C1:
X1= 4, X2=2, X3=0, C1=36;
X1=18, X2=5, X3=2, C1=52;
X1=28, X2=4, X=6, C1=57.

[A further search produces these three best points with regard to $c_1$]

. . .
. . .
. . .

POINTS WITH BEST C2:
X1=35, X2=11, X3=47, C2=39;
X1=45, X2=38, X3= 3, C2=47;
X1=41, X2= 2, X3=13, C2=51;
X1=36, X2=43, X3=30, C2=53.
[The four best points with regard to $c_2$.]

. . .
. . .

POINTS WITH BEST C3:
X1= 3, X2=47, X3=49, C3=196;
X1=27, X2=13, X3= 5, C3=343;
X1= 6, X2= 7, X3=21, C3=352;
X1=11, X2=15, X3=34, C3=355;
X1= 3, X2= 4, X3=18, C3=374.

*S*: If I take, say, 36 for $x_1$, 5 for $x_2$ and . . . and a small value, say, 2 for $x_3$, I should get just about the minimum . . . I don't think I can do any better . . . not when this noise is on all the time. | Output: $c_1$= 48, $c_2$=21, $c_3$=408, $\Sigma c$=477. End. |

* * *

THE BEST POINT SO FAR:
X1=28, X2=2, X3=47, SUM=463.
[P was cut off here.]

* * *

The following points may be worth mentioning with regard to the comparison between the above two records:

The simulation of both the results of *and* the reasoning behind the subject's decision making is fairly faithful, although the trial points are, of course, not identical. The only serious shortcoming of the model can be seen at Stage (B), when it does not notice the effect of large step sizes. Consequently, the program's best points with regard to $c_3$ just, so to speak, happen to be the best. The first point on the list ($x_1$=3, $x_2$=47, $x_3$=49, $c_3$=196) is "too good" and its weight causes the final selection mechanism to choose a "global optimum" with a much too large $x_3$. (The $x_3$=47 value of the first point on the best—$c_2$—list was also guilty in this decision.)

The quality of the computer search for minimum was also very similar to the human one. The machine obtained a minimum of 463 after 68 trials, as contrasted with the subject's minimum of 477 after 59 trials.

# COMPUTER EXPERIMENTS IN MOTOR LEARNING*

Gene R. Bussey
*Grafix, Incorporated*
*Albuquerque, New Mexico*

## INTRODUCTION

Motor learning can be defined as the information processing adaptations necessary for a system to achieve advantageous control over its relationship to its physical environment. For convenience, such learning is divided into two major processes: (1) perfection of the subsystem organization necessary to convert highly coded command sequences into the multitudinous signals necessary to the efficient utilization of musculature or other effectors, etc., and (2) the development of the control capabilities necessary for the timely generation of effective coded command sequences. This paper considers only the latter process as it might possibly be achieved in so-called "artificially intelligent" systems such as those using electronic data processing system control, as exemplified by a program described in detail in reference 1. Because of time and space limitations, this program will only be briefly summarized here to the extent necessary to explain the significance of the results obtained with various simple systematic creative means of response generation or synthesis.

*The research covered by this paper was sponsored by the Air Force Office of Scientific Research of the Air Force Office of Aerospace Research, under contracts AF 49(638)-1203 and AF 49(638)-1476.

### *Approaches to Artificial Intelligence*

Different efforts on the problem of achieving artificial intelligence owe the variety of their approaches to many valid practical and abstract considerations. Minsky has described a number of such approaches in his useful summary,[2] but no one seems to have dealt similarly with the rationales involved. Ultimately, of course, the goal is at least simply to extend the capabilities of machines so that they are capable of doing human work generally regarded as requiring significant intelligence, and most workers in the field choose an approach loosely characterizable by the fact that it requires the development through human analysis of a mechanizable scheme for accomplishing some particular intellectual task. Using such an approach, the contributory role of the machine is limited to parameter adjustment and other optimizations for procedures which are basically given and remain essentially unchangeable except by further human programming. The main attractiveness of such an approach is that it tends to produce effective results sooner and at lower cost. Samuel,[3] Doyle,[4] Lindsay,[5] Slagle,[6] Gelernter,[7] and Newell, Shaw, and Simon[8] have all produced programs of this kind.

However useful and efficient such effort may be, the devising of specialized procedures for playing checkers or recognizing patterns is more concerned

with devising task theories than it is with devising machines having artificial intelligence. To whatever extent such machines exhibit intelligence of the human beings who devised the procedures incorporated in the machine. This interpretation does not of course render the intellectual work done by such machines any less impressive or less welcome, but the fact still remains that operation of this type leaves much to be desired with respect to the full attainment of artificial intelligence.

Obviously, for every task which a machine can skillfully perform, it must have a perfected procedure whether this be one developed by human task analysis or by a machine. Devising such procedures and writing programs to perform them is intellectual work of a most demanding type, and naturally it constitutes another task we would like to see mechanized. For a machine to be truly intelligent, then, it must not only be able to optimize and efficiently perform skills perfected for it by human beings, it must be able to devise and perfect such skills completely on its own.

If one analyzes, say, how a naive subject learns checkers, given only the rules and the opportunity to play, it seems obvious that he does not learn by using processes which are specific to checker playing. Rather, what he uses must be transferable from other learning, and thus must have general usefulness in learning. Such capabilities seem mandatory for any intelligent system, artificial or not, if it is to cope with unexpected tasks. Useful or effective though they may be, artificially intelligent systems which are initially prepared for specific tasks, or researches following such an approach, avoid the ultimate question of intelligence—namely, how an intelligent system can acquire new capabilities for which it was not specifically prepared.

A few researchers have attempted a machine-creative approach, mostly on pattern recognition problems, with limited success. Friedberg[9] originated a program writing program, while Uhr and Vossler[10] have devised a pattern recognition program which creates and adjusts its own operators. These, and indeed virtually all known operational programs or systems of this type, appear to have no better method of generating "operators," "trials," or other possibly adaptive constructs, than a random one — either as a randomly generated construct or as a selection or strengthening of randomly given connections or logic.

The approach adopted here was directed along similar lines, but random method was subordinated to systematic procedures of various kinds. No effort was made to analyze task requirements and provide the machine with prepared skills or subskills suitable for handling them effectively. Instead, tasks in general and the basic creative processes for devising effective procedures for them were analyzed to find effective methods of creating such procedures, given only the means for obtaining pertinent information and for testing results. This effort is only in its early stages, so that only a few very rudimentary creative methods suitable for motor (or efferent) use have been investigated, but these methods involve rather general principles and techniques, even though such a severe constraint limited immediate results.

### *Motor Capabilities*

For automata, the nonhomeostatic motor activities of major concern are (1) accommodation, (2) locomotion, (3) manipulation, and (4) speech. "Accommodation" as used here refers to those motor processes necessary to maintaining balance, adjusting vision, and so forth. The literature on artificial intelligence emphasizing purely motor learning, as exemplified by the papers of Ernst[11] and Scott[12] is exceedingly sparse—undoubtedly due to priority interest in cognitive processes.

In the computer program to be discussed herein, only rudimentary capabilities were provided for accommodation and very limited capabilities were provided for locomotion and manipulation. No speech capabilities at all were provided, thus restricting linguistic experiments to coded or written modalities. Since the object of the research was to study the generation of effective coded command sequences, no effort was made to conceive practical muscle substitutes; rather, these were simulated by convenional programming which translates the coded command sequences into hypothetical physical effects according to a predetermined and invariable scheme, with these effects then being used in the simulation of inputs to the learning program.

## PROGRAM DESCRIPTION

### *Functional Organization*

The practical object of the research was to develop an effective motor learning program based on

general principles, both as a test of such principles and to provide an experimental system for further tests. To reduce machine costs, the experimental system consisted of two major program sections, the learning program, called PUPIL, and a simulation program called TUTOR which interacts with PUPIL as the latter's environment and as a proxy for the researcher. In addition to providing inputs to PUPIL in response to PUPIL outputs, TUTOR also computes and compiles evaluative data for use in analyzing performance and interpreting results.

TUTOR, it might be noted, contains a number of subprograms which are examples of the task-oriented type of approach to artificial intelligence. These subprograms replace a human being in the reinforcement feedback loop by scoring performance and providing task elements. If it were desired, say, to try PUPIL on learning the task of checker playing, then a program such as Samuel's would be a necessary subprogram for TUTOR if one wished to continue to avoid the involvement of a human being in the feedback loop. Herein lies one great importance of specialized programs, for the development of general programs cannot proceed apace unless there is available an adequate library of specialized programs for testing purposes.

The interaction between TUTOR and PUPIL is in the form of alternating responses. PUPIL emits a response code consisting of 15 bits, 9 of which are interpreted by TUTOR as motor control signals. What this 9-bit word does depends on the portions of TUTOR selected for the experiment, but in general the word is translated by computation or table lookup into a TUTOR response to PUPIL consisting of reinforcements and circumstantial data which a simulated sensing process in PUPIL converts to (1) a 6-bit "payoff" measure serving as a reinforcer, (2) 9 bits representing coded data pertaining to circumstances, and (3) 2 bits denoting food and shock reinforcement components. Thus, PUPIL and TUTOR alternately exchange responses, the one giving rise to the other, in a continuous manner.

The coded exchanges between PUPIL and TUTOR can signify any interaction whatever that can be represented by such codes. Depending on the particular TUTOR subroutines, they may presently represent the following tests:

1. Orientation Learning (an operant conditioning test)
2. Food Tray and Latch and Tray (an operant conditioning test)
3. Compensatory Tracking (a stimulus-response test)
4. Figure Drawing (a stimulus-response test)

The following tests are also planned:

1. Figure Tracing
2. Assembly Work
3. Maze Running
4. Weight Judging
5. Pursuit Rotor
6. English Discourse

*Reinforcement*

Reinforcement has two distinct aspects, the environmental aspect simulated by TUTOR, which provides positive, zero, or negative stimuli to PUPIL's several reinforcement inputs, and the learning system aspect, which evaluates the various inputs and consolidates them where necessary. Ultimately, the process of reinforcement in PUPIL will be a complex one involving not only a capability of learning reinforcement consolidation but its selective allocation to different processes as well. Presently, however, all reinforcement inputs are consolidated into one single payoff measure, using fixed initialized weighting factors, and the resultant figure is used directly in a fixed scheme to reinforce various processes involved. The payoff measure used has two kinds of components, objective ones due to TUTOR and subjective ones self-determined by PUPIL. The objective components from TUTOR range from negative to positive values according to the particular reward-punishment computation scheme being followed. In addition, program controls allow these reinforcements to be applied intermittently according to random or fixed ratio schedules.

Typical objective components, with their ranges expressed in TUTOR units, are:

1. FOOD — a "food" reward, range 0 to +1.
2. SHOCK — a negative reward, or punishment, range 0 to −1.
3. COACH — a "hotter-colder" reinforcer, range −1 to +1.
4. SCORE — a performance evaluator, range 0 to 1.

The subjective components of payoff are experimental ones incorporated to provide guidance when objective reinforcement is not immediately available. These components are provided as basic drives, not directly subject to self-modification, and presently consist of the following:

| | |
|---|---|
| 1. NOVELTY | A measure based on input and output variety. Ranges generally from minus to plus, its limits and range being determined by program initialization controls. |
| 2. ORGANIZATION IMPROVEMENT | A measure based on the change in potential payoff, as indicated by the summed payoff expectations for all learning. Ranges from minus to plus. |
| 3. PAYOFF DELTA | A component based on the changes in other payoff components. Ranges from negative to positive values. |

The relative weighting factors presently used by PUPIL which are initially set by program controls are the following:

| | | |
|---|---|---|
| 1. Objective Components | | |
| (a) FOOD | 0.30 | |
| (b) SHOCK | 0.26 | |
| (c) COACH | 0.30 | |
| (d) SCORE | 0.12 | |
| 2. Subjective Components | | |
| (a) NOVELTY | 0.10 | |
| (b) ORGANIZATION | 0.00 | (Suppressed) |

Obviously, the specified proportions of these weights will have a considerable influence on ultimate behavior. Such weighting, together with an arbitrary scale factor, results in a +1 food reward from TUTOR giving rise to a +19.53 component of payoff in PUPIL, while a −1 shock from TUTOR gives rise to a −16.42 payoff component, etc.

The incorporation of self-reinforcement possibilities in a learning system obviously has its pitfalls, for the danger exists that the system may be more preoccupied with its subjective self than with its real-world tasks. On the other hand, as indicated before, a need was felt for learning guidance in the absence of immediate objective reinforcement.

Regarding *novelty,* the postulation was that—other things being equal—it is generally desirable for an artificially intelligent system to function so as to attain considerable input variety as well as output variety. A system with such a drive should prove to be an active, curious one not susceptible to vicious circles of behavior which might trap it. On the other hand, it could become so obsessed with novelty that objective performance is seriously degraded. A possible solution here is to put the pertinent thresholds and weights involved under adaptive control, with the goal of setting them so that objective payoff is maximized, but this has not yet been undertaken.

The *organization* component is a rather complex one. In brief, it is based on the postulate that any response which increases the system's capability to perform by improving its organization deserves a proportional payoff. The problem, of course, is to gauge the degree to which organization is changed due to a response or response sequence. In the case of PUPIL, organization is presently measured by summing the utilities of all the response concepts, herein called classifiers, held by the system. The utility of such classifiers is computed as follows:

Define the *effectiveness, E,* of a classifier as the average payoff associated with its employment in making a response. Define the *relevance, R,* of a classifier as the ratio of the times it was used to the times it could have been used. Define its *applicability, A,* as the ratio of the response bits the classifier determines to the total number, 15. Then utility is arbitrarily defined as

$$U = ERA.$$

Other measures are of course possible—one based on summing rather than producting being one interesting alternative.

A classifier in the system is a Boolean expression specifying response bits as functions of the input bits. Initially, these functions are nonexistent, so that they must be constructed by the system as it learns. Any such classifier is of course of no use whatsoever if it has a zero or negative effectiveness, is never relevant, or has zero applicability, while it is very useful if it is highly effective, is often relevant, and determines a maximum number of response bits.

In summary, then, organization is improved whenever a new classifier having positive utility is acquired, whenever the utility of a classifier increases, whenever a better classifier replaces a poorer one, whenever a superior classifier is formed by combining two inferior ones, etc.

The classifiers concerned, pertaining as they do to motor control, differ markedly from classifiers derived from receptor-based systems. Contrast "red" with "push" or "round" with "turn." PUPIL, of course, does not develop classifiers in precisely this form, but nevertheless there are—and indeed must be—classifiers signifying such things by virtue of the fact that their coded representations are translated by the simulated musculature in TUTOR to the appropriate simulated physical effects.

The *delta* component of payoff is an experimental one based on changes in non-delta components and can be specified for a test when desired. When used, the cessation of a positive payoff is interpreted in a preorganized, nonmodifiable way as bad, while the cessation of a negative payoff is interpreted as good, etc. The reasoning behind such a second order payoff scheme was the intuition that a sensitivity to such changes is essential to proper learning performance. The central issue here lies in the role of drives to learning. My own belief is that basic drives are—and indeed must be—inherent in a system of the sort under discussion. A positive stimulus, or a positively increasing stimulus, must be inherently perceived as positive, while the converse is true for a negative or negatively-going stimulus. In actual fact, however, the immediate realization of an effective drive component based on differencing following simple schemes has so far been discouraging because the system inevitably found ways of achieving satisfactory net payoffs based on more or less complex response sequences where the nondifferencing or objective components were in fact quite low.

Owing to space limitations, the drive problem cannot be adequately discussed here. A separate paper on the topic is in preparation for early publication.

*Response Generation*

For any particular test situation, the problem for the PUPIL is one of generating responses which will yield satisfactory payoffs on the average, while the problem for the experimenter is proper evaluation of the response generation means being used. Of particular interest at this stage were means suitable for more or less "instinctive" and reflexive purposes. These are characterized by simplicity, utility, and immediacy such that they are consistently effective in improvising the many adaptations necessary to provide successful performance under continually different—although often nondistinctive—circumstances.

The difference between an adaptive and a nonadaptive system is that the one can modify its relationship with—and thus its average payoff from—its environment, while the other cannot. For reasons involving anti-chance presently to be pointed out, it is necessary for completeness to consider negatively adaptive—that is, systems which significantly degrade rather than improve their payoffs-as well as positively adaptive systems.

In what follows, special significance is given to the concept of *random behavior* which, for a system like PUPIL, would result from generating its successive 15-bit responses completely at random, as if no information pertinent to effective behavior were available. Now, for a given system in a given environment, such random behavior would yield a certain average payoff, termed the *random payoff rate,* which is defined as that associated with zero effective relative intelligence. Also, there exists a *maximum payoff rate limit* and a *minimum payoff rate limit,* the one being that rate theoretically attainable by a system having maximum effective relative intelligence and the other—purely for logical completeness—being that attainable by a system having maximum negative effective relative intelligence. Now, the payoff continuum defined by the range from maximum to random and random to minimum constitutes a scale useful for gauging the performance of a particular system operating systematically—that is, making effective use of available information. Thus, the effective relative intelligence (hereafter called simply "intelligence") is arbitrarily defined as

$$+I_{er} = \frac{\text{Actual Payoff Rate} - \text{Random Payoff Rate}}{\text{Minimum Payoff Rate} - \text{Random Payoff Rate}}$$

which ranges in value from zero to one for Actual Payoff Rate $\geq$ Random Payoff Rate. Similarly,

$$-I_{er} = -\frac{\text{Actual Payoff Rate} - \text{Random Payoff Rate}}{\text{Minimum Payoff Rate} - \text{Random Payoff Rate}}$$

defines negative intelligence for Actual Payoff Rate $\leq$ Random Payoff Rate.

PUPIL cannot compute the *I* measure precisely, because only a privileged observer can know the maximum or minimum possible payoff rates, but it can achieve a practical approximation by using the maximum and minimum observed payoff instead, or—probably better—an average of the *m* best and *n* worst payoffs.

Associated with any particular observed intelligence, whether positive or negative, is an empirical probability and thus a level of significance, such that an observed actual payoff rate differing substantially from the random payoff rate is extremely unlikely to be due to chance alone. Viewed in this way, intelligence then is that capability which produces behavior having a payoff rate significantly different from the chance or random payoff rate. If intelligence is sufficiently negative, it will nevertheless still be significant—that is, anti-chance—and presumably can be made positive by reversing the logic involved. Thus, in explanation of a previous remark, behavior which proves negatively adaptive is negatively intelligent, and the system manifesting it could in principle either be readily modified by the designer so as to achieve commensurate positive intelligence or else could so modify its own behavior.

*Response Generation Means*

The PUPIL system must generate responses in a continuous, unending sequence so long as it is in operation. These responses, of course, can be and often are null responses that cause little or no change in the PUPIL internally or between the PUPIL and its environment. There are $2^{15}$ or 32,768 distinct individual response codes that PUPIL can generate and these can be arranged in an extremely large number of different behavior sequences. Such generation involves the usual search problem—namely, that of efficiently retrieving or synthesizing "good" responses when there are so many possibilities. PUPIL was designed in part as an experimental system for the testing and evaluation of various means for response generation.

The most powerful means of response generation is believed to be one which functions so that all pertinent factors and past experience actively combine to suggest or elicit, through associations, either fruitful response possibilities or additional factors (such as applicable new classifications, discovered relationships, etc.) which will help in retrieving or synthesizing such fruitful responses. But this is an advanced approach requiring an extensive associative memory, and although it has been blocked out it has not yet been attempted with PUPIL. Rather, simple response generation means which were believed to have general motor utility have been evaluated, with the intent of perfecting evaluation techniques as well as the means themselves, either as complete generation techniques or as constituents in more elaborate schemes. Table 1 gives the means evaluated so far.

Table 1. Response Generation Means Used.

| Means No. | Descriptive Name | Type | Principle |
|---|---|---|---|
| 1 | Random Generation | Creative | Random |
| 2 | Classification | Learned | Proven Utility |
| 3 | Covert Response Repeating | Creative | Heuristic |
| 4 | Overt Response Repeating | " | " |
| 5 | Negative Analogy with Experience | " | " |
| 6 | Positive Analogy with Experience | " | " |
| 7 | Covert Response Copying | " | " |
| 8 | Overt Response Copying | " | " |

Before discussing these means, it should be recalled that the response word of PUPIL has two portions, an *overt* one of nine bits representing instructions to the simulated musculature, and a *covert* one of six bits which, in effect, serves as instructions from PUPIL to itself in a language of its own devising. This covert capability was provided to facilitate the making of identical overt responses in different response sequences and thus, in effect, represents context information.

The random means is one employing a standard pseudorandom number generator subroutine, used to fill in any part of either the covert or overt response not systematically determined.

*Classification* is the primary means provided for generating responses. It operates by learning consistent correlations between situation classes and response classes. These correlations are expressed in the form of Boolean *classifiers* which give output code elements in terms of logical functions of input code elements. Each such classifier is derived from experience and has recorded in association with it certain accumulated statistics permitting a computation of utility which, if greater than some arbitrary threshold, ensures that the classifier will be retained and used whenever applicable. In practice, classifiers are stored on a list according to utility rank as coded Boolean functions plus statistics on the number of times tried, the number of times used, the number of response bits coded, the total associated payoff accumulated, etc. Roughly speaking, a classifier as used by the program, is similar to several large multi-input, multi-output neurons having considerable internal logic. These classifiers are expressions giving output code elements as explicit functions of input code elements, with no provision yet for finding probably good implicit relationships through associative processes—earlier mentioned as believed to be the most powerful means for generating responses when explicit knowledge is not available.

The remaining response generation means other than the random means are ones that try to derive good responses using previous experience, operating whenever classification cannot provide a complete response. These means make use of short-term memory, either the last good response held in a special register or responses stored in a limited memory, called the *Stream of Experience Memory* (or SEM) where the last $n$ memorable responses and their antecedent situations are stored. For a response to be memorable it must be surprising in that its payoff differs from the expected payoff by some margin equal to or exceeding a threshold called MNEWS, presently set at 2.0 payoff units. The purpose of such selection is to reduce the storage and search burden while maintaining a relatively high quality of material for fill-in purposes.

## EVALUATION OF THE RESPONSE GENERATION MEANS USED

### *Experimental Conditions*

The data used in the following analysis were obtained from Run 1 of PUPIL on January 25, 1965. This particular PUPIL, the program of which was compiled on that date, was "primed" with 365 classifiers, a full Stream of Experience Memory, and other data obtained as a result of running PUPIL on January 24, 1965. The changes which justified the recompilation were minor ones. The test chosen for the run was an operant conditioning test in which the PUPIL was confined in an invisible pen measuring 16 by 16 units within which, on the first test made, a food tray was located at coordinates 8,-8. The PUPIL was initially positioned at these same coordinates and made 948 responses which resulted in its exploring the pen rather thoroughly, making moves which terminated in 87 percent of the 256 locations within the pen and contacting 95 percent of the pen boundary locations. The food tray was discovered and eaten from one time. Reinforcement arrangements were such that the delta and organization components of payoff were suppressed. Under these conditions the maximum average possible payoff was probably 19.53 for eating from the food tray while producing varied covert responses and changing position to avoid boredom.

Figure 1 of the Appendix shows the 9th Experience Performance Summary page for this 19-page run. The listing gives in order the response cycle number, the means generation methods used, the number of classifiers used, the number of associated payoffs used, the address of the best matching analogous situation in the Stream of Experience Memory, the payoff expectation for the response, the composite octal word containing PUPIL's response (last five digits) and the conversion of the environmental response from TUTOR (first seven digits), the resultant X,Y coordinates, the distance outside of the invisible pen attempted by the response, the SHOCK, COACH, FOOD and NOVELTY components of payoff, and the payoff less delta components (which is equal here to a measure called PAY), the energy reserve, the number of classifiers built on the response cycle, and the number of classifiers in the classifier table. The EVAL measures at the bottom of the page are the cumulative objective records for the means used as computed on a privileged information basis by TUTOR. These are not discussed here but full information is available from reference 1, which incidentally should be consulted for information on any item not adequately covered herein. The times-used figures shown include not only those for this run but from the 1451

responses of the priming run. The YIELD figures shown are simplified ones obtained by a rough moving average, and should not be confused with the payoff yield figures to be presented in the subsequent analysis.

The purpose of this test was to evaluate the available means of response generation and to test a newly introduced means-allocation control which could control usage for the various nonrandom means available to PUPIL but could not eliminate any of them altogether. The minimum relative frequency for any means was arbitrarily set at 10 percent, with reductions in such availability being made in accordance with the performance of the particular means as compared to the random means.

*Means Used*

*Analogy.* The two means using analogy are based on finding a situation recorded in the Stream of Experience Memory which corresponds most closely bit-for-bit with the existing situation. This is considered a very weak sort of analogy. (A more elaborate scheme effecting associative summing was abandoned earlier because of economic considerations.) Having found the most analogous situation, PUPIL examines the payoff associated with the response made, copying needed bits if the payoff was good (means 5) and logically complementing needed bits if the payoff was bad (means 6.).

*Response Copying and Repeating.* These four means involving response iteration are based on the heuristics (1) that an action which is working out well should be continued (response repeating), (2) and that a good action that was discontinued, with unfortunate results, is worth trying again (response copying). A compilation here is that PUPIL makes both an overt and a covert response, so that the payoff possibilities in particular cases may be more determined by one than the other. For example, at some particular juncture in a task the proper covert response may be that the part being worked with still isn't turned the right way while the proper overt response would be to try a new orientation. On the other hand, a proper overt response might be to repeat a move toward a desired object, while the proper covert response might be one that signifies full speed is now appropriate—a former hazard situation no longer prevailing. Brief descriptions of these means follow.

1. Covert Response Repeating. (Means 3) Repetition of a good covert response just made.
2. Overt Response Repeating. (Means 4) Repetition of a good overt response just made.
3. Covert Response Copying. (Means 7) The covert part of the last good response is copied.
4. Overt Response Copying. (Means 8) The overt part of the last good response is copied.

Recall that the above six means are all systematic fill-in or creative means—that is, means used to fill in part or all of a response after the classification means has been exhausted.

*Means Evaluations*

*General.* Means analysis based on payoffs (called YIELD) was a recent introduction in the PUPIL program, its purpose being to provide PUPIL with a criterion for allocating means, as previously described. The TUTOR program already had an evaluative subroutine, based on perfect information, which provided the objective EVAL measures of the effectiveness of various means, the results of which are not available to PUPIL. The YIELD computations as originally set up to minimize computational and memory requirements lacked the breadth and depth desirable for a comprehensive analysis. For this reason, a manual analysis has been made of the subject run, the data for which are tabulated in Appendix I. For those who would like to reproduce the data, the complete run tabulation itself is available from the author, as is the program which produced the run; this same program is also published in reference 1.

The subsequent analysis was made along lines, some aspects of which will probably be incorporated into PUPIL for control or into TUTOR for analytic purposes. Basically, the techniques offered have nonrigorous utility for evaluating the worth of various response generation means, both for the PUPIL program and for the researcher, and to provide information for program diagnostic purposes, since the analyses not only show the utility of the various means, but often disclose abnormalities due to design or program errors as well.

As responses were generated by PUPIL, various

means were used and recorded along with other data in the Experiment Performance Summary, making it possible to tabulate payoffs by means as listed in the Appendix. Table 2 summarizes the most important aspects associated with the various means used during the run. These means or their combinations are listed in rank order according to their average payoff. The table gives the rank, the combination of means used, the number of times the combination was used during the run, minimum and maximum payoff occurring, the payoff range, an effective relative intelligence measure based on the average of all completely random responses, and an effective relative intelligence measure based on random responses which were made under favorable conditions—namely, in the central region of the pen near the food tray (denoted as means 1* in Table 5 in the Appendix). These results are discussed in what follows.

Table 2. Means Combinations Data Ranked by Performance

| Rank | Means | No. of Cases | Avg. Payoff | Min. Payoff | Max. Payoff | Payoff Range | $I_{er}$ based on Means 1 | $I_{er}$ based on Means 1* |
|---|---|---|---|---|---|---|---|---|
| 1 | 4,5 | 12 | +2.51 | − 5.56 | +23.23 | 28.79 | +.288 | +.238 |
| 2 | 2,3 | 5 | +1.25 | − 0.06 | + 6.30 | 6.36 | +.235 | +.181 |
| 3 | 3,4 | 49 | +0.37 | −11.48 | + 7.03 | 18.51 | +.198 | +.142 |
| 4 | 1,4 | 17 | +0.30 | − 9.43 | + 6.30 | 15.73 | +.195 | +.139 |
| 5 | 2,4,5 | 1 | 0.00 | 0.00 | 0.00 | 0.00 | +.183 | +.125 |
| 6 | 3 | 78 | −0.19 | − 0.64 | 0.00 | 0.64 | +.175 | +.117 |
| 7 | 2,3,4 | 4 | −0.41 | − 1.41 | 0.00 | 1.41 | +.165 | +.107 |
| 8 | 4,6 | 29 | −0.45 | − 9.60 | + 7.09 | 16.69 | +.164 | +.105 |
| 9 | 7 | 54 | −0.51 | − 3.10 | 0.00 | 3.10 | +.161 | +.103 |
| 10 | 1,8 | 17 | −0.52 | −11.22 | + 4.72 | 15.94 | +.161 | +.102 |
| 11 | 2 | 43 | −1.75 | − 7.23 | + 7.09 | 14.32 | +.109 | +.047 |
| 12 | 7,8 | 33 | −2.12 | −11.58 | + 4.38 | 15.96 | +.094 | +.030 |
| 13 | 1* | 39 | −2.80 | −13.18 | + 7.09 | 20.27 | +.065 | .000 |
| 14 | 5 | 26 | −3.19 | −15.10 | + 5.40 | 20.50 | +.049 | −.029 |
| 15 | 5,8 | 17 | −4.16 | −11.95 | + 3.78 | 14.73 | +.008 | −.100 |
| 16 | 1 | 324 | −4.36 | −16.42 | + 8.10 | 24.52 | .000 | −.115 |
| 17 | 6,8 | 10 | −4.83 | −14.03 | + 1.45 | 15.48 | −.036 | −.149 |
| 18 | 6 | 228 | −5.13 | −16.42 | + 9.45 | 25.87 | −.064 | −.171 |
| 19 | 2,4,6 | 1 | −5.29 | − 5.29 | − 5.29 | 0.00 | −.077 | −.183 |
| 20 | 1,2,4 | 1 | −5.29 | − 5.29 | − 5.29 | 0.00 | −.077 | −.183 |

*Means 4,5.* This combination is one in which the overt portion of the response was filled in by repetition from a response just made and the covert part by positive analogy from a similar situation recorded in the Stream of Experience Memory. The average payoff of +2.51 was greatly enriched by a singular success in finding the food tray and eating from it, which provided a total payoff of +23.23. This means combination was used only 12 times, which is not fully satisfactory for statistical analysis purposes, but if we accept it as indicative, then we can make an estimate of the anti-chance significance of such an average by assuming it to be a sample average from a population of random averages. Table 3 summarizes the averages of the randomly derived payoffs for each of the 19 response blocks. From these values, the standard deviation obtained was 1.33 payoff units and the mean was −4.36. With these values as criteria, the significance of the 4, 5 combination is that associated with a deviation ratio of $5.17\sigma_R$, as shown by Table 4. A measure such as this, nonrigorous as it is, may yet have utility in learning programs as one permitting a rational ranking of response generation means.

At any rate, the success of this means combination was due to the fact that the repetition of a successful overt response (means 4, a spatial move) also tended to be successful. The covert portion filled in by means 5 cannot be considered significant at this stage of PUPIL development, since the self-formed language comprised by covert re-

Table 3. Means 1, Random Response Averages, by Blocks.

| i, Block No. | Times Used | $X_i$, Block Average | Abs. Deviation $\lvert X_i - X_R \rvert$ | $\lvert X_i - X_R \rvert^2$ |
|---|---|---|---|---|
| 1 | 15 | –4.29 | 0.07 | .0049 |
| 2 | 20 | –6.05 | 1.69 | 2.8561 |
| 3 | 8 | –5.45 | 1.09 | 1.1881 |
| 4 | 15 | –5.74 | 1.38 | 1.9044 |
| 5 | 11 | –3.51 | 0.85 | .7225 |
| 6 | 20 | –3.83 | 0.53 | .2809 |
| 7 | 13 | –2.78 | 1.58 | 2.4964 |
| 8 | 20 | –3.97 | 0.39 | .1521 |
| 9 | 13 | –3.02 | 1.34 | 1.7956 |
| 10 | 15 | –4.56 | 0.20 | .0400 |
| 11 | 24 | –5.26 | 0.90 | .8100 |
| 12 | 15 | –1.94 | 2.42 | 5.8564 |
| 13 | 18 | –1.25 | 3.11 | 9.6721 |
| 14 | 24 | –4.94 | 0.58 | .3364 |
| 15 | 17 | –4.25 | 0.11 | .0121 |
| 16 | 24 | –4.08 | 0.28 | .0784 |
| 17 | 13 | –4.96 | 0.60 | .3600 |
| 18 | 15 | –4.49 | 0.13 | .0169 |
| 19 | 24 | –6.63 | 2.27 | 5.1529 |
| | | | 19 \| | 33.7312 |

Average, $X_R = -4.36$

Sample Variance, $\sigma_R^2 = 1.7753$

Std. Deviation, $\sigma_R = \sqrt{1.7753} = 1.3324 \approx 1.33$

Table 4. Average Payoffs of Means Combinations Compared to Means 1 (Random) Averages.

| Rank, *i* | Means Comb. | No. of Cases | $X_i$, Avg. Payoff | Abs. Deviation, = $\lvert X_i - X_R \rvert$ | Critical Ratio = $\lvert X_i - X_R \rvert / \sigma_R$ |
|---|---|---|---|---|---|
| 1 | 4,5 | 12 | +2.51 | 6.87 | $5.17_R$ |
| 2 | 2,3 | 5 | +1.25 | 5.61 | $4.22_R$ |
| 3 | 3,4 | 49 | +0.37 | 4.73 | $3.56_R$ |
| 4 | 1,4 | 17 | +0.30 | 4.66 | $3.50_R$ |
| 5 | 2,4,5 | 1 | 0.00 | 4.36 | $3.28_R$ |
| 6 | 3 | 78 | –0.19 | 4.17 | $3.14_R$ |
| 7 | 2,3,4 | 4 | –0.41 | 3.95 | $2.97_R$ |
| 8 | 4,6 | 29 | –0.45 | 3.91 | $2.94_R$ |
| 9 | 7 | 54 | –0.51 | 3.85 | $2.89_R$ |
| 10 | 1,8 | 17 | –0.52 | 3.84 | $2.89_R$ |
| 11 | 2 | 43 | –1.79 | 2.57 | $1.93_R$ |
| 12 | 7,8 | 33 | –2.12 | 2.24 | $1.68_R$ |
| 13 | 1* | 38 | –2.80 | 1.56 | $1.17_R$ |
| 14 | 5 | 26 | –3.19 | 1.17 | $0.88_R$ |
| 15 | 5,8 | 17 | –4.16 | 0.20 | $0.15_R$ |
| 16 | 1 | 325 | –4.36 | 0.00 | $0.00_R$ |
| 17 | 6,8 | 10 | –4.83 | 0.47 | $0.35_R$ |
| 18 | 6 | 228 | –5.13 | 0.77 | $0.58_R$ |
| 19 | 2,4,6 | 1 | –5.29 | 0.93 | $0.70_R$ |
| 20 | 1,2,4 | 1 | –5,29 | 0.93 | $0.70_R$ |

sponses is in an extremely rudimentary or provisional stage of self-development.

*Means 2,3.* Here, classifiers built on successful overt responses provided that portion of the response, while means 3 filled in the covert portion from a good response just made. Only five cases occurred, hence the significance of the payoff average of +1.25 (which was largely due to the maximum payoff of +6.30) cannot be considered conclusive.

*Means 3,4.* The 49 cases, with an average payoff of +0.37 and and a vigorous range, indicate rather conclusively that this is a useful means, mostly at this stage due to the overt component furnished by means 4 (which incidentally cannot occur alone).

*Means 1,4.* Here the covert component was filled in at random with results quite similar to those for 3,4.

*Means 2,4,5.* An isolated inconclusive case.

*Means 3.* Since this means provides a response containing only a covert portion, nothing is done by PUPIL to change its relationship with its environment. Under many test circumstances, such immobilization could yield large positive or negative payoffs, but on the occasions when it occurred here only a slight negative component due to boredom was usually present. Thus the significance of the average payoff attained was that, in this particular negative environment on the occasions used, it was a rather sure way to avoid a possibly substantial negative payoff. The ultimate purpose of such a hesitant means is to provide for mental accommodations which may be necessary in forming successful action chains. In other words, it provides in elementary form for reasoning processes using classifier chains which could lead to a correct response. At this stage of development, however, this type of chaining was not to be expected. If it were to occur, the means following would involve means 2, and for the means 3 usage to be effective, the subsequent means 2 payoffs would have to be significantly better than the random average under similar conditions.

Theoretically, there being no program facility yet for ordering means according to their utility, the allocation of means 3 should have been more nearly equal to that for means combination 3-4, while

means 4-5 and 4-6 should have occurred more often. This discrepancy led to the discovery of a minor program error.

*Other Means.* The remaining means deserve little additional comment, their results largely being evident by inspecting Tables A and C. However, it is worth pointing out that means 8 and combinations involving means 8 were noted often to have repeated an overt response which had already been tried unsuccessfully by means 4. This led to the discovery of a program error which occasionally permitted this undesired event to occur.

Note in general that means involving overt elements obtained by 2, 3, and 4 were fairly satisfactory, but that means involving overt elements obtaied from 6 were worse than random but not significantly so. Note also that means 6 was used 228 times, or for 24 percent of the responses. A programming fault is surely involved here, for 5 and 6, at present, without means sequencing, are means of last resort. The use of means 6 was expected to exceed the use of means 5 in about the proportion shown, since there were many more negative experiences than positive ones. Means 6, it should be recalled, makes use of "negative analogy" in that the response previously made in an analogous situation—which had associated with it a negative payoff —was logically complemented to provide a trial response.

The fact that the average payoff for means 6 is less than the average payoff at random, while not scientifically significant, is practically significant in that it has led to the discovery of program faults which have not yet been fully investigated. Briefly, best matches were being made that were far from being the best available and there is a persistent and probably abnormal sequential trend with time in the best match addresses.

*Evaluation*

The results obtained indicate that most of these means were effective under the operant conditioning test conditions in use. Means 6, the worst means, was in fact used less often as the run progressed, but this failed to improve performance materially because its performance was not significantly worse than random anyway and the random means usually had to be used in its place. Later, when a planned "internal perception" is introduced, PUPIL will have an opportunity to develop and learn criteria for means selection not based on performance records alone. The current introduction of means or means combination sequencing according to payoff yield figures of the type presented herein, the correction of various program faults, and the planned introduction of additional primitive means and an adaptive subprogram for generating new means, should all contribute material to PUPIL's performance.

## ENTROPY CONSIDERATIONS

An appropriate place to measure the so-called entropy associated with a system like PUPIL is believed to be with respect to its coded output responses. Such a measure was approximated by applying the usual entropy formula for information theory to each 50 responses. The result was then normalized by dividing it by the maximum possible 50 response entropy to achieve an entropy index,

$$H_{BLOCK} = \frac{-\sum_{i=1}^{50} p_i \cdot \log_2 p_i}{-\sum_{i=1}^{50} \frac{1}{50} \cdot \log_2 \frac{1}{50}} = -\frac{1}{5.64} \sum_{i=1}^{50} p_i \cdot \log p_i$$

where the $p_i$ are the observed relative frequencies of the different responses within the block of 50 responses. This index was high, beginning at about 0.8 and rising steadily (and unexpectedly) to a level over 0.9 at the end of the run). Most of this increase is attributed to an increase in use of the random means (from 36 times in the first 100 responses to 45 times in the last 100 responses) resulting from reduced use allocation for means 6. However, a part of the increase probably meant that the PUPIL was growing more resourceful in its attempt to solve the test problem. But had it learned to find the food tray and eat from it as continuously as its novelty drive would permit, its entropy index would surely have decreased drastically to low levels.

While not yet used in such a manner, it is believed that the entropy index is a useful measure of the general intellectual health of learning systems—both for the system itself and for its designer. Surely a vigorous system should exhibit a high entropy index while striving to solve a problem, but this should also be accompanied by significant I measures for its response means. Earlier PUPILs with a

low entropy index have invariably also exhibited mediocre *I* measures—graphs of which present a curiously "squeezed" appearance suggesting a vital intellectual deficiency. Besides indicating inadequate variety of behavior, a low entropy index may also mean that the system has developed inefficient coding.

## SUMMARY

A coded-input/coded-output general learning program, PUPIL, interacting with a program, TUTOR, simulating its environment, was tested in operant conditioning experiments involving elementary spatial orientation, locomotion, accommodation, and manipulation. The responses generated by PUPIL consist of a covert part, representing context information, transmitted to itself for use in generating the next response, and an overt part transmitted to TUTOR for conversion into an input word to PUPIL, one part of which represents physical circumstances and the other part of which is a reinforcer having positive or negative value.

PUPIL's main means of response generation uses self-constructed classifiers (representing concepts) which specify output elements in terms of external and internal input elements. This means was not fully discussed, for its development is not satisfactorily complete due to inadequate length of computer runs, but brief runs on various tests indicate it to have promising utility in machine applications requiring intelligent control.

The main results reported concern an objective measure of performance, arbitrarily termed the *effective relative intelligence,* which was used to evaluate six elementary creative means of response generation under conditions when insufficient information was available for complete classification. This evaluation indicated that eleven combinations largely involving four of these were significantly intelligent under the experimental circumstances (means 3, 4, 5, 7) with one other (means 8) showing promise, and another (means 6) being no better than random, evidently due to program errors.

## CONCLUSION

It is believed that the TUTOR program is an effective one for the efficient objective testing of learning programs such as PUPIL under controlled conditions, and that the PUPIL program constitutes an adequate basic system for the testing of response generation means or other learning system features or techniques by incorporation and subsequent testing with TUTOR, which incidentally analyzes and evaluates performance and prints out results in a convenient form for assimilation. Both programs are available in punched card form from Grafix, Inc., for card reproduction costs only. The programming language is FORTRAN IV.

It is also believed that the $I_{er}$ measure or the systematic/random deviation ratio are simple objective ones which make it possible, in relatively short computer runs, to determine the effectiveness of response generation means or other various techniques or procedures involved in system performance, so that the system can improve its performance or be improved by its designer without incurring undue expenses.

## REFERENCES

1. G. R. Bussey, "A General Adaptive Motor Learning Program for a Digital Computer," Parts I and II, Defense Documentation Center document AD-611334; Parts III and IV, AD-611335.

2. M. L. Minsky, "Steps Toward Artificial Intelligence," *Computers and Thought,* E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 406-450.

3. A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers," ibid., pp. 71-105.

4. W. Doyle, "Recognition of Sloppy Hand-Printed Characters," in *Proceedings of the Western Joint Computer Conference,* May 3-5, 1963, pp. 133-142.

5. R. K. Lindsay, "Inferential Memory as the Basis of Machines Which Understand Natural Language," in *Computers and Thought,* E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 217-233.

6. J. R. Slagle, "A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus," ibid., pp. 191-203.

7. H. Gelernter, "Realization of a Geometry-Theorem Proving Machine," ibid., pp. 134-152.

8. A. Newell, J. C. Shaw and H. A. Simon, "Empirical Explorations with the Logic Theory Machine: A Case Study in Heuristics," ibid., pp. 109-133.

9. R. M. Friedberg, "A Learning Machine, Part I," *IBM Journal of Research and Development,* Jan. 1958, pp. 2-13.

10. L. Uhr and C. Vossler, "A Pattern-Recognition Program that Generates, Evaluates, and Adjusts Its Own Operators," in *Computers and Thought,* E. A. Feigenbaum and J. Feldman, eds., McGraw-Hill, New York, 1963, pp. 251-268.

11. H. A. Ernst, "MH-1, A Computer-Operated Mechanical Hand," dissertation Ph.D., E.E., Massachusetts Institute of Technology, 1961; presented at the Western Joint Computer Conference, May 1961.

12. K. R. Scott, "A Stochastic Motor Learning Analog," Defense Documentation Center document AD-260218.

## APPENDIX

EXPERIMENT PERFORMANCE SUMMARY RUN 1 25 JAN 65 PUPIL 25016500

| RCNO | METHODS | USED | APPLY | MBS | PAYOFF | LAST | JX | JY | OUT | SWITCH | CUTOFF | SCORE | F(S) | C S | NOVEL PAY-OFF | PAY RESERV | BIL | HAV E | SKILL | TSKIL | MOD E |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1851 | 000010000 | 0 | 15 | 6 | 4.00 | 400002523053 | 1 | 9 | 0 | 0. | 0. | 0. | 0. | -0. | -0.12 | -0.12 -0.12 | -178. | 0 | 463 | 0. | 0. | 1 |
| 1852 | 100000000 | 0 | 15 | 8 | 0. | 000002160957 | 1 | 9 | 0 | 0. | 0. | 0. | 0. | -0. | 0. | 0. 0. | -178. | 0 | 463 | 0. | 0. | 1 |
| 1853 | 100000000 | 0 | 15 | 8 | 0. | 000002112444 | 1 | 5 | 0 | 0. | 0. | 0. | 0. | -0. | 0. | 0. 0. | -182. | 0 | 463 | 0. | 0. | 1 |
| 1854 | 100000000 | 0 | 15 | 8 | 0. | 610002517440 | 0 | 5 | 4 | -7.17 | -1.35 | 0. | 0. | -0. | 0. | -8.52 -8.52 | -184. | 0 | 463 | 0. | 0. | 1 |
| 1855 | 100000000 | 0 | 15 | 12 | 0. | 470002500151 | 0 | 5 | 2 | -3.94 | 0. | 0. | 0. | -0. | 0. | -3.94 -3.94 | -184. | 0 | 463 | 0. | 0. | 1 |
| 1856 | 010000000 | 1 | 0 | -0 | 10.47 | 450002554200 | 0 | 7 | 1 | -2.68 | 0. | 0. | 0. | -0. | 0. | -2.68 -2.68 | -186. | 0 | 463 | 0. | 0. | 1 |
| 1857 | 000001000 | 0 | 15 | 14 | 8.00 | 201212077626 | 6 | 7 | 0 | 0. | 8.10 | 0. | 0. | -0. | 0. | 8.10 8.10 | -184. | 2 | 465 | 0. | 0. | 1 |
| 1858 | 001100000 | 0 | 15 | -0 | 8.00 | 450002577626 | 12 | 7 | 0 | 0. | -2.70 | 0. | 0. | -0. | 0. | -2.70 -2.70 | -193. | 0 | 463 | 0. | 0. | 1 |
| 1859 | 000000110 | 0 | 15 | -0 | 8.10 | 670003577626 | 15 | 7 | 4 | -7.17 | -4.05 | 0. | 0. | 0. | -0.36 | -11.58 -11.58 | -200. | 0 | 463 | 0. | 0. | 1 |
| 1860 | 000001000 | 0 | 15 | 20 | 2.00 | 020002000151 | 14 | 6 | 0 | 0. | 1.35 | 0. | 0. | -0. | 0. | 1.35 1.35 | -200. | 2 | 465 | 0. | 0. | 1 |
| 1861 | 001100000 | 0 | 15 | -0 | 2.00 | 020002000151 | 13 | 5 | 0 | 0. | 1.57 | 0. | 0. | -0. | -0.36 | 1.21 1.21 | -199. | 1 | 466 | 0. | 0. | 1 |
| 1862 | 000101000 | 0 | 15 | 16 | 1.20 | 020003023151 | 12 | 4 | 0 | 0. | 1.89 | 0. | 0. | 0. | -0.42 | 1.47 1.47 | -198. | 1 | 467 | 0. | 0. | 1 |
| 1863 | 000101000 | 0 | 15 | 2 | 0.40 | 430003575151 | 11 | 3 | 0 | 0. | -1.35 | 0. | 0. | 0. | -0.28 | -1.63 -1.63 | -201. | 0 | 464 | 0. | 0. | 1 |
| 1864 | 000010010 | 0 | 15 | 22 | 0.40 | 430003575151 | 10 | 2 | 0 | 0. | -1.35 | 0. | 0. | 0. | -0.64 | -1.99 -1.99 | -203. | 0 | 464 | 0. | 0. | 1 |
| 1865 | 100000000 | 0 | 15 | -0 | 0. | 550002553344 | 10 | 0 | 2 | -3.94 | -2.70 | 0. | 0. | -0. | -0.30 | -6.94 -6.94 | -208. | 0 | 464 | 0. | 0. | 1 |
| 1866 | 000001000 | 0 | 15 | 20 | 2.00 | 470002500151 | 10 | 0 | 2 | -3.94 | 0. | 0. | 0. | -0. | 0. | -3.94 -3.94 | -208. | 0 | 464 | 0. | 0. | 1 |
| 1867 | 010000000 | 1 | 0 | -0 | 6.95 | 052612454200 | 10 | 2 | 0 | 0. | 2.70 | 0. | 0. | -0. | 0. | 2.70 2.70 | -207. | 1 | 465 | 0. | 0. | 1 |
| 1868 | 010101000 | 2 | 6 | 28 | 5.34 | 520002577617 | 15 | 7 | 2 | -3.94 | -1.35 | 0. | 0. | -0. | 0. | -5.29 -5.29 | -213. | 0 | 463 | 0. | 0. | 1 |
| 1869 | 000001010 | 0 | 15 | 20 | 0.80 | 450003500200 | 15 | 9 | 1 | -2.68 | 0. | 0. | 0. | -0. | 0. | -2.68 -2.68 | -215. | 0 | 463 | 0. | 0. | 1 |
| 1870 | 000001000 | 0 | 15 | 30 | 3.00 | 020002023577 | 10 | 14 | 0 | 0. | 1.35 | 0. | 0. | -0. | 0. | 1.35 1.35 | -219. | 2 | 465 | 0. | 0. | 1 |
| 1871 | 001000000 | 0 | 15 | -0 | 2.40 | 400002523077 | 10 | 14 | 0 | 0. | 0. | 0. | 0. | -0. | -0.06 | -0.06 -0.06 | -219. | 0 | 463 | 0. | 0. | 1 |
| 1872 | 000000100 | 0 | 15 | -0 | 1.08 | 400003123077 | 10 | 14 | 0 | 0. | 0. | 0. | 0. | -0. | -0.42 | -0.42 -0.42 | -219. | 0 | 463 | 0. | 0. | 1 |
| 1873 | 100000000 | 0 | 15 | 8 | 0. | 400002123665 | 4 | 14 | 0 | 0. | 0. | 0. | 0. | -0. | -0.38 | -0.38 -0.38 | -225. | 0 | 463 | 0. | 0. | 1 |
| 1874 | 000010000 | 0 | 15 | 36 | 1.00 | 400002123077 | 4 | 14 | 0 | 0. | 0. | 0. | 0. | -0. | -0.34 | -0.34 -0.34 | -225. | 0 | 463 | 0. | 0. | 1 |
| 1875 | 100000000 | 0 | 15 | 8 | 0. | 113312470337 | 7 | 11 | 0 | 0. | 4.72 | 0. | 0. | -0. | 0. | 4.72 4.72 | -223. | 2 | 465 | 0. | 0. | 1 |
| 1876 | 001000000 | 0 | 15 | -0 | 0. | 403312170037 | 7 | 11 | 0 | 0. | 0. | 0. | 0. | -0. | -0.06 | -0.06 -0.06 | -223. | 0 | 463 | 0. | 0. | 1 |
| 1877 | 000000100 | 0 | 15 | -0 | 3.78 | 403313570037 | 7 | 11 | 0 | 0. | 0. | 0. | 0. | -0. | -0.42 | -0.42 -0.42 | -223. | 0 | 463 | 0. | 0. | 1 |
| 1878 | 100000000 | 0 | 15 | 40 | 0. | 063212467132 | 8 | 10 | 0 | 0. | 3.15 | 0. | 0. | -0. | 0. | 3.15 3.15 | -221. | 2 | 465 | 0. | 0. | 1 |
| 1879 | 000110000 | 0 | 15 | 18 | 3.20 | 110002077132 | 9 | 9 | 0 | 0. | 4.72 | 0. | 0. | -0. | 0. | 4.72 4.72 | -217. | 1 | 466 | 0. | 0. | 1 |
| 1880 | 001000000 | 0 | 15 | -0 | 2.56 | 400003577032 | 9 | 9 | 0 | 0. | 0. | 0. | 0. | -0. | -0.16 | -0.16 -0.16 | -217. | 0 | 463 | 0. | 0. | 1 |
| 1881 | 000000100 | 0 | 15 | -0 | 3.78 | 410003577032 | 9 | 9 | 0 | 0. | 0. | 0. | 0. | -0. | -0.52 | -0.52 -0.52 | -217. | 0 | 463 | 0. | 0. | 1 |
| 1882 | 100000000 | 0 | 15 | 46 | 0. | 450002540215 | 11 | 11 | 0 | 0. | -2.70 | 0. | 0. | -0. | 0. | -2.70 -2.70 | -222. | 0 | 463 | 0. | 0. | 1 |
| 1883 | 100000000 | 0 | 15 | 30 | 0. | 620002526527 | 15 | 11 | 2 | -3.94 | -5.40 | 0. | 0. | -0. | 0. | -9.34 -9.34 | -232. | 0 | 463 | 0. | 0. | 1 |
| 1884 | 000010000 | 0 | 15 | 36 | 1.00 | 450002523077 | 15 | 11 | 1 | -2.68 | 0. | 0. | 0. | -0. | 0. | -2.68 -2.68 | -232. | 0 | 463 | 0. | 0. | 1 |
| 1885 | 000001000 | 0 | 15 | 2 | 9.00 | 560002554700 | 15 | 15 | 4 | -7.17 | 0. | 0. | 0. | -0. | 0. | -7.17 -7.17 | -236. | 0 | 463 | 0. | 0. | 1 |
| 1886 | 100000000 | 0 | 15 | 2 | 9.00 | 022612403153 | 14 | 14 | 0 | 0. | 1.35 | 0. | 0. | -0. | 0. | 1.35 1.35 | -235. | 2 | 465 | 0. | 0. | 1 |
| 1887 | 001000000 | 0 | 15 | -0 | 7.20 | 402612503053 | 14 | 14 | 0 | 0. | 0. | 0. | 0. | -0. | -0.16 | -0.16 -0.16 | -235. | 0 | 463 | 0. | 0. | 1 |
| 1888 | 000000110 | 0 | 15 | -0 | 1.35 | 022513003153 | 13 | 13 | 0 | 0. | 1.57 | 0. | 0. | -0. | -0.12 | 1.45 1.45 | -235. | 2 | 465 | 0. | 0. | 1 |
| 1889 | 000110000 | 0 | 15 | 8 | 0.40 | 032412003153 | 12 | 12 | 0 | 0. | 1.89 | 0. | 0. | -0. | -0.38 | 1.51 1.51 | -234. | 1 | 466 | 0. | 0. | 1 |
| 1890 | 001100000 | 0 | 15 | -0 | 0.40 | 032313003153 | 11 | 11 | 0 | 0. | 2.36 | 0. | 0. | -0. | -0.44 | 1.92 1.92 | -232. | 1 | 467 | 0. | 0. | 1 |
| 1891 | 001100000 | 0 | 15 | -0 | 0.40 | 042213403153 | 10 | 10 | 0 | 0. | 3.15 | 0. | 0. | -0. | -0.70 | 2.45 2.45 | -230. | 1 | 468 | 0. | 0. | 1 |
| 1892 | 001100000 | 0 | 15 | -0 | 0.40 | 072113403153 | 9 | 9 | 0 | 0. | 4.72 | 0. | 0. | -0. | -0.86 | 3.86 3.86 | -226. | 1 | 469 | 0. | 0. | 1 |
| 1893 | 000110000 | 0 | 15 | 10 | 0.40 | 372013603153 | 8 | 8 | 0 | 0. | 4.72 | 0. | 19.53 | 0. | -1.02 | 23.23 23.23 | -203. | 1 | 470 | 0. | 0. | 1 |
| 1894 | 001100000 | 0 | 15 | -0 | 0.40 | 420003103153 | 7 | 7 | 0 | 0. | -1.35 | 0. | 0. | 0. | 0. | -1.35 -1.35 | -206. | 0 | 466 | 0. | 0. | 1 |
| 1895 | 000000110 | 0 | 15 | -0 | 23.24 | 430003503153 | 6 | 6 | 0 | 0. | -1.35 | 0. | 0. | 0. | -0.36 | -1.71 -1.71 | -208. | 0 | 466 | 0. | 0. | 1 |
| 1896 | 000001000 | 0 | 15 | 16 | 1.00 | 450002574624 | 12 | 6 | 0 | 0. | -2.70 | 0. | 0. | -0. | 0. | -2.70 -2.70 | -217. | 0 | 466 | 0. | 0. | 1 |
| 1897 | 100000000 | 0 | 15 | 6 | 0. | 620002553527 | 15 | 6 | 3 | -5.44 | -4.05 | 0. | 0. | -0. | 0. | -9.49 -9.49 | -224. | 0 | 466 | 0. | 0. | 1 |
| 1898 | 100000000 | 0 | 15 | 2 | 0. | 560002502315 | 15 | 6 | 4 | -7.17 | 0. | 0. | 0. | -0. | 0. | -7.17 -7.17 | -224. | 0 | 466 | 0. | 0. | 1 |
| 1899 | 010000000 | 1 | 0 | -0 | 10.47 | 050002423357 | 12 | 3 | 0 | 0. | 2.70 | 0. | 0. | -0. | 0. | 2.70 2.70 | -224. | 1 | 467 | 0. | 0. | 1 |
| 1900 | 001100000 | 0 | 15 | -0 | 10.47 | 550002523357 | 9 | 0 | 1 | -2.68 | -4.05 | 0. | 0. | -0. | -0.26 | -6.99 -6.99 | -231. | 0 | 466 | 0. | 0. | 1 |
| | RUN TOTALS | | | | | | | | | -1215. | 47. | 0. | 20. | 0. | -46. | -1194. -1194. | | | | | | |

HBLOCK (50 RESPONSE ENTROPY ) 0.782 EVAL 1)-0.011 2) 0.007 3) 0.076 4) 0.061 5)-0.017 6)-0.000 7)-0.013 8)-0.016 9) 0.
USED 1) 731. 2) 143. 3) 241. 4) 242. 5) 95. 6) 566. 7) 180. 8) 161. 9) -0.
YIELD 1)-3.352 2)-0.916 3) 0.034 4) 1.370 5)-0.172 6)-2.911 7)-1.081 8)-2.029 9)-0.720

Figure 1. Data sample page.

Table 5. Payoff Data by Means Combinations from PUPIL 25016500,

| Means 1 | | | | Means 1, Cont'd. | | | | Means 1, Cont'd. | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| PG. NO. | RCNO | PAYOFF | BLOCK AVG. | PG. NO. | RCNO | PAYOFF | BLOCK AVG. | PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
| 1 | 1454 | -4.05 | | 4 | 1601 | -13.75 | | 6 | 1745 | +2.36 | |
| 1 | 1457 | -16.42 | | 4 | 1604 | +4.05 | | 6 | 1748 | -4.05 | |
| 1 | 1460 | -11.33 | | 4 | 1607 | -5.72 | | 6 | 1750 | +1.35 | -3.83 |
| 1 | 1462 | -3.94 | | 4 | 1609 | -16.42 | | 7 | 1755 | -12.88 | |
| 1 | 1465 | -4.03 | | 4 | 1611 | -3.94 | | 7 | 1760 | -6.72 | |
| 1 | 1468 | -2.68 | | 4 | 1613 | -2.68 | | 7 | 1768 | -1.41 | |
| 1 | 1470 | -2.68 | | 4 | 1615 | -3.94 | | 7 | 1779 | 0.00 | |
| 1 | 1472 | +1.35 | | 4 | 1621 | -3.94 | | 7 | 1783 | 0.00 | |
| 1 | 1485 | -2.68 | | 4 | 1622 | -13.75 | | 7 | 1785 | 0.00 | |
| 1 | 1490 | -8.08 | | 4 | 1624 | -16.42 | | 7 | 1786 | +3.15 | |
| 1 | 1491 | -7.17 | | 4 | 1626 | -9.13 | | 7 | 1789 | 0.00 | |
| 1 | 1493 | -2.68 | | 4 | 1629 | -7.17 | | 7 | 1790 | +2.70 | |
| 1 | 1496 | -1.33 | | 4 | 1630 | +4.05 | | 7 | 1796 | -0.38 | |
| 1 | 1498 | -2.68 | | 4 | 1634 | 0.00 | | 7 | 1797 | -6.75 | |
| 1 | 1499 | +4.05 | -4.29 | 4 | 1636 | +2.70 | -5.74 | 7 | 1798 | -6.64 | |
| 2 | 1503 | -3.06 | | 5 | 1654 | -9.13 | | 7 | 1800 | -7.17 | -2.78 |
| 2 | 1504 | -11.33 | | 5 | 1656 | -7.17 | | 8 | 1801 | +2.70 | |
| 2 | 1505 | -3.94 | | 5 | 1661 | +1.35 | | 8 | 1805 | -7.17 | |
| 2 | 1506 | -9.13 | | 5 | 1670 | +2.70 | | 8 | 1806 | -7.17 | |
| 2 | 1507 | -16.42 | | 5 | 1675 | -10.78 | | 8 | 1807 | -2.68 | |
| 2 | 1510 | -9.13 | | 5 | 1678 | -7.17 | | 8 | 1810 | +5.40 | |
| 2 | 1512 | -2.74 | | 5 | 1680 | -13.75 | | 8 | 1813 | -1.35 | |
| 2 | 1515 | -2.68 | | 5 | 1689 | +6.75 | | 8 | 1814 | +2.36 | |
| 2 | 1524 | -6.79 | | 5 | 1693 | -5.44 | | 8 | 1817 | -4.05 | |
| 2 | 1525 | -7.17 | | 5 | 1694 | +8.10 | | 8 | 1819 | -2.68 | |
| 2 | 1526 | +4.05 | | 5 | 1697 | -4.05 | -3.51 | 8 | 1820 | -16.42 | |
| 2 | 1529 | 0.00 | | 6 | 1702 | 0.00 | | 8 | 1821 | -5.44 | |
| 2 | 1530 | -12.57 | | 6 | 1704 | -9.87 | | 8 | 1825 | -5.47 | |
| 2 | 1532 | -3.94 | | 6 | 1705 | -13.75 | | 8 | 1826 | +2.70 | |
| 2 | 1538 | -11.83 | | 6 | 1706 | -7.17 | | 8 | 1829 | 0.00 | |
| 2 | 1539 | -2.68 | | 6 | 1708 | -11.33 | | 8 | 1831 | +1.35 | |
| 2 | 1540 | -5.44 | | 6 | 1709 | -2.68 | | 8 | 1834 | 0.00 | |
| 2 | 1541 | -5.44 | | 6 | 1714 | -5.29 | | 8 | 1835 | -10.48 | |
| 2 | 1546 | 0.00 | | 6 | 1721 | +1.57 | | 8 | 1836 | -16.42 | |
| 2 | 1547 | -10.84 | -6.05 | 6 | 1724 | -1.43 | | 8 | 1846 | -1.35 | |
| 3 | 1557 | +7.09 | | 6 | 1729 | -2.70 | | 8 | 1847 | -13.18 | -3.97 |
| 3 | 1567 | -13.75 | | 6 | 1730 | +2.70 | | 9 | 1852 | 0.00 | |
| 3 | 1569 | -5.29 | | 6 | 1733 | -14.15 | | 9 | 1853 | 0.00 | |
| 3 | 1585 | +2.70 | | 6 | 1734 | -2.68 | | 9 | 1854 | -8.52 | |
| 3 | 1588 | -11.33 | | 6 | 1736 | -2.68 | | 9 | 1855 | -3.94 | |
| 3 | 1590 | -3.94 | | 6 | 1739 | -3.94 | | 9 | 1865 | -6.94 | |
| 3 | 1597 | -16.42 | | 6 | 1743 | -0.18 | | 9 | 1873 | -0.38 | |
| 3 | 1599 | -2.68 | -5.45 | 6 | 1744 | -2.70 | | 9 | 1875 | +4.72 | |

Means 1 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 9 | 1878 | +3.15 | |
| 9 | 1882 | -2.70 | |
| 9 | 1883 | -9.34 | |
| 9 | 1886 | +1.35 | |
| 9 | 1897 | -9.49 | |
| 9 | 1898 | -7.17 | -3.02 |
| 10 | 1910 | -1.33 | |
| 10 | 1911 | -2.68 | |
| 10 | 1912 | 0.00 | |
| 10 | 1914 | -11.33 | |
| 10 | 1915 | -13.75 | |
| 10 | 1916 | -1.33 | |
| 10 | 1927 | -12.13 | |
| 10 | 1935 | +4.36 | |
| 10 | 1938 | 0.00 | |
| 10 | 1939 | +6.30 | |
| 10 | 1942 | -8.52 | |
| 10 | 1943 | -2.68 | |
| 10 | 1946 | -11.33 | |
| 10 | 1948 | -2.68 | |
| 10 | 1949 | -11.33 | -4.56 |
| 11 | 1953 | -5.40 | |
| 11 | 1954 | 0.00 | |
| 11 | 1955 | -6.64 | |
| 11 | 1956 | -2.68 | |
| 11 | 1957 | -8.52 | |
| 11 | 1961 | -5.44 | |
| 11 | 1965 | +1.89 | |
| 11 | 1972 | -1.35 | |
| 11 | 1974 | 0.00 | |
| 11 | 1975 | 0.00 | |
| 11 | 1977 | +2.70 | |
| 11 | 1980 | -14.03 | |
| 11 | 1981 | -16.42 | |
| 11 | 1983 | -2.68 | |
| 11 | 1986 | -2.68 | |
| 11 | 1989 | -9.13 | |
| 11 | 1990 | -5.29 | |
| 11 | 1991 | -2.68 | |
| 11 | 1993 | -7.17 | |
| 11 | 1994 | -2.68 | |
| 11 | 1995 | -3.94 | |
| 11 | 1996 | -11.33 | |
| 11 | 1999 | -9.13 | |
| 11 | 2000 | -13.75 | -5.26 |

Means 1 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 12 | 2005 | -3.94 | |
| 12 | 2007 | -2.68 | |
| 12 | 2008 | -16.42 | |
| 12 | 2013 | -7.17 | |
| 12 | 2015 | -2.68 | |
| 12 | 2017 | -2.68 | |
| 12 | 2018 | -5.44 | |
| 12 | 2020 | -2.68 | |
| 12 | 2022 | -3.94 | |
| 12 | 2030 | -2.68 | |
| 12 | 2031 | +4.05 | |
| 12 | 2034 | +5.67 | |
| 12 | 2039 | +2.70 | |
| 12 | 2043 | +4.72 | |
| 12 | 2046 | +4.05 | -1.94 |
| 13 | 2051 | -7.17 | |
| 13 | 2053 | -2.68 | |
| 13 | 2054 | -2.74 | |
| 13 | 2056 | +5.40 | |
| 13 | 2059 | 0.00 | |
| 13 | 2061 | -16.42 | |
| 13 | 2063 | +1.35 | |
| 13 | 2074 | 0.00 | |
| 13 | 2075 | 0.00 | |
| 13 | 2076 | 0.00 | |
| 13 | 2078 | -4.05 | |
| 13 | 2079 | +3.78 | |
| 13 | 2083 | -4.05 | |
| 13 | 2087 | +2.70 | |
| 13 | 2091 | -2.70 | |
| 13 | 2096 | +5.40 | |
| 13 | 2099 | -1.35 | |
| 13 | 2100 | 0.00 | -1.25 |
| 14 | 2101 | -2.70 | |
| 14 | 2102 | -5.29 | |
| 14 | 2103 | -5.60 | |
| 14 | 2104 | -2.68 | |
| 14 | 2105 | -2.74 | |
| 14 | 2106 | -2.70 | |
| 14 | 2107 | -2.68 | |
| 14 | 2108 | -5.44 | |
| 14 | 2109 | -11.33 | |
| 14 | 2110 | -5.82 | |
| 14 | 2112 | -5.29 | |
| 14 | 2113 | -5.60 | |

Means 1 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 14 | 2115 | -7.17 | |
| 14 | 2117 | -5.44 | |
| 14 | 2123 | 0.00 | |
| 14 | 2126 | -7.17 | |
| 14 | 2130 | +1.81 | |
| 14 | 2133 | -2.68 | |
| 14 | 2137 | -7.02 | |
| 14 | 2139 | -11.33 | |
| 14 | 2140 | -13.75 | |
| 14 | 2144 | -2.68 | |
| 14 | 2145 | -2.68 | |
| 14 | 2149 | -2.68 | -4.94 |
| 15 | 2151 | -16.42 | |
| 15 | 2152 | -5.44 | |
| 15 | 2161 | -3.26 | |
| 15 | 2166 | -0.08 | |
| 15 | 2167 | +2.36 | |
| 15 | 2176 | -2.68 | |
| 15 | 2177 | +4.05 | |
| 15 | 2182 | -6.64 | |
| 15 | 2184 | -3.94 | |
| 15 | 2186 | -5.44 | |
| 15 | 2187 | -2.68 | |
| 15 | 2189 | -7.17 | |
| 15 | 2191 | -11.33 | |
| 15 | 2194 | -5.44 | |
| 15 | 2195 | +4.05 | |
| 15 | 2198 | -9.49 | |
| 15 | 2199 | -2.68 | -4.25 |
| 16 | 2204 | +7.38 | |
| 16 | 2207 | 0.00 | |
| 16 | 2208 | -2.70 | |
| 16 | 2209 | -2.70 | |
| 16 | 2210 | -12.68 | |
| 16 | 2211 | -2.68 | |
| 16 | 2213 | -7.17 | |
| 16 | 2217 | -7.17 | |
| 16 | 2222 | 0.00 | |
| 16 | 2224 | -9.13 | |
| 16 | 2225 | +4.05 | |
| 16 | 2228 | 0.00 | |
| 16 | 2229 | 0.00 | |
| 16 | 2230 | -10.48 | |
| 16 | 2232 | 0.00 | |
| 16 | 2233 | 0.00 | |

Means 1 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 16 | 2234 | 0.00 | |
| 16 | 2235 | -9.13 | |
| 16 | 2236 | -6.79 | |
| 16 | 2237 | -16.42 | |
| 16 | 2243 | -8.52 | |
| 16 | 2246 | -3.94 | |
| 16 | 2248 | -8.52 | |
| 16 | 2250 | -1.33 | -4.08 |
| 17 | 2255 | -11.33 | |
| 17 | 2257 | -11.33 | |
| 17 | 2261 | -3.94 | |
| 17 | 2263 | -5.44 | |
| 17 | 2267 | -14.03 | |
| 17 | 2268 | +2.70 | |
| 17 | 2271 | -11.45 | |
| 17 | 2277 | -4.35 | |
| 17 | 2281 | -10.78 | |
| 17 | 2286 | -2.68 | |
| 17 | 2287 | +1.35 | |
| 17 | 2289 | +1.35 | |
| 17 | 2300 | +5.40 | -4.96 |
| 18 | 2304 | -5.44 | |
| 18 | 2305 | +2.70 | |
| 18 | 2309 | -13.75 | |
| 18 | 2314 | -2.68 | |
| 18 | 2323 | -2.68 | |
| 18 | 2325 | -13.75 | |
| 18 | 2333 | -8.36 | |
| 18 | 2334 | -2.68 | |
| 18 | 2335 | -2.68 | |
| 18 | 2339 | -2.68 | |
| 18 | 2341 | -2.68 | |
| 18 | 2342 | -2.68 | |
| 18 | 2343 | +4.05 | |
| 18 | 2347 | -11.33 | |
| 18 | 2348 | -2.68 | -4.49 |
| 19 | 2352 | -6.79 | |
| 19 | 2357 | -11.22 | |
| 19 | 2359 | -2.68 | |
| 19 | 2360 | -16.42 | |
| 19 | 2361 | -2.68 | |
| 19 | 2362 | -2.68 | |
| 19 | 2365 | -2.68 | |
| 19 | 2367 | -2.68 | |
| 19 | 2368 | -13.75 | |

Means 1 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 19 | 2371 | +1.35 | |
| 19 | 2375 | +2.70 | |
| 19 | 2380 | 0.00 | |
| 19 | 2381 | -9.43 | |
| 19 | 2382 | -2.68 | |
| 19 | 2383 | -2.68 | |
| 19 | 2384 | -11.33 | |
| 19 | 2385 | -13.75 | |
| 19 | 2386 | -13.75 | |
| 19 | 2387 | -16.42 | |
| 19 | 2388 | -11.33 | |
| 19 | 2389 | -9.13 | |
| 19 | 2391 | +6.75 | |
| 19 | 2394 | -1.53 | |
| 19 | 2396 | -16.42 | -6.63 |

324 cases, -4.36 avg.

Means 1*

*Selected cases in pen center area.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 1 | 1454 | -4.05 | |
| 2 | 1529 | 0.00 | |
| 2 | 1530 | -12.57 | |
| 2 | 1546 | 0.00 | |
| 2 | 1547 | -10.84 | |
| 3 | 1557 | +7.09 | |
| 4 | 1634 | 0.00 | |
| 5 | 1675 | -10.78 | |
| 5 | 1697 | -4.05 | |
| 6 | 1702 | 0.00 | |
| 6 | 1743 | -0.18 | |
| 6 | 1744 | -2.70 | |
| 6 | 1745 | +2.36 | |
| 6 | 1748 | -4.05 | |
| 7 | 1783 | 0.00 | |
| 7 | 1796 | -0.38 | |
| 7 | 1797 | -6.75 | |
| 8 | 1813 | -1.35 | |
| 8 | 1817 | -4.05 | |
| 8 | 1846 | -1.35 | |
| 8 | 1847 | -13.18 | |
| 9 | 1878 | +3.15 | |
| 9 | 1882 | -2.70 | |

Means 1* Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 10 | 1927 | -12.13 | |
| 11 | 1972 | -1.35 | |
| 11 | 1974 | 0.00 | |
| 11 | 1975 | 0.00 | |
| 12 | 2043 | +4.72 | |
| 13 | 2074 | 0.00 | |
| 13 | 2075 | 0.00 | |
| 13 | 2076 | 0.00 | |
| 13 | 2078 | -4.05 | |
| 13 | 2091 | -2.70 | |
| 16 | 2207 | 0.00 | |
| 16 | 2208 | -2.70 | |
| 17 | 2277 | -4.35 | |
| 17 | 2281 | -10.78 | |
| 19 | 2380 | 0.00 | |
| 19 | 2381 | -9.43 | -2.80 |

39 cases, -2.80 avg.

Means 2

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 3 | 1592 | -7.17 | -7.17 |
| 5 | 1651 | -7.17 | |
| 5 | 1658 | -7.23 | |
| 5 | 1660 | -7.23 | |
| 5 | 1662 | +1.51 | |
| 5 | 1669 | -4.39 | |
| 5 | 1682 | -7.23 | |
| 5 | 1683 | +1.35 | |
| 5 | 1684 | +1.51 | -3.61 |
| 6 | 1725 | -4.37 | |
| 6 | 1726 | +2.70 | |
| 6 | 1749 | -6.85 | -2.84 |
| 7 | 1769 | +1.35 | |
| 7 | 1770 | +1.57 | |
| 7 | 1774 | +1.77 | +1.56 |
| 8 | 1848 | +4.05 | +4.05 |
| 9 | 1856 | -2.68 | |
| 9 | 1867 | +2.70 | |
| 9 | 1899 | +2.70 | +0.91 |
| 10 | 1909 | -7.17 | |
| 10 | 1929 | +1.35 | |
| 10 | 1944 | -4.03 | |
| 10 | 1947 | -7.17 | -4.26 |
| 11 | 1962 | +4.05 | |

Means 2 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 11 | 1987 | -7.17 | |
| 11 | 1988 | 0.00 | -1.04 |
| 12 | 2011 | -7.17 | -7.17 |
| 14 | 2118 | +4.05 | |
| 14 | 2148 | -6.79 | -1.37 |
| 15 | 2153 | -7.17 | |
| 15 | 2171 | -1.61 | |
| 15 | 2178 | +7.09 | |
| 15 | 2181 | -4.41 | -1.53 |
| 16 | 2218 | +4.05 | |
| 16 | 2240 | -7.17 | |
| 16 | 2245 | -7.17 | |
| 16 | 2247 | -1.33 | -2.91 |
| 17 | 2252 | +1.35 | +1.35 |
| 18 | 2310 | +4.05 | |
| 18 | 2326 | +1.35 | |
| 18 | 2349 | -3.94 | |
| 18 | 2350 | -1.33 | +0.13 |
| 19 | 2397 | +4.05 | +4.05 |

43 cases, -1.75 avg.

Means 3

| | | | |
|---|---|---|---|
| 1 | 1473 | -0.16 | |
| 1 | 1476 | -0.34 | |
| 1 | 1479 | -0.62 | |
| 1 | 1488 | -0.16 | -0.32 |
| 2 | 1517 | -0.16 | |
| 2 | 1519 | -0.16 | |
| 2 | 1527 | -0.06 | |
| 2 | 1534 | -0.16 | |
| 2 | 1536 | -0.38 | |
| 2 | 1544 | -0.42 | |
| 2 | 1549 | -0.16 | -0.21 |
| 3 | 1553 | -0.50 | |
| 3 | 1574 | -0.32 | |
| 3 | 1576 | -0.16 | |
| 3 | 1578 | -0.16 | |
| 3 | 1586 | -0.06 | -0.24 |
| 4 | 1640 | 0.00 | 0.00 |
| 5 | 1695 | -0.06 | |
| 5 | 1700 | -0.16 | -0.11 |

Means 3 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 6 | 1727 | -0.16 | |
| 6 | 1731 | 0.00 | |
| 6 | 1741 | -0.06 | -0.07 |
| 7 | 1752 | -0.42 | |
| 7 | 1758 | -0.06 | |
| 7 | 1763 | -0.16 | |
| 7 | 1765 | -0.38 | |
| 7 | 1781 | -0.06 | |
| 7 | 1791 | -0.16 | |
| 7 | 1794 | -0.16 | -0.20 |
| 8 | 1803 | -0.06 | |
| 8 | 1811 | -0.16 | |
| 8 | 1815 | -0.16 | |
| 8 | 1823 | -0.16 | |
| 8 | 1827 | -0.06 | |
| 8 | 1832 | -0.06 | |
| 8 | 1841 | -0.32 | |
| 8 | 1844 | -0.22 | -0.15 |
| 9 | 1871 | -0.06 | |
| 9 | 1876 | -0.06 | |
| 9 | 1880 | -0.16 | |
| 9 | 1887 | -0.16 | -0.11 |
| 10 | 1921 | -0.64 | |
| 10 | 1923 | -0.16 | |
| 10 | 1930 | -0.16 | |
| 10 | 1932 | -0.48 | |
| 10 | 1936 | -0.06 | -0.30 |
| 11 | 1951 | 0.00 | |
| 11 | 1963 | -0.06 | |
| 11 | 1968 | -0.48 | |
| 11 | 1978 | -0.16 | -0.18 |
| 12 | 2024 | -0.16 | |
| 12 | 2027 | -0.54 | -0.35 |
| 13 | 2057 | -0.06 | |
| 13 | 2065 | -0.42 | |
| 13 | 2067 | -0.34 | |
| 13 | 2093 | -0.06 | |
| 13 | 2097 | -0.16 | -0.21 |
| 14 | 2119 | -0.06 | |
| 14 | 2121 | -0.06 | |
| 14 | 2128 | -0.16 | |
| 14 | 2135 | -0.16 | -0.11 |
| 15 | 2156 | -0.16 | |

Means 3 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 15 | 2159 | -0.64 | |
| 15 | 2164 | -0.06 | |
| 15 | 2168 | -0.16 | |
| 15 | 2179 | -0.06 | -0.22 |
| 16 | 2202 | -0.16 | |
| 16 | 2205 | -0.06 | -0.11 |
| 17 | 2265 | -0.06 | |
| 17 | 2275 | -0.58 | |
| 17 | 2280 | -0.06 | |
| 17 | 2288 | -0.06 | |
| 17 | 2298 | -0.06 | -0.16 |
| 18 | 2329 | -0.52 | |
| 18 | 2331 | -0.16 | |
| 18 | 2334 | -0.06 | -0.25 |
| 19 | 2372 | 0.00 | |
| 19 | 2392 | -0.06 | -0.03 |

78 cases, -0.19 avg.

Means 5

| | | | |
|---|---|---|---|
| 1 | 1483 | -1.53 | -1.53 |
| 2 | 1523 | -0.40 | -0.40 |
| 5 | 1667 | 0.00 | |
| 5 | 1668 | -1.35 | |
| 5 | 1692 | -15.10 | -5.48 |
| 6 | 1713 | -0.12 | |
| 6 | 1720 | -0.14 | -0.13 |
| 7 | 1754 | -1.14 | |
| 7 | 1767 | -1.10 | |
| 7 | 1777 | -2.70 | |
| 7 | 1778 | 0.00 | -1.24 |
| 8 | 1808 | -13.75 | -13.75 |
| 9 | 1851 | -0.12 | |
| 9 | 1874 | -0.34 | |
| 9 | 1884 | -2.68 | -1.05 |
| 10 | 1934 | -1.20 | -1.20 |
| 12 | 2037 | -9.51 | |
| 12 | 2050 | -7.17 | -8.34 |
| 13 | 2077 | -1.35 | |
| 13 | 2092 | +4.72 | +1.69 |
| 16 | 2221 | -4.05 | |
| 16 | 2223 | -6.79 | -5.42 |

Means 5 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 17 | 2296 | -2.11 | -2.11 |
| 18 | 2308 | -13.75 | |
| 18 | 2313 | -6.73 | |
| 18 | 2315 | +5.40 | -5.03 |

26 cases, -3.19 avg.

Means 6

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 1 | 1453 | -2.70 | |
| 1 | 1455 | 0.00 | |
| 1 | 1456 | -5.38 | |
| 1 | 1458 | -2.68 | |
| 1 | 1459 | -5.44 | |
| 1 | 1461 | -16.42 | |
| 1 | 1463 | -11.33 | |
| 1 | 1464 | -5.44 | |
| 1 | 1466 | -2.68 | |
| 1 | 1467 | -11.33 | |
| 1 | 1469 | -5.44 | |
| 1 | 1471 | -11.33 | |
| 1 | 1484 | -12.57 | |
| 1 | 1486 | +2.70 | |
| 1 | 1492 | -3.94 | |
| 1 | 1494 | -16.42 | |
| 1 | 1495 | -7.17 | |
| 1 | 1497 | -16.42 | -7.44 |
| 2 | 1508 | -5.44 | |
| 2 | 1509 | -2.68 | |
| 2 | 1511 | -16.42 | |
| 2 | 1513 | -2.68 | |
| 2 | 1514 | -2.74 | |
| 2 | 1516 | +1.35 | |
| 2 | 1531 | -2.68 | |
| 2 | 1533 | +1.35 | |
| 2 | 1542 | +2.70 | |
| 2 | 1548 | +1.35 | -2.59 |
| 3 | 1560 | +3.78 | |
| 3 | 1563 | -2.68 | |
| 3 | 1564 | -9.13 | |
| 3 | 1565 | -9.19 | |
| 3 | 1566 | -2.68 | |
| 3 | 1568 | -9.13 | |
| 3 | 1570 | +1.35 | |

Means 6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 3 | 1580 | -1.35 | |
| 3 | 1581 | -1.35 | |
| 3 | 1582 | 0.00 | |
| 3 | 1583 | -7.99 | |
| 3 | 1584 | -2.68 | |
| 3 | 1589 | -9.13 | |
| 3 | 1591 | -2.68 | |
| 3 | 1593 | +2.70 | |
| 3 | 1596 | -2.68 | |
| 3 | 1598 | -2.68 | |
| 3 | 1600 | -2.68 | -3.23 |
| 4 | 1602 | -3.94 | |
| 4 | 1603 | -16.42 | |
| 4 | 1608 | -9.13 | |
| 4 | 1610 | -13.75 | |
| 4 | 1612 | -2.68 | |
| 4 | 1614 | -2.68 | |
| 4 | 1616 | +2.70 | |
| 4 | 1619 | -15.10 | |
| 4 | 1620 | -3.94 | |
| 4 | 1623 | -13.75 | |
| 4 | 1625 | -2.68 | |
| 4 | 1627 | -2.68 | |
| 4 | 1628 | -9.13 | |
| 4 | 1635 | -6.64 | |
| 4 | 1639 | +4.72 | |
| 4 | 1642 | -2.68 | |
| 4 | 1643 | +4.05 | |
| 4 | 1647 | -5.40 | |
| 4 | 1648 | -8.68 | |
| 4 | 1649 | -9.13 | |
| 4 | 1650 | -7.17 | -5.91 |
| 5 | 1652 | -9.13 | |
| 5 | 1653 | -7.17 | |
| 5 | 1655 | -9.13 | |
| 5 | 1657 | -2.68 | |
| 5 | 1659 | -2.80 | |
| 5 | 1676 | -9.13 | |
| 5 | 1677 | -7.17 | |
| 5 | 1679 | -9.13 | |
| 5 | 1681 | -2.68 | |
| 5 | 1687 | -4.03 | |
| 5 | 1688 | -9.13 | |
| 5 | 1698 | +1.89 | -5.86 |
| 6 | 1703 | -2.70 | |

Means 6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 6 | 1707 | -2.68 | |
| 6 | 1710 | +1.35 | |
| 6 | 1715 | -2.68 | |
| 6 | 1716 | +2.70 | |
| 6 | 1735 | -2.68 | |
| 6 | 1737 | -2.68 | |
| 6 | 1738 | -5.44 | |
| 6 | 1740 | +6.75 | -0.90 |
| 7 | 1756 | -16.42 | |
| 7 | 1757 | +1.35 | |
| 7 | 1761 | -13.75 | |
| 7 | 1762 | +1.35 | |
| 7 | 1780 | +4.72 | |
| 7 | 1784 | -4.05 | |
| 7 | 1799 | -5.44 | -4.61 |
| 8 | 1809 | -9.13 | |
| 8 | 1818 | -12.68 | |
| 8 | 1822 | +1.35 | |
| 8 | 1830 | -11.83 | |
| 8 | 1837 | -2.68 | |
| 8 | 1838 | -2.68 | |
| 8 | 1839 | +1.35 | -5.19 |
| 9 | 1857 | +8.10 | |
| 9 | 1860 | +1.35 | |
| 9 | 1866 | -3.94 | |
| 9 | 1870 | +1.35 | |
| 9 | 1885 | -7.17 | |
| 9 | 1896 | -2.70 | -0.50 |
| 10 | 1902 | -2.68 | |
| 10 | 1903 | -5.44 | |
| 10 | 1904 | -5.44 | |
| 10 | 1905 | -11.33 | |
| 10 | 1906 | -2.68 | |
| 10 | 1907 | -3.94 | |
| 10 | 1908 | -7.17 | |
| 10 | 1913 | -8.52 | |
| 10 | 1917 | +1.35 | |
| 10 | 1928 | -2.68 | |
| 10 | 1945 | -16.42 | |
| 10 | 1950 | +9.45 | -4.63 |
| 11 | 1958 | -7.17 | |
| 11 | 1959 | -16.42 | |
| 11 | 1960 | -2.68 | |
| 11 | 1973 | 0.00 | |

Means 6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 11 | 1976 | -12.19 | |
| 11 | 1982 | -2.68 | |
| 11 | 1984 | -16.42 | |
| 11 | 1985 | -16.42 | |
| 11 | 1992 | -2.68 | |
| 11 | 1997 | -7.17 | |
| 11 | 1998 | -11.33 | -8.65 |
| 12 | 2001 | -16.42 | |
| 12 | 2002 | -2.68 | |
| 12 | 2003 | -3.94 | |
| 12 | 2004 | -5.44 | |
| 12 | 2006 | -2.68 | |
| 12 | 2009 | -2.68 | |
| 12 | 2010 | -11.33 | |
| 12 | 2012 | -16.42 | |
| 12 | 2014 | -2.68 | |
| 12 | 2016 | -2.68 | |
| 12 | 2019 | -5.44 | |
| 12 | 2021 | -2.68 | |
| 12 | 2023 | +1.35 | |
| 12 | 2029 | -11.22 | |
| 12 | 2038 | -7.17 | -6.14 |
| 13 | 2052 | -3.94 | |
| 13 | 2055 | -7.17 | |
| 13 | 2060 | -8.08 | |
| 13 | 2062 | -5.44 | |
| 13 | 2084 | +4.72 | |
| 13 | 2095 | -11.33 | -5.21 |
| 14 | 2111 | 0.00 | |
| 14 | 2114 | -5.44 | |
| 14 | 2116 | -2.68 | |
| 14 | 2124 | -8.14 | |
| 14 | 2125 | -2.68 | |
| 14 | 2127 | +4.05 | |
| 14 | 2134 | +1.35 | |
| 14 | 2138 | -5.78 | |
| 14 | 2141 | -2.68 | |
| 14 | 2142 | -7.17 | |
| 14 | 2143 | -5.44 | |
| 14 | 2146 | -16.42 | |
| 14 | 2147 | -9.98 | |
| 14 | 2150 | -2.68 | -4.55 |

Means 6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 15 | 2154 | +1.35 | |
| 15 | 2162 | -0.12 | |
| 15 | 2163 | +3.15 | |
| 15 | 2173 | -2.70 | |
| 15 | 2174 | -8.52 | |
| 15 | 2175 | -9.13 | |
| 15 | 2183 | -1.33 | |
| 15 | 2185 | -9.13 | |
| 15 | 2188 | -9.13 | |
| 15 | 2190 | -16.42 | |
| 15 | 2192 | -5.44 | |
| 15 | 2193 | -2.68 | |
| 15 | 2200 | +5.40 | -4.21 |
| 16 | 2206 | -1.35 | |
| 16 | 2212 | -2.68 | |
| 16 | 2214 | -16.42 | |
| 16 | 2215 | -2.68 | |
| 16 | 2216 | -16.42 | |
| 16 | 2231 | -2.68 | |
| 16 | 2238 | -9.13 | |
| 16 | 2239 | -7.17 | |
| 16 | 2241 | -1.33 | |
| 16 | 2242 | -9.13 | |
| 16 | 2244 | -7.17 | |
| 16 | 2249 | -9.13 | -7.11 |
| 17 | 2251 | -7.17 | |
| 17 | 2254 | -3.96 | |
| 17 | 2256 | -1.33 | |
| 17 | 2258 | 0.00 | |
| 17 | 2259 | -4.03 | |
| 17 | 2260 | -9.13 | |
| 17 | 2262 | -9.13 | |
| 17 | 2264 | +4.05 | |
| 17 | 2272 | +2.70 | |
| 17 | 2278 | +5.67 | |
| 17 | 2282 | +4.05 | |
| 17 | 2285 | -11.49 | |
| 17 | 2297 | +7.56 | -1.71 |
| 18 | 2303 | -7.17 | |
| 18 | 2317 | +1.35 | |
| 18 | 2320 | -5.44 | |
| 18 | 2321 | -2.68 | |

Means 6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 18 | 2322 | -5.44 | |
| 18 | 2324 | -11.33 | |
| 18 | 2336 | -13.75 | |
| 18 | 2337 | -16.42 | |
| 18 | 2338 | -2.68 | |
| 18 | 2340 | -16.42 | |
| 18 | 2346 | -5.29 | -7.75 |
| 19 | 2351 | -2.68 | |
| 19 | 2353 | +2.70 | |
| 19 | 2358 | -16.42 | |
| 19 | 2363 | -2.68 | |
| 19 | 2364 | -16.42 | |
| 19 | 2366 | -16.42 | |
| 19 | 2369 | -11.33 | |
| 19 | 2370 | -2.68 | |
| 19 | 2374 | -11.33 | |
| 19 | 2379 | -0.08 | |
| 19 | 2390 | -2.68 | |
| 19 | 2395 | -8.12 | -7.35 |

227 cases, -5.13 avg.

Means 7

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 1 | 1482 | -0.12 | |
| 1 | 1489 | -0.52 | -0.32 |
| 2 | 1537 | -0.74 | |
| 2 | 1545 | -0.78 | -0.76 |
| 4 | 1618 | 0.00 | |
| 4 | 1638 | -0.52 | |
| 4 | 1646 | -0.16 | -0.23 |
| 5 | 1674 | -0.26 | |
| 5 | 1691 | 0.00 | |
| 5 | 1696 | -0.42 | -0.23 |
| 6 | 1701 | -0.52 | |
| 6 | 1712 | -0.06 | |
| 6 | 1723 | -0.32 | |
| 6 | 1728 | -0.52 | |
| 6 | 1732 | -0.36 | |
| 6 | 1742 | -0.42 | -0.37 |
| 7 | 1753 | -0.78 | |
| 7 | 1759 | -0.42 | |

Means 7 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 7 | 1766 | -0.74 | |
| 7 | 1776 | 0.00 | |
| 7 | 1782 | -0.42 | |
| 7 | 1788 | -0.42 | |
| 7 | 1795 | -0.52 | -0.47 |
| 8 | 1812 | -0.52 | |
| 8 | 1816 | -0.52 | |
| 8 | 1824 | -0.52 | |
| 8 | 1833 | -0.42 | -0.50 |
| 9 | 1872 | -0.42 | |
| 9 | 1877 | -0.42 | |
| 9 | 1881 | -0.52 | -0.45 |
| 10 | 1926 | -0.26 | |
| 10 | 1933 | -0.84 | |
| 10 | 1941 | -0.52 | -0.54 |
| 11 | 1952 | -0.36 | |
| 11 | 1964 | -0.42 | |
| 11 | 1971 | -0.26 | |
| 11 | 1979 | -0.52 | -0.39 |
| 12 | 2028 | -0.90 | |
| 12 | 2033 | -0.42 | -0.66 |
| 13 | 2098 | -0.52 | -0.52 |
| 14 | 2129 | -0.52 | |
| 14 | 2132 | -2.68 | |
| 14 | 2136 | -0.52 | -1.24 |
| 15 | 2165 | -0.42 | |
| 15 | 2172 | -0.12 | -0.27 |
| 16 | 2220 | 0.00 | |
| 16 | 2227 | -0.42 | -0.21 |
| 17 | 2266 | -0.42 | |
| 17 | 2276 | -0.94 | |
| 17 | 2284 | -3.10 | -1.49 |
| 18 | 2312 | 0.00 | |
| 18 | 2332 | -0.52 | -0.26 |
| 19 | 2356 | 0.00 | |
| 19 | 2393 | -0.42 | -0.21 |

54 cases, -0.51 avg.

Means 1,4

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 2 | 1521 | -0.18 | -0.18 |
| 3 | 1555 | -0.26 | -0.26 |
| 6 | 1746 | 0.00 | 0.00 |

Means 1,4 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 7 | 1793 | +6.30 | +6.30 |
| 12 | 2041 | 0.00 | |
| 12 | 2047 | +4.72 | +2.36 |
| 13 | 2069 | +2.36 | |
| 13 | 2070 | +3.15 | |
| 13 | 2081 | 0.00 | |
| 13 | 2085 | +9.43 | |
| 13 | 2089 | 0.00 | -0.78 |
| 15 | 2170 | -1.35 | |
| 15 | 2196 | -0.06 | -0.71 |
| 17 | 2294 | -1.99 | -1.99 |
| 18 | 2311 | -1.35 | -1.35 |
| 19 | 2354 | +3.15 | |
| 19 | 2355 | 0.00 | +1.58 |

17 cases, +0.30 avg.

Means 1,8

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 3 | 1554 | +1.80 | |
| 3 | 1556 | -1.97 | -0.09 |
| 8 | 1828 | 0.00 | |
| 8 | 1842 | +1.89 | |
| 8 | 1845 | 0.00 | +0.63 |
| 10 | 1924 | +4.72 | +4.72 |
| 12 | 2042 | -0.06 | -0.06 |
| 13 | 2058 | -0.02 | |
| 13 | 2066 | +1.39 | |
| 13 | 2068 | +1.89 | |
| 13 | 2073 | -0.08 | |
| 13 | 2090 | 0.00 | +0.64 |
| 14 | 2122 | -4.17 | -4.17 |
| 15 | 2169 | +3.03 | +3.03 |
| 17 | 2299 | -8.10 | -8.10 |
| 18 | 2330 | +2.08 | |
| 18 | 2345 | -11.22 | -4.57 |

17 cases, -0.52 avg.

Means 2,3

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 6 | 1722 | -0.06 | -0.06 |
| 7 | 1771 | 0.00 | |
| 7 | 1775 | 0.00 | 0.00 |

Means 2,3 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 8 | 1802 | +6.30 | |
| 8 | 1849 | 0.00 | +3.15 |

5 cases, +1.25 avg.

Means 3,4

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 1 | 1475 | +1.39 | |
| 1 | 1478 | +1.90 | +1.65 |
| 2 | 1543 | +2.89 | +2.89 |
| 3 | 1551 | +1.19 | |
| 3 | 1552 | +1.55 | |
| 3 | 1573 | +1.73 | +1.49 |
| 4 | 1605 | -0.26 | |
| 4 | 1631 | +5.61 | |
| 4 | 1637 | -1.61 | |
| 4 | 1644 | +7.03 | |
| 4 | 1645 | -1.35 | +1.88 |
| 5 | 1671 | +3.62 | +3.62 |
| 6 | 1717 | +1.53 | +1.53 |
| 7 | 1751 | +1.09 | |
| 7 | 1787 | -4.31 | -1.61 |
| 8 | 1840 | +1.41 | |
| 8 | 1843 | +2.30 | +1.86 |
| 9 | 1858 | -2.70 | |
| 9 | 1861 | +1.21 | |
| 9 | 1890 | +1.92 | |
| 9 | 1891 | +2.45 | |
| 9 | 1892 | +3.86 | |
| 9 | 1894 | -1.35 | |
| 9 | 1900 | -6.99 | -0.23 |
| 10 | 1919 | +1.47 | |
| 10 | 1920 | +1.88 | |
| 10 | 1925 | 0.00 | |
| 10 | 1940 | -5.66 | -0.58 |
| 11 | 1966 | +2.30 | |
| 11 | 1967 | +2.83 | |
| 11 | 1970 | 0.00 | +1.71 |
| 12 | 2026 | +1.19 | |
| 12 | 2032 | -0.26 | +0.47 |
| 13 | 2064 | +1.09 | |
| 13 | 2071 | +4.56 | |
| 13 | 2072 | -0.02 | +1.88 |
| 15 | 2158 | +1.41 | +1.41 |

Means 3,4 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 16 | 2226 | -1.61 | -1.61 |
| 17 | 2253 | -6.90 | |
| 17 | 2273 | -2.89 | |
| 17 | 2292 | +2.04 | |
| 17 | 2293 | +2.47 | +0.13 |
| 18 | 2316 | -11.48 | |
| 18 | 2318 | -10.74 | |
| 18 | 2328 | +1.53 | -6.90 |
| 19 | 2377 | +4.56 | |
| 19 | 2378 | -0.02 | |
| 19 | 2398 | +4.66 | |
| 19 | 2399 | -4.37 | +1.21 |

49 cases, +0.37 avg.

Means 4,5

| | | | |
|---|---|---|---|
| 4 | 1632 | -0.02 | -0.02 |
| 5 | 1673 | 0.00 | |
| 5 | 1685 | -0.32 | -0.16 |
| 9 | 1879 | +4.72 | |
| 9 | 1889 | +1.51 | |
| 9 | 1893 | +23.23 | +9.82 |
| 12 | 2048 | -4.05 | -4.05 |
| 13 | 2088 | +3.15 | +3.15 |
| 17 | 2274 | +4.30 | |
| 17 | 2291 | +1.63 | +2.97 |
| 18 | 2301 | -5.56 | |
| 18 | 2327 | +1.57 | -2.00 |

12 cases, +2.51 avg.

Means 4,6

| | | | |
|---|---|---|---|
| 1 | 1481 | -1.41 | |
| 1 | 1487 | +3.15 | |
| 1 | 1500 | +4.72 | +2.15 |
| 2 | 1501 | -4.31 | -4.31 |
| 3 | 1558 | -1.35 | |
| 3 | 1561 | -8.08 | |
| 3 | 1571 | +1.35 | |
| 3 | 1572 | +1.57 | |
| 3 | 1594 | -1.35 | -1.57 |

Means 4,6 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 4 | 1617 | -2.76 | -2.76 |
| 5 | 1672 | +6.30 | |
| 5 | 1690 | -6.75 | |
| 5 | 1699 | +2.36 | +0.64 |
| 9 | 1862 | +1.47 | |
| 9 | 1863 | -1.63 | -0.08 |
| 10 | 1918 | +1.41 | +1.41 |
| 12 | 2035 | -5.46 | |
| 12 | 2040 | +1.89 | |
| 12 | 2044 | -4.11 | -2.56 |
| 13 | 2080 | +6.30 | +6.30 |
| 14 | 2131 | -6.73 | -6.73 |
| 15 | 2155 | +1.35 | +1.35 |
| 16 | 2201 | +7.09 | |
| 16 | 2219 | 0.00 | +3.55 |
| 17 | 2269 | -7.99 | |
| 17 | 2279 | +4.72 | |
| 17 | 2283 | -9.60 | |
| 17 | 2290 | +1.57 | -2.83 |
| 19 | 2376 | +3.15 | +3.15 |

29 cases, -0.45 avg.

Means 5,8

| | | | |
|---|---|---|---|
| 2 | 1502 | -7.35 | |
| 2 | 1520 | +1.25 | |
| 2 | 1522 | -1.49 | -2.53 |
| 5 | 1666 | -1.41 | -1.41 |
| 7 | 1764 | +1.13 | |
| 7 | 1772 | +1.89 | -1.51 |
| 8 | 1804 | -10.91 | -10.91 |
| 9 | 1864 | -1.99 | -1.99 |
| 10 | 1901 | -7.79 | |
| 10 | 1922 | +2.55 | -2.62 |
| 12 | 2045 | -7.06 | -7.06 |
| 13 | 2086 | -11.69 | -11.69 |
| 14 | 2120 | +3.78 | +3.78 |
| 15 | 2180 | -2.70 | -2.70 |
| 17 | 2270 | -9.49 | -9.49 |
| 18 | 2319 | -11.95 | -11.95 |
| 19 | 2400 | -7.41 | -7.41 |

17 cases, -4.16 avg.

Means 6,8

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 2 | 1518 | +1.35 | +1.35 |
| 3 | 1562 | -9.13 | |
| 3 | 1595 | -5.54 | -7.34 |
| 4 | 1606 | -10.39 | -10.39 |
| 5 | 1686 | -1.43 | -1.43 |
| 9 | 1869 | -2.68 | -2.68 |
| 15 | 2157 | +1.45 | +1.45 |
| 16 | 2203 | -5.82 | -5.82 |
| 17 | 2295 | -2.05 | -2.05 |
| 19 | 2373 | -14.03 | -14.03 |

10 cases, -4.83 avg.

Means 7,8

| | | | |
|---|---|---|---|
| 1 | 1474 | +1.33 | |
| 1 | 1477 | +1.49 | |
| 1 | 1480 | +2.37 | +1.73 |
| 2 | 1528 | -0.22 | |
| 2 | 1535 | +1.13 | |
| 2 | 1550 | +1.13 | +0.68 |
| 3 | 1559 | -4.11 | |
| 3 | 1575 | +2.18 | |
| 3 | 1577 | +2.93 | |
| 3 | 1579 | -0.02 | |
| 3 | 1587 | -10.09 | -1.82 |
| 4 | 1633 | -4.43 | |
| 4 | 1641 | -9.40 | -6.92 |
| 5 | 1664 | +1.89 | +1.89 |
| 6 | 1719 | -2.88 | |
| 6 | 1747 | -1.35 | -2.12 |
| 7 | 1792 | +3.76 | +3.76 |
| 8 | 1850 | -4.11 | -4.11 |
| 9 | 1859 | -11.58 | |
| 9 | 1888 | +1.45 | |
| 9 | 1895 | -1.71 | -3.95 |
| 10 | 1931 | +1.03 | |
| 10 | 1937 | -4.27 | -1.62 |
| 11 | 1969 | +4.38 | +4.38 |
| 12 | 2025 | +1.13 | |
| 12 | 2036 | -8.74 | |
| 12 | 2049 | -6.73 | -4.78 |
| 13 | 2082 | -2.70 | |
| 13 | 2094 | -9.65 | -6.18 |

Means 7,8 Cont'd.

| PG. NO. | RCNO | PAYOFF | BLOCK AVG. |
|---|---|---|---|
| 15 | 2160 | -0.90 | |
| 15 | 2197 | -1.57 | -1.24 |
| 18 | 2302 | -8.94 | |
| 18 | 2307 | -2.68 | -5.81 |

33 cases, -2.12 avg.

Means 1,2,4

| | | | |
|---|---|---|---|
| 18 | 2306 | -5.29 | -5.29 |

1 case, -5.29 avg.

Means 2,3,4

| | | | |
|---|---|---|---|
| 5 | 1663 | 0.00 | 0.00 |
| 6 | 1711 | 0.00 | |
| 6 | 1718 | -0.22 | -0.11 |
| 7 | 1773 | -1.41 | -1.41 |

4 cases, -0.41 avg.

Means 2,4,5

| | | | |
|---|---|---|---|
| 5 | 1665 | 0.00 | 0.00 |

1 case, 0.00 avg.

Means 2,4,6

| | | | |
|---|---|---|---|
| 9 | 1868 | -5.29 | -5.29 |

1 case, -5.29 avg.

# A SURVEY OF READ-ONLY MEMORIES

Morton H. Lewin
*RCA Laboratories*
*Radio Corporation of America*
*Princeton, New Jersey*

## INTRODUCTION

Consider the problem of the design of a combinational circuit with $A$ inputs and $B$ outputs, where each of the output variables is given as a Boolean function of the input variables. Such a circuit might be part of the control unit of a digital computer, where the $A$ inputs are the operation code of an instruction, and the $B$ outputs are the signals which directly control the opening and closing of gates throughout the machine to effect an execution of that instruction. The circuit might be a code converter, where the $A$ inputs are an input code (for example, the machine-code of an alphanumeric character); and the $B$ outputs are an output code (for example, the pattern of signals required for a display of that character). The circuit might be a table look-up device, where, for example, the input variables are a code for the numeric value of a given argument, and the output variables are a code for the value of some function of that argument. Finally, the circuit might be considered as a memory, with fixed information stored, where the A input bits are an address, and the $B$ output bits are the word stored at that address. It is called a "read-only" or "fixed" memory if the information stored is not alterable at electronic speeds.

For applications such as those given above, typical values for $A$ and $B$ are sufficiently large and the given Boolean functions are sufficiently complicated that the circuit is normally constructed in two parts as shown in Fig. 1. Thus, most read-only memories are word-organized or linear-select stores.

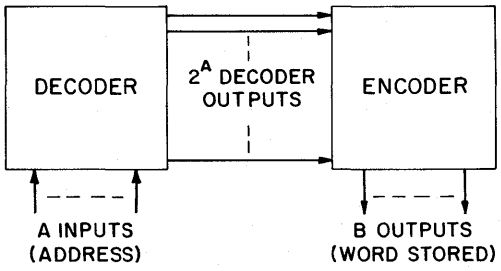



Figure 1. Usual read-only memory structure.

The input address causes only one of the $2A$ decoder outputs to be energized, and the function of the encoder is to selectively couple this signal to the $B$ output lines in accordance with the stored information pattern. The information-bearing portion of the memory, then, can be viewed as a selective signal-coupling device.

Since the design of decoders is well known,[1,2] this paper deals primarily with the structure of the various encoders which have been proposed. One must

bear in mind, however, that the decoding system accounts for an appreciable part of the cost of a typical read-only store. The body of this paper contains qualitative descriptions of a number of memories. Some are early experimental attempts. Others are in developmental stages. Sill others are already in use in operating digital systems. For the convenience of the reader, this kind of information, along with additional data, where applicable, is contained in the annotated references.

## LINEAR ARRAYS

Resistive and capacitive arrays generally have the matrix form shown in Fig. 2. Rows are word lines and columns are bit lines. A given word line signal

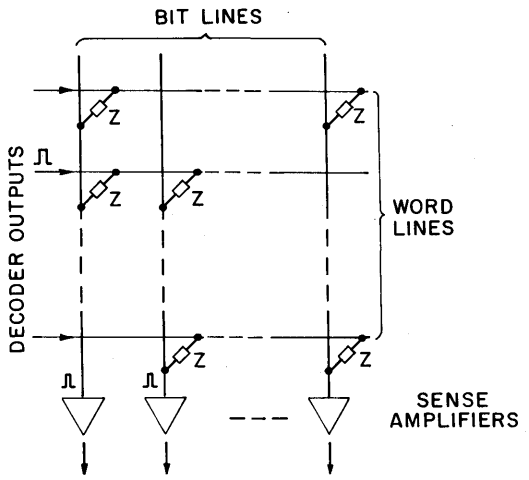



Figure 2. Resistive or capacitive matrix (Z = R or Z = C).

is coupled to a particular column wire if a coupling element is present at the appropriate intersection. Since a signal developed on a column may be coupled to unselected rows, columns are usually terminated in low input-impedance sense amplifiers. The array then acts as a severe attenuater and, for a reasonable capacity, the required word line signals are 30 to 100 volts, while sense signals are in the millivolt range.

### *Resistive Arrays*

Resistive matrices were used as early as 1943 for storage of function tables[3] and were also used in the Eniac machine.[4] Until recently, little attention was paid to such memories. However, with the development of new techniques for the deposition of resistor arrays, interest in these stores has been renewed.[5,6] One approach[5] involves a memory consisting of a stack of paper or plastic cards (with conventional punched card dimensions), each of which contains, on one surface, an interconnected array of silk-screened resistors as shown in Fig. 3. Holes punched in each card by a conventional keypunch have one of two purposes. Some insert infor-

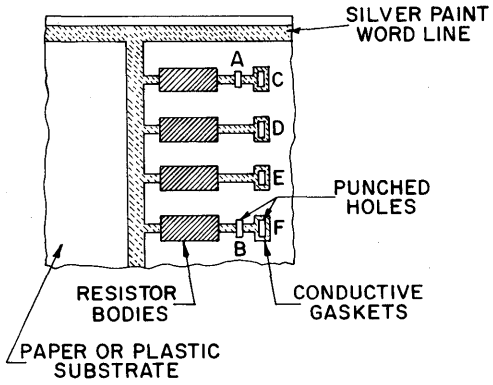



Figure 3. Part of a resistor punched card.

mation by breaking appropriate printed connections (holes A and B in Fig. 3). Others (holes C through F), surrounded by conductive "gaskets", are used to interconnect all cards in a deck. When the cards are stacked, the open channels formed by these holes are filled with a low-temperature molten alloy which later solidifies. This results in the interconnection of all gaskets in any given position. Thus, the common conductors on the cards are the word lines and the alloy columns through the stack are the bit lines. While information is readily inserted into each card, the information stored in a finished stack is not easily changed.

Resistive arrays have an important advantage in that they are direct-coupled systems. On the other hand, for sufficiently large storage capacities and wide resistor tolerances, an appreciable amount of power can be dissipated in a memory stack at full-speed operation.

### *Capacitive Arrays*

Capacitor read-only memories which have been discussed may be divided into two classes—those in which stored information can only be changed by a partial disassembly of the memory stack (involving the breaking and making of a relatively large number of contracts) and those which are designed to allow information change via insertable, low-cost

cards or strips (involving very little, if any, breaking and making of contacts).

In the first case,[7,8] arrays of parallel-plate capacitors may be constructed by appropriate conductive patterns on either side of a thin insulating sheet. Holes are punched, as explained above, to remove certain capacitors from the network, and sheets are interconnected in a stack either by using conventional connectors[8] or by allowing metallized eyelets on the sheets (similar to the gaskets discussed above) to be connected to each other under pressure applied to the stack.[7] The capacitor array can also be constructed using vacuum evaporation techniques to permit thinner insulating layers and, consequently, larger capacities per unit area. The evaporation masks may be designed to already include the required information pattern.

In the second case,[9-12] the capacitor pattern is modified by the presence of a removable card or strip. One approach[9-11] involves the use of two plates facing each other, one containing word lines, the other bit lines as shown in simplified form in Fig. 4. With the plates in close proximity to each other, coupling capacities exist at all intersections. If a thin card, containing an insulated ground foil, is inserted between the plates, all coupling capacities are reduced appreciably by the presence of the shield. If the shield card has a pattern of holes in it, those word and bit lines associated with intersections at which a hole is present will be capacitively coupled. The shield card may be of conventional

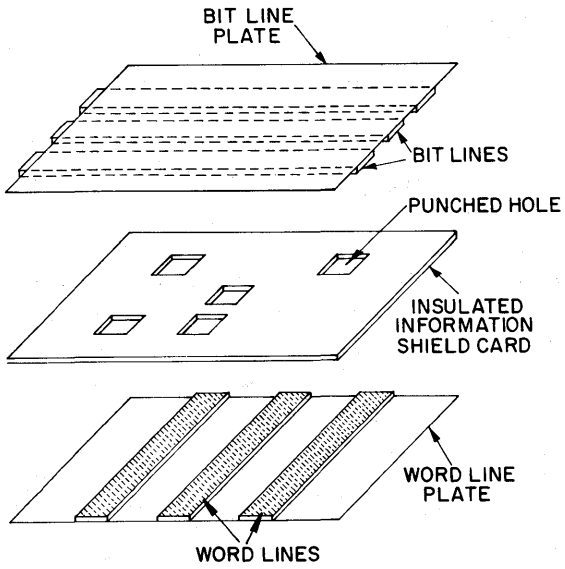



Figure 4. Shielded capacitor memory structure (layers separated).

punched card dimensions, punched by an ordinary keypunch,[9] or it may contain a very thin metal layer with holes formed by a spark discharge.[11] In another approach,[12] the prospective capacitor plates do not face each other. They are etched in the same surface, as shown in Fig. 5. With this structure, very little coupling capacity exists. If a card or strip, containing a pattern of insulated metallized areas, is placed directly over the capacitor plate pattern, coupling capacity will exist only in those positions at which a metal area is present. The coupling capacitor is then composed of the series combination of two equal capacities. No connections need be made to the information-bearing strip. Through-connections (indicated in Fig. 5) are required if word lines and bit lines are placed on opposite sides of the same matrix sheet.

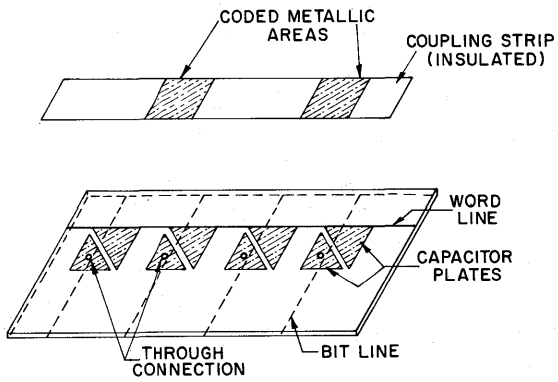



Figure 5. Part of a capacitive array using coded coupling strips (layers separated).

Balanced capacitor arrays have also received attention recently[13] since they permit reflectionless signal propagation in the encoder, when it is operated at very high speed.

In all of the capacitive systems thus far proposed (with the exception of the thin-film, vacuum-evaporated array), practical limits on the effective distance between capacitor plates (particularly in the card-changeable cases) and on the plate areas (if a good packing density is to be achieved) restrict coupling capacity values below 5 picofarads. The memory stacks must be carefully designed to insure that stray coupling capacity is small compared to this.

### *Incudtive Arrays*

In all linear inductive read-only memories, a drive current pulse is passed through one of the

word lines, and this signal is inductively coupled to the sense lines as a function of the stored information pattern. At the outset, these memories have the important advantage of decoder simplicity, since a word line is normally selected by the closing of two "switches," one on either end. Thus, two decoders, each with $2^{A/2}$ outputs, are required in place of one with $2^A$ outputs. This represents a significant reduction in the cost of the selection circuits.

Inductive fixed stores may be divided into three classes—those involving only air coupling, those with open magnetic flux paths, and those with closed magnetic flux paths.

*Air Coupling.* Aircoupled inductive arrays which have been discussed are either relatively fixed (in the sense that stored information is changed only by the use of many-contact connectors or "card-changeable" with no contacts required to the removable card).

An example of the first type[14] is illustrated in Fig. 6. Word lines and sense loops are formed on either side of a thin insulator sheet. At each intersection, the word current originally has two alternate paths. Information is inserted by the breaking of one of these (using an appropriate cutting device, for example). The polarity of the induced signal in the sense loop depends on which current path is taken.

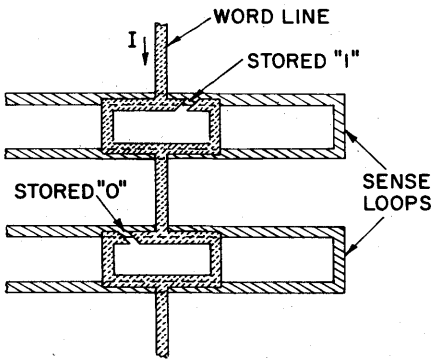



Figure 6. Alternate current path inductive store.

Card-changeable systems have been proposed using cards which act as shields, preventing inductive coupling, and using cards in which induced eddy currents enhance inductive coupling. The first type[15] is the inductive counterpart of the shielded-capacitor card-changeable approach, discussed earlier. Word loops and sense loops are formed on two separate sheets, facing each other, as shown in Fig. 7. An insulated metal shield card containing a pattern of holes is inserted between them. Whenever a hole exists at an intersection, the mutual inductance between word and bit lines is relatively high. Without a hole it is low.

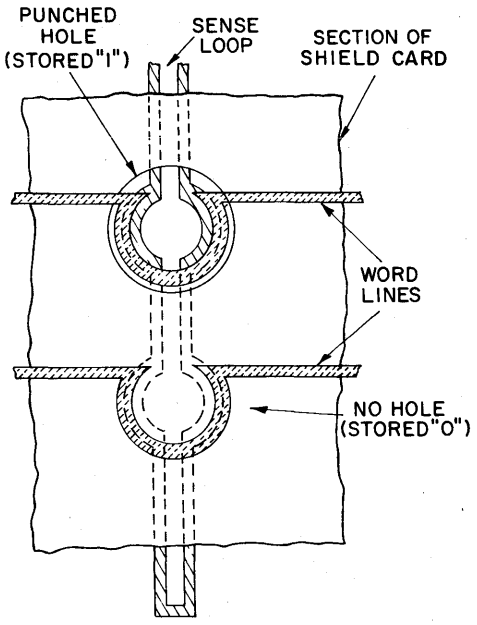



Figure 7. Shielded inductive memory principle.

Eddy-current memories of two kinds have been discussed. In one case,[16] insulated drive loops and sense loops are constructed orthogonal to each other as shown in Fig. 8. Inductive coupling is virtually zero until a small, insulated coupling loop is added at each intersection. When a current is passed thrgouh the word loop, an eddy current induced around the coupling loop will induce a signal in the sense loop. Insertable cards are constructed to include an array of small coupling loops for all intersections. To destroy coupling at a given intersection, the coupling loop is broken with a punched hole, preventing eddy current flow. A modification of this approach[17] uses an insertable card on which each coupling "loop" has one of two geometries (depending on whether the stored bit is to be one or zero. For a given word current pulse, a sensed output pulse may be positive or negative, depending on the geometry of the eddy-current coupling "loop" at the corresponding intersection.

Another eddy-current memory[18] uses nonorthogonal word and sense lines with solid metallic rec-

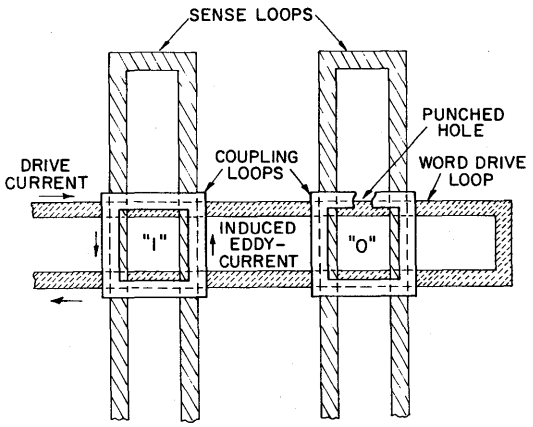



Figure 8. Eddy-current memory principle.

tangles (in which eddy currents are induced) on the insertable cards. Holes punched in these cards partially or completely remove these rectangles. With no punched hole, the inductive coupling is high. Where a hole is punched, it is low.

*Open Magnetic Paths.* Two linear read-only systems using open magnetic coupling paths will be discussed here. The first,[19] used in the Atlas I computer, consists of a woven mesh of insulated word and bit wires as shown in Fig. 9a. At each bit intersection, a small ferrite rod (to increase coupling) is inserted if the stored bit is to be a one. Around each bit intersection, a number of identical ferrite rods ("keepers") are placed, unconditionally, to localize the field and prevent it from returning through other "information rods." To allow for easier change of stored information, the information rods are enclosed in nylon tubes, each twice the length of a ferrite rod, in a balanced winding system shown in Fig. 9b. Each rod has two possible positions in its tube, thus enhancing inductive coupling either in the upper layer or the lower one.

The second open magnetic linear system[20,21] also uses ferrite rods (relatively long ones) as shown in Fig. 10a. Word lines, on cards placed in a stack over the rods, may pass any given rod on one of two sides, as a function of the required stored inforation. When a word line is pulsed, the polarity of the signal induced in the rod sense winding depends on whether the word line passed above or below the rod. The word line path may be determined by a set of punched holes as shown in Fig. 10b.

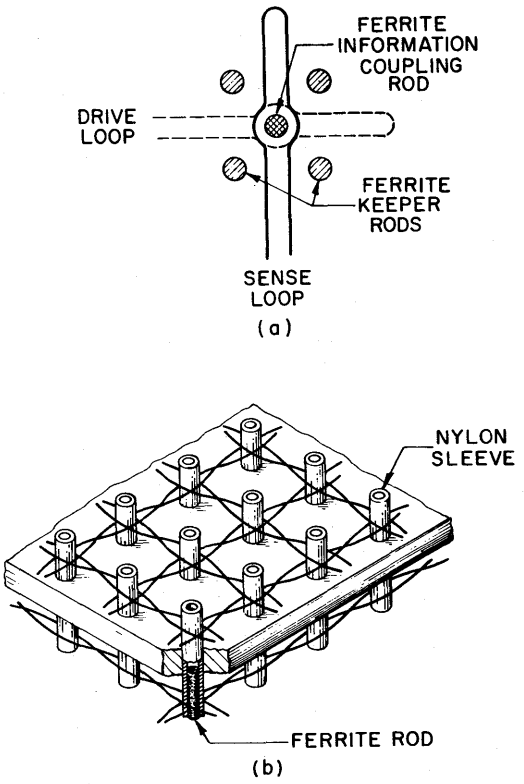



Figure 9. Ferrite rod inductive store.

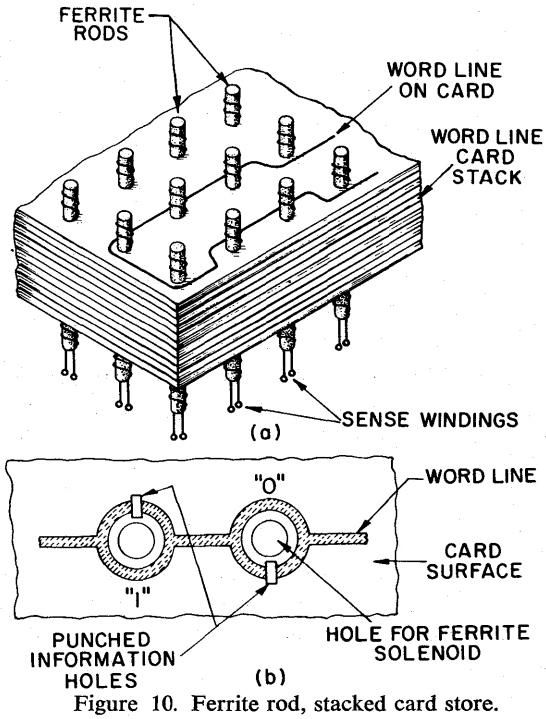



Figure 10. Ferrite rod, stacked card store.

*Closed Magnetic Paths.* Transformer read-only stores of various types[22-27] have been described in the literature. Most operate in the following manner:

Consider a set of $B$ large magnetic cores ($B$ is the number of bits per word), each with a sense winding as indicated in Fig. 11. A given word line threads only those cores in positions corresponding to ones in its word. When a given word line is pulsed, only those sense windings corresponding to store ones develop output signals. A balanced system[28] using two-aperture cores, with a sense winding on the center leg between the apertures, permits positive or negative sense signals to be induced. The word line threads one aperture (stored one) or the other (stored zero).

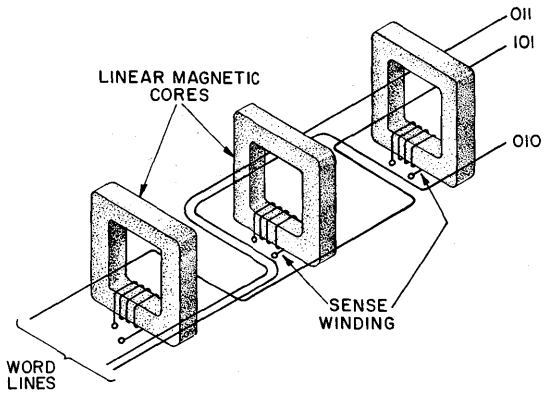



Figure 11. Transformer read-only memory structure.

The closed magnetic path system allows for the development of large output signals (typically 0.5 to 1.0 volt), often eliminating the need for sense amplifiers. It has the disadvantage of not being easily changeable. Usually, one changes a word by disconnecting the old word line (leaving it physically in place) and threading and connecting a new word line. Information may be more easily inserted through the use of thin, word line punched cards or strips, similar to those discussed above for the ferrite rod system. However, the number of word lines which will fit into a given size core aperture is reduced considerably.

*Other.* Many variations on the systems discussed above are possible. In particular, shielded L-C arrays can be used to improve sense signal 1/0 ratios over those achievable with simple shielded L or sielded C approaches.

## NON-LINEAR ARRAYS

Having discussed linear R, L, and C read-only stores, we can proceed to nonlinear systems of the same types. These have the important advantage of eliminating many of the sneak signal paths present arrays[29-30] can be used to improve sense signal 1/0 ratios and word drive requirements. Nonlinear resistive memories which have been proposed are primarily of the diode matrix type. Nonlinear inductive approaches all involve the switching of square-loop magnetic material.

### *Diode Arrays*

Semiconductor diode matrices were discussed in the literature as early as 1949.[31] While diode decoding networks are well known,[1,2] diode matrices as encoding networks were not considered economical, compared to other realizations, until recently. Again, with the advent of batch fabrication techniques, a few diode read-only stores have received attention.

One approach[32] uses the vacuum deposition of organic semiconductor material to form diode arrays on thin, punchable printed circuit cards. Punched holes break appropriate connections, and an encoding matrix as shown in Fig. 12 results. Each bit column is the output terminal of a $k$ -input diode OR gate, where $k$ is the number of words which have a one in its position. Since diodes may also be used in the decoder, it is possible to construct the entire memory (encoder and decoder) using the same fabrication technology, thereby drastically reducing the number of required interconnections. With this approach, however, the information stored is fixed and cannot easily be changed unless a system of connectors is used.

Diode matrices using conventional semiconductor fabrication techniques have received attention recently[33,34] and can be expected to receive more

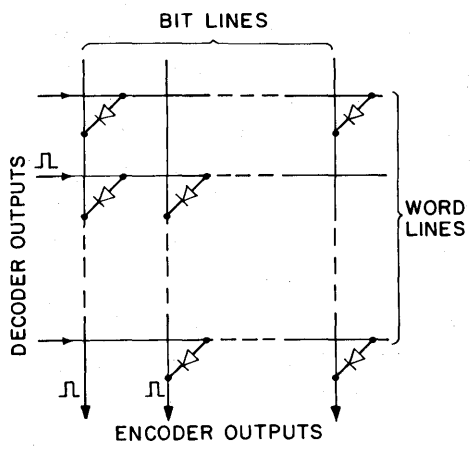



Figure 12. Diode encoding matrix.

attention in the future. Again, although stored information change is not easy, encoders and decoders can be constructed on the same substrate. As tchniques for the encapsulation of these arrays are imporved, semiconductor diode matrices (because of their ideal nonlinear properties) will compete much more favorably in the read-only memory field —particularly for applications where stored information need not be changed.

*Magnetic Switching*

Square-loop magnetic core arrays, permanent magnet-twistor[35] systems and magnetic film arrangements are treated here.

*Magnetic Cores.* The read-only store of the Edsac II[36] utilizes a conventional core selection array, with one core per word and $x$, $y$ and bias windings. Using coincident-current techniques, only the selected core is switched during a memory cycle. There are $B$ sense windings ($B$ is the number of bits per word), each threading only those cores corresponding to words requiring stored ones in its position. Bypassing a core is equivalent to a stored zero. When the selected core switches, only those bit lines threading it develop induced signals. Multiple turns are used to reduce drive current requirements and increase sense signal magnitudes (to 9 volts).

A serial read-only memory technique, utilizing multiaperture ferrite disks, was recently described.[37] One use of the system is illustrated in Fig. 13. A ramp selection current, applied through the center aperture, causes flux switching around the center hole to proceed radially outward as a linear function of time. At a given distance from the center, when switching occurs, if a sense line threads the associated aperture, a signal is induced in it. Thus, information is stored in the threading of the small radial apertures. A number of these can be distributed around the center hole as indicated in Fig. 13. Also, a number of sense wires can thread along the same radius.

*Permanent Magnet-Twistor Arrays.* An important class of read-only memories may be placed under this general heading. The operation of the first versions of such memories[38,39] is explained with reference to Fig. 14. Initially, all twistor segments along the lengths of the sense wires are in a reference state. When a given word is selected (by a core selection switch), current induced in the word strip or word "solenoid" is in such a direction as to acuse all twistor segments immediately inside it to switch. If, however, a small bar permanent magnet (appropriately poled) is in the immediate vicinity of a twistor segment, its switching is prevented. If a twistor segment does switch, a signal is induced in its sense wire. The small permanent magnets are contained on a thin, insertable card so that information can easily be changed.

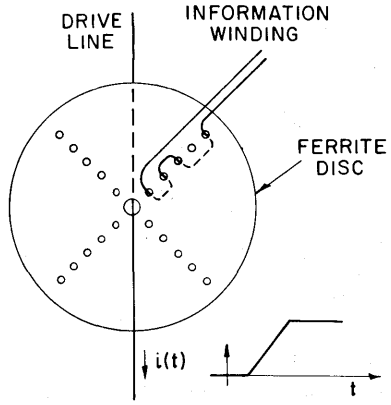



Figure 13. Ferrite multiaperture disk serial storage.

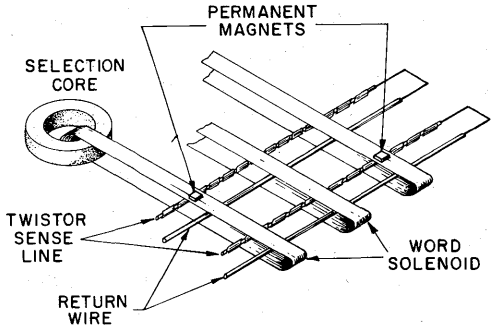



Figure 14. Permanent magnet-twistor store (original structure).

Many modifications and improvements in this arrangement have since been discussed.[40-48] The twistor wires and their return wires need not be inside the word solenoid, but may be outside it, thereby reducing the distance between the permanent magnet and the twistor segment.[40] In this system, eddy currents, induced in the conducting card which holds the permanent magnets, enhance the field produced by the word solenoid to cause twistor switching. Another variation involves making the return wire another twistor wire, either reverse-

wrapped to increase the sense signal[41] or in a two-magnet-per-bit arrangement to achieve bipolar output signals. The insertable cards normally contain magnets in very bit position, some magnetized, some not—depending on the desired information pattern. The pattern on a card may then be easily changed by passing the card under an appropriate set of magnetizing and demagnetizing heads. Another modification[44,45] changes the information insertion method by eliminating the permanent magnets from the changeable card. Instead, holes are punched in the card, in positions where it is desired to prevent local eddy currents from enhancing the switching field. Still another recent approach[46] reverses the roles of the twistor wire and word solenoid, making the twistor wire the word line and the solenoid the sense line. Finally, a recent paper[47] described a system using electroplated continuous magnetic film wire rather than twistor wire.

Many of the detailed varitions between all of thses approaches are beyond the scope of this paper. The reader is referred to the references cited for detailed discussions. It is clear that the activity generated around the original permanent magnet-twistor technique has been great. What is probably the largest capacity, all-electronic, read-only operating store has been constructed using this approach.[48]

*Magnetic Flat Film Arrays.* Systems using permanent magnets and flat magnetic films have also been described.[49,50] In one arrangement, each circular film (electroplated and uniaxially anisotropic), because of its geometry, is normally demagnetized. When a word current is passed through a drive line over the film, it switches, by rotation, into the hard direction, inducing a signal in a sense line. When the drive current disappears, the film is again demagnetized. If, however, a permanent magnet is in its vicinity, the film remains satuarated in the hard direction and switching is prevented. Continuous magnetic films may also be used.

Another flat magnetic film approach[51] replaces the biasing permanent magnet with another (high coercivity) film, which is part of a companion memory array in which information can be written electrically, although relatively slowly. The state of the bias film either prevents or allows switching of the readout film. This type of memory may be considered as not in the read-only class, because information can be written electrically. It is mentioned here because the access time to read the store is much shorter than the write time, to change the information stored.

## OPTICAL SYSTEMS

The basic property of an optical read-only memory is that information is stored as a pattern of opaque or transparent areas on a normally flat surface, such as a card, plate or disk. Storage of information in photographic form[52] permits bit packing densities approach only by the finest magnetic surface recording systems now in use.

Early optical fixed stores[53,54] utilized mechanically selected punched cards, read by sensing holes with a light beam and a photoelectric cell. Systems involving the semiconductor phenomena of electroluminescence[55] and photoconductivity have also been proposed. A matrix, having a series combination of a phototransistor and a diode at each intersection, has also been discussd.[56] The phototransistors act as switches, actuated by a light pattern. The circuit which results is a diode encoding matrix, where connections (stored information) can be changed simply by changing the light pattern. Recent interest in optical read-only stores, however, has been primarily directed at photographic storage systems, because of the high storage densities which may be achieved.

### *Flying-Spot-Stores*

An additional major advantage of some optical systems is the simplification in the selection device (decoder) afforded by using a cathode ray tube. A given word can be selected simply by moving the output light spot to a given position on the tube face. The spot can be moved vary quickly in going from cycle to cycle—hence the term "flying-spot" store.

A number of such approaches have been described in the literature.[57,58,69] One system[58] which has received much attention is outlined in Fig. 15. The CRT face is imaged, by a lens array, as a set of $B$ such areas on a photographic plate, where $B$ is the number of bits per word. Light emitted from a spot on the CRT screen is thus focused into $B$ corresponding sports on the photographic plate. As the CRT spot moves over its allowable area (defined as a square on the CRT face), the $B$ spots on the photographic surface move correspondingly over their allowable areas. Behind each of these areas is a

condensing lens to collect transmitted light through that area and focus it onto a photomultiplier detector tube. Thus, the system consists of one CRT, $B$ imaging lenses, a photographic information plate, $B$ condensing lenses and $B$ photomultiplier tubes. A word is selected by appropriately positioning the CRT spot. In parallel, $B$ spots on the photographic plate are selected. This plate contains information in the form of opaque and transparent dots, so that, within each of the $B$ areas, the light may or may not pass on to the photomultiplier. Thus, a word is read out in parallel. Each of the $B$ areas on the photographic sheet contains $W$ bits, where $W$ is the number of words. It contains the same bit position from each word. The information stored can be changed by changing the photographic plate, being very careful, of course, about plate registration.

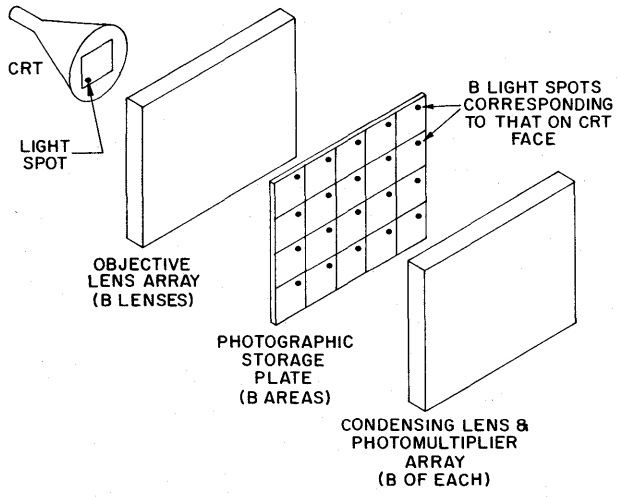



Figure 15. Flying spot store.

An optical system has recently been proposed[60] which replaces the imaging and condensing lens arrays with fiber-optic systems. The light spot is thus distributed to the areas on the photographic information sheet and then collected from the various spots on a given area and transmitted to the photodetector by "light pipes." One may also replace the CRT light source by an $x$-$y$ selected array of light-emitting diodes.

*Photographic Disk Sotre*

Another important development in the field of photographic read-only storage has been the development of a glass disk store,[61] outlined in Fig. 16, for use as a natural language translation dictionary. The disk contains a number of tracks of photographic information in an outer annular ring: A spot from the CRT face is focused as a finer spot on the disk surface by a lens, as shown, with the photomultiplier acting as a detector. Access to the system is similar to that for a conventional magnetic disk store in the sense that, first, a track is selected, and second, information is read serially from it.

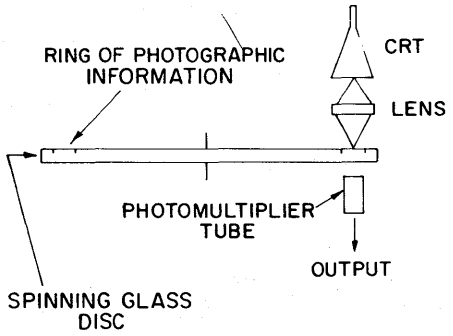



Figure 16. Spinning-disk photographic store.

## COMPARISON CRITERIA

Read-only memories are generally compared by using the conventional memory figures of merit (e.g., storage capacity, cost per bit, speed) and by evaluating the ease with which stored information can be changed.

*Storage Capacity*

The largest capacity read-only stores constructed to date have been the photographic systems. One can expect that, in the future, these systems will continue to be used for fixed, bulk storage. Of the nonoptical systems, the largest capacity store presently in operation is the permanent magnet-twistor system.

*Cost Pet Bit*

While data on costs of various system approaches are difficult to obtain, one can make some qualitative observations. In considering the cost of a fixed store, the effect of the selection system and the sensing system (i.e., the system exclusive of the information encoder) is always of major importance. For example, random-access optical systems, while offering very high storage densities, are normally burdened by complicated and expensive selection

and sensing optical networds. Linear R and C arrays, while simple in construction, normally require large drive signals from a giant decoder and deliver very small sense signals, each requiring a number of amplification stages. This is especially true as the impedance tolerances widen. Magnetic systems, on the other hand, normally require much less decoding complexity. In particular, the transformer store, because it involves closed magnetic paths, delivers in addition large sense signals needing little, if any, amplification. At present it appears to be one of the best present systems, from the cost point of view. It is conceivable that, with the advent of low-cost integrated sense amplifiers, other linear systems will compete more favorably on a cost basis.

*Speed*

It is difficult, at present, to single out any one of the approaches discussed as being the fastest. To date, the highest speed systems have been the linear arrays (excluding the transformer store) and some of the thin magnetic film arrays.

*The Question of Changeability.*

As the discussions in this paper point out, read-only memory systems vary from those in which information stored is more or less absolutely fixed (or at least changed with great difficulty) to those in which information can be changed at slow electronic speeds. An important class of these stores consists of the card-changeable memories. Associated with the question of changeability, but not necessarily included in it, is the question of ease of information insertion. A number of encoder arrays can be set up easily to begin with (using some punching device, for example) but cannot be easily changed subsequently. It has generally been true that, as one adds more flexibility to the system, such as by making it card-changeable, the cost of the system goes up—particularly if high bit packing densities (which require accurate card registration) are involved. In some cases, the cost of the "card holder" becomes comparable to that of a conventional magnetic core memory plane.

The requirements for information changeability vary widely with the application. Fixed, well-defined tables, code conversions are debugged subroutin, for example, may be stored in nonchangeable memories. However, it is probably safe to say that, for most present applications, the stored information pattern is not completely permanent, but will have to be changed after memory construction. If the amount of change is small (for example, as certain program errors are found), a more permanent system which accepts a small number of changes may still be usable. Clearly, however, for many applications the degree of updating required will be such that only card-changeable types will be satisfactory.

## ACKNOWLEDGMENTS

The author wishes to acknowledge the assistance received from previous read-only memory surveys[17,62] in the writing of this paper. In any paper such as this, invariably some pertinent accomplishments are inadvertently omitted and others, which are included, are not given the emphasis they deserve. The author apologizes, in advance, for these shortcomings.

## REFERENCES

NOTE: Notations are added to the references only where they are deemed appropriate. The following abbreviations are used:

IEEETEC: *IEEE Transactions on Electronic Computers*

*Proc. ICMT*: *Proceedings of International Colloquium on Memory Techniques, Paris, 1965* (to be published by Editions Chiron, Paris).

*LCMTCS*: *Large Capacity Memory Techniques, for Computing Systems,* M. C. Yovits, ed., Proceedings of May 1961 Symposium, Macmillan, 1962.

1. Y. Chu, *Digital Computer Design Fundamentals,* McGraw-Hill, 1962, chap. 9.

2. R. S. Ledley, *Digital Computer and Control Engineering,* McGraw-Hill, 1960, chap. 17.

3. J. A. Rajchman, U.S. Patent 2,428,811, "Electronic Computing Device," filed Oct. 1943.

4. A. W. Burks, "Electronic Computing Circuits of the ENIAC," *Proc. IRE,* vol. 35, p. 756 (1947).

5. M. H. Lewin et al, "Fixed Resistor-Card Memory," *IEETEC,* June 1965, pp. 428-434. [Experimental stack of 120 cards, 60 bits each.]

6. C. David, "Fixed Memory with Resistive Coupling," *Proc. ICMT.*

7. L. I. Gutenmakher, *Electronic Information-Logic Machine,* English translation, Interscience, 1963, pp. 41-47. [Developmental. Up to 1024 capacitor sheets in a stack, 3 pf coupling capacity, 40 v drive pulse, 60 mv sense signal.]

8. D. H. Macpherson and R. K. York, "Semi-permanent Storage by Capacitive Coupling," IRETEC, Sept. 1961, pp. 446-451. [Experimental. 576-bit sheets, 5 pf coupling capacity, 20 v 0.5 $\mu$sec drive pulse, 10 mv sense signal, 3$\mu$sec cycle time. Planned—1K words, 34-bits-per-word.]

9. H. R. Foglia et al, "Card Capacitor—A Semi-Permanent, Read Only Memory," *IBM Journal,* Jan. 1961, p. 67. [Experimental. 100 v drive pulse, 1 mv sense signal estimated for $4.8 \times 10^4$ bit store.]

10. J. W. Haskell, "Printed Cards for the Card Capacitor Memory," *IBM Journal,* Oct. 1962, p. 462. [Experimental. 3000 bit array operated at 1 mc.]

11. S. Takahashi and S. Watanabe, "Capacitance Type Fixed Memory," *LCMTCS,* pp. 53-62. [Developmental. 4K words, 50 bits each, 0.2 $\mu$sec cycle time.]

12. J. Van Goethem, "The Capacitance Semi-Permanent Information Store and Its Uses in Telephone Exchanges," *Proc. IEE* (Brit.) vol. 107B supplement 20, p. 346 (1960). [4 pf coupling capacity, 2.5 v drive, 0.16 mv sense for 10K words.]

———, "Operating Principle, Manufacture and Application of a Semi-permanent Memory," *Proc. ICMT.* [$5 \times 10^5$ bits, 4 $\mu$sec cycle time.]

13. D. M. Taub, "Analysis of Sneak Paths and Sence-Line Distortion in an Improved Capacitor Read-Only Memory," *Proc. IEEE,* vol. 51, no. 11, p. 1554 (Nov. 1963).

14. Available under the name "Permacard" from Fabri-tek, Inc., Amery, Wis., in 1920-bit sheets.

15. I. Endo and J. Yamato, "The Metal Card Memory—A New Semi-Permanent Store," *LCMTCS,* pp. 213-230. [Experimental $2 \times 10^4$ bit store. 3 nh coupling mutual inductance. Developing an improved 14,700 word, 32 bits per word, memory.]

J. Yamato and Y. Suzuki, "Forming Semi-Permanent Memories with Metal Card Storage," *Electronics,* Nov. 17, 1961.

16. T. Ishidate, S. Yoshizawa, and K. Nagamori, "Eddycard Memory—A Semi-Permanent Storage," *Proc. EJCC,* Dec. 1961. [Developmental. $4.6 \times 10^4$ bit capacity, 200 ma, 0.05 $\mu$sec drive pulse, 1 mv sense signal, 100 nsec cycle time.]

17. D. M. Taub, "A Short Review of Read-Only Memories," *Proc. IEE* (Brit.), vol. 110, no. 1 pp. 157-166 (Jan. 1963). [Includes a discussion of an experimental "noughts and crosses" eddy-current system using 100 ma drive pulses, 12 mv sense signals, with a 200 nsec cycle time.]

18. A. M. Renard and W. I. Newman, "Unifluxor: A Permanent Memory Element," *Proc. WJCC,* 1960, p. 91. [Experimental 3000 bit store. 500 ma, 150 nsec drive pulse, 10 mv sense signal, 420 nsec cycle time.]

19. T. Kilburn and R. L. Grimsdale, "A Digital Computer Store with Very Short Read Time," *Proc. IEE,* vol. 107B, pp. 567-607 (1960). [Operating store with 4096 words, 52 bits each, involving 50 ma drive pulses, 3 mv sense signals, and 200 nsec cycle time.]

20. D. B. Brick and G. G. Pick, "Microsecond Word Recognition System," *IEEETEC,* Feb. 1964, pp. 27-35. [Developmental. $1.3 \times 10^5$ system with 200 ma drive pulses, approximately 20 mv sense signals and 5.4 $\mu$sec cycle time.]

21. I. R. Butcher, "A Pre-Wired Storage Unit," *IEEETEC,* vol. EC 13, no. 2, pp. 106-111 (Apr. 1964). [Developmental 4K word, 40-bit-per-word-store with 75 ma drive pulses and 1.5 $\mu$sec cycle time.]

22. T. L. Dimond, "No. 5 Crossbar AMA Translator," *Bell Lab. Record,* vol. 29, p. 62 (1951).

23. E. G. Andrews, "The Bell Computer, Model 6," *Proc. 2nd Symposium on Large Scale Digital Calculating Machinery,* Harvard U., 1951, p. 20.

24. J. Goldberg and M. W. Green, "Large Files for Information Retrieval Based on Simultaneous Interrogation of All Items," *LCMTCS,* pp. 63-77.

25. E. L. Younker et al, "Design of an Experimental Multiple Instantaneous Response File," *Proc. 1964 Spring Joint Computer Conference,* pp. 515-528. [Experimental 1K word, 300 bit-per-word information retrieval store.]

26. P. Kuttner, "The Rope Memory—A Permanent Storage Device," *Proc. 1963 Fall Joint Computer Conference,* pp. 45-57. [Developmental 256 word, 64 bit-per-word store with 4 $\mu$sec cycle time.]

27. D. M. Taub and B. W. Kingston, "The Design of Transformer (Dimond Ring) Read-Only Stores," *IBM Journal,* Sept. 1964, pp. 443-459. [Excellent discussion of system operation and design calculations. Stores of the type described are used in some current product line machines.]

28. A. D. Beard, "RCA Spectra 70, Basic Design and Philosophy of Operation," 1965 WESCON, San Francisco, Paper 12.1. [1024 word, 53-bit-per-word modules with 960 nsec cycle time. Up to 4 modules

are operated in an interleaved fashion, to give an effecitve 480 nsec cycle time.]

29. J. Yamato and T. Shimizu, "Large Capacity Metal Card Memory," *Proc. ICMT.* [20 K words, 50 bits each, 5 $\mu$sec cycle time, 3 mv sense signal.]

30. L. A. P. M. De Bot, "Linear Read-Only Memories," *Proc. ICMT.*

31. D. R. Brown and N. Rochester, "Rectifier Networks for Multiposition Switching," *Proc. IRE,* 1949, pp. 139-147.

32. M. H. Lewin et al, "Fixed, Associative Memory Using Evaporated Organic Diode Arrays," *Proc. 1963 Fall Joint Computer Conference,* pp. 101-106. [Experimental. 100 bits operated at 10 $\mu$sec cycle time.]

33. "Thin Silicon Wafers Used in Fixed Digital Memory," *Electronic Design,* Jan. 18, 1961.

34. "Diode Matrix in Integrated Circuit Form," *Computer Design,* June 1965, p. 12.

35. A. H. Bobeck, "A New Storage Element Suitable for Large-Sized Memory Arrays: The Twistor," *Bell System Technical Journal,* vol. 36, pp. 1319-1340 (1957).

36. M. V. Wilkes et al, "The Design of the Control Unit of an Electronic Digital Computer," *Proc. IEEE,* vol. 105B, p. 121 (1958).

37. B. E. Briley, "MYRA—A New Memory Element and System," *Proc. 1964 INTERMAG Conference,* Paper no. 14-8. [Experimental. 64 words, 5 bits each, 2 v sense signals.]

38. D. H. Looney, "A Twistor Matrix Memory for Semi-Permanent Information," *Proceedings Western Joint Computer Conference,* 1959, p. 36.

39. J. H. DeBuske, J. Janik and B. H. Simons, "A Card Changeable Non-Destructive Readout Twistor Store," *Proceedings of the Western Joint Computer Conference,* 1959. Compansion article to ref. 38. [Developmental. $1.3 \times 10^4$ bit store, 1.8 amp 2 $\mu$sec drive pulse, 8 mv sense signal, 5 $\mu$sec cycle time.]

40. W. A. Barrett et al, "A Card Changeable Permanent-Magnet-Twistor Memory of Large Capacity," *IRE Transactions on Electronic Computers,* September 1961. [Developmental. $3.6 \times 10^5$ bit store with 5 $\mu$sec cycle.]

41. E. J. Alexander et al, "A Permanent Magnet Twistor Memory Element of Improved Characteristics," *J. Appl. Physics* (supplement) vol. 33, no. 3, p. 1075 (Mar. 1962).

42. U. F. Gianola et al, "Large Capacity Card Changeable Permanent Magnet Twistor Memory," *LCMTCS,* p. 177.

43. L. W. Stammerjohn, "An Evaluation of Design and Performance of the Permanent Magnet Twistor Memory," *Proc. 1964 INTERMAG Conference,* Paper no. 8-4.

44. K. E. Krylow et al, "Semipermannt Memory —Latest Use for Twistors," *Electronics,* vol. 36, no. 11, p. 80 (Mar. 15, 1963). [7000 bits, 550 ma, 4 $\mu$sec drive pulse, 15 mv sense signal, 10 $\mu$sec cycle time.]

45. J. P. Shuba, "An Eddy Card Twistor Memory for Semi-Permanent Storage," *Poc. 1965 INTERMAG Confeence,* Paper no. 14-2. [2K words, 66 bits per word, 1.2 A drive pulse, 4 mv sense signal, 3 $\mu$sec cycle time.]

46. F. J. Procyk and L. H. Young, "A High-Speed Card-Changeable Permanent Magnet Memory —The Inverted Twistor," *Proc. 1964 INTERMAG Conference,* Paper no. 8-6. [Experimental 512 word, 24 bit-per-word store. 400 ma drive pulse, 10 mv sense signal, 250 nsec cycle time.]

47. U. F. Gianola, "Analysis of Operating Modes for a Read-Only Memory Using Cylindrical Film Sensing of Permanent Magnet Arrays," *Proc. ICMT.* [Less than 100 nsec cycle time.]

48. C. F. Ault et al, "No. 1 ESS Program Store," *Bell System Tech. J.,* vol. 43, no. 5, part 1, p. 2097 (Sept. 1964). [Operating 65K word, 88-bit store. 2A drive pulses, 2.5 mv sense signal, 5.5 $\mu$sec cycle time.]

49. R. E. Matick, "Thick Film Read-Only Memory Device," *J. of Appl. Physics,* vol. 34, no. 4, p. 1174 (Apr. 1963).

50. C. Sie et al, "A High-Speed Thick Film Read-Only Store," *Proc. IFIPS Congress,* New York City, May 1965. [1K words, 288 bits-per-word.]

51. R. J. Petschauer and R. D. Turnquist, "A Nondestructive Readout Film Memory," *Proc. 1961 Western Joint Computer Conference,* pp. 411-425. [Experimental 1024 word, 36 bit memory. Read (access) time: 1.5 $\mu$sec. Write time: 100 $\mu$sec.]

52. G. W. King et al, "Photographic Techniques for Information Storage," *Proc. IRE,* vol. 41, p. 1421 (1953).

53. L. N. Hampton and J. B. Newsom, "The Card Translator for Nation-Wide Dialing," *Bell System Tech. J.,* vol. 32, p. 1037 (1953).

54. T. Kilburn and E. R. Laithwaite, "Servo Control of the Position and Size of an Optical Scanning System," *Proc. IEE,* vol. 101, part IV, p. 129 (1954).

55. G. R. Hoffman et al, "High-Speed Light Output Signals from Electroluminescent Storage Systems," *Proc. IEE,* vol. 107B, suppl. 20, p. 257 (1960).

——— and P. L. Jones, "An Electroluminescent Fixed Store for a Digital Computer," *Proc. IEE,* vol. 109B, p. 177 (1962).

56. H. Hagiwara et al, "The KT Pilot Computer —A Microprogrammed Computer with a Photo-Transistor Fixed Memory," *Proc. IFIPS Congress,* Munich, 1962, North Holland Publishing Co., 1963, p. 684. [256 word, 80-bit-per-word, store.]

57. F. Roberts and J. Z. Young, "The Flying Spot Microscope," *Proc. IEE,* vol. 99, part IIIA, p. 747 (1952).

58. C. W. Hoover and G. Haugk, "The Flying-Spot Store," *LCMTCS,* pp. 79-98. [Operating $2.2 \times 10^6$ bit store. 5 $\mu$sec cycle time.]

59. D. J. Parker et al, U.S. Patent 3,191,157; "Optical Memory," filed Jan. 21, 1960. [Utilization of an optical tunnel with a CRT selection system is discussed.]

H. E. Haynes, U.S. Patent 3,184,732; "Computer Circuit," filed Apr. 15, 1960. [An optical tunnel-fiber optic system is discussed.]

See also "Electro-Optics," brochure of 19 technical papers, Report no. DEP/SCN 009-64-5M, available from Manager, Marketing Services, RCA Defesne Electronic Products, Building 2-6, Camden, N. J.

60. G. R. Hoffman and D. C. Jeffreys, "A Computer Fixed Store Using Light Pulses for Read-out," *J. Brit. IRE,* vol. 25, no. 2, p. 99 (Feb. 1963).

61. J. L. Craft et al, "A Table Look-Up Machine for Processing of Natural Languages," *IBM Journal,* July 1961, pp. 192-203.

62. J. M. Donnelly, "Card Changeable Memories," *Computer Design,* June 1964, pp. 20-30.

# A HIGH-SPEED, WOVEN READ-ONLY MEMORY

H. Maeda
M. Takashima
*TOKO, Inc.*
*Tokyo, Japan*

and

A. J. Kolk, Jr.
*General Precision Inc.*
*Librascope Group*
*Glendale, California*

## INTRODUCTION

The woven, plated-wire memory concept[1] has been shown to provide an economical fabrication technique for the construction of high-speed DRO and NDRO memory arrays. The economies of the woven memory arise from two factors: first, the memory element consists of permalloy-plated, alloy copper wire which is made by an inexpensive, readily controllable, continuous plating process; and second, the weaving technique constitutes a highly automated method for providing the array wiring.

Recently, the woven concept has been extended to the fabrication of very inexpensive, high-speed, read-only memories. This paper describes the fabrication techniques for the preparation of a woven permanent memory matrix and the electrical properties of some of the possible organizations.

Other implementations of read-only memory include the core rope memory[2] and various capacitively or inductively coupled word-organized memories.[3] The woven permanent memory is more similar to the word-organized capacitive or inductive read-only memories than to the rope memory. It appears to have an appreciable speed advantage over other word-organized, read-only memories because of high sense signal output and very small sense line delays due to high packing density.

The applications of read-only memories have been described in other papers [2,3] and will not be reviewed here.

## THE MEMORY ARRAY

The memory element in the woven array consists of a resilient copper alloy wire, 8 mils in diameter, upon which a uniaxially anisotropic, permalloy thin film is plated with a circumferentially directed easy magnetic axis. The magnetic coating is 81Ni19Fe permalloy plated from an aqueous solution containing mainly $NiSO_4 \cdot 7H_2O$, $NiCl_2 \cdot 6H_2O$, $FeSO_4 \cdot 7H_2O$ and $H_3BO_3$.

The memory matrix is fabricated by weaving the permalloy-plated wires as the woof, and 3-mil diameter, insulated, conducting wires as the warp, in the configuration shown in Fig. 1. Selected warp wires are joined together at the ends to provide drive coils of the appropriate number of turns and spacing. The joining of the warp wires can be automated so that this step does not affect the economy or reliability of the fabrication process.

A schematic of the parts of a loom, important to the description of a woven read-only memory, is presented in Fig. 2. In a conventional loom, the warp lines are controlled by the heddles. The hed-

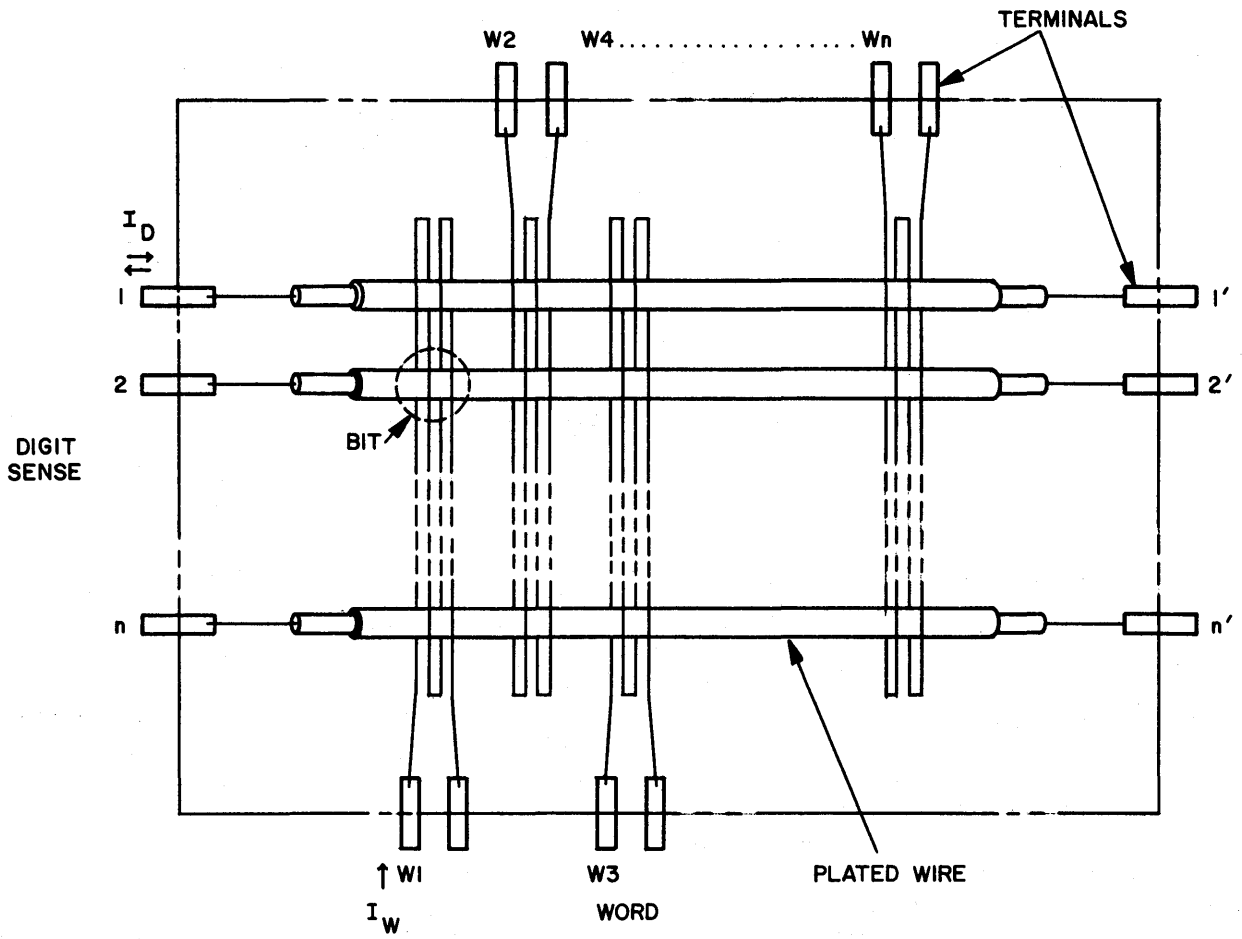



Figure 1. Structure of TOKO memory plane.

dles are connected to bars called harnesses, and upon raising one or more of these harnesses, a group of warp lines are lifted to permit insertion of the woof. A Jacquard loom is a modification which permits individual control of each heddle so that any pattern of warp lines may be selected. The raising of the heddles in a Jacquard loom is activated by punched card input. As a matter of historical interest, this type of loom, which was developed at the start of the 19th century, was the first machine controlled by punched card input.

The weave pattern for a conventional DRO or NDRO woven memory is similar to that shown in Fig. 1. This pattern can be formed by controlling the heddles with only two harnesses, since the weave is constructed by alternately lifting adjacent warp wires and inserting the magnetic woof wire. Several warp wires are then connected in series to form a multiturn word line drive.

One type of read-only memory weave is shown in Fig. 3. This weave, which can only be conveniently formed by using the Jacquard loom, employs the usual woven coil where a "one" is to be read, and bypasses the permalloy-plated wire at a "zero" position by having all the warp wires either above or below the plated wire.

The simplicity of the production method is a key factor in making this variation of the read-only memory economically attractive. Other types of batch-fabricated, word-organized read-only memories utilize some form of punched metallic conductor or printed circuit board to provide a cheap capacitive or inductive-coupling mechanism to store the information patterns. The loom-woven memory array, constructed by using punched cards generated by a computer, can be fabricated at costs only slightly higher than the capacitive or inductive type. In addition, this type of array has a relatively high signal output, high packing density, low sense line delay per bit and other faetures necessary for the con-

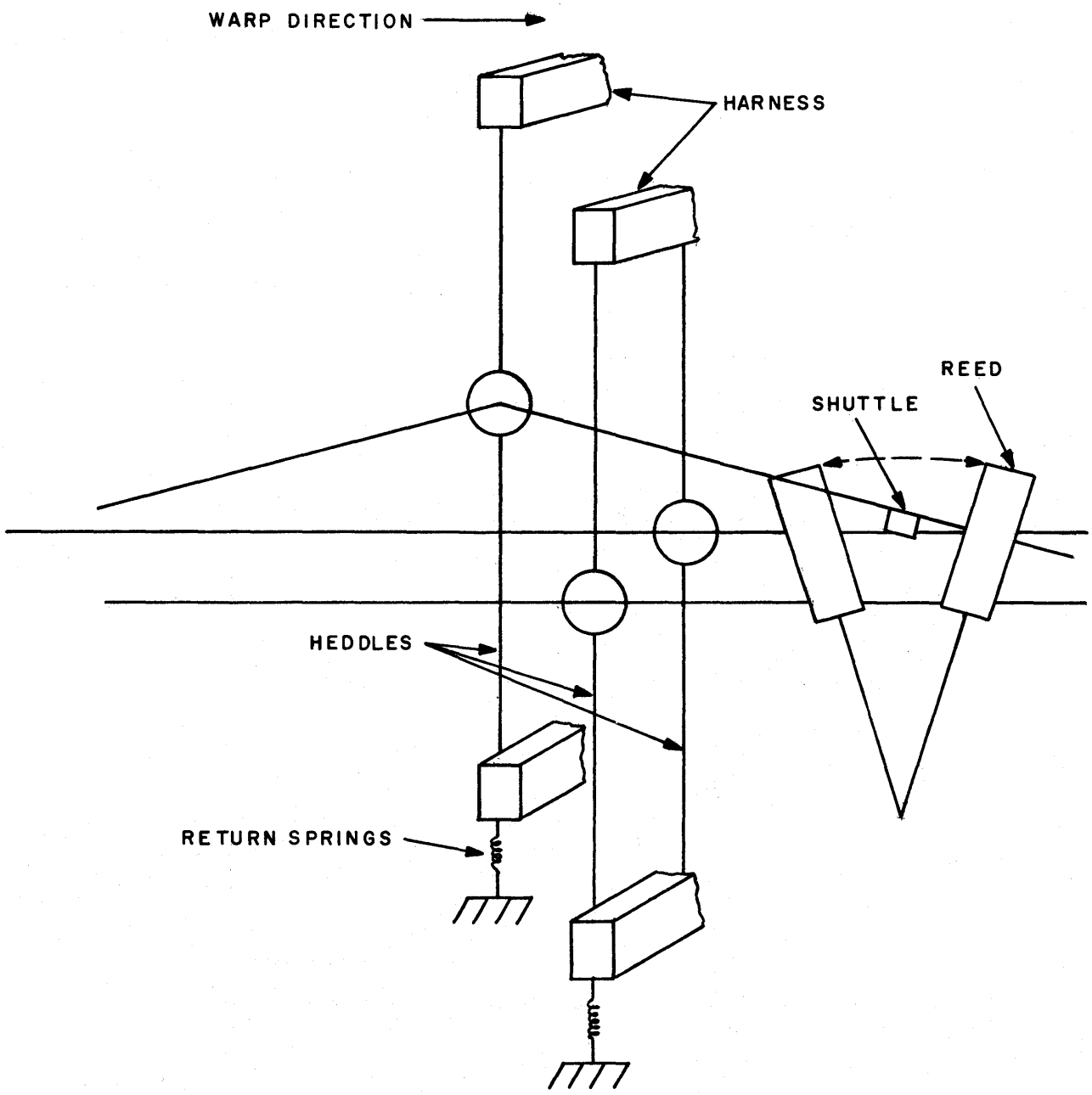



Figure 2. Significant parts of a loom.

struction of a memory with a cycle time in the 0.01-nanosecond range.

## PRINCIPLES OF OPERATION

When used in a normal DRO or NDRO memory array, the permalloy-plated wire is the digit sense line, and the insulated copper wires form the word drive coils. The permalloy-plated wire has a circumferentially directed magnetic easy axis so that the remanent magnetization lies in a closed path around the wire.

The information is written by applying an axially directed field with the word line and an easy direction field down the digit line. The word line field must be removed prior to the digit field. In order to read, a field is applied along the woven word line and the information is sensed on the plated wire.

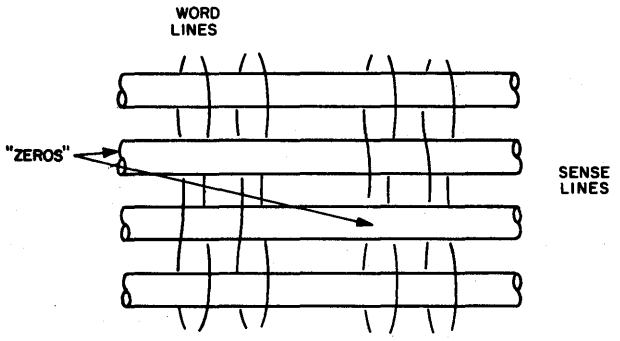



Figure 3. Weave pattern for the read-only memory.

The read output voltage as a function of word write current $I_W$, and digit write current $I_D$, of a typical undisturbed memory bit is shown in Fig. 4.

The read-only memory could be operated in precisely the same manner as the DRO memory using a unipolar digit pulse. In most cases, the preferable mode of operation employs a DC bias to reset the bits using a circuit configuration such as shown in Fig. 5. The DC bias mode has the advantage of permitting very high-speed memory operation since (with the elimination of digit write current delay and the serious noise problem which accompanies the digit write operation) the major factors affecting speed are only the element switching time, sense line delay and recovery, and amplifier delay. Moreover, information is not re-written selectively, so the addressing of the next word can be a least partially accomplished during the time required for the element switching transient to decay. A quantitative discussion of memory speed is contained in the section on memory cycle time.

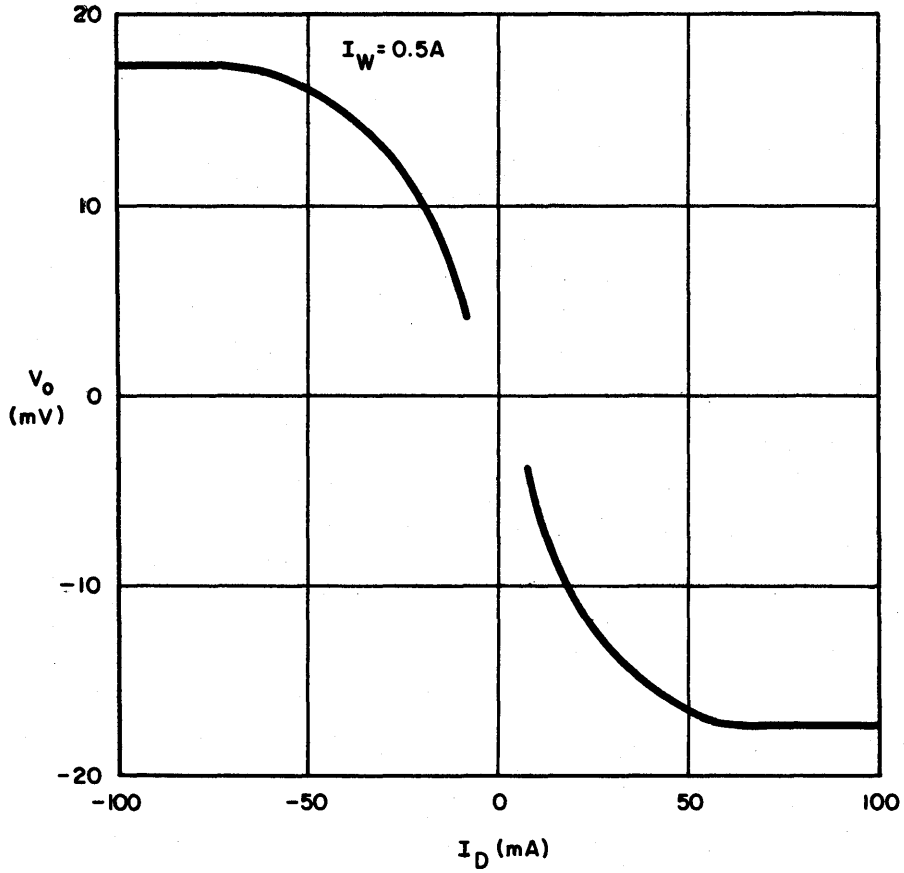



Figure 4. Output voltage vs digit write current for an undisturbed bit in a DRO memory.

The price paid for the DC bias organization is power, but fortunately the losses due to DC bias are only on the order of $10^{-5}$ watts per bit. In memories operated with submicrosecond access times, no power saving would be accomplished by using a pulsed digit current, in fact the DC bias has a power advantage. However, in slow memories operated with bit transfer rates on the order of $10^5$ bits per second, an appreciable power saving can be obtained by using a pulsed digit current to reset the bits.

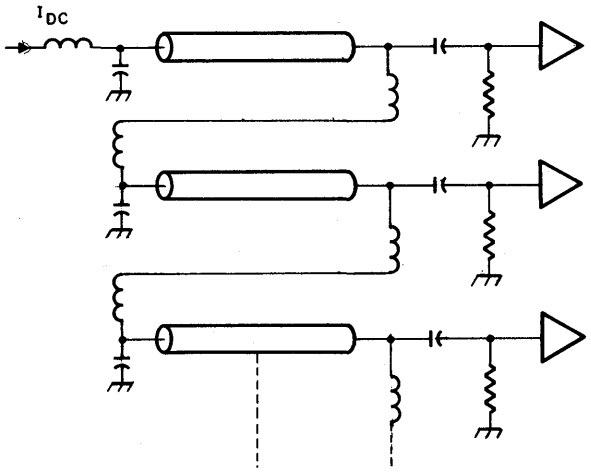



Figure 5. Circuit configuration for a dc-biased, read-only memory.

## MEMORY ELEMENT CHARACTERISTICS

The economical construction and reliable operation of the woven read-only memory depend on the sensitivity of the plated wire to changes in DC bias level, to temperature, and to bending and twisting strains. Another important characteristic is the peaking and switching time of the memory element. These characteristics have been measured and the test results are described below.

The memory element output as a function of drive fields has been measured using the test configuration shown in Fig. 6. The output signal voltage was measured as a function of the DC bias current $I_{DC}$, the word current $I_w$, and the magnetic plating thickness. The effect of the variation of the DC bias level is shown in Fig. 7. The 0.7-micron-thick permalloy

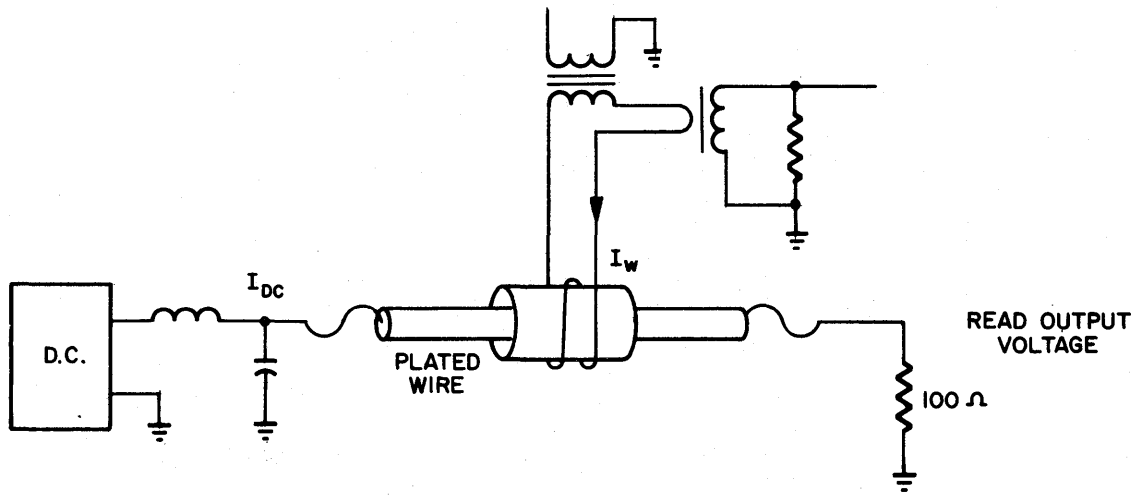



Figure 6. Read-only memory test circuit.

thick plate is seen to peak in output as a bias of about 60 milliamps and then drop. The initial increase in peak voltage with bias current is due to incomplete resetting of the magnetic film at low DC bias levels. The drop in voltage at higher levels of bias occurs because the word drive field no longer completely rotates the magnetization into the hard direction. The 2-micron-thick film shows a much greater output and a lower DC bias requirement than the 0.7 micron. A 5-micron-thick plate has also been tested but in this case the output was no greater than that obtained with the 2-micron film because of the slower switching speed of the thick film.

The read output voltage has been determined as a function of temperature over the range –60°C to +175°C. At a constant word read current of 500 milliamps and a reset current of 50 milliamps, the temperature coefficient of signal output was only 60 microvolts per degree Centigrade. This is sufficiently small so that word drive temperature compensation is not necessary.

The change in coercive force from –60°C to + 175°C has also been measured. The temperature coefficient of coercive force is –0.0027 oersteds per degree Centigrade. This magnitude of the temperature coefficient would not require any change of DC bias level with temperature.

An important factor in the packing density and reliability of a woven plane is the susceptibility of the plated wire to strains introduced by bending. Special looms have been designed to minimize and control the local bending stresses normally encountered in the weaving process. However, in order to minimize cost by maximizing yield in the read-

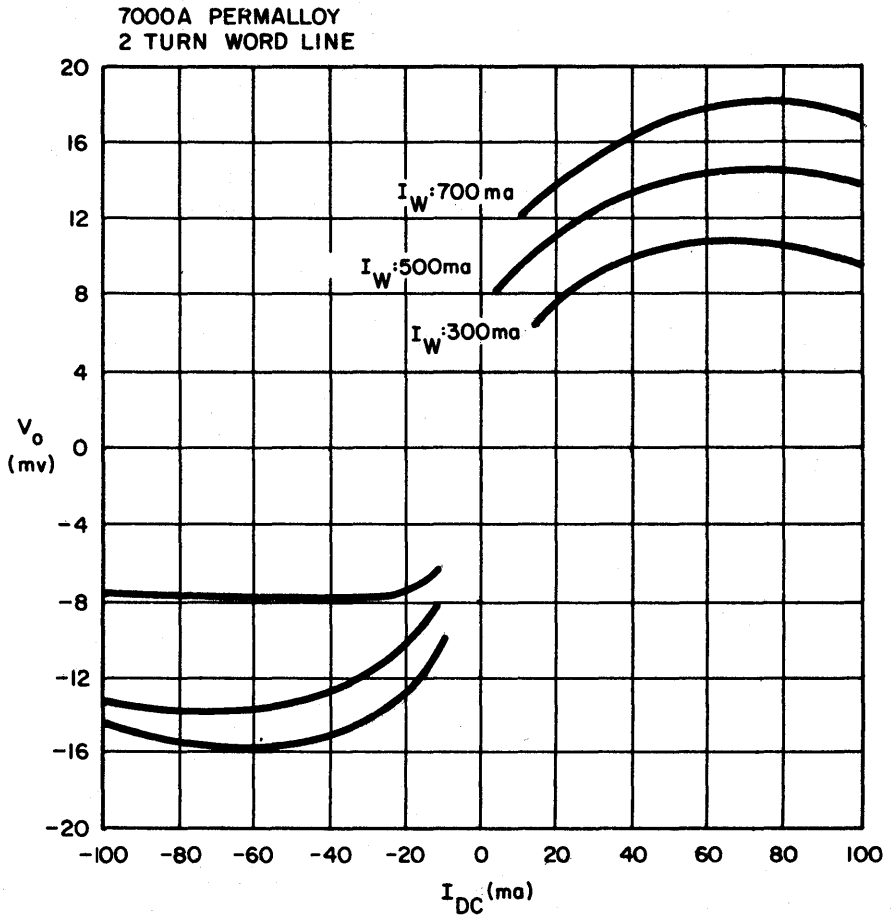



Figure 7. Output voltage as a function of dc-bias current and word drive current.

only memory planes, a relatively high tolerance to strains introduced by the weaving process is desirable. The effect of bending on memory operation was determined using the apparatus shown in Fig. 8. The results of the test are shown in Fig. 9. Within the range of expected bending angles, no serious change of signal level is evident.

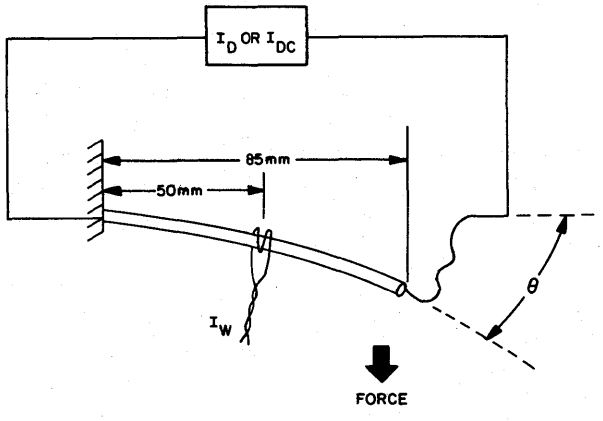



Figure 8. Apparatus for determining the effect of bending on the output voltage.

The effect of twist was also determined using the test fixture shown in Fig. 10. A very great tolerance to twist was obtained as shown by the data presented in Fig. 11.

The peaking time of the memory element is equal to or less than the rise time of the word read pulse at least down to rise times of 40-nanoseconds. The peaking time in the read-only memory is more important than the switching time because the bit is completely reset in every case by the DC bias current during the fall of the word current.

## MEMORY PLANE CHARACTERISTICS

The important characteristics of a memory plane include packing density, sense line delay and attenuation, sense line characteristic impedance, word

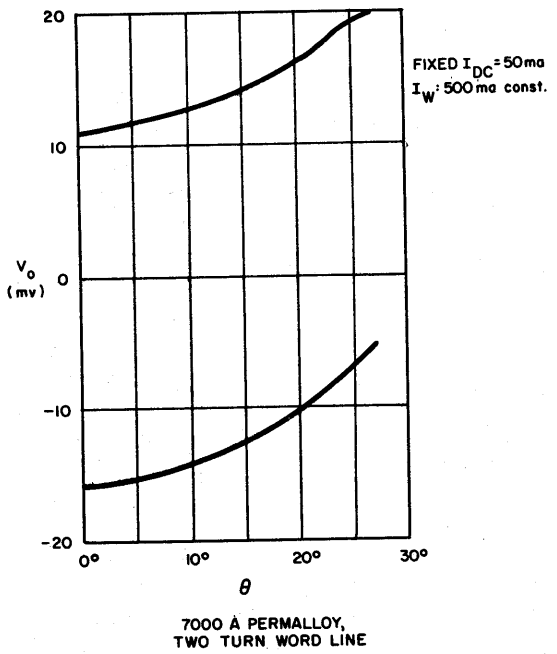



Figure 9. Effect of bending strains on the output voltage in a read-only memory.

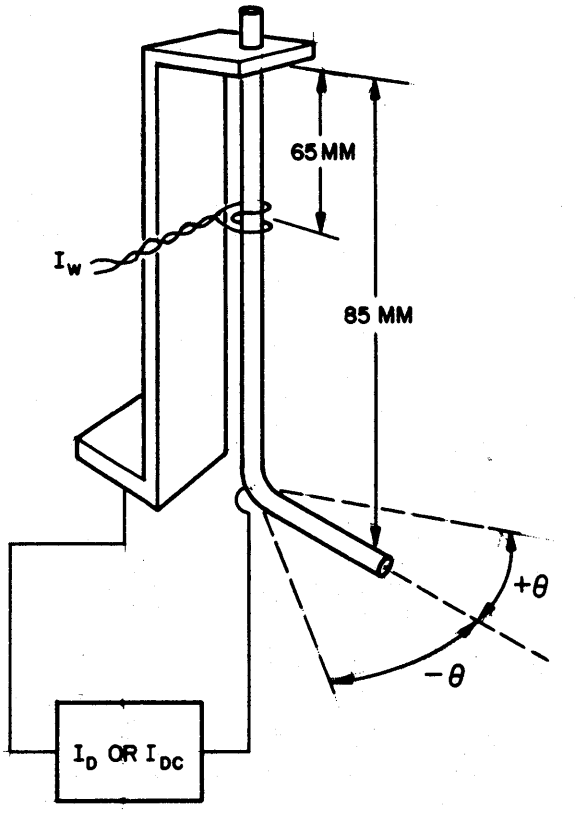



Figure 10. Apparatus for measuring the effect of bending strains.

line inductance and capacitance and finally signal-to-noise ratio.

*Packing Density.* A read-only memory array has been assembled with a packing density of 800 bits per square inch. A photograph of this array is shown in Fig. 12. This packing density was chosen because of compatibility with existing production equipment for DRO memories. It represents a density of 20 bits per inch along the word line and 40 bits per inch along the sense line.

The packing density along the sense line in a DRO plated-wire memory is limited by adjacent bit interactions. There is no such limitation in the permanent memory; the bits can be spaced as tightly as the spacing of the warp lines allows. In the case of a 2-turn drive coil with 3-mil diameter warp line wires, this spacing is 80 bits per inch.

In a DRO or NDRO plated-wire memory, the signal output is also directly related to the bit density along the sense line. This is not true in the read-only memory for the following reasons. If a single turn coil is considered for illustration, it is found that the rotation of the magnetization is spread over an appreciable distance from the coil because of the slow drop-off of the magnetic field due to the demagnetizing fields in the film. In a read-only memory, there is no need to place both turns of a two-turn drive coil on adjacent warp wires, even if every warp wire is used in the word drive lines. The word drive coils can be interleaved as shown in Fig. 13 to share magnetic material between adjacent bits, and thereby, provide a much higher output signal than would be obtained with adjacent coils.

*Electrical Characteristics of the Sense Line.* The delay per bit of the plane shown in Fig. 12 is quite long, 50 picoseconds per bit, because of the large spacing along the sense line and because a 5-micron-thick plating of permalloy was employed.

If every warp wire is used in forming the word so that no warp wires are left for spacing as in the memory plane shown, and if a 2-micron-thick plating of permalloy is employed, the delay per bit along the sense line would be 8 picoseconds per bit.

The woven memory plane has a ground plane beneath the woven mat. The capacitance to ground, and consequently the characteristic impedance of the line, depends upon the position chosen for the

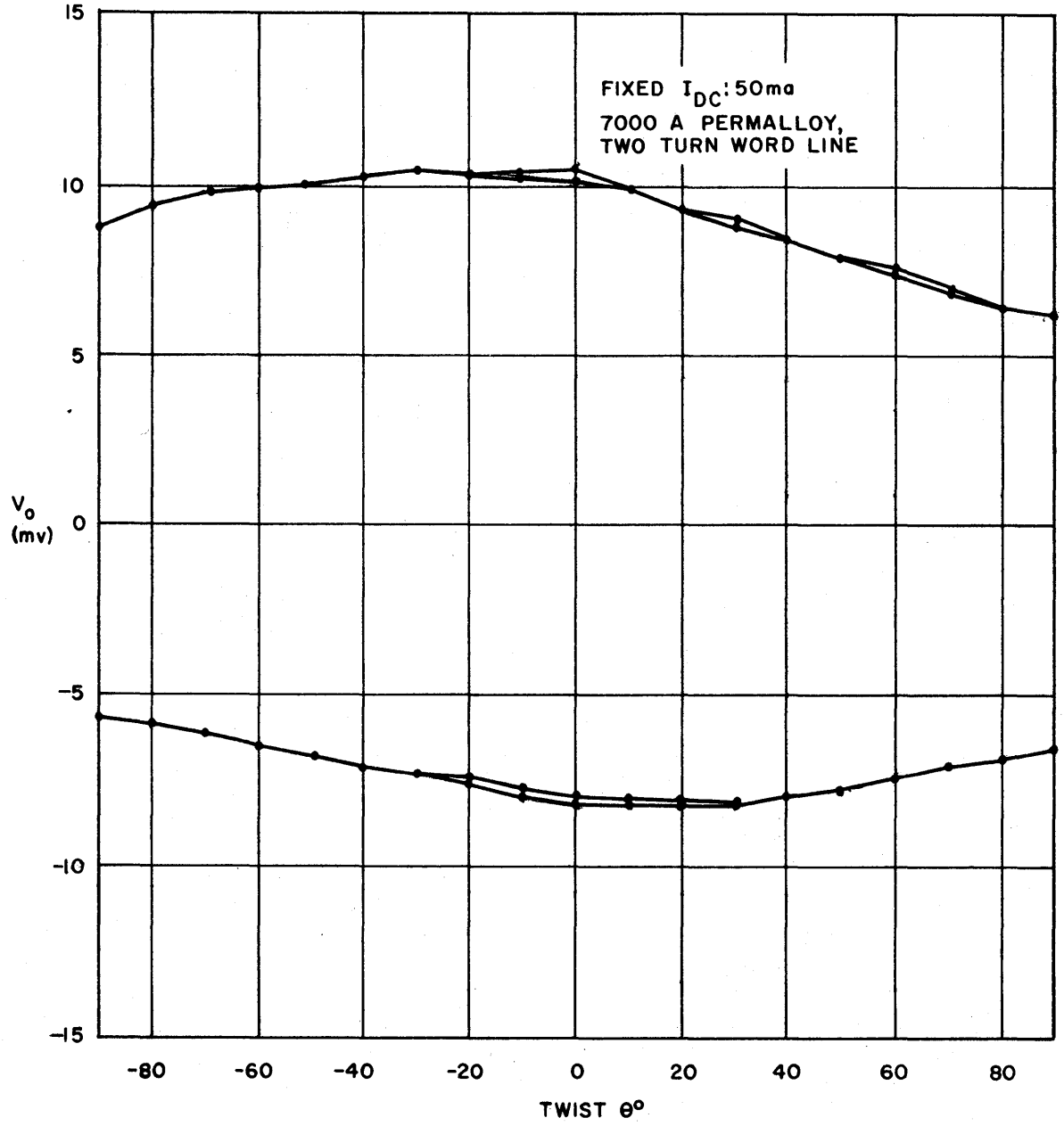



Figure 11. Effect of twist on the output voltage.

ground plane. This can be varied at will but a typical value is about 100 ohms.

*Word Line Characteristics.* The word line can best be considered as an inductance and not as a transmission line. The value of the inductance will vary with the configuration chosen for the word coil but a typical value for a two-turn coil and a 2 micron thick plate is on the order of 10 to 20 nanohenries per bit.

*Signal-to-Noise Ratio.* Figure 14 presents the sense signal output of the experimental read-only memory array shown in Fig. 12. The measured 1-to-0 ratio over the plane was better than 5 to 1. The peaking time for this plane was relatively large because a 5-micron-thick permalloy plate was used.

## TWO-INTERSECTION-PER-BIT ARRAY

An alternative organization for the woven read-only memory plane is shown in Fig. 15. This organization employs two intersections per bit and

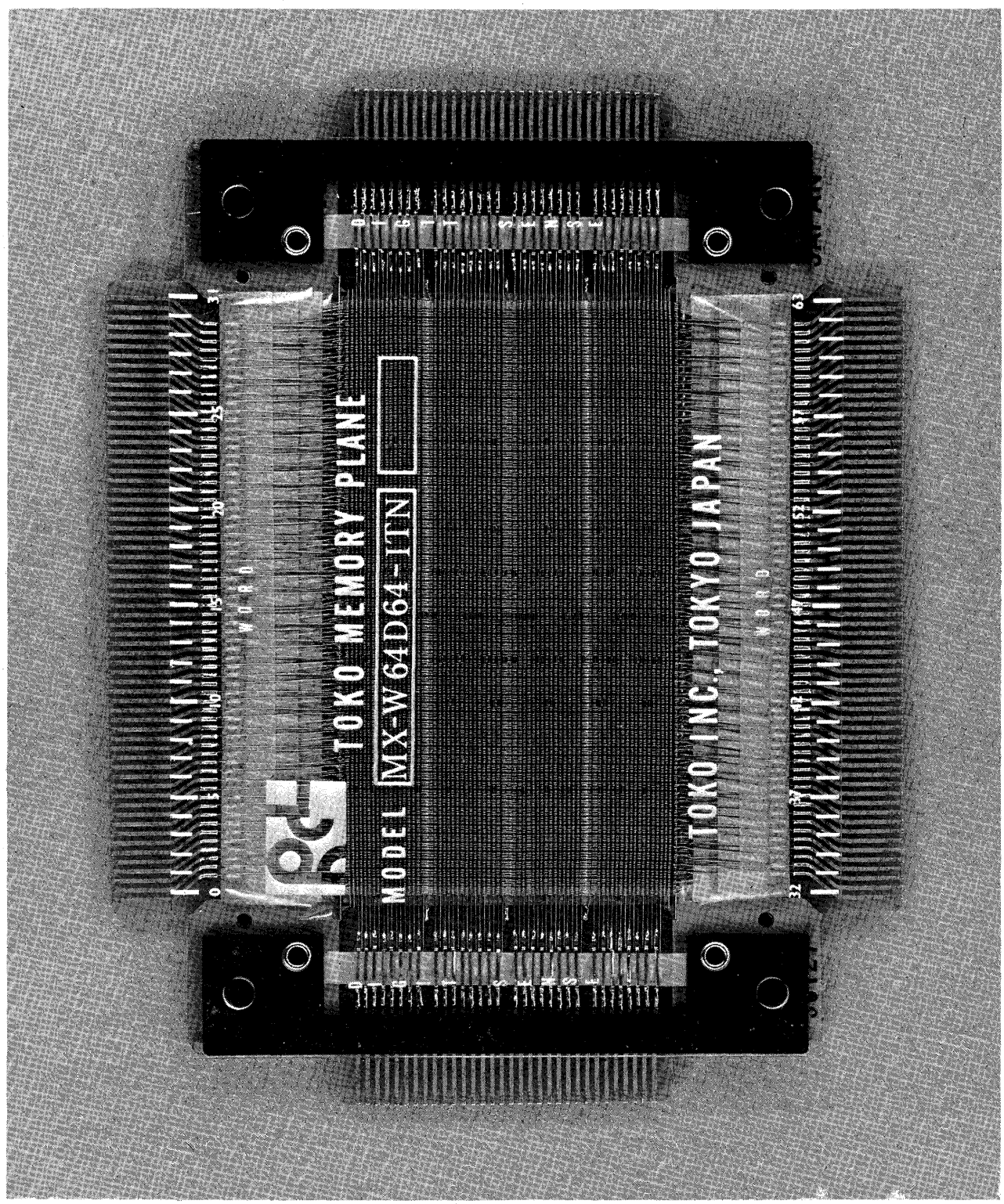



Figure 12. Photograph of the test read-only memory plane.

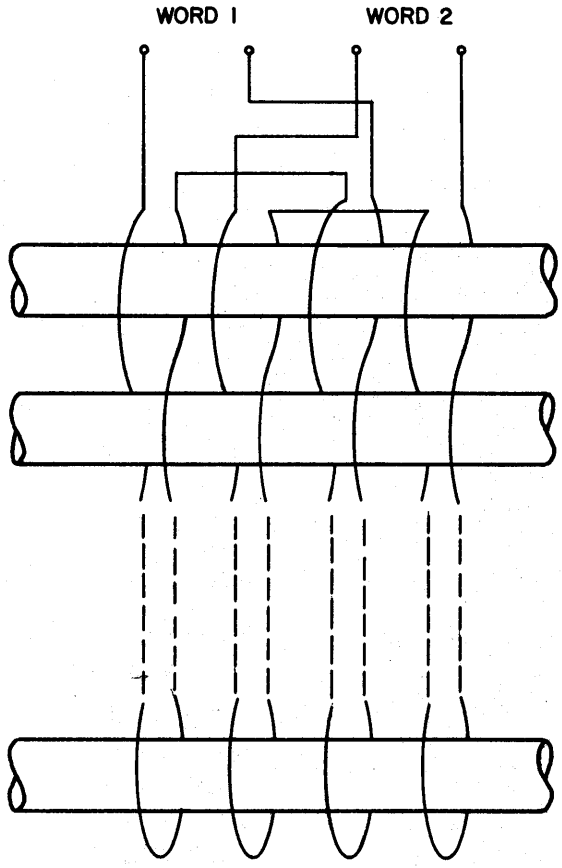



Figure 13. Interleaving configuration for the word drive lines.

yields a bipolar sense output for a "one" and a "zero." It is attractive from the point of view of presenting a constant word line inductance to the driver. It also provides improved discrimination between a "one" and a "zero." In the one-intersection-per-bit organization, a word line could contain all "zeros" and another all "ones." The difference in the back emf of these two lines using a 500 milliamp drive and a 50-nanosecond rise time would be as high as 10 volts for a 64-bit word.

## MEMORY CYCLE TIME

The read-only memory, employing a DC bias, represents a rather simple memory design problem as compared with an electrically alterable memory. The digit write operation, which is one of the principal sources of noise and delay in a memory system, is eliminated. A block diagram of a read-only memory system is shown in Fig. 16.

The factors limiting speed in this system are the delays associated with address decoding, the delay in generating the 500-milliamp word line drive, the transmission delays in the word access circuits and word lines, the peaking time of the element, the transmission delay in the sense line and the delay in the sense amplifier.

The address decoding can be accomplished with a small delay, on the order of 10 to 20 nanoseconds. The setting up of new addresses in the registers can be taking place during the latter part of the reading cycle.

The delay in generating the 500 ma pulse is a problem in view of the word-to-digit line capacitance. This capacitance must be charged in all the paralleled word lines in front of the line diodes. The problem is compounded by the fact that the common mode signal generated by this charging occurs during read time and, unless compensated by careful balancing, will generate an intolerable noise pulse.

There are several ways to handle this problem in a small memory; one is to open the "B" switches

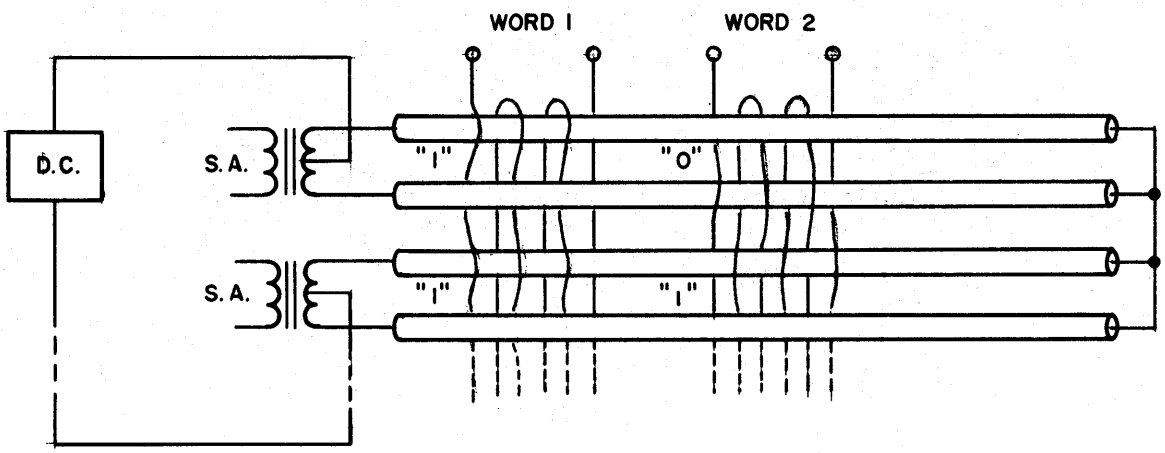
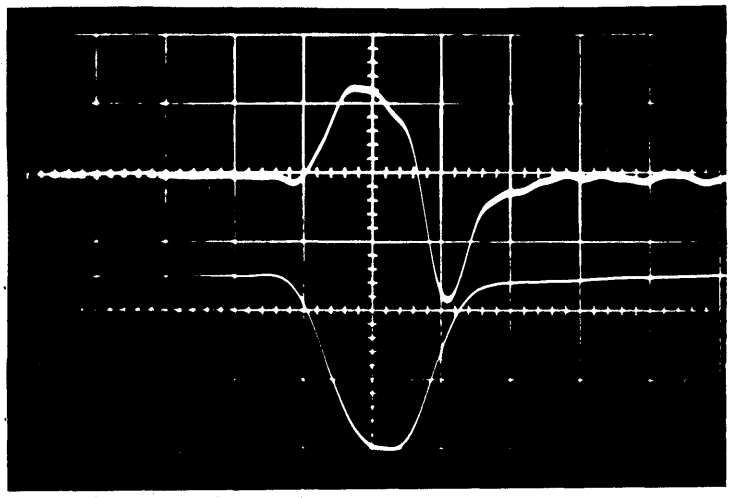
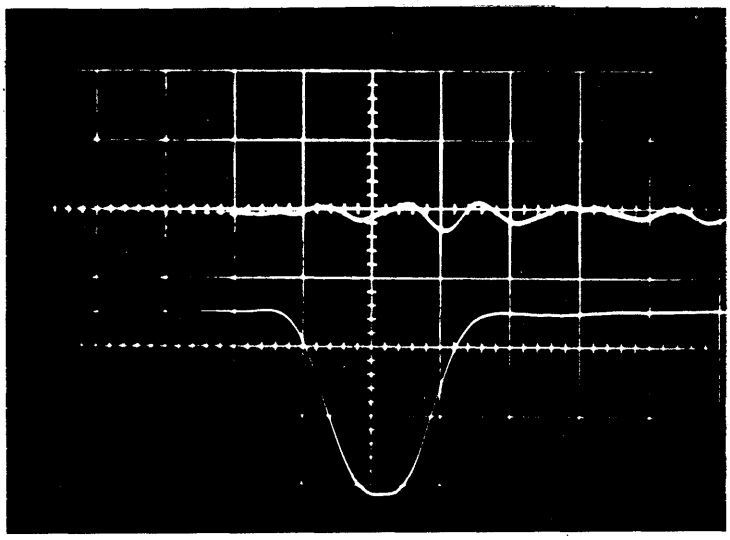



Figure 14. Bipolar, two-intersection-per-bit memory array.



"One" Output Voltage (Upper trace)
20 mv per div
40 ns per div

Word Current
200 ma per div
40 ns per div



"Zero" Output Voltage (Upper trace)
20 mv per div
40 ns per div

Word Current
200 ma per div
40 ns per div

Figure 15. Output signal from a "one" and a "zero" in the test read-only memory plane.

(See Fig. 16) first and provide the required charging current. In a large memory, the problem can be eliminated by providing transformer coupling to the word line. The transformer coupling reduces the charging current and can eliminate common mode noise if proper balancing is employed.

The delay in the word access circuits for the word lines is generally small. However, the word line delay for a 2-turn weave with a 40-mil spacing between bits on the word line is about 50 picoseconds per bit. If unusually long words are employed, this could be a significant delay.

The peaking time of the element is significant in this memory rather than the switching time, since the element will always be reset by the DC bias when the word current is removed. For a 500-milliamp word current at a 50-nanosecond rise time, a nominal value of the peaking time for a 2-micron-thick permalloy plate is 30 nanoseconds.

An outstanding feature of the memory is the small value of the sense line delay. With a 2-turn word drive coil, the sense line delay is only 8 picoseconds per bit. The delay, of course, increases linearly with the number of turns on the word drive coil.

The remaining delay is that of the sense amplifier. A suitable sense amplifier would require a gain of about 200. With presently available transistors, the delay in such an amplifier would be approximately 10 nanoseconds.

# A THIN MAGNETIC FILM COMPUTER MEMORY USING A RESONANT ABSORPTION NONDESTRUCTIVE READOUT TECHNIQUE

M. May, W. W. Powell, and J. L. Armstrong
*Hughes Aircraft Company*
*Culver City, California*

## INTRODUCTION

H. D. Toombs and T. E. Hasty have described a technique utilizing ferromagnetic resonance to obtain nondestructive readout in thin permalloy film memories.[1] A study has been made by the authors to determine how this technique could best be utilized in medium and large sized computer memories. The study culminated in the construction of a 32-word, 24-bit film plane tester utilizing absorption resonance readout which served to provide data on the resonant behavior of various films and to lend practical experience in the design of a resonance memory.

Resonance absorption may be demonstrated by subjecting a ferromagnetic material sample to a steady magnetic field which (viewed classically) sets up an axis of precession for the electron spins responsible for the materials magnetic moment. If a R.F. field is now applied perpendicular to this axis with a frequency near the natural precession frequency given by the Larmor relation $\omega = \frac{geH}{2mc}$ (gaussian units), the spins will begin to precess in sympathy, absorbing energy from the R.F. source. In the present application the steady field is provided by the internal anisotropy field $H_K$ of the material. Due to the shape anisotropy of a thin film, the precession is distorted so that the magnetization vector $M$ remains very nearly in the plane of the film. This fact modifies the Larmor expression so that the resonant frequency is given by $\omega_0 = \frac{ge}{2mc}\ (4\pi M H_K)^{1/2}$ where the subscript on $\omega$ implies that no external field other than the rf field is applied. If an external field $H_{ap}$ is superimposed parallel to the external anisotropy field this expression becomes

$$\omega = \frac{ge}{2mc}\ (4\pi M \mid \vec{H}_K + \vec{H}_{ap} \mid)^{1/2}$$

Thus the resonant frequency shifts above or below $\omega_0$ depending upon whether the magnetization vector $M$ is parallel or antiparallel to $H_{ap}$. It is this phenomena that is used to nondestructively determine the magnetic state of film cell.

Figure 1 shows schematically the configuration of a resonance memory. Power from a uhf oscillator is equally distributed by means of a power splitter over the digit lines. Film cells lie under the digit lines such that their anisotropy axis is perpendicular to the uhf field direction. A small fraction of the power is therefore absorbed by each cell, the remainder proceeding to the end of the line to bias a demodulator detector. The operating frequency is set somewhat below the resonant frequency of the film cells as shown in Fig. 2. Interrogation of the memory is

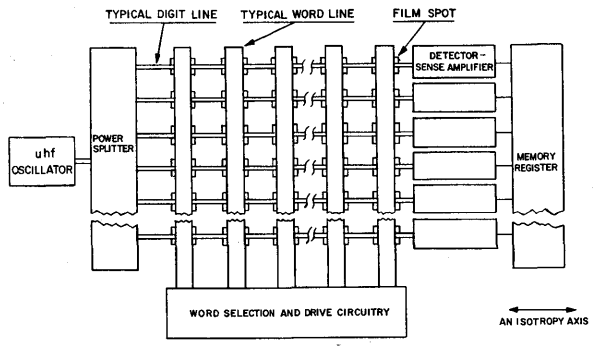



Figure 1. Configuration of resonance absorption memory.

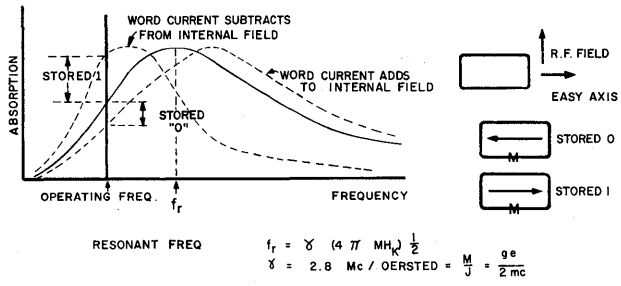



Figure 2. Resonant absorption-frequency characteristics.

accomplished by pulsing a word line thereby reducing or augmenting the absorption depending on the cell's magnetization. This change in absorption is demodulated at the detector so that an output pulse is produced which is a replica of the interrogate pulse with a polarity dependent on film state.

Writing is done by the technique conventional for film memories. Note however that the film axis direction dictates that the role of the word and digit lines be exchanged for the writing operation. This necessitates transposing the data to be stored. Other writing schemes involving separate write conductors to circumvent this difficulty were considered but these were not developed in this study.

## DESIGN CONSIDERATIONS

### *Selection of an Operating Frequency*

From the design viewpoint a low absorption frequency is preferred because this favors oscillator efficiency, detector efficiency, and ease of matching. It was found by experiment that obtaining good control of resonant absorption with an external field requires the use of film with an $H_K$ of 5 oersteds or greater, and a frequency of 550 megacycles or greater. The frequency of 550 mc was satisfactory for the design of a solid state oscillator using a 2N3375 transistor having an efficiency above 40 percent, and also for the design of a transistor detector. A higher frequency and $H_K$ can be used but no advantage was recognized. Operation below film resonance was much preferred to operation above resonance both because of greater signal output and because circuit design problems significantly increased at the higher frequencies near 1,000 megacycles.

Thin film absorption characteristics in terms of signal output versus frequency are shown in Fig. 3. Figures 3a, 3b, and 3c show the need for selection of film having relatively low dispersion (under 2°)

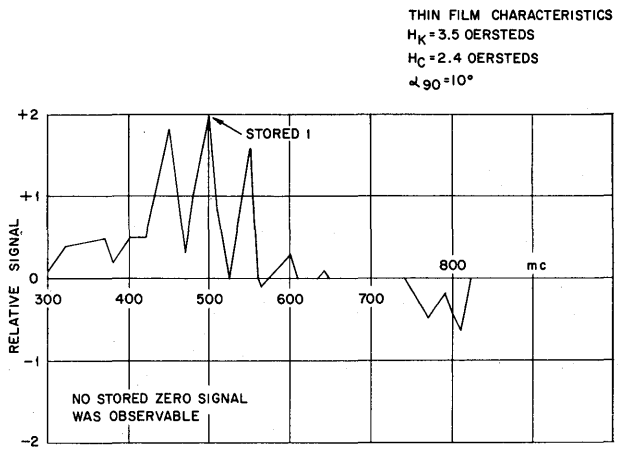



Figure 3a. Resonant absorption readout of 1000Å-thick permalloy, 25 x 50 mil size. Applied field to readout is in direction of stored 1.

and a moderately high $H_K$ of 5 oersteds or greater. The R.F. line was 5.5 mil wide and of 50 ohms impedance.

Figure 3a shows the effect of $H_K = 3.5$ oersteds and dispersion of ($\alpha_{90}$) of 10°. The signal output versus frequency for a stored 1 is shown. Too much creep occurred on this film to show a stored 0 when the applied read field was opposite to the magnetized state of the cell.

Figure 3b shows some improvement for $H_K = 3.8$ oersteds and dispersion of 4°. A signal was obtained for a stored 1 and stored 0.

Figure 3c shows the 1 and 0 output versus frequency for films having $H_K = 5.3$. In this film $\alpha_{90}$ was less than 2°. The externally applied field was generated by passing 300 milliamps through a 60-mil–wide conductor adjacent to the cell in all cases.

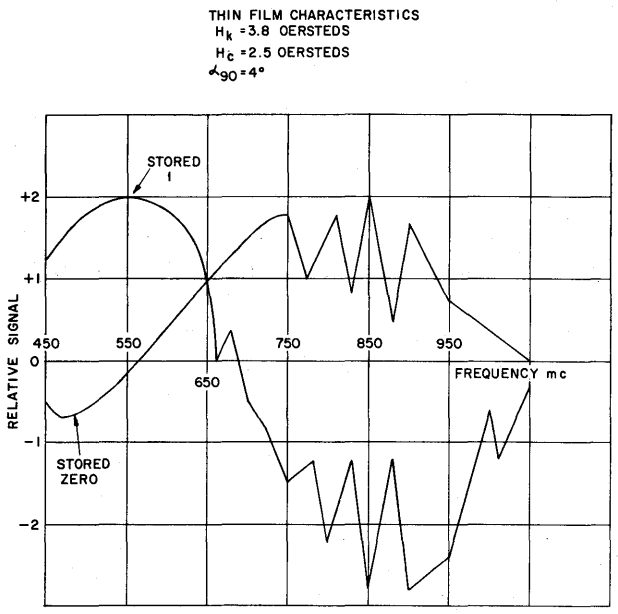



Figure 3b. Resonant absorption readout of 1000Å-thick permalloy, 30 x 60 mil size. Applied field to generate signal is in direction of stored 1.

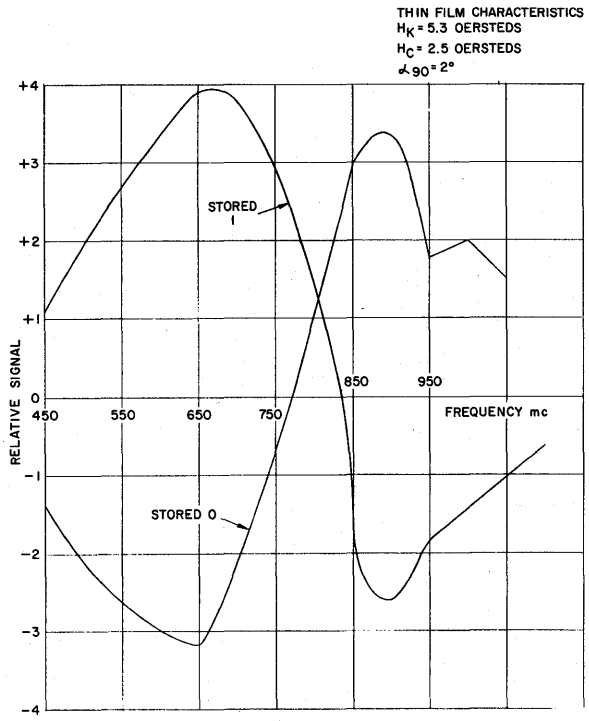



Figure 3c. Resonant absorption readout of 1000Å-thick permalloy, 30 x 60 mil size. Applied field to generate signal is in direction of stored 1.

The reason for the inflections in signal amplitude with increasing frequency for films with low $H_K$ and high $\alpha_{90}$ is not understood by the authors. The line and detector for these measurements had an SWR of less than 1.1 over the frequency range shown. A diode detector was used for these measurements.

## CALCULATING THE SIGNAL MAGNITUDE

Neglecting copper and dielectric losses the power attenuation down a digit line is given by $dP/dn = -kP$ or $P = P_0e^{-kn}$ where $P_0$ is the input power, $n$ is the cell number, and $k$ is the fraction of the power reaching a cell that is absorbed by that cell. If one of the cells is interrogated, its power absorption is changed by $akP$ where $a$ is the fraction of the absorption that can be controlled by the word pulse. Considering the attenuation due to the remaining cells as given above, the power signal at the output is $\Delta P = akP_0e^{-kN}$. This signal is maximized with respect to $k$ when $kN = 1$. Thus to obtain the best power signal the cell size must be adjusted to produce an attenuation down the line (less copper and dielectric losses) of $1/e$ or 4.3 db.

Under this condition the signal power at the last cell where the R.F. lines connect to the detector is $\pm \frac{P_0}{e}\left(\frac{a}{N}\right)$ where $N$ is the total number of cells along the digit line.

The fraction $a$ was measured for a number of cells on a 50-ohm strip line. A power meter monitored the output and input. A large cube coil placed over the line applied a field of 0, +1.4 or –1.4 oersteds. This was the largest field that together with the R.F. would not permanently change the stored magnetization of the memory cells. R.F. level was 10 milliwatts. With the data adjusted for $1/e$ total attenuation, the power ratios measured were:

| Frequency | Line Loss | Variation in Line Loss with Applied Field |
|---|---|---|
| 450 mc | 4.3 db | –0.81 db |
| | | +0.38 |
| 550 | 4.3 | –1.17 |
| | | +0.58 |
| 640 | 4.3 | –1.53 |
| | | +0.76 |

It was found that 10 milliwatts peak power at 550 megacycles was near a practical maximum for the 5.5-mil–wide 50-ohm R.F. line used. Above this value the stored information might be lost due to creep on readout. This power provided a peak calculated R.F. field of close to 1 oersted. The limit

on power is influenced by the R.F. line dimensions and the properties of the thin film.

The power at the output with 10-milliwatt input would be

$$\frac{0.010}{e} \text{ watts} \pm \text{the signal power}$$

If all the memory cells at 550 mc could provide a −1.17 db change, then each cell would provide $-1.17/n$ db change of the output power.

For 32 cells used in the memory film substrate transistor was used in place of a diode for detection tester described later, a −1.17/32 or 0.0365 db change could be expected, assuming the cells were adjusted so that 32 cells provided 4.3 db attenuation. In practice 30 × 60 mil cells 1,000 A thick with $H_K = 5.3$ will closely achieve this. The R.F. level change in a 50-ohm line using 10 milliwatts into the input would be 0.43 volts +1.55 millivolts. This modulation was confirmed using a General Radio detector on the above-mentioned film tester.

Figure 4 is a circuit diagram of the detector and signal amplifier used on the film tester. A 2N918

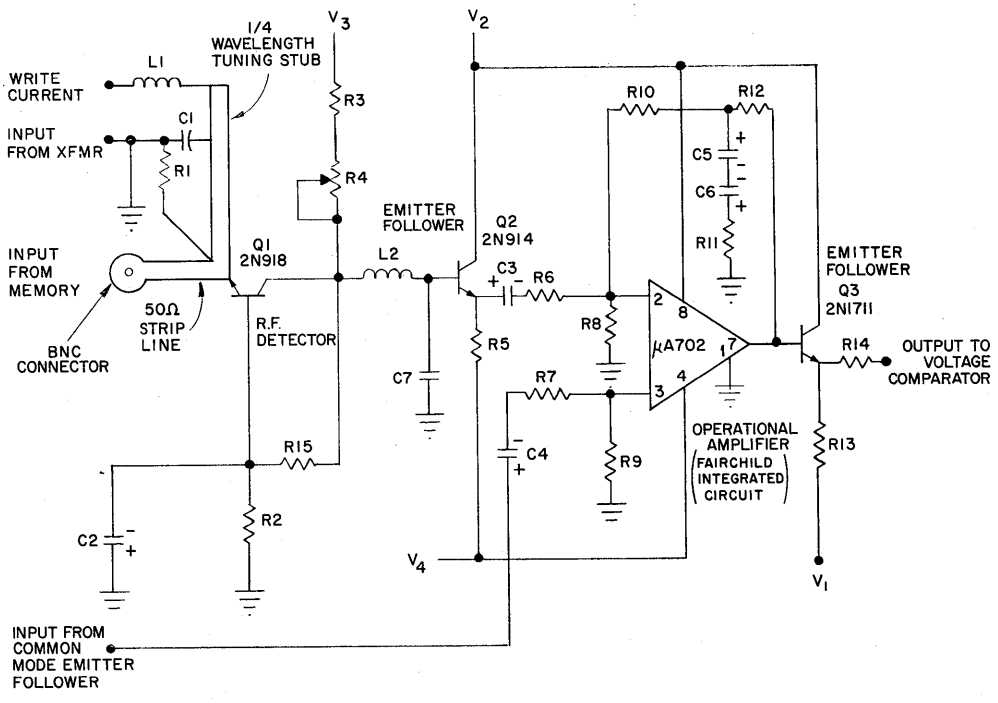



Figure 4.

since it was possible to obtain a signal voltage gain of 32 as compared to a diode. This gain avoided noise problems in the following amplifier and resulted in a better signal-to-noise ratio being obtained as compared to a diode.

Detector noise as observed on an oscilloscope was below 0.1 volt peak at the amplifier output. Referred to the input at the emitter there is a voltage gain of 8,000, thus noise peaks at 12.5 microvolts for this particular detector. Turning the oscillator on and off makes almost no difference to the observed noise so that the solid state oscillator

driving 10 milliwatts into the R.F. lines does not appear to be a principal source of noise. It is believed that a better noise figure could be obtained with an improved detector design.

The observed signal-to-noise ratio was 120 to 1 for a stored 1 and 70 to 1 for a stored 0. If the number of cells on the R.F. line were increased (by 8) to 256, the signal noise would be 15 to 1 and 8.7 to 1. This assumes that attenuation and signal from each memory cell was reduced by 8 to 1 to maintain optimum line attenuation. Adjustment of cell width, thickness, and $H_K$ could achieve this. Since a considerable allowance must be made for variations in detectors, standing waves, and variations in memory cell performance, this represents a practical limit for the setup as tested. However, a better detector might be designed and better film might be made so that extension of the number of memory cells per R.F. line is quite possible.

## R.F. LINE IMPEDANCE

The initial choice of a 50-ohm R.F. line was dictated by the convenience of using existing fittings, cables, and R.F. measuring equipment. However, given a required memory cell center to center spacing (80 mils was initially selected), the lower the line impedance the wider the strip line must be, assuming spacing to ground is fixed by glass substrate thickness. Thus R.F. line-to-line coupling will be increased as the spacing between lines is reduced. There is therefore a lower limit of impedance which is reached for a given layout when R.F. line-to-line coupling becomes excessive. The worst-case coupling that must be considered is all of the lines to one other line. If all the bits being read out are 0's (or 1's) except one bit, then R.F. coupling will provide a competing signal of polarity which will reduce the required signal.

It was determined experimentally that the absorption (and therefore signal output) from a given memory cell increases almost linearly until the cell is three times the width of the R.F. line. This is probably true only for the narrow (5.5 mil) R.F. lines used where fringing fields are excessive. Spacing to ground was 6 mils. It was also found that writing into a memory cell by rotation of its magnetization required approximately the same word current whether the word line was 1/6 of the cell width or equal to the cell width. This was due in

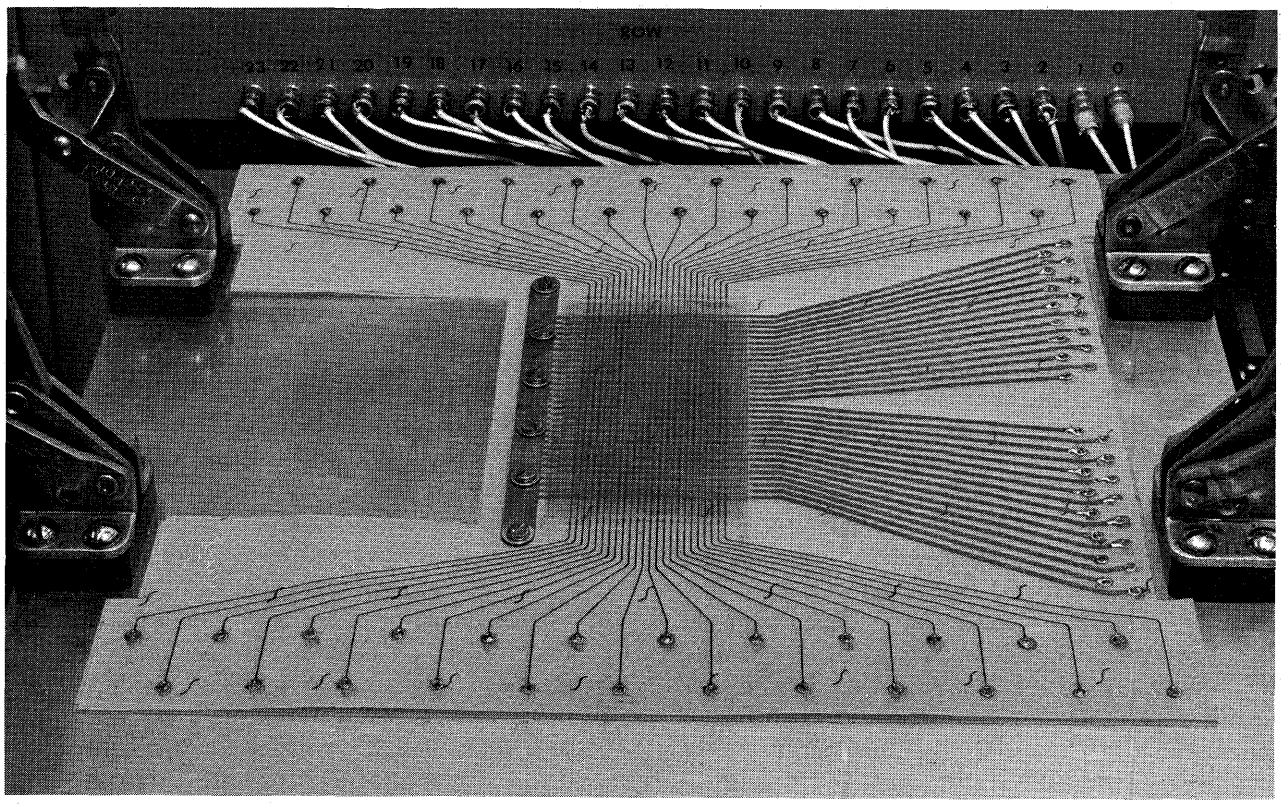



Figure 5.

part to fringing fields, in part to the fact that demagnetizing fields prevent alignment of the edge of a thin film cell in the hard direction and in part to substantial magnetic coupling from region to region within the thin film cell.

The disadvantage of a narrow line to obtain 50 ohms did not appear very substantial. The advantage of wider R.F. lines might be the use of greater R.F. power without causing magnetic creep during reading. The line current $I^2$ = Power/$Z$. Approximately,

$$Z = \frac{\text{spacing to ground}}{\text{conductor width}} \times \text{constant } K_1$$

but space to ground is determined by glass substrate thickness so

$$Z = \frac{K_1}{\text{Width}} \text{ and } I^2 = \frac{\text{Power} \times \text{Width}}{K_1} = \frac{P_i W_i}{K_1}$$

The R.F. magnetic field on the surface of the flat strip line is approximately given by $H_{RF} = \frac{I \times K_2}{\text{Width}} = \left(\frac{P_i}{W_i}\right)^{1/2} K_3$, so that if peak R.F. magnetic field is the main cause of creep during reading it can be reduced as the inverse root of the R.F. line width, although in most practical layouts R.F. coupling will make lines of less than 50 ohms impractical apart from the problem of making special fittings cables and measuring equipment.

The principal source of coupling between the R.F. lines was capacity coupling between the R.F. lines and the digit write lines (see Fig. 5). The layout in Fig. 5 has 768 small capacity coupling points at the matrix crossover points. When 23 R.F. lines are driven with equal R.F., the worst-case coupling on the 24th line was measured as 16 db down on the driven lines. The 24th line is matched at its input with 50 ohms for this measurement. Removing the digit write lines reduced the R.F. coupling for this test to 35 db below the driven lines. It would not therefore be practical to extend this layout much beyond 32 memory cells per R.F. line. However if a 1-mil metal sheet is placed over the R.F. lines, and it is suitably grounded at intervals, then almost no R.F. will be found outside the thin metal top plate. If the digit write lines are insulated and placed over this sheet they no longer form a coupling network. However at a readout rate of the order of 1 megacycle the field from these lines will penetrate the thin metal shield and permit readout of words from the memory, and also permit information to be written in the memory.

## THE MATRIX TESTER

A memory thin film tester was designed and constructed both for testing film for use in an R.F. absorption type memory and to determine the magnitude of the problems in reaching a practical design for a useful memory. Figure 6 shows the completed tester. Thin film substrates could be inserted under a grid of R.F. lines (24) and digit lines (32) as shown in Fig. 5. To obtain information on writing in the memory, and reading the memory, it was arranged that the 768 switches shown would provide unlimited variation of the pattern to be written, and a check of the contents on a subsequent read cycle.

A complete read write cycle is repeated at a 30 cycle rate so that a flicker-free display could be obtained (on the CRT shown) of the memory contents or error pattern or switch pattern. Word current, digit 1 and digit 0 current, and read current were continuously variable by adjusting the power supplies shown so that the effect of any variable could be seen on an error pattern. To determine creep and stability, the tester can be switched to read only and the error pattern observed. The low clock frequency of 25 kilocycles made possible a relatively simple layout using pulse rise times of about 0.5 microseconds and allowed recovery time for drive transformers when writing. Even so some shielding of the 1.5-amp maximum word currents was necessary.

Both 1 and 0 signal outputs when reading were checked for signal level by voltage comparators whose trigger point was continuously variable from 2 to 7 volts. This enabled a memory cell to be identified rapidly if its readout signal was below the set margin. Display lights and stop on error provided the address of a memory cell that had a weak output. Identification of a memory cell on the CRT display was quite easy and this feature was normally used. The setup as shown allowed easy connection of a signal generator to any R.F. line so that a faulty memory cell could be tested over a range of R.F. frequencies to determine whether its resonance characteristics caused a read error. It was as a result of such tests that 550 megacycles was selected to make most of the film planes operate correctly.

Word currents were variable from zero to 1.5

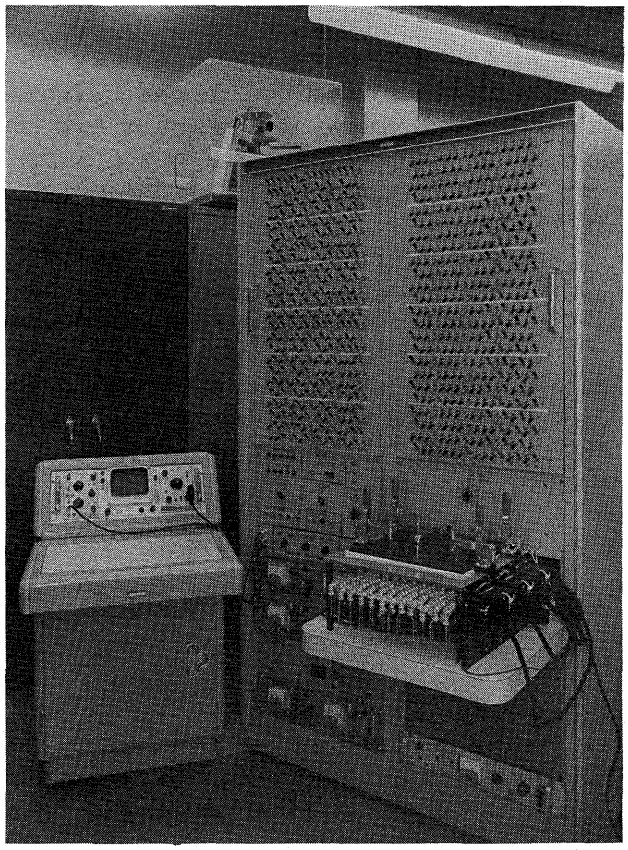


Figure 6.

amp through a 5.5-mil wide R.F. strip line. Digit currents were variable 0-300 milliamps through a 60-mil wide strip line.

Figure 5 shows the R.F. lines which are placed next to the thin film. The substrates are loaded under the matrix with the magnetic film on the top side of 6-mil glass. Four grounding screws must be removed to change the glass substrate.

The digit lines were divided into 3 12-mil lines spaced 12 mils apart to reduce capacity loading on the R.F. lines. This precaution was just sufficient to make the 32-word layout usable from the R.F. coupling viewpoint. As mentioned earlier, the R.F. coupling problem can be reduced by an order of magnitude if a "Tri-plate" layout is used with a thin ground plane separating the digit lines from the R.F. lines. It appeared that this could be a solution to the R.F. coupling problem in larger memories.

The R.F. source was a series-tuned oscillator using a 2N3375 transistor. This oscillator drove a strip line power splitter etched on a 10-mil glass epoxy copper laminate. A transformation from 50 to 2 ohms was made with a tapered line and 25 lines connected to the 2-ohm point through 50-ohm resistors to 25 coax lines. The resistors were required to absorb the reflected waves which had a strong effect on the signal amplitude.

The use of 50-ohm cables and connectors was almost essential in the R.F. system to allow measurements of power loss, SWR, and coupling. These problems must be well understood before making a memory layout comprising etched card strip line configurations which would not necessarily provide accessibility for making measurements.

In testing thin film planes a rather severe "creep" test is made when writing information because of a peak R.F. field below resonance of near 1 oersted acting in the hard direction for the film. Both R.F. and d-c were passed down the R.F. lines when writing. However, it did not prove necessary to turn off the R.F. when writing since the same digit current was used for reading and writing, and creep would occur on the read cycle if it affected writing. Nickel iron cobalt films were used with an overlay of copper diffused into the film. This appreciably helped the creep problem. A number of copper diffused substrates were tested and found to have reasonable write current margins. In addition they did not lose information on continuous read.

The use of a 5.5-mil word line for writing into a 30-mil wide memory cell did not appear to affect the digit current margins appreciably as compared with the use of a wider word line. However in testing 20-mil wide word lines (which required a strip line transformer for impedance matching at both ends) there was so much R.F. coupling because of the increased capacity coupling to the digit write lines that this test was not continued.

The R.F. absorption memory has a peculiarity that the word lines (R.F. lines) when writing become the digit lines when reading. Thus the memory must be loaded with one digit of every word at a time requiring that the information be prepared in this form before loading a memory. If it were required that one digit of each word be read out at a time (perhaps in searching for information) then

rotating the information format through 90° would be an advantage.

## DESCRIPTION OF A HYPOTHETICAL 4K WORD MEMORY

Figure 7 shows how an NDRO memory using resonant absorption phenomena for reading might

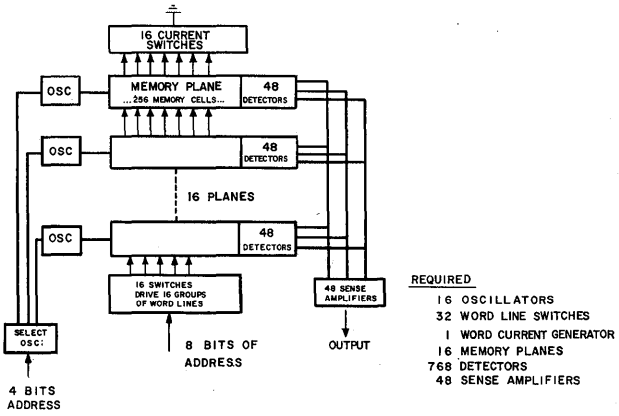



Figure 7.

be laid out to take advantage of the fact that R.F. is only required in the small block of the memory being interrogated. The block diagram shows a layout using 16 blocks with 16 separate oscillators and the present predicted limit of 256 memory cells per R.F. line. There is a transient settling time of 3-4 microseconds for the detector shown on Fig. 4 (except that $C_2$ was reduced to 200 picofarads) after switching the oscillators. No detectable noise or output was observed at the detectors (that resulted from the read current) when an R.F. oscillator was turned off because the detectors include a ¼-wave 550-megacycle grounded stub which would provide a short circuit for transients of lower frequency. It should therefore be possible to parallel the detectors to a common sense amplifier as far as read noise is concerned. Switching the detectors on and off simultaneously with the oscillators is not costly in terms of components if this should prove necessary.

## SUMMARY

*Possible Advantages of Resonant Absorption Readout*

1. The use of R.F. oscillators provides an extra switching dimension which might provide economies over word selection usually used to read out thin film memories. This is more likely to be relevant if the memory is very large than for the size of memory considered.
2. In large memories it might be economical to use relatively long pulse rise times to read out thin film. The resonant absorption readout technique makes this possible without reducing signal amplitude since the output signal is not proportional to the rate of change of the read current, but only dependent on the R.F. frequency and absorption characteristics.
3. In large memories the operating power could be quite low compared to other thin film types because very little power is required by a block of memory not being used.
4. Current margins for readout can be quite wide. Temperature compensation should not be required.
5. In some applications, turning the information format thru 90° could be an advantage, particularly when it is required to address bits within words rather than complete words.

*Disadvantages*

1. The present limit of 256 cells per R.F. line and detector may not prove sufficiently attractive economically. However it may be possible to make considerable improvements with better magnetic film and a better detector.
2. Turning the information format through 90° is likely to be a disadvantage for many applications. The use of an extra set of write conductors to correct this (in zigzag form) is a possible though not a proven practical solution.

## REFERENCES

1. H. D. Toombs and T. E. Hasty, *Proc. IRE*, June 1962.

# DEVELOPMENT OF AN E-CORE READ-ONLY MEMORY

P. S. Sidhu
*Ampex Corporation*
*Culver City, California*
and
B. Bussell
*University of California, Los Angeles*

## INTRODUCTION

Electronic memories in data processing equipment can be divided into two categories. In the first category are easily alterable memories such as ferrite core matrices, ultrasonic delay lines and thin films. These are used for temporary storage of information, for example, the program of instructions for a particular calculation, initial data and intermediate results. In the second category are fixed or "read-only" memories used to store information that is seldom, if ever, changed. Writing the information into these memories is part of the manufacturing process, and in order to change it one has either to replace a part of the memory or to alter its construction in some way.

The main advantages of a read-only memory over erasible read/write memory are:

Nondestructive read
No rewrite electronic and power
Faster cycle time
High reliability
Low cost
Nonvolatility

Many types of read-only memories have been devised. The information in matrix memories is stored by forming a matrix of word and data lines and placing a coupling element at each intersection whenever a "1" is stored. The main disadvantages are the low output signal, and low signal-to-noise ratio in large matrices.

A memory using inductive coupling but not built in the form of a matrix is the transformer memory or Diamond Ring Translator.[1] There is one address for each word stored and one transformer core for each binary digit in the output word. Each address wire threads through or bypasses a particular core, depending on whether the corresponding digit in that word is respectively "1" or "0." To read a given word a current pulse is passed through the corresponding address wire. This induces a pulse, representing 1, in the sense windings of the transformers through which it is threaded. No output appears from the transformers which are bypassed.

From the point of view of speed, storage medium, and cost of components, the Diamond Ring Memory is the most suitable of all the memories described by Taub.[2] However, in large memories, because of coupling between the word lines, sig-

nal-to-noise ratio suffers and the sense amplifier design becomes elaborate.

The problem of sensing can be solved by the use of an E-core in the place of a regular transformer core, making the polarity of 1 opposite to the polarity of 0. Two E-cores put together form two windows, as shown in Fig. 1. When a current is passed through window 1, it induces flux in one direction through the central leg; and if a current is passed through window 0, the flux is induced in the opposite direction. Since the sense winding can have many turns, the output signal may be large. Additionally, no change of state is involved as in ferrite cores, therefore lower drive energy is required and the output is not delayed.

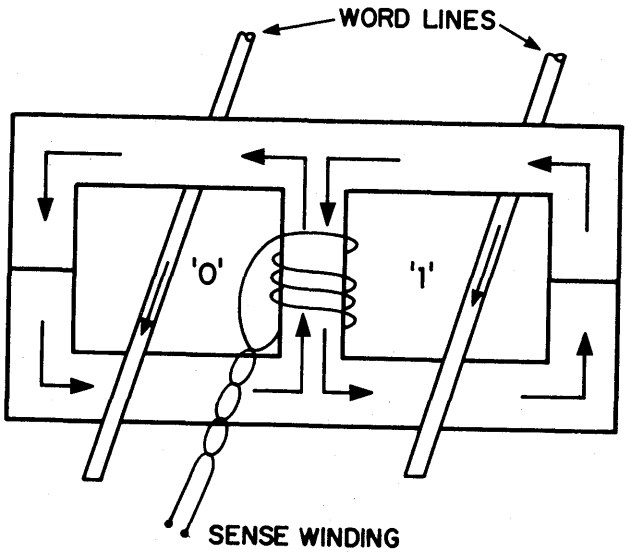



Figure 1. E-cores.

## MEMORY DESIGN

A memory of 1,024 words of 24 bits each was designed and built. For purposes to be described below, information was stored in two sets of E-core pairs. In each set 256 wires were passed through 48 pairs of E-cores. Thus, each of the 512 wires stored 2 words of information. Two groups of 256 wires each were used in order to cut down the capacitive coupling between the word lines and to reduce the drive current requirement. The electronics were designed to read one word at a time out of the memory. The electronics consisted of:

- An address register comprising 10 flip-flops which held the 10-bit address representing the addressed line or word.
- An address decoder comprising 6 1-by-4 decoders, to decode the address and to select a particular line driver.
- Sixteen voltage drivers and 32 current drivers to drive a current through the selected word line.
- Information register consisting of 24 flip-flops representing one full word length of information for parallel transfer.
- Forty-eight sense amplifiers to amplify the signal output of the E-core when a word was read.
- Timing control pulses to generate the required signals for the memory operation.

The logical design of the memory (Fig. 2) is considerably simpler than a regular read/write type

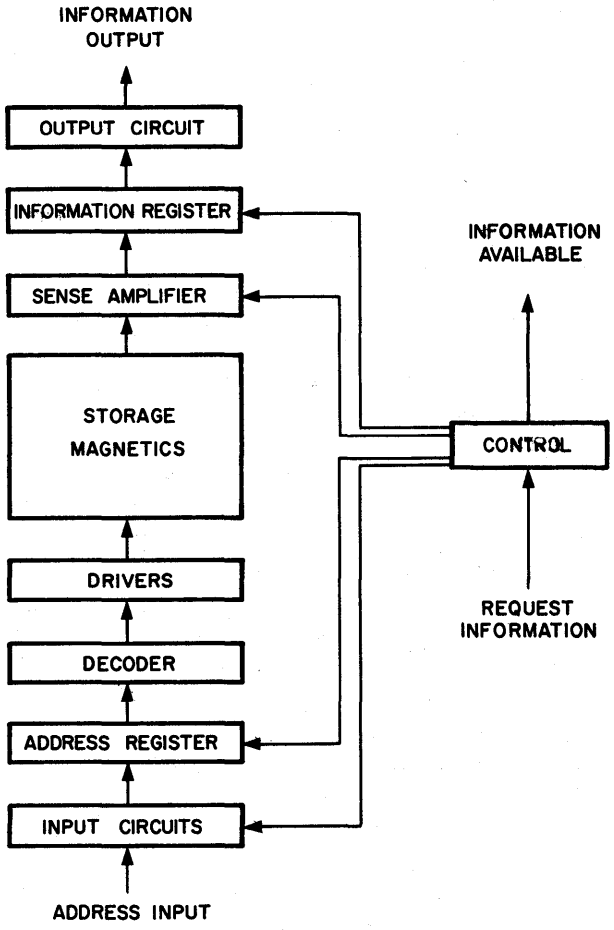



Figure 2. Block diagram of memory.

of memory because only the read cycle is performed and, secondly, the read is nondestructive—that is, regeneration of the information is not required. The

read cycle operation can be broken down into the following operations:

1. On receiving the "Request Information" command the address register is strobed to transfer the new address information.
2. The address is decoded to select a word line in which the read current will flow.
3. When the current is established in the word line, the sense amplifiers are strobed to sense the output signal of the E-core. This signal sets the information register.
4. As soon as the Information Register is set, the "Information Available" signal is sent to the computer.

The memory timing is represented by the timing diagram in Fig. 3.

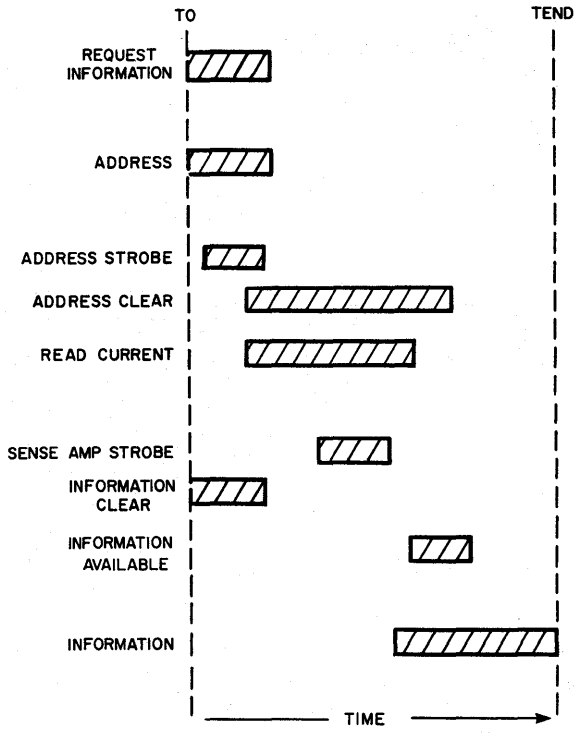



Figure 3. Timing diagram.

## MAGNETICS

The magnetics of the memory was constructed on a wooden board. Two rows of 48 E-cores were placed in line on the board as shown in Fig. 4. Two wooden panels were mounted along the cores to hold them in position. One-half inch distance between the cores was allowed to permit 256 wires to pass from the 1 window to the 0 window of the next or vice versa. Forty-gauge enameled wire was used to wire the word lines through the cores. The ends of the word line were terminated in a printed circuit board mounted on the ends of the wooden board. Each line passed through the 1 or 0 window of the cores depending on the information word stored. It is proposed that after wiring the word lines through the E-cores, the magnetics be removed from the wooden board and mounted on the end cards as shown in Fig. 5. The two cards can then be secured to each other to form a plug-in module.

The minimum size of the E-core is determined basically by the number of words to be stored.

The sense winding was wound in a spiral and placed around the central leg of the E-core. Several tests were made to determine an optimum number of turns. The result (Table 1) shows that the

Table 1.

| No. of Turns | Word Drive *ma* | Sense Termination Res: 4,700 | 3,300 | 2,200 | 1,500 | 1,000 | 680 | 330 |
|---|---|---|---|---|---|---|---|---|
| | | Output in volts | | | | | | |
| 10 | 20 | +0.9 | 0.8 | 0.65 | 0.55 | 0.45 | 0.4 | 0.3 |
| | 40 | +1.8 | 1.7 | 1.4 | 1.2 | 1.0 | 0.8 | 0.6 |
| 20 | 40 | +2.1 | 1.8 | 1.5 | 1.3 | 1.1 | 0.8 | 0.6 |
| | 20 | +1.0 | 0.3 | 0.75 | 0.60 | 0.5 | 0.35 | 0.25 |
| 5 | 20 | +0.6 | 0.5 | 0.4 | 0.3 | 0.25 | 0.2 | 0.1 |
| | 40 | +1.0 | 0.3 | 0.8 | 0.7 | 0.55 | 0.4 | 0.2 |

optimum number of sense turns should be 8 for a drive of 50 milliamps in the word line.

The magnetics was diode-decoded as shown in Fig. 6. Thirty-two current switches were decoded in two groups of 16 each. Address bits 0 through 3 decoded 1 out of 16 voltage switches; bits 4 through 7 decoded 2 current switches out of 32 (1 in each group); but 8 selected 1 current switch out of the 2 sets by gating bits 6 and 7. Bit 9 gated the sense amplifier strobe, selecting 1 group out of the 2 sense outputs.

## DRIVE SYSTEM

The drive system was designed as a single block and consisted of input circuits, address flip-flops, decoders and drive switches. The circuit diagrams are given in Figs. 7 and 8. The drive system operation is briefly described below.

Referring to Fig. 7, normally strobe 2 is negative which keeps both sides of the address flip-flops in the false state and thus all the decoder transistors

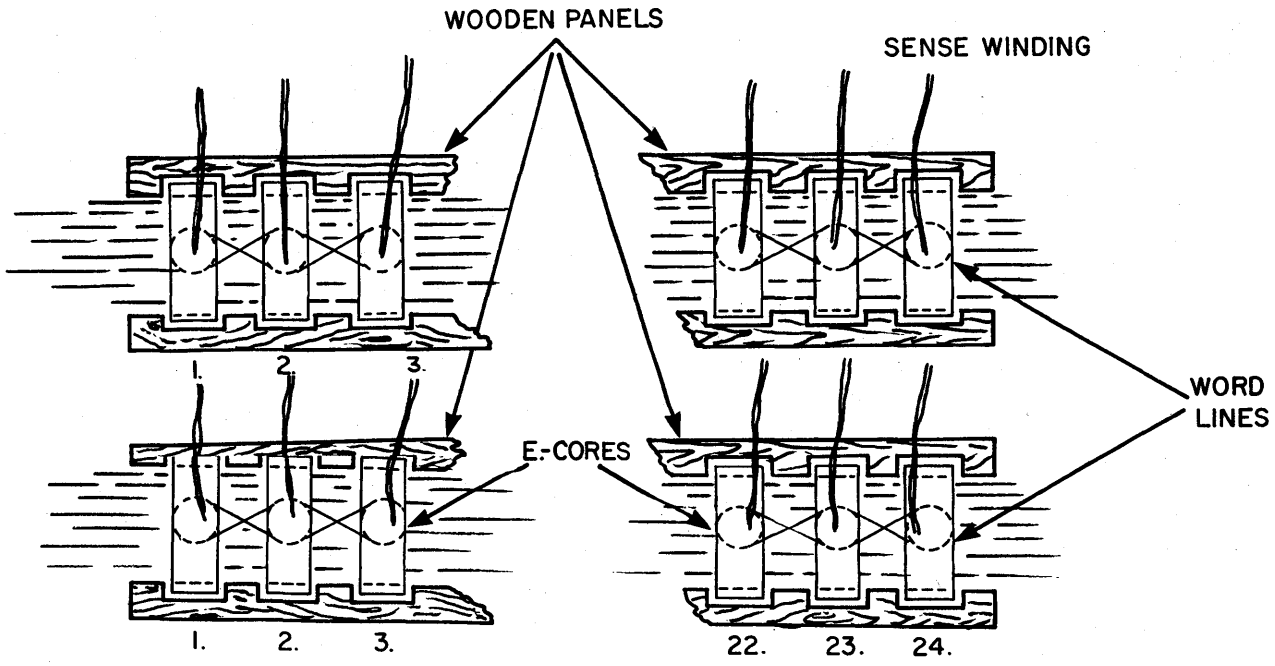



Figure 4. Magnetics assembly.

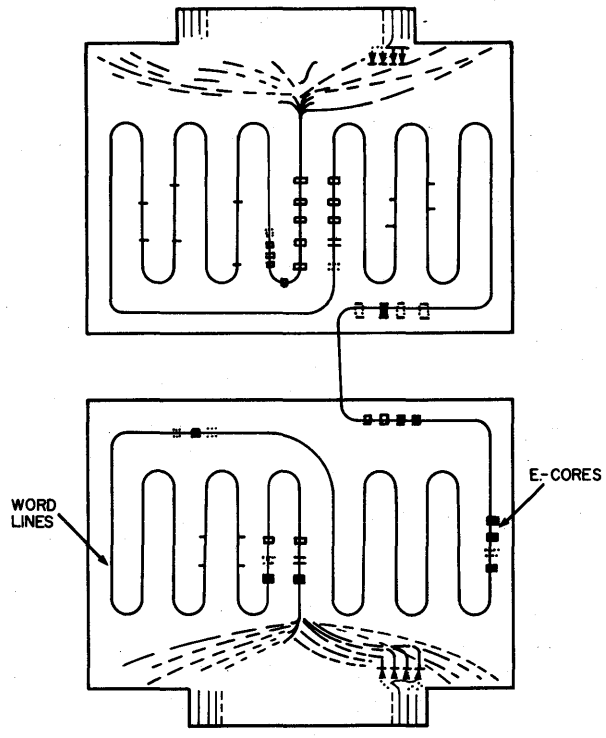



Figure 5. Magnetics packaging.

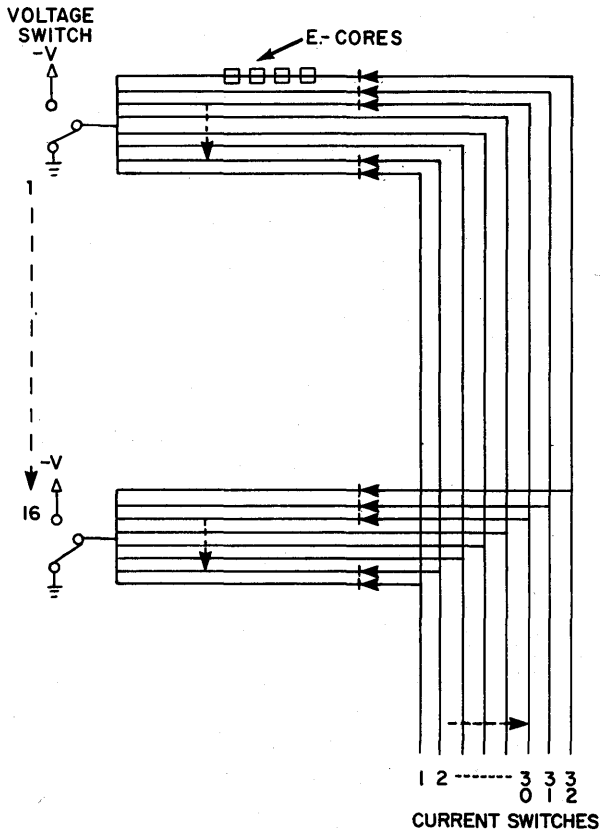



Figure 6. Magnetics decoding.

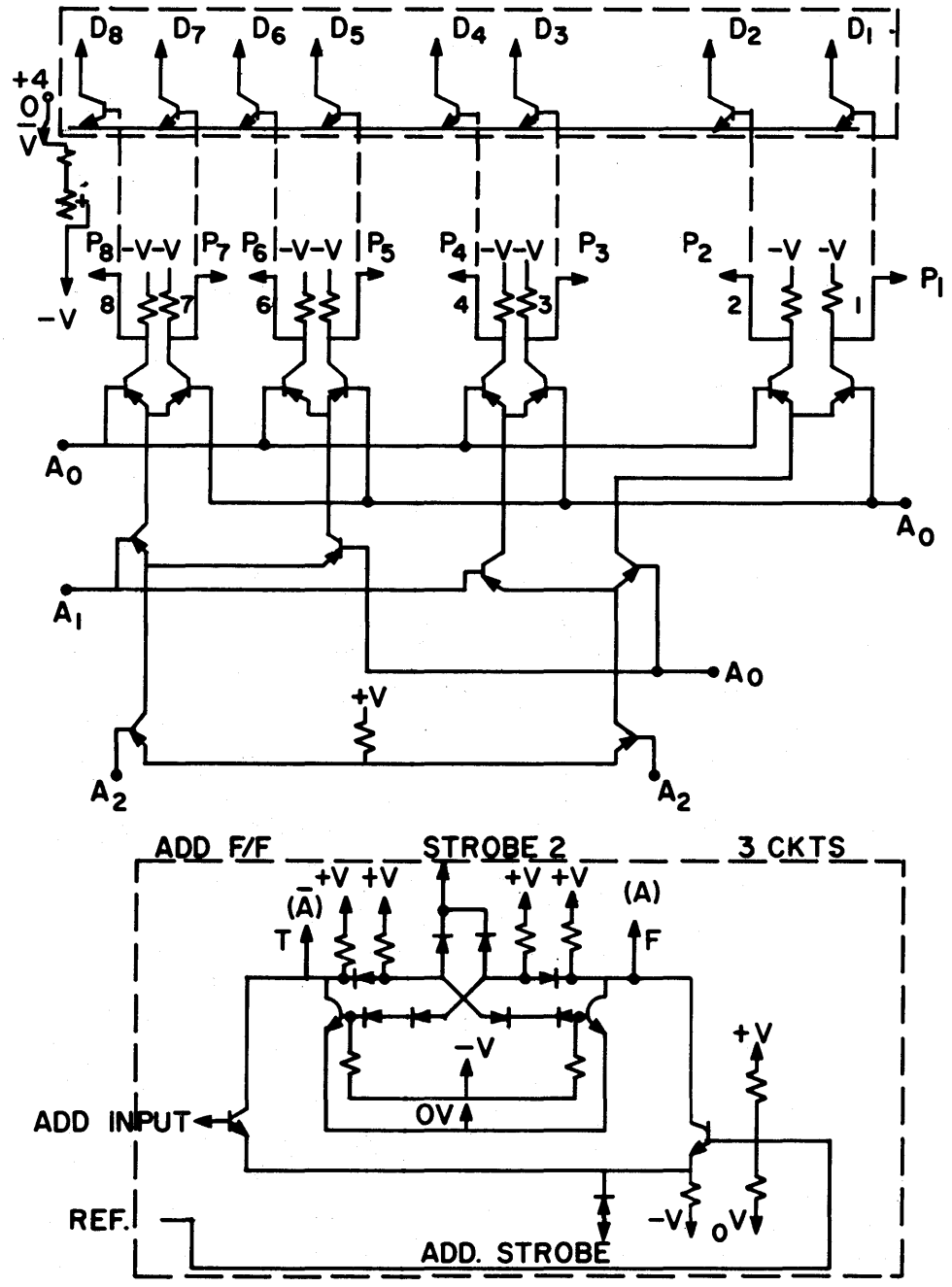



Figure 7. Drive system.

are reverse-biased, or all eight outputs are false. All the drive switches, positive and negative, are off. Both of the transistors of the input circuit are reverse-biased by the positive level of the address strobe.

When a "Request Information" pulse is received at the control, the address strobe goes negative. One of the decode outputs goes positive depending on the address information. Referring to Fig. 8, 1-out-of-16 drive switches is selected by the coincidence of 2 true 1-out-of-4 decoder outputs.

Approximately 40 nanoseconds after the address strobe, strobe 2 goes positive (Fig. 7), which sets the address flip-flop. Near the end of the cycle, strobe 2 goes negative and both the transistors of the address flip-flop are reverse-biased which in

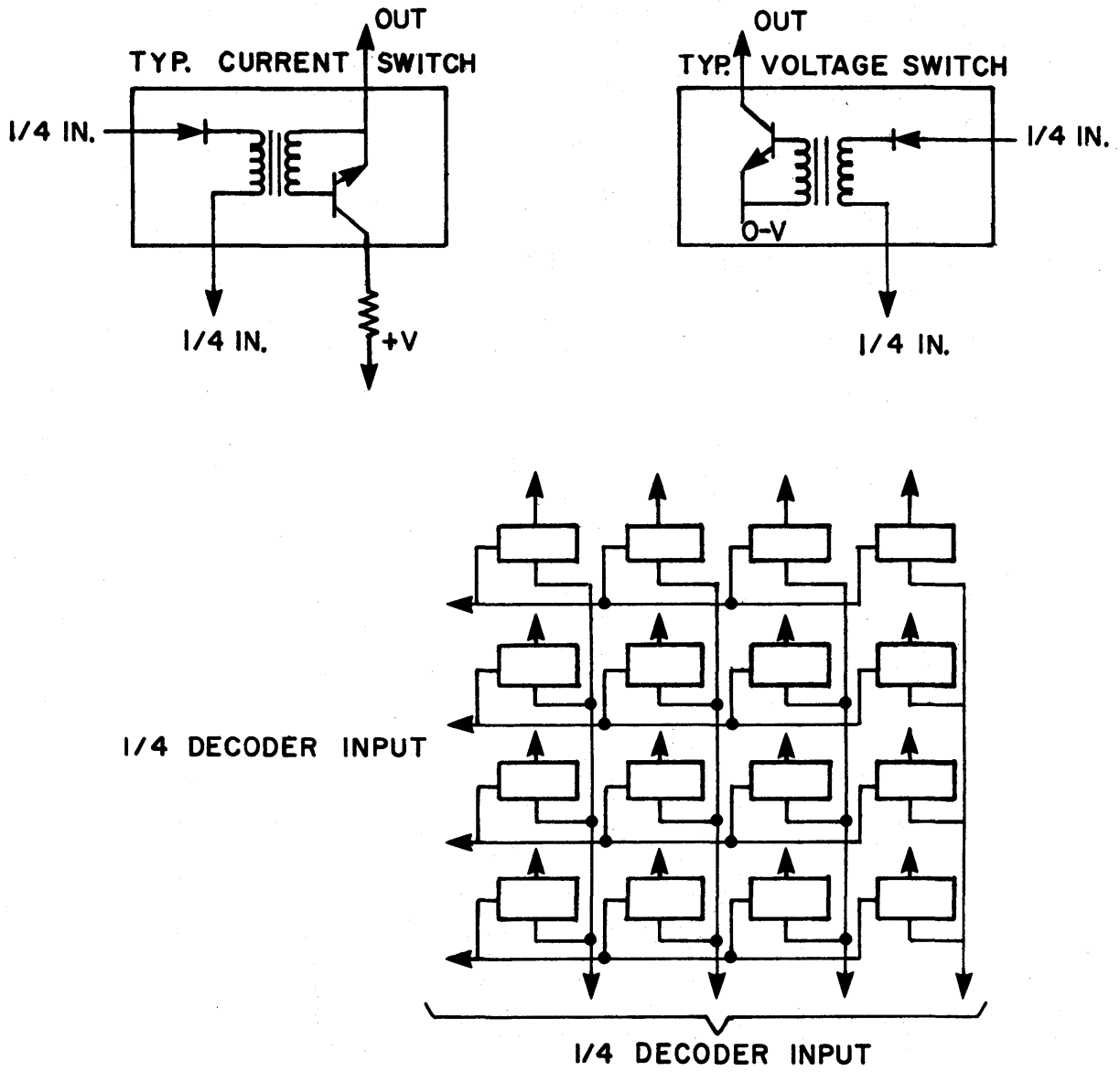



Figure 8. Drive switches.

turn reverse-biases all the decoder transistors and the drive switches.

## TIMING

The timing circuit controls the operation of the memory through the following timing signals:

1. Address strobe
2. Strobe 2
3. Sense strobe
4. Data clear

These signals were generated on receiving the "Request Information" command, as shown in Fig. 9, by tapping a delay line.

## BIT SYSTEM

The bit system consists of the sense amplifier, Information Register and transmitter circuits. The sense amplifier circuit was designed to amplify positive going signals and reject negative going signals. The circuit was strobed because of signal kickback.

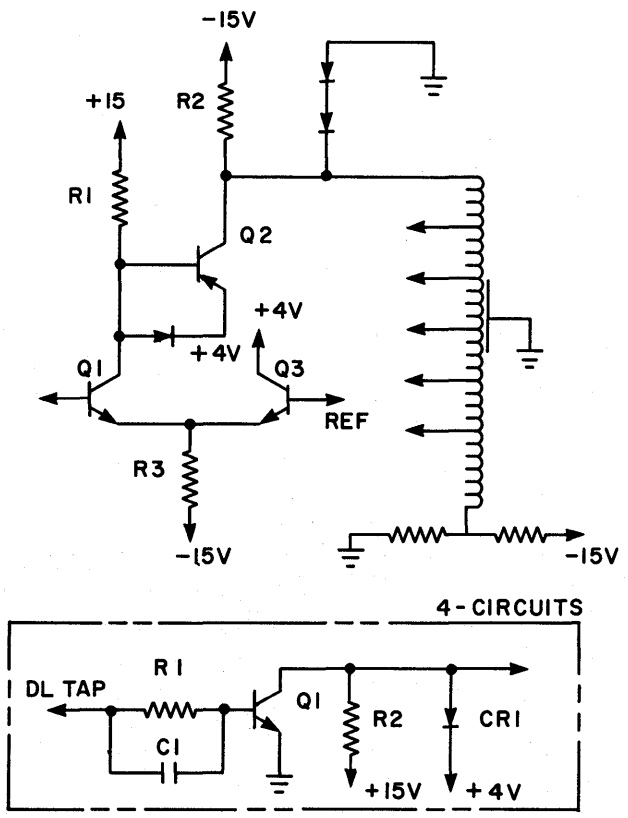



Figure 9. Schematic timing.

A negative going 0 signal was followed by a positive going overshoot which could falsely set the flip-flop. The output of the sense amplifier was connected to the true side of the information flip-flop, and also transmitted to the computer through the transmitter circuit (Fig. 10).

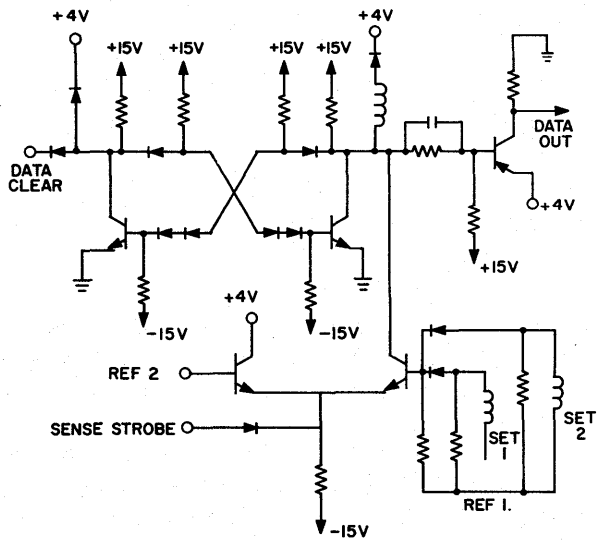



Figure 10. Schematic — bit system.

*Operation*

The memory operated at a cycle time of 250 nanoseconds with the information available at 150 nanoseconds. The decoder and drive switch were selected in 50 nanoseconds and the current was established through the word line in 100 nanoseconds. The transmission delay through the word line was 15 nanoseconds for a drive current of 50 milliamps. The drive current was set at 50 milliamps to get the maximum sense output voltage and fastest recovery 100 nanoseconds. E-cores of 4A material of Ferroxculie were used for this model.

The optimum termination of the sense winding was 2,700 ohms (see Table I). The pattern of information or of addressing the memory had no effect on the recovery and output voltage of the sense winding. Diagrams of Figs. 11 and 12 show the sense line output and cycle times of the memory with two addresses selected.

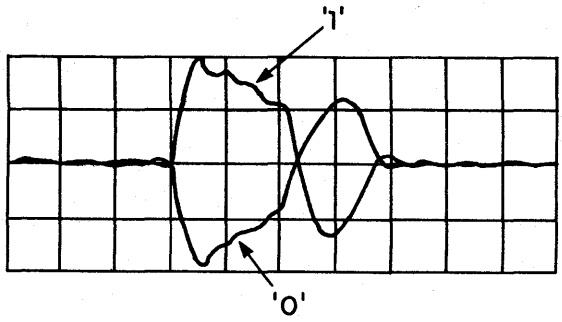



Figure 11. Sense line output. Horizontal scale = 40 nanoseconds/division; vertical scale = 1.0 volt/division.

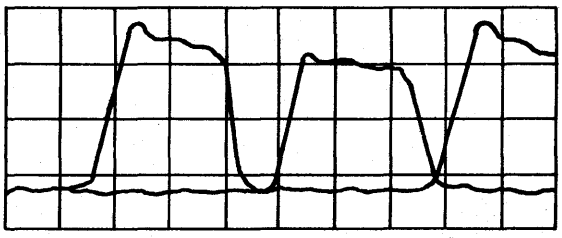


Figure 12. "1" Output with respect to "Request Information." Horizontal scale = 40 nanoseconds/division; vertical scale = 2.0 volts/division.

*Multiple Selection*

The memory is organized in a linear select mode. That is, a single line is selected in order to read information. This type of selection technique is costly because the number of lines wired is directly pro-

portional to the number of words stored and a high level of decoding is required.

To overcome the problem of cost and size limitations of the memory in a single wire per word scheme, the multiple select (coincident) technique was investigated. In this technique two wires instead of a single wire are selected at a time to read one word of information (Fig. 13). The output signals are then combined as given in Table 2.

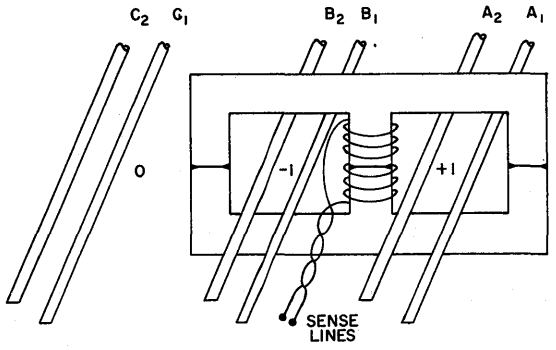



Figure 13. Multiple selection organization of E-core Memory.

Table 2.

| Word Lines Selected | Logical Combination | Sense Line Output *volts* | Logic Output | Comments |
|---|---|---|---|---|
| $A_1+B_1$ | +1 −1 | 0 | 0 | Positive flux cancels negative flux. |
| $A_1+C_1$ | +1 +0 | +1 | 1 | Positive signal is 1. |
| $A_1+A_2$ | +1 +1 | +2 | 1 | Positive signal is 1. |
| $B_1+C_1$ | −1 +0 | −1 | 0 | Negative signal is 0. |
| $B_1+B_2$ | −1 −1 | −2 | 0 | Negative signal is 0. |
| $C_1+C_2$ | 0+0 | 0 | 0 | No flux coupling into the E-core. |

It is assumed that the current flowing through one word line induces 1 volt in the sense line. The logical representation of sense line output is arbitrarily taken as positive flux for 1, no or negative flux for 0; it can be changed to suit the type of information to be stored. Six different combinations help to reduce the number of lines required to store the information.

The information stored at line 1 or line 2 does not represent the word of information, but it is the logical combination of these two lines which gives the word of information. In order to store $N$ words of information, $N$ combinations need to be stored. As a minimum, only $2x$ $N$ different words (lines) have to be wired in order to store $N$ words. This technique of multiple selection can be extended from double to triple and quadruple selection, which further reduces the number of different states. In triple selection, only $3x^3N$ and in quadruple selection $4x^4N$ different states are needed for $N$ words of storage. Table 3 shows a comparative organization of the memory under various selection techniques. The cost of a diode is taken as 1 unit and the remainder of the entries are calculated based on this unit. Fixed cost is estimated at 1,500 units (without power supplies) which includes receiver, flip-flop, timing and transmitter circuits, assembly, packaging and testing. A 1 to 0 ratio is considered (*1*) high when the sense output polarities are opposite, (*2*) good when the ratio is 5 to 1 or greater, and (*3*) poor when the ratio is less than 2 to 1.

Table 3 shows that series parallel gives the optimum configuration for straight linear select mode. Double select technique is the most attractive from a cost point of view. Out of 1033 logarithmic and exponential constants (*3*), 256 words were selected to give 16 + 16 = 32 different states for storage by double selection. The model operated at a speed of 250 nanosecond cycle time and access time of 150 nanoseconds. The sense line output is shown in Fig. 14. An effort was made to program the information in attempt to yield $2xN$ states. However, this optimum lower limit was never achieved. The addressing logic for multiple selection becomes complex and the memory cost increases beyond that of a linear select. This approach was therefore abandoned. Further work in programming the information is suggested.

Table 3. Comparison of Cost and Speed of E-Core Memory for Different Selection Techniques.

| Mode | Word Line Cost | Cores and Sense Winding | Sense Amplifier | Drive Switch and Decoder | Additional Units | Access Time | Cycle Time | Signal-to-Noise Ratio | Comments |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | *nanoseconds* | | | |
| *1. a)* Linear select | 2048 | 48 | 144 | 470 | 2710 | 150 | 250 | High | Costly |
| *b)* Series parallel | 1024 | 192 | 288 | 375 | 1879 | 150 | 250 | High | Optimum |
| *c)* 4-Series | 576 | 192 | 576 | 280 | 1624 | 200 | 300 | High | Long drive line |
| *2.* Double select | 128 | 48 | 144 | 150 | 470 | 200 | 300 | Good | |
| *3.* Triple select | 64 | 48 | 288 | 150 | 550 | 200 | 300 | Poor | |

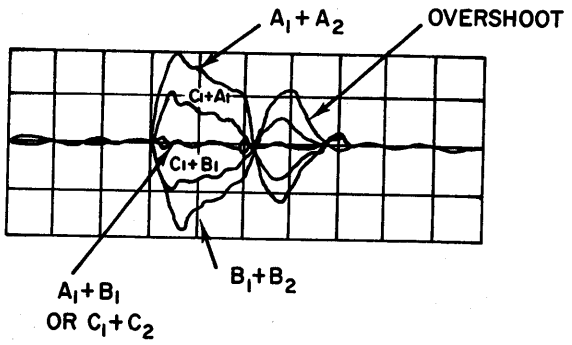



Figure 14. Sense line output for a double select technique. Horizontal scale = nanoseconds/division; vertical scale = 2 volts/division.

## AUTOMATIC WIRING

A computer program can generate the word line pattern for storage. The lines are printed on a double sided strip. The strip is punched in the middle to pass the central leg of the E-core. The cross-over from one bit position to next is accomplished by passing 1 to 0 word lines on one layer and 0 to 1 on the other. The number of words on each strip is determined by the number of lines which can be printed in one half of the strip. The computer program guides a photopointer to produce a photoprint of these lines. After making, these strips, each of which may contain up to 64 words, are placed in the E-cores.

## INTEGRATED CIRCUITRY

Integrated circuits can be used for almost all the circuits in this memory. The flip-flops are two cross-coupled DTL gates. The decoders and drive switches can be combined into one circuit like the Fairchild 932 or 946 gate. For sense amplifier and receiver circuit, Motorola MIC logic can be used. The use of integrated circuits cuts down the component count, thus cutting down the cost considerably, and makes the memory more reliable.

The operation of the memory demonstrated the following:

1. The speed of the memory depends upon:
   (a) the recovery of the sense line;
   (b) the time required to decode the address and to select a voltage and a current switch.
2. The access time depends upon:
   The time required to decode the address and to select a voltage and a current switch, together with the time required to amplify the signal to set the data flip-flop and the transmitter.
3. For the E-core used, 50 milliamps of word-drive current gives the optimum signal of +1.2 volts. The number of sense turns for this output is eight.
4. Because of the high output voltage, the sense needs only one stage of amplification to set the data flip-flop.

The model clearly demonstrated that it is possible to use E-core as a storage element. The speed obtained with the prototype was 250 nanoseconds. The recovery of the sense line depends upon the material of E-core. Ferrite material, with response up to 100 megacyles per second is available, as compared to the 10 megacyles per second material used in this model E-core. The faster the material, the faster the core recovery; hence the present recovery time of 150 nanoseconds can be substantially reduced simply by changing the core material. For computer storage, cycle time of 250 nanoseconds and access time of 150 nanoseconds are considered very high speed compared with the present state of the art. The 0 signal is of opposite polarity to 1, thus any problem of discrimination between 1 and 0 signal is avoided. The E-core wiring can be automated very easily since the wires can be prebent or printed according to the information and then just dropped into the open windows of the E-core. Also, the number of E-cores used in the memory is small compared to any other type of memory. The reduced number of cores and the automatic wiring reduces the cost of the magnetics of the memory profitably.

The automatic wiring eliminates wiring errors to which the human hand is susceptible; the clear discrimination between 1 and 0 signals also contributes to a highly reliable operation.

To summarize—an E-core memory is feasible and seems to possess the following unique advantages:

1. High speed — 250 nanoseconds.
2. High reliability:
   (a) Opposite polarity of 1 and 0 signals.
   (b) Automatic word wiring.
3. Low cost — at least one half that of any other memory.
4. No limitations on the size of the memory.

## REFERENCES

1. T. L. Diamond, "No. 5 Crossbar AMA Translator," *Bell Laboratories Record,* 29.62 71, 1 (1961).

2. D. M. Taub, "A Short Review of Read-Only Memories," *Journal IEE* (England), vol. 19, pp. 29-31 (Jan. 1963).

3. D. Cantor, G. Estrin and R. Turn, "Logarithmic and Exponential Function Evaluation in a Variable Structure Digital Computer," *IRE Transactions on Electronic Computers,* vol. EC-11, pp. 155-164 (Apr. 2, 1962).

# MAGIC — A MACHINE FOR AUTOMATIC GRAPHICS INTERFACE TO A COMPUTER

D. E. Rippy and D. E. Humphries
*National Bureau of Standards*
*Washington, D. C.*

## INTRODUCTION

The Computer Technology Section of the National Bureau of Standards is currently engaged in an extensive program to develop advanced techniques for improving user communication with large ADP systems. This program is an outgrowth of a number of projects in which the Section has been called upon to assist other agencies in the solution of a variety of data processing applications.

These projects have provided contact with many kinds of data processing problems which generally involve such functions as command and control, design and mapping, updating and utilization of active files, editing, and information retrieval. Development of data processing hardware for use as tools to aid the investigation of such functions is usually required.

This report describes a machine which has been developed within the Computer Technology Section as a research tool for the investigation of man-machine communication techniques involving CRT displays. This machine has been designated MAGIC (*M*achine for *A*utomatic *G*raphics *I*nterface to a *C*omputer). This machine combines large-diameter cathode-ray displays with a specially designed programmable digital computer. It is designed as a remote display station and is intended to be connected to a large ADP system via voice quality communication lines. Extensive design effort has been devoted to removing from the ADP system the time-consuming and repetitive tasks of display regeneration and manipulation, and to minimize the limitations introduced by the communication lines. Particular emphasis has been placed on establishing the proper balance between hardware and software functions.

MAGIC was originated in August 1964 and completed in February 1965. It is currently being used to conduct experiments and perform demonstrations in order to better define the optimum characteristics for equipment of this type.

## SYSTEM DESCRIPTION

Design specifications of MAGIC evolved from a number of basic considerations such as memory type and organization, display type and control, word formats for the machine instruction and display data, operator controls, and system economics. Underlying the over-all display data organization within MAGIC and the digital manipulation of these data is the realization that most CRT displays function in a point-to-point manner, implying that

the digital data required to drive the display device must exist in a serial list form with respect to time. Therefore, it would seem logical to generate, manipulate and store display data with a processor having list processing abilities. Also, as will be shown in detail later, this list processor organization results in a significant savings in the software required for display operation. This concept has provided the basis for the hardware design of the MAGIC system.

*System Hardware Organization*

There are two major hardware sections of the MAGIC system as seen in Fig. 1. These are the display unit and the processor unit. The display unit consists of a primary and a secondary CRT display.

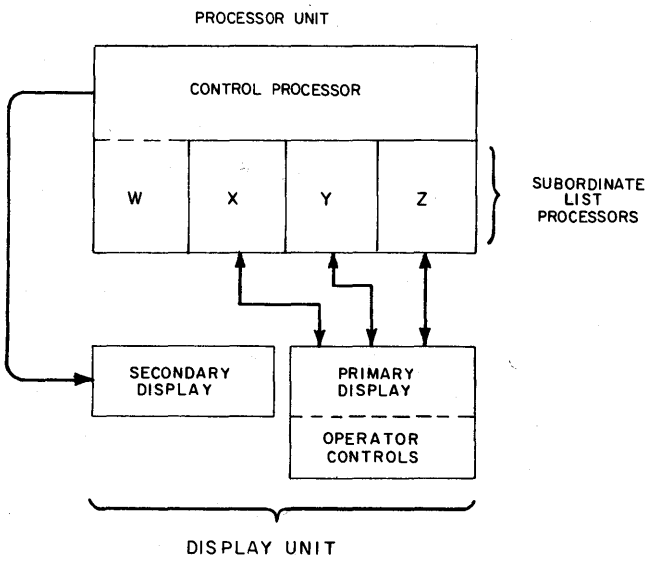



Figure 1. Machine for automatic graphics interface to a computer.

The operator uses the primary display and its associated controls to perform the majority of his communications with MAGIC. The secondary display is used as a passive display device only.

The processor unit is subdivided into one control processor and four identical subordinate list processors designated W, X, Y and Z. Subordinate list processors X, Y and Z operate directly on the portion of memory used by the primary display. Subordinate list processor W is considered part of the control processor and allows the control processor to perform list manipulations without disturbing the primary display.

*Drum Memory*

The magnetic drum memory within MAGIC is similarly subdivided. The control processor communicates with the portion of memory termed general memory. The portion of memory designated as display memory is used by the subordinate list processors (including W) and the primary display. The secondary display, however, receives its display data from three channels of the general memory.

A rotation speed of 1800 rpm was chosen for the drum since it provides a display refresh rate of 30 frames/sec, which is generally recognized as the lower limit for flicker-free presentation of stationary information. When the limitations of operating speed of the display consoles and the available logic hardware were taken into account, it was found that 128 12-bit words per drum memory channel could be accommodated; this results in a system clock rate of slightly less than 54 kHz. General memory capacity is 90 channels.

*Word Formats*

The processor unit functions as a serial, single address processor. The single operand address and the various operation codes are organized in a double length instruction word format as shown in Fig. 2. Bits designated W, X, Y and Z in the instruction format address the respective subordinate list processors. Note that any combination of W, X, Y

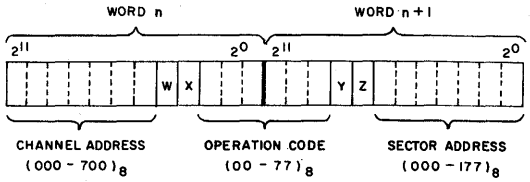



Figure 2. Instruction word format (double length).

or Z may be specified simultaneously in any one instruction, allowing additional flexibility in manipulating display data.

Display data within MAGIC may be considered to consist of three fields: the X coordinate field, the Y coordinate field and the Z field. These fields are the X, Y and Z lists in display memory and are operated on by subordinate list processors, X, Y and Z respectively. The Z field in the list specifies the display characteristics for the associated X and Y coordinate words. The X and Y coordinate data consist of 10 bits centered in the 12-bit standard

data word. This allows an increase or decrease in display size by a factor of two without loss of coordinate data bits by utilizing shifting techniques.

Graphic presentation on both displays represents the first quadrant of the cartesian coordinate X-Y plane. The 10-bit coordinate fields provide a resolution of 1024x1024 discrete positions. Since all numbers within the first quadrant are positive, all numerical data in MAGIC are assumed positive and are therefore unsigned. Also, since MAGIC is display-oriented and therefore not primarily intended for general purpose processing, arithmetic operations in MAGIC are of the simplest form, consisting primarily of straightforward binary addition. Further, there is no parity bit associated with MAGIC data.

The Z word format is shown in Fig. 3. The desired Z parameters may be processor-generated or

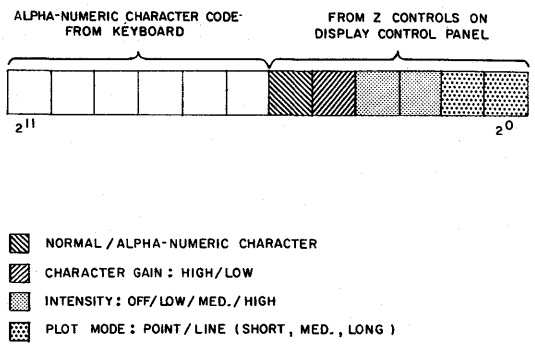



Figure 3. Z word format.

may be specified manually via the Z switch register on the operator's control panel.

The instruction repertoire of MAGIC may be divided into two categories: list instructions and non-list instructions, as tabulated in Fig. 4. List instructions are considered as those affecting data in the four display memory channels. They may be performed on a single sector within display memory or on a list of specified length in display memory. Non-list instructions pertain primarily to the control processor. Since the execution of the instructions categorized in Fig. 4 is intimately tied to the hardware which performs them, no attempt will be made to discuss them as a separate subject. Rather, it would be better to discuss them as they individually become pertinent to the detailed discussion of the hardware of MAGIC which follows.

LIST

1. BLOCK TRANSFER
2. ADD
3. INSERT
4. DELETE
5. SCAN (MASKED)
6. SHIFT

NON-LIST

1. JUMP
2. REGISTER FILL
3. REGISTER EMPTY
4. CONTROL FUNCTIONS
   - A. INTERRUPT
   - B. DISPLAY DATA TRANSFER
   - C. CONTROL PROCESSOR REGISTER MANIPULATIONS
   - D. HALT/BREAKPOINT/SENSE LIGHT SET

Figure 4. Control processor instructions.

## CONTROL PROCESSOR

As stated earlier, the processing unit is divided physically and logically into a control processor and four subordinate list processors (W, X, Y and Z), each communicating with its assigned portion of the drum memory. Figure 5 is a block diagram of the control processor. Also shown are blocks representing both displays and the four subordinate list processors. The control processor contains all registers necessary for executing programs within the control processor and for controlling the subordinate list processors. These registers are the instruc-

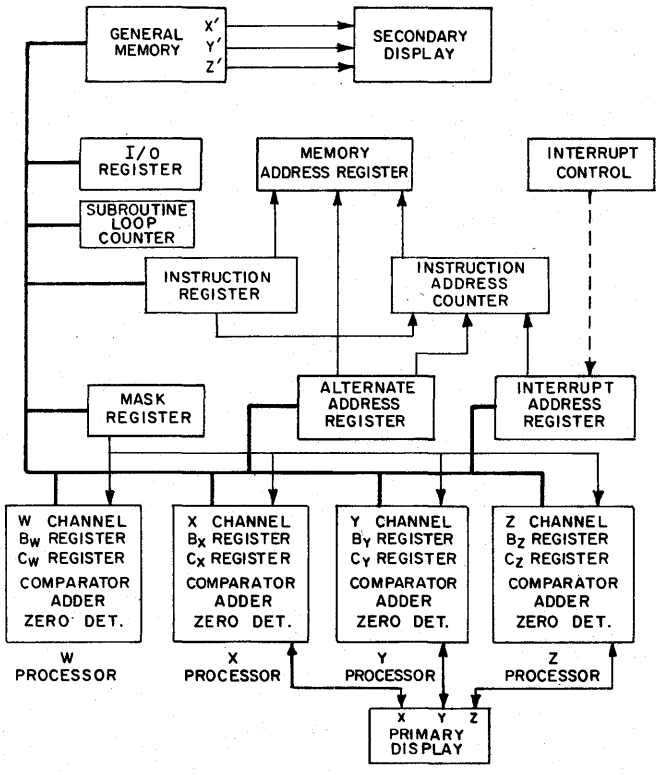



Figure 5. MAGIC block diagram.

tion register, instruction address counter, memory address register, alternate address register, mask register, interrupt address register, loop counter, and I/O register. All registers listed above with the exception of the instruction register, instruction address counter, and memory address register may communicate both ways with general memory as a result of the execution of a register fill or a register empty instruction. An additional exception to the above is the loop counter which may be filled only.

All instructions are decoded for execution from the instruction register contents. The instruction fetch phase of control processor operation loads this register. The contents of the instruction address counter determine the memory address for obtaining the next instruction from memory. Normally, this register advances by two following an instruction execution. However, the contents of this register may be forced to non-sequential addresses by jump or interrupt operations.

All memory addressing is performed via the memory address register. The channel address selects the proper general memory read/write head and the sector address is compared with the memory word counter (which is not shown in Fig. 5) to provide sector selection and timing for general memory or display memory, whichever is specified by the instruction.

The alternate address register is provided for alternate addressing. When so specified in the instruction, the sector contents of the instruction address counter may be replaced by the contents of the alternate address register before execution of the instruction. The alternate address register may be loaded via a register fill instruction, a scan operation or as a consequence of addressing a displayed object with the light pen. The alternate address register may also be incremented by the execution of an instruction for this purpose. Also, if the contents of this register correspond to sector $127_8$ (the last sector of any drum channel), the execution of an alternate address register sense and jump instruction will cause the program to jump to the address specified in the instruction. If the contents are not $127_8$, no action is taken.

Masking data for scan operations are supplied by the contents of the mask register, which is shared with all subordinate list processors for masking purposes. Associated with the mask register is an instruction which allows the mask register contents to be shifted right by one bit each time the instruction is executed. This was implemented as a programming aid for code conversion subroutines.

The interrupt address register acts as an intermediate buffer for the interrupt sector address during the execution of an interrupt instruction. The interrupt mode of operation of MAGIC is further described in the section headed System Operation.

Four general-purpose sense flip-flops (designated W, X, Y and Z) are available as programming aids. Control of these flip-flops is provided by a set instruction and a sense and jump instruction. If such a jump is performed, the involved sense flip-flops are reset.

Another programming aid is provided by the loop counter. When filled with the desired constant, it permits reiterative program loops to be performed without additional software tallying which would normally be part of the loop. Execution of the sense and jump instruction for the loop counter interrogates the loop counter for the following conditions: if the loop counter is not zero, it is decremented and the program jumps to the operand address specified in the instruction (beginning of the loop). If the loop counter is zero, no action results and the instructions are taken in normal sequence, allowing the program to break out of the loop.

Local input/output may be performed via the single length I/O register. This register is additionally linked to a special general memory I/O channel with dual read/write capabilities. With this register and its associated dual controls and sector addressing, it is possible to communicate between the I/O channel and a peripheral device without interrupting operation of the control processor. This I/O channel is also utilized by the hardware used to interface MAGIC to the central ADP system. This will be detailed later in the paper.

## SUBORDINATE LIST PROCESSORS

Digital data presented to the primary display are manipulated by the three subordinate list processors designated X, Y and Z. These processors (as well as processor W) are identical in design. Figure 6 shows the logical arrangement of one of them. The read head and write head shown in this figure are separated by exactly one word (or sector) as related to drum circumference, allowing the A register and the associated drum channel to form a nonprecessing recirculating loop. Since it is not possible to

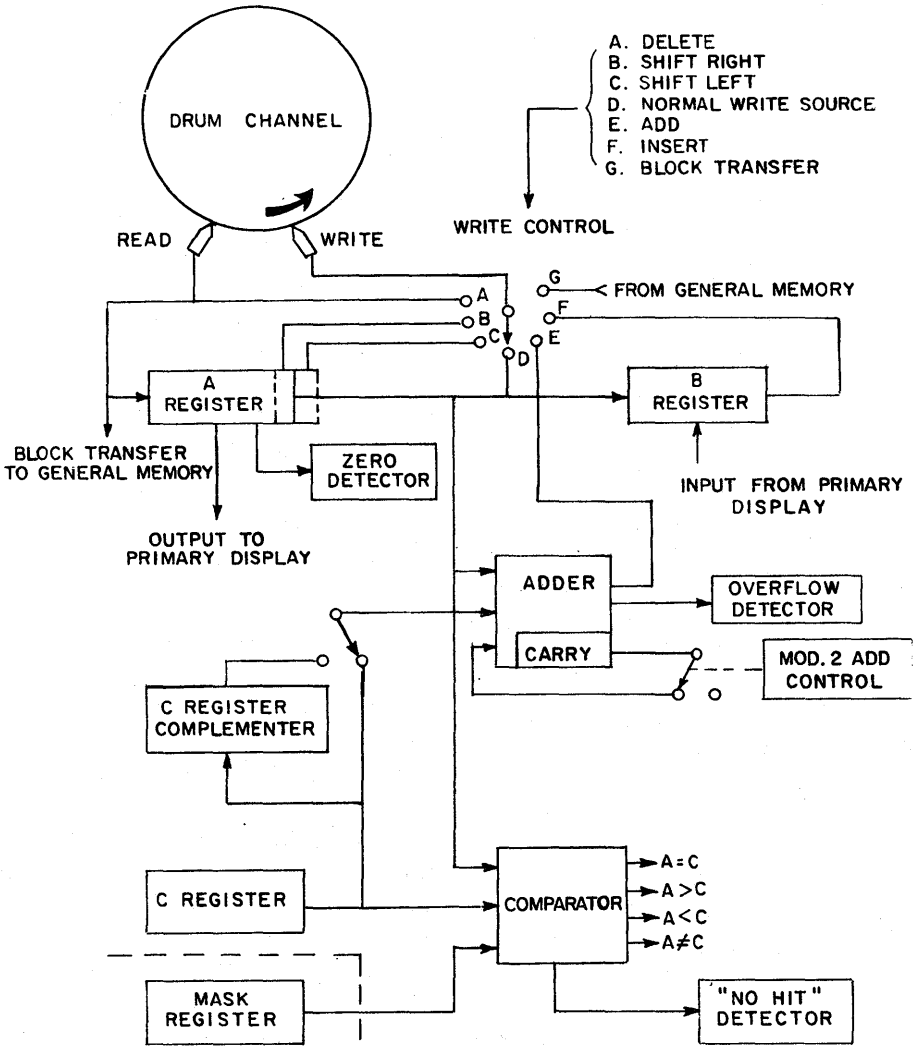



Figure 6. W, X, Y or Z subordinate list processor.

place heads this close together on the same channel (one sector occupies approximately ⅛″ of drum circumference), a pair of channels are used, with the write head on one and the read head on the other. A restoring pair of read/write heads are placed a convenient integral number of sectors around the circumference of the drum from the pair associated with the A register to close the loop. Consequently, the memory "channel" shown in Fig. 6 represents the two actual memory channels as described. However, this arrangement will continue to be considered as a single memory channel since this is its logical function.

Normally, when the subordinate list processor is inactive, data flow from the memory channel, via the read head, through the A register and the write control switch (which, when inactive, is in position D), and back onto the memory channel via the write head. During the time period for any specified sector, the data contents of this sector are being written from the A register onto the drum. To perform list manipulations on these data, the control processor switches the write control to the proper switch position for the appropriate length of time. At the end of the specified list manipulation, the write control is returned to position D. This concept of a recirculating loop with write control makes it possible to perform true list manipulations within the subordinate list processors. Reiterating, these list manipulations are: insert, delete, block transfer, shift (right or left), add and scan. The following describes these manipulations in detail.

Data are inserted into a display channel via the associated B register as shown in Fig. 6. The B reg-

ister may be filled by the control processor with data from general memory or from the locator coordinate buffers and the Z parameter switches on the operator's control panel. The word location or sector into which an insertion is to take place may be defined by the zero detector, or the sector may be defined in the instruction. Thus, the following types of insertion may be accomplished: (a) at the beginning of a list, (b) within a list, and (c) at the end of a list if the last word of the list is followed by a word containing all zeros.

At the time coincident with the beginning of the desired sector, the write control is changed to Position F, where it remains until the end of the channel. This places the B register in the recirculating memory loop, creating a total delay of two words. Consequently, the contents of the B register are written into the desired location in the list and the previous contents of that sector and all following sector contents are automatically moved down one sector to accommodate the inserted word. This represents a true insertion within a list of data, with data properly relocated by the completion of the operation.

The following table illustrates the insert operation. Here, it is assumed that the data word 7777 (octal) is to be inserted into sector 056 of a display memory channel.

Table 1. Illustration of an Insert Operation.

| Data List Before Insertion | | Data List After Insertion | |
|---|---|---|---|
| Sector | Contents | Sector | Contents |
| 054 | 0123 | 054 | 0123 |
| 055 | 4567 | 055 | 4567 |
| 056 | 7654 | 056 | 7777 |
| 057 | 3210 | 057 | 7654 |
| 060 | 0123 | 060 | 3210 |
| : | : | : | : |

Deletion is accomplished by switching the write control to position A at the beginning of the desired sector where it remains until the end of the channel. The sector is defined in the same manner as in insertion operations. This operation removes the A register and the one word delay it represents from the recirculating loop. Consequently, the contents of the specified sector are bypassed and the data in the remainder of the channel are moved up one sector position to fill the gap. As in the insert operation, this represents a true list delete. The types of deletion possible are the same as for insertion.

Table 2 illustrates the delete operation. Here it is assumed that the contents of sector 010 (octal) are to be deleted.

Table 2. Illustration of a Delete Operation.

| Data List After Insertion | | Data List After Delete | |
|---|---|---|---|
| Sector | Contents | Sector | Contents |
| 007 | 1111 | 007 | 1111 |
| 010 | 2222 | 010 | 3333 |
| 011 | 3333 | 011 | 4444 |
| 012 | 4444 | 012 | 5555 |
| : | : | : | : |

Memory to memory "block" transfers of data between display memory and the control processor general memory can occur in both directions. Block transferring from the display memory to general memory requires no action by the subordinate list processor and is under direct control of the control processor via the system's memory bus (heavy line shown in Fig. 5). Block transferring from general memory through the memory bus to display memory requires the write control (Fig. 6) to be switched to position G during the performance of this operation. A block transfer may be specified for a particular sector or from a sector to the end of the channel. This block transfer capability is generally used for storage in or retrieval from general memory of pictures or sub-pictures for presentation on the primary display.

Increasing or decreasing the size of the display by factors of two may be accomplished by shifting the data in the display channels left or right respectively. By switching the write control to position C, data from the A register are written one bit late, effectively shifting left or increasing the binary weight of each data bit by a factor of two. By switching the write control to position B, data bits are written one bit early, effectively shifting right. In either of the above cases, there is no overflow from word to word, and each execution of the operation constitutes a shift of one bit position only. Shift operations may be performed on a single specified sector or from a specified sector to the end of the channel.

An add operation within the subordinate list processor utilizes the serial full adder as shown in Fig. 6. During the execution of an add operation, the

contents of the control processor addressable C register are serially added to the A register. Also during the add operation, the write control is switched to position E and the adder sum is written on the drum as it is generated. If an overflow condition exists after any single word has been added, the overflow detector is set. A subsequent sense and jump instruction may be used to test for an overflow condition. The overflow detector is reset only at the beginning of any add operation. As implied in Fig. 6, variations in the add instruction permit Modulo 2 add or complementing of the C register before addition. Addition may be performed on a single sector or from a specified sector to the end of the channel.

Addition of a constant to X or Y coordinate data has the effect of moving the picture presented on the primary display; e.g., adding a constant to the X coordinate data list moves the picture to the right or subtracting the constant moves the picture left. Repetitive addition creates continuous movement of the picture. The speed of movement is determined by the size of the constant.

Scan operations perform table look-ups on data in display memory. During the scan operation the contents of the A register are serially compared with the contents of the C register by the comparator shown in Fig. 6. This operation requires no switching of the electronic write control. Scanned data are always masked by the contents of the mask register. The data in a specified display memory may be scanned for A = C, A> C, A <C, or A ≠ C. A scan operation is initiated at the sector specified by the instruction and continues until terminated by one of the following methods. If the scan operation is successful, the comparator output terminates the operation at the end of the sector for which the scan was successful. At this time the contents of this sector are placed in the B register and the sector address is placed in the alternate address register. If the scan operation is not successful, the operation is terminated at the end of the channel and the "no hit" detector is set. The no hit detector may be tested by a sense and jump instruction which will cause a jump to occur if the no hit detector is set. The no hit detector is not reset at this time but is reset at the beginning of any scan operation.

The salient feature of the list manipulations just discussed is that each requires a single control processor instruction which requires one drum revolution or less for execution. Since, as previously described, one bit in the control processor instruction format is reserved uniquely to each of the four subordinate list processors, a specified list manipulation may be performed simultaneously in any desired combination of these processors.

## PRIMARY DISPLAY

Dual magnetic deflection is employed by both display consoles. Object position on the CRT screen is provided by the main deflection system. The secondary deflection system provides the relatively fast, limited deflection characteristics required for alpha-numeric character and locator (primary display only) presentation. Figure 7 depicts the primary display and its associated control logic.

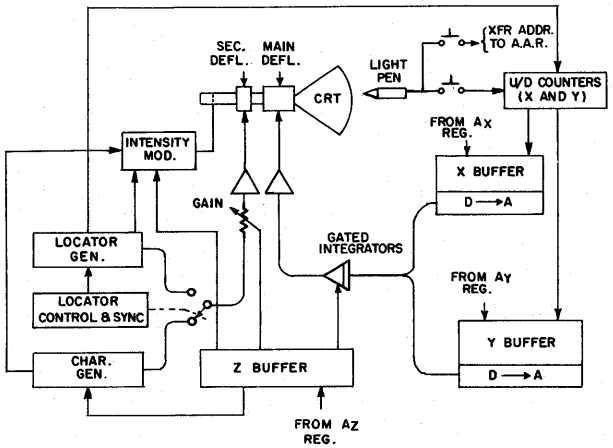



Figure 7. Primary display.

Digital display data are parallel transferred from the A registers of subordinate list processors X, Y and Z to the primary display buffers (X, Y and Z). The X and Y coordinate buffers drive digital to analog converters to provide object position. The contents of the Z buffer (see Fig. 3) are decoded to provide control information for use as indicated in Fig. 7.

Included in the main deflection system is a pair of preamplifiers which act as gated integrators for vector generation and which are controlled by decoded plot mode information from the Z buffer. When in vector mode (solid or dashed vector), these preamplifiers act as integrators. When in any other mode, they act as unity gain amplifiers.

As recalled from Fig. 3, six bits of the Z word are used by the alpha-numeric character generator. This character generator is commercially available

and utilizes a formatted stroke method of character generation. To obtain two character sizes, the character gain bit of the Z buffer controls the gain of a pair of preamplifiers in the secondary deflection system.

The digital intensity modulator receives information from the locator generator, character generator and Z buffer for generating the necessary control, timing, and intensity level information required by the various objects being displayed.

*Light Pen and Locator*

The light pen in the MAGIC system is used in conjunction with an electronically generated crosshair locator pattern for manually generating display data. The locator may be turned on or off by a manual switch on the control panel and appears on the primary display. The locator control circuit allows the locator to be generated once each revolution of the drum during the time period allocated to the last sector (sector $127_8$) of memory channel timing. Consequently, this last sector cannot be used for display data storage. Generation of each segment of the locator is synchronized with the locator coordinate up/down counters in a manner which allows the locator to servo to the light pen. The speed at which the locator servos to the light pen is proportional to the distance of the light pen from the center of the locator. The locator position is updated by transferring the contents of the locator up-down counters to the main deflection X and Y buffers immediately after each locator generation.

Once the locator has been positioned and the object at that position has been described by the Z parameter switch register on the control panel, it is necessary for the data to be transferred to the B registers of the X, Y and Z display processors for subsequent insertion. This may be accomplished manually by the depression of any keyboard key or by the execution of the display data transfer instruction by the control processor.

Another use of the light pen is for the identification of data being displayed on the primary CRT. Depression of a manual switch permits the sector address synchronous with the light pen output pulse to be transferred to the alternate address register. Subsequent processing using alternate addressing techniques permits the desired manipulation to be performed on any object pointed to with the light pen.

## SYSTEM OPERATION

*Manual Controls*

The physical appearance of the MAGIC system is shown in Fig. 8. The primary display is on the right and the secondary display is on the left. To the upper left of the primary CRT is the light pen.

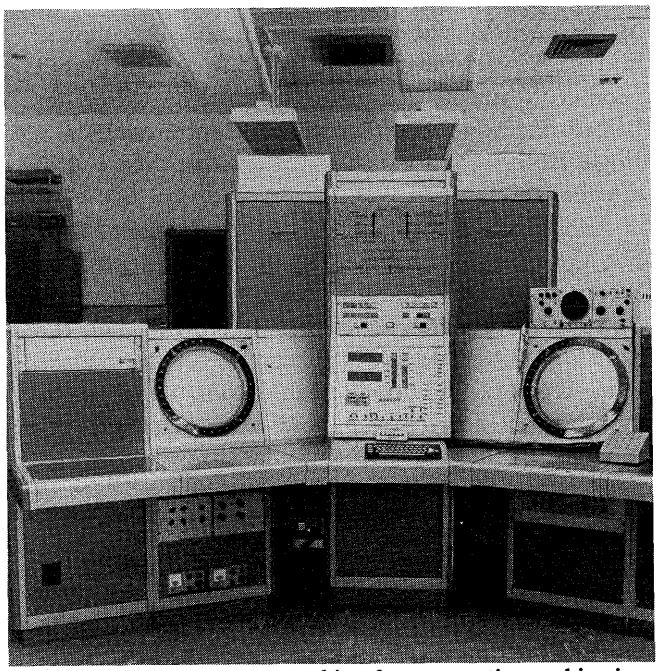


Figure 8. MAGIC — a machine for automatic graphics interface to a computer.

Panels from top to bottom of the center rack are: (1) control processor monitoring indicators, (2) control processor manual controls, and (3) display control panel containing the Z switch register, locator and light pen controls, and keyboard interrupt control. The keyboard, used primarily for alphanumeric data entry, is below this panel. The vertical rows of buttons on the right side of the display control panel and the two vertical rows of illuminated pushbuttons at top center of this panel are interrupt pushbuttons. Additional interrupt pushbuttons are on the small chassis shown at the lower right of the primary CRT.

*Interrupt Feature*

Operation of MAGIC generally consists of using the light pen and the interrupt pushbuttons to either generate display data or manipulate data already being displayed. Also, for most purposes, MAGIC

executes subroutines on an interrupt basis. This interrupt feature functions as follows. The receipt of an interrupt instruction by the instruction register causes the control processor to "hang up" in the instruction execution phase. This condition will remain until one of the 63 interrupt pushbuttons available to the user is depressed; at which time, the sector address assigned to this pushbutton is transferred to the interrupt address register. This sector address is then transferred to the instruction address counter and the operation is simultaneously terminated. The next instruction is taken from this new sector address and normally consists of an unconditional jump instruction which is one of a list of links to display manipulation subroutines; each link corresponding to an interrupt button. Thus the interrupt pushbuttons are programmable and their meaning may be changed by modifying the list of unconditional jump instructions associated with the interrupt jump instruction.

It follows that any particular interrupt pushbutton may mean different display manipulations for different users. Therefore, annotated overlays are used for describing the functions of the pushbuttons. About half of the interrupt buttons are designed to operate continuously, i.e., the subroutine accessed by one of these interrupt buttons is repeated as long as the button is depressed.

*Subroutine Examples*

The following examples of simple display manipulations are accessible by interrupt as previously described. Each subroutine, therefore, is linked back to the interrupt instruction by an unconditional jump instruction. Also, the following examples are representative only; a complete discussion of MAGIC software would be beyond the scope of this paper. However, detailed programming and operating procedures for MAGIC are described in National Aeronautics and Space Administration document number N65-25010: "MAGIC—A Machine for Automatic Graphics Interface to a Computer" (reference manual) by D. E. Rippy (29 Dec. 1964).

A typical insert subroutine generally consists of two machine instructions: an insert instruction, and a jump instruction for return to the interrupt. An insert may be performed at the beginning of a display list by specifying the beginning sector of that list in the insert instruction.

Insertion within the display list utilizes alternate addressing and is generally executed in the following manner. The user points with the light pen at the displayed feature (line, point, etc.) within the picture where it is desired that new graphic data be inserted. Depression of the light pen address switch loads the alternate address register with the sector address coincident with that feature. The Z switch register contents (assumed previously set for the desired parameters), keyboard buffer contents, and the locator coordinates (from the locator up-down counters shown in Fig. 7), are then transferred to the $B_x$, $B_y$ and $B_z$ registers. This is accomplished either by depressing any keyboard key or by the execution of a display data transfer instruction which would be included in the subroutine. The choice is, of course, dictated by user preference.

The interrupt button assigned to this subroutine is then depressed. In this case, the insert instruction will include the alternate addressing code, allowing the alternate address register contents to govern the sector location within the display list where insertion is to take place.

The A register zero detector (Fig. 6) is utilized for insertion at the end of a display list. The end of a display list is defined as the first sector of a display memory channel in which no data exist. In this case, the execution of the insert instruction monitors the zero detector as the display list flows through the A register. If the display list in question is followed by at least one sector containing no data, a signal from the zero detector initiates the insert process at that sector.

A typical delete subroutine also consists of two or three machine instructions, depending on whether or not a display data transfer instruction is included. Deletion at the beginning of, within, or at the end of a display list is performed as described for the insertion subroutines described above.

A subroutine to move a picture or subpicture across the CRT face requires three machine instructions. The first instruction fills the $C_x$ or $C_y$ register with the desired constant to be added to the X or Y coordinate data. The add instruction is performed next. This may be a complement-before-add instruction (as previously described) for such purposes as creating movement left or down. Finally, an unconditional jump instruction returns program control to the interrupt instruction.

Movement at any angle may be accomplished by the addition of another fill instruction which would

allow the $C_x$ and $C_y$ registers to be filled with different constants. Simultaneous add instruction execution may then be performed by the X and Y subordinate list processors if both X and Y are specified in the add instruction (see Fig. 2).

To tie the last point (or line) in a display list to any preceding point (or line) in that list with, generally, a line, requires three machine instructions. The user identifies the feature he wishes to tie to with the light pen and depresses the interrupt button used for this "Tie" subroutine. The X and Y coordinate values in the sector location just identified are transferred to the $B_x$ and $B_y$ registers, respectively, by the first instruction, utilizing alternate addressing. An end-of-list insert instruction is performed next, resulting in a new line being drawn to the identified location. The remaining instruction is the usual unconditional jump to the interrupt instruction.

Calling up whole pictures or subpictures after they have been generated, stored in general memory, and identified by interrupt buttons and associated linkage software requires a subroutine of four machine instructions. These are three blocks transfer instructions (one each for the X, Y and Z lists) and the usual unconditional jump back to the interrupt.

On a somewhat larger scale, a "visual assembler" program has been devised, requiring (irrespective of look-up tables and picture storage) 103 machine instructions. Upon entry into this program, a picture is displayed on the primary display consisting of alpha-numeric mnemonics representing all of the machine instruction and the four subordinate list processors, a number table for composing channel and sector addresses, and mnemonics used for control purposes. The processor then goes to a light pen interrupt mode. The user then points to a displayed mnemonic with the light pen and depresses the light pen address switch. By alternate addressing, the program is then directed to the proper subroutine for assembly if the binary equivalent of the mnemonic pointed to. This process is continued; assembling the operation and address fields in their proper locations to result in a machine instruction or, as desired, a numeric data word. Various control mnemonics allow such operations as assembly of machine instructions or data words, error correction and transfer of the assembled information to its permanent general memory location. More sophisticated programs of this nature will be generated in the near future for compiling programs for MAGIC and for the central computer.

## MAGIC-CENTRAL PROCESSOR COMMUNICATION

By November 1965 MAGIC will be interfaced to a large ADP system which is particularly oriented toward time-sharing communication with a large number of peripheral devices. These devices will be of varying complexity, ranging from simple teletypes to more complex devices such as MAGIC or satellite computers. Such an arrangement will allow further investigation of time sharing, interfacing and on-line processing techniques. Where MAGIC is particularly concerned, various experiments will be performed involving applications of a display device which require the large data base and the faster, more sophisticated computing abilities provided by a large ADP system.

The central processing system to be used for this purpose is a MOBIDIC B twin computer and the NBS PILOT multicomputer, each computer having the ability to communicate with the other. Four magnetic tape units and two disc files provide a large data base for program and data storage. Minor modifications to the I/O Converters associated with each processor and to the I/O interrupt logic have greatly increased the time-sharing capabilities of the system.

The interface associated with MAGIC utilizes a voice quality, half-duplex line. The data rate has been set at 2.4 kHz. Although the data rate could be considerably higher, the use of voice quality lines is generally a necessity for many remote station applications as dictated by availability and economic considerations.

Two forms of intercommunication between MAGIC and MOBIDIC are used: one for communication of graphic data to and from display channels X, Y and Z, and one for communication of programs and related data or control information to and from the I/O channel in the control processor. When used on line with MOBIDIC, MAGIC is assumed to be in the receive mode except when actually transmitting. In MAGIC, reception of data is concurrent with control processor operation. MOBIDIC is similarly assumed to be always in the receive mode. However, unlike MAGIC, MOBIDIC

utilizes an interrupt for recognizing data receipt when operating in the time sharing mode.

Transmission of blocks of data from either MAGIC or MOBIDIC is program initiated. In MAGIC, two machine instructions are used for initiating transmission: one for display data and one for control information or programs as related above. Once initiated, the transmission process will continue until self-terminated without further interruption to MAGIC.

MAGIC-MOBIDIC interactive display data processing will generally adhere to the following philosophy. MAGIC will provide local storage and basic manipulation of display data under direct user control. MOBIDIC will provide a large data base for storage of graphic data and graphic data processing routines for both MAGIC and MOBIDIC. MOBIDIC graphic data processing will be relegated mainly to relatively complex functions; i.e., curve fitting, geometric calculations, generation of graphic solutions to mathematical equations, etc. Detailed descriptions of the programming required for this interactive graphic data processing, the interfacing hardware required and the over-all system configuration are beyond the scope of this paper, but will be presented in forthcoming papers by various members of the section.

## CONCLUSIONS

MAGIC is an operating display system. The preceding paragraphs have described the salient hardware, software and operational features as of May 1965. The remaining questions are: What has been learned from the present system, and what of the future?

First of all, it can be said that the concepts of list processing have proven to be desirable in the manipulation of display data. Second, the hardware and software design of MAGIC as described provides sufficient local processing abilities to significantly remove the burden of display data processing from the C.P.U. to which it is to be interfaced. Software has proven to be quite efficient, and operation of MAGIC using the programmed interrupts, keyboard and light pen has proven to be straightforward and very easy to learn.

Flexibility of original design has provided considerable latitude in experimentation with the hardware and software of MAGIC. As was expected and desired, this has brought to light a number of excesses and deficiencies in the hardware and software areas.

Three major excesses in hardware design have been determined. It has been found that there is no great need for a secondary display unit except in special cases such as when MAGIC may be used as a teaching machine. The serial adders in the subordinate list processors, elementary as they are, may be further simplified by eliminating the complement-and-add feature. The use of this operation has proven to be negligible. Subordinate list processor W may be essentially eliminated except for block transfers, since it has proven to be of little use in display data manipulation. Many of the features in processor W could be directly implemented within the control processor.

Hardware deficiencies have been minor except in the analog area. More work is required on the vector generators and consideration is being given to the implementation of a circle generator. The major problem in the analog area concerns boundary conditions involving portions of vectors which extend beyond the display boundaries. Such a problem arises in magnifying to full screen size small portions of a displayed picture. Preliminary investigation indicates that a combination of digital and analog techniques will provide the necessary intensity blanking and vector segmentation for objects extending beyond the visible boundaries automatically, with little or no required software.

In the software area, certain basic deficiencies have arisen. The instruction format, as it stands, does not fully describe a list; it only specifies the initial address of a list and the list is assumed to terminate at the end of the channel. For the list manipulations (block transfers excepted) a minor addition to the hardware of the control processor will eliminate this difficulty. However, the list manipulations as they stand have proven to be quite efficient in manipulating display data and the difficulty described above does not detract from the concepts of list processing incorporated in MAGIC.

As for the subordinate list processors, it has been found that additional register-to-register transfers would be helpful in decreasing the software required to directly enter and retrieve data from the display channels. It has also been found that the implementation of hardware and software for direct addition of general memory data to data in the display channels would be of considerable help in re-

ducing software required for magnification or division of display picture size, subpicture assembly and manipulation, and the like. However, even when these minor software deficiencies are taken into account, the machine language programming of MAGIC is straightforward and has proven to be surprisingly minimal.

Future additions to MAGIC in respect to the hardware and software deficiencies described above will be performed on an experimental or "spare time" basis. This is dictated by the following considerations. First, as it may be inferred from Fig. 8, the actual hardware used in the construction of MAGIC consists of surplus equipment which, by today's standards, considerably lags the present state-of-the-art. Consequently it has been decided that there is little value in attempting to increase the performance of this Model I MAGIC which is already operating at maximum capability. However, this has in no way affected the implementation of the concepts of display equipment design as discussed in this paper.

Second, preliminary design work on a Model II MAGIC is in progress and final design and construction will begin in the fall of 1965. Model II will incorporate state of the art hardware components and will be designed to eliminate the previously described hardware deficiencies of Model I. This will result in greatly improved operating speed, display quantity and quality, and list manipulating capabilities. It will also be interfaced to same central computer as Model I. The final design of the interface hardware for Model II will, of course, be dictated from what is learned from the implementation of the interface for Model I.

## ACKNOWLEDGMENT

The development and programming of MAGIC has been sponsored by the National Bureau of Standards and the National Aeronautics and Space Administration. Every member of the Computer Technology Section has contributed to the success of this project, but particular appreciation is extended to James A. Cunningham and Paul A. Meissner for their valuable counsel concerning all aspects of the project.

## REFERENCES

The following represents a collection of general subject material concerning man-machine communication using display devices.

1. J. C. R. Licklider and W. F. Clark, "On Line Man-Machine Computer Communication," *Proc. S.J.C.C.*, vol. 21, p. 113 (1962).

2. H. H. Loomis, Jr., "Graphic Manipulation Techniques Using the Lincoln TX-2 Computer," Report No. 516-0017(U), Lincoln Laboratory, M.I.T., Cambridge, Mass. (November 1960).

3. E. E. Sutherland, "Sketchpad: A Man-Machine Graphic Communication System," *Proc. S.J.C.C.*, vol. 23, p. 329 (1963).

4. T. R. Allen and J. E. Foote, "Input Output Software Capability for a Man-Machine Communication and Image Processing System," *Proc. F.J.C.C.*, vol. 26, p. 387 (1964).

5. R. Stotz, "Man-Machine Console Facilities for Computer Aided Design," *Proc. S.J.C.C.*, vol. 23, p. 323 (1963).

6. B. Hargreaves, J. D. Joyce, G. L. Cole, et al., "Image Processing Hardware for a Man-Machine Graphical Communication System," *Proc. F.J.C.C.*, vol. 26, p. 363 (1964).

7. P. B. Lazovick, et al., "A Versatile Man-Machine Communication Console," *Proc. E.J.C.C.*, vol. 20, p. 166 (1961).

8. T. E. Johnson, "Sketchpad III: A Computer Program for Drawing in Three Dimensions," *Proc. S.J.C.C.*, vol. 23, p. 347 (1963).

9. D. T. Ross and J. E. Rodriquez, "Theoretical Foundation for the Computer-Aided Design System," *Proc. S.J.C.C.*, vol. 23, p. 305 (1963).

# A MAGNETIC DEVICE FOR COMPUTER GRAPHIC INPUT

M. H. Lewin
*RCA Laboratories, Radio Corporation of America*
*Princeton, New Jersey*

## INTRODUCTION

Recent work on systems to facilitate the input of graphical information to a computer has resulted in the development of the light pen[1] and the Rand tablet.[2] Both of these devices allow a user to "write" on a flat surface with a special, hand-held electronic pen. Periodically, the pen position is detected and converted into a machine-readable address. In this way, the pattern which is traced out by the pen is directly converted into binary code and stored in the machine. Devices such as these promote the easy input of graphical data such as curves, maps, diagrams, and other drawings. They should also be of interest to many researchers concerned with character and pattern recognition.

The light pen is normally used in conjunction with a cathode-ray tube as the writing surface. A light-sensitive element in the pen generates a signal when the flying spot on the tube face reaches the pen tip. The timing of this signal, relative to the timing of the scanning pattern, establishes the pen position. Appropriate digital and analog peripheral circuits are necessary to convert this signal into an equivalent binary address for storage. Clearly, the speed of movement of the pen is limited by the scanning frame rate of the CRT. Also, one cannot insert between the CRT face and the pen any material (such as a sheet of paper) which will prevent light transmission.

The Rand tablet consists of a thin Mylar sheet containing on one side, an array of etched copper lines in the X direction and, on the other side, a similar array of fine lines in the Y direction. By means of capacitor encoding networks, also etched on the same sheet, a unique voltage pulse train is applied to each X and Y line from a common pulse pattern generator. The pen in this case is merely a metallic electrostatic pickup connected to a high input-impedance amplifier. The pulse train picked up by the pen depends on the X and Y lines nearest to its tip. This serial pulse pattern (in Gray code to eliminate errors) is converted into a parallel binary address with appropriate peripheral logic, which includes a shift register and a code converter. The system is entirely digital and the tablet is relatively inexpensive. In addition, thin paper sheets can be inserted between the tablet surface and the pen for tracing maps and curves.

Both of the approaches described above utilize the pen as the signal pickup device and the writing surface as the signal generator. While the Rand tablet system materially simplifies the writing surface used and reduces the complexity of the peripheral

electronics required for a given pen position resolution, the amount of circuitry needed, for the generation of the appropriate pulse sequences and for the conversion of detected pulse sequences into parallel binary addresses, is not negligible.*

The work on which this paper is based was initiated to develop a graphic input device which would require a minimum of associated circuits, while maintaining simplicity in the construction of the writing surface. The system to be described utilizes the pen as the signal generator and the writing surface as the address detector. The pen contains in its tip a small magnetic head which periodically generates a localized magnetic field pulse. (Since the coupling is magnetic, it is not shielded by most materials placed between the pen and the tablet.) The writing surface contains a number of thin winding layers in a laminated structure. Each winding layer consists of a single, continuous wire pattern designed to detect one of the pen address bits. Thus, there are as many layers as there are address bits, each developing a positive or negative induced voltage as a function of the pen position. All layers generate output pulses in parallel and these signals are of sufficient magnitude to set a register directly.

## WRITING SURFACE WINDING PATTERN

The magnetic head in the pen tip consists of a small, linear ferrite core with an air gap and winding as indicated in Fig. 1. The coil is periodically driven with a voltage pulse as shown. Any wire, brought in the vicinity of the air gap and oriented so as not to be perpendicular to the air slot, will link some of the magnetic flux generated and will thus develop an induced voltage pulse whose shape is similar to that of the drive signal. The polarity of the induced voltage is determined by the familiar right-hand rule. Its magnitude is greatest when the wire is parallel to the slot.

face is divided into two sets of areas or sectors, which may be labeled "even" and "odd." When the pen tip is positioned over any one of the even sectors, a positive voltage pulse (binary "one") is induced across the two winding terminals. When the pen is over any of the odd sectors, a negative out-

Consider a wire winding pattern in a plane surface over which the pen is "writing." The pen tip is

*The authors state that the system "contains some 400 transistors and about 220 diodes; however, little attempt has been made to minimize the number of components."

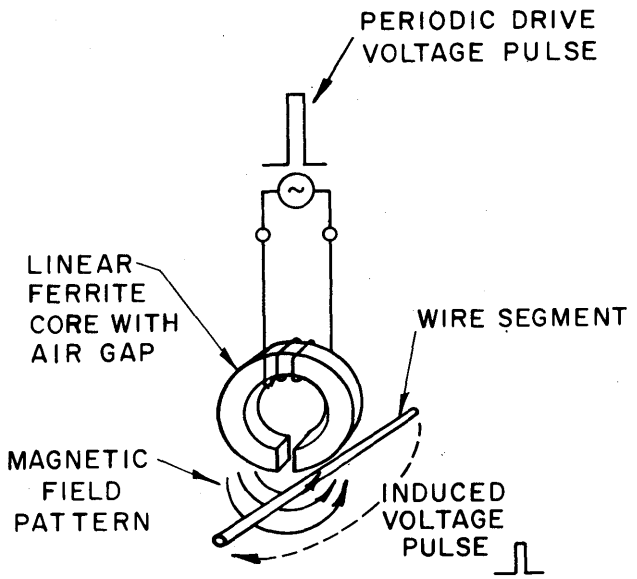



Figure 1. Magnetic head in pen tip.

in close proximity to the winding layer. It is desired to arrange the winding pattern so that the surput signal (binary "zero") is obtained. A winding configuration which will satisfy these requirements is shown in Figs. 2 and 3.

Assume the plane surface is divided into m sectors (m an even number), half even and half odd. The odd and even sectors alternate and can be labeled 1, 2, . . . $m$ as shown in Fig. 2. Each sector consists of $n$ winding "stripes" or wires, each of which is a segment of the total length of wire used in the winding. Let the stripe $ij$ be the $j^{th}$ stripe of the $i^{th}$ sector, where $1 \leq i \leq m$ and $1 \leq j \leq n$.

The procedure for laying out the winding pattern is shown in Fig. 3. In making a given winding "pass" over the surface, from left to right, one winds the wire vertically up in a specified stripe position, then continues the winding horizontally to the right to the next designated stripe position, then winds the wire vertically down, then horizontally to the right, then vertically up, . . . etc., until the right end of the plane is reached. The wire is then returned horizontally from right to left and another pass is started from left to right. The procedure indicated in Fig. 3 is summarized in Table 1. As a simple example, the pattern for four sectors, each containing four stripes, is given in Fig. 4. An examination of this winding configuration reveals that

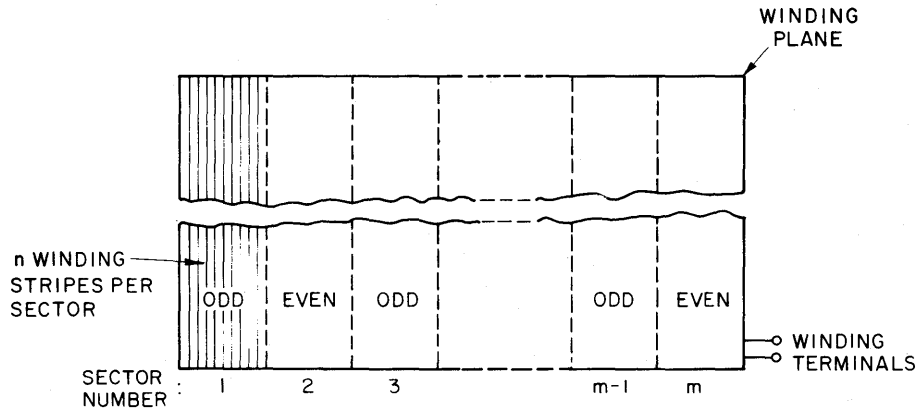



Figure 2. Partitioning of a winding plane.

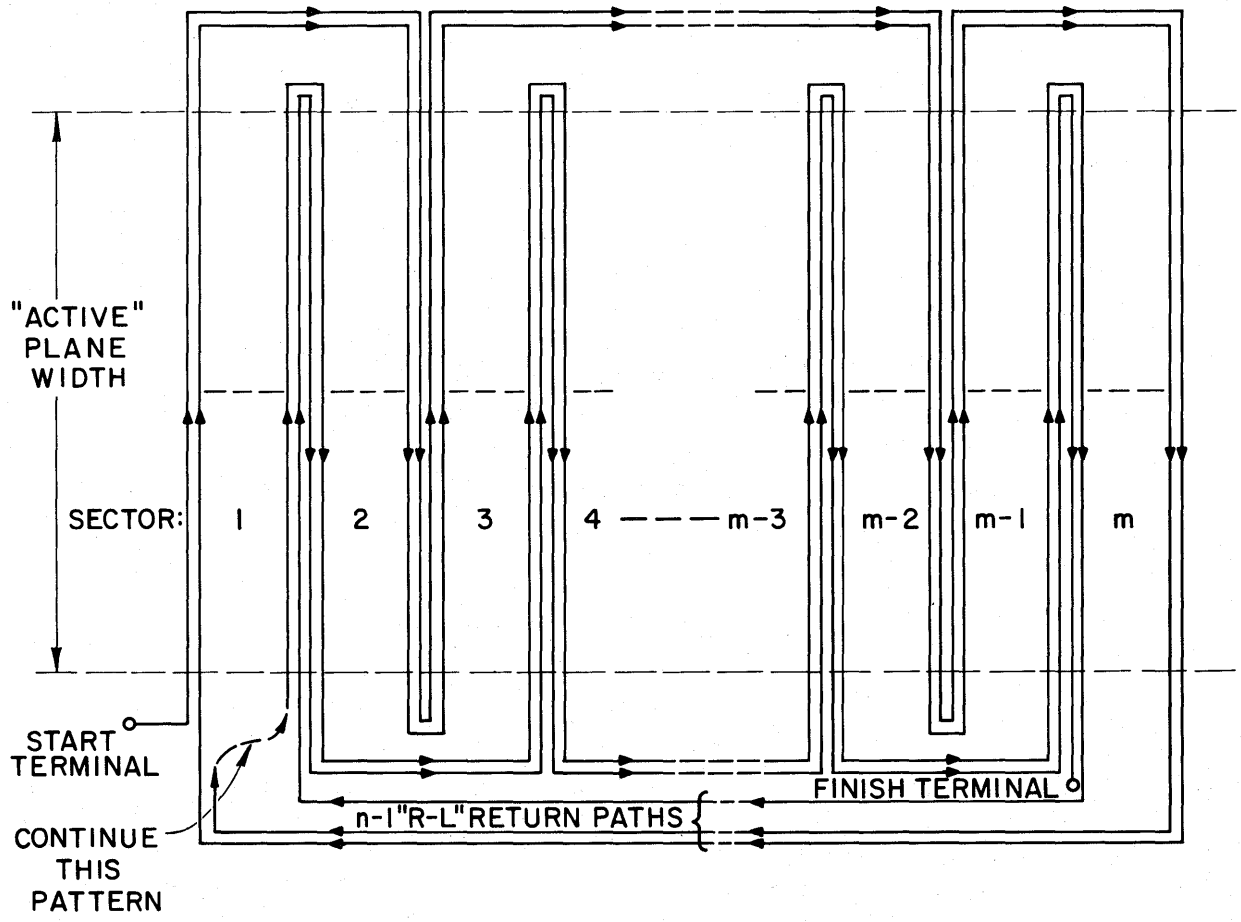



Figure 3. Winding layer pattern.

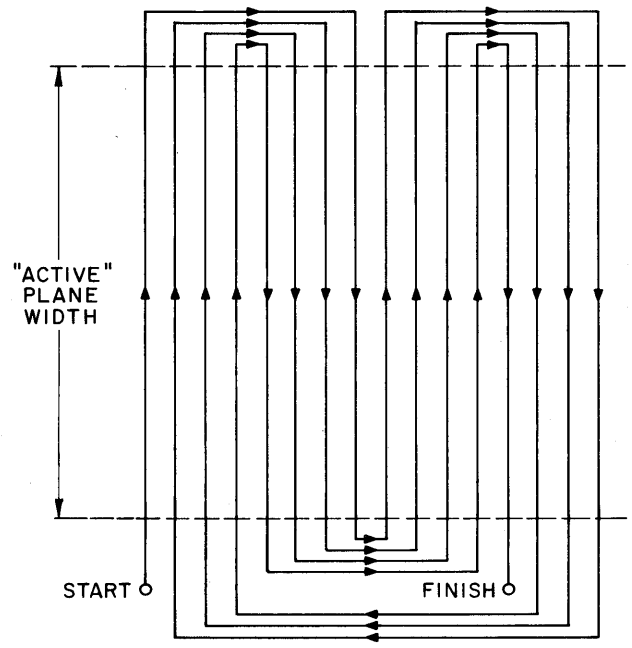



Figure 4. Configuration for m = 4, n = 4.

all stripes in the odd sectors have the same sense, opposite to that of the stripes in the even sectors.

Figure 5 shows a side view of two adjacent sectors, with three possible head positions indicated. The field pattern for position A causes a given polarity signal to be developed across the winding terminals. For position C, because the sense of the windings is reversed, the opposite polarity signal will be induced. If the pen tip is in the immediate vicinity of the boundary between the sectors (position B), very little signal will be generated since positive and negative components will cancel. Thus, the output pulse polarity determines whether the pen is over an odd or an even sector. The number of stripes ($n$) required in a sector depends on the desired output pulse magnitude. As $n$ increases, for a given sector width, the induced signal increases.

The winding pattern shown in Fig. 3 is interesting because it contains no wire crossovers. It can therefore be photoetched on a thin, conductor-clad insulator sheet, such as copper-clad Mylar, or otherwise deposited via screening or evaporation techniques on a thin insulator substrate. Two patterns can be placed on either side of a given sheet.

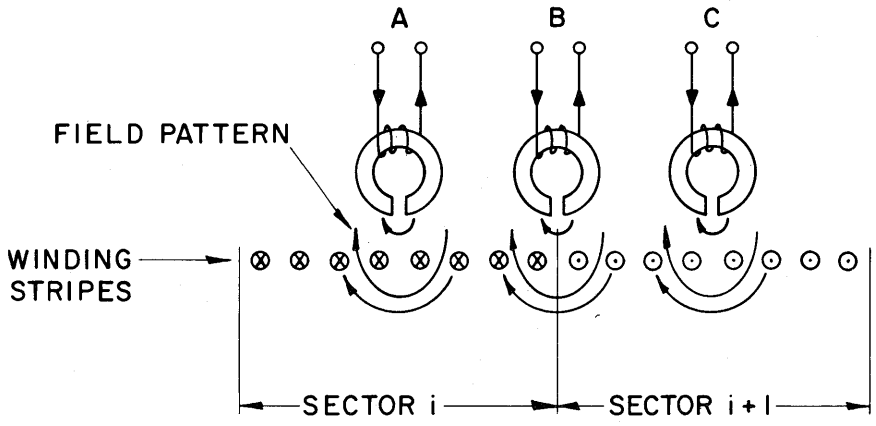



Figure 5. Side view of winding stripes with three possible head positions.

## MULTILAYER TABLET

The writing surface is constructed by stacking or laminating as many thin winding layers as there are pen address bits. Thus, for a tablet to resolve any one of 1024 × 1024 locations, ten-double-sided sheets are required. Half of the winding layers are oriented in the X direction, half in the Y direction. The total thickness of the system can be kept small by using sheets only a few mils thick. Each X layer has an identical companion Y layer oriented orthogonal to it.

The layout of a winding pattern to detect a given address bit is, of course, a function of the position-to-address coding scheme used. By using a closed, cyclic code, such as Gray code, one is as-

sured that no more than one address bit in a given coordinate direction can be undecided. That is, for any pen position, the head can be located over, at most, one boundary between sectors. For these reasons, it would appear that a conventional binary coding scheme should not be used because the pen point may be positioned over more than one indecision boundary. However, the addition of a small amount of external logic, no more complicated than that required for a parallel Gray-to-binary conversion, may allow a conventional binary code to be used. (The indecision correction algorithm involved is described in the next section.) Assuming such a code, winding patterns can be laid out as illustrated by the simple $8 \times 8$ example shown in Fig. 6. The "most significant" X or Y layer has only two sectors, the next four, then eight, etc. The total number of X or Y winding layers (address bits) depends on the resolution required in the location of the pen tip. The "least significant" layer (the one with the largest number of sectors) may have only one stripe per sector (i.e., $n = 1$).

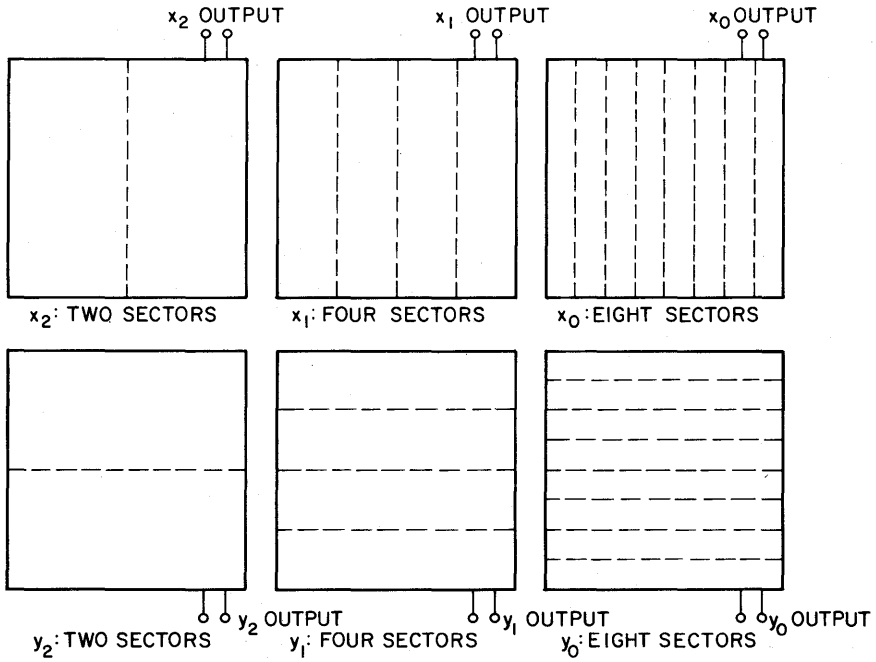



Figure 6. Six winding layers for a 64 ($8 \times 8$) position array using conventional binary coding.

All of the winding layers must be close to the pen point to allow the generation of sufficiently large output signals. The output voltage induced in a more significant layer (one with many stripes per sector), when the pen is over a given sector, is the sum of the voltages induced in all the stripes of that sector (refer to Fig. 5). This integrating effect allows one to locate a more significant layer at a distance from the pen tip which is larger than that for a less significant layer. Thus, the tablet is laminated with the most significant winding layers at the bottom and the least significant layers nearest to the surface.

Note that, in order to detect approximately equal magnitude signals at the outputs of the X and Y layers, the air gap in the magnetic head must be oriented at 45° to the X or Y orthogonal stripes. Thus, the pen must be marked or shaped appropriately to insure that it is held roughly in the correct orientation. Small variations from 45° will not change the induced signals appreciably.

Other more sophisticated head designs, which allow the system to operate independently of pen orientation, are possible. For example, one can use more than one air gap in the pen tip. Using two orthogonal gaps (pulsed at different times)* as the pen orientation changes, the magnitude of the induced signal from one gap increases while that from the other decreases. Orthogonal gaps can also be used to generate a rotating magnetic field. Another method† involves using two or more air gaps to

*Suggested by J. A. Rajchman.
†Due to J. Avins.

generate a number of discrete field orientations (say, three), one of which will always be acceptable for any pen orientation. Periodically, these orientations are sequentially tested and the acceptable one is chosen. This testing may involve the use of an additional test winding layer whose stripes are all oriented at 45° to the X and Y stripes. Each of these arrangements, however, increases the complexity not only of the magnetic head but also of the peripheral electronics. At this stage, the requirement of proper pen orientation, which allows the system to be very simple, does not appear to be a very severe user restriction. If necessary, some simple mechanical approach, such as housing the head at the end of a flexible shaft (similar to that used in speedometer cable), would permit the sleeve of the pen to rotate while the head orientation stays relatively fixed.

## INDECISION CORRECTION ALGORITHM

For a system such as the one described above, there are actually three possible output signals (call them one, X and zero) available from each layer. One and zero are acceptable signals (positive and negative pulses). X represents almost no output pulse—an undecided bit. An examination of the one, zero transitions, when counting in conventional binary code, will show that if one follows the following simple rules, errors at multiple-transition boundaries can be resolved and the conventional binary pattern can be used:

> Detect the most significant bit which is undecided (i.e., the most significant X output). Arbitrarily decide this bit to be one or zero. *Force* all less significant bits to be the complement of the bit chosen above.*

The addition of a very small amount of external logic will allow this procedure to be used. For example, in the circuit shown in Fig. 7, an undecided output is arbitrarily decided as a zero and all less significant outputs are forced to be one.

*This method will work provided that the winding patterns are designed such that, for any two adjacent address bits having a transition boundary in the same position, the "zone of indecision" for the more significant bit overlaps that of the less significant bit.

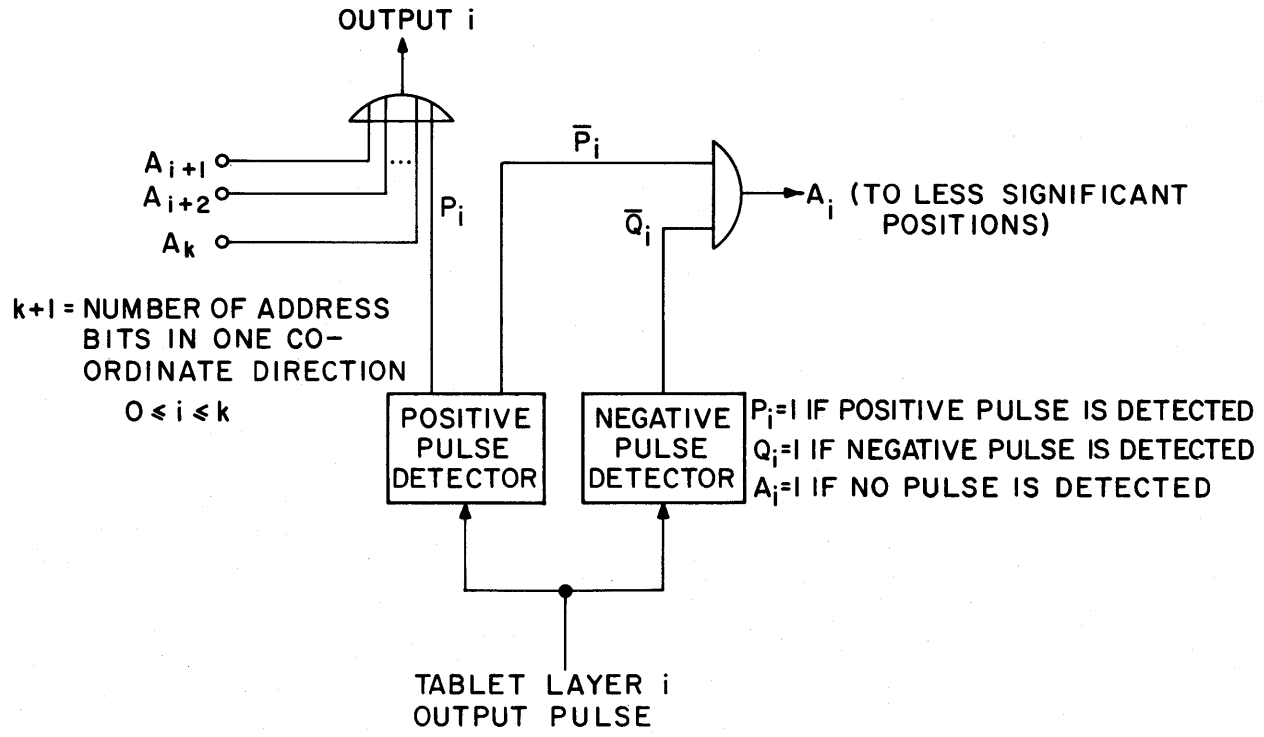



Figure 7. Mechanization of indecision correction algorithm for conventional binary address coding.

## EXPERIMENTAL MODEL

An initial experimental model consisting of a 32 × 32 array, using ten winding layers (five X and five Y) to resolve any one of 1024 pen positions, has been constructed and is operating as described above. A photograph of the pen and tablet is shown in Fig. 8. The winding layers for this model were wound by hand using conventional No. 33 insulated coil wire. Each of the layers follows the configuration given in Fig. 3. A conventional binary code was used. The windings were potted with an epoxy resin to allow the tablet to present a flat surface to the pen. The stripes are ⅛″ apart, and the total thickness of the ten-layer system is approximately 0.1″. Clearly, a much higher stripe density is achievable using present photoetching techniques. Also, one can easily laminate a ten double-sided sheet system, required for a 1024 × 1024 array, and obtain a total thickness less than 0.1″.

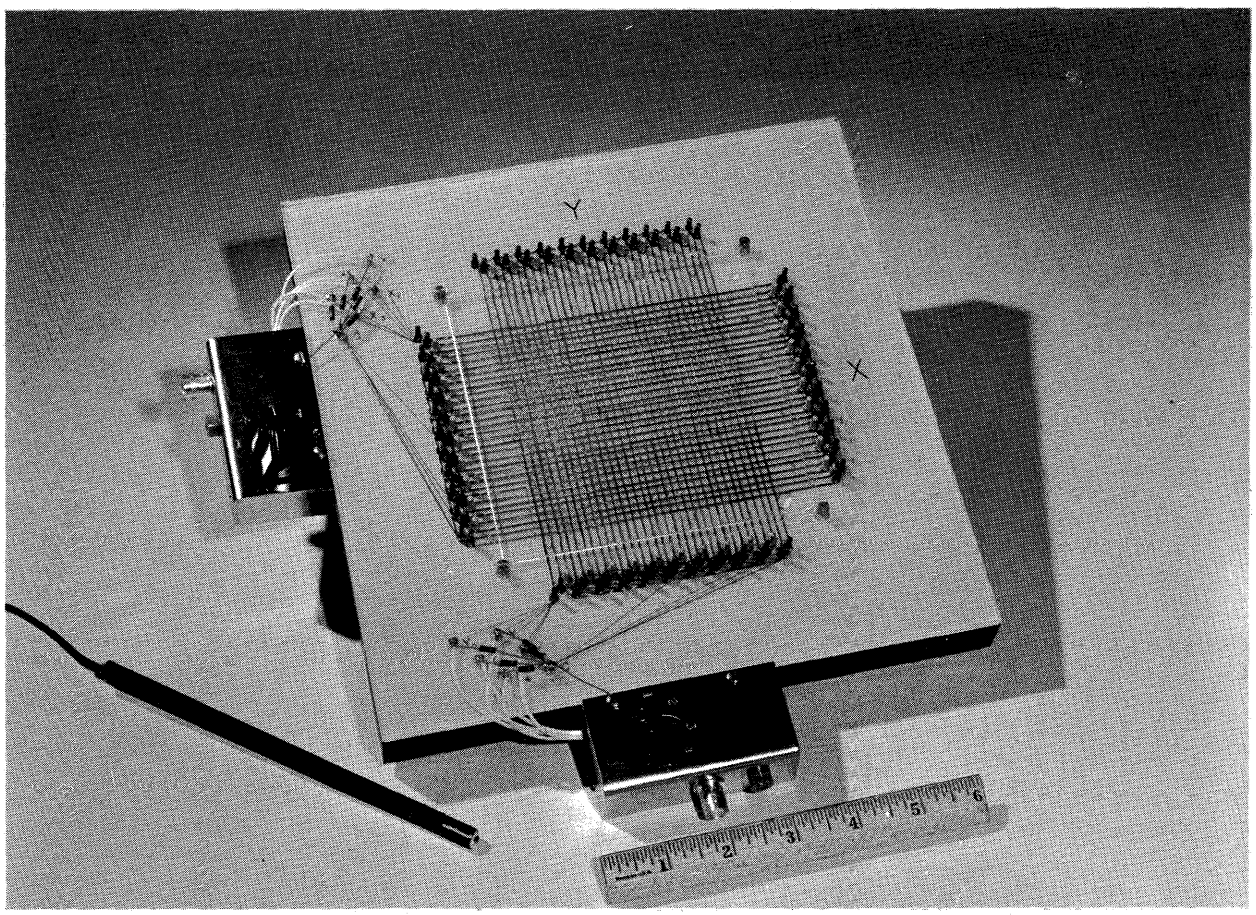



Figure 8. Experimental 32 × 32 tablet with pen.

The pen tip contains a linear ferrite core, 3/16″ O.D. and ⅛″ I.D., wound with 30 turns, and driven from a conventional General Radio pulse generator. Approximately 100 volts is developed across the head winding during the pulse peak. The core has a 15 mil air gap. Little attempt was made to optimize the core drive circuit so as to obtain optimum output signals. Each of the winding layers is terminated in 100 ohms. This value was chosen to critically damp output ringing. A photograph of a typical output impulse is shown in Fig. 9. The reverse polarity signal has the same shape. The waveform is clean and has sufficient amplitude to set a flip-flop. It can no doubt be made larger with appropriate pen drive circuit design. The timing indicated shows that one need not be concerned with the speed of movement of the pen. The pen is marked to permit proper

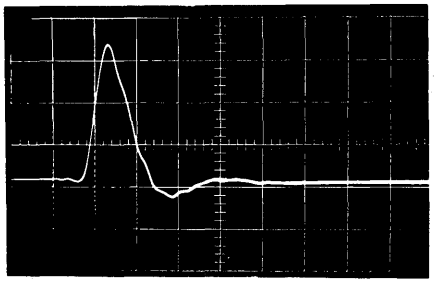


Figure 9. Typical winding output pulse across 100 ohms. Vertical scale: 0.1 volt/div. Horizontal scale: 0.2 μsec/div.

orientation of the air gap with respect to the winding stripes.

A set of ten peripheral circuits, which includes the logic given in Fig. 7 and which also contains digital-to-analog converters, is used to demonstrate the operation of the tablet by permitting the position of the pen to be displayed as a spot on a CRT face.

## CONCLUSIONS

By constructing the writing surface as the superposition of a number of patterns, each of which is designed to detect one of the pen address bits directly, the amount of associated circuitry is minimized. Although the initial artwork involves the laying out of as many patterns as there are address bits in one coordinate direction, subsequent fabrication of a number of tablets should be simple and inexpensive.

## ACKNOWLEDGMENTS

The author wishes to express his appreciation to H. Schnitzler, who constructed the experimental devices and who assisted in many of the tests.

## REFERENCES

1. B. M. Gurley and C. E. Woodward, "Light-Pen Links Computer to Operator," *Electronics,* pp. 85-87 (Nov. 20, 1959).

2. M. R. Davis and T. O. Ellis, "The Rand Tablet: A Man-Machine Graphical Communication Device," *Proc. 1964 Fall Joint Computer Conference.*

Table 1. Winding procedure.

| | "UP" IN STRIPE POSITION NUMBER | "DOWN" IN STRIPE POSITION NUMBER |
|---|---|---|
| L-R PASS NO.1 (START) | 11 | 2n |
| | 31 | 4n |
| | 51 | 6n |
| | ETC. | |
| | (m-1)1 | mn |
| RETURN R-L | | |
| L-R PASS NO.2 | | |
| | 12 | 2(n-1) |
| | 32 | 4(n-1) |
| | (m-1)2 | m(n-1) |
| RETURN R-L | | |
| | CONTINUE THIS PATTERN | |
| L-R PASS NO. n | | |
| | 1n | 21 |
| | 3n | 41 |
| | (m-1)n | m1 (FINISH) |

# GRAPHIC 1 — A REMOTE GRAPHICAL DISPLAY CONSOLE SYSTEM

William H. Ninke
*Bell Telephone Laboratories, Inc.*
*Murray Hill, New Jersey*

## INTRODUCTION

Graphical information exchange using direct-view display consoles is a rapidly growing means of communicating between a human and a computer. One reason for this growth is that results presented to a human in graphical form are concise and readily understandable. Another reason is that through alternate viewing of graphical output displays and entering of new inputs, either graphically or by other means, based on these output displays, a user can monitor and guide a computer during the course of a complex problem solution. Such interaction generally produces more rapid and often better problem solutions. In many cases, interaction allows a problem to be solved which would not even be attempted using other techniques.[1]

Most work to date on providing man-computer graphical communication facilities has been based on consoles which require full-time occupancy of, or, in the least, immediate access to a large digital computer.[2,3,4] Display maintenance has been provided by the large computer or by a data channel off the large computer. The high transfer rates needed for such an organization have dictated that the consoles be very close, if not physically adjacent, to the supporting computer or channel.

Significant research work is now concentrating on providing computing power to a large user community through a sharing of the facilities of a central computer.* The user community is distributed over a wide geographical area and communicates with the central facility using remote consoles. The pace of such work definitely seems to be accelerating, the reason being that early experience has shown that a time-shared central computer facility can serve many people very well.

The console facilities used to date to communicate with time-shared computer centers have been teletype or other typewriters. With such low-speed character oriented devices, slow access to the central computer (a few hundred milliseconds) is not readily apparent to a user. The real limiting factor in the use of such devices is the transmission delay, the time required to transmit messages between the computer and a user. For long messages with 10 character/second rates, this delay becomes both long and annoying, and strongly affects what can be done.

For a high-speed remote graphical display, the transmission rate can be increased so that the delay problem is reduced. Access time then becomes a

*Such work is going on at Bell Laboratories, M.I.T., G.E., Dartmouth, Carnegie Tech, U.C.L.A., I.B.M., S.D.C., RAND, and other places.

major problem. A few hundred milliseconds is certainly suitable access for certain aspects of a problem. To allow a user, however, to do real-time composing, editing, or other manipulation of graphical information with a light pen or other graphical input device, the sum of access time and transmission delay must be only a few milliseconds. For a truly remote console in a time-shared center, such timing will generally not be available. Therefore, to allow such light pen operations, a remote graphical console facility in a time-shared central computer environment should have local computing power.

Also if such high speed consoles are to be used remotely, local display maintenance is essential. Thus, a picture need be transmitted only once, maybe over low-cost low-speed lines. The central facility is then relieved of any further immediate responsibility. The console itself provides the broad band capabilities to maintain a picture at a flicker-free rate. Local computing power and display maintenance not only reduce the requirements on the central computer, but also should allow many interesting things to be done without even disturbing the central facility.

The GRAPHIC 1 console has been designed for use in exploring the problems associated with local computing power and display maintenance for a remote graphical console which eventually will be operating in a time-shared central computer environment. A photograph of the console is shown in Fig. 1.

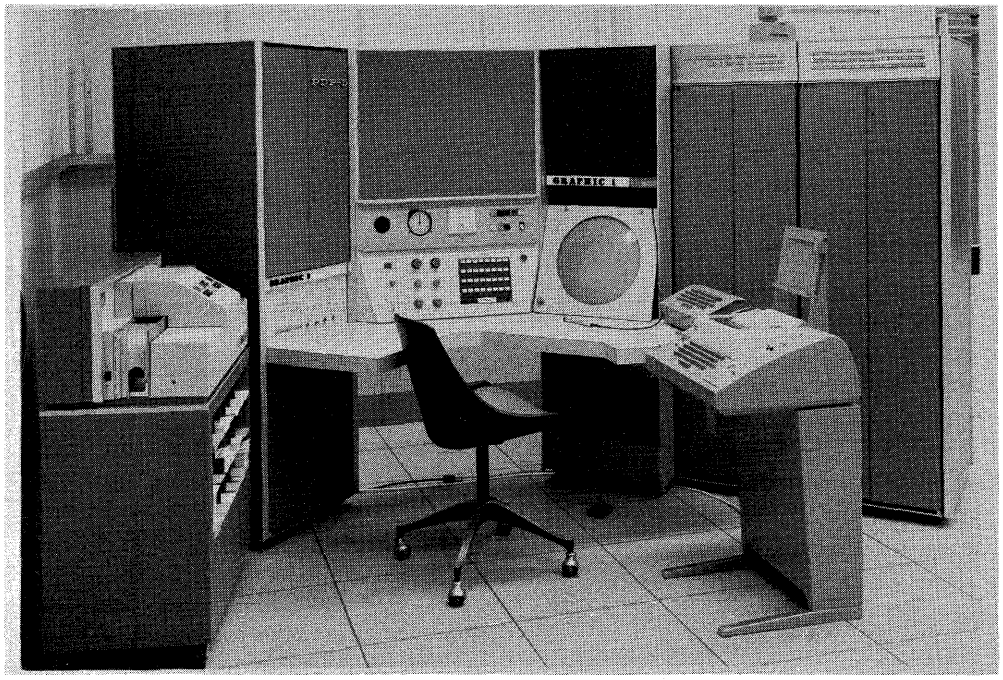


Figure 1. The GRAPHIC 1 console (photo no. B65-3727- MH).

## GRAPHIC 1 SYSTEM ORGANIZATION

The console system can be divided into two major units. The first unit consists of a control computer, which provides the local console computing power, and the console input devices. The second unit contains a display scope and an associated core buffer memory for storing display material. These last two elements and their connecting interface make possible a locally maintained output display. The two console units are connected by an interface which allows control signals and other information to be passed between them.

This interface also allows communication between the control computer and/or the display memory and the central computer facility, an IBM 7094 in the Murray Hill Computation Center. A block diagram of the described organization is shown in Fig. 2.

The console system is currently connected to the 7094 by a parallel transfer data connector. However, a request for service by the console system can

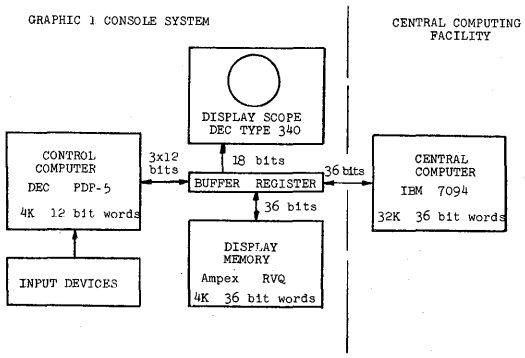



Figure 2. Organization of the GRAPHIC 1 console.

only be recognized at interjob points, i.e., between jobs on the normal input batch tape to the 7094. Response time is governed by the length of jobs on the batch tape, an average of from two to six minutes during daytime working hours.

This access is far from ideal and certainly will be improved when the Laboratories converts to a multiprogrammed central computer facility. The present waiting time does, however, simulate to some extent the access time and transmission delay that would take place if the console were connected to the central facility over a serial voice-band transmission line. It has strongly reenforced the necessity of having local computing power at the console.

The control computer in the console, a Digital Equipment Corporation PDP-5, is a small fixed-point core-memory digital machine.[5] The 6 microsecond cycle time memory contains 4096 12-bit words. The instruction set allows addition, subtraction, indexing, direct and subroutine branching, accumulator manipulation, and accumulator testing to be done internally. A microprogrammed input-output instruction allows extensive external manipulation and control operations to be performed. This instruction set can be used to compose, edit, translate, group, or otherwise manipulate graphical information in real time. However, the limited word size and the absence of hardware multiply-divide instructions prohibit computations in the PDP-5 itself involving a wide range of numbers or computations for rotations.

The display scope, a modified DEC Type 340 Precision Incremental Display, allows the rapid conversion of digital data to graphical form through point plotting on a 1024 × 1024 raster.[6] This plotting area occupies about a 9⅜ inch square on the scope face. Plotting is accomplished through use of six standard modes of operation. One of these is a control mode which allows the entering of parameter information such as scale factor, intensity, scope image reflection, and light pen enabling. The other five are display modes which allow the plotting of points, curves, vectors, axes, and characters. Internal vector and character generators produce the points needed to form lines and characters from the coded input words. A seventh non-standard mode for use in display linkage has been added. This will be described shortly. Of the display modes, only one, the point plot mode, is absolute. By absolute is meant that fixed X and Y absolute coordinates are assigned to a point. The remaining display modes (vectors, curve, axis, and character) are incremental in nature. Thus they essentially contain only differential or ΔX and ΔY movement information.

The plotting rate in one of the incremental modes is 1½ microseconds per point. For the X-Y or point plot mode, 35 microseconds are required per point. Since the characters average 20 points, average character plotting time is 30 microseconds. Considering these timing values and the scope grid size, about 20 scope diameters or 200 inches of line can be maintained on the scope at a flicker-free rate (30 frames/sec). This is equivalent to about 1000 characters.

The display material memory is an Ampex RVQ core unit with 4096 36-bit words and a 5 microsecond cycle time. This storage capacity represents about eight large pictures, i.e., those which take about 1/30-th of a second to display.

As was previously stated, the console system is divided into two major units: the control computer and input devices, and the scope and display memory. The former is the supervisory and input unit; the latter, the output unit.

The function of the output unit is the conversion of words in the display memory to an output picture on the display scope. These words can be a mixture of those composed in the 7094 and transferred to the console and those composed at the console using the input devices and the control computer. Successive words are supplied to the scope from the memory through operation of a rather complex interface. Since the scope accepts 18-bit words the interface accomplishes the gating necessary to load first the left half and then the right half of each 36-bit word from the memory into the scope. The interface also operates in conjunction with the scope and the display memory to execute display linkage instructions. These instructions, a direct transfer or

jump and a subroutine jump, operate essentially as will be described. There are slight additional complications which result from the left-right usage by the scope of the halves of a word from the display memory.

For the direct jump instruction, the next word is taken from the location specified by the jump instruction address. The first word of a subroutine is used to store return information. Therefore, for a subroutine jump, the next word is taken from the location one beyond that specified by the jump address. Before control actually passes on to that location, however, the location where control is to return upon completion of the subroutine is planted in the location specified by the subroutine jump. Exit from the subroutine is then accomplished using the planted return information. This method of display subroutine linkage allows multilevel display part subroutining. However, it does not allow reentrant subroutines.

Since the interface can provide successive words to the scope and the linkage words themselves can cause control to flow throughout the display memory, display maintenance is independent of control computer intervention. Thus, once started by the control computer, the display continually refreshes itself with a direct jump word at the end of the display picture providing the link back to the start. The only exception to this independence is that the control computer must restart the scope after display stoppages produced by edge overflows.

While the output unit is displaying a picture, the control computer can do other operations. Generally these are only looking for flags or interrupts indicating that an input device requires attention or that the central computer requests information. Once a flag or interrupt is found, the control computer can stop the output unit. The status of the scope can be saved. During the stoppage, the desired display composing, editing, or manipulation can then be done or information in the display memory can be read or written by the central computer. If desired, the display status can then be restored and the display cycling resumed or a new display can be started.

Cooperative communication programs in the control and central computers supervise any interchanges between the console and the central facility. The central computer can interrupt the console at any time. However, as was mentioned, the console can only be recognized between jobs on the central computer batch input tape.

In addition to the display linkage features, other additions made to the standard 340 Display involve the inclusion of reflection properties which can be enabled using the parameter setting mode, and addition of flagging bits to the parameter setting mode, jump modes, and point plot mode. The reflection properties allow display material to be reflected about a horizontal axis, or a vertical axis, or both. By changing the reflection setting before calling for a graphical subroutine, one block of data can be used to produce a figure in different orientations. There are no hardware rotation matrix features.

The flagging bits have been added to allow programmable display stoppage and computer signalling when certain conditions are found. This has been done since the material in the display memory is a mixture of actual display words plus list and buffer areas and aid is needed in tracing such material to make changes or additions. Use of programmable trapping means that interpretive programs do not have to operate upon the display memory to find desired locations or conditions. The scope itself does the tracing. These features are particularly valuable for use in conjunction with the light pen.

Additions have also been made to the single interrupt line of the PDP-5. This permits six different signals to be passed on to the interrupt line with the allowable interrupt signals at any time being controlled by a programmable masking register.

## INPUT DEVICES

The GRAPHIC 1 console is experimental as are the problems attempted using the system. Therefore, a wide variety of input devices has been included both to allow a user to pick which seems best for a particular function and to provide programming and human factors experience in the use of different devices.

The main input device for the console system is a DEC Type 370 Light Pen. The pen, which is a light sensing device, consists of a hand piece with mechanical shutter at one end of a fiber optics bundle. The other end of the bundle provides input to a photomultiplier system. The combined optic photomultiplier system is tuned to pick up the blue flash from the P7 phosphor of the scope face that is

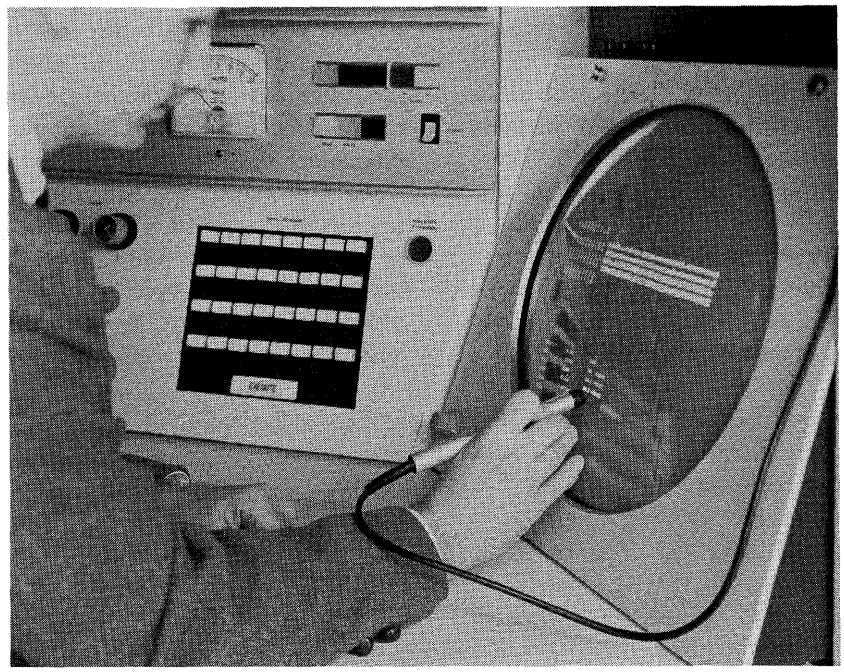


Figure 3. Package placement using GRAPHIC 1 (photo no. B65-4838-MH).

produced when a point is initially intensified. The pen is not sensitive to the yellow persistence. The field of view of the light pen is such that the tip can be held right on the scope face or up to three inches away without affecting proper operation. The photomultiplier sets a flag and stops the scope. Programmed operations in the control computer can then read the coordinate or address information needed for tracking or pointing before restarting the scope again.

A Teletype Model 33 ASR provides standard keyboard input-output functions. A Teletype Model 33 Self-Contained Keyboard has been added mainly for use in entering text where the display scope is used for feedback. The keyboard can be placed immediately in front of the scope for such occasions. For other occasions where the physical presence of the keyboard on the front counter interferes with light pen manipulations, it can be moved off to the side to the position shown in Fig. 1. Programs which use the self-contained keyboard echo any typed characters on the regular Teletype machine so that, in addition to the character image on the scope, a hard copy record is also available.

A track ball has been included for use in two-dimensional positioning problems. It consists of a 4 inch diameter nylon ball which drives two pickoff wheels, one for vertical travel and one for horizontal. No pickoff has been provided for rotation. The pickoff wheels turn potentiometers. Outputs from the potentiometers go through a multiplexor into an analog-to-digital converter in the PDP-5.

To reduce the directional polarization usually associated with track balls, the ball itself is not supported by the pickoff wheels. Instead, the ball rests on three ball end casters, the pickoff wheels being held against the track ball by spring tension. Mechanical lockouts are provided so that a pickoff wheel can be moved away from the ball if it is desired to leave one coordinate unchanged while the other is moved or to protect a setting from changes due to accidental bumping.

In addition to the two from the track ball, outputs from six potentiometers have been connected through multiplexor gating to the analog-to-digital converter in the PDP-5. Included are two ten-turn, two three-turn and two one-turn potentiometers. Through appropriate programming, an operator can use the knobs attached to the potentiometers to enter or change parameters in his problem.

Four switches have also been provided. Two are alternate-action and two are momentary. The alternate-action switches and one momentary are mounted on the corner panel for hand manipulation. The remaining momentary switch is located in a foot pedal. Testing instructions in the PDP-5 can determine the status of these switches.

A function keyboard consisting of 32 buttons connected by a mechanical lock and interlock is also located on the corner panel. The keyboard is wired so that the binary number of the currently depressed button can be read by the PDP-5. There is an additional EXECUTE bar located below the other keys.

The function keyboard can be used for vectoring into various special routines in the PDP-5. This is done as a two step process. First, the button corresponding to a desired function is depressed. When a user desires the function represented by the depressed button to be executed, the EXECUTE bar is pressed. This two step method of operation was chosen for accuracy in operation and for ease in repeating functions. To allow the vectoring operation to be accomplished, the output flag from the EXECUTE bar is connected to the interrupt facility of the PDP-5. The trap handler program can read in the depressed button number and enter a transfer vector table which causes a jump to the proper routine.

Since different users might desire different routines to be performed using the function keyboard, the changing of the functioning of a particular button can easily be accomplished by changing the transfer vector linkage and adding the routine to be linked. Changing of a label on a button can be accomplished by changing a labeling mask which fits over the keys. Despite this flexibility, the function keyboard has had little use. The reasons will be described in the next section.

A DEC Type 451A card reader allows the entering of information from cards into the console system. Reading speed is 200 cards/minute. A connector position for use in attaching any special or experimental devices has also been provided.

## EXPERIENCE WITH THE LIGHT PEN

There seems to be considerable debate over what is the best graphical input device — RAND tablet,[7] graphic pencil[4] or light pen. There are advantages and disadvantages associated with each device.

A RAND tablet is a stylus-tablet device which generates 10-bit x and 10-bit y coordinate information representing the stylus position on the tablet. Thus two-dimensional information can be obtained on a surface not coincident with the display device. A graphic pencil is a voltage-pickup position pencil. It operates in conjunction with an impressed voltage on a transparent conductive coating over the display screen to give position information. The advantage of these devices is that coordinate information can be entered at any time or place including on a blank area of a display or when a display is not even present. The disadvantage is that a positive reference between the stylus or pencil position and an object on the display scope does not exist. Thus a fairly complicated coordinate trace must be performed for demonstrative or pointing uses of these devices.

As was pointed out previously in the description of the console input devices, the light pen is only a sensing device. It must sense or *see* light from a display on the scope before it can function, i.e., set a flag and stop the display. Thus a disadvantage is that a display must be present before it can be used. Once the display is stopped and before it is restarted, however, considerable information can be extracted by the control computer. In addition to coordinate information, a positive reference to what is being plotted when the scope was stopped is available. Thus a coordinate trace does not have to be performed for demonstrative uses.

The light pen allows good feedback in pointing since programming can brighten what unit is being seen or display only the total unit being seen. The importance of such feedback cannot be appreciated until one works with a graphical system. Because of its particular value in pointing operations, a light pen has been chosen as the graphical input device for GRAPHIC 1.

To make the light pen as effective in drawing applications as the other devices, a good tracking program must support it. For the GRAPHIC 1, the tracking program uses a conventional four arm cross.[8] Tracking is on a 60 cps interrupt basis. Thus tracking performance is constant despite variations in the amount of display material. First order prediction is incorporated. Special testing allows the center of the cross to be moved right up to the edge of the display area on the scope with the corresponding arm shortening. Backed by this programming support, the light pen is equally suitable for both pointing and drawing.

In fact, the ease of use of the light pen for pointing operations has almost eliminated the originally envisioned extensive use of the function keyboard for entering various control routines. Instead, controls have almost exclusively been placed on "light

buttons" which are displayed on the scope face. A light button is a word or figure on the scope face which has a transfer vector associated with it. When the light button is touched by the light pen, the transfer vector is used to pass control to the appropriate routine.

Placing control functions on the scope face has two advantages. First, only those controls which should be present at a particular stage of a problem are displayed. If a light button labeled "MOVE" is one of those present, a user knows that he *can* move picture parts around. Similarly, if only light buttons in the form of PNP and NPN transistor symbols are displayed, a user knows he *must* select a particular type of transistor at that time. Thus, in effect, a user is steered through a problem. Second, during most operations there is only one center of attention, the scope face, on which a user need concentrate. This allows faster and smoother work on a problem.

Any shape, no matter how complicated, can be a light button since the shape has no effect on detection by a light pen. This is extremely important since the shapes can be problem oriented. The complicated zone bounding necessary when performing a similar function using a RAND tablet or graphic pencil need not be done. Light buttons can easily be moved around on the scope face. Also, the transfer vector for one light button can easily be transferred to another shape. This mobility and flexibility combine with the previously mentioned advantages to make light pen detected light buttons very powerful control elements.

## BASIC PROGRAMMING SUPPORT

Software support for the GRAPHIC 1 console system consists of three basic units. First there is an assembler and assembly postprocessor which generates, merges, and links relocatable programs for the PDP-5. This assembler has been written using MACRO-FAP and runs on the 7094. Therefore, all the features of MACRO-FAP are available for use when assembling programs. This is particularly useful when generating higher level languages. The combined assembler-postprocessor aids considerably in dealing with the page boundary problems of the PDP-5. The second basic unit is a display material assembler and postprocessor. This also is written in MACRO-FAP and runs on the 7094. Symbolic linkage between PDP-5 code and display code can be accomplished. The third major software unit consists of communications routines to allow the easy passing of information between the 7094 and the GRAPHIC 1 console.

Using these three basic units, several higher level graphical languages are being developed. Furthest along of these is the GRIN (GRaphical INput) language. GRIN is particularly suitable for use in problems requiring the extensive real-time manipulation of graphical information at the console. It takes full advantage of the incremental display structure of the scope. (Recall that there is only one absolute plotting mode. The remainder are incremental.) Thus, if a display part is composed only of a sequence of incremental words, its position on the display scope can easily be changed by changing only the initial absolute entry point. Also if a part is represented only incrementally, it can be called up using the display part subroutine linkage at many places on the scope face. The part has to exist in storage only once, however.

No hard copy facility is available directly in the console system. A SC 4020 microfilm recorder is attached to the 7094. So, hard copy of the current display picture is achieved by transferring an image of the display buffer contents to the 7094 where translation programs map the scope code into comparable SC 4020 code to produce a microfilm print. Turnaround time to receive a print is from two to four hours.

## USAGE

The major use of the console system so far has been for problems requiring at some stage "dynamic scratchpad" capabilities. Included in such problems have been printed circuit component and wiring placement, schematic circuit design, block or flow diagram design, text composing and editing, and placement of cards on a chassis to achieve minimum connecting wire length. Initial input or configurations to be manipulated can be provided by programs in the IBM 7094. For example, a card placement programs to minimize connecting wiring length gives a good initial placement. However, such placements are subject to local minima and frequently neglect maintenance considerations. An operator at the console, when provided with appropriate displays, can break out of local minima for reruns in the 7094 and can make placements for

easy maintenance. The photograph in Fig. 3 shows a user working on such a placement problem.

The result of any console manipulations can serve as input to a 7094 program also. As an example, text in the form of a computer program which has been composed at the console can be entered, assembled, and run. A system block diagram composed at the console can be used as input to a special compiler, the compiled program can be run, and the results viewed at the console. As a further example, a schematic circuit composed at the console can be used as input to a circuit analysis program and the results of the program returned to the console. The number of interesting uses of the console continues to grow as the existence of the console becomes known to people in varied fields of work.

## SUMMARY

GRAPHIC 1 provides very flexible man-computer graphical communication facilities in a time-shared central computer environment. Local computing power and display maintenance capabilities make possible extensive real-time graphical manipulations at the console. Interaction between the console and the central computer permits attacks on large complex problems.

## ACKNOWLEDGMENTS

During the period that this system was designed and constructed many people have contributed useful ideas. Chief among these contributors were H. S. McDonald, A. D. Hause, C. Christensen, and A. G. Faulkner. The author wishes to express his appreciation to all these people.

## REFERENCES

1. G. J. Culler and R. W. Huff, "Solution of Non-Linear Integral Equations Using On-Line Computer Control," *S.J.C.C.*, Vol. 21, 1962.

2. I. E. Sutherland, "Sketchpad, A Man-Machine Graphical Communication System," *S.J.C.C.*, Vol. 23, Spartan Books, Inc., Washington, D.C., 1963.

3. T. E. Johnson, "Sketchpad III, A Computer Program for Drawing in Three Dimensions," *S.J.C.C.*, Vol. 23, Spartan Books, Inc., Washington, D. C., 1963.

4. B. Hargreaves, J. D. Joyce, G. L. Cole, et al, "Image Processing Hardware for a Man-Machine Graphical Communication System," *F.J.C.C.*, Vol. 26, Spartan Books, Inc., Washington, D. C., 1964.

5. *"PDP-5 Handbook,"* Digital Equipment Corp., Maynard, Mass. (1963).

6. *"Type 340 Precision Incremental CRT Display,"* Digital Equipment Corp., Maynard, Mass. (1965).

7. M. R. Davis and T. O. Ellis, "The RAND Tablet: A Man-Machine Graphical Communication Device," *F.J.C.C.*, Vol. 26, Spartan Books, Inc., Washington, D. C., 1964.

8. R. Stotz, "Man-Machine Console Facilities for Computer-Aided Design," *S.J.C.C.*, Vol. 23, Spartan Books, Inc., Washington, D.C., 1963.

# THE BEAM PEN: A NOVEL HIGH-SPEED, INPUT/OUTPUT DEVICE FOR CATHODE-RAY-TUBE DISPLAY SYSTEMS*

Donald R. Haring
*Electronic Systems Laboratory*
*Massachusetts Institute of Technology*
*Cambridge, Massachusetts*

## INTRODUCTION

In recent years there has been considerable interest in providing rapid communications between a man and a computer. A most useful communication system in this regard is a cathode ray tube (CRT) computer display and a hand-held "light pen"[1] to report to the computer whenever a lighted spot on the CRT display falls within its small field of view. The pen may be used to identify or select a particular displayed item which the man wishes to call to the attention of the computer. Also, a feedback loop may be programmed to keep a displayed pattern of spots centered in the pen's field of view. With such a loop the displayed pattern follows the moving pen, and the system becomes a very versatile graphical input device. This process is known as "pen tracking."

A typical application of the light pen and CRT as a graphical input/output system has been discussed by Ward and Ross[2] for "Computer-Aided Design" in general, and by Sutherland[3] for his "Sketchpad" system which is used to communicate between man and computer by sketches. In the Sketchpad system, the user sketches such things as machined parts and electronic circuit diagrams on a computer CRT display with the light pen. Specifically, the light pen is used to draw displayed lines on the CRT by using pen tracking, to position predefined parts of the sketch on the display and to point to such parts in order to change them. A set of push buttons operate in conjunction with the pen to indicate changes such as erasing or moving. Because of this hardware-software package, no typed statements are necessary to the computer (except for legends), hence greatly increasing the speed of man-machine communication. For additional discussions of light-pen applications see references 1-5.

Light pens detect the light produced on the CRT screen by a pulse of electron-beam current. The electron beam is under computer control. The light-pen output for a given pulse is determined by the combined time responses of the light-producing mechanism in the phosphor, the light-detecting mechanism in the light pen, and the pen amplifier. Each of these responses is characterized by a delay in buildup and a delay in decay, with the result that

*This work was made possible through the sponsorship extended the M.I.T. Electronic Systems Laboratory 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 pen output corresponding to a single narrow square pulse of beam current is a delayed and much broadened pulse with a long tail. The net pen output for a given intensification pulse is the sum of the response for that pulse and any residual tails from previous pulses. Thus, there is an integrating effect and it is necessary that sufficient time be allowed between pulses for almost complete decay if a reliable solution among points is to be maintained. In other words, the system has a limited bandwidth.

In the past, typical CRT display systems used with light pens plotted points at intervals of about 20-30 microseconds, and the combined decay time of the pen response was such that these points could be resolved in time. However, because of the desire to display more complicated pictures flicker-free with modern computer display systems, it is not unusual to display points every 1-2 microseconds, and it is found that it is impossible to distinguish among points displayed this closely in time, even with the fastest available light detectors—photomultiplier tubes. The present limit appears to be about 8 microseconds.

The major limiting factor in further improvement in light-pen speed is in the phosphor screen itself, which must be of a persistent type in order to reduce display flicker. Fortunately, all persistent phosphors are of the two-layer type, with a relatively fast phosphor layer which responds directly to the electron beam, and a slow phosphor layer which does not respond to the electron beam but is excited by the light from the fast layer. The widely used P7 phosphor, for example, has decay times of 40 microseconds and 100 milliseconds for the fast and slow phosphors, respectively. Since the light output of the fast layer is much greater than that of the slow layer, the fast layer essentially determines the pen response.

The development of a new phosphor with a sufficiently fast exciting (fast) phosphor to improve the light-pen response speed would entail considerable cost and does not appear to be the best long-term solution. Instead, we have taken a new approach to the problem. We have developed a system to detect the electron beam causing the screen light rather than the light itself. This system is called the *beam-pen system*. This new approach to increasing the speed of the CRT display system for man-machine communication is described in this paper.

## BEAM-PEN SYSTEMS: BASIC THEORY

Reference to Fig. 1 provides a qualitative understanding of the operation of a beam-pen system. The CRT circuitry selects the area of the CRT screen to be illuminated and, in the typical computer display, turns the electron beam on for a prescribed period of time after the desired screen area has been reached, i.e., the selected deflection signals have stabilized. The *beam pen* is a conducting probe that is hand-held in front of and very near the CRT screen. The conducting probe is connected to a high-input-impedance "pen amplifier." When the pen is placed in front of the CRT screen, the probe is capacitively coupled to the beam such that as the distance (defined in Fig. 1) is decreased, the beam-to-probe capacitance, and thus the signal detected by the probe, increases. One can use either the analog distance-varying output of the pen amplifier or add a threshold element to produce a digi-

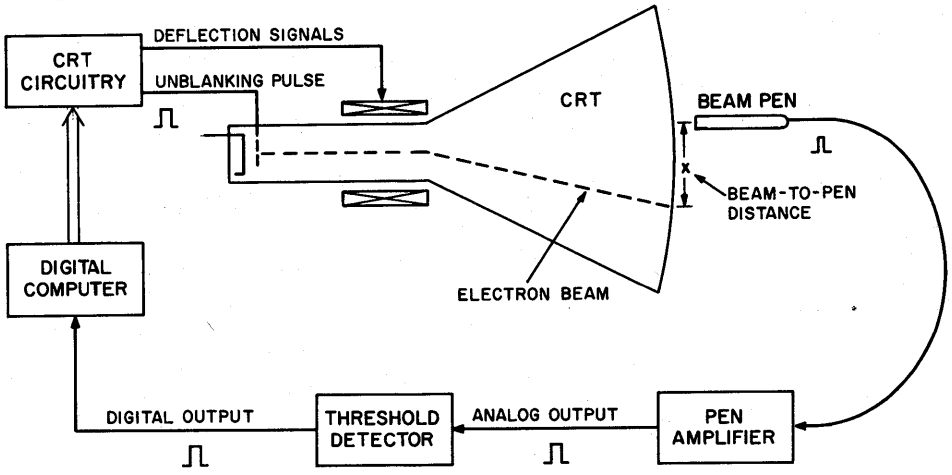



Figure 1. Simplified beam-pen system.

tal "seen-not-seen" output. We see then, that the beam-pen system is similar to a radar system.

Let us now take a closer look at the system to obtain a quantitative understanding of its operation. Figure 2 shows a simplified circuit model of the beam-to-pen coupling. The CRT screen can be represented by a parallel RC circuit. In non-aluminized tubes this time constant is in the order of 0.1 seconds.[6] Hence, since we are concerned with beam pulses in the order of microseconds, the screen acts as an integrator. Similarly, the pen amplifier input can be represented as a parallel RC circuit. This impedance is a parameter of the design. Finally, the coupling between the beam and pen is clearly primarily capacitive.

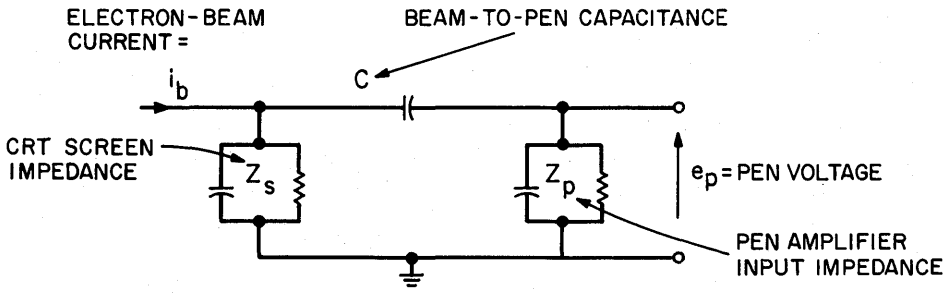



Figure 2. Simplified circuit model of beam-to-pen coupling.

The geometry of the CRT screen controlling the beam-to-pen capacitance is illustrated in Fig. 3. Here the electron beam is represented by a stream of electrons impinging upon the CRT phosphor. Upon striking the phosphor, the incident electrons create some secondary electrons which leave the struck area of the phosphor, but since the phosphor is operated at secondary emission ratios of less than one, the net charge on the screen at the struck area is negative. The beam is represented by a conducting probe surrounded by a shield and is shown displaced a distance $x$ along the safety plastic from the center of the electron beam. Typically, the diameter of the electron beam ($d$) is in the order of 5 to 10 mils, the distance ($r_0$) from where the beam strikes the phosphor to the surface of the safety plastic is in the order of ½ inch, and the probes used are approximately 1/10 inch in diameter. From

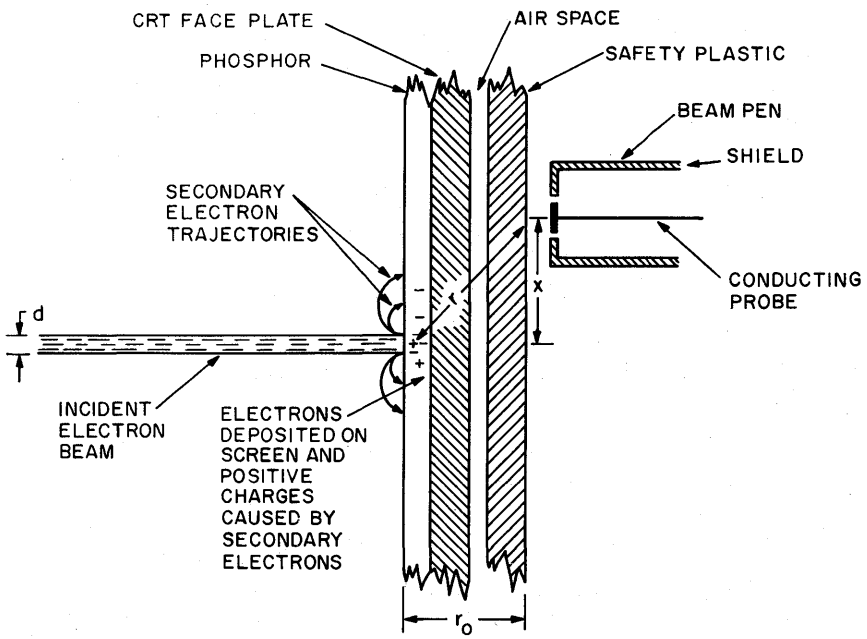



Figure 3. Cathode ray tube screen details.

these values it is clear that even the maximum value of the beam-to-pen coupling capacitance $C$ is extremely small. For example, using a parallel-plate capacitor model shows that this capacitance is less than 1/10 of a picofarad. Hence, the input impedance of the pen amplifier must be high to obtain a reasonable signal from the beam.

From the circuit model in Fig. 2 and the above observations, the pen voltage $e_p$ is approximately equal to $C(j\omega Z_s Z_p)i_b$. Furthermore since $Z_s$ has such a large time constant, i.e., $Z_s \approx 1/j\omega$, to a close approximation, $e_p \approx kC Z_p i_b$, where $k$ is a constant. Hence as we observed previously, the pen voltage is directly proportional to the coupling capacitance and the pen amplifier input impedance.

Because the distance $r$ in Fig. 3 is so much larger than the areas of the beam and pen, a parallel-plate capacitor model for the beam-to-pen capacity is poor. Unfortunately, a more exact capacitor model results in very difficult field equations. Hence, we have used experimental data to derive the following polynomial approximation to the variation of capacitance with pen displacement along the safety plastic when $x \leq 4\,r_0$, where $x$ and $r_0$ are defined in Fig. 3:

$$C = C_{\max}\left[\frac{1.6}{\sqrt{1+\alpha^2}}\right] - \frac{0.46}{1+\alpha^2} - \frac{0.14}{1+\alpha^4} \tag{1}$$

where $\alpha = \frac{x}{r_0}$. We are not interested in distances $x > 4\,r_0$. Clearly, the pen voltage is the same function of $x$. As a matter of fact, the data used to derive (1) was obtained by measuring the pen voltage in a system with the following properties:

CRT = 16ADP7 (non-aluminized)
Beam diameter = 0.005 inch
$r_0$ = 0.5 inch
Probe diameter = 0.085 inch
Shield diameter = 0.5 inch

Variations in these parameters influence the coefficients in (1). For a discussion of these effects see reference 7.

Figure 4 is a normalized plot of (1). The shape of this response curve is of interest in resolution considerations. In other words, one important property of a man-machine communication system is to designate specific items in a CRT display using a digital seen-not-seen signal. The higher the resolution (i.e., the smaller the field of view) of the beam pen, the smaller the distance between two different items that can be independently designated. The digital signal from a beam pen is derived from a threshold detector attached to the analog output. Because a threshold detector has a finite "dead band" for which its decision is not predictable, to insure a stable resolution the pen analog output should change as rapidly as possible with a change in displacement, and be as noise-free as possible at the value of the output to which the threshold detector is adjusted.

Since $r_0$ has a marked effect on the shape of the beam pen response as shown in Fig. 4, it is important to operate the pen as close to the CRT screen as possible. The minimum value of $r_0$ is determined by geometry that, for the most part, is not under control of the designer. Hence, this distance is a major limitation of the beam-pen system.

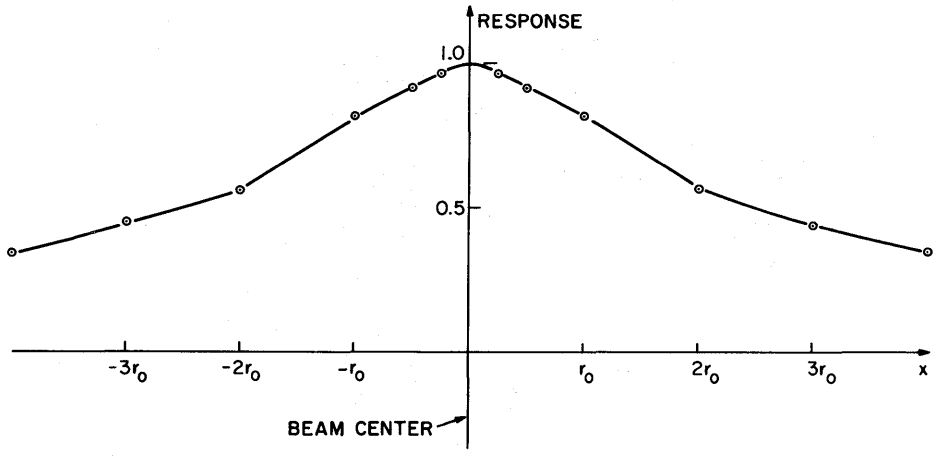



Figure 4. Normalized plot of Eq. (1)—$r_0 = \frac{1}{2}$ inch.

There are several noise sources in the beam-pen system. The primary sources are: the electron beam, the CRT phosphor, the CRT high voltage supply, the computer, the display system, the pen amplifier. Since signals in the order of tens of microvolts have been measured at the screen, the noise contributions of a well-designed pen amplifier are small compared to this signal. Because the noise in the CRT high voltage supply, the computer and the display system are principally centered about d.c., the contributions from these noise sources can be greatly reduced by modulating the unblanking pulse with a high-frequency signal and using an appropriate high-frequency band-pass pen amplifier followed by a detector. That is, in Fig. 5 we show the usual computer-generated unblanking pulse. This pulse is modulated by a high-frequency sine wave, also shown in Fig. 5 as the resulting beam charge. Figure 5 also shows the corresponding phosphor charge, viz, approximately the integral of the beam charge.

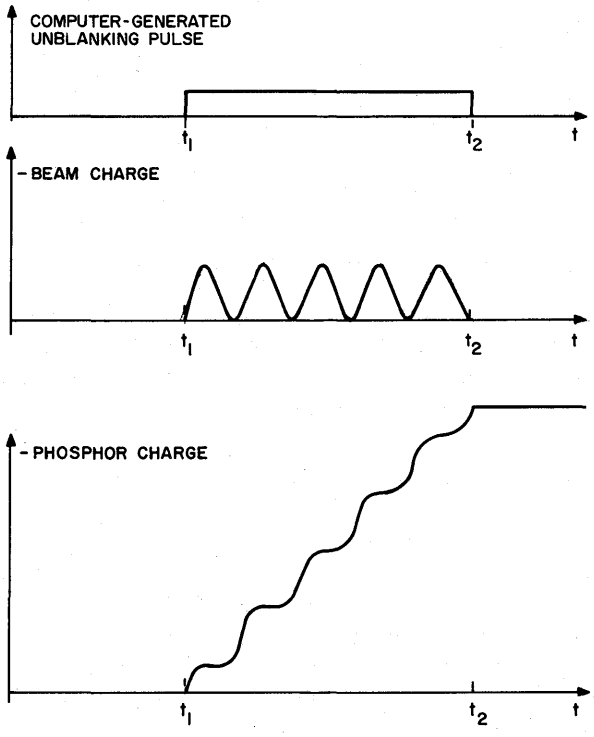



Figure 5. Charge variations.

Using a modulation system, the remaining primary noise sources are the electron beam and the CRT phosphor. There is reason to believe that both of these sources produce both wanted signal and noise. Hence both can be viewed as noise in a communication or radar channel. Thus, methods used in communications and radar can be used to increase the signal to noise ratio. As yet, little has been done in this direction. One approach presently being studied experimentally is to use a synchronous detector rather than an envelope detector.

The beam pen operates because of local charge variations on the CRT phosphor. Any modification to the phosphor that eliminates or modifies these charge variations will render the beam pen inoperative. Hence, a beam pen will not operate with an aluminized-screen CRT because the aluminization greatly reduces the resistivity of the phosphor, thus essentially eliminating local charge variations. When a CRT screen is not aluminized, placing foreign objects at the outside surface of the screen can modify the charge distribution. A way to eliminate this effect is to cover the outside of the CRT or the safety plastic with a static reducing agent, such as Statnul, manufactured by Daystrom, Inc.

## THE ESL BEAM-PEN SYSTEM

This section decribes the beam-pen system built at the Massachusetts Institute of Technology, Electronic Systems Laboratory (ESL) for use with a display console on the M.I.T. multi-access computer (Project MAC). This display is an incremental type in which output pictures are composed of discrete dots displayed in the following manner: The CRT beam is turned on for a duration of 0.5 microseconds to display a point, turned off, relocated, and turned on for the next point. The beam never moves while turned on. The 0.5 microsecond duration that the beam is turned on is hereafter referred to as the *display time* of the unit. Points are displayed at a maximum rate of one every 1.5 microseconds. The display is manufactured by *Digital Equipment Corporation* (DEC) and is similar to their Type 330. A Type 16ADP7 non-aluminized CRT is used.

A block diagram of the ESL beam-pen system is shown in Fig. 6. A frequency of 10 megacycles was chosen as the modulation frequency for this unit because of the availability of a commercal amplifier at this frequency. The lower frequency limit for such a system is imposed by the system bandwidth requirement which in turn is dictated by the display time. The upper frequency limit for such a system is imposed by stray capacitances and radiation.

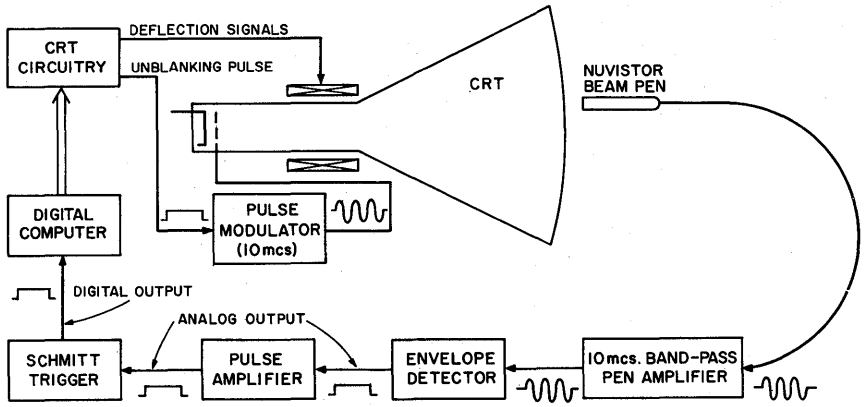



Figure 6. The ESL beam-pen system.

As seen in Fig. 6, the ESL beam-pen system consists of the following units:

1. A pulse modulator to intensity modulate (approximately 100 percent) the CRT beam with a 10-megacycle signal.
2. The beam pen itself, a hand-held, high-input-impedance pickup consisting of a shielded conductive probe and a preamplifier to pick up the electron-beam signal.
3. A broad band-pass 10-megacycle amplifier to amplify the beam-pen output.
4. An envelope detector to remove the 10-megacycle carrier.
5. A pulse amplifier to drive the threshold detector and produce signals compatible with DEC signals.
6. A threshold detector to produce a digital output.

Each unit is now briefly described. First the pulse modulator. In the present system, this modulator is simply a high-Q parallel resonant circuit having a normally turned-on switching transistor connected in series with a d.c. power source. When the computer generates an unblanking pulse, this switching transistor is turned off, rapidly changing the current through the high-Q circuit and resulting in a 10-megacycle "ringing" for the duration of the unblanking pulse. This signal is coupled to the CRT first grid through an emitter follower and the amplitude is adjusted to cause about 100 percent modulation. (For the 16ADP7 this is approximately 15 volts peak to peak.) Hence there is a 10-megacycle signal present only when the beam is turned on.

A photograph of the present beam pen is shown in Fig. 7. To produce a stable high input impedance a *Nuvistor* cathode follower is used. The probe characteristics are those specified in the previous section to derive Eq. (1). Other pertinent characteristics are: input capacitance = 7.25 picofarads, input resistance = 2.0 megohms, and voltage gain with 50-ohm load = 0.25.

The band-pass amplifier is a Type EV 1002 manufactured by RHG Electronic Laboratory, Inc. Pertinent characteristics are: center frequency = 10 megacycles, 3 db bandwidth = 2.1 megacycles, noise figure = 2.1 db, maximum power gain = 97 db, maximum voltage gain = 106 db, and maximum voltage output = 40 volts. The amplifier contains an envelope detector and a linear pulse amplifier. The latter drives a simple external pulse amplifier to make the signal levels compatible with DEC signal levels and to provide a threshold adjustment. The RHG amplifier also has a manual gain control for the 10-megacycle section and an AGC for this section. The manual gain control is adjusted to prevent overload at any time, and the AGC is disabled to prevent integrating effects in the AGC circuit.

The threshold detector is simply a DEC Schmitt trigger and it drives a flip-flop in the display system. This flip-flop is under program control. For additional details of this system see reference 7.

## PERFORMANCE OF A BEAM-PEN SYSTEM

Figure 8 shows the analog response of a DEC *Type 370* fiber-optic, photomultiplier-tube light-pen system to a sequence of displayed points occur-

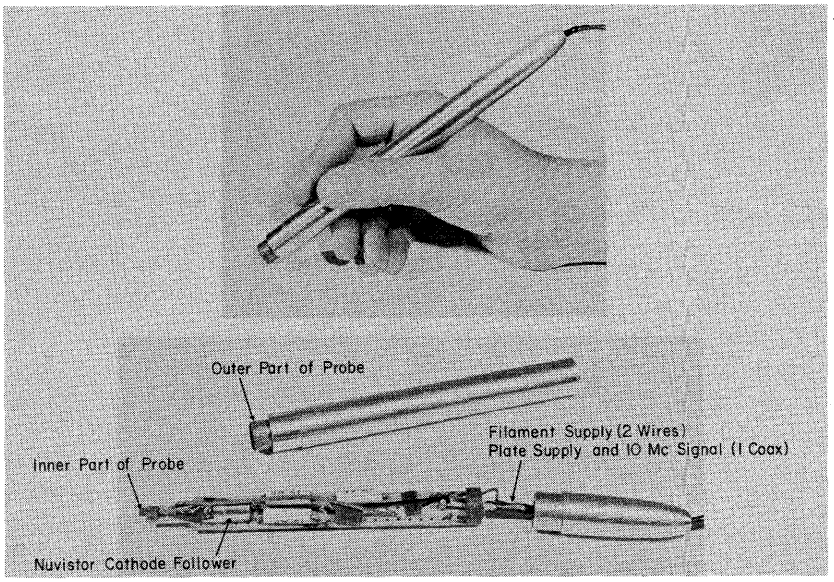



Figure 7. The beam pen.

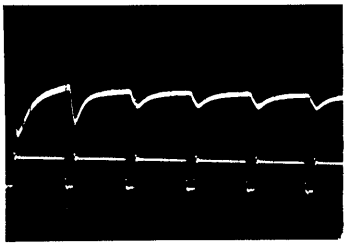



Figure 8. Light-pen response.

ring at a display rate of 1 point every 3.5 microseconds. This light pen is presently the fastest commercial light pen available. The first point displayed is on the left-hand side of the picture. As can be seen, the pen response to this first point is greater than that to the second point, which is in turn greater than that to the third point, etc. Although not shown in this figure, after a sufficient number of points essentially no response is obtained. This figure clearly demonstrates the deficiency of a light-pen system for high-speed operation—the inability to discriminate between points occurring at high display rates. For reference, the Type 370 light pen can operate reliably, by proper signal processing, at a display rate of one point every 8 microseconds or more.

Figure 9 shows the analog response of the ESL beam-pen system described in the previous section. The display rate is identical for this photograph as for the one in Fig. 8. The superior performance of the beam-pen system as regards the speed of response is clearly evident. For reference, this beam-pen system has been tested at a display rate of 1 point every 1.5 microseconds and it still was not speed-limited. Of course, it operated equally well at slower display rates.

On the other hand, the light pen has a better resolution than the beam pen. The particular ESL beam-pen system described in the previous section

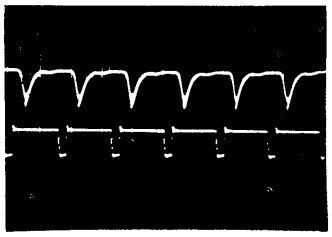



Figure 9. Beam-pen response.

has a reliable field of view of 1.5 inches in diameter. This is approximately a factor of 10 larger than that of the DEC 370 pen. The primary cause of the relatively poorer resolution of the beam pen is that the phenomenon by which it works is a slower varying function of distance from the displayed point than is the light by which the light pen works. This is evident from the plot of (1) in Fig. 4 and the realization that optical signal processing can be used on the light pen. A secondary cause of this poorer resolution is that the noise level in the present system is higher than it should be. This latter factor is presently being studied.

The poorer resolution of the beam pen does not markedly affect its performance in respect to the light pen when "pen tracking" is being used. The poorer resolution does show up of course when one desires to point to one of several objects that are close together.

Another factor to consider in the beam-pen system is the delay between the occurrence of the electron beam and the output of the system. This is caused by the band-pass pen amplifier. From Fig. 9 we see that the delay is in the order of 1 microsecond. For many applications (e.g., the ESL Display Console) this delay is no problem. If the delay cannot be tolerated, then a wider band-pass amplifier would reduce this delay with some loss in signal-to-noise ratio.

For more details of the performance of the ESL Beam-Pen System and other considerations, see reference 7.

## CONCLUSIONS

From experimental evidence we conclude that it is possible to make a system to detect when and where the electron beam of a CRT strikes the screen, thus essentially eliminating the bandwidth-limiting effects of the CRT phosphor and making a high-speed man-machine communications system possible. The system that is described here is presently being used in an operating environment. It is superior to the light-pen system as regards to speed, and is relatively simple, inexpensive, and insensitive to the background light of the operating room. There are also several possible extensions of the system. For example, by modulating the electron beam with several differential selectable frequencies, one could "color" or "classify" various displayed points on the CRT screen. The present disadvantage of poorer resolution than the light pen does not appear to be insurmountable and is presently being studied.

## ACKNOWLEDGMENTS

Thanks are due to Mr. J. E. Ward, Assistant Director of the M.I.T. Electronic Systems Laboratory and Mr. R. H. Stotz, also from the Laboratory, for their continued interest in this work, and to Mr. P. G. Arnold, formerly from the Laboratory, for his contributions to the development of the hardware.

*This work was made possible through the sponsorship extended the M.I.T. Electronic Systems Laboratory 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).

## REFERENCES

1. B. M. Gurley and C. E. Woodward, "Light-Pen Links Computer to Operator," *Electronics*, Nov. 20, 1959, pp. 85-87.

2. J. E. Ward and D. T. Ross, "Investigations in Computer-Aided Design," M.I.T. Electronic Systems Report 8436-IR-1, chap. 8 and 9 (May 30, 1960).

3. I. E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," M.I.T. Lincoln Laboratory Technical Report 296 (Jan. 30, 1963).

4. T. E. Johnson, "Sketchpad III: Three Dimensional Graphical Communication with a Digital Computer," M.I.T. Electronic Systems Laboratory Report ESL-TM-173 (May 1963).

5. J. C. R. Licklider and W. E. Clark, "On-Line Man-Computer Communication," *Proc. AFIPS Spring Joint Computer Conference,* vol. 21, pp. 113-135 (1962).

6. Williams and Kilburn, "A Storage System for Use With Binary Digital Computers," *Proc. IEE,* vol. 96, pt. 2, no. 81, pp. 183-200 (Mar. 1949).

7. D. R. Haring and P. G. Arnold, "The Beam Pen: A New Approach to High-Speed Light Pens," M.I.T. ESL Report (in preparation).

# VOICE OUTPUT FROM IBM SYSTEM/360

A. B. Urquhart
*Systems Development Division*
*International Business Machines Corporation*
*Kingston, New York*

## INTRODUCTION

The IBM 7770 and 7772 and elements of the New York Stock Exchange Market Data System form a family of IBM devices providing voice output facilities. The devices function similarly in that each gives a computer-generated voice response to a dialed inquiry. The audio generation principle incorporated in the 7770 is a derivative of the original Voice Answer Back principle—that of adjusting word length to fit machine time-slots. Then by access to these words from many input lines, sentences are formed into a voice response. The 7772, on the other hand, generates audio on the "vocoder" principle—that of energizing tone filters and combining the output result first to form words and then sentences in a manner similar to that of the 7770.

Input to the New York Stock Exchange Market Data System was accomplished through the use of the IBM 7750 program transmission control. In contrast, the 7770 and 7772 are self-contained input/output devices designed for widely diversified applications requiring various types and lengths of inquiries and responses in such industries as banking, insurance, manufacturing, and retailing.

They are, therefore, modular in increments of numbers of "lines" and "words." A "line" is defined as a half-duplex communication channel to which more than one telephone may be connected, but where only one transaction takes place at one time. In the case of the 7770 and 7772, a line is used for transmission of digital information in the input direction and voice in the output direction. A "word" is a unit of vocabulary which, for speech-processing purposes, may be either in analog or digital form.

There are differences in the number of lines and words available on each device and in the method of generating voice output. There are three models of the 7770 and one of the 7772. The difference among the three models of the 7770 is in methods of attachment to host processors. Table 1 shows how many lines and words are available on each and the type of processors to which each device can be attached.

Both units are available in languages other than English, but because of the many variations in vocabulary and methods of attachment to communication equipment on a worldwide basis, this discussion is limited to that of the 7770 Model 3 and 7772 as attachable to IBM System/360 within the continental United States. The external appearance of the two machines is similar because the same

Table 1.

| | No. of Lines | No. of Words | Processor |
|---|---|---|---|
| 7770-1 | 4 to 48 (in increments of 4) | 32 to 126 (in increments of 16) | IBM 1401, 1140, 1460 |
| 7770-2 | 4 to 48 (in increments of 4) | 32 to 127 (in increments of 16) | IBM 1410, 7010 |
| 7770-3 | 4 to 48 (in increments of 4) | 32 to 128 (in increments of 16) | IBM System/360 Mod. 30, 40, or 50 |
| 7772 | 2 to 8 (in increments of 2) | Any amount from available vocabulary list. Limited by available storage. | IBM System/360 Mod. 30, 40, or 50 |

frame with similar means of connection to communication facilities and host processors is used for both. (See Fig. 1.) In function, the 7770 and 7772 are the same; the difference between the two is in the method of generating the voice output as will be described later in this article.

There are three basic sections to each device—input, output, and control. The input and output sections connect to the common-carrier communications network. The control section connects the input and output sections to the host processor. A transaction takes place in the following way.

Using the telephone, an inquirer first dials the telephone number allotteed to a 7770 or 7772. When the ringing stops, a tone will be heard, indicating that the call has been answered. The inquirer now dials his input message. The 7770 or 7772 forwards this message, a character at a time, to the attached computer which processes the input data and returns a digital output message. The message is converted to audio and heard by the inquirer.

## METHODS OF ATTACHING COMMUNICATIONS EQUIPMENT

The design of the communications interface required consideration of the various uses of a telephone in a machine environment. The telephone, in this case, was the prime input and output device involving dialed digits as input and audio as output.

Some of the first design problems affecting the telephone as an inquiry terminal involved consideration of the human element. How does the inquirer react to a telephone that he knows is connected to a machine? If the inquirer receives no reply, does he hang up? If so, right away? In 20 seconds? In 2

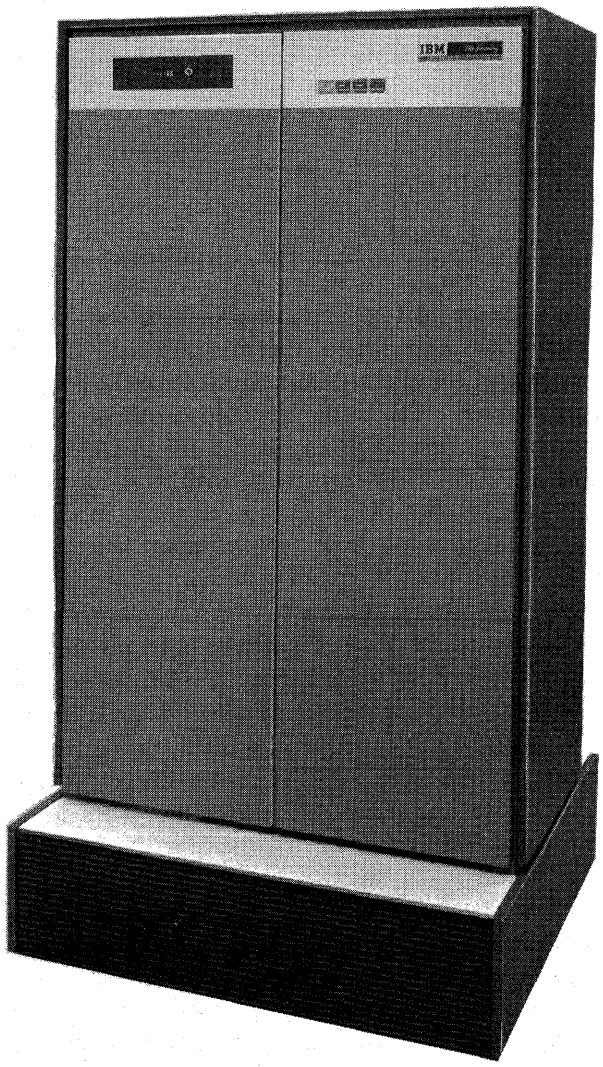


Figure 1. An audio response unit in background.

minutes? It seeemed that most human-factor problems fell in the category of "normal telephone practice" and that the real problem was not what the inquirer would do but how the 7770 or 7772 would react to questions like these:

- What does it do if the inquirer misdials?
- How will it recognize end-of-inquiry?
- What codes will be presented to it?
- How much error-checking can reasonably be done?
- Should it accept d-c dial pulses or tones?
- Should it accept serial and/or parallel data?

In adition, there were questions concerning Audio Output over the communications interface.

- Could any data sets be modified to tranmit audio?
- Could this be done on balanced lines? On unbalanced lines?
- What should be the level at which audio is transmitted?
- What happens if an inquirer dials-in while audio is being transmitted?

The answer to most of these questions lay in designing the 7770/7772 "front end" to fit a parallel communications data set interface; namely, that of the AT&T 400 series data sets or their equivalent. These data sets are serial-by-character, parallel-by-bit data transmitters and receivers which are capable of handling numeric and/or alphanumeric data. In addition, one receiver type has since been modified to allow audio transmission in the output direction. One data set receiver (equivalent to the Western Electric 403A or 401J3) must be used per 7770 or 7772 line as shown in Figs. 2 and 3 respectively.

Common to all of the connections in Fig. 2 is the use of pushbuttons for inquiry. Entering digits in this manner is faster, more reliable and is gaining in popularity, but there still are two basic types of telephones that can be used—the rotary dial telephone and the pushbutton telephone. If a rotary dial telephone is used, a pushbutton attachment is usually added. An inquirer would first dial the system number with the rotary dial telephone and enter the inquiry with the pushbutton attachment. If the inquirer uses a pushbutton telephone as shown at Fig. 2D, he dials the number and enters the inquiry using the same telephone.

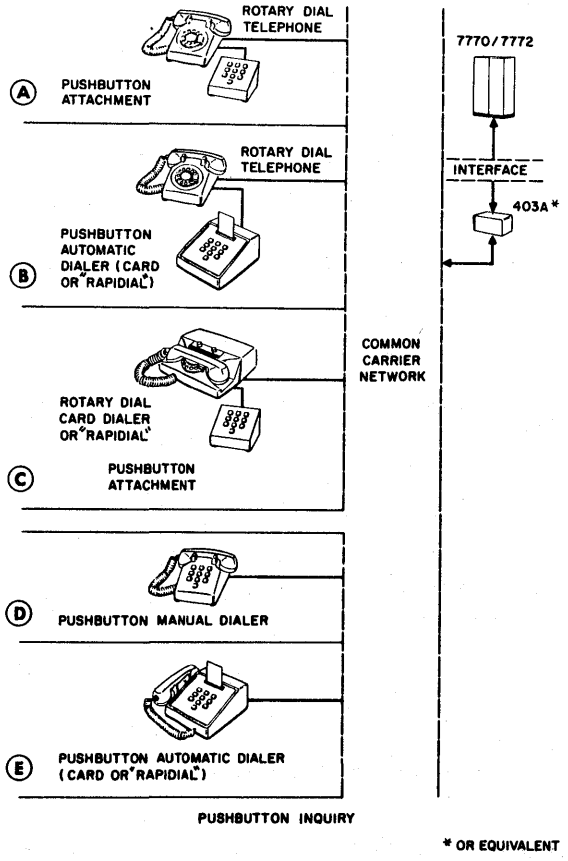



Figure 2. Telephone equipment appropriate for inquiry.

Other inquiry terminals shown in Fig. 2 are variations of either the basic rotary dial or the pushbutton type of telephone. These are the card or RAPIDIAL* type of telephones. In the card-dial telephone, a number that is frequently used can be punched in a card which is then used to enter the inquiry.

Since inquiry from a telephone is restricted to ten digits, there are some applications where it is desirable to have alphabetic as well as numeric inquiry. In those cases, a terminal, such as the IBM 1001 shown in Fig. 3, can be used. Numeric and/or alphabetic data can be entered from the 1001 using a punched card. When an IBM 1001 inquiry terminal is used, a data set transmitter (401E3) per line is required in addition to the data set receiver per line for inquiry. In a method similar to the direct telephone attachment shown in Fig. 2, the initial number is dialed using either a card or a RAPIDIAL* telephone.

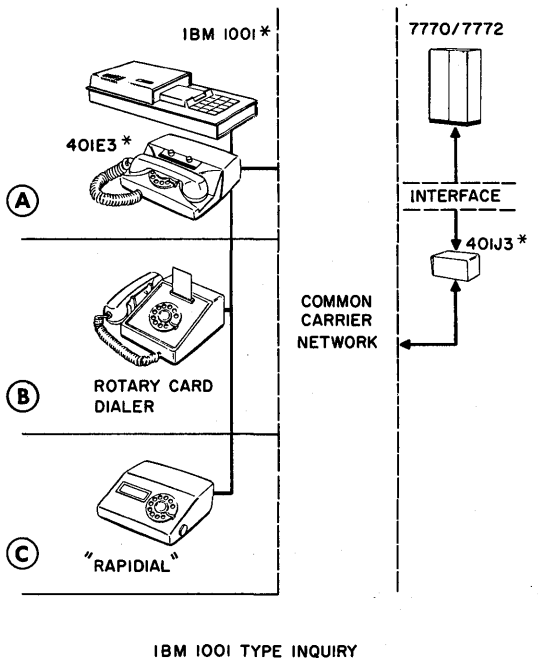



Figure 3. IBM 1001–type inquiry or equivalent.

## AUDIO OUTPUT

On the 7770, vocabulary is stored on a drum similar in form to storage of words on a tape recorder. Words are stored around the circumference of the drum surface on tracks. The drum has 128 tracks; it is 4 inches in diameter, 10 inches long, and rotates at 120 revolutions per minute. Each drum track has an associated read head and amplifier for retrieving the recorded word impulses. The rotational speed of the drum dictates that the information per track must fall within a 500-millisecond time period. A process has been developed in IBM to compress words or segments of speech into 500-millisecond time-slots. Words having a time duration greater than 500 milliseconds are placed in 2, or more time-slots or tracks.

The vocabulary is first generated in the following way. An elocutionist speaks the vocabulary words onto a tape recorder. This tape is then digitized through an analog to digital encoder, the output of which is edited and processed by a computer program to fit the words into 500-millisecond time-slots; these digital time-sloted words are stored on tape. At such times as a specific vocabulary is required, words are converted to analog form and placed on the drum at the specific track locations required by the application. This is normally done once for each application. Vocabulary modification is accomplished by removal of the recorded drum cylinder and its replacement with another cylinder having a different vocabulary.

The way the vocabulary is accessed is depicted in Fig. 4, which shows a functional diagram of the drum and the associated analog circuits. For each application, the processor has a table of addresses corresponding to vocabulary words. Upon analyzing an input message, the processor formulates the required output message which is transmitted to the 7770. This digital output message consists of a series of drum track addresses preceded by a line address. Each track address conditions a specific word analog gate allowing a word to be gated onto the Pulse Amplitude Modulation Bus (PAM BUS). From this bus, the word is gated through any message analog gate conditioned by a specific line address. This allows each word to be transmitted to any line and simply represents time-division-multiplexing of the analog word signal. Since this leaves the audio in a rather chopped-up state, the signal passes through a reconstruction filter before being transmitted to the output line. As long as a relatively high sample frequency (e.g., 12KC) is maintained, no appreciable degradation in audio quality is noticed. It must be remembered that the audio output is in the 200 to 4000 cycle per second range.

For the 7772, the method of recording vocabulary is similar to that for the 7770; however, the processing phase is different. The speech on tape is converted to digital data through an analog-to-digital encoder. The processing of vocabulary is then accomplished by band-compressing each word, thereby limiting the numbers of digits per word. This results in providing a stream of digital data which is stored on cards or tape for later transfer to a disk file or similar random-access storage device within the system. This digital data, called Digitally Coded Voice (DCV), consists of sequential aggregate and excitation functions. An aggregate function is 45 bits in length and represents a portion of the analog signal. The excitation function is 8 bits in length and acts as a counter determining the length of time an aggregate function should be used for a specific segment of analog signal. The sequential combina-

*Registered trademark of McGraw-Edison Co.

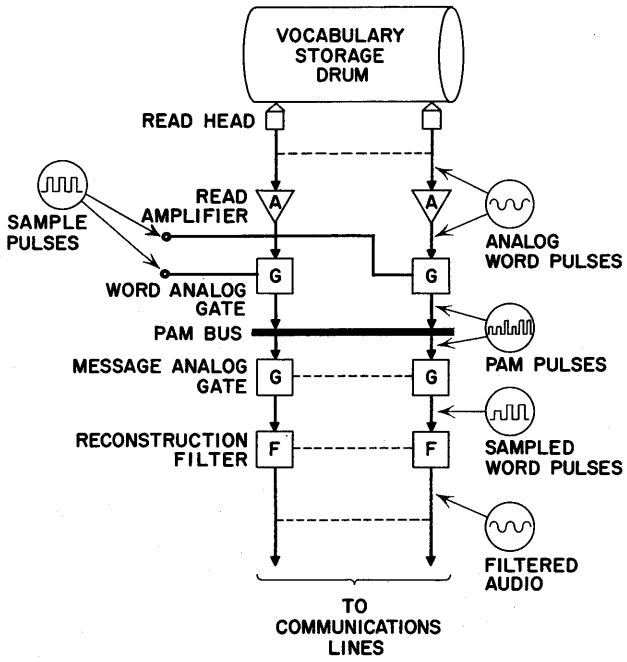



Figure 4. PAM bus and related gating in IBM 7770.

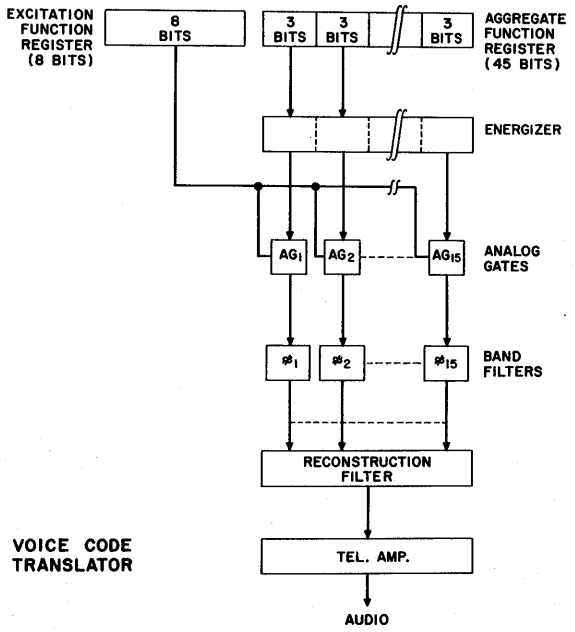



Figure 5. Voice code translator in the IBM 7772 audio response unit.

tion of these aggregate and excitation functions makes the DCV representation of a particular word of vocabulary.

Approximately 300 bytes of data (DCV) are required per second of audio. The word "balance," for example, would be stored on a disk file as approximately 75 bytes of digital data (600 bits).

Vocabulary in the 7772 is not stored on a drum in analog form but is stored within the system in bit form on a disk file. The CPU reads the required vocabulary words into its own core storage from the file. The information from the file is transferred through the multiplexer channel to the 7772 control unit which, in turn, transmits the data to a Voice Code Translator (VCT); the VCT then converts the data from digital to analog form. One Voice Code Translator is shared by every two input lines.

The Voice Code Translator consists basically of a set of 15 filters, each with suitable energizers and covering the voice frequency range of approximately 200 to 3,700 cycles per second. Each filter covers a specific portion of the voice band. For example, filter No. 1 covers the 200 to 300 cps range, whereas filter No. 15 covers the 3,150 to 3,700 cps range. An audio output is obtained when a combination of these filters is, in effect, energized by a pattern of input data.

A functional diagram of the VCT is shown in Fig. 5. An aggregate function representing a particular portion of a word is placed in the aggregate function register with its corresponding excitation function in the excitation function register. The DCV data (aggregate function) is converted to analog form and gated into the band filters, thereby energizing each band filter to a level and frequency determined by the format of the aggregate function. The output from the band filters is integrated, reconstructed, amplified, and transmitted to a telephone line as audio. The length of time that each aggregate function is used is determined by the excitation function which in turn is dependent on the dynamic range of the analog signel. For example, a constant tone would be signified by using an aggregate function having a corresponding excitation function with a large count.

### *Vocabulary*

Since the 7770 vocabulary is stored within the device on a drum, the number of words per application is limited to the number of tracks on the drum. (See Table 1.) The words are chosen from an available vocabulary list. Special words, proper names, and dialects, etc., can be obtained on a charge basis. A user may change his vocabulary by requesting the drum rotor in the machine to be rerecorded with new words of his choice.

There is no storage of vocabulary within the 7772 unit; instead, the words are stored on available random-access devices within the system to which the 7772 is attached. Because the words are in digital form, they require about 2400 bits of storage for each second of speech. If an average speaking rate of 180 words per minute is considered, this means approximately three words per 2400 bits of storage. There is, therefore, only the system limitation of available storage restricting the size of vocabulary per application for the 7772. Changing the vocabulary is simple since it requires only the reading of new words from cards or tape into storage. A list of words is also available for the 7772 with special words or dialects again offered on a charge basis.

## DATA FLOW

Both the 7770 Model 3 and the 7772 connect to the common-carrier data set receivers on one side and, on the other, to the multiplexer channel of an IBM System/360 Processor. Input information is entered via a telephone or similar terminal through a common-carrier data set to the 7770 or 7772 input section. It is then converted to data interpretable by the processor and is forwarded, a character at a time, to the processor. The multiplexer channel has the capability of interleaving operations from many low-speed input/output devices and thus provides a high degree of I/O efficiency and adaptability.

The multiplexer channel operates asynchronously with the central processing unit and contains several subchannels. One subchannel is used per I/O line. Data is transmitted to and from the subchannel and the line in parallel 8-bit-byte form. Data operations between the 7770 and the multiplexer channel are controlled by a sequence of commands which, in turn, are controlled by the Operating System/360 control program. The functional block diagrams of a basic 4-line 7770 Model 3 and a 2-line 7772 are shown in Fig. 6 and 7, respectively.

To understand the operation in more detail it is first necessary to define some basic commands used in IBM System/360 for communication devices.

ENABLE — is used to condition a line for accepting or maintaining a call.

DISABLE — is used to condition a line for terminating a call.

READ — is used when information is being transmitted from the 770 to CPU storage.

WRITE — is used when information is being transmitted from CPU storage to the 7770.

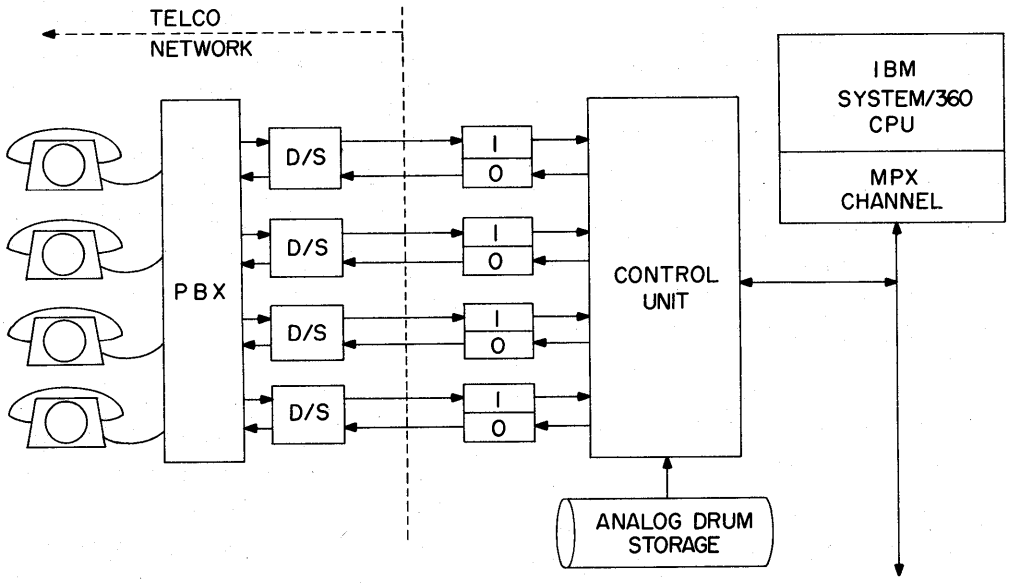



Figure 6. Functional block diagram of a 4–line IBM 7770-3 audio response system.

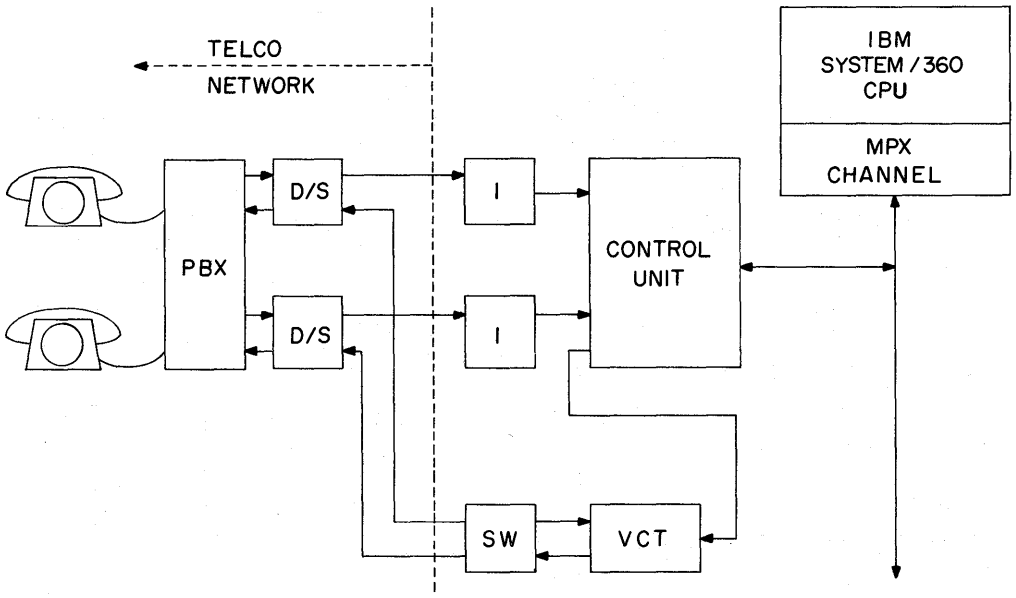



Figure 7. Functional block diagram of a 2–line IBM 7772 audio response system.

All of the above commands are under the control of the IBM System/360 Operating System Control Program. The flow chart (Fig. 8) describes the operation.

A transaction is initiated by removing the telephone receiver and dialing the machine number. If the number is not busy and the line has been previously ENABLED, an inquirer hears a tone that is 2 to 5 seconds in length, indicating that the call has been answered. The inquirer then dials an inquiry character and the 7770 or 7772 requests service from the multiplexer channel. If the caller does not hang up and the character is accepted, it is transferred to processor core storage. This operation continues until the caller stops entering characters (as is indicated by a time-out of 5 seconds or by the caller's depression of an "end of message" key). If this time-out occurs, the device indicates "end of message" to the multiplexer channel which interrupts the processor. The processor will analyze the completed inquiry and formulate the response message in the form of a sequence of drum addresses (7770) or "DCV" (7772). This sequence of drum addresses or DCV is preceded by the address of the line requiring the response. The multiplexer channel uses this address to identify the line requiring the response. Each time the multiplexer channel sends an output message to the 7770 or 7772, an audio word or portion thereof is gated onto the respective line requiring it.

This is repeated until the channel control word (CCW) count is zero indicating the last word. If conversation mode *is not* indicated, the line is then disconnected by issuance of a DISABLE command and then re-ENABLED for a new inquiry. If conversation mode *is* indicated, it will either be a READ command indicating more information from input, or a WRITE command indicating additional response will be transmitted by the processor. Conversation mode is a means by which a caller can effectively conduct a conversation with the computer under control of the operating program.

## APPLICATIONS

The choice of either a 7770 or 7772 Audio Response Unit depends entirely on the application. The number of calls per day, the length of the output message, and the size and variations in vocabulary are all factors that must be considered. The 7770 has greater throughput offering service to more lines than the 7772 but has the limitation of approximately 128 vocabulary words per application. The 7772, on the other hand, is limited in vocabulary only by available storage. The 7772, has a lesser number of input/output lines and has less

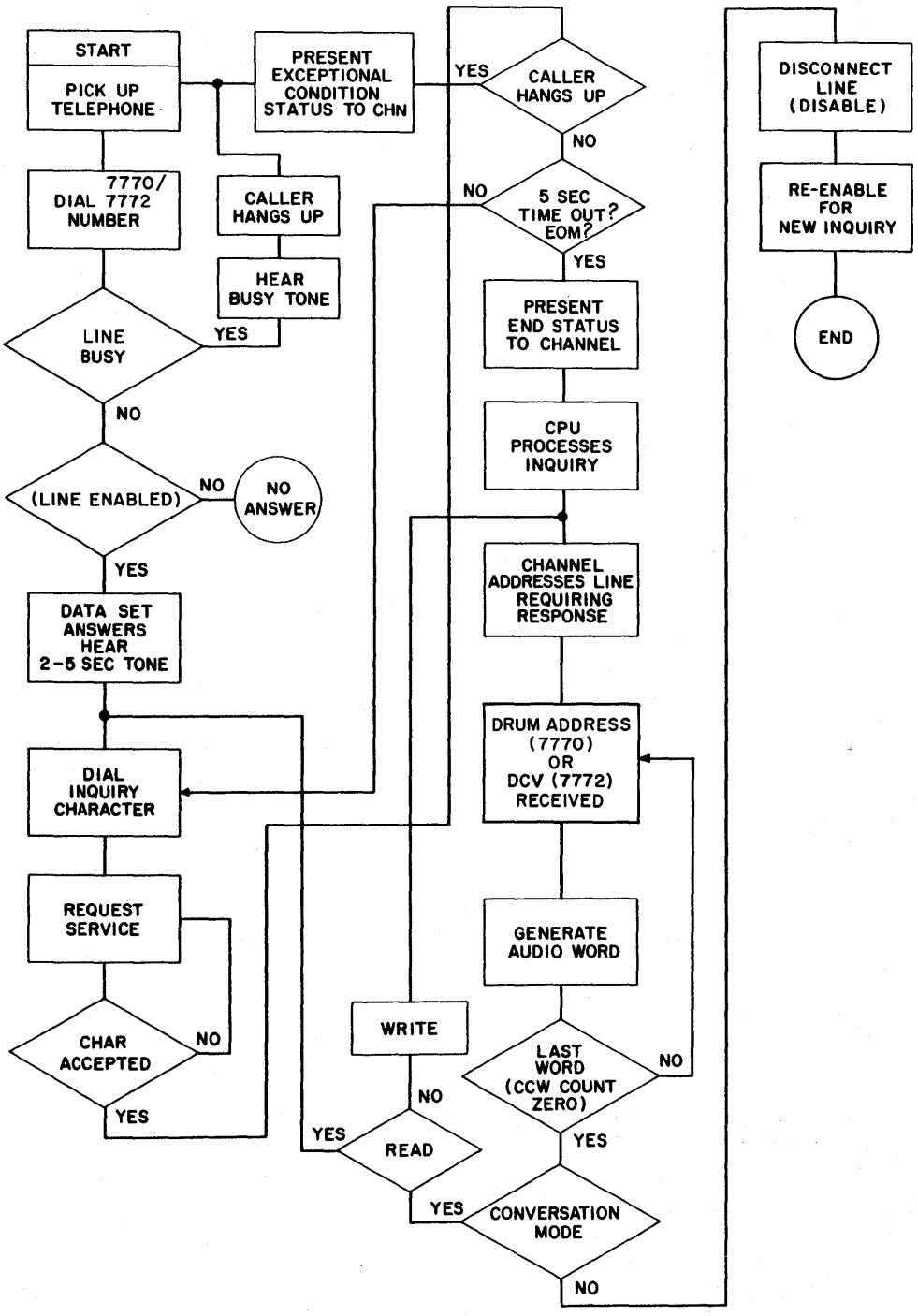



Figure 8. Flow chart of inquiry and response.

throughput imposed by the higher data transfer rate across the device/processor interface during vocabulary transfer from the processor.

Both Audio Response Units are suitable for use by any business having a centralized file system in which the telephone is useful as an inquiry terminal. Responsible individuals within an organization can dial the computer directly for any information

about an account and receive a voice answer within seconds. Only a small number of words in the vocabulary list take more than ½ second; hence, a 20-word message generally takes less than 10 seconds.

Because audio response is a recent addition to data processing, the application of these devices is left largely to the imagination of the system application engineer. A number of applications have been defined but are too lengthy to describe within the scope of this article. A brief description of a banking application follows:

*Banking*

Account, loan, credit or mortgage type inquiries can easily and quickly be handled by voice response. When a teller needs to determine the account status of a customer who desires to cash a check, he simply picks up this telephone receiver and dials the account number, and follows with some predetermined code; for example, dial 6 for account status information. The voice response received confirms the account number by repeeating the number the teller dialed, and then supplies the account balance and any other information the teller requested. After the customer cashes his check on the teller's approval, the teller, still holding the line, dials the code to inform the processor to update the customer's account by debiting the amount of the check. In most cases, the transaction from initial inquiry to final response will take the teller less than one minute; there is no paper-work involved and the teller is using a terminal he is already familiar with—the telephone.

Other applications are:

| | |
|---|---|
| Finance . . . . . | - stock quotations<br>- margin account balance |
| Insurance . . . . . | - policy status<br>- premium information |
| Retail . . . . . | - credit inquiry<br>- inventory inquiry |
| Manufacturing . . . | - inventory inquiry<br>- job status<br>- parts cost inquiry |

The addition of Audio Response to the family of IBM System/360 input/output devices now provides a closer link between man and the equipment that stores and processes the data with which he is concerned.

# CORRUGATOR PLANT OPERATING SYSTEM

Walter J. Koch
*IBM Corporation*
*San Francisco, California*

## INTRODUCTION

The corrugated container industry currently comprises approximately 850 plants, centered mostly in larger industrial areas. The average container plant employs 88 people and has an annual sales revenue of two million dollars.[1] In a working day, the plant will produce upwards of 100,000 containers to meet the shipping requirements of neighboring industry.

The process of converting rolls of kraft paper into these containers begins with the production of corrugated paper board. The corrugator joins outer sections of the board to the inner section, which has been corrugated for strength, and cuts the board into rectangular blanks representing specific container orders. The blanks are then processed by a variety of machines, such as printer-slotters and folder-gluers, to produce the finished containers.

This manufacturing process is becoming steadily more complex because of the tremendous variety of containers required by American industry, and also because of the trend in the container industry to provide more specialized and marketable container designs and services. This results in a continual search for more efficient container production techniques. One technique which has been recently investigated, and is now being implemented, is the extensive use of the digital computer for production planning and monitoring in the container plant.

The most significant result of these recent studies into using a digital computer profitably in corrugator plant operation has been the conclusion that top plant operating efficiency can best be obtained through a computer-oriented Plant Operating System. Such a system, with its computational, analytical, and data collection capabilities, would supply plant management with the timely, pertinent operational information needed for most effective operating decisions. Applying these capabilities to all aspects of plant operation would provide the necessary integration of these interrelated requirements:

- Efficient order scheduling and processing
- Maintaining the lowest reasonable plant inventory
- Providing the best customer service possible

## SYSTEMS FUNCTIONS

A group of container orders is scheduled on the corrugator using a series of patterns which contain orders combined with each other. Combining these orders to produce more efficient corrugator sched-

ules is a natural area to begin using the computational capabilities of the system. It has been shown that significant savings can result in this area of plant operation alone. However, to realize as much of these potential savings as posible, the computer must have complete, accurate files showing the status of orders and plant inventory. Entrusting the maintenance of these files to the system, with its inherent advantages in accuracy, becomes a natural consequence of corrugator scheduling on the computer.

The maintenance of complete inventory files leads to the application of scientific inventory management techniques which permit plant management to exercise more effective inventory control. These techniques, many of which are being used successfully today, will be used to maintain the lowest possible number of stock widths for each of the various paper grades, as well as the smallest number of rolls of each width.

In addition to the files showing raw material inventory, files will be kept on orders being processed. The progress of these orders can then be shown for management review and action. Timeliness and accuracy of these files will be obtained by maintaining them automatically, using input from data collection stations in the work centers on the plant floor.

## EXTERNAL SYSTEM FEATURES

The true effectiveness of the Plant Operating System depends on how well it performs in the corrugator plant environment. The system should require as little manual data preparation and manipulation as possible. On the other hand, it should have features which make it easy for plant management to obtain the operating information which resides in the system. The inherent flexibility of digital computer logic and the variety of computer input/output devices, make these system communication features readily available.

## INTERNAL SYSTEM FEATURES

In addition to these required external features, there are two basic internal features which are required:

- Large random access memory
- Comprehensive computer monitor program

A large random access memory auxiliary to the main computer memory is essential to store the many data files and computer programs which will be incorporated in the completed system. A computer monitor program is one which controls and supervises the execution of the various operating procedures. This control is needed to allow plant personnel to quickly and easily specify the procedures they wish performed.

## CORRUGATOR PLANT OPERATING SYSTEM DEMONSTRATION

This demonstration was prepared to show the advantages of the Plant Operating System concept by putting a significant portion into operation. Processing box orders was chosen to highlight the value of integrating operating requirements so that they can be monitored on one computer system. The demonstration performs the routine processing of order information as well as optimum scheduling of the orders on the corrugator and the costing and operational analysis of the corrugator schedules. Plant operating personnel need only enter orders into the system as they are received at the plant. The system then provides all information needed for the operating decisions which result in complete, effective processing of these orders. The demonstration is designed to run on an IBM 1620, equipped with an IBM 1311 Disk Storage Drive and running under control of the 1620 Monitor. These corrugator plant applications are included:

- Corrugated Container Cost Estimation
- Corrugator Order Entry
- Order Request
- Stock Width Request
- Corrugator Scheduling
- Corrugator Schedule Costing and Operational Analysis
- Order File Maintenance

These procedures are performed by simply entering the monitor control card specifying the execution of a particular procedure, together with the required data cards. All data cards, other than the initial order cards, are supplied by the system.

The order processing cycle in a plant begins when order cards are entered into the system by executing the Corrugator Order Entry Procedure. These new orders are merged with the existing open

order file on the disk and the updated order file is printed on the option of the planner.

Scheduling activity begins by executing the Order Request Procedure which scans the open order file on the disk, selects those orders which are for specified grade-flute codes and which fall within specified due dates. These orders, which are grouped by grade-flute codes, are printed as well as punched, allowing the production planner to select only those cards for orders he desires to schedule.

Selected cards from the Order Request Procedure are then fed into the Stock Widths Request Procedure. The stock widths available for each grade-flute group are punched as well as typed for the planner's information. Other information required for scheduling is also punched, such as material costs and operating costs. Schedule information which the planner might wish to vary frequently, such as minimum corrugator run and maximum number of slitter knives, may be entered through the typewriter and punched. The punched output from this procedure represents a complete input deck for the order scheduling procedure for the orders for each grade-flute group.

Input decks for each grade-flute group are fed into the Order Scheduling Procedure to be scheduled on the corrugator. This procedure uses a special linear programming technique developed by the IBM Research Laboratory, Yorktown Heights, N.Y., to determine optimum schedules for the grade-flute order group.[2,3] In addition to printing the schedules, this procedure punches them, making schedule information available as input to the costing and order file maintenance procedures. Since the schedule is in punched cards, the production planner has the ability to change schedule information as he wishes before proceeding to these succeeding procedures.

Schedule cards from the scheduling procedure make up the input to the Corrugator Schedule Costing Procedure which determines material, trim, setup and operating costs for each corrugator run as well as a schedule summary showing costs and operating statistics for the entire schedule. This procedure can be run for a projected corrugator schedule or for a schedule that has already been run, giving projected or actual operating costs.

Schedule cards from the Scheduling Procedure also make up the input to the Order File Maintenance Procedure. After a schedule has been run on the corrugator, this procedure updates the order file by analyzing the amounts run for those orders scheduled. Orders which have been completed are deleted from the order file. Orders which have been partially completed are reduced by amount run.

Using the Corrugated Container Cost Estimation Procedure occurs somewhat independently of the other procedures, although it would frequently precede the order entry procedure. This procedure checks box orders for validity and determines the corrugator blank size to be run from the dimensions of the box order. Various material and production costs are combined to determine the sales price for the order. Finally, the shipping weight of the order is determined.

Since this family of procedures can be executed in a variety of sequences, the planner has the ability to do different phases several times, varying such things as order grouping, tolerances, and stock width availability. In this way, he uses his experienced judgment in applying the procedures most effectively. For example, having reviewed a particular schedule, he may decide to remove an order from that group and try to obtain a new schedule with a higher corrugator utilization. On the other hand, he may decide to go back to the Order Request Procedure with different due dates, obtain more orders to add to the original group, and then obtain a more efficient schedule with the new larger group.

## CONCLUSIONS

The Plant Operating System Demonstration highlights the desirability of implementing a complete Plant Operating System. The basic principles of integrating related plant applications to operate in a computer monitored environment can be extended to include plant production scheduling, inventory management, production analysis and production control through on-line production monitoring. The Plant Operating System Demonstration represents the beginning of the systems design for such a truly comprehensive system.

## REFERENCES

1. "1958 Census of Manufacturers," *Summary Statistics,* Bureau of the Census, vol. 1.

2. P. C. Gilmore and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Prob-

lem," *Operations Research* vol. 9, pp. 849-859 (1961).

3. P. C. Gilmore and R. E. Gomory, "A Linear Programming Approach to the Cutting Stock Problem, Part II," *Operations Research,* vol. 11, pp. 863-888 (1963).

# REAL-TIME PROGRAMMING AND ATHENA SUPPORT AT WHITE SANDS MISSILE RANGE

William G. Davidson
*Computer Directorate*
*White Sands Missile Range, New Mexico*

## THE ATHENA SYSTEM AND GROUND COMPUTER ASSIGNMENTS

The Athena is a four-stage missile launched from Green River, Utah toward White Sands Missile Range, New Mexico, covering a ground range of 450 miles. It is designed to deliver a payload into White Sands with prescribed atmospheric re-entry velocities and angles, so that re-entry phenomena may be studied, and radar performance against re-entry bodies may be evaluated.

A ground computer (presently an IBM 7044) operates in real time during an Athena flight, with seven assigned tasks:

1. Providing instantaneous impact prediction data during first and second stage burning, for range safety use. This function is especially important since the Athena is flying over populated areas and, in case of missile malfunction, must be destroyed by safety personnel so as to land only along non-populated sections of its flight path.
2. Computing and transmitting midcourse (between second stage burn-out and third stage ignition) guidance correction commands to bring the missile into the proper re-entry orientation.
3. Evaluating telemetered data from the missile following the transmission of the guidance commands, to determine if all system functions are operating within prescribed limits.
4. Transmitting the third stage ignition command to the missile at the proper time, once it has been established that the third stage can be safely fired.
5. Transmitting precise pointing data to several tracking and measurement sites throughout the flight, to assist them in acquiring and following the missile.
6. Providing ground support personnel with data displays and flight status information throughout the mission.
7. Logging all computer inputs and outputs in real time for postflight analysis.

The following sections, while relating to real time philosophy in general, will at the same time show the development of the above tasks as a multiprocessing job under the real time constraints of input and output data demands.

## MONITOR PHILOSOPHY

All real time programs within the Computer Directorate at White Sands operate under the control of an Executive Control Monitor which directs the flow of activity among various functional modules to assure that all program functions are carried out at the proper times and in the necessary contingency to related activities. The Monitor was originally adapted for Athena use from the Mercury project monitor, but has been revised to make it more applicable to specific support problems encountered at White Sands.

A multi-processing job is a programming task that can be subdivided into functional modules or processors, each of which executes a specific function in the overall program, and can be in various stages of completion throughout execution of the entire job. Each functional processor should have well defined inputs and outputs in relation to other processors in the system, and in relation to the outside (non-computer) world. The Monitor might well be referred to as a multi-processing monitor, since in controlling the relative flow of activity between functional processors, it monitors the state of completion and the dynamic requirements of each processor in the job throughout execution of that job.

A multi-processing job becomes a real time job when execution of various program functions becomes dependent on some regular or random time constraint, such as input or output data requests from a device operating dynamically during program execution. Thus, for a real time job an additional, or perhaps we should say a master, processor must be added to the other functional processors in order to service the real time demands and pass any pertinent control information on to the Monitor. For the Athena system this processor is known as the trap or interrupt processor, to which control is given immediately whenever a data input or output demand is recognized, interrupting whatever functional processor may be in process at that time. The trap processor examines the real time request, reads in or transmits the required data, and provides the Monitor with information regarding the input or output action just completed so that either execution of pertinent program functions may be initiated, or control will be returned to the interrupted functional processor.

Each real time processor can be classified in one of three time-oriented categories, and relative execution priorities can be assigned on this basis. First are those activities that must be executed immediately upon recognition of some event or time in the program. Examples in the Athena program are the transmission of the third stage ignition command within a critical time interval, or the plotting of range safety display data without undue delay after they have been computed. The second category consists of functions that must be executed within a given time interval after they are requested, such as the editing and processing of direct data messages before the next sample arrives at a remote terminal, or the calculation of regularly transmitted computer outputs within a time dictated computation-transmission cycle for those data. The third group of processors involves program functions whose execution can be deferred until some later point in the mission, when higher priority functions have been completed and computing time is available. Included in this category would be such functions as selecting a radar for later use on the basis of current data from several radars, or the calculation of guidance commands to be transmitted at some later point along the trajectory. The first class of functions described must, of course, have highest priority, since initiation of these actions usually cannot be delayed for any other purpose. Other processors will be assigned lower priorities as the importance of their roles and the judgment of the program organizer dictate. The trap processor, as mentioned before, will usually assume control whenever a data demand appears at a remote terminal, but even trap processing can be temporarily inhibited if an extremely critical function is in process at the time the demand occurs.

Figure 1 shows the relationship between functional processors, a trap processor, and the Executive Control Monitor. A simplified version of the Monitor is shown since this paper intends only to outline its control functions, omitting many of the details and system tools available. When an input or output data demand interrupts a functional processor, the contents of all machine registers are saved as control is given to the trap processor. When control is eventually returned to the processor at the point where it was interrupted, these machine registers can be restored and processing can resume. Numerous control tables and indicators link the functional processors to the Monitor. Some of these

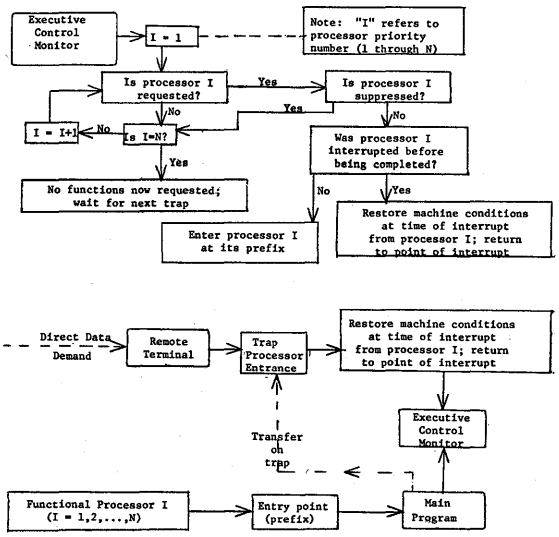



Figure 1. Monitor-trap processor — functional processor linkage.

are set initially to define, among other things, the entry points and job priorities assigned each processor, while others are maintained dynamically for use in monitoring the status of each processor.

Among the indicators that change during job execution are three basic ones described here. First is a "request" indicator for each processor, turned on whenever execution of that processor is requested by another segment of the program. For example, the program might contain a processor which edits or evaluates radar data, in which case this function would probably be requested by the trap processor each time a new set of radar data enters the computer. When control is returned to the Monitor by the trap processor following the data interrupt, the Monitor would see the request indicator on for the radar processor, and would transfer control there as soon as any higher priority activities had been completed. The second indicator is a "suppression" flag for each processor, causing the Monitor temporarily to suppress execution of that processor even though its request indicator might be on. Any processor can be suppressed by any other processor, and can likewise be released by turning off the suppression indicator. The third indicator is the "in process" indicator, turned on whenever execution of the functional processor associated with it has begun but has not been completed, and turned off upon normal exit from that processor.

## PROCESSOR QUEUEING AND ROUTING SHARING

Real time processes often encounter times of peak activity where processor execution requests occur more frequently than control can be given to the processor. In order to avoid losing any of these requests, a queueing facility is provided by the Monitor, which allows the stacking of requests until computing time is available for servicing them. A basic assumption, of course, is that overall program timing has been so established as to assure the execution of queued functions within some limiting time frame and before the queue tables for that processor have been filled.

Another problem occurs when two or more segments of a multi-processing job share the same subprocessor, resulting in a possible loss of indicators and temporary storage within that subprocessor. At least four solutions may be applied to this problem, all of which have been used at the White Sands facility with varying degrees of success. First, and perhaps most obvious, is to avoid the problem by loading duplicate subprocessors, though this is a space consuming procedure. Second is a modified queueing philosophy applied to subprocessors. This is ideal in some instances, but may tend to subvert the priority assignment given to processors unless handled carefully at the subprocessor level. Third is the disabling of data traps during execution of a subprocessor, thus forcing the completion of that subprocessor prior to its call by another processor. This should be done only for short subprocessors, and even then there is some danger of losing vital segments of data while in the disabled mode. Fourth is the saving of temporary cells and indicators for the subprocessor at the time a data interrupt occurs. This method respects the initial priority establishment, but is often wasteful of computer storage and execution time.

## INPUT-OUTPUT AND INTER-COMPUTER COMMUNICATIONS

Inputs to the Athena real time program are radar tracking data (polar coordinates, site identification and tracking mode) from five radars; 15 channels of telemetry data from the missile in both a data and a calibrate mode format; signals such as lift off, stage ignition and burn out; and timing from four sources—a millisecond clock attached to a multiplex-

er control unit to provide timing pulses for radar interrupts, an astrodata clock measuring range timing in millisecond increments, a range timing word attached to each sample of telemetry data, and an internal computer core clock with a 16⅔ millisecond resolution, to be used if other timing sources fail. Computer outputs include acquisition data (polar or rectangular coordinates and site identification) to eleven tracking sites; commands to the missile via a command transmitter station, including stage firing commands and midcourse pitch and yaw maneuver instructions; impact prediction, present positions and other information sent to seven plotter display boards; and numerous binary and decimal displays showing the present trajectory status, the results of dynamically changing computer decisions, and various data and error messages appearing on an on-line printer throughout the mission. Input/output formats and data rates may vary throughout a mission.

Since 7044 core storage and computer speed prevent loading or executing the ideal large scale real time program from core, any data handling or formatting that can be done outside the computer simultaneous with computer processing is always welcome. For this reason the personnel of the Real Time Data Center at White Sands have designed a Direct Data Buffering System to handle many of the computer inputs and outputs without tying up valuable computer storage and time.

Radar data are transmitted from the radar sites to the Real Time Data Center over telephone lines via kineplexes, one sample requiring 15 8-bit bytes. These bytes come from the kineplex receivers every 3⅓ milliseconds, so that one complete frame of radar information is transmitted every 50 milliseconds with the data multiplexed for the five input radars, adding a time word from a master clock. Telemetry data and calibration words are transmitted over 14 pairs of telephone lines, formatted and buffered by the Direct Data Buffering System, and similarly sent to the 7044 on an interrupt basis about 11 times per second. The Direct Data Buffering System includes an output buffer to format and transmit acquisition data to the tracking sites via the kineplexes, and to transmit computer generated missile commands and display data to the proper device. Output data demands interrupt the computer every 50 milliseconds. The System displays the computer-buffer inputs and outputs dynamically, and records all real time inputs serially on analog tape as they enter the Real Time Data Center. Communication between the direct data buffer and the 7044 main computer is of the demand-response type through the direct data connection on Channel B of the 7044. Communication from the 7044 to the analog computer and plotting boards through the digital-to-analog converter is via the direct data connection on Channel C of the 7044.

A Direct Couple System (7044-7094) has recently been installed in the Computer Directorate, and the present Athena program is being rewritten for the new configuration. The new program will rely largely on the 7044 side of the system to edit and format data to and from input-output 7288 subchannels, replacing in large part the function of the present Direct Data Buffering System.

## DEVELOPMENT OF THE ATHENA PROCESSORS

Figure 2 outlines the organizational structure of the Athena real time program. The solid lines indicate a direct transfer of control from a processor to a subprocessor, in the manner of standard deferred time subroutines. The dashed lines represent a request for the execution of a processor by another processor. The function of each program in the system is described below.

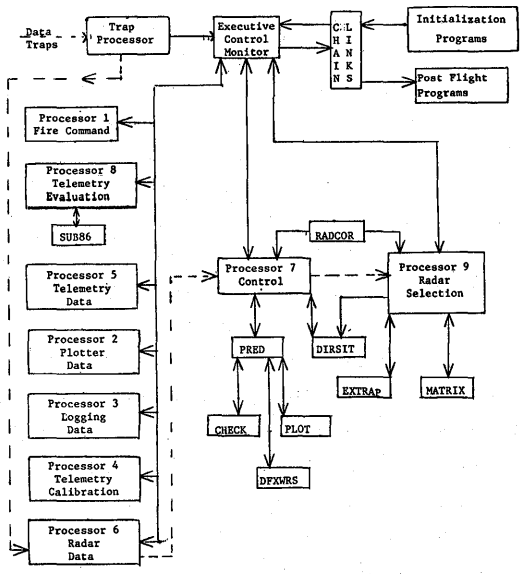



Figure 2. Athena program organization.

1. Priority Processor #1: transmits, at the proper time, a command to ignite the third stage of the missile.

2. Priority Processor #2: transmits data to display plotters via digital-to-analog converter and an analog computer.
3. Priority Processor #3: logs computer inputs, outputs, and other pertinent data on tape during a mission for post flight analysis.
4. Priority Processor #4: processes raw telemetry calibration messages.
5. Priority Processor #5: edits and processes raw telemetry data messages, and transmits midcourse guidance commands to the missile.
6. Priority Processor #6: edits and processes input radar data, monitors the tracking status of the radars, and transmits acquisition and display data.
7. Priority Processor #7: formats acquisition and display data, and controls the linkage and synchronization between the processors generating acquisition data, range safety plots, and various digital displays.
8. Priority Processor #8: evaluates telemetry data from the missile after midcourse guidance commands have been transmitted to decide whether all parameters are within prescribed limits.
9. Priority Processor #9: evaluates radar data after second stage burn-out, and chooses a radar from which input data for the guidance equations can best be obtained.

A. Subprocessor RADCOR: a subroutine of Processors 7 and 9, RADCOR performs coordinate transformations and generates acquisition data.

B. Subprocessor PRED: a subroutine of Processor 7, PRED generates Kepler extrapolated impact prediction and other plotting board data.

C. Subprocessor DIRSIT: a subroutine of Processors 7 and 9, and of PRED, DIRSIT generates smooth positions, velocities and accelerations from raw radar position data.

D. Subprocessor SUB86: a subroutine of Processor 8, SUB86 computes predicted payload impact points.

E. Subprocessor EXTRAP: a subroutine of Processor 9, EXTRAP performs trajectory extrapolations for guidance equations inputs.

F. Subprocessor MATRIX: a subroutine of Processor 9, MATRIX solves the guidance equations, computes missile attitude commands for midcourse guidance, and determintes third stage ignition time.

G. Subprocessor DFXWRS: a subroutine of PRED, DRXWRS selects a radar for instantaneous impact prediction plots on the basis of digitally filtered position and velocity data from all input radars.

H. Subprocessor PLOT: a subroutine of PRED, PLOT scales and formats plotting board data for transmission to the digital-to-analog converter.

I. Subprocessor CHECK: a subroutine of PRED, CHECK is a limiting subroutine for output data.

Note the high priority given to routines to transmit third stage ignition command and plotting board data for range safety use. This indicates that these functions must be executed within critical time frames. The data editing and processing routines are generally given next highest priorities since they must be completed before the next data samples reach the remote terminals. Functions that may span longer time intervals for completion, and which may, to some extent, be completed as available time permits, are given lowest priority.

## REAL TIME STORAGE LIMITATIONS

Most real time programs currently in use are of the medium to large scale variety, as is the Athena program at White Sands. An immediate problem arises when the entire executable program is too large to be accommodated by the immediately accessible core at one time. Three techniques are used by the Athena program to alleviate this problem, and may be worthy of mention here.

First, all data input and initialization programs are loaded into core and executed, and are then chained over or overlaid by the real time processors and subprocessors, the two program links sharing only the data and control storage common to each. The same procedure is followed after execution of the real time phase of the job, with post flight routines overlaying the real time processors, retaining only those data and control words common to both.

Second, all data which are generated during the real time mission but are intended only for post-flight use, are moved out of the computer immediately via a direct data connection or onto logging tapes, to preserve storage within the main cores.

The 7044 currently in use at the Real Time Data Center does not have a disc or drum storage unit, hence cannot use these facilities for conservation of core.

Third, much preprogramming, formatting, filtering, buffering, and functions of this nature are handled externally by the input-output Direct Data Buffering System and by the analog-to-digital link between the 7044 and the plotting boards. This eliminates large blocks of digital programming as well as conserving computer time. Future real time work at White Sands will probably move more into this field of hybrid operation as systems become larger and more complex.

## CHECK-OUT PROCEDURES

Check-out of real time programs requires methods not usually necessary in deferred time programming. First of all, of course, each module or functional processor must be checked out individually, as far as possible. If a multiprocessing job has been properly formulated, with each processor assigned well defined functions and communications with other program parts, this check-out at the processor level will be greatly simplified.

Repeatability of input data is necessary in order to reproduce the results of computation processes during check-out phase. Unfortunately, real time processes dependent upon external timing and data demands cannot usually be reproduced with any degree of precision. However, the ability to record actual input data during a real time process, in such a manner that it can be played back into the program exactly as it was recorded, assists the programmer in his check-out diagnostics. A related process is the establishment of a data simulator which will provide the real time program with repeatable data demands at prescribed time intervals. Both procedures have been used extensively in checkout at White Sands.

The complexity of interprocessor and monitor interactions is frequently exceeded by a maze of intercomputer and data communications problems. Errors in data formatting and transmission are difficult to trace during or following the completion of a real time process, pointing up a need for thorough diagnostic check programs for all external devices and intercomputer links prior to the execution of a real time job. Even with thorough premission check-out, the possibility always exists that some linkage device or external module may fail during a real time mission. Such possibilities demand the use of a limited amount of equipment monitoring by the program during a real time process. Upon occasion facilities can be provided for the switching of modules during the run in case errors are detected, thus endowing real time jobs with an additional feature not usually found elsewhere, that of dynamic check-out.

## DOCUMENTATION

The preparation for and writing of a real time program requires the close cooperation of a number of persons, among them the program administrators, coders, equipment specialists, and those establishing the functional requirements and job specifications. This, together with the extreme complexity of most existing real time systems, compels a formalization and documentation of requirements and procedures not usually needed for a standard programming job.

Once the functional requirements for a job have been set down, the program management group should incorporate the system functions into an arrangement of processors and subprocessors which will form the backbone of the programming assignments. This initial work must be done with great care and imagination, since the establishment of time dependent linkages and relative priorities among processors is basic and can be altered only with great difficulty once the programming work has begun.

After the overall system documentation and flow charting has been completed (ideally before any other work has begun), program specifications and detailed flow charts can be drawn up for each functional processor. Specifications must be controlled by someone familiar with all parts of the program so that logical program linkages may be defined. Detailed flow charts should be primarily the responsibility of the individual programmer, and just as in deferred time programs, can be compiled before, during, or after the program has been written.

Standards for program management and documentation need to be set down early in the game, and followed rigidly even when the pressures of time begin to close in on the programming team. Not only will these standards aid the program man-

ager and his real time staff during the program writing stage, but also will establish a basis for future changes and program module rewriting when that becomes necessary. The competent real time programmer will be called upon to use all the ingenuity and creativity he possesses to solve the problems he faces, and poor standardization and documentation will make his task more difficult and lead to a substandard quality of work.

Most of the difficulties encountered in the Athena and other White Sands real time systems have been due to breakdowns in communication and documentation. The establishment, check-out and maintenance of a sound real time system is sufficient in itself to tax the patience of the most erudite team, and the use of such a conceptually simple yet basic tool as documentation cannot be overemphasized.

## REFERENCES

1. F. N. Berry, "Re-entry Vehicle Testing at White Sands Missile Range," (not for distribution outside White Sands Missile Range) (April 1965).

2. R. V. Head, "Real-Time Programming Specifications," *ACM Communications* (July 1963).

3. T. A. Holdiman, "Management Techniques for Real Time Computer Programming," *ACM Journal* (July 1962).

4. E. H. Schuetze, "The Direct Data Buffering System at WSMR," (not for distribution outside White Sands Missile Range) (June 1964).

# QUALITY EVALUATION OF TEST OPERATIONS VIA ELECTRONIC DATA PROCESSING

A. A. Daush
*Hughes Aircraft Company*
*Space Systems Division*
*El Segundo, California*

## INTRODUCTION

The test operations and data analysis engineers have taken giant steps in utilizing computers and data processing methodology in the various scientific fields. This is especially true in electronic weapons systems and space vehicle and related systems being developed and proposed. Procedures, command sequences, and data collection and reduction have reached high density and are being treated via EDP. However, to what extent have related disciplines such as quality engineering utilized this same approach? How have they integrated or been integrated into this complex? What steps appear to be in the offing to insure that the attention of the electronic data processing community is focused upon this problem and can present a case suitable to secure management backing where capital expenditures, personnel, and space are considerations? More questions may be raised than answers given. However if some actions are stimulated the basic objective will be fulfilled.

## TEST OPERATIONS USAGE OF EDP

The testing of complex electronic and related systems has had a series of requirements placed upon the operations team and has created a real man-computer methodology that can be the slave or master of the program. The design of the testing system normally follows a very hybrid and interwoven sequence involving many decision paths and management constraints. Some of the rationale might include:

Technician operation vs automatic command

Quantity of command and control elements per timing sequence

Real or near real time status displays and decision capability

Types and quantities of data secured

1. Analog
2. Digital
3. Tape, paper or magnetic
4. Visual vs recorded and confirmed

Near real time analysis and decisions on validity of results

Fluidity and control of change of test procedures and system test requirements

Quantity of peripherals to support the program., i.e., card sorts, printouts, key punch, etc.

Program change control to assure machine match with equipment being tested

Level of data verification and trouble shooting routines, i.e., self-test and failure isolation

Machine costs—capital vs project

Reliability

We have seen in the test field numerous interfaces with instrumentation, controls, high-speed analog and digital inputs, outputs, and computations involving test operators, all or many of which may be external to the actual computer complex.

For a fully automatic test or checkout system the programmer must understand the above and have researched the procedures and must understand the requirements. He must understand the equipment being evaluated almost as well as the design and system engineer. He must then define the complete test program to accomplish the task. Obviously this must all be accomplished (1) to a schedule, (2) within cost, (3) to meet performance, and (4) it must be reliable. How can such a demanding set of diverging requirements accommodate yet another delay like Q.C. evaluation overlays and requirements? If the checkout system requires that the computing complex control and execute a series of complex test sequences and during these monitor and analyze a detailed number of variables for incorrect performance, how does the requirement for quality approval and for acceptance of this operation become specified, funded, and certified as true by the in-house Quality Control and the customer?

## THE QUALITY DILEMMA

Consider the major segments of a typical computer-controlled test operation including automatic real time performance evaluation and test certification. They are shown in Fig. 1.

How are the quality requirements integrated into this total data acquisition picture? *They for the most part are not!* Several organizations are noted exceptions. The normal procedure is for Engineering (a) to define the parameters to be measured, (b) define the allowable limits and then, (c) proceed to design the testing such that the test/test complex rationale leaves no unexplainable discrepancies. A data review then occurs as a later date with only some "quick look" information to verify moving on to the next phase. What, therefore, are the "Quality EDP" challenges?

The number of Q.A. and Q.C. managers and engineers available that can plan in depth in terms of EDP for quality acceptance are limited in number.

Upper management understanding and backing can be lacking in furthering Quality's role.

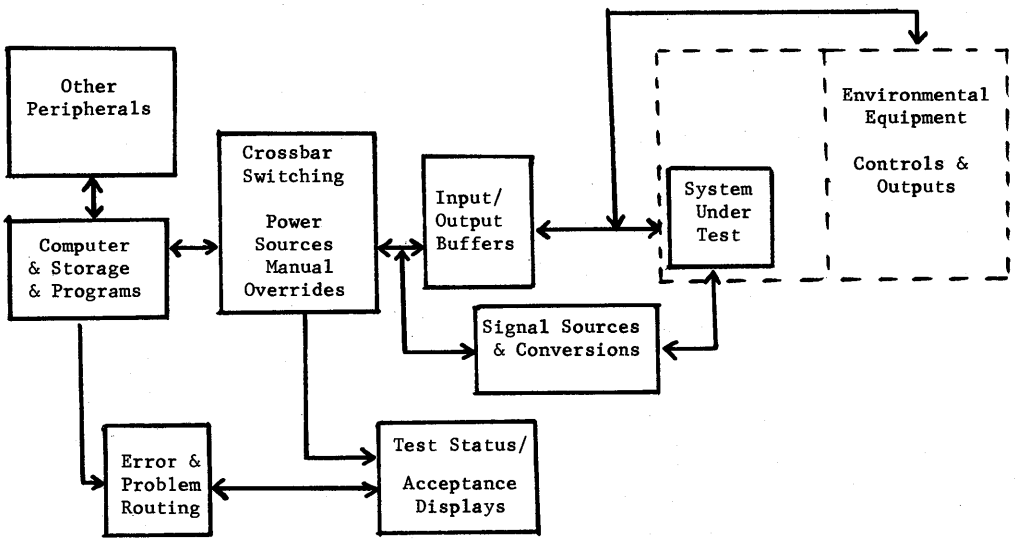



Figure 1. Major segments of a typical computer-controlled test operation.

The ability of Q.A./Q.C. to understand and to make significant technical contributions to the data mix. Too often this is only a paper mill increase at best and does not improve product, insure test results, or customer satisfaction.

This lack of understanding results in lack of trust in the data evaluation and long laborious manual data reviews are required to accomplish that which a computer can accomplish is several seconds.

What must Q.A. managers do to become capable in these fields?

They must develop competency in planning, analyzing, understanding, and contributing to test requirements, procedures, plans, and customer reviews.

They must move out to provide systems test planning with inputs to insure that Q.A. will be an accomplished factor in EDP evaluation programs, both analytically and factually.

They must provide a *complete technical service* to systems test, not just a paper service, or rubber stamp on results not understood.

Q.A. must become machine conscious in terms of EDP utilization for data summaries, quality control history records, calibration requests, customer mandatory product control, and customer acceptance review. This implies that machine techniques, instrumentation, error analysis, circuit design, workmanship, calibration, and checkout procedures have all been reviewed in depth and approved and Q.A. has only to review the results of a computer status report to make a "buy" decision and obtain customer concurrence.

Here is where the technical and management decision lies. Do the Quality organizations existing today in most hardware producing establishments have the opportunity and management backing to gear up technically to meet and utilize the EDP that computer engineering is refining. Can they then adapt these to quality systems and detail hardware evaluation? We must each answer this based upon our own experiences and known organization. I suspect that the answer will be "NO" more often than not, however, there are certain exceptions that are pointing the way.

## POTENTIAL SOLUTIONS

The following represent some thoughts that lead to an increase in communications and technical understanding of the changing evaluation methodology, i.e., real time computer data analysis and quality acceptance of product.

The buy-off criteria for most electronic systems performance and configuration must be designed from inception to accommodate in real time electronic data processing, analysis, and acceptance.

In-house quality organizations should have a strong but technically sound initial input into final system sell-off criteria. This requires a broad-based systems-oriented engineering capability with strong test and EDP background plus strong management backing.

Customer buy-off criteria must be adaptable to EDP and must be completely knowledgeable and must accept such output as evidence of satisfactory performance. In some fields where enormous quantities of data are produced and analyzed this is mandatory to economical operation.

Public acceptance of such techniques will also need to be cultivated, particularly where safety or welfare are concerned. Many agencies are already engaged in EDP techniques, such as banks, ticket information, billing, automatic machine tool operation, etc., and extending this to other disciplines is only a matter of time and economics which can be hastened through extensive missionary activity which is shaped to provide increased service, lower cost and decrease schedule span for Q.A. operations.

## CONCLUSIONS

If the above potential solutions are to be considered seriously, these actions should be actively pursued to insure a continued growth and increased Quality Assurance role in the utilization of EDP. This can be done if you:

1. Learn EDP.
   Think EDP.
2. Talk EDP.
3. Teach EDP.

4. Encourage engineering and manufacturing counterparts to do likewise.
5. Include EDP as part of your quotes and budgets but be really ready to technically and economically defend them by convincing yourself that the application is correct. Wrongly applied, your case can be pulled down around your ears.
6. Update your management as required. This must be skillfully done; such a seed needs careful planting, tender loving care, and much hard work in upward communications. Here, among others, films, items about potential competition usage vie related fields, i.e., case histories, visiting experts, and participation in EDP meetings, are in order.
7. Consort with the enemy. Find out what engineering, design, manufacturing, the customer is doing and planning in EDP. Take the initiative and assist and help design better techniques and applications that will, then, be completely acceptable to you — "quality."
8. Call upon the services of EDP organizations to assist with (a) meetings, (b) arranging speakers for programs, and (c) as a source for technically qualified assistance. Services are available for defining, reviewing, and planning implementation for extending the use of Electronic Data Processing disciplines deeper into the acceptance of product sphere for which Quality has final authority.

Since I have been operating in the role of the enemy for a number of years, i.e., electronic systems test operations and evaluation, the above is offered in the hope that this will provide incentive in many organizations to investigate and accept the tremendous technological advantage that the proper application and utilization of Electronic Data Processing will make in almost any major industrial effort involving equivalent systems sell-off and involving many parameters and technical judgements. The return in

- Performance
- Cost savings
- Schedules
- Reliability
- Technical satisfaction

will be your reward as well as your management's and will provide an important contribution to advancing and cross pollinating the Quality and EDP mixture to the betterment of each.

# THE INTRODUCTION OF MAN-COMPUTER GRAPHICS INTO THE AEROSPACE INDUSTRY

S. H. Chasen
*Research Laboratory, Lockheed-Georgia Company*
*Marietta, Georgia*

## INTRODUCTION

With the exponential growth of computer facilities, a geat deal of attention has been given to the analytical description of man's decision-making processes. Yet little has been accomplished of general value to automation. We can think of many examples where the human mind can assimilate information and quickly reach a decision where we would be hard pressed to computerize the thought process. Since it will be many years before man's general decision making powers can be channeled into computers, he must be given an optimum remedial problem-solving capability. This means that he must be given the facility to communicate or interact directly with the computer and he must be given adequate tools to accomplish this interaction. In an idealized man-computer system, facilities will exist to yield a homogenous mix of man's decisions with routine computation. With the addition of fast response, it will be possible to shorten span time and to increase the learning and the retention of significant results. To this end the concept of real-time on-line computer graphics will play a major role.

An optimum *real-time* capability is one in which the man receives a response to an inquiry to the computer or a requested action by the computer as fast as the requested response or action can be assimilated. Our present space program, for example, would not exist, as we know it, without this real-time capability. The closer we get to a true and general real-time environment, the closer we will be to maximum problem solving capability. Only with the most recent advances in computer speed and scope performance has a real-time on-line *computer graphics* system become practicable. With a visual display and the ability to interact with the computer through the geometric representations, it is possible to perceive and absorb significant information such as shape, area, proximity, density, and intersection to a degree that may obviate the requirements for special-purpose, complex, and cumbersome computations.

The graphic medium of communication is but one of many media by which man is attempting to maximize computer utilization. It is, however, a very important medium to which considerable research and development is being applied with the promise of rewarding results. The contribution of graphics in the real-time on-line system is manifested in all problems where a visual display and the facility to work with the display is desired.

This can be perceived in a broad spectrum of ap-

plications. For example, the designer wishes to create a small part or perhaps a large section of an aircraft. In either case, the ability to view, to evaluate, and to change the design while maintaining the mathematical definition on the computer will be an invaluable contribution. Then there is the program manager who would like to have an up-to-date review of his program and who would like to consider the effects of his proposed changes. The display of a PERT network and the facility to make direct alterations to observe the effect on the critical path will accomplish this function.

Thus the graphical capability augmented to the real-time on-line system will significantly increase the efficiency of solutions of many problems and will open the horizons to solutions of a new class of problems which have not been tractable in the past.

Research in the area of computer graphics reached a significant milestone when Ivan Sutherland completed his initial "Sketchpad" system in 1961. Using a cathode ray display interfaced to the Lincoln Laboratory TX2 computer, Sutherland showed the feasibility of supplying graphical or geometric information to the computer via the display. Cathode ray output has been around for a long time in the computer age, but two-way geometric communication was a revolutionary concept. Certainly, Sutherland is not the only one to have considered the significance of graphical, on-line I/O. General Motors has had a program in computer graphics which was initiated in 1959 and was, for competitive reasons, veiled in secrecy until the Fall Joint Computer Conference in October 1964. Another program which has considerable bearing on computer graphics is Project MAC (Machine Aided Cognition) under the direction of Professor R. M. Fano at M.I.T. Project MAC is a broad-based program delving into all aspects of man-computer systems. Significant achievements are also being realized under programs directed by Douglas T. Ross and Steven Coons at M.I.T. Mr. Ross is developing Automated Engineering Design (AED) compilers which will aid the user of man-computer systems in formulating and solving his problem with increased facility and versatility. Professor Coons is responsible for the development of programs associated with computer-graphic applications. His work is well known as Computer Aided Design.

The dramatic innovation of man-computer graphics adds a new dimension to computer technology. It is a new link in the chain which leads toward more complete automation, as information communicated by pictures or displays is often many times more effective than the written word.

## LOCKHEED-GEORGIA'S COMPUTER GRAPHICS

In its belief in the strong future role of man-computer graphics, Lockheed-Georgia was a pioneer in the aerospace industry when it acquired a system with emphasis on two-way graphic communication as defined above. In December 1964 the UNIVAC 418 Computer connected to the Digital Equipment Corporation's 340 Scope became operational in our Research Laboratory. The system whose graphical section is exhibited in Fig. 1 is dedicated to a research program in the application of man-computer graphic. Our research team consists of experienced personnel in a variety of

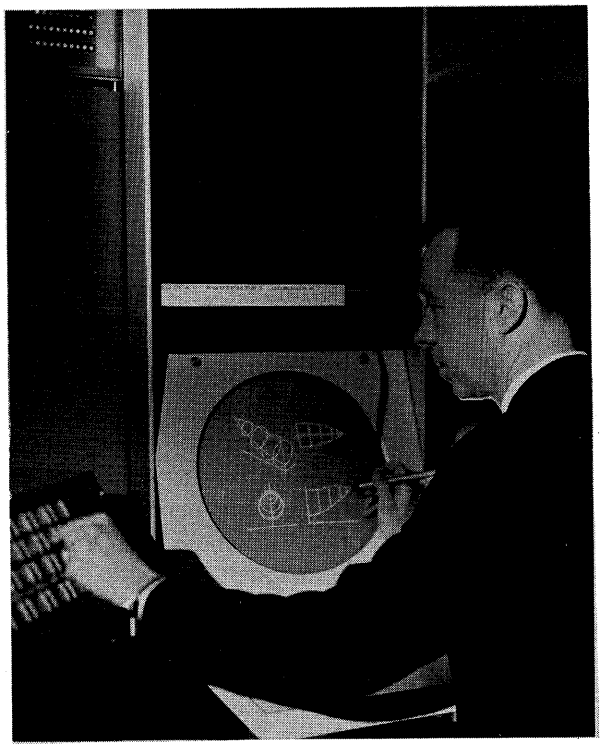


Figure 1. Design of an aft fuselage in 3-D.

disciplines including systems programming, mathematical analysis, electronics, information retrieval, design, numerical control and engineering loft. Under the Man-Computer Systems Program of Sys-

tems Sciences, there are about 20 team members in all. In addition, regular consultation is carried on in other specialty areas.

Since the UNIVAC 418 had never before been used for graphical operations, development of the systems programming was undertaken by our specialists. The development of "software" is quite involved for computer graphics. Many technical problems are associated with the creation of a drawing or sketch on a cathode ray tube by input from a light sensitive pen or from other sources. In order to perform analyses on or make changes to a display on a scope, it is necessary to program the computer such that the displayed configuration is, at all times, stored in mathematical form in computer memory. Provisions must be made for efficient storage and retrieval of displayed data and the protection of computer memory against careless destruction of vital data. In addition to the light pen input, data such as coordinates may be transferred to the display from input on a standard computer keyboard. Also, certain subroutine functions such as deriving the area, drawing a line, rotating, changing scale, and deleting an entity must be accomplished. These special subroutines are "called up" by the button box, a panel of 28 buttons designed by our engineers and integrated with the computer system. These buttons are under program control; that is, they may have different meanings for different applications programs.

Through the combination of the various input media and the general-purpose software system, it will be possible to develop special-purpose programs to solve particular problems. There is no intent that the presently developing software and general programs are to be the final system. Only those functions of obvious general utility are being incorporated. Then, as specific solutions are sought through the computer-graphic system, need for further extensions to the general package will become apparent. A feedback relationship between solutions to specific problems and extensions to the general program package seems the most reasonable doctrine in a research environment.

The capabilities that exist at the end of August 1965 on the graphic system include the following basic features:

*Three-Dimensional Capability*

- Four views: three principal projections and a perspective. Drawings are created by working in any combination of the three principal views
- Conversion to single view and return to four views upon request
- Isometric: drawing is created by locating X and Y first, then Z (available as a separate experimental program)
- Definition of points
- Definition of lines
- Definition of circular arcs (elliptic arcs in projection)
- Change of scale
- Rotation about any designated axis perpendicular to any view
- Translation
- Deletion of any element
- Multiple figures can be displayed simultaneously. Proper definition of figures permits alteration of parts of the complete drawing while other parts are held fixed
- Multiple rotation axes: this permits the study of relative motion. Axes may be parallel or orthogonal to each other. Angular rotation rates may differ

*Two-Dimensional Capability*

The two-dimensional capability includes the definition of points, lines, circular arcs and standard manipulation features using both keyboard and light pen inputs. In addition, a host of special features are planned, or exist, for the numerical control milling application which is explained later. Examples are:

- Construct a circle tangent to a displayed circle and a displayed line with its radius input by keyboard
- Construct a circle tangent to two circles. The radius of the required circle is input by keyboard
- Construct a circle through three displayed points
- Move a point while maintaining connectivity

*Features in Other Programs*

- Shape or mold a stored geometric shape to fit a preconceived concept or specification
- Freehand sketching
- Alphanumeric display
- Alphanumeric printout of designated text

- Translation of displayed data to hard copy output on an X-Y plotter

A small tracking cross is used as the medium of communication between scope and computer. Its position as directed by the "light pen," shown in Fig. 1, and the activation of appropriate subroutines by the use of the button panel direct the creation or functioning of the indicated graphical features.

Some of the above features are illustrated in the following figures:

- Figure 1 shows the 4-view representation of the design of an aft fuselage
- Figure 2 illustrates drawing in isometric
- Figure 3 is a series of photographs to illustrate multiple axes of rotation
- Figure 4 shows the cross section of a wheel pod. It has been shaped from a starting circle
- Figure 5 shows how connectivity may be maintained when points are moved

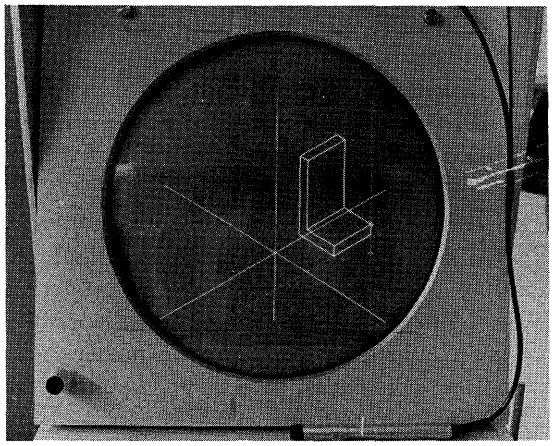


Figure 2. Design in isometric.

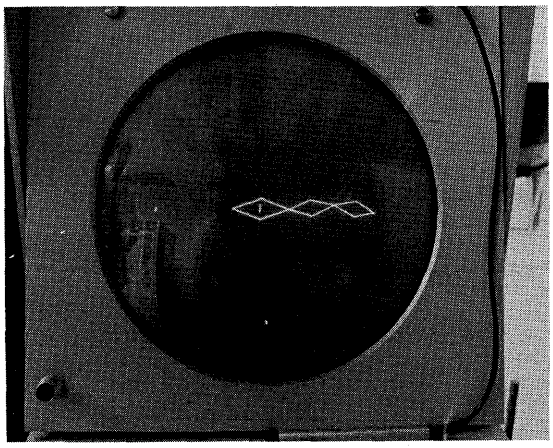
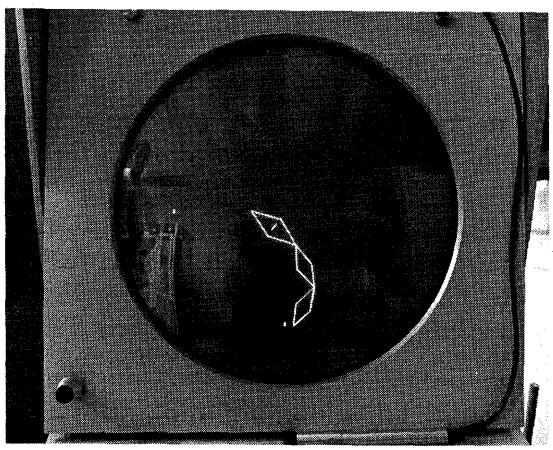
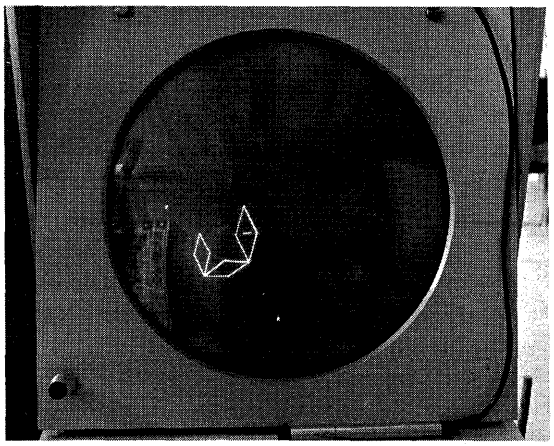
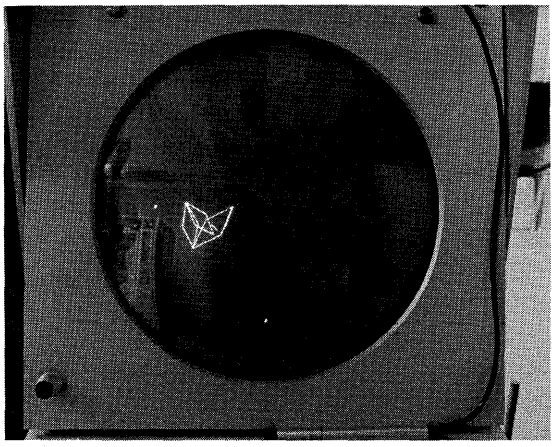


Figure 3. Multi-axis rotation.



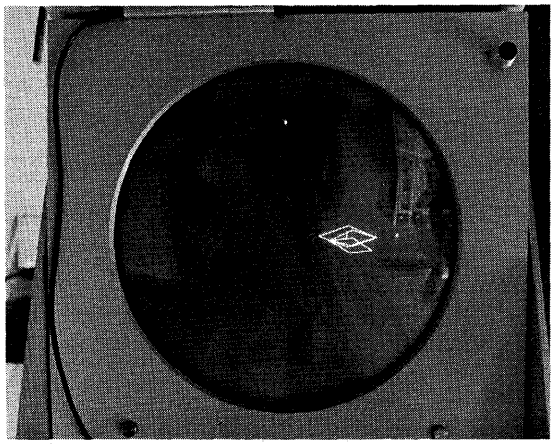


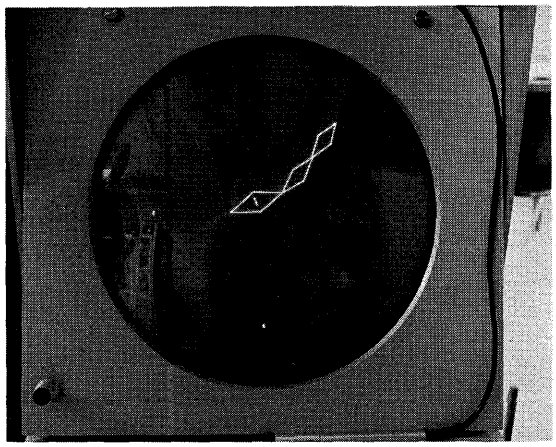


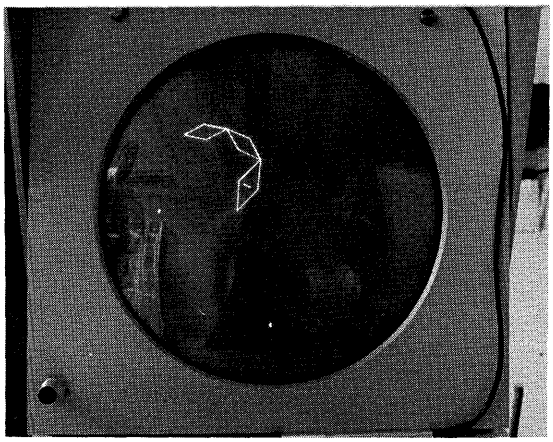
## PHILOSOPHY OF COMPUTER GRAPHICS RESEARCH

Before discussing our applications activity, I would like to discuss some aspects of the general national interest in man-computer graphics.

Figure 3. Multi-axis rotation.







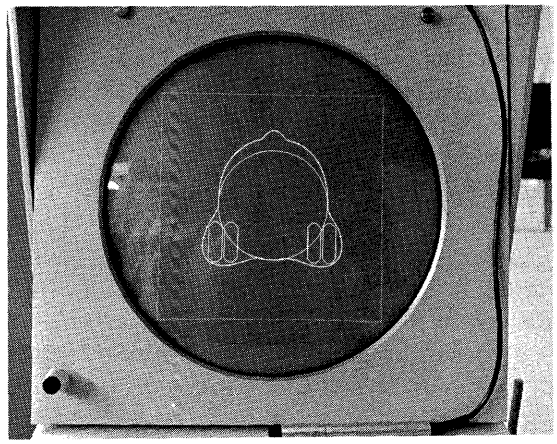
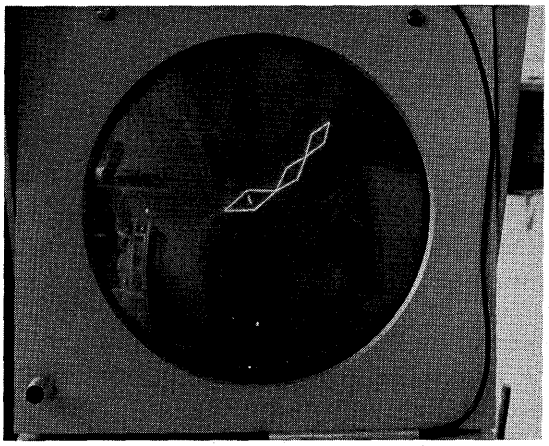


Figure 4. Wheel pod — shaped on the scope starting with a circle.



At the present time, various degrees of activity are springing up around the country. Some computer manufacturers, in recognition of the future role of computer graphics, have strong programs of their own. They hope to offer a complete system that will give newcomers to the use of computer graphics an up-to-date capability. Actually, it is unlikely that the manufacturers will be able to anticipate all of the problems and demands that will be forthcoming from this new area. Users of the packaged systems may find that their special applications will require additional computer programs that will be difficult to acquire. Though Lockheed-Georgia may use some of the manufacturer's software features when they become available, we believe that the creation of our own program system for our own applications offers the greatest flexibility and, therefore, the greatest success in long term operations.

Because of the tender age of the man-computer graphic concept, we believe that a modest initial effort with freedom to grow with experience is the most feasible course. There are many problems both of a technological and of a human engineering nature which must be dealt with in due time, and

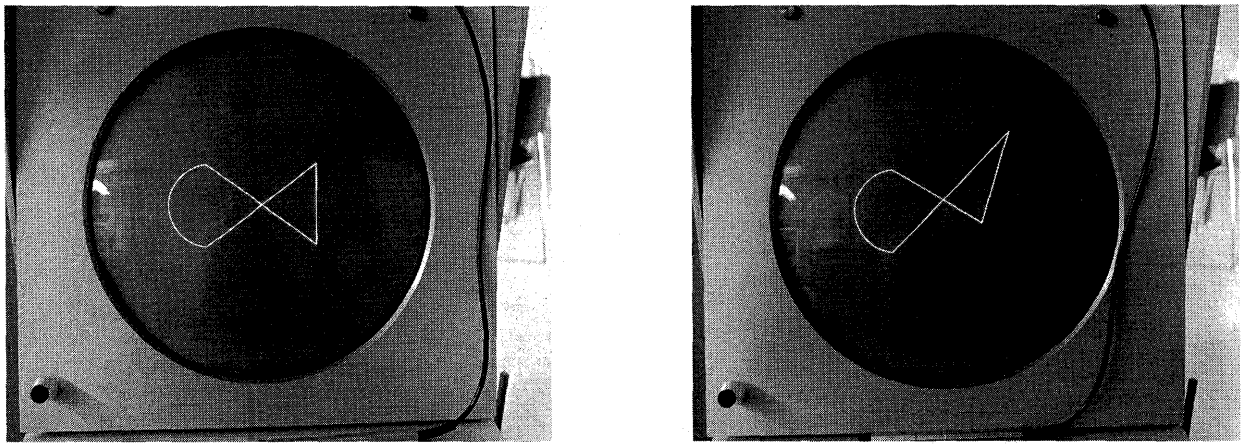


Figure 5. Connectivity may be maintained when end points are moved.

which will influence the growth pattern. Considerable experience and strong familiarization with the MCG concept and its ramifications will be necessary to make accurate judgments to distinguish the primary problems of concern from the secondary problems. In that regard there are many problems that require classification according to priority because any team of modest size can cope with only so many problems in a short time span. Because of this, we tend to prefer that the manufacturer deal with complex hardware problems while we furnish specifications that must be met to allow the system to function properly and to grow. Such problems as time sharing and remoting scopes from the C.P.U. must be given priority attention while the optimum tilt of the drawing surface, the nature of the drawing surfaces, and the number of buttons on the button panel are the type problems that can await attention without jeopardizing the main purposes and payoff of the man-computer graphic system.

## APPLICATIONS

Many long-range applications are contemplated for computer graphics, with current emphasis being placed on the design process. Structural analysis, management systems, information processing, electrical circuit layout, process control, and command and control are other areas that are receiving or will receive early attention. The first applications will be fairly specific and of limited technical complexity. In this connection, a "Near-Term" Group has been formed to seek technical contributions to Lockheed-Georgia which can be completed in 1965. This is in addition to the more long-range goals of the computer-graphics team. The "Near-Term" Group is investigating two areas: complete mathematical definition of all surfaces of an aircraft envelope and two-dimensional* numerical control milling procedures with completion dates of September and December 1965 respectively. The problems of mathematical definition of surfaces and of 2-D numerical control are described below.

### *Mathematical Definition of Surfaces*

In regard to the mathematical definition of surfaces, it should be noted that a large percentage of present designs are defined in mathematical terms. However, on some aircraft, there are regions where mathematical definition has not been practical. For example, on the C-141 the wing and fuselage surfaces are mathematically defined but the fillets between these surfaces were created more or less empirically. When complete mathematical definition is attained, the need for "master models" will be greatly reduced. Furthermore, the various activities that must utilize the shape will find that a central source of data in mathematical form will expedite their analyses and will eliminate cumulative errors which accrue when data is used, converted, and passed on to other activities in a serial fashion. Although this "Near-Term" task is not likely to utilize computer graphics per se, follow-on work will see surfaces displayed on the scope to study esthet-

*In the subsequent discussion on 2-D N/C, it should be borne in mind that we are actually working in a 3-D environment, but the 2-D nature of the basic problem should be clear from the context.

ics of design, to permit design alterations, and to perform functional evaluations. Various methods of surface definition have been investigated in depth in preparation for the follow-on work.

By defining surfaces on the display, it will be possible to vary unconstrained parameters and ascertain both the geometric and the analytical effects on the created surfaces. Computations such as surface area and volume will be performed. Parameters may be varied within allowable degrees of freedom to achieve "optimum" results with respect to design specifications.

*Two-Dimensional Numerical Control*

In our manufacturing process, many items are milled automatically by numerical tape-controlled milling machines. The creation of this tape is a laborious task. To produce the numerical control tape for a part or tool, an accurate drawing of the item must first be produced. Then a part programmer must painstakingly go through the drawing and define the points, each distinct line, curves, and other significant features. A series of computer instructions is then written to represent the path that the cutting tool must follow to mill the item according to the design specifications. The language and computer program for writing, for compiling, and for interpreting the instructions is called APT (Automatically Programmed Tools). The resultant output is a magnetic tape. With additional post processing, a paper tape for directing a particular milling machine is created. The APT system has been developing for many years, and the task of producing it has been a formidable one.

Still, the various steps leading to the production of a numerical control tape require many man hours. It was recognized that the application of new techniques in automation could significantly reduce the manual effort. It is estimated that about 80 percent of the items produced by means of numerical control at Lockheed-Georgia are of a two-dimensional line and circle geometry. Therefore, the desirability of using the early computer-graphic capability to assist the 2-D N/C problem became apparent. First the item would be defined directly on the scope using the various input media—light pen, buttons, keyboard, etc. The geometry of the item would be stored in computer memory on a permanent file. This geometry, which would be represented mathematically in the computer, would serve as the basis for the description of the cutter path.

Figure 6 shows a typical part that has been created on the scope. The two views show the circular top of the cutting tool at the beginning and near the finish of its path around the periphery of the part. Each line or circular arc of the figure is a surface since the part is viewed from the top.

To create the part, the part programmer must describe a path, composed of linear elements, for the cutting tool. In general, there is no way to determine the optimum path. The part programmer only knows that he must consider each of the detailed elements and their dimensions. The computer-graphics program to assist him is called PATH, and its operation is summarized in the sequel.

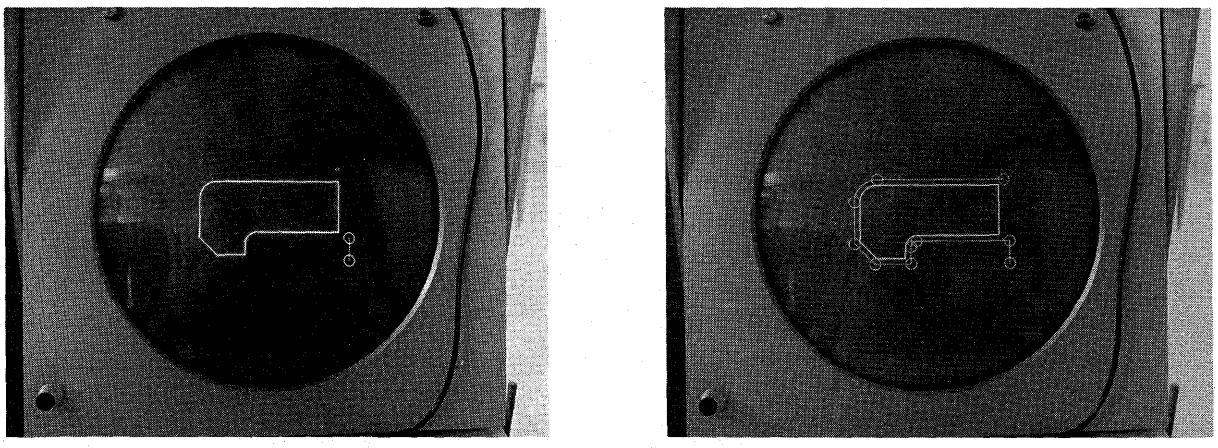


Figure 6. Top view of a typical part with cutter path de- scription.

When the part of concern has been displayed as in Fig. 6, the mathematical description will exist in the computer. Then the process of defining the cutter path can begin.

The initial position of the cutter is, in general, not on the part. The starting position in the X-Y plane is input as is the depth coordinate Z. The cutting tool will cut to the indicated depth and the Z value will be automatically associated with each (X, Y) of the succeeding path until a change in Z is requested and keyed in. The cutter radius must also be input at the beginning of a sequence of steps to enable the computer to automatically create appropriate offsets. Now, a point on a line (surface) of the part is designated as the first (X,Y) to which the cutting tool will move. Before the movement, however, it must be ascertained to which side of the line the cutting tool will be tangent or if the center of the cutter is to be positioned directly on the line. This is accomplished by moving the tracking cross to one of three approximate positions with respect to the line. That is, the cross is moved by the "light pen" distinctly to one side, "near" to, or distinctly on the other side of the line of interest. When the appropriate button is pushed, the circle representing the diameter of the cutter will be automatically positioned on the display and the center coordinates will be automatically recorded. The next step in the process of cutting out a part profile is to indicate the second surface of interest. Tangency or direct centering must be established as before. The cutter center will then move on the display and in the computer to the (X,Y) location of the intersection of the two surfaces. Where tangency is sought, appropriate offsets will be automatically allowed. Cutter path definition for succeeding surfaces is derived similiarly.

In many cases, an automatic mode for defining many steps of the cutter path may be employed. With respect to the geometry, the path will be continuously defined in this mode until the operator sees an error and stops the process or until the cutter arrives at a position where the next move is insufficiently defined. In the former case, the console operator may delete the erroneous portion of the path and correct it by stepping through the questionable region as previously described. In the latter case, the operator may resolve the ambiguity which the computer had indicated and restart the automatic process or continue step by step. In all cases the cutter path for several preceding steps is displayed for the perusal of the console operator.

When the outer profile has been described, the operator may choose to change Z and move to another section of the part for profiling or for swathing a flat top. When the cutting tool is placed tangent to a circle and when it is necessary to circumnavigate the circle, it is required that the operator key in the allowable error tolerance. The successive coordinates are then defined automatically by the computer such that the cutter may move linearly from point to point around the circle without exceeding the indicated tolerances.

When the path has been derived, it can be redisplayed step by step or continuously for operator approval. From an operator's point of view, the system is reasonably simple because there are relatively few operations of buttons, keyboard, and tracking cross which need to be learned.

With successive coordinates thus described, an output tape can be generated that describes the necessary cutter motion and completely bypasses the APT system requirement.

It is estimated that the average part programming time for approximately 1500 parts amenable to this graphic representation was 60 hours per part for the C-141. The estimate for the same requirement using the computer-graphic PATH program is 10 hours. Considerable savings will also be manifested in tooling and template manufacture. In addition to the great reduction in manhours, the decrease in span time will be even greater because the graphical system should inherently reduce program errors and because batch processing is replaced by real-time operation.

As of this writing, an evaluation of competitive bids for a three-scope system is being completed. The system will be dedicated to the single but important numerical control application. Delivery is expected early in 1966. The system will be separate from the continuing computer-graphic Research facility.

## MAN-COMPUTER GRAPHICS IMPLICATIONS

The specific areas of application which have been alluded to are relatively simple in many respects. Until computer graphics reaches a somewhat more advanced level of sophistication, even the seemingly simple problems offer a considerable

challenge. It is one thing to solve specific problems but quite another thing to solve large classes of problems and to integrate newly developed problem solutions into systems of somewhat greater breadth. An optimum solution to a particular problem may not be optimum when a solution to a more all-encompassing problem is sought. Thus the successful introduction of computer graphics into our technological "bag of tricks" will necessitate our investigation of new methods and techniques for computer graphics. For example, current design practices require a sequence of relatively autonomous operations. The layout is created. Drawings are distributed to the various specialty areas. Information is extracted and forwarded to the central computer facility for the analyses that characterize each specialty area. Results of the computer runs are interpreted and, perhaps, design changes are requested. Among the many conflicting design requirements, compromises must be made and the cycle from designer to specialists to computer and back again is repeated.

With computer-aided design, the team concept may be altered considerably. Specialists may actually take part in the early design process. Some of their evaluations might be accomplished directly as the design develops and this may cause almost immediate modifications before extensive time is lost by the creation of unacceptable designs. As another approach, the various specialty areas may have their own display systems which are linked to a common central computer. Design specifications and alterations may be perceived at each station when information is called up. This capability will compress the time expended for the entire design process.

The types of future computer-display systems that will best suit Lockheed-Georgia or any other company will depend on many factors, including the number of departments which will utilize the system, the interdependence of the encompassed activities and, of considerable importance, the direction of growth in computer facilities which will be made available by computer manufacturers.

## SELECTION OF AN "OPTIMUM" SYSTEM

The evaluation of equipment for our follow-on man-computer research facility is now in process. This evaluation involves many subjective decisions which are characteristic of the selection of any major computer system. Timing is most important because a good system for today may lack flexibility to grow into tomorrow's requirements. Once we are committed to a computer system and the development of the associated software, it takes considerable time to justify and implement a change. It is therefore incumbent upon us to plan for the solution of tomorrow's problems though they are not totally defined today.

The computer-graphics research facility at Lockheed-Georgia has provided a rare opportunity for gaining insight and experience in the vast area of man-computer systems. Familiarization with the problems and the general capability of computer-graphics will equip our personnel with the background and training to adjust quickly to new and uncharted areas. Indeed, analogous to the portion of an iceberg that lies beneath the water's surface, there are many more as yet undiscerned applications which lie just beneath the surface of current comprehension.

## ACKNOWLEDGEMENTS

I would like to express my appreciation for the encouragement and support that the Man-Computer Graphics Program has received from many people in the Engineering, Computing, and Manufacturing Branches of the Lockheed-Georgia Company.

The basic team has been staffed with personnel from each of these branches. Their enthusiasm and competence have been a major factor in our continued progress. In particular, I would like to acknowledge the important contributions of O. V. Hefner, the lead programmer for the software development.

A special note of thanks is extended to Mr. M. D. Prince, Associate Director of Research-Systems Sciences, who initiated the program, and who has maintained an active role in charting its course.

## REFERENCES

1. R. W. Mann and S. A. Coons, "Computer-Aided Design," *McGraw-Hill Yearbook of Science and Technology,* 1965, pp. 1-9.

2. *Lockheed Georgia Quarterly,* Summer 1965, vol. 2, no. 2, Lockheed Georgia Company.

3. S. H. Chasen, "APT-less Contouring Tapes?" *American Machinist,* pp. 69-70, (July 5, 1965).

4. E. L. Jacks, "A Laboratory for the Study of Graphical Man-Machine Communication," *Proceedings — Fall Joint Computer Conference,* Spartan Books, Inc., Washington, D.C., 1964, pp. 343-350.

5. I. E. Sutherland, "Sketchpad: A Man-Machine Graphical Communication System," Report No. 296, Lincoln Laboratory M.I.T., (30 January 1963).

6. J. C. R. Licklider and W. E. Clark, "On-Line Man-Computer Communication," *Proceedings—Spring Joint Computer Conference,* 1962, Spartan Books, Inc., Washington, D.C., pp. 113-128.

# HYBRID COMPUTING FOR LUNAR EXCURSION MODULE STUDIES

Arthur Burns
*Analog Computing Section*
*Grumman Aircraft Engineering Corporation*
*Bethpage, New York*

## INTRODUCTION

Hybrid computation plays an important role in our man-in-space effort. Large scale combined analog-digital studies in support of NASA's Project Apollo are now being performed. The computers utilized are an IBM 7094-II digital computer and three consoles of Reeves 500 analog computing equipment. These are linked by an Adage 770 Computer Link that provides an analog-to-digital and digital-to-analog capacity of 48 and 55 channels, respectively.

The effort described herein concerns one of the latest applications of a computing technique that has become a "semicontroversial" topic in recent years. Reluctance to use both computing systems concurrently in the solution of one problem has stemmed from their inherent incompatibility—the continuous versus the discrete domain. As a result, most analog and digital computer installations have been physically and philosophically isolated from one another. Professional computer organizations also have been slanted toward one or the other type of machine.

The arrival of the space age has changed this picture considerably. Recently, as evidenced by the large number of hybrid facilities being developed around the country, there appears to be an awakening to the fact that modern day problems require the advantages of both computers. Efforts have been made to organize computing teams for hybrid projects. It is apparent that a softening of the old barriers has occurred. In addition, an important by-product has been produced in the form of "cross education" between the analog and digital programmers. By necessity, each is beginning to appreciate the advantages and disadvantages of both machines.

At Grumman Aircraft Engineering Corporation, preliminary investigations into the field of combined analog-digital computation began late in 1957. The results of these studies led to the acquisition of a small, flexible linkage system to interconnect an IBM 704 digital computer with a large analog computer installation. The initial application of this system was a verification of the theoretical studies that had revealed that, for certain types of problems, a hybrid technique had distinct advantages over all-analog and all-digital simulations.[1] When the IBM 704 was replaced by the IBM 7090, the linkage system was modified accordingly in order to be compatible with and to utilize the best features of this new digital computer.

The next few years (through 1963) saw no major changes in the linkage system as the IBM 7090

evolved into the IBM 7094-II. During this period, relatively small-scale problems were solved using the hybrid technique. These included some simulation work on the stabilization and control system of the Orbiting Astronomical Observatory,[2] a system identification study,[3] and a missile homing dynamics problem.[4]

Active work on the development of the LEM (Lunar Excursion Module) commenced at Grumman in January 1963. Although the initial computer studies were of the all-analog and all-digital variety, it was immediately clear that for some of the more complicated programs, then in the planning stage, an elaborate hybrid computing complex would be required. Accordingly, a decision was made to expand the existing linkage equipment to accommodate these large problems. This expansion ultimately proved unfeasible and a completely new, large scale system was ordered and delivered early in 1964.

This system is now being used in the LEM hybrid studies. The following discussion describes the reasons for the hybrid approach, the equipment utilized, the problem that is solved, and the implementation of the hybrid computing technique.

## DISCUSSION

### *Why the Hybrid Approach?*

The choice of the type of computer complex to be used in simulation studies is becoming increasingly difficult. This is due largely to:

- The continually increasing scope and complexity of simulation programs,
- The rising popularity of digital, real-time simulation techniques, and
- The reluctance of many engineers to depart from conventional all-analog and all-digital methods.

As part of the LEM project some computer studies are planned that combine many subsystems into an integrated simulation. The result will be a considerably detailed representation of the LEM vehicle. Add to this already ambitious undertaking the facility for including a pilot, external visual displays, and actual flight hardware, and the result is a problem that exceeds the state of the art of all-analog and all-digital techniques.

Even the high speed digital computers in use today are not fast enough for an all-digital real time solution. Sampled data studies indicate a maximum allowable computation interval that, when exceeded, produces instability and prohibitive inaccuracies. For the large scale studies, the interval required by an IBM 7094-II to solve the appropriate equations exceeds this limit. The only alternative for an all-digital solution is thus a relaxing of the complexity of the mathematical models. This can be a painful process. Rather than compromise the aims of the program, a more desirable solution is to abandon the all-digital idea and investigate other computing approaches.

The problems involved in an all-analog solution are of staggering proportions. Equipment requirements quickly exceed a feasible amount. Resolution requirements on some trajectory parameters are on the order of 0.001 percent. This is impossible to obtain without resorting to multiple amplitude scaling techniques. For real-time operation, these scale changes would have to be made automatically, resulting in an even larger equipment load. Finally, a huge amount of logic and memory is necessitated (particularly by the various guidance laws). It is generally well known that these operations are very unwieldy on the analog computer.

Having eliminated these approaches, the logical choice is to "go hybrid." Although the utilization of the hybrid technique does present some unique problems, these are mostly of a logistical nature (computer scheduling, coordination, etc.). The present state of the art of computer linkage hardware is such that analog and digital equipment may be confidently interconnected. The resulting computing system is one that, unlike the all-digital and all-analog techniques, can result in a large scale simulation study that meets the required objectives.

### *The LEM Mission*

The Lunar Excursion Module is the vehicle in which, as part of Project Apollo, two astronauts will land on the moon. The mission starts with the LEM detaching from the orbiting Command Service Module (CSM) and inserting itself into a coasting elliptical orbit (Fig. 1a). At a point in the vicinity of the pericynthion, the powered descent portion begins. This is accomplished by firing the throttleable descent engine. Execution of the proper

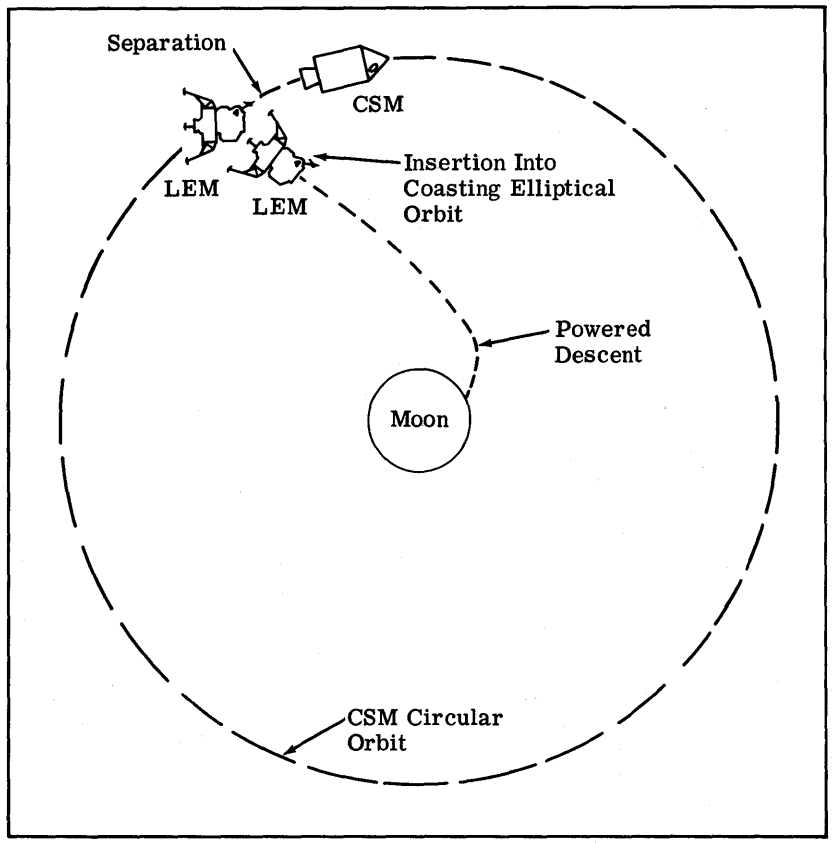



Figure 1a. LEM mission (descent).

attitude profile by the LEM causes sufficient deceleration for the landing maneuver.

After an exploration period, and with the CSM in the proper orbit position, the ascent engine is ignited, using the descent stage as a launching pad. The LEM is launched from the lunar surface and then performs the rendezvous and docking maneuvers with the CSM (Fig. 1b).

At any time during the descent the mission may be aborted (Fig. 1c) and the LEM will perform an "early rendezvous" with the CSM.

A reaction jet control system, including jet select logic and pulse ratio modulators, is used for rotational and translational control. Additional attitude control is provided during powered descent by descent engine gimballing. Guidance is implemented via on-board computer, manual inputs, or combinations thereof.

*Simulation Facilities*

The simulation complex shown in Fig. 2 can be thought of as organized into three general categories:

1. The hardware necessary to insert a man in the loop including an instrumented LEM cockpit, external visual displays that present appropriate visual cues to the pilot, and control consoles for monitoring the runs, failure insertions, etc.
2. The computer facility consisting of an IBM 7094-II digital computer, a Reeves 500 analog computer, and special purpose computing hardware, and
3. An Adage 770 analog-digital computer linkage system.

Although the features of general-purpose analog and digital computers are well known, computer linkage systems are still fairly unique. The Adage 770[5-7] is composed of two ADC's (analog-to-digital converters), two 24-channel multiplexers, 35 sample and hold amplifiers, and 55 DAC's (digital-to-analog converters). Three channels of fixed discrete

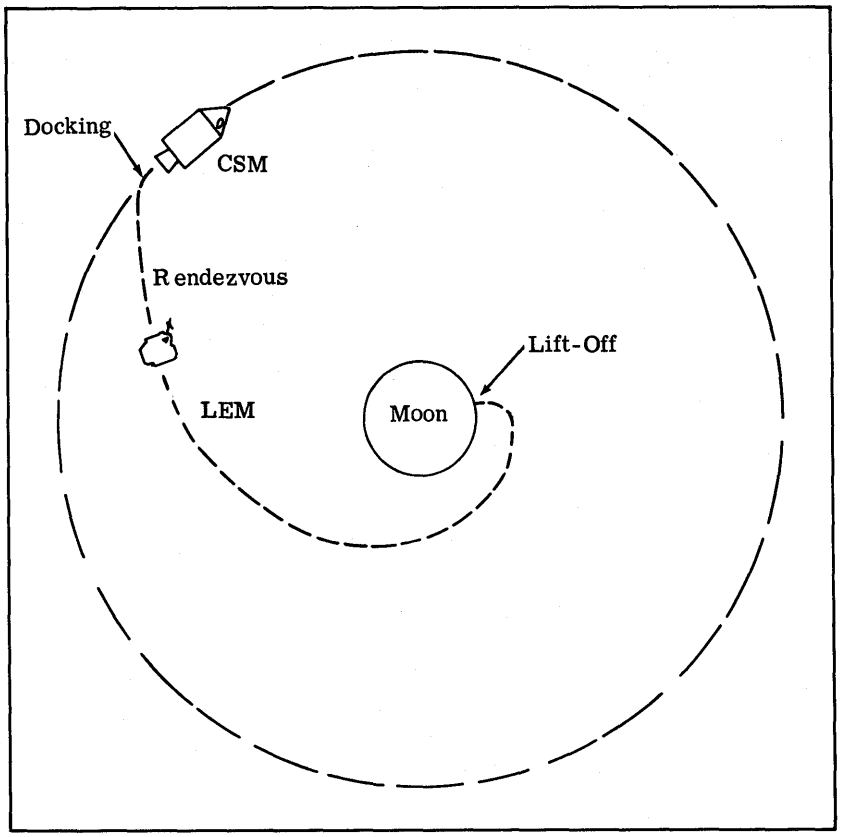



Figure 1b. LEM mission (ascent).

data information (switch positions), each consisting of a 14-bit digital word, may also be transmitted to the digital computer.

Utilization of the mode control feature enables the digital computer to control the OPERATE, REST, BALANCE CHECK, and HOLD modes of the analog computer. However, the analog consoles, recorders, control console, and the digital computer are "mode slaved" in such a way that the lowest commanded mode always dominates. Regardless of what mode it has commanded, the digital computer can sense the actual mode of the analog.

Because the digital computer can write initial conditions into the analog integrators and then throw them into HOLD, it may now use the same DAC for any other problem input during the OPERATE portion of the run. Thus, the output of a "time-shared" DAC may be patched to the initial condition input of an integrator as well as any other input (except another integrator initial condition).

Digital input/output sequences are initiated by appropriate READ and WRITE commands from the IBM 7094-II. The A-to-D, D-to-A, and discrete data channels may be selected at random.

The outputs of four flip-flops whose states are controlled by the digital computer are made available at the analog patch panel. These are called Function Outputs and can be used to drive relays on the analog computer.

A prominent feature of the Adage 770 is the control panel shown in Fig. 3. This unit contains the switches and indicators necessary to operate and test the linkage equipment. In the manual mode, with the aid of the control panel, all instructions to the Link may be entered and all data and control registers examined. This enables a major portion of checkout and trouble-shooting procedures to be accomplished without using the digital computer.

*Assignment of Computer Tasks*

Allocating portions of the problem between the

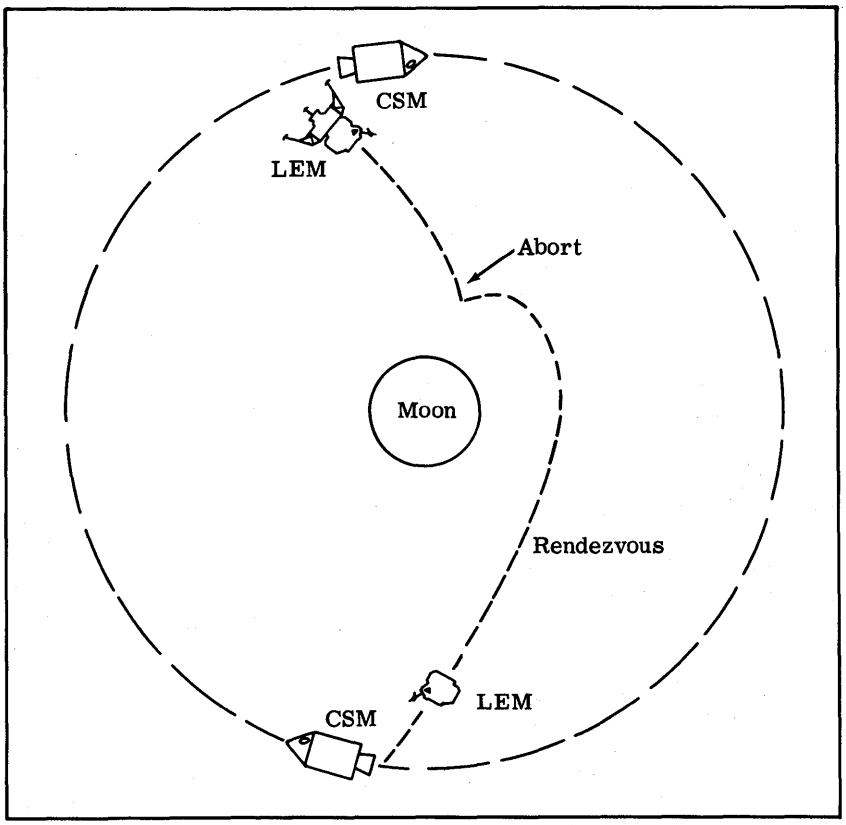



Figure 1c. LEM mission (typical abort).

analog and digital computer should not consist merely of having the analog relieve the digital of some of its computational burden. The primary consideration, rather, should be the utilization of the best features of each. Efficient computer usage is also an important factor. With as powerful (and expensive) a machine as the IBM 7094-II being utilized, it is imperative for computer efficiency that its computation interval be as close as possible to (but less than) the maximum allowable for real time. Once this is achieved, the requirement for a minimum amount of analog equipment should naturally follow.

The usual starting point is to assign the high frequency dynamic equations to the analog computer and those involving large dynamic ranges and considerable logic to the digital. For example, the LEM reaction control system modulators provide rapid pulses of thrust to the vehicle, resulting in relatively high-frequency attitude accelerations. Thus the control system, calculation of the body forces and moments, and resulting rotational dynamics are placd on the analog. In addition, descent engine gimballing, and reaction control system fuel computations are also placed on this computer. Translational and trajectory equations, calculation of variable mass and inertias, descent and ascent engine thrusts, and all axis transformations are placed on the IBM 7094-II. Appropriately, the on-board guidance computer is also simulated on this dital computer.

Reaction control system modulators, jet select logic, thruster shaping circuits, and gimballed engine logic are taken care of by special purpose computing hardware.

The resultant computer configuration is shown in Fig. 4.

### *Operation of Problem*

*Real Time:* A typical automatic mission run begins with the analog computer in BALANCE

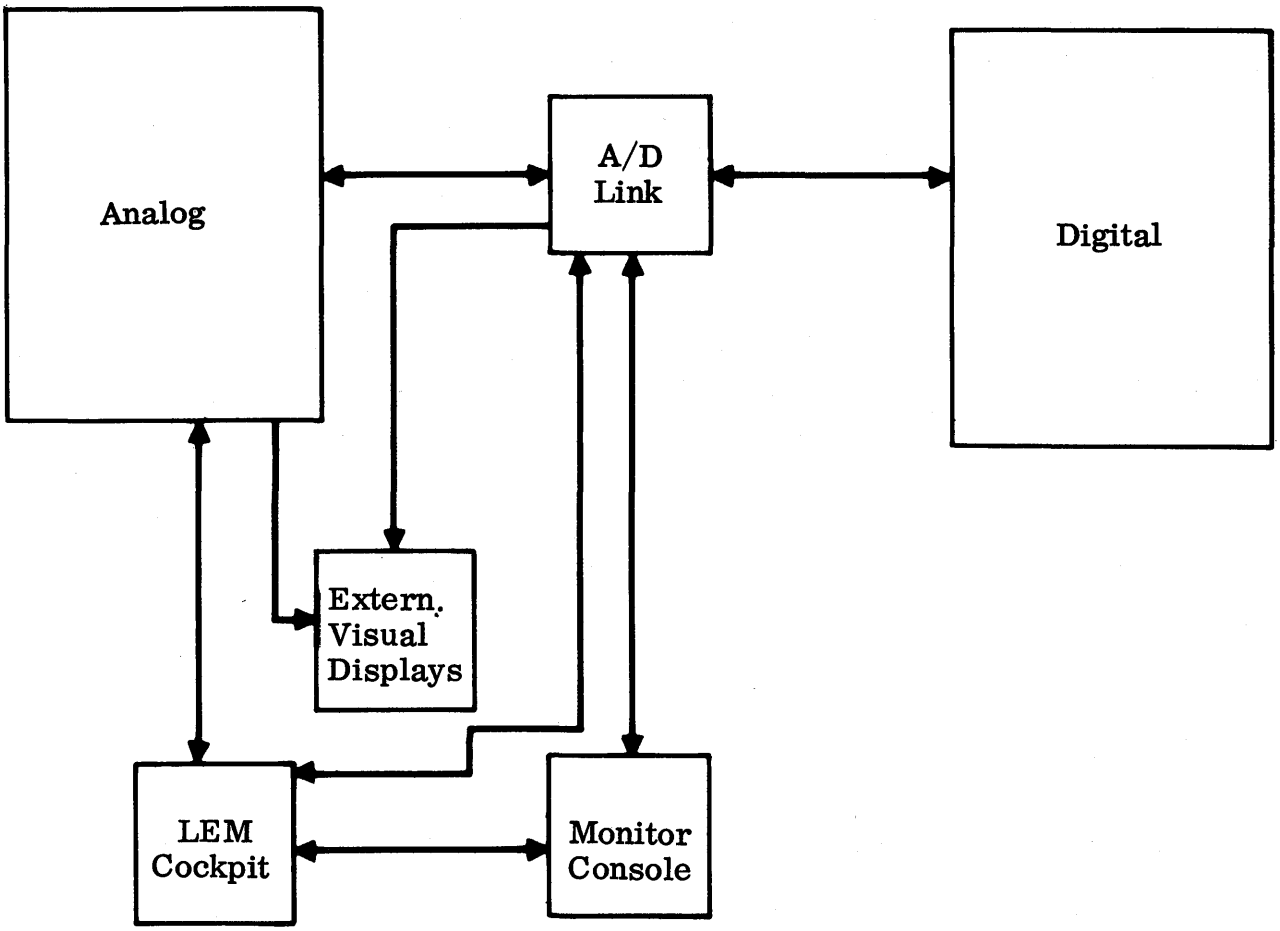



Figure 2. Simulation facilities.

CHECK (all amplifier inputs grounded). The digital computer then writes initial conditions into the appropriate DAC's and sends the analog into RESET. The analog integrators now have the appropriate initial condition output voltages. The analog computer is then sent into HOLD. If time sharing of DAC's between initial condition and variable quantities is used, the values of the latter at time = 0 are then transferred to the particular DAC's involved. The digital computer then causes the analog to go into OPERATE. The precision interval clock is simultaneously turned on, and the basic computation interval has started.

Body attitude rates generated on the analog are sampled by the digital and integrated to produce vehicle attitude. By applying the appropriate guidance law, attitude errors are generated. These are sent back via DAC's to the control system that produces corrective moments to the LEM. The guidance laws also determine when translational commands are required. These are also transmitted to the analog where the translational forces are generated. Because these forces contain high frequencies and cannot be sampled fast enough, they are integrated first and then sampled. The digital computer then takes the derivative using successive values. From the calculated average force, the resultant translational motion is calculated.

Descent or ascent engine thrust is calculated and sent to the analog for computing moments due to engine misalignments and center of gravity offsets. Updated values of moments of inertia and center of

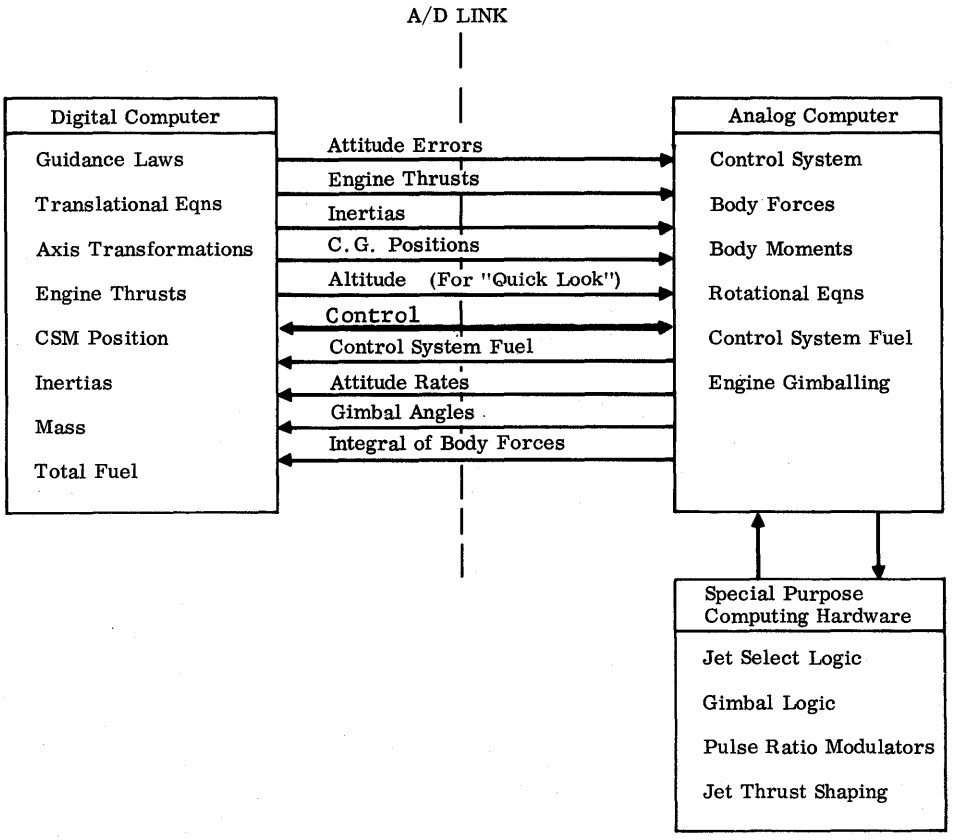



Figure 3. Adage 770 control panel and manual control instructions.

*Manual Controls*

POWER ON/OFF — Power to 770.
REMOTE MANUAL — Permits either manual or digital computer control of read/write operations in the 770.
READ SCAN UPPER LIMIT — Sets upper limit of sequential read scan.
WRITE SCAN UPPER LIMIT — Sets upper limit of sequential write scan.
REFERENCE VOLTAGE INTERNAL/EXTERNAL — Selects either internal or analog computer reference voltage.
MANUAL INPUT REGISTER — For entering manual data and instructions into the interface write buffer.
MANUAL WRITE — Enters the contents of the manual input register into the 770.
MANUAL INTERRUPT COMMAND — Generates a manual interrupt signal.
MANUAL READ — Manually duplicates a read request signal from the digital computer.

*Indicators*

READ ADDRESS — Displays the read channel address.

gravity positions from consumed values of reaction control system and engine fuels are transmitted to the analog at each computation interval. Altitude above the lunar surface is also sent to the analog for general monitoring purposes.

During the course of a run, the digital computer also selects control system deadband values and feedback gains. This is accomplished by the 7094-II setting the proper Function Output flip-flops. Function relays on the analog computer are thus energized and the appropriate analog circuitry switched in.

*Fast Time:* Although the studies are primarily run in real time, certain coasting phases of the mission may be accomplished in fast time. For this mode the analog computer is sent into RESET. The LEM is then assumed to remain at a constant attitude, and the digital computer goes through its computations as fast as it can using larger iteration intervals. The problem is returned to real time by new initial conditions being written by the digital computer and the analog sent once more into OPERATE.

*Future Studies*

At the time of this writing, the LEM hybrid study has been made assuming fully automatic guidance. The capability for both manual and automatic control should be available soon. The simulation will then contain, in addition to the hybrid computer complex, a fixed base, instrumented LEM cockpit, a control and monitor console, and external

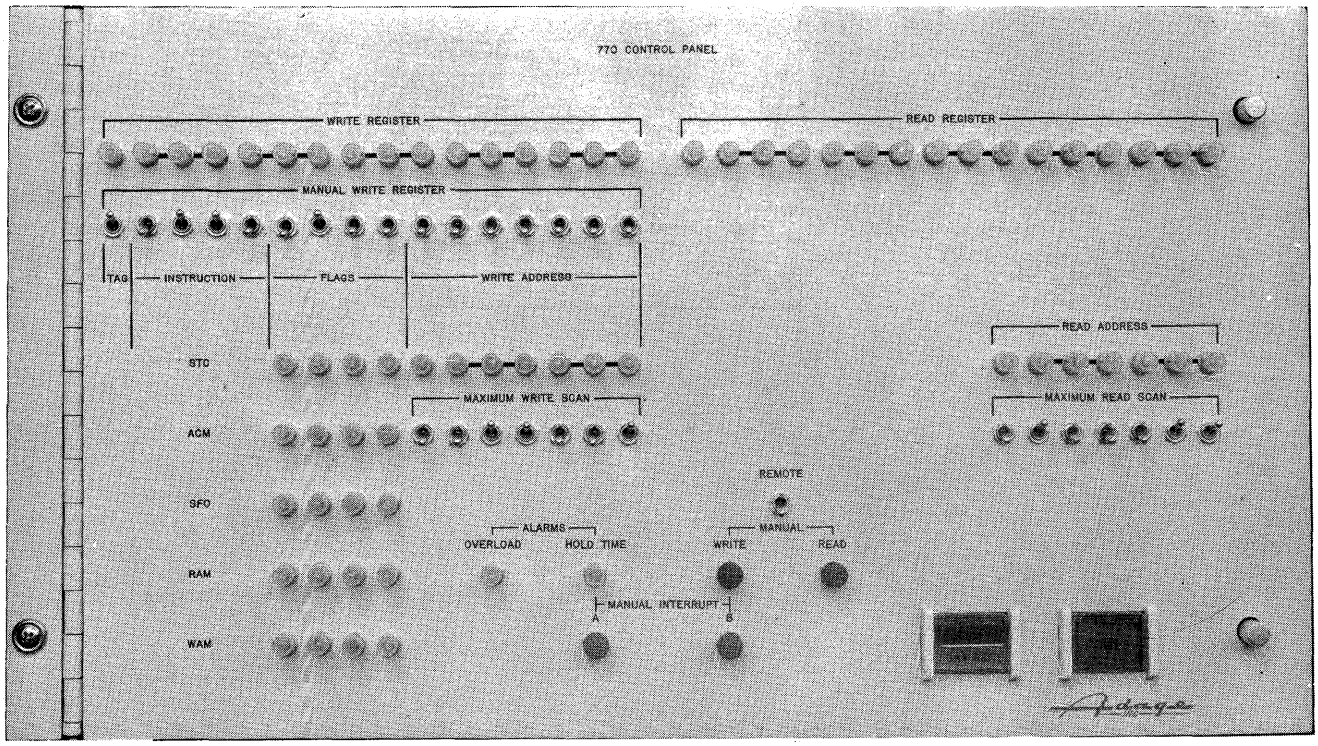



Figure 4. Allocation of computer tasks.

visual displays. The displays will present a view of the lunar surface, CSM, and stars as would be observed by the crew members during a major portion of the mission.

With the introduction of this additional simulation hardware, computer interface requirements become much more severe. Not only are more DAC's and ADC's required but some digital-to-digital type converters as well. These are necessary since the digital computer must now read both analog (throttle and controller) signals and digital (switch positions) information as well as drive both analog and digital displays in the cockpit.

## CONCLUSIONS

The preceding has described an application of a hybrid computing technique that is now making contributions to the Apollo project. It has been shown that, for the large scale LEM computer studies, this is the only feasible method to use. The stringent speed and resolution requirements, and the complexity of the problem eliminate the all-analog and all-digital approaches.

The combined analog-digital system, on the other hand, provides the engineer with a computer complex that:

- contains a considerable amount of logic and memory,

- provides high resolution where needed, and
- is capable of real-time and fast-time operation.

Discussions of combined analog-digital techniques usually end with a debate on the probability of the digital computer replacing the analog. Many believe that hybrid computation is only a part of a transitional period and that due to the rapidly increasing speed and input/output flexibility of digital machines, a decade from now they will be performing most if not all simulation-type problems.

To those companies that in the past few years have expanded their analog and digital facilities and find *both* being used to full capacity, these prophecies are only of academic interest at present. This also is true for the personnel involved in landing men on the moon in the beginning of the next decade. They know that many times before 1970, with the aid of computers, they must simulate this feat. They are concerned with the 1965, not 1975, state of the art.

The hybrid technique described here is being used today and similar techniques will be used in the next several years. For large scale simulation studies such as are being made for the LEM—for problems of such high order of complexity—the hybrid approach is the only computing method by which the required objectives may be attained.

## ACKNOWLEDGMENTS

Regarding the implementation of the hybrid computing technique described here, the author has merely reported the work of many people. He would especially like to acknowledge the significant contributions made by R. Phagan and H. Ahders of LEM Dynamic Analysis, and J. Sachleben, G. Connelly, R. Alleva, A. Mackenzie, and J. Casey of Computing Sciences.

## REFERENCES

1. A. J. Burns and R. E. Kopp, "Combined Analog-Digital Simulation," presented at the 1961 Eastern Joint Computer Conference, Washington, D. C. (Dec. 12-14, 1961).

2. G. Zetkov and R. Fleisig, "Dynamic Analyses of OAO Spacecraft Motion by Analog-Digital Simulation," presented at the Space Electronics Session of the 1962 IRE International Convention, New York City, (March 29, 1962).

3. R. E. Kopp and R. J. Orford, "Linear Regression Applied to System Identification for Adaptive Control Systems," presented at the 17th Annual Meeting and Space Flight Exposition, Pan Pacific Auditorium, Los Angeles, California (Nov. 13-18, 1962).

4. W. Valckenaere, R. Helm and H. Ahders, "Dynamics of Homing Guidance," Grumman Aircraft Engineering Corporation Report ADR 06-05-63.1 (March 1963).

5. Adage, Incorporated, *Reference Manual for the Adage 770 Hybrid Computer Linkage System,* (Jan. 1964).

6. J. H. Sachleben, "Why Hybrid Computing?," Grumman Aircraft Engineering Corporation Research Department Computing Report CR 65-2 (Feb. 1965).

7. G. Connelly and F. Romani, "A Report on the ADAGE 770 Computer Link and Operating Procedures Applicable to Analog Computation," Grumman Aircraft Engineering Corporation Research Department Computing Report CR 65-3 (March 1965).

# OPTIMUM DESIGN AND ERROR ANALYSIS OF DIGITAL INTEGRATORS FOR DISCRETE SYSTEM SIMULATION

Andrew P. Sage
*University of Florida*
*Gainesville, Florida*
and
Roger W. Burt
*Motorola, Inc.*
*Phoenix, Arizona*

## INTRODUCTION

In digital differential analyzers and digital computers, simulation is carried out by some form of numerical integration or of replacing a difference differential equation by a difference equation. This paper is concerned with the development of optimum numerical integration and digital simulation techniques and a discussion of the accuracy of these methods when compared with ideal integration.

## CLASSICAL APPROACH TO DEVELOPING INTEGRATION RULES

A given function may be approximated by some polynomial over a short interval, $t$, and then the polynomial integrated rather than the original function. Newton's formula for representation of the function from $t_0 \leq t \leq t_0 + nT$ is the polynomial[1,8]

$$P(t) = x_0 + u\Delta x_0 + \frac{u(u-1)}{2!}\Delta^2 x_0 + \frac{u(u-1)(u-2)}{3!}\Delta^3 x_0 + \frac{u(u-)(u-2)(u-3)}{4!}\Delta^4 x_0 \ldots (1)$$

Where $x(t)$ is the function, $P(t)$ the polynomial, $T$ is the sampling rate, and

$$u = \frac{t-t_0}{T}$$

$$t = t_0 + Tu$$

$$\Delta x_0 = x_1 - x_0$$

$$\Delta^2 x_0 = \Delta x_1 - \Delta x_0 = x_2 - 2x_1 + x_0$$

and where $x_0, x_1, x_2, \ldots x_n$ are the values of $x(t)$ at $t_0, t_0 + T, t_0 + 2T, \ldots t_0 + nT$. The value of the integral

$$y(t) = \int_{t_0}^{t_0 + nT} x(t)\, dt \quad . \; . \quad (2)$$

is then approximately

$$\int_{t_0}^{t_0 + nT} x(T)\, dt \approx \int_{t_0}^{t_0 + nT} P(t)dt \quad . \; . \quad (3)$$

If Eq. (1) is substituted and integrated term by term the result is

$$\int_{t_0}^{t_0+nT} X(t)\,dt \approx T\left[nx_0 + \frac{n^2\Delta x_0}{2} + \left(\frac{n^3}{3} - \frac{n^2}{2}\right)\frac{\Delta^2 x_0}{2!} + \left(\frac{n^4}{4} - n^3 + n^2\right)\frac{\Delta^3 x_0}{3!}\right] \quad (4)$$

Various classical integration rules are obtained from Eq. (4). The trapezoidal rule results when $n = 1$ and Simpson's rule is obtained when $n = 2$.

As $n$ is increased in Eq. (4), the approximation to the true value of the integral usually becomes better. The error involved in this approximation is known as truncation error and will be the only type of error considered in detail in this paper. Another source of error in digital integrators is known as round-off or quantization error which has an approximate root mean square value of $q/\sqrt{12}$ where $q$ is the quantization level.

A complete discussion of DDA theory, operations, mechanization, and programming is the presentation by Gschwind.[3] Truncation errors are discussed in references 3-6 while round-off errors are discussed in references 2, 3, 5, and 6. The material to be presented here applies equally to digital differential analyzers of the serial or parallel type as well as conventional digital computers.

## DEVELOPMENT OF OPTIMUM NUMERICAL INTEGRATION AND SIMULATION RULES

The theoretical design problem for optimum numerical integration and transfer function simulation rules may be visualized by referring to Fig. 1.

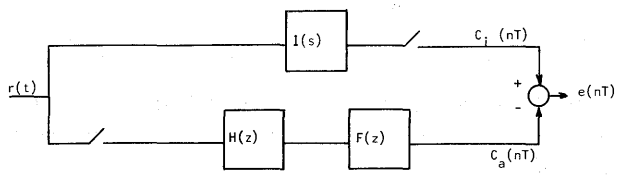



Figure 1. System for optimization problem.

In Fig. 1, $r(t)$ is the input, $I(s)$ is the ideal operation, $H(z)$ is the desired transfer function, and $F(z)$ is the fixed portion of the system. The error sequence of the system is

$$e(nT) = c_i(nT) - c_a(nT)$$

where $c_i(nT)$ is the ideal output after sampling, and $c_a(nT)$ is the actual sampled output. After substitution the error sequence in $z$ transform notation is

$$E(z) = [R(s)I(s)]^* - R(z)F(z)H(z) \quad (5)$$

where $[R(s)I(s)]^*$ is the $z$ transform of $R(s)I(s)$. The sum of error squared may be written

$$\sum_{n=0}^{\infty} e(nT)^2 = \frac{1}{2\pi j}\int_{\Gamma} E(z)E(z^{-1})z^{-1}dz \quad (6)$$

where the contour of integration is the unit circle.[1] Substitution of Eq. (5) into this expression yields

$$\sum_{n=0}^{\infty} e(nT)^2 = \frac{1}{2\pi j}\int_{\Gamma} [A(z)-R(z)F(z)H(z)]\,[A(z^{-1})-R(z^{-1})F(z^{-1})H(z^{-1})]\,z^{-1}dz \quad (7)$$

where $A(z) = [R(s)I(s)]^*$.

$H(z)$ is to be determined to reduce the sum of error squared given by Eq. (7) to a minimum. This problem may be solved by applying the calculus of variations,[7,9] which yields the result

$$H_0(z) = \left\{\frac{R(z^{-1})F(z^{-1})A(z)}{[R(z)R(z^{-1})F(z)F(z^{-1})]_-}\right\} P.R. \quad (8)$$
$$\overline{[R(z)R(z^{-1})F(z)F(z^{-1})]_+}$$

where the symbol { } $P.R.$ refers to the physically realizable portion of the term within { }, and the + and − subsubscripts refer to the conventional spectrum factorization operator denoting extraction of the multiplicative term containing poles and zeros inside (+) or outside (−) the unit circle.

## DISCUSSION OF THE FIXED PORTION OF THE SYSTEM

If a solution of the equation

$$\frac{dy}{dt} = g\,[x(t) - y] \quad (9)$$

is attempted by a numerical integration process such as the standard trapezoidal rule.

$$y(nT) = y(n-1)T + \frac{T}{2}[y'\,(n-1)T + y'\,(nT)] \quad (10)$$

it is seen that the simulation of the system described by Eq. (9) and illustrated in block diagram form in Fig. 2a is unrealizable since $y'(nT)$ cannot be obtained until $y(nT)$ is known. In order to implement

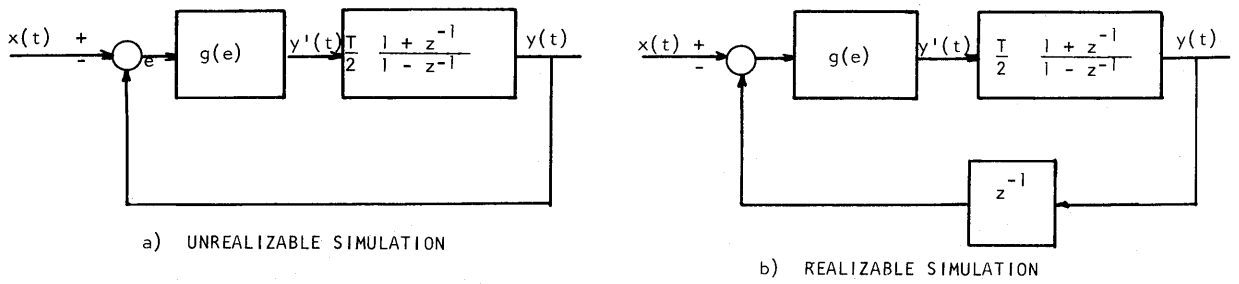



Figure 2. Digital simulation of $dy/dt = g(x-y)$.

the simulation a delay may be introduced[10,11] as shown in Fig. 2b. In open loop problems this delay is not necessary. However, for closed loop problems such as those occurring in the digital simulation of control systems, the introduction of the delay may cause major errors as compared with using optimal realizable discrete transfer function. If the integral is estimated by using only previous values of the dependent variable the need for the delay is eliminated. It will be seen that if

$$F(z) = e^{-snT} = z^{-n} \quad n = 1, 2, \ldots \qquad (11)$$

the need for the delay is eliminated. In letting $F(z) = z^{-1}$ the discrete transfer functions developed are forced to give the least sum of error squared when the present value of the dependent variable is not available. Discrete transfer functions with delay will be referred to as open loop realizable and those without delay as closed loop realizable. In any single loop only one closed loop realizable descrete transfer function would be required. It will be shown that the block diagram of Fig. 3, which uses an optimum realizable integrator, will give less error for a given sample period than the simulation of Fig. 2b.

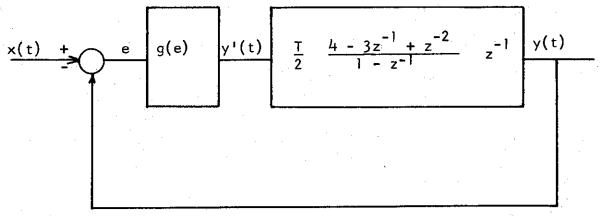



Figure 3. Optimum digital simulation of $dy/dt = g(x-y)$.

## INTEGRATORS DETERMINED FOR STEP, RAMP AND PARABOLA INPUTS

Substitution into Eq. (8) with $I(s) = 1/s$ $f(z) = z^{-n}$ and given $R(s)$ results in Table 1. An example of the procedure is the solution for $R(s) = 1/s^2$ and $n = 1$. When it is noted that

$$R(z)R(z^{-1})F(z)F(z^{-1}) = \frac{T}{(1-z^{-1})^2} \frac{T}{(1-z)^2}$$

Eq. (8) becomes

$$H_0(z) = \frac{\left\{\frac{T^2}{2} \frac{z^{-1}(1+z^{-1})(z)(Tz)}{(1-z^{-1})^3(1-z)^2}\right\} P.R.}{\frac{T}{(1-z^{-1})^2}}$$

$$\frac{\left\{\frac{T^2}{2} \cdot \frac{z(1+z^{-1})}{(1-z^{-1})^3}\right\} P.R.}{T/(1-z^{-1})^2} \qquad (12)$$

The physically realizable portion of the term within the brackets must now be found. Expansion of this term yields

$$\frac{z+1}{1-3z^{-1}+3z^{-2}-z^{-3}} \qquad (13)$$

If $z$ is subtracted from this it becomes

$$\frac{4-3z^{-1}+z^{-2}}{(1-z^{-1})^3} \qquad (14)$$

which is physically realizable. The optimum integrator transfer function is therefore

$$z^{-1}H_0(z) = \frac{T}{2} \frac{(z^{-1})(4-3z^{-1}+z^{-2})}{(1-z^{-1})} \qquad (15)$$

## SINE LOOP ERROR ANALYSIS

Probably the most demanding function required of digital integrators is the generation of sine waves (far less demanding would be, for instance, a decaying exponential generator). Thus it is desirable to develop means by which truncation errors involved in sine wave generation may be compared.

Figures 4a and 4b show the sine loop for the

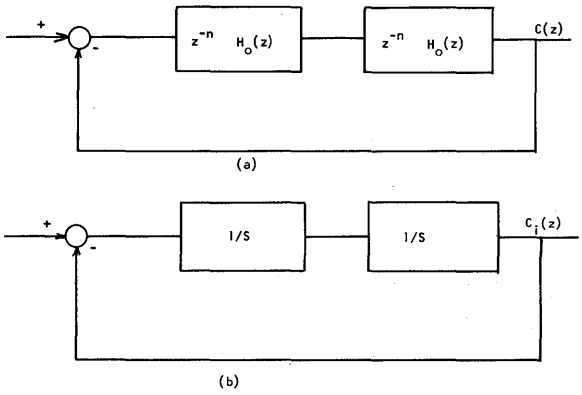



Figure 4. Sine loops for the digital and ideal integrators.

digital and ideal integrators. If impulses are assumed for inputs, the outputs are

$$C(z) = \frac{[z^{-n}H_0(z)]^2}{1 + [z^{-n}H_0(z)]^2} \tag{16}$$

and

$$C_i(z) = [\frac{1}{s^2+1}]^* \tag{17}$$

Now, if the expressions for $z^{-n}H_0(z)$ from Table 1 are substituted into Eq. (16), results are obtained which are listed in Table 2. The expressions for $C(z)$ in Table 2 may be divided out for any desired value of $T$ in order to determine the output sequences. This has been accomplished for each of the integrators

Table 1.

Expressions for Optimum Digital Integrators.

| | $R(s)$ | $n$ | $z^{-n}H_0(z)$ | |
|---|---|---|---|---|
| **a** | $1/s$ | 0 | $\frac{Tz^{-1}}{1-z^{-1}}$ | Rectangular Rule |
| **b** | $1/s$ | 1 | $\frac{Tz^{-1}}{1-z^{-1}}$ | Rectangular Rule |
| **c** | $1/s^2$ | 0 | $\frac{T}{2}\frac{(1+z^{-1})}{(1-z^{-1})}$ | Trapezodial Rule |
| **d** | $1/s^2$ | 1 | $\frac{T}{2}\frac{(z^{-1})(4-3z^{-1}+z^{-2})}{(1-z^{-1})}$ | Three Point Extrapolation |
| **e** | $1/s^3$ | 0 | $\frac{T}{3}\frac{1+4z^{-1}+z^{-2}}{1-z^{-2}}$ | Simpson's Rule |
| **f** | $1/s^3$ | 1 | $\frac{T}{3}\frac{(z^{-1})(8-5z^{-1}+4z^{-2}-z^{-3})}{(1-z^{-2})}$ | Four Point Rule |

with $T = \pi 8$, and results (except for the four-point rule) are shown in Figs. 5 through 10.

From Eq. (16), it is seen that the characteristic equation of the sine loop is

$$1 + [z^{-n}H_0(z)]^2 = 0 \tag{18}$$

After substitution from Table 2, solution of this equation for several values of $T$ gives corresponding pole locations on the root locus. Figs. 11 through 17 show root locus plots for the sine loops for various integrators considered.

Although root loci may be found in the $s$-plane from $s = \frac{1}{T}\ln z$, the different modes of transient or steady-state behavior are characterized by the root loci in the $z$ plane. Complex conjugate poles inside the unit circle correspond to a damped oscillatory output sequence while complex conjugates outside the unit circle indicate an exponentially rising or unstable output sequence. An ideal sine wave output sequence is indicated by complex conjugate poles on the unit circle. If poles fall near the origin so that their magnitude is much less than one, they may be neglected in favor of complex conjugate pairs near the unit circle. Poles near the origin represent fast decaying transients.

In all but one of the root locus plots, it is seen that for small values of $T$ one complex conjugate pair lies near the unit circle while the remainder of the poles are near the origin. Thus, in these cases, it is a very good approximation to deal only with what may be called the control poles near the unit circle, especially after the first few samples.

From Figs. 5 and 10, it appears that the rectan-

Table 2. Sine Loop Output Expressions.

| | Double Integrator Approximation | $C(z)$ |
|---|---|---|
| **a** | $(1/S^2)*$ | $\dfrac{z^{-1}\sin T}{1-2z^{-1}\cos T+z^{-2}}$ |
| **b** | $\left[\dfrac{Tz^{-1}}{1-z^{-1}}\right]^2$ | $\dfrac{T^2z^{-2}}{1-2z^{-1}+(T^2+1)z^{-2}}$ |
| **c** | $\left[\dfrac{T}{2}\ \dfrac{(1+z^{-1})}{(1-z^{-1})}\right]^2$ | $\dfrac{\dfrac{T^2}{(T^2+4)}\ \dfrac{1+2z^{-1}+z^{-2}}{1+\dfrac{2(T^2-4)z^{-1}+z^{-2}}{(T^2+4)}}}{}$ |
| **d** | $\dfrac{T}{2}\ \dfrac{1+z^{-1}}{1-z^{-1}}\ \dfrac{T}{2}\ \dfrac{1+z^{-1}z^{-1}}{1-z^{-1}}$ | $\dfrac{\dfrac{T^2}{4}(1+2z^{-1}+z^{-2})}{1-2z^{-1}\left(1-\dfrac{T^2}{8}\right)+z^{-2}\left(1+\dfrac{T^2}{2}\right)+z^{-3}\left(\dfrac{T^2}{4}\right)}$ |
| **e** | $\left[\dfrac{T}{2}\ \dfrac{z^{-1}(4-3z^{-1}+z^{-2})}{(1-z^{-1})}\right]^2$ | $\dfrac{T^2}{4}\dfrac{(16z^{-2}-24z^{-3}+17z^{-4}-6z^{-5}+z^{-6})}{1-2z^{-1}+z^{-2}+\dfrac{T^2}{4}(16z^{-2}-24z^{-3}+17z^{-4}-6z^{-5}+z^{-6})}$ |
| **f** | $\dfrac{T}{2}\ \dfrac{1+z^{-1}}{1-z^{-1}}\ \dfrac{T}{2}\ \dfrac{z^{-1}(4-3z^{-1}+z^{-2})}{1-z^{-1}}$ | $\dfrac{\dfrac{T^2}{4}\ (4+z^{-1}-2z^{-2}+z^{-3})z^{-1}}{1-2z^{-1}\left(1-\dfrac{T^2}{2}\right)+z^{-2}\ 1+\left(\dfrac{T^2}{4}\right)-z^{-3}\left(\dfrac{T^2}{2}\right)+z^{-4}\left(\dfrac{T^2}{4}\right)}$ |
| **g** | $\left[\dfrac{T}{3}\ \dfrac{(1+4z^{-1}+z^2)}{(1-z^{-2})}\right]$ | $\dfrac{\dfrac{T^2}{T^2+9}\ (1+8z^{-1}+18z^{-2}+8z^{-3}+z^{-4})}{1+\dfrac{8T^2}{T^2+9}z^{-1}+\dfrac{2T^2-2}{\dfrac{T^2+1}{9}}z^{-2}+\dfrac{8T^2}{T^2+9}z^{-3}+z^{-4}}$ |
| **h** | $\left[\dfrac{T}{3}\ \dfrac{z^{-1}(8-5z^{-1}+4z^{-2}-z^{-3})}{(1-z^{-2})}\right]^2$ | $\dfrac{T^2}{9}\dfrac{(64z^{-2}-80z^{-3}+89z^{-4}-56z^{-5}+26z^{-6}-8z^{-7}+z^{-8})}{1-2z^{-2}+z^{-4}+\dfrac{T^2}{9}(64z^{-2}-80z^{-3}+89z^{-4}-56z^{-5}+26z^{-6}-8z^{-7}+z^{-8})}$ |

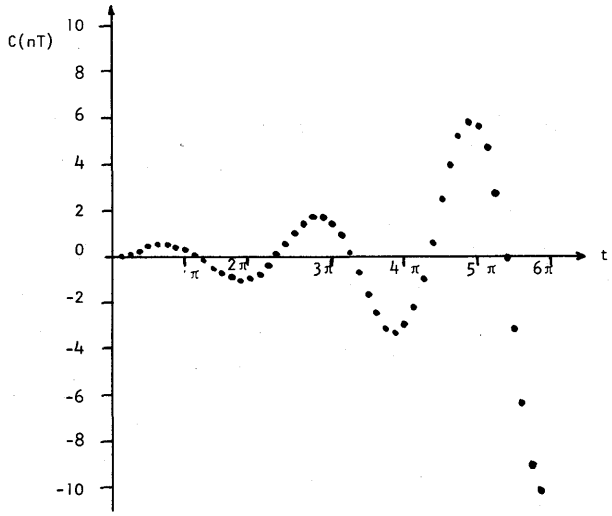



Figure 5. Output sequence for rectangular rule (Table 2b).

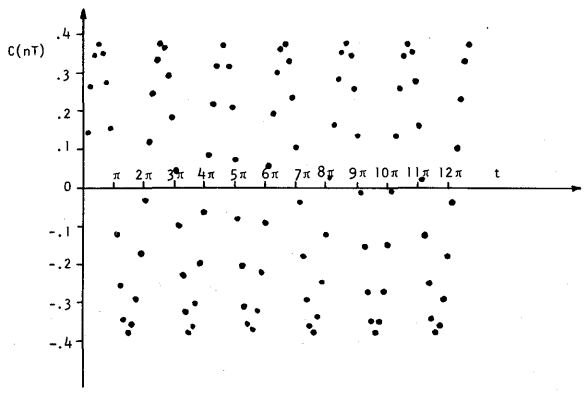



Figure 6. Output sequence for two unrealizable trapezoidal rule integrators (Table 2c).

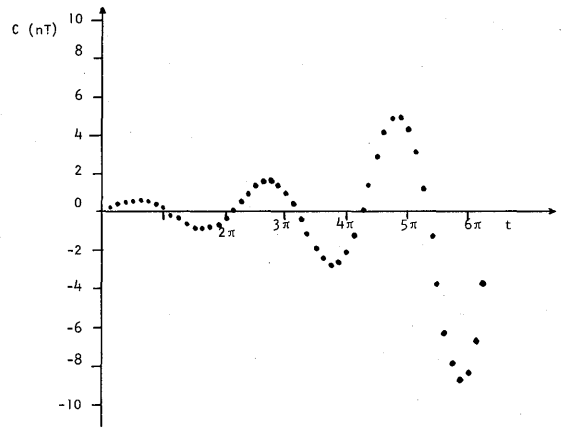



Figure 7. Output sequence for realizable trapezoidal rule (Table 2d).

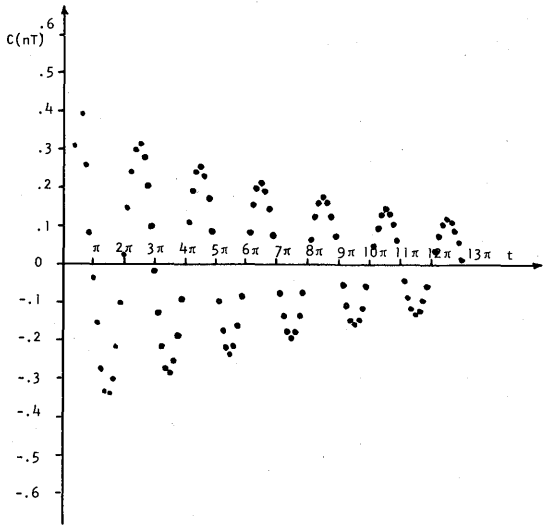



Figure 8. Output sequence for two three-point rule integrators (Table 2e).

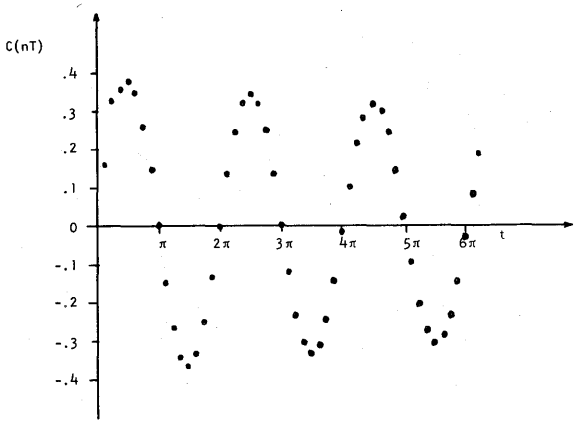



Figure 9. Output sequence for three-point rule—trapezoidal rule combination (Table 2f).

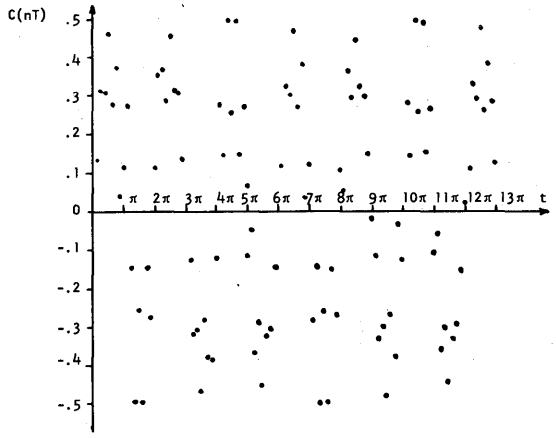



Figure 10. Output sequence for two Simpson's 1/3 rule unrealizable integrators (Table 2g).

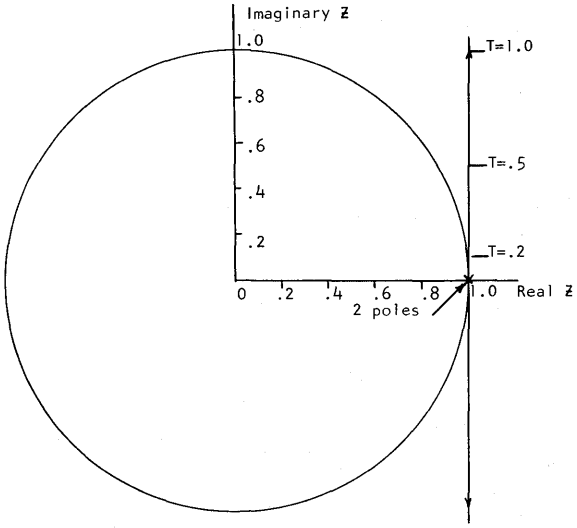



Figure 11. Root locus for rectangular rule since loop (Table 2b).

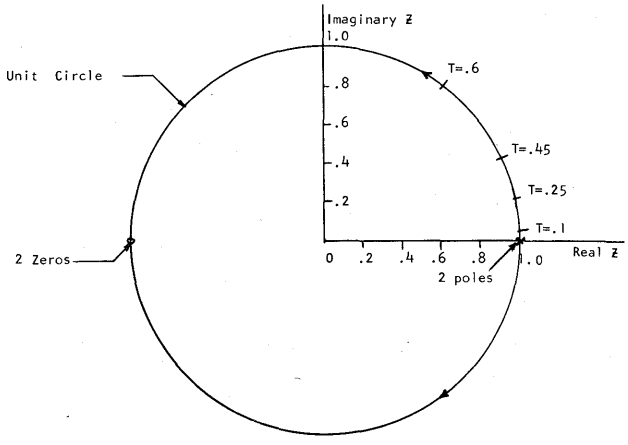



Figure 12. Root locus for unrealizable trapezoidal rule sine loop (Table 2c).

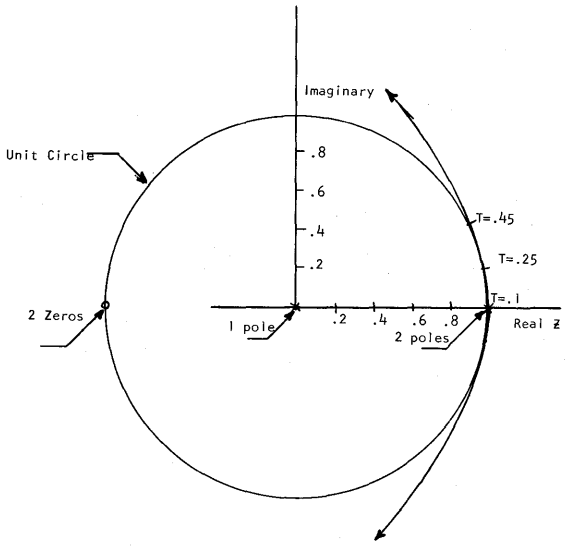



Figure 13. Root locus for realizable trapezoidal rule sine loop.

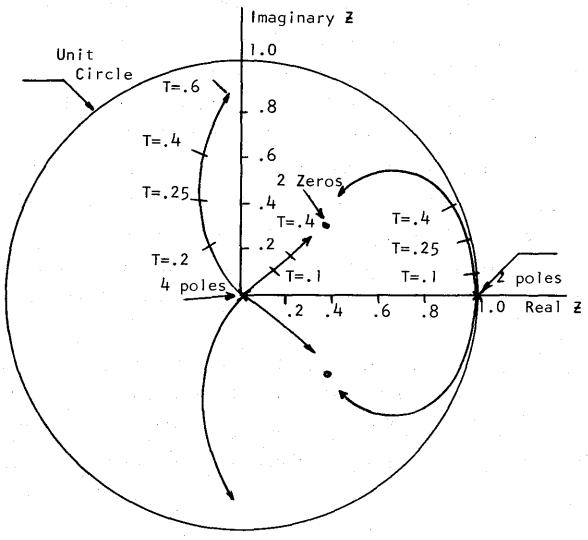



Figure 14. Root locus for two three-point rule integrators (Table 2e)

gular rule, although very simple, compares unfavorably with the trapezoidal and three-point rules. Therefore, it will not be discussed throughout the remainder of this paper. Simpson's ⅓ rule has two pairs of complex conjugate poles on the unit circle. The effect of the extra pair may be seen as a high frequency oscillation of the sample points in Fig. 4. A similar but worse effect may be seen with the four-point rule. Simpson's ⅓ rule and the four-point rule are thus unsatisfactory when sine wave generation is used as a criterion and so are also not discussed further.

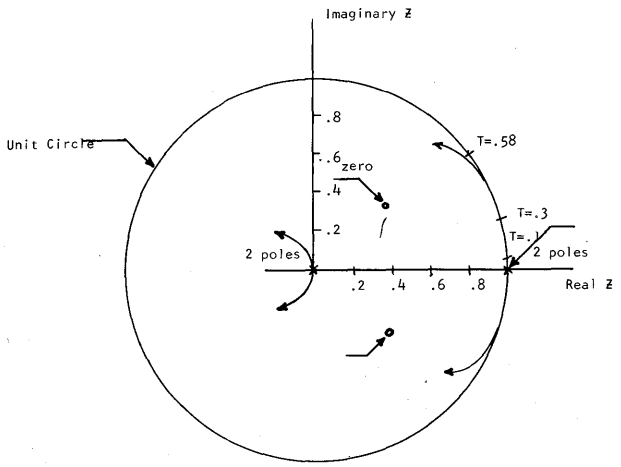



Figure 15. Root locus for trapezoidal integrator three-point rule integrator (Table 2f).

In general, it may be seen that as the integration rule becomes more complicated more poles are added to the discrete transfer function with a corresponding increase in chances of instability and spurious modes of response. The three-point rule adds four extra poles. Simpson's ⅓ rule adds only two extra, but these are very damaging. The four-point rule, which is the closed loop counterpart of Simpson's rule, has six extra poles of which two are very damaging.

The best choice is now seen to be the combination of the open loop trapezoidal rule and the closed loop counterpart of the open loop trapezoidal rule, the three point rule. Almost as desirable is the cascade combination of two closed loop three point rules. The cascade combination is somewhat more desirable in terms of systematization of computer programming.

## AMPLITUDE AND FREQUENCY ERRORS

From the locations of the control poles it is possible to find "$a$" and "$b$" in the expression

$$c(nT) = e^{-anT} \cos bnT \tag{19}$$

since the $z$ transform of this expression is

$$C(z) = \frac{e^{-aT} \sin bTz^{-1}}{1-2e^{-aT} \cos dTz^{-1} + e^{-2aT}z^{-2}} \tag{20}$$

Assuming that all but the control poles may be neglected, Eq. (20) may be used to find the values for "$a$" and "$b$" in relation (19). The results are

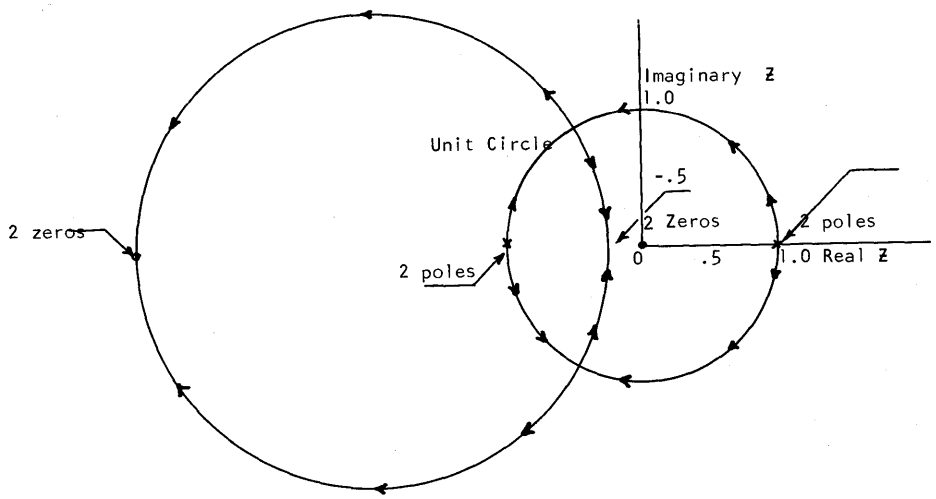



Figure 16. Root locus for Simpson's 1/3 rule sine loop (Table 2g).

presented in Figs. 18 and 19 for the unrealizable trapezoidal integrators, realizable three-point integrators, and combination trapezoidal and three-point rule. As might be expected the unrealizable trapezoidal rule has no amplitude error. However, both the realizable three-point integrators and combination three-point integrator trapezoidal integrator give less frequency (or time) error than the (unrealizable) trapezoidal integrators.

## OTHER APPLICATIONS OF THE METHOD

The authors have had considerable success in applying this method of discrete simulation of control systems to a number of examples. Systematic application of the technique is obtained by a phase variable description of the system and repeated application of the three-point formula for realizable closed loop integration. It is not necessary that this be done. It is convenient with respect to computer programming, however.

The following example illustrates an alternate method of applying this optimum discrete simulation technique. A similar example was first treated by Hurt[10] in his discussion of the IBM simulation method.

Consider the nonlinear system represented in block diagram form by Fig. 20. The three-point integration rule could be used to simulate the $5/s+2$ term. An alternate approach is to find an optimum discrete

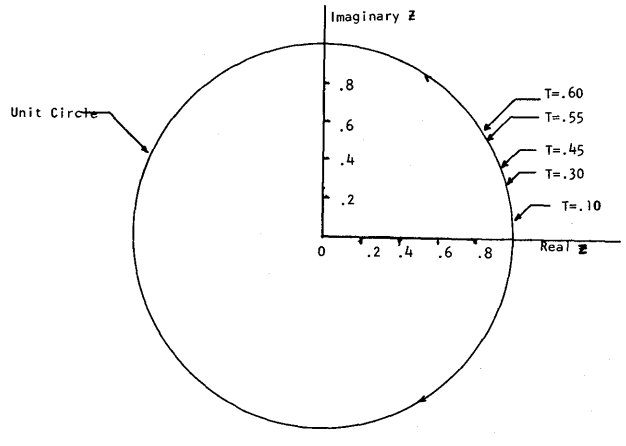



Figure 17. Ideal pole locations (Table 2a).

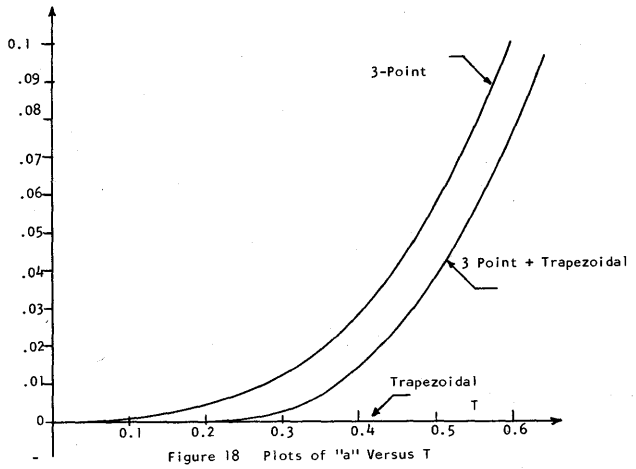



Figure 18. Plots of "$a$" vs $T$.

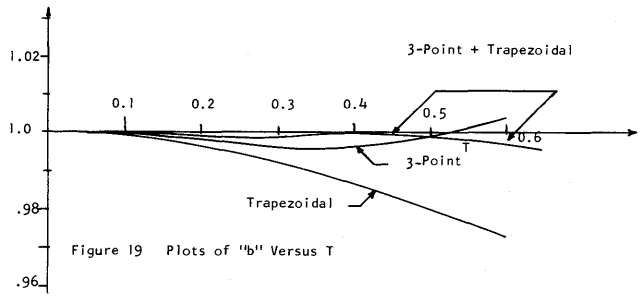



Figure 19. Plots of "$b$" vs $T$.

approximation of the entire transfer function $5/s+2$ for a suitable input such as a ramp. The approximation need only be open loop realizable. From Eq. (8), the discrete transfer function under the stated conditions becomes

$$\frac{5}{4T}\,\frac{(2T-1+e^{-2T})+z^{-1}(1-e^{-2T}-2Te^{-2T})}{(1-e^{-2T}z^{-1})}$$

By using the closed loop realizable three-point rule for integration the block diagram for the discrete

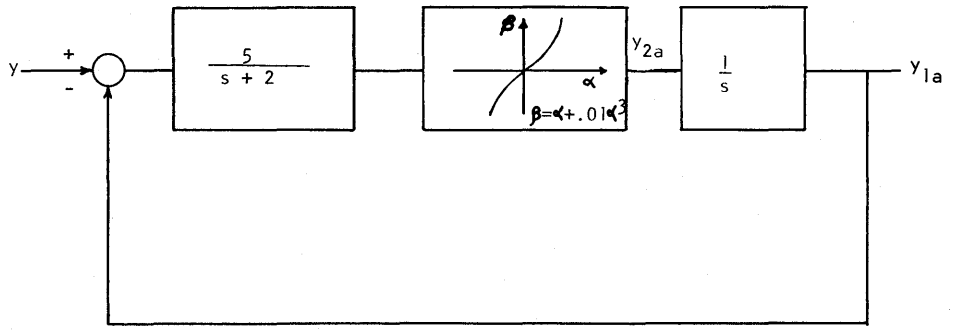



Figure 20. Feedback system with limiter.

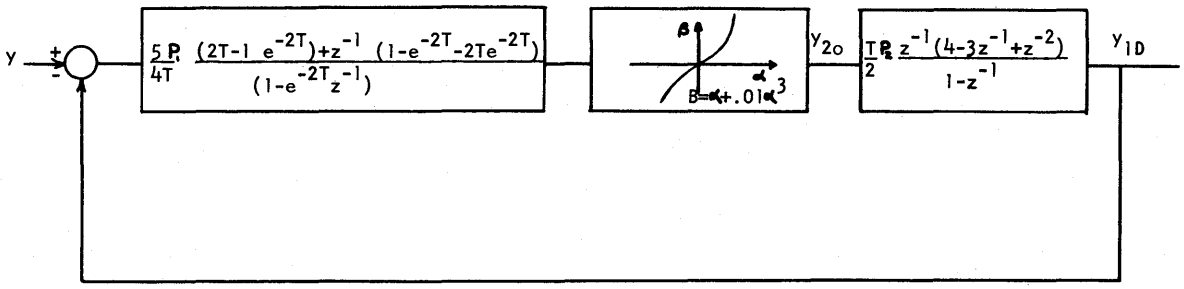



Figure 21. Discrete simulation of Fig. 20.

simulation becomes that shown in Fig. 21. ($P_1=P_2=1$).

Figure 22 shows the unit step response of the system for three sample periods $T=0.1$, 0.2, and 0.5 seconds. While the step response for $T=0.5$ is not as accurate as the IBM method for $T=0.5$, it is considerably better than Tustin's method of simulation.[10] Since this method does not require a detailed root locus redesign of the loop gains or computation of the "input transfer function" as does the IBM method and since it provides considerable reduction in error over the Tustin and other approaches,[11] it is felt that its use in discrete system simulation is warranted.

If the system is decidedly nonlinear, a more useful result is obtained if the nonlinear characterics are taken into account. For the example treated here, as the amplitude of the input step is increased, the approximation error for a given sampling period increases. This is due to the fact that the "optimum" discrete approximation has been derived on a linear basis and the system is highly nonlinear for large inputs. The approximation error may be reduced by decreasing the sampling period or by making use of the fact that the system is acually nonlinear in determining the "optimum" discrete approximation. The IBM method[10] essentially consists of adjusting gain $P_1$ or $P_2$ such that the closed loop eigenvalues for the discrete approximation are the same as for a linearized version of the continuous system. Here a somewhat different method is proposed which takes into account the actual nonlinear charactertistic of the system. It is desired that the digital system output, $\underline{y}_d(t)$ approach the analog output $\underline{y}_a(t)$, where $\underline{y}_a(t)$

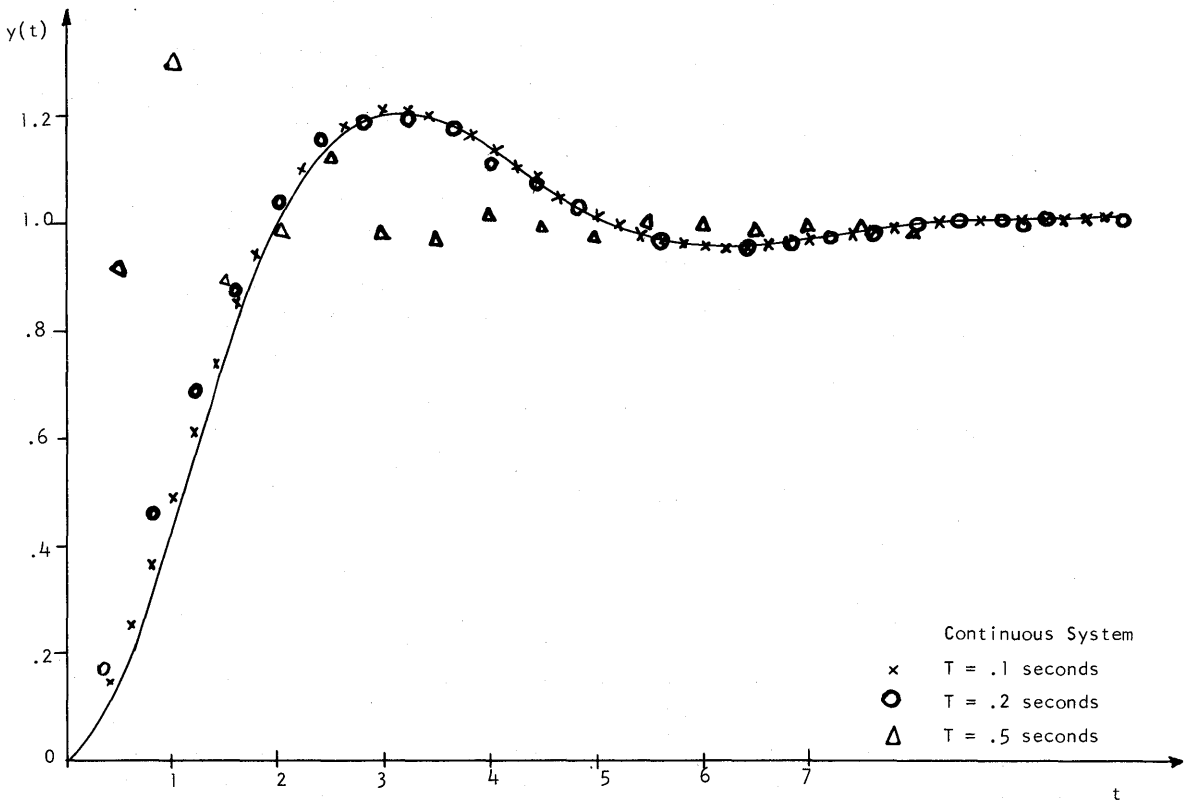



Figure 22. Step response of the system of Fig. 20.

and $\underline{y}_d(t)$ are $m$ vectors describing the system output state for a given input. It is assumed that the analog state vector output $\underline{y}_a(t)$ is completely known, as is the input to the system. The form of the digital system has been determined by the previously presented method and is known except for a certain number of constant parameters $\underline{P}$, which are to be determined. $\underline{P}$ will be interpreted as a $p$ vector and will be adjusted to minimize the cost function

$$J = \tfrac{1}{2} \sum_{k=0}^{N} \| \underline{y}_a(kT) - \underline{y}_d(kT) \|_R^2 \qquad (21)$$

subject to the constraint that

$$\underline{y}_d(\overline{n+1\ T}) = f[\underline{y}_d(nT), \underline{P}] \qquad (22)$$

$$\underline{y}_d(0) = \underline{y}_a(0)$$

$$\underline{P}(\overline{n+1\ T}) = \underline{P}(nT) \qquad (23)$$

Application of standard variational calculus procedures demonstrates that the optimum parameter vector P is determined by solution of difference Eqs. (22) and (23) together with adjoint vector difference equations

$$\underline{\lambda}_y(\overline{nT}) = -\nabla \underline{y}f[\underline{y}_d(nT), \underline{P}]\,\underline{\lambda}_y(n+1\ T) + R[\underline{y}_a(nT) - \underline{y}_d(nT)] \qquad (24)$$

$$\underline{\lambda}_p(n+1\ T) = -\Delta_p \underline{f}[\underline{y}_d(nT), \underline{P}]\,\underline{\lambda}$$

$$\underline{\lambda}_p(nT) = -\nabla\, pf[\underline{y}_d(\overline{nT}), P]\,\underline{\lambda}_p(n+1\ T) \qquad (25)$$

$$\underline{\lambda}_y(NT) = \underline{\lambda}_p(NT) = \underline{\lambda}_p(0) = 0 \qquad (26)$$

where $\Delta$ represents the gradient operator. Equations (22) through (26) represent a two-point nonlinear boundary value problem which may be solved by the method of quasilinearization,[12] a technique due to Bellman.[13] A new $2(m+p)$ vector

$$\underline{x}'(nT) = [\underline{y}'_d(nT), \underline{P}', \underline{\lambda}'_y(nT), \underline{\lambda}'_p(nT)] \qquad (27)$$

will be defined such that Eqs. (22) through (26) may be described by the difference equation

$$\underline{x}(\overline{n+1\ T}) = \underline{q}[x(nT)] \qquad (28)$$

with the boundary conditions

$$\underline{C}_i(jT), \underline{x}(jT) = b_i(jT) \quad \begin{matrix} j=0, N \\ i=1, 2, \ldots (m+p) \end{matrix} \qquad (29)$$

where C and x are $2(m+p)$-dimensional vectors $<,>$ denotes the inner product. If $x^0(nT)$ is the initial

guess to the solution of Eq. (28), the $(q+1)$th approximation is obtained from the $q$th by

$$\underline{X}^{q+1}(\overline{n+1}\ T) = \underline{q}[\underline{x}^q(nT)] + J\{\underline{q}[\underline{x}^q(nT)]\}\{\underline{x}^{q+1}(nT) - \underline{x}^q(nT)\} \quad (30)$$

where $J$ is the Jacobian matrix whose $ij$th element is the partial derivative $\partial g_i/\partial x_j$. If $\Phi^{q+1}(nT)$ is the fundamental matrix of

$$\Phi^{q+1}(\overline{n+1}\ T) = J\{\underline{q}[\underline{x}^q(nT)]\}\Phi^{q+1}(nT)$$
$$\Phi^{q+1}(0) = I = \text{identity matrix} \quad (31)$$

and $\Omega^{q+1}(nT)$ is the particular vector solution of

$$\Omega^{q+1}(n+1\ T) = \underline{q}[\underline{x}^q(nT)] + J\{\underline{q}[\underline{x}^q(nT)]\}\{\Omega^{q+1}(nT) - \underline{x}^q(nT)\} \quad (32)$$

The solution of Eq. (30) is

$$\underline{x}^{q+1}(nT) = \Phi^{q+1}(nT)\,\underline{V}^{q+1} + \underline{\Omega}^{p+1}(nT) \quad (33)$$

where $V^{q+1}$ is a constant vector which is determined by solving

$$<\underline{C}_i(jT), \Phi^{q+1}(jT)\,\underline{V}^{q+1} + \underline{\Omega}^{q+1}(nT)> = b_i(jT)$$
$$j = 0, N \quad (34)$$
$$i = 1, 2, 3, \ldots (m+p)$$

Application of this technique to the specific problem of Fig. 21 with $T = 0.1$, $P_2 = 1$, $N = 100$, $J = \sum_{k=0}^{100} [y_a(kT) - y_d(kT)]^2$ yields a parameter $P_1$ which decreases with increasing input step size as illustrated in Fig. 23. Without the change in $P_1$ as

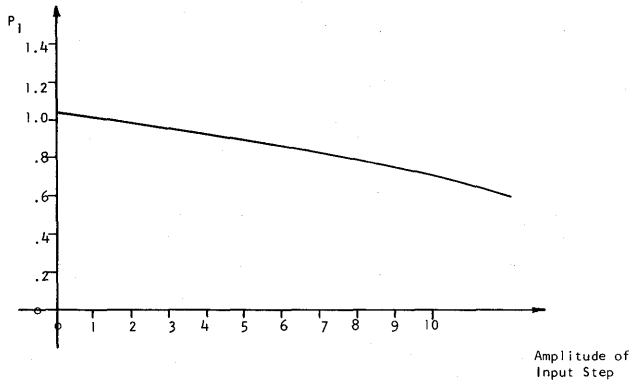



Figure 23. Optimum gain parameter setting for nonlinear system.

indicated by this figure, the discrete simulation becomes unstable for a given sampling period as the step size increases. The use of the optimization technique allows determination of $P_1$ as a function of the input such that very large (compared to conventional methods) sampling periods can be used. The price paid for this "tailoring" is of course an increase in the time required to determine the discrete system parameters. However, the quasilinearization method is readily programmed on a digital or hybrid computer and computation time is short if rapid convergence is assured by a good initial guess to $x^0(nT)$, as is easily the case here.

## CONCLUSIONS

A new method for the discrete simulation of systems has been presented. The technique consists of determining best least squares approximations for the linear elements of a continuous system and adjusting loop gains via the quasilinearization approach such that an accurate approximation to any decidedly nonlinear elements is obtained. In many cases tailoring of the complete closed loop system is not necessaryy. If tailoring is necessary, actual treatment of the nonlinearities involved via quasilinearization allows resolution of any stability problems concerned with the discrete approximation.

## ACKNOWLEDGMENTS

The first part of this research was initiated by the authors while at the University of Arizona. The principal author is indebted to his students, at the University of Florida, in particular B. Eisenberg and S. Goldberg for assisting in the digital computer studies necessary to evaluate the theoretical results in the latter portions of the paper.

## REFERENCES

1. J. T. Tou, *Digital and Sampled Data Control Systems,* McGraw Hill, 1959.

2. B. Widrow, "Statistical Analysis of Amplitude-Quantized Sampled-Data Systems," *Applications and Industry,* vol. 52, pp. 555-567, (Jan. 1961).

3. Ho Wo Gschwind, "Digital Differential Analyzers," *Electronic Computers,* P. Von Handel, ed., Prentice-Hall, 1961, p. 139.

4. D. J. Nelson, *A Foundation for the Analysis of Analog-Oriented Combined Computer Systems,* Doctoral Thesis, Stanford University, Stanford, Calif., Apr. 1962.

5. F. B. Hills, *A Study of Incremental Computation by Difference Equations,* Masters Thesis, Massachusetts Institute of Technology, 1958.

6. M. Pavlevsky, *Computer Handbook,* Korn and Huskey, eds., McGraw-Hill, 1961, pp. 19-14—19-74.

7. J. T. Tou, "Statistical Design of Digital Control Systems," *IRE Transactions on Automatic Control,* vol. AC-5, pp. 290-296, (Sept. 1960).

8. F. B. Hildebrand, *Introduction to Numerical Analysis,* McGraw-Hill, 1956.

9. R. W. Burt, *Optimum Design and Error Analysis of Digital Integrators,* Thesis, University of Arizona, Tucson, 1963.

10. J. M. Hurt, "New Difference Equation Technique for Solving Non-Linear Differential Equations," *AFIPS Conference Proceedings,* vol. 24, 1964.

11. M. E. Fowler, "A New Numerical Method fo Simulation," *Simulation,* May 1965.

12. A. P. Sage, "Suboptimal Control via Frequency Domain Techniques, Quasilinearization, and Differential Approximation," *Proceedings, Third Allerton Conference on Circuit and System Theory,* University of Illinois, Oct. 1965.

13. R. Bellman, R. Kalaba and R. Sridhar, "Adaptive Control via Quasilinearization and Differential Approximation," *Rand Corporation Research Memorandum RM* -3928 *PR* (Nov. 1963).

# SEQUENTIAL ANALOG-DIGITAL COMPUTER

Hermann Schmid
*General Electric Company*
*Light Military Electronics Department*
*Johnson City, New York*

## INTRODUCTION

The applications of digital computers to analog control systems, where the inputs to the computer are the outputs from analog sensors and the outputs from the computer must drive analog controls, increase steadily, even in cases where analog computer accuracy (1 percent) would be sufficient.
The reasons for this are:

- Digital computers, built with integrated circuits, are small, reliable, use little power and are insensitive to changes in environment.
- Conventional analog computers[1,2] in comparison are large, unreliable, vary considerably with change in environment, require precision components, stable power supplies and many adjustments. In addition, conventional analog computers do not lend themselves easily to sequential operation.

Although the statements above are correct, a comparison of this type is worthless and often misleading because the interface equipment required with the digital computer is not included. In control applications with many signals but few computations, the size, weight and cost of the interface circuits may equal, or even exceed, those of the computer. Besides, the analog-to-digital and digital-to-analog conversion circuits are subject to the same shortcomings and limitations as the analog computer circuits.

This paper describes a Sequential Analog-Digital Computer (SADC) which overcomes many of the limitations of the conventional techniques described above, by combining an analog arithmetic unit with a digital memory and a digital control unit. The computer, thus, exploits the advantages of the analog technique (no interface circuits required, ease of summing and scaling, high resolution) with the advantages of the digital technique (drift-free storage, logic decisions, ease in signal switching). Except for a few precision components, SADC can be built entirely with integrated circuits.

There are only a few sequential analog computers described in literature[3,4] and only one known technique which is similar to SADC.[5] E. V. Bohn describes a pulsetime computer which uses vacuum tube integrators, vacuum tube current switches and a magnetic drum for storage.

## COMPUTER ORGANIZATION

As shown in Fig. 1, the organization of SADC is very similar to that of a sequential digital computer.[6,7] SADC combines an analog arithmetic unit with a digital memory and a digital control unit in a unique way. The inputs to SADC may be a-c or d-c voltage or pulse-time signals, whereas the outputs are either in pulse-time or d-c voltage form.

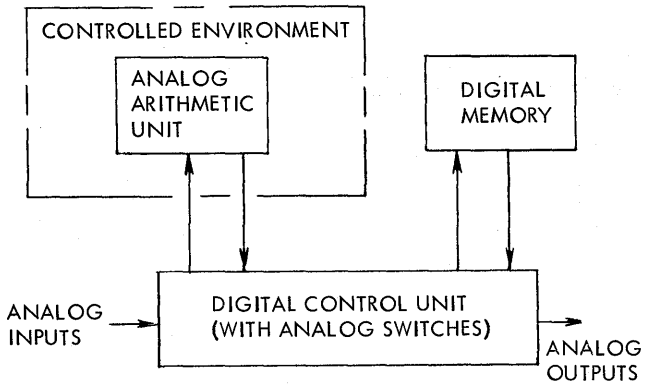



Figure 1. Basic building blocks of SADC.

Analog computing elements in the arithmetic unit are connected in various ways to perform different arithmetic operations under control of the program. The control unit connects signals to and from the arithmetic unit in appropriate time intervals according to the stored program.

The arithmetic unit performs each arithmetic operation without the use of the memory in one basic timing interval, no matter how complex. Often, the results of one arithmetic operation are used as the initial conditions for the next operation, thus eliminating the need for storage and associated transfers.

Pulse-time signals are used for transferring information from the arithmetic unit to the memory, and vice versa. The outputs of SADC are not provided continuously (sampled-data), and one buffer element is required for each output signal, just as in a sequential digital computer.

## ARITHMETIC UNIT

The arithmetic unit in Fig. 2 consists of integrators, inverters and comparators which may be used separately or combined. One integrator and one comparator perform; e. g., multiplication or division.

The design of the analog integrator is conventional,[1,2] with resistor $R$ in input and capacitor $C$ in the feedback path of a high-gain operational amplifier, the output of which is

$$V_o = -1/RC \int V_I dt$$

The quality of this integrator depends on the precision and the stability of $R$, $C$, and on the zero offsets of the amplifier.

The design of the analog summer-inverter is also conventional,[1,2] with resistors $R_{31}$ to $R_{3n}$ in the input and $R_4$ in the feedback path of a high-gain operational amplifier, the output of which is

$$V_o = -\left[\frac{V_1}{R_{31}} + \frac{V_2}{R_{32}} + \ldots + \frac{V_n}{R_{3n}}\right] R_4$$

The quality of the inverter depends on the precision of $R_3$, $R_4$, and on the zero offsets of the amplifier.

The comparator uses a differential amplifier and logic circuits to indicate on two wires the result of the comparison $V_o$-$V_c$. When $(V_o$-$V_c)$ is larger than +2mv, the amplifier output $V_P$ is $+V_B$, and when $(V_o$-$V_c)$ is smaller than –2mv, $V_P$ is zero. The transition from $+V_B$ to zero requires 50 nanoseconds. $V_P$, which indicates the polarity of $(V_o$-$V_c)$, is stored in a flip flop. NOR-gating provides the pulse-time outputs $Pt_1$ if $(V_o$-$V_c) > 0$, and $Pt_1$ if $(V_o$-$V_c) < 0$.

The number of integrators, comparators and inverters used in an arithmetic unit is a function of the problem complexity and the required computation speed. The larger the number of computing elements, the more arithmetic operations can be performed in parallel.

All amplifiers, resistors and capacitors are susceptible to changes in environment. The arithmetic unit and the power supply regulators, therefore, are put into a small oven, operating at 70 ± 2°C to minimize computation errors due to changes in temperature.

## MEMORY

In the sequential analog computer, relatively few variable signals need be stored because:

- Arithmetic unit does not require storage when executing an arithmetic operation.

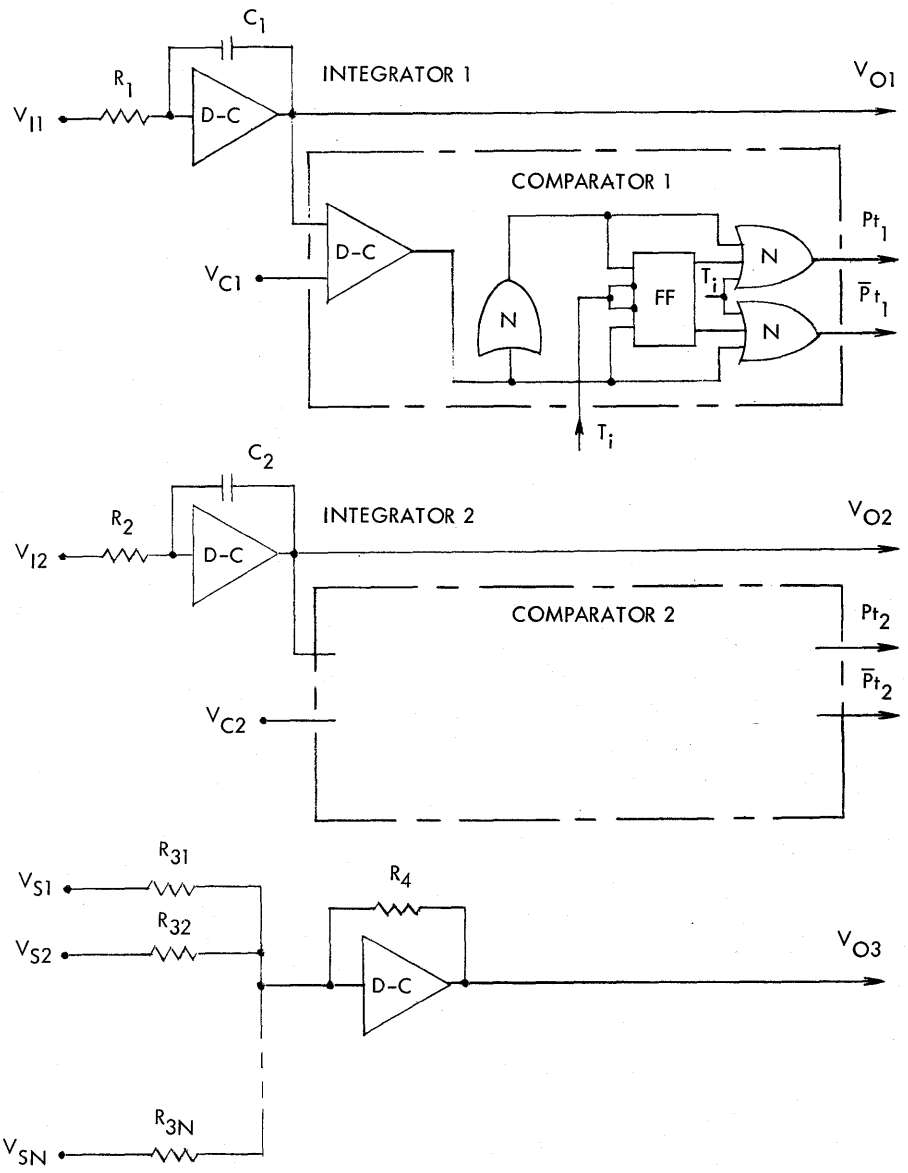



Figure 2. Typical arithmetic unit.

- Inputs are continuously available.
- Arithmetic unit has itself limited storage capability.
- Often, outputs of one operation remain as inputs for the next operation.

Therefore, the requirements on the memory elements for the SADC are entirely different from those of conventional digital computer memory elements. For a pulse-time memory element, it is important that the circuitry can be packaged into integrated-circuit packages and that it can be driven from and read by integrated circuits. The method of storing pulse-time signals with continuously operating counters[8] fulfills these conditions.

In the pulse-time memory as shown in Fig. 3 the output square-waves from the most significant stage of the master and the slave counter, both driven by the same frequency $f_c$, are gated to detect the phase shift between them. This phase shift, which is proportional to the difference in count between these two counters, produces the pulse-width output signal $t_x$.

The value of $t_x$ varies when the number of pulses

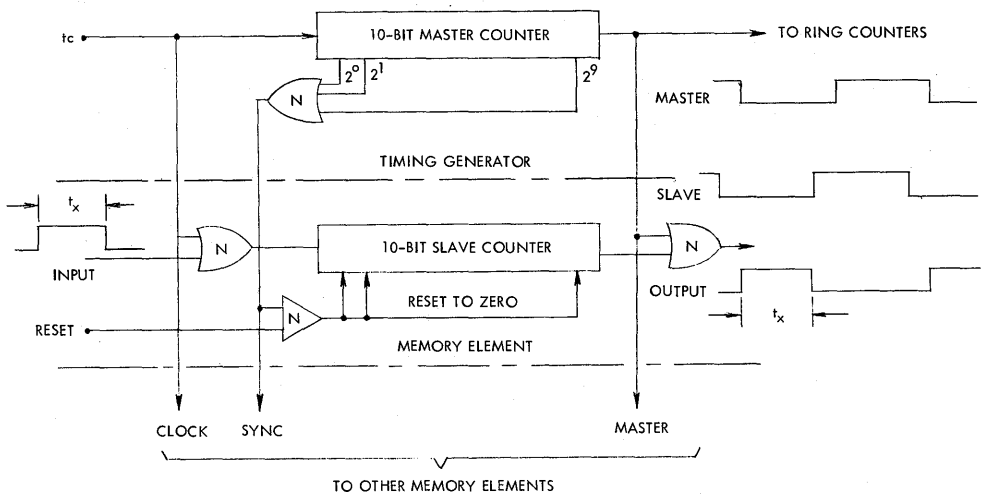



Figure 3. Memory element.

reaching the slave counter is different from the number of pulses reaching the master counter. When pulses are added to the clock pulses $f_c$, $t_x$ increases; when pulses are subtracted, $t_x$ decreases. When no pulses are added or subtracted, $t_x$ stays constant.

The memory can be read out nondestructively. To read in new information, the slave counter must be reset when the master counter is zero.

Only unidirectional counters are necessary. These may be available shortly in a single integrated-circuit package. One master counter, with appropriate buffers at output, can supply many slave counters with the reference square-wave signal.

## CONTROL UNIT

The control unit in Fig. 4, which consists of the timing generator, the program generator and the input/output switches, regulates the flow of information to and from the arithmetic unit and the memory by energizing voltage switches and digital logic circuits.

The timing generator shown produces, in sequence, 24 timing signals of equal length by gating the outputs of a 4-bit high-speed ring counter with outputs of a 6-bit low-speed ring counter in 24 buffer NOR circuits.

Variable program storage, as in general-purpose machines,[6,7] provides flexibility but requires write-read memories (cores, drum, etc.). Special-purpose machines, using fixed or wired program storage, require simple read-only memories (rope cores, diode matrices, etc.).

In the fixed-storage program generator of SADC, the program memory and the digital switching are comprised in one logic network which combines timing control signals with pulse-time input signals and generates n analog switch control lines and m pulse-time output signals. The logic circuitry of the program generator can be defined precisely for any particular application with a specific set of Boolean equations.

The switching of analog voltages is still a problem at present when size and cost must be considered, since low-impedance integrated metallic oxide semiconductor (MOS) switches[9] are not yet available, integrated photo-electric switches[10] are too expensive ($100), and transformer-coupled transistor switches[11] are too bulky. The best compromise is the direct-coupled transistor switch in Fig. 4, capable of switching signals with ± 5V excursions but which require a base drive from − 6V to + 12V, a low source and high load impedance. The voltage across the saturated transistor $V_{CE}$ is dependent on the signal voltage, the base current and the load current. In the proposed circuit, $V_{CE}$ is maintained within ± 2mv. Special driver amplifiers, presently built with one transistor, are required to provide the large-swing base drive signal. Zener diodes are used to shift the level from zero to − 6V.

## ARITHMETIC OPERATIONS

A change in the interconnection of the computing elements permits the execution of various arithmetic operations. The program specifies in each operation interval $T_i$ what arithmetic operations are to

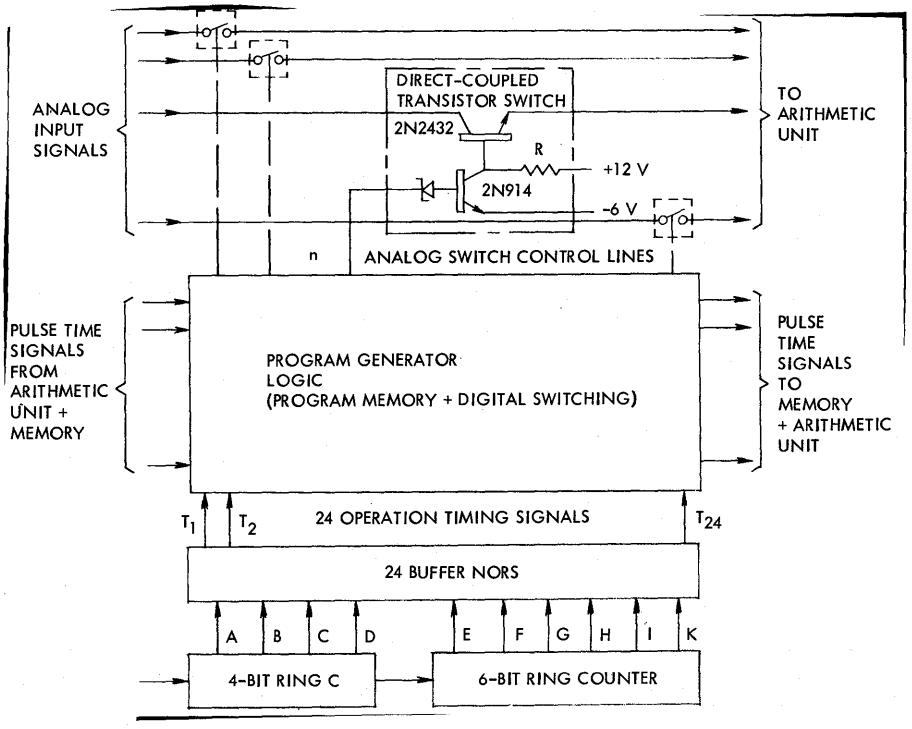



Figure 4. A control unit for 24 sequential operations.

be performed and how the various computing elements must be interconnected by energizing the appropriate analog and digital switches. The arithmetic unit in Fig. 2 can perform two multiplications, two divisions, two additions, two SET or two READ operations simultaneously.

In all arithmetic operations, a voltage or voltage-time function is integrated for a controlled period of time. The integrator output voltage $V_o$, at the end of the integration period, or the time required to make $V_o$ equal to some specific potential, are the results desired.

The SET operation establishes the initial conditions prior to some arithmetic operations by integrating a certain analog voltage during $T_i$ or by integrating a reference voltage $V_R$ during a certain pulse time.

The READ operation converts the output voltage $V_o$ into a pulse time after certain arithmetic operations by integrating a reference voltage with appropriate sign until $V_o$ is zero.

Often, however, the voltage outputs of one arithmetic operation are the inputs or the initial conditions for the next operation. Most SET and READ operations are thus eliminated.

Information is transferred from one computing element to another by having pulse-time signals, generated by a READ operation on the transmitting element, control a SET operation on the receiving element. When the TRANSFER operation is applied twice between two elements, errors due to variations in the time constants and the reference voltage cancel.

Each arithmetic or auxiliary operation is completed in one operation interval $T_i$. All intervals are of equal length and only one occurs at any one time.

### *Conversion of Signals*

The arithmetic unit may be used to convert analog signals from one form into another form. Any conversion can be performed before or in between the arithmetic operations of a computer program.

The conversion from pulse time to d-c voltage requires only a SET operation.

The conversion from d-c voltage to pulse time, which is usually referred to as pulse-width modulation,[12] requires a SET operation to be followed by a READ operation.

The conversion from a-c voltage to d-c voltage is performed by integrating one-half cycle of an a-c signal. The integrator output voltage at the end of a half-cycle is proportional to the amplitude of the a-c voltage.

The conversion from a-c voltage to pulse time is performed similarly to a d-c to pulse-time conversion.

The conversion from three-wire synchro signals to pulse time[13] is accomplished by converting into two-wire signals with a Scott T transformer and by performing a-c to d-c conversion on each of these two signals. At the end of the integration interval, the integrator voltages are:

$$V_{01} = kE_o \sin A \qquad V_{02} = kE_o \cos A$$

and present the components of a vector $\bar{R}$. A coordinate transformation operation is performed in the next half-cycle by rotating R until $V_{02}$ is zero. The time required for this rotation is proportional to the angle presented by the synchro signals, but independent of the reference voltage amplitude $E_o$ and the synchro transformation ratio k.

Conversions with a-c voltage inputs require that the computer timing is synchronized to the a-c signal.

*Addition -Subtraction*

Any addition can be performed serially or parallel; subtraction is performed by negative addition.

Parallel addition, as shown in Fig. 5a, is carried out by summing currents,[1,2] proportional to the variables in the sum, at the input of the integrator or the inverter. Since this requires additional precision resistors, parallel addition is used only when sequential addition is too time consuming.

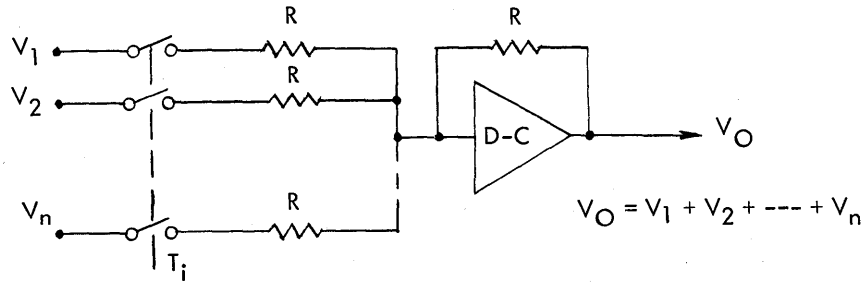



Figure 5a. Parallel addition with inverter or integrator.

Sequential addition, as shown in Fig. 5b, is carried out by integrating each of the voltages representing a variable for one operation interval, separately and sequentially in time, and without resetting the integrator. Sequential addition can be performed also with a memory element.

*Integration*

Integration, as illustrated in Fig. 6, is approximated by sequential addition in a modified memory element. The pulse-time memory element has a capability of summing and storing information and,

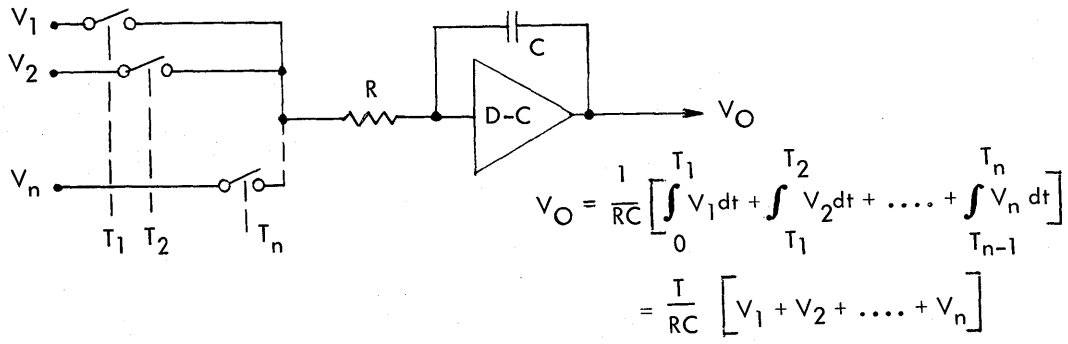



Figure 5b. Sequential addition with integrator.

thus, has a capability of approximating integration.

Rectangular or trapezoidal integration[7] is performed by summing properly scaled fractions or multiples of the integrand in the memory element once or several times during each computing interval.

To provide adequate input and output resolution, a 10-bit delta counter must be added to the memory element, to which $f_c$ is connected when $t_x$ (representing an integrand X) is present. The delta counter fills up and overflows at a rate $R_x$ which is proportional to $t_x$. Each time the delta counter overflows a fixed number of pulses $N_T$ is added to the clock pulses for the slave counter. The time constant of the integrator is inversely proportional to $N_T$.

The computing elements in the arithmetic unit perform the required scaling or multiplying and the conversion of the integrand into a pulse-time signal.

Integrations in the SADC are subjected to all limitations of numerical integration, but have no other intrinsic limitations.

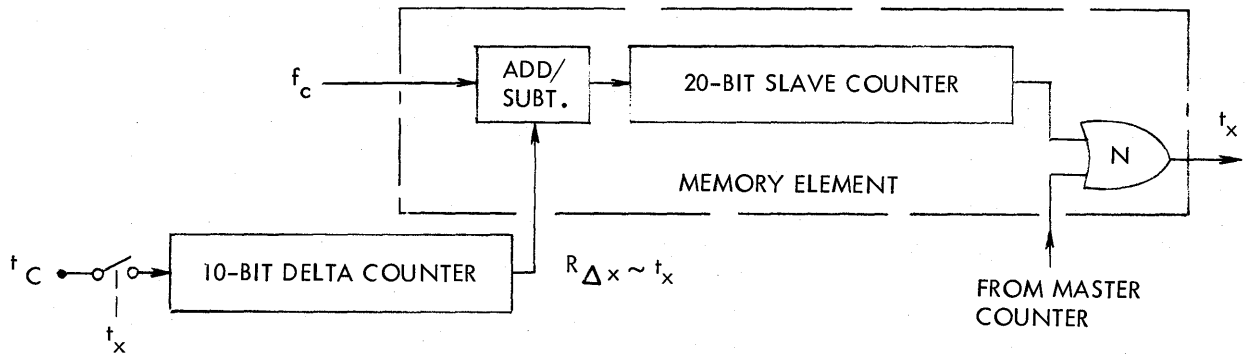



Figure 6. Digital pulse-time integrator.

*Multiplication*

Multiplication, as shown in Fig. 7, is performed by integrating a voltage $V_x$ for a time $t_y$. The voltage across the integrator at the end of $t_y$ is proportional to the product XY, when $V_x$ is constant during $t_y$, since

$$V_o = \frac{1}{RC} \; {}_0\int^{t_y} V_x \, dt = k \, V_x t_y$$

One variable must be a d-c voltage, the other a pulse-time signal. If both are d-c voltages, one must be converted into pulse-time form; if both are pulse-time signals, one must be converted into a d-c voltage.

The pulse-time signal $t_y$ operates the switch which connects the voltage $V_x$ to the integrator. Both signals may be bipolar.

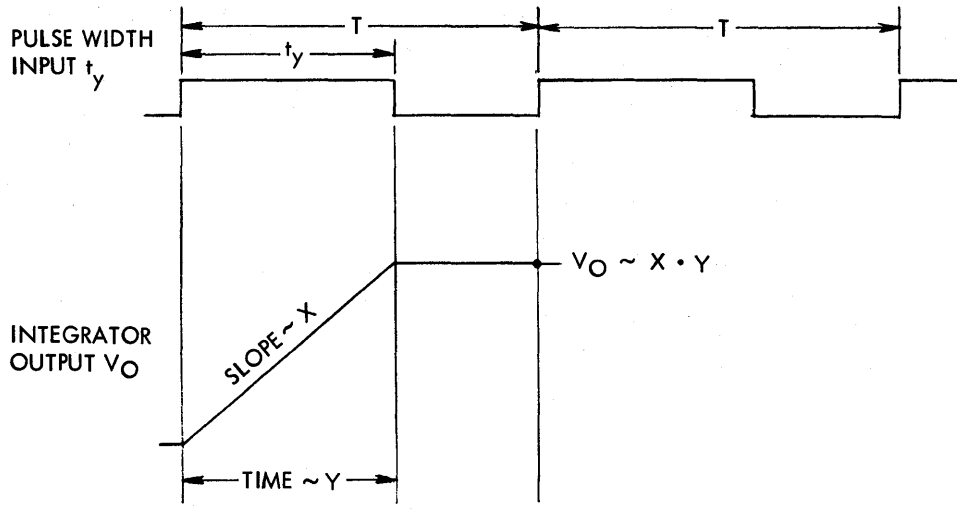



Figure 7. Multiplication waveforms.

*Division*

Division, as illustrated in Fig. 8, is performed by integrating $V_u$ until the output of the integrator reduces from $V_w$ to zero.[14]

The voltage $V_u$ represents the divisor; $V_w$ represents the dividend which must be set into the integrator before the division operation starts. The integrator output voltage decreases with a constant slope as:

$$V_o(t) = V_w - \frac{1}{RC} \int_0^{t_Q} V_u dt = V_w - kV_u t_Q$$

when $V_o(t)$ is zero, $t_Q = V_w/kV_u$

$V_u$ is connected to the integrator by an analog switch which is controlled by the pulse-time signal $t_Q$, which starts at the beginning of the operation interval and ends when the integrator output voltage reaches zero. $V_w$ can be either a positive or a negative potential. $V_u$ must have the opposite polarity of $V_w$.

*Arbitrary Function Generation*

In SADC arbitrary functions are approximated with linear segments.[15] A staircase waveform approximating $f'(x)$, which is the derivative of the desired function, is generated by connecting the reference voltage sequentially to the set of scaled

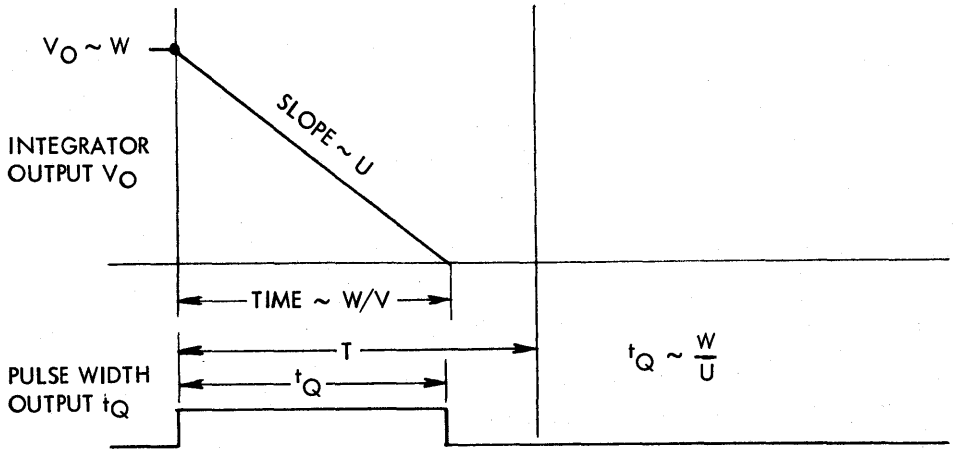



Figure 8. Division waveforms.

resistors at the input of the inverter. This is illustrated in Fig. 9.

The number of segments n required depends on the accuracy desired and on how fast $f'(x)$ is changing. The length of the segment $t_i$ is $T_i/n$. Preferably, $t_i$ should be made a binary fraction of $T_i$, for ease of generating these timing intervals. The segment timing intervals $t_i$ are generated like the operation intervals in Fig. 4 with ring counters and gates.

The staircase waveform is integrated to produce the linear segment curve $f(t)$. The time of integration is determined by the pulse-time signal $t_x$. At the end of $t_x$, the integrator output voltage is:

$$V_o = \frac{1}{RC} \int_0^{t_x} f'(t)\, dt = \frac{1}{RC} f(t_x) - f(0)$$

One set of precision resistors and one inverter are needed for each function to be generated. The accuracy of the function generator depends on the number of segments used, the function to be generated, and the precision of the components.

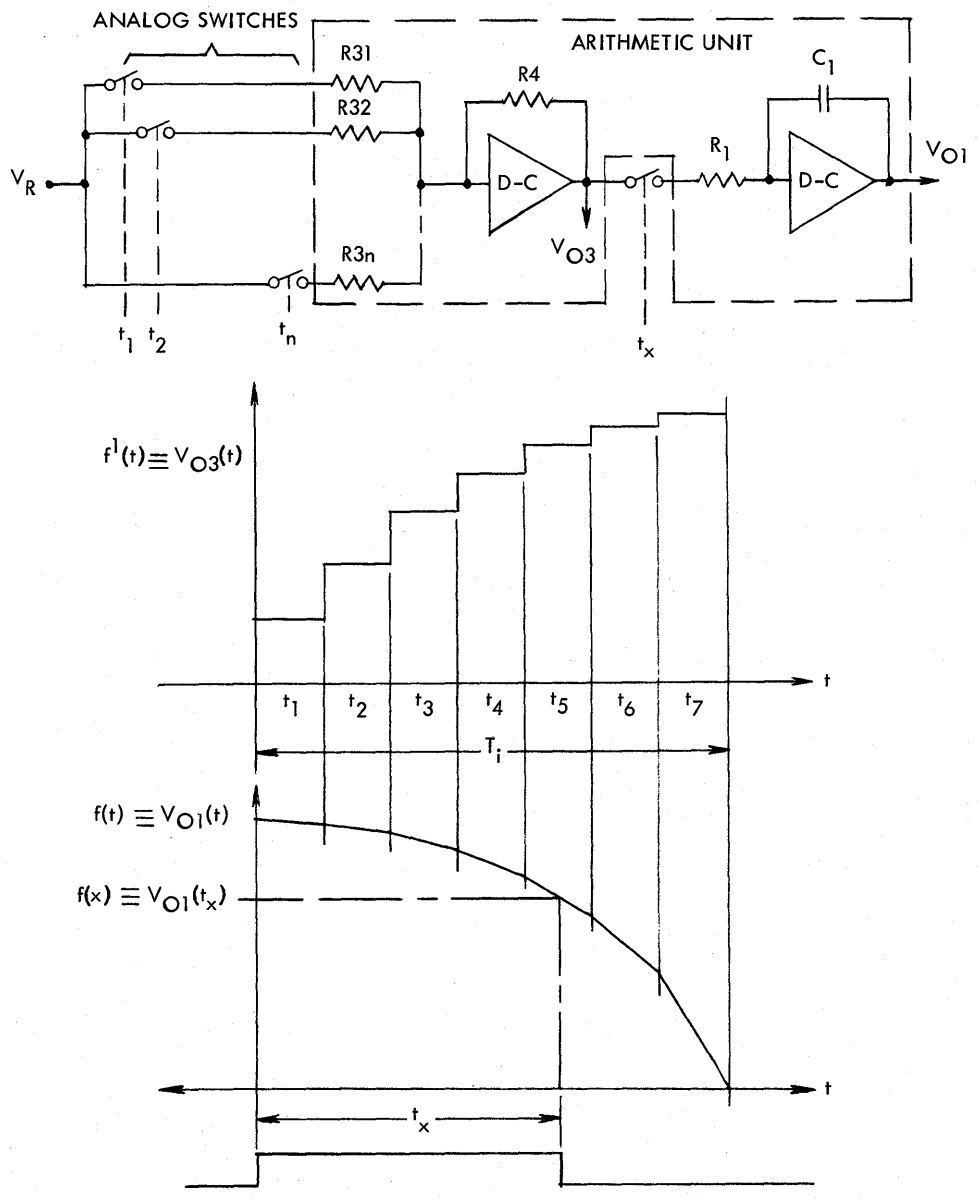



Figure 9. Linear-segment function generator and waveforms.

The generation of inverse function can be accomplished by setting the integrator to $V_x$ and then integrating a voltage function $g'(t)$ until $V_{01}$ is zero. The time required to reduce $V_{01}$ from $V_x$ to zero is the desired inverse function, since:

$$V_x - \int_o^{t_x} g'(t)dt = V_x - g(t_x) + g(o)$$

or, if $g(o) = 0$

$$t_x = g^{-1}(V_x)$$

*Coordinate Rotations/Transformations and Trigonometric Function Generation*

The electronic analog resolver[13] rotates and transforms coordinates and generates trigonometric equations by controlling the initial conditions and the operating time of an harmonic oscillator (Fig. 10).

Two integrators and one inverter are connected into a loop as an harmonic oscillator to solve the differential equations $\dot{X} = -kX$. The outputs of the integrators, $V_{01}(t)$ and $V_{02}(t)$, are the solutions to the differential equations and represent the components

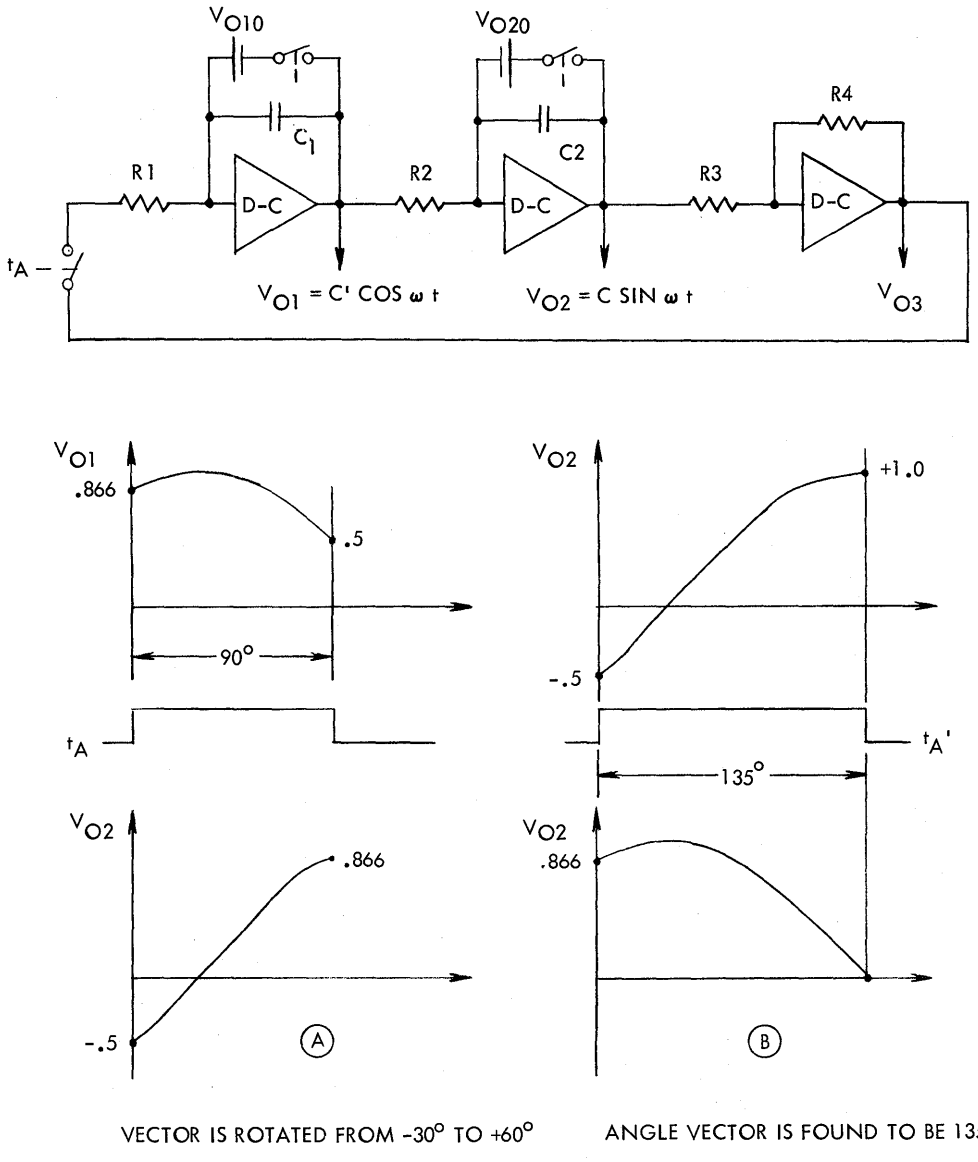



Figure 10. The controlled harmonic oscillator and waveforms.

of the imaginary vector R.

The vector R "rotates" with constant velocity when the harmonic oscillator loop is closed and the integrator outputs change in a sinusoidal fashion. The time during which the loop is closed is directly proportional to the angle through which R is rotated, since $A = kwt$.

Coordinate rotation can be performed by rotating $\overline{R}$ from its initial components $V_x$, $V_y$ for a time $t_A$, which is proportional to the desired angle of rotation. The integrator voltages at the end of $t_A$ represent desired outputs, since

$$V_{01}(t_A) = V_x \cos t_A + V_y \sin t_A$$

$$V_{02}(t_A) = V_y \cos t_A - V_x \sin t_A.$$

Coordinate transformation can be performed by rotating $\overline{R}$ from its initial components $V_x$, $V_y$ until $V_{02}(t)$ becomes zero. The time required for $V_{02}(t)$ to decrease from $V_y$ to zero is:

$$t_A' = \arctan (V_y/V_x).$$

The value of $V_{02}$ at the time $t = t_A'$ is:

$$V_{02}(t_A') = (V_x^2 + V_y^2)^{1/2}$$

The initial components $V_x$, $V_y$ are SET into the integrator prior to the rotation or transformation operation.

Modification of the basic rotation and the transformation equations permits the generation of sine,

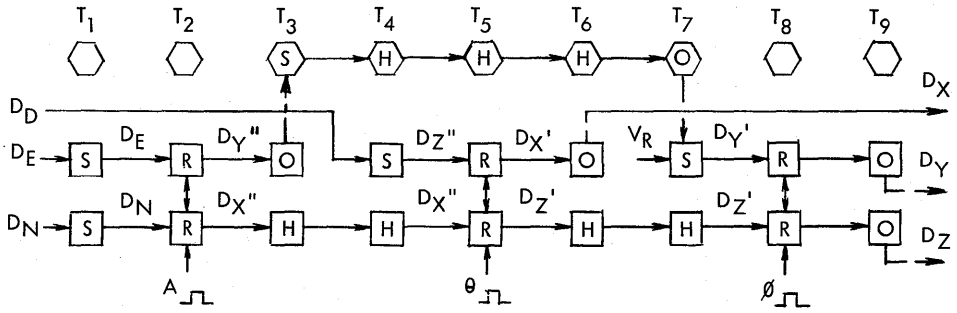
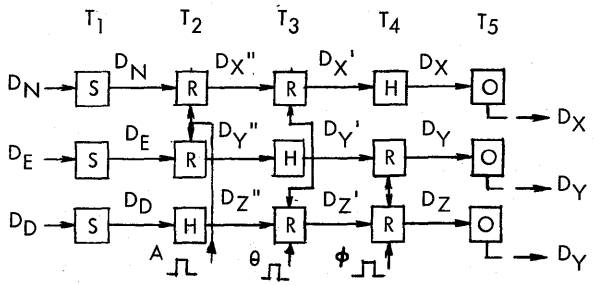



Figure 11a. Flow diagram with arithmetic unit in figure 2.




FLOW DIAGRAM LEGEND:

□ = INTEGRATOR & COMPARATOR

⬡ = MEMORY ELEMENT

Figure 11b. Flow diagram with arithmetic unit having three integrator-comparator combinations.

cosine, arcsine, arccosine, and other trigonometric functions. Similarly, the solutions to the differential equations $\ddot{X} = +kX$ and $\dot{X} = -kX$ can be exploited to generate exponential, logarithmic and hyperbolic equations.

## APPLICATION

The application of the SADC to the coordinate conversion problem demonstrates the elegance of this method of computation and emphasizes most of the advantages.

Conversion from earth coordinates to ship coordinates, as frequently used in navigation, ground control of missiles, fire control, etc., is defined by the matrix equation:

$$\begin{bmatrix} D_x \\ D_y \\ D_z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\phi \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos A & \sin A & 0 \\ -\sin A & \cos A & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} D_N \\ D_E \\ D_D \end{bmatrix}$$

which can also be written as:

$$D_x = (D_N \cos A + D_E \sin A)\cos\theta - D_D \sin\theta$$

$$D_y = (D_E \cos A - D_N \sin A)\cos\phi + [(D_N \cos A + D_E \sin A)\sin\theta + D_D \cos\theta]\sin\phi$$

$$D_z = (D_N \sin A - D_E \cos A)\sin\phi + [(D_N \cos A + D_E \sin A)\sin\theta + D_D \cos\theta]\cos\phi$$

where $D_x$, $D_y$, $D_z$ are ship coordinates; $D_N$, $D_E$, $D_D$ are earth coordinates; and $A$, $\theta$, $\phi$ are azimuth, pitch and roll angle, respectively.

The programming of the SADC is best illustrated with the flow diagrams in Fig. 11, in which each column of squares represents the arithmetic unit and the required memory elements in one operation interval. Inputs to and outputs from a computing element and its function are explicitly indicated: S = SET initial condition, O = READ OUT, R = ROTATION, H = HOLD. A resolver operation is depicted by interconnecting two squares.

With the coordinate inputs $D_N$, $D_E$ and $D_D$ in a-c or d-c voltage form and the angles $A$, $\theta$ and $\phi$ in pulse-time form, the coordinate conversion problem can be solved in two ways.

*Solution 1*: With the arithmetic unit in Fig. 1 and one memory element, the problem can be solved in nine operation intervals $T_i$.

*Solution 2*: With an arithmetic unit consisting of three integrator-comparator combinations, the problem can be solved in five operation intervals $T_i$, and without the use of any memory element.

### *Coordinate Converter Circuit*

A SADC solving the coordinate conversion problem according to Solution 2 consists, as shown in Fig. 12, of the arithmetic unit with three integrator-comparator combinations, the 15 analog voltage switches, the timing generator, and the control logic.

It is assumed that the coordinate inputs $D_N$, $D_E$ and $D_D$ are d-c voltages and the angle inputs are in pulse-time form. The output signals $D_x$, $D_y$ and $D_z$ appear in pulse-time form.

The required number of analog voltage switches and digital logic circuits can be derived most easily from a list of input signals which must be connected to the integrators and inverter in the arithmetic unit during each of the five time periods $T_i$.

| Timing Periods | Integrator I | Integrator II | Integrator III | Inverter |
|---|---|---|---|---|
| $T_1$ | $D_N$ | $D_E$ | $D_D$ | — |
| $T_2$ | $V_{04}$ | $V_{01}$ | — | $V_{02}$ |
| $T_3$ | $V_{04}$ | — | $V_{01}$ | $V_{03}$ |
| $T_4$ | — | $V_{04}$ | $V_{02}$ | $V_{03}$ |
| $T_5$ | $V_R$ | $V_R$ | $V_R$ | — |

The circuit in Fig. 12 can be built with approximately 25 digital integrated circuits (flip-flops and NORs) and 25 linear integrated circuits (amplifiers and analog voltage switches). It would require less than ten watts of power, less than 100 cubic inches in volume, and weigh less than five pounds.

To perform the same problem with a conventional analog or a conventional digital computer would require a circuit complexity at least one order higher.

## PERFORMANCE

The performance of any analog computing element is always a function of the quality of the com-

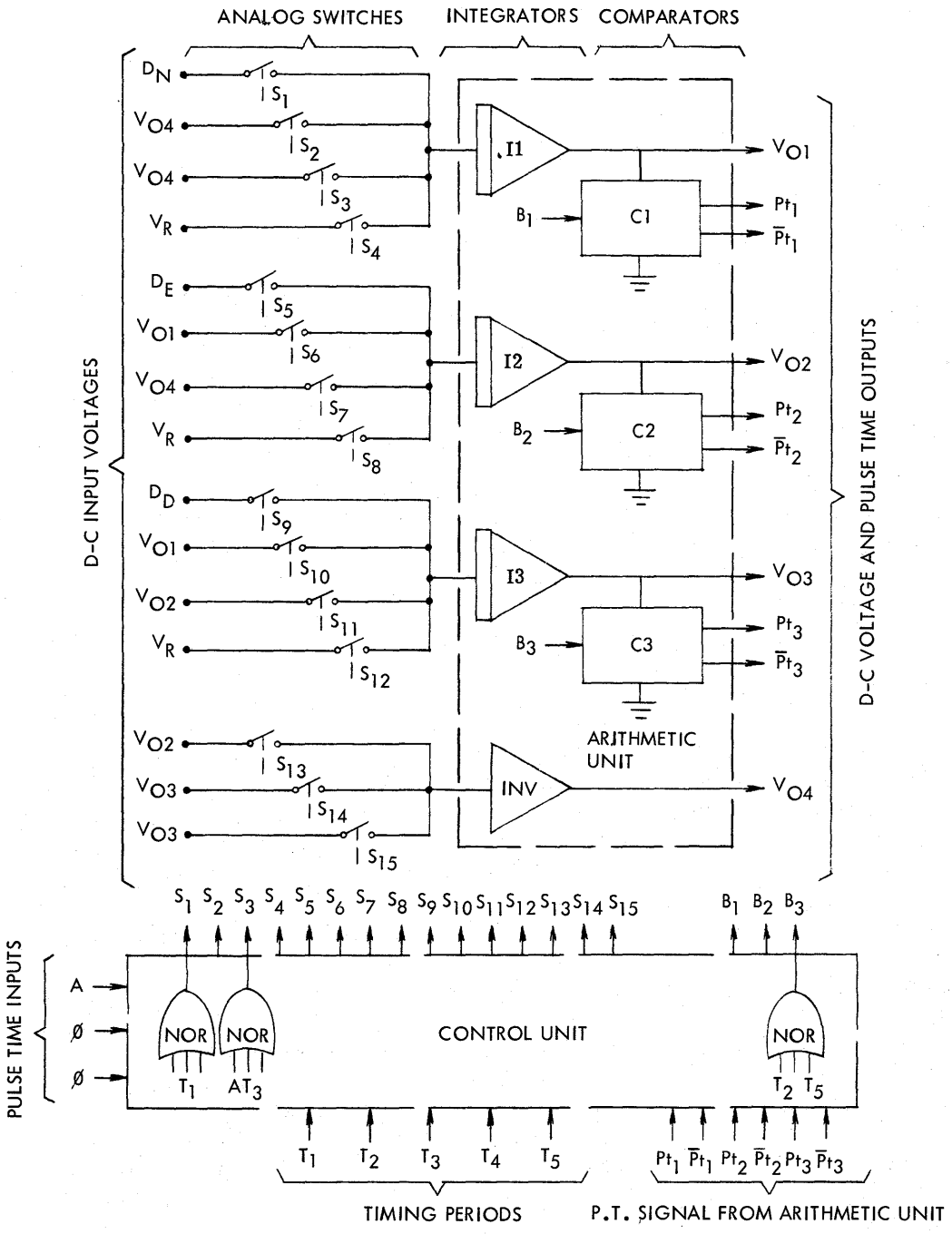



Figure 12. Earth-to-ship coordinate converter.

ponents used and a compromise between static accuracy and speed of operation.

Accuracy and speed of the SADC is solely determined by the quality of the analog computing elements, whereas resolution and dynamic range are largely functions of the number of stages in the memory counters and the length of the operation interval.

Accuracy is also dependent on the variations in temperature and power supplies. With higher quality components, these effects can be made smaller. However, for the few, small components involved,

it is cheaper to control the environment of the arithmetic unit.

The performance data given below refer to an arithmetic unit with the following components, timing and environment:

| | |
|---|---|
| Amplifiers: | Fairchild $\mu$A702 adjusted for zero offset |
| Analog Switches: | Direct-coupled 2N2432 |
| Resistors: | $\pm 0.05\%$ of nominal value |
| Capacitors: | Trimmed to $\pm 0.05\%$ of nominal value |
| Temperature: | 25°C ± 2°C |
| Operation Interval: | 1 ms |
| Power Supplies: | –6V ± 0.1%, +5V ± 10%, +12V ± 1% |

Static accuracy for addition, multiplication, division,* integration, SET, READ and transfer operations is $\pm 0.1\%$ of full-scale and $\pm 0.2\%$ for coordinate rotation, coordinate transformation,* sine-cosine generation, etc.

With a 1-ms operation interval, 1,000 sequential operations can be performed in 1 second.

## CONCLUSION

The ability of computing analog and storing digital provides the SADC with the unique capability of operating as a sequential computer which accepts analog signals as inputs and provides analog signals as outputs, and operates with analog computer accuracy and sequential digital computer speed.

Time-sharing the simple arithmetic unit and requiring no interface circuits make the SADC the most economical method of computation in its speed-accuracy domain.

Due to the small number of components used and due to the fact that almost all of these components are integrated circuits, SADC has an inherently high reliability. In addition, redundancy techniques can be applied to SADC just as to a digital computer.

The performance figures given have been obtained with relatively low-performance integrated analog circuits. With better components, such as chopper-stabilized amplifiers, higher precision will be possible. However, in achieving and maintaining this higher accuracy, there may be a problem in finding accurate and stable capacitors.

To date, only parts of the SADC have been built and tested, and considerably more work is needed — both at the circuit and system levels.

With its inherently high reliability, minute size, low power consumption and minimum cost, the SADC should be well suited for all those military and industrial control applications where the computer inputs and outputs must be in analog form.

*Only when inputs are larger than 50 percent of full-scale.

## REFERENCES

1. G. A. Korn, *Electronic Analog Computers,* McGraw Hill, New York, 1956.

2. S. Fifer, *Analog Computation,* Volumes 1 to 4, McGraw Hill, 1961.

3. R. Lee and F. Cox, "A High-Speed Analog-Digital Computer for Simulation," *IRE Transactions on Electronic Computers,* June 1959, pp. 186-196.

4. A. Herzog, "Pulsed Analog Computer for Simulation of Aircraft," *Proc. IRE,* May 1959, pp. 847-851.

5. E. V. Bohn, "A Pulse Position Modulation Analog Computer," IRE Transactions on Electronic Computers, June 1960, pp. 256-261.

6. M. Phister, *Logical Design of Digital Computers,* John Wiley and Sons, 1959.

7. R. S. Ledley, *Digital Computer and Control Engineering,* McGraw Hill, 1960.

8. W. R. Seegmiller and E. C. Underkoffler, "Static Sync Drive Development," General Electric Company TIS Report R61APS47 (Dec. 1961).

9. H. Ruegg, "An Integrated FET Analog Switch," Solid-State Conference, Philadelphia, Pa. (Feb. 1964).

10. No author, "Molecular Opto-Electronic Multiplex/Chopper," Texas Instruments, Product News Release (March 23, 1964).

11. H. Schmid, "Four-Quadrant All-Electronic Pulse-Time Multiplier," General Electric Company TIS Report 62APJ43, pp.11-13.

12. H. Schmid, and B. Grindle, "Inexpensive Pulse-Width Modulation," *Electronics,* pp. 29-31, Oct. 11, 1963.

13. H. Schmid, "Electronic Analog Resolver," to be published in *Electronics.*

14. H. Schmid, "Repetitive Analog Computing Technique," Aerospace Conference, Phoenix, Arizona, April 1964.

15. H. Schmid, "Linear Segment Function Generator," *IRE Transactions on Electronic Computers,* Dec. 1962, pp. 780-788.

# DESIGN OF A HIGH SPEED DDA

Mark W. Goldman
*Guidance and Control Department*
*Martin Company*
*Baltimore, Maryland*

## INTRODUCTION

The objective of the company-funded task which supported this work was to develop techniques for high-speed solutions to differential equations, particularly those which are common in aerospace problems. For example, the solution requirements for reentry guidance are very time-limited and must be processed at the highest priority level. To solve this type of problem, depending on the accuracy required, the number of iterations can get unreasonably large and require an inordinate amount of computer "power." Therefore, the solution time requirements led us to investigate means other than the general purpose computer to solve these time-critical differential equations. The nature of the problem and the aerospace requirements of long term drift stability and accuracy led us to choose the digital differential analyzer (DDA) as one of the candidates for investigation. This paper, then, is concerned with the new techniques in DDA design which were developed in order to meet the solution time objectives.

Our work on DDA's centered about the Chapman predictive reentry guidance equation. This equation was chosen as a vehicle for study, first, because it is representative of a class of aerospace second order nonlinear differential equations and secondly, because the solution time of predictive reentry equations, particularly near impact, are very critical. One of our earlier DDA designs required approximately 200 seconds to solve the entire Chapman equation from entry to terminal; the new design can solve the same problem in 0.56 seconds—using less hardware and at the same accuracy. These new techniques are:

1. *Shared integrators.* A method of combining several (5 in one case) integrators which contain the same variable, and sharing the Y register and adder network with a multiple bank of R registers.
2. *Automatic rescaling.* Instead of scaling the DDA to handle the worst case range of variables, it is scaled to handle a nominal range. When a variable exceeds the range, certain critical integrators are rescaled to modify the increment size of the variable. Since the iteration time of a DDA is dependent on the variable size, this technique also allows the DDA to run at the optimum speed at each region of its solution.
3. *Asynchronous timing.* No fixed clock is used in the design; rather, as each iteration is completed the next one is started. For

example in the Chapman design, the longest critical string of integrators is nine, but on the average only four will overflow in any one iteration cycle. In the asynchronous mode the next iteration cycle is started when the longest critical path has been completed.

In addition, advanced logic techniques are used throughout. For example, carry look-ahead adders —each iteration cycle time is dependent on how many integrators overflow, which in turn is dependent on the time that it takes the integrator adder to detect this overflow and complete the addition. The carry look-ahead adder can add two operands in as little as five gate delays and is completely independent of carry. The gate delays for the microelectronics we are using result in an add time of 50 to 90 nsec.

In order to demonstrate the feasibility of the new features it was not necessary to build the complete DDA to solve the Chapman equation. To do so would have risked hiding the principles of the new features in the complexity of the Chapman equation. Instead, each of the new features is amply demonstrated in a DDA designed to solve the simples equation, $xy'' + \frac{1}{2}y' + y = 0$. This simpler equation also has the advantage of having a closed form solution for ease of demonstration. This computer is capable of approximately two million iterations per second.

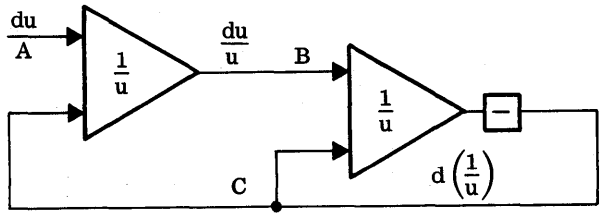
Therefore, the design goal of the project was to study the systems and logical organization of DDA's and to develop a machine with a significant speed increase over contemporary designs. Since speed was the target, economics of the design was deemphasized. That is, where cost and speed were in conflict, the faster but costlier technique won. In some cases, the speed return may not have been worth the added cost, but until a specific application is to be designed, no yardstick is available to measure the economic worth of one design against another. When weight and size become important, as they often do in aerospace equipment, then the yardstick is available to tailor down the design to meet the constraints.

## NEW TECHNIQUES

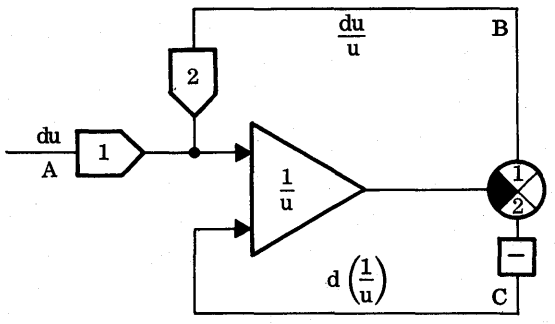
During the system's study, several improvements were made over contemporary DDA's. Two of these, shared integrators and automatic rescaling, are an extension to the theory, while the third, asynchronous timing, is an improvement in the construction.

### *Shared Integrators*

When two or more integrators store the same variable, they can be combined into one integrator with a corresponding saving in hardware. An example of the shared technique is shown in the following sketches. Sketch (a) shows the conventional DDA hookup to obtain reciprocals, in this case $\frac{1}{u}$. Both integrators store the variable $\frac{1}{u}$ (in the Y register). Combining the two integrators, sketch (b) provides the shared integrator which also yields $\frac{1}{u}$. The equivalents between sketches (a) and (b) are illustrated by the flow of pulses through each figure. A du pulse travels along path A. The du pulse generates a $\frac{du}{u}$ output pulse which travels along path B. In turn, the $\frac{du}{u}$ pulse generates a $d\left(\frac{1}{u}\right)$ output pulse which travels along path C.

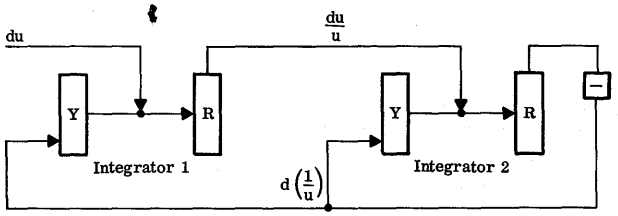


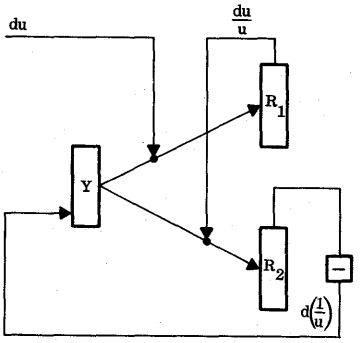

(a




(b)

A more detailed comparison is illustrated below by a block diagram of each integrator. The Y registers in Integrators 1 and 2 contain the same numbers, but not the R registers. Therefore, the shared integrator in sketch (b) has two R registers. Register $R_1$ is identical with the R register in Integrator 1 and $R_2$ identical with the R register in Integrator 2.




The shared integrator above requires 1 adder, 1 Y register and 2 R registers where the conventional DDA hookup requires 2 adders, 2 Y registers and 2 R registers.

*Automatic Rescaling*

A major disadvantage of a conventional DDA is that it uses fixed point arithmetic. This is exemplified by the fact that the scaling is based upon the maximum value that each variable can assume. If some of the variables very over a large range, then an extremely small independent increment may be necessary to maintain accuracy. As the size of the independent increment decreases, the number of iterations increases. A GP computer could solve the same problem with the same accuracy using a larger independent increment if it employs floating point arithmetic. If the GP computer were restricted to fixed point arithmetic, then it would have fundamentally the same scaling problems encountered by the DDA. To overcome the scaling problems of fixed point arithmetic, we have developed several techniques.

The first technique, referred to as multiple scales, is a compromise between fixed point and floating point arithmetic. The procedure is to divide the complete range of the variables and scale each subdivision separately. The individual scales are combined into one computer. The DDA computes with the scale corresponding to the subdivision which the variable size happens to be in at the moment. When the variable size changes to a different subdivision, the DDA automatically switches to the corresponding scale. Each scale uses fixed point arithmetic, but the switching from scale to scale as the variable size changes simulates a floating point method.

The following is a decimal example demonstrating how the DDA switches from one scale to another. The problem is to compute the reciprocal of u where the desired range of u is

$$0.02 \leq u \leq 1.000.$$

Therefore, the complete range of $\frac{1}{u}$ is

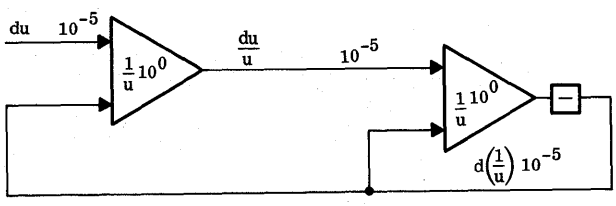
$$0.001 \leq \frac{1}{u} \leq 50.$$

Consider dividing the range of $\frac{1}{u}$ into two parts:

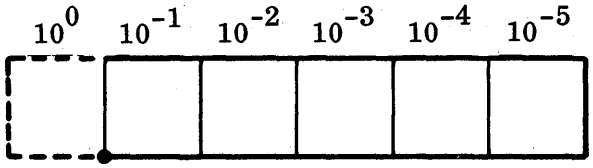
$$0.001 \leq \frac{1}{u} < 1$$

$$1 \leq \frac{1}{u} \leq 50.$$

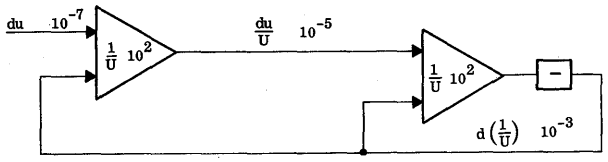
When $\frac{1}{u}$ is in the range from 0.001 to 1 we use Scale A. When $\frac{1}{u}$ is in the range from 1 to 50 we use Scale B. Suppose we decide that we want to carry 5 decimal places, then from DDA scaling theory, in Scale A each du, $\frac{du}{u}$ and $d\left(\frac{1}{u}\right)$ pulse equals $10^{-5}$ as shown below.



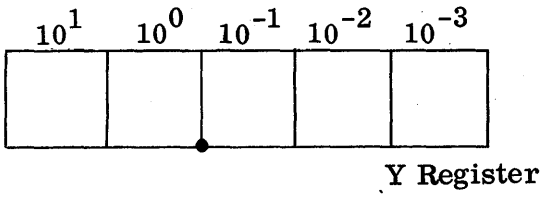
Then from the normal DDA scaling techniques[3] both Y registers are 5 digits and $10^0$ inside each triangle means the decimal points are at the extreme left as shown below.




For Scale B, since we wish to maintain the same number of decimal places as before, each du pulse must equal $10^{-7}$, $\frac{du}{u}$ becomes $10^{-5}$, and $d\frac{1}{u}$ becomes $10^{-3}$ as shown below.



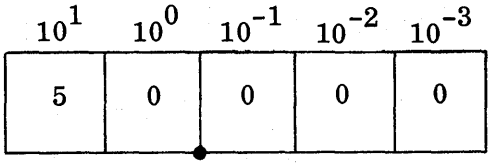

As in scale A both Y registers are 5 digits long. The $10^2$ inside each triangle means the decimal points are two digits to the right as shown below.



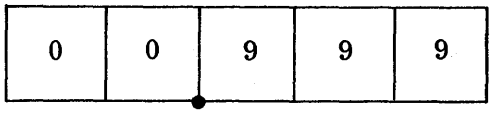

Incidentally, the values of the scaling constants of Scale B are those that would be used for the complete range in a convential DDA.

An example will show how the DDA automatically switches from one sale to the other.

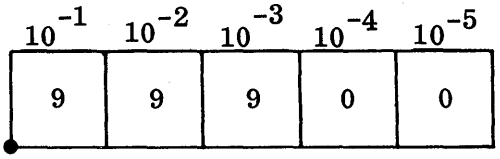
If $\frac{1}{u} = 50$ at the start of the computation, then Scale B is used and the Y register reads




As the computation proceeds the number in the Y registers decreases. The number will eventually reach the value

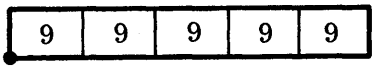



At this point $\left|\frac{1}{u}\right| < 1$ and hence in Scale A. The change to Scale A is sensed by detecting the zeros in the two leftmost positions. The scale switching is accomplished by shifting the number two places to the left (and the imaginary decimal point also) so it reads

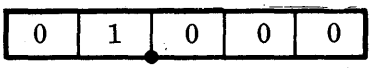



and the R register is reset at 0.5 (its median value) and the DDA is started again.

If the computer happens to be in Scale A and the number in the Y register is increasing, then the Y register will eventually read




The next increase (by $1 \times 10^{-5}$) will cause the Y register to overflow. The new value of Y would be 1 which is in Scale B. The overflow triggers the scale switch from A to B which sets Y register at




and resets the R register at 0.5. The number of iterations required to compute the reciprocal of u from u = 0.02 to u = 1000 using Scale A for the complete range is

$$10^7 (1000 - 0.02) \cong 1.0 \times 10^{10}.$$

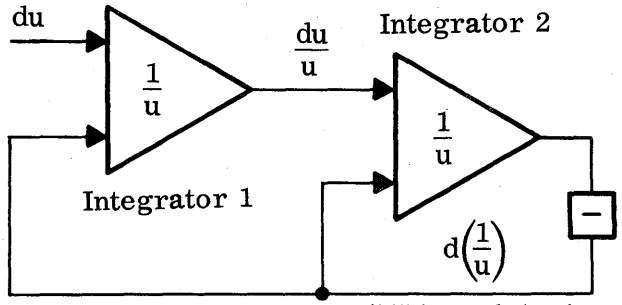
Using multiple Scales A and B, the same accuracy is obtained and the number of iterations is

$$10^7 (1 - 0.02) + 10^5 (1000 - 1) \cong 1.1 \times 10^8.$$

Another technique being considered is to actually use floating point arithmetic. We are presently exploring several methods for mechanizing a DDA to operate with floating point numbers. Some of the ideas are as follows. The decimal (or binary) point of the Y register is no longer predetermined, and special registers are needed to specify the position of the decimal point. Special control logic will solve the scaling equations for each iteration and perform the scaling as the DDA is computing. This technique shows a great amount of promise.

*Asynchronous Timing*

Asynchronous timing is a method of decreasing the iteration time. In conventional DDA's the iteration time is constant because a fixed clock is used (synchronous timing). In contrast, asynchronous timing means the next iteration is started as soon as the last one is finished. The DDA hookup for obtaining the reciprocal of u will again be used to explain the principle. Suppose the time required to process any input pulse is T. At the start of each iteration a du pulse triggers Integrator 1 (see following sketch). If the R register does not overflow, then no output pulse occurs and the iteration is finished in time T. If an output pulse does happen to occur then Integrator 2 is triggered. If the R register of Integrator 2 does not overflow, then no output pulse from it occurs and the iteration is finished in 2T. If an output pulse from Integrator 2 does occur, then the lower inputs to both integrators are triggered, and the iteration requires 3T.




A summary of the three possibilities and the time necessary to perform the calculations in each case is tabulated here.

| Integrator 1 Occurrence of an Output Pulse | Integrator 2 Occurrence of an Output Pulse | Iteration Time Required |
|---|---|---|
| No | No | 1T |
| Yes | No | 2T |
| Yes | Yes | 3T |

For a fixed clock system the iteration time is set for the longest pulse path which in this case is 3T. With the asynchronous timing technique each integrator has a control section which indicates whether the integrator is calculating. The master control examines each integrator, and it will generate the next du pulse each time both integrators become inactive. In this way the iteration time will be 1, 2 or 3T depending on whether none, one, or both R registers overflowed. A good estimate as to the probability a particular R register will overflow is ½. Based on this estimate, the probability of no overflows is ½, of one overflow is ¼, and of two overflows is ¼. Therefore the average iteration time using asynchronous timing would be ½ x IT + ¼ x 2T + ¼ x 3T = (1¾) T which is an improvement over the 3T required for synchronous timing.

## DDA DESIGN FOR CHAPMAN'S EQUATION

In the Introduction we stated our objective was to design a DDA as part of a guidance computer for a lifting re-entry vehicle. This work was based on references 1, 2 and a Martin report, "An Automatic Predictive Guidance Method for Unmanned Lifting Vehicles Re-entering with Circular Velocity" by V. Blaes. The report discusses the equations to be solved by the guidance computer. The computations divide into two parts: subroutine FUTURE and subroutine PREDICT. Figure 1 shows the entire control system which includes a block diagram of both subroutines and their interconnection with the vehicle dynamics. The advantages were in favor of a GP (General Purpose) computer to solve subroutine FUTURE and a DDA to solve subroutine PREDICT. Subroutine PREDICT requires an accurate, high speed solution of a differential equation which is best suited for a DDA.

The purpose of subroutine PREDICT is to calculate the footprint size based on the spacecraft position. By varying the lift-drag ratio *L/D* and bank

angle $\phi$ of the spacecraft, the landing spot can be controlled. The footprint specifies the region of the earth where the spacecraft could possibly land. The idea is to control the spacecraft so the target is in the middle of the footprint. The footprint size is defined by three ranges: $(R_x)_{max}$, $(R_x)_{min}$, and $(R_y)_{max}$ which are calculated from Eqs. (1-4). Equation (1) is Chapman's equation — a second order, nonlinear differential equation.

$$u\,Z'' = Z' - \frac{Z}{u} + \frac{1-u^2}{uZ} - \sqrt{\beta r}\frac{L}{D}\cos\phi \qquad (1)$$

Equations (2), (3), and (4) are used to calculate the X and Y components of the range

$$R_y = \frac{r}{30}\int_{u_f}^{u_i} \frac{\sin\Psi}{Z}\,du, \text{ ft} \qquad (2)$$

$$R_x = \frac{r}{30}\int_{u_f}^{u_i} \frac{\cos\Psi}{Z}\,du \qquad (3)$$

$$\frac{d\Psi}{du} = \frac{1}{u}\left(\frac{L}{D}\right)\sin\phi \qquad (4)$$

While the spacecraft is entering the atmosphere, subroutine FUTURE calculates the input conditions $Z_i$, $Z'_i$, $u_i$ from the vehicle dynamics. It should be

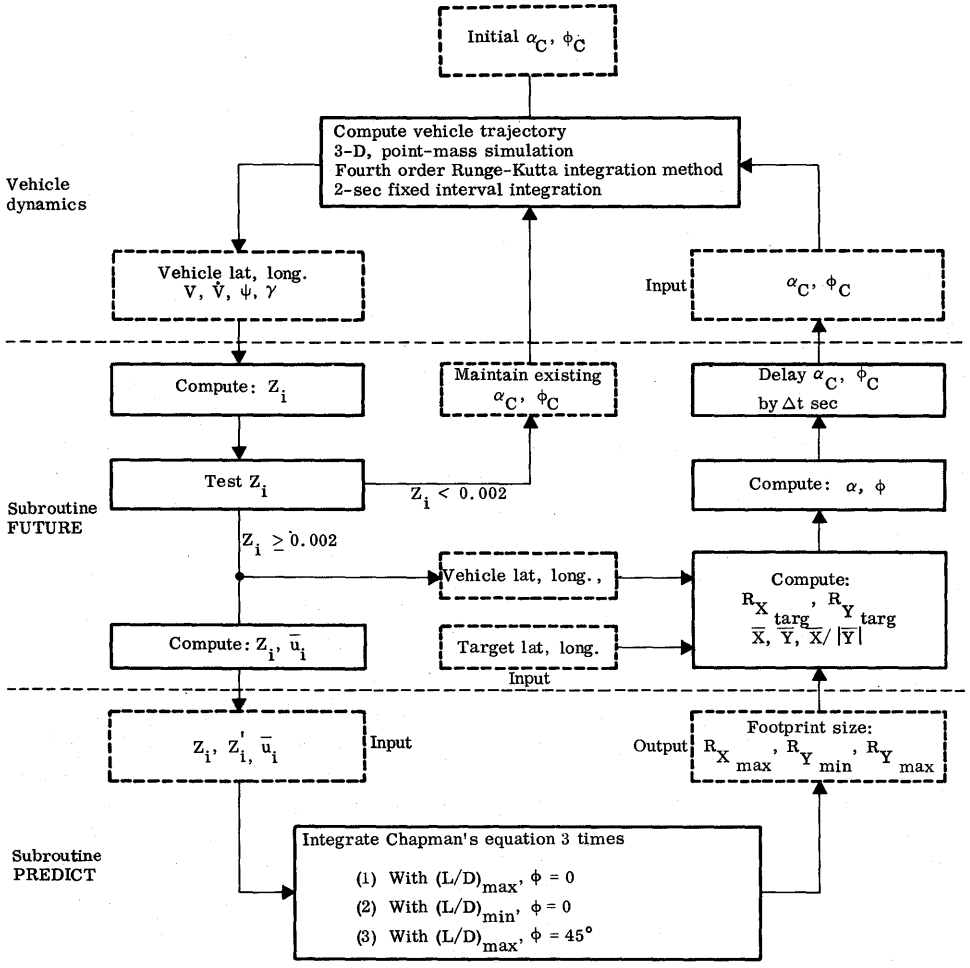



Figure 1. Digital computer simulation — automatic predictor guidance method.

emphasized that $Z_i$, $Z'_i$, $u_i$ are the values of $Z$, $Z'$, $u$ corresponding to the spacecraft's position during the prediction. Suppose the spacecraft's $L/D$ and $\phi$ are known for the entire reentry. If $L/D$ and $\phi$ are substituted into Chapman's equation and $Z_i$, $Z'_i$, $u_i$ are the initial conditions, then the solution of the range equations, together with Chapman's equation, predicts where the spacecraft should land. The miximum downrange $(R_x)_{max}$ is the value of $R_x$ that would be achieved if the spacecraft would maintain $L/D = (L/D)_{max}$ and $\phi = 0$ during reentry. Similarly, the minimum downrange $(R_x)_{min}$ is the value of $R_x$ for $L/D = (L/D)_{min}$ and $\phi = 0$. The maximum crossrange $(R_y)_{max}$ is the value of $R_y$ for $L/D = (L/D)_{max}$ and $\phi = 45°$. As often as possible during the reentry a new prediction is made of the footprint size. Each prediction requires that Chapman's equation and range equations be solved three times in order to obtain $(R_x)_{max}$, $(R_x)_{min}$, and $(R_y)_{max}$.

The DDA design used to solve Chapman's equation and the range equations (subroutine PREDICT) is given in Fig. 2. The square boxes with du inside represent the independent variable increment which starts each iteration cycle. The triangles represent the integrators where the inputs, outputs, and the variables stored in the Y registers are labeled. The arrows indicate the flow of pulses. The pentagons and divided circles are used to represent multiplexed inputs to the shared integrators. Triangles 1, 6, 9 and 10 are shared integrators. The function performed by shared Integrator 1 would normally require four integrators—one for each pentagon. Similarly, shared Integrator 6 is equivalent to four integrators, and Integrators 9 and 10 are equivalent to two integrators each. Consequently, by using shared integrators we have reduced the total number of integrators from 20 to 12.

Integrators 1 through 6 are used to solve Chapman's equation. The outputs from Integrators 2, 3, 4, and 6 are summed together; the resulting sum is

$$Z'du - \frac{Z}{u}du + \frac{1-u^2}{uZ}du - \sqrt{\beta r}\frac{L}{D}\cos\phi\ du.$$

Factoring out du results in the right side of Chapman's equation, Eq. (1). Therefore, the sum of the

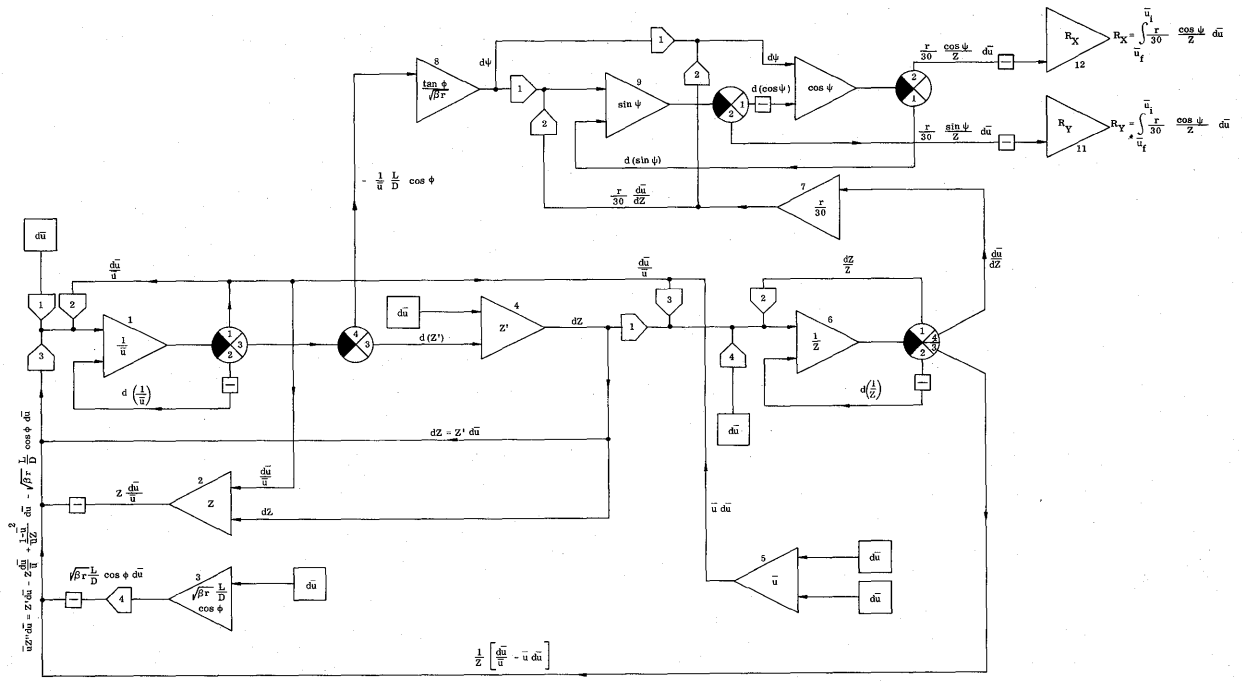


Figure 2. DDA for Chapman equation.

outputs composing input 3 of Integrator 1 is equal to $u\,Z''\,du$:

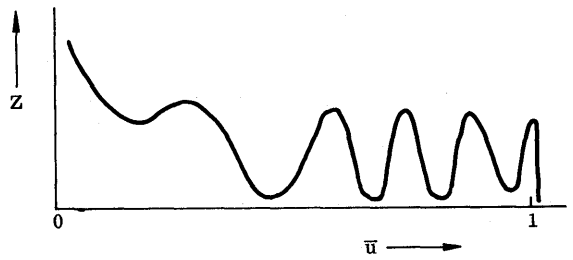
$$uZ''du = \left( Z' - \frac{Z}{u} + \frac{1-u^2}{uZ} - \sqrt{\beta r}\frac{L}{D}\cos\phi \right) du.$$

The change in the heading angle $d\Psi$ is derived by Integrators 3, 1, and 8 in that order according to Eq. (4). The $y$ component of the range is obtained by Integrators 6, 7, 9, and 11 where the solution $R_y$ is in Integrator 11. The $x$ component of the range is obtained by Integrators 6, 7, 10, and 12 where the solution $R_x$ is in Integrator 12.

The DDA for solving Chapman's equation could not be scaled to achieve the accuracy and solution time requirements without introducing the technique of automatic rescaling. For a skipping trajectory the value of $Z$ may vary from 0.0001 to 10. In turn, the value of $1/Z$ in Integrator 6 will vary from 0.1 to 10,000. Because the DDA uses fixed point arithmetic, the large range of $1Z$ requires an extremely small independent increment $\Delta u$. To obtain an accuracy of three decimal places, an increment size of $10^{-8}$ was necessary. Since $u$ varies from 0 to 1, $\Delta u = 10^{-8}$ is equivalent to $10^8$ iterations. The number of iterations is directly proportional to the solution time.

The desired accuracy and solution time were obtained by applying the technique of automatic rescaling. Instead of using one scale for the complete range of $1/Z$, we divided it into four scales: A, B, C, and D. The table below shows the range of $1/Z$, value of $\Delta u$, and number of iterations for each scale. The accuracy for each scale is approximately the same. The following sketch shows a typical plot of $Z$ versus $u$ for a skipping trajectory. From the sketch, Scales A and B are used during most of the integration, and Scales C and D are used less than 10 percent of the time. Therefore, the average $\Delta u$ is about $10^{-6}$. The use of multiple scales has reduced the number of iterations, and consequently the solution time, by a factor of 100.

| Scale | Range of $1/Z$ | $\Delta u$ | Number of Iterations |
|---|---|---|---|
| A | $0.1 \leq 1/Z < 10$ | $10^{-5}$ | $10^5$ |
| B | $10 \leq 1/Z < 100$ | $10^{-6}$ | $10^6$ |
| C | $100 \leq 1/Z < 1000$ | $10^{-7}$ | $10^7$ |
| D | $1000 \leq 1/Z < 10{,}000$ | $10^{-8}$ | $10^8$ |




A test of the accuracy for different $\Delta u$ sizes was made using a simulation program. Figure 3 shows the results of solving Chapman's equation with the same initial conditions but with different sizes of $\Delta u$. The three solid curves are the solutions for $\Delta u = 10^{-4}$, $\Delta u = 10^{-5}$, and $\Delta u = 10^{-6}$. A computer program at the Martin Company, for solving differential equations, entitled Unitrac was used to compute an accurate solution represented by the dotted curve for means of comparison. The curve for $\Delta u = 10^{-6}$ more closely follows the dotted curve than the curve for $\Delta u = 10^{-5}$, demonstrating that an increase in accuracy results as the independent increment size decreases.

## DDA DESIGN FOR DEMONSTRATION PROBLEM, $xy'' + \frac{1}{2}y' + y = 0$

To demonstrate the principles of the new techniques we have designed and built a special purpose DDA to solve $x\frac{d^2y}{dx^2} + \frac{1}{2}\frac{dy}{dx} + y = 0$. Consequently, we did the detailed logic design needed to incorporate automatic rescaling, asynchronous timing and shared integrators into a DDA.

The general solution to $x\frac{d^2y}{dx^2} + \frac{1}{2}\frac{dy}{dx} + y = 0$ is

$$y = A\cos(2\sqrt{x}) + B\sin(2\sqrt{x})$$

where $A$ and $B$ are arbitrary constants. If the initial conditions are

$$y = 100 \text{ and } \frac{dy}{dx} = 0 \text{ at } x = 0$$

then the particular solution is

$$y = 100 \cos (2\sqrt{x})$$

for which a graph is shown in Fig. 4. We confined our interest to computing the solution $y = 100 \cos (2\sqrt{x})$ from $x = 0$ to $x = \frac{9\pi}{4}$, although other solutions could be computed by changing the initial conditions.

The DDA hookup used to solve $x \frac{d^2y}{dx^2} + \frac{1}{2}\frac{dy}{dx} + y = 0$ is shown in Fig. 5. The box with $dx$ inside generates the independent increment which starts each iteration. The figures labeled 1T, 2T, and 3T are counters for the purpose of scaling and will be explained later. The triangles labeled 1I, 2I, and 3I are the integrators storing $\frac{1}{x}$, $\frac{dy}{dx}$ and $y$, respectively. The lower input is the differential of the variable being stored. The output is the product of the upper input and the variable being stored.

Integrator 1I is a shared integrator having one Y register and three R registers labeled $R_1$, $R_2$, and $R_3$. Input 1 to Integrator 1I, $dx$, causes Y to be accumulated in register $R_1$. The overflow from $R_1$, $\frac{dx}{x}$, becomes input 2. Input 2 causes Y to be accumulated in Register $R_2$. The sign of the overflow from $R_2$ is switched; the result $-\frac{dx}{x^2}$ is equal to $d\frac{1}{x}$ and hence becomes the lower input to 1I. Input 3 which comes from Counter 2T is equal to $-\frac{1}{2}\frac{dy}{dx} \cdot dx$, and input 4 which comes from Integrator 3I is equal to $-y\,dx$. Both inputs 3 and 4 result in Y being accumulated in Register $R_3$. So it is possible for Y to be added to $R_3$ twice in the same iteration if both 3I and 2T overflow. Multiplying the equation $x\frac{d^2y}{dx^2} + \frac{1}{2}\frac{dy}{dx} + y = 0$ by $dx$, it is clear that the sum of inputs 3 and 4 is equal to $x\frac{d^2y}{dx^2} \cdot dx$. Therefore the overflow from $R_3$ is $\frac{d^2y}{dx^2} \cdot dx$ which equals $d\left(\frac{dy}{dx}\right)$ by identity.

The next phase is the scaling which determines the increment size corresponding to each input and output pulse and the number of bits in each register. To demonstrate the technique of automatic rescaling, we used two scales. Scale A is for the range from $x = 0$ to $x = 16$, and scale B is for the remaining

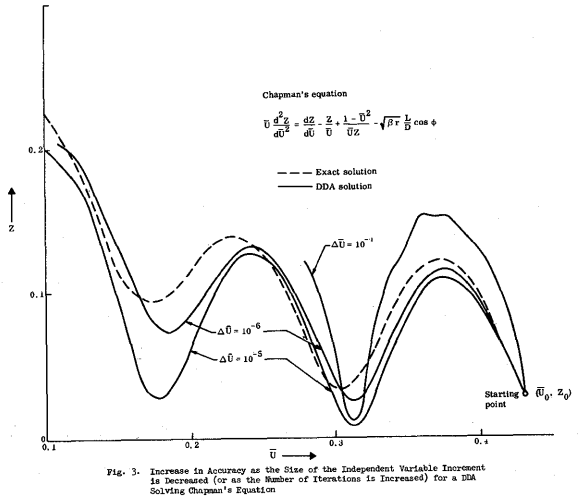



Fig. 3. Increase in Accuracy as the Size of the Independent Variable Increment is Decreased (or as the Number of Iterations is Increased) for a DDA Solving Chapman's Equation

Figure 3. Increase in accuracy as the size of the independent variable increment is decreased (or as the number of iterations is increased) for a DDA solving chapman's equation.

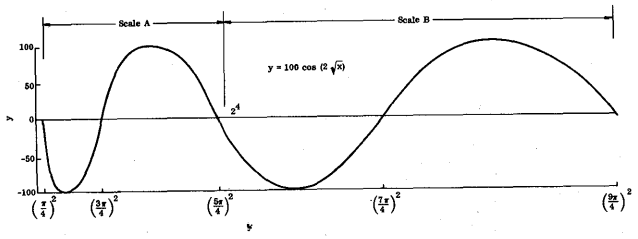



Figure 4. Solution of $XY'' + \frac{1}{2}Y' + Y = 0$

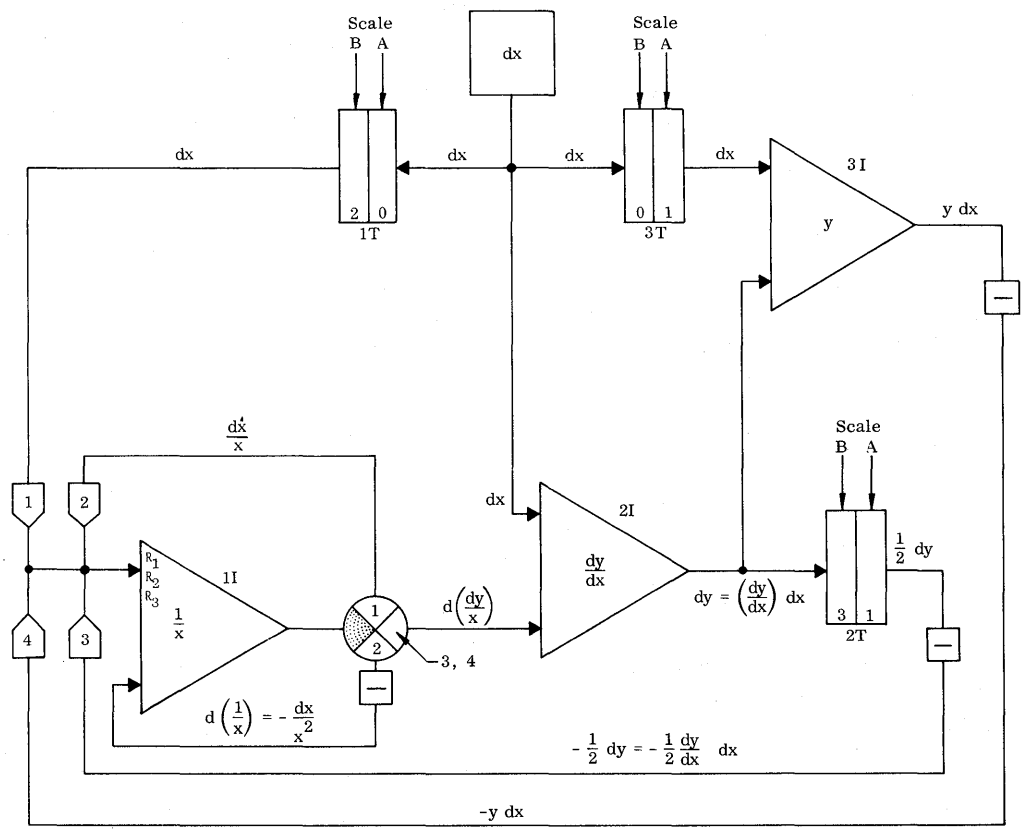
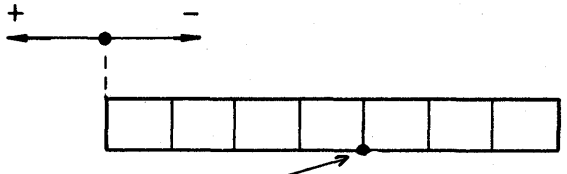



Figure 5. DDA to solve $x\frac{d^2y}{dx^2} + \frac{1}{2}\frac{dy}{dx} + y = 0$

range from $x = 16$ to $x = \left(\frac{9\pi}{4}\right)^2$. The absolute maximums of $\frac{1}{x}$, $\frac{dy}{dx}$, and $y$ plus the scaling coefficients for scale A and scale B are given in Fig. 6. For example, a 5 on a line would indicate that the weight of the variable increment at that point equals $\frac{1}{2^5}$ or it takes $2^5$ pulses to equal one unit of the variable. The position of the binary point in any register can be determined by the number inside the triangle. The following sketch shows the sign convention. The scaling coefficients were purposely chosen so that th length of each register was 7 bits for Scale A and Scale B. The counters, 1T, 2T, and 3T, in Scales A and B are used to decrease the scaling coefficients so that they satisfy the scaling equations.




For example, a $-4$ would place the binary point here

We also employed asynchronous timing in the computer. No further explanation of asynchronous timing is needed here.

Another problem not mentioned so far is sequencing, which is related to the timing. While Integrator 1I is processing input 2, it is possible for inputs 3 and 4 to arrive. The integrator must process each input one at a time in series. To solve this problem, the integrator has been designed to store any input pulse which arrives while the integrator is integrat-

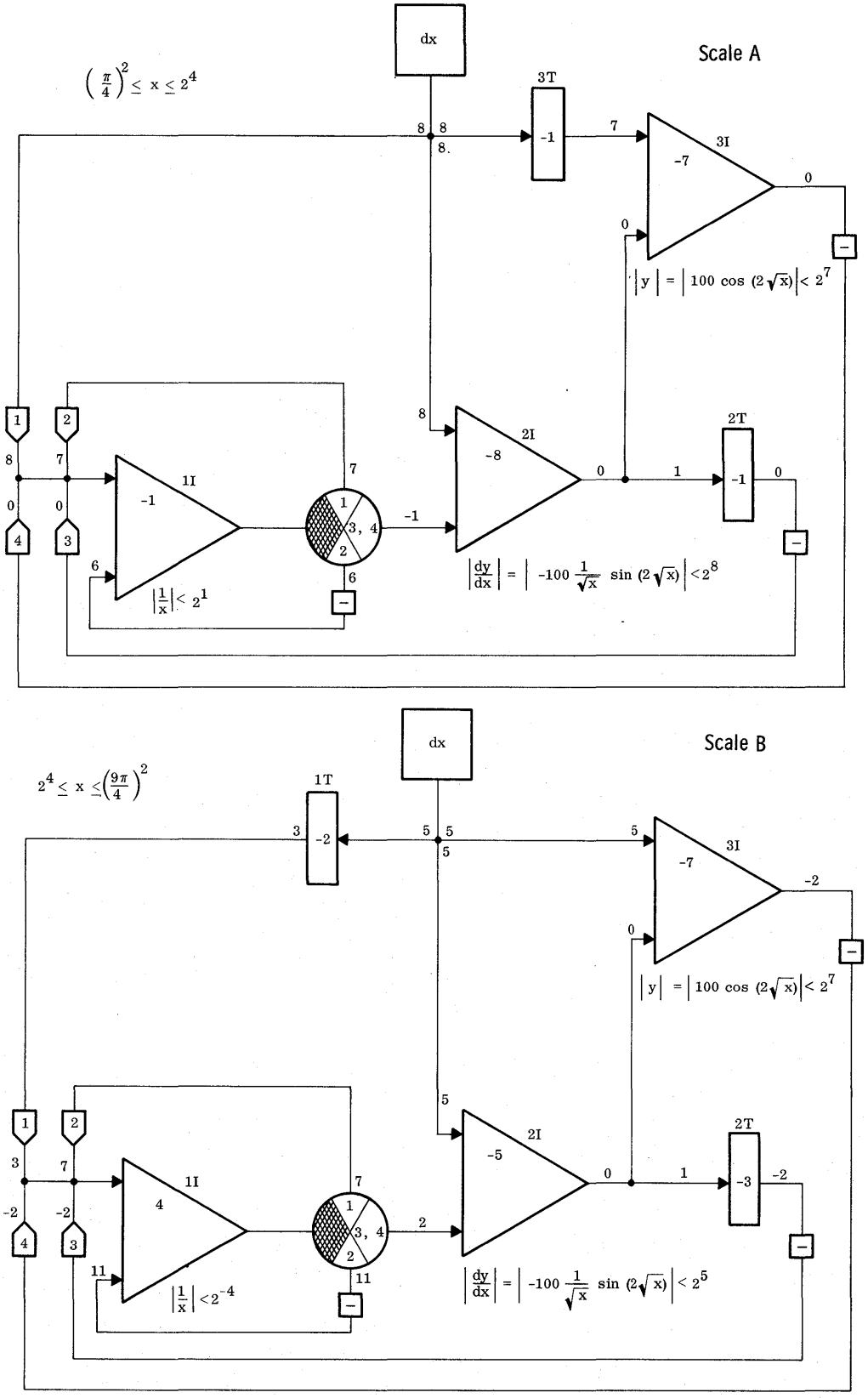
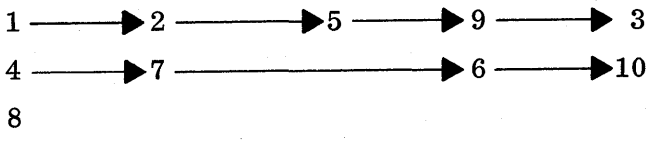



Figure 6. Scale A and B for demonstration DDA.

ing. When the integrator is finished, it will act upon the stored input pulse.

The sequencing problem is best illustrated by examining the pulse paths. Referring to the Sequencing Chart (Fig. 7) there are three basic time zones after a ΔX generator starts the iteration cycle. In zone A, Integrators 1I, 2I and 3I do a ΔX cycle (triangles labeled 1, 4 and 8) and there is no possibility of an integrator time conflict. In zone B, however, there are three pulse paths where Integrator 1I can be actuated in a ΔX cycle; they are pulse path 1 (triangle 2), pulse path 2 (triangle 5) and pulse path 4 (triangle 9). In the worst case, if all integrators in time zone A overflow, then all three requests for 1I in time zone B must be honored. This is done by sequencing the 1I ΔX requests on a first come, first served basis. There is also a time conflict in time zone C with Integrator 2I, pulse path 2 (triangle 6) and pulse path 4 (triangle 10). The worst possible series path of integrators is shown below:



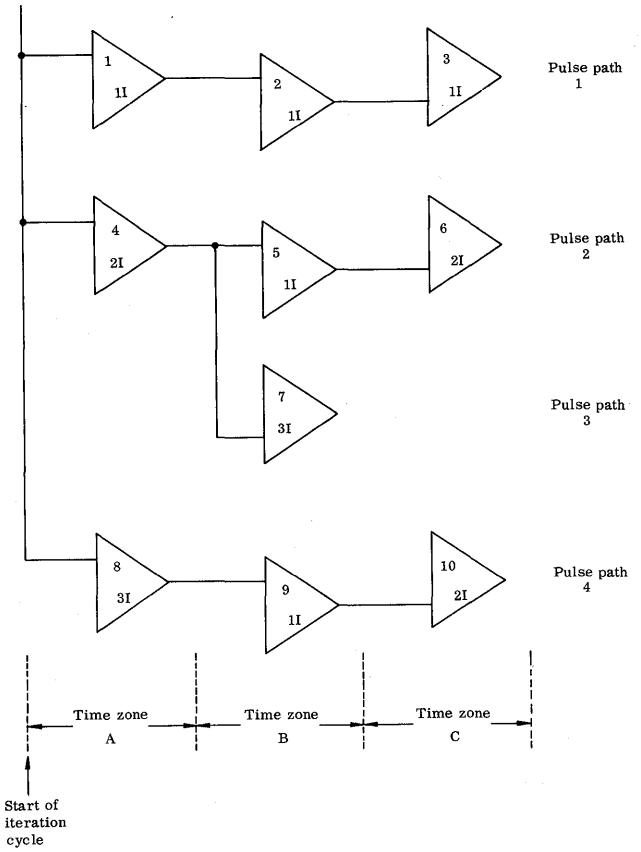




Figure 7. ΔX generator sequencing chart.

The worst path is four ΔX cycles and one ΔY cycle. They are Integrators 1, 2, 5, 9 and 3. The other integrators, Nos. 4, 7, 6, 10 and 8, are processed in parallel and take no added time. For example, while Integrator 1 is processing, Integrators 4 and 8 are also processing. In this description of the pulse paths and parallel processing we have ignored the

influence of the 1T, 2T and 3T counters. The most they can do, however, is decrease the occurrence of the worst case series path. At the completion of all the pulse paths, the ΔX generator starts the next cycle. Since the longest path is five integrators (which occurs infrequently) and the shortest is one (which occurs frequently), this technique of asynchronous timing results on the average in a great time saving.

Test data obtained from the demonstration computer indicates the number of iterations per second to be in excess of 2 million.

## LOGIC DETAILS OF DEMONSTRATION DDA

The logic of the demonstration DDA to solve $xy'' + \frac{1}{2} + y = 0$ can be divided into three sections: Integrators, Counters and Master Timing Control.

### *Integrators*

There are two types of integrators, shared (1I) and normal (2I and 3I) both of which use the normal rectangular integrator technique. To ease the description, we will describe the less complicated normal integrator and note the differences of the shared integrator where appropriate.

The integrators are divided into two sections: (*a*) Register and Adder, and (*b*) Timing and Control.

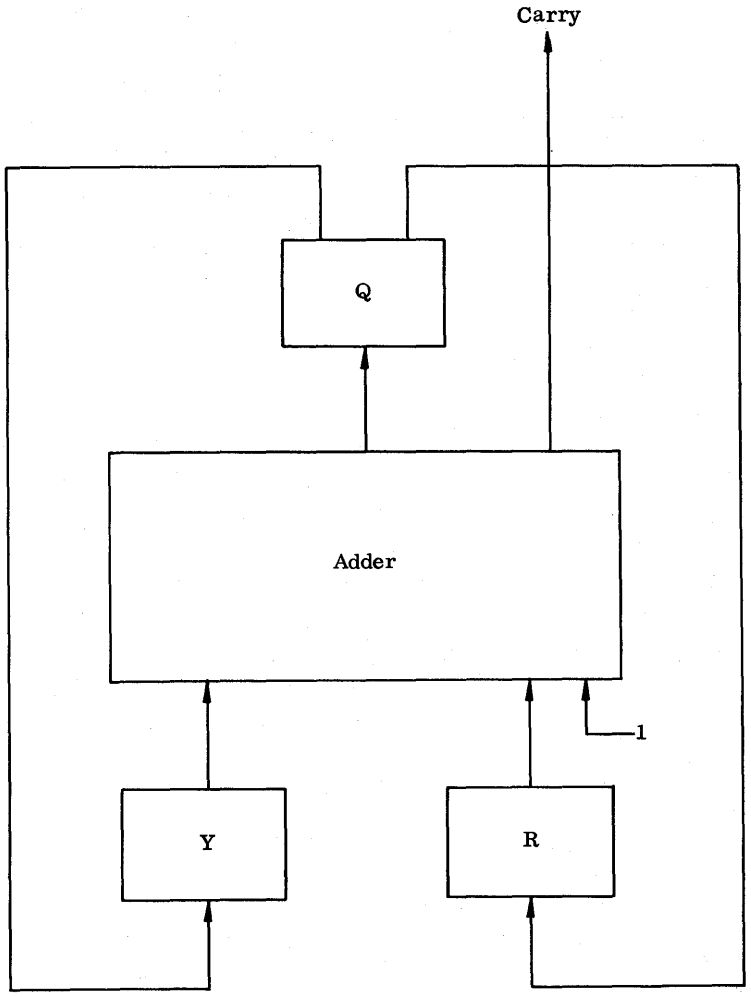



Figure 8. Registers and adder interconnection.

Registers and adder. The functions of the Y and R registers are as normally used in a DDA (see reference 7 for full description of DDA integrators). There is also a Q register which is used as a temporary storage from the adder. The block diagram of the registers and adder is shown in Fig. 8.

The possible register-to-adder transfers are

$$Y + R \rightarrow Q \rightarrow R$$
$$1 + Y \rightarrow Q \rightarrow Y$$

(In the case of a shared integrator, Q transfers to $R_1$, $R_2$, or $R_3$ depending on which $\Delta X$ is in operation.)

Both the R and Q are 7 bit unsigned registers and the Y register is 7 bits plus sign. Subtraction is performed by a modified 1's complement addition plus some sign control and zero detection. The Y and Q registers can transfer out their true value ($Y_T$ or $Q_T$) or their 1's complement value ($Y_c$ or $Q_c$).

The total algorithm for the Y, R and Q registers to the adder is given in Table 1. The symbols used in the table are:

$Y_M \equiv$ Y register magnitude
$Y_S \equiv$ Y register sign
$Y_T \equiv$ Y register true value
$Y_c \equiv$ Y register 1's complement value
$C \equiv$ overflow carry-out of adder
$\overline{C} \equiv$ no overflow carry-out of adder
$\rightarrow \equiv$ transfers to
DN $\equiv$ do nothing
$\equiv$ yields

Let's look at an example (line 4): Suppose the integrator is in the condition $Y_M \neq 0$ (the contents of Y register are not equal to zero), $Y_M \neq$ MAX (the contents of Y register are not equal to all ones or the maximum value) and $Y_S = -$ (the sign of Y register is negative). If the input to the integrator is a $\Delta X +$ cycle (column 1), then from the table it can be seen that: the 1's complement value of Y is added to the value of R and the result stored in Q ($Y_c + R \rightarrow Q$). Then the true value of Q is transferred back to R ($Q_T \rightarrow R$). If an overflow carry from adder position 7 did not exist, then the integrator signals a minus ($\Delta X -$) output (C –OUTPUT) to the next integrator.

In addition to the normal functions of the registers and adder, they are also used to accomplish the rescale function. The rescale is really only a shift left of the contents of the Y register. Rather than put in the shift hardware and set up a special timing sequence it was found to be cheaper and faster to use the adder. This is done by adding a 0 to the contents of Y (which is only a trick to get the output of the adder to contain Y) and transferring the adder output to the shifted rather than the normal position of Q. For example, in Integrator 2I, the output of $S_0$ (the zero stage of the adder) is sent to $Q_4$, $S_1$ to $Q_5$ and $S_2$ to $Q_6$. All lower stages of Q, that is $Q_0$, $Q_1$, $Q_2$ and $Q_3$ are filled with 0's. This results in a shift left 4 of the Y register.

The adder network is a fully parallel 6 delay carry look-ahead adder. (For more details on adders, see references 4, 5 and 6.) In brief this adder is made ripple free because each stage "looks ahead" to the input of all previous stages to determine whether a carry-in will exist. This technique, although extravagant in hardware, is the fastest method for binary addition. In passing, it should be noted that this adder could have been made 1 delay faster (from 6 delays maximum to 5 delays maximum), but the amount of hardware would have been approximately doubled. The gate delays in the microelectronics we are using result in an add time of 50 to 90 nanoseconds.

*Timing and Control*

Each integrator is required to perform two types of operations: a $\Delta X$ cycle, where Y is accumulated to R and a $\Delta Y$ cycle, where Y is incremented or decremented by 1. Control flip-flops in each integrator store the present status of the integrator as well as the input commands ($\Delta X$ or $\Delta Y$). Another series of control flip-flops store the action to be taken by the integrator, based on the decoding of the input commands and the present status.

See Table 1 for the combinations of input commands, present condition of integrator and decoded commands. The input commands are also used to initiate the timing chain. No clocks exist in the timing chain, or anywhere in the machine. Rather, the timing and control is designed so that each integrator only operates on command and will process commands according to a priority sequence as shown in the following tabulation.

TABLE 1

Algorithm for Registers and Adder

| Condition of Integrator | Input to Integrator | | | |
|---|---|---|---|---|
| | ΔX + | ΔX − | ΔY + | ΔY − |
| $Y_M = 0$; $Y_S = +$ | DN | DN | $1 \rightarrow Y$<br>$+ \rightarrow Y_S$ | $1 \rightarrow Y$<br>$- \rightarrow Y_S$ |
| $Y_M = 0$; $Y_S = -$ | DN | DN | $1 \rightarrow Y$<br>$+ \rightarrow Y_S$ | $1 \rightarrow Y$<br>$- \rightarrow Y_S$ |
| $Y_M \neq 0$; $Y_S = +$<br>$Y_M \neq$ MAX | $Y_T + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C + OUTPUT | $Y_c + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C − OUTPUT | $Y_T + 1 \rightarrow Q$<br>$Q_T \rightarrow Y$ | $Y_c + 1 \rightarrow Q$<br>$Q_c \rightarrow Y$ |
| $Y_M \neq 0$; $Y_S = -$<br>$Y_M \neq$ MAX | $Y_c + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C − OUTPUT | $Y_T + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C + OUTPUT | $Y_c + 1 \rightarrow Q$<br>$Q_c \rightarrow Y$ | $Y_T + 1 \rightarrow Q$<br>$Q_T \rightarrow Y$ |
| $Y_S = +$<br>$Y_M =$ MAX | $Y_T + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C + OUTPUT | $Y_c + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C − OUTPUT | 1 OVERFLOW | $Y_c + 1 \rightarrow Q$<br>$Q_c \rightarrow Y$ |
| $Y_S = -$<br>$Y_M =$ MAX | $Y_c + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C − OUTPUT | $Y_T + R \rightarrow Q$<br>$Q_T \rightarrow R$<br>C + OUTPUT | $Y_c + 1 \rightarrow Q$<br>$Q_c \rightarrow Y$ | 1 OVERFLOW |

1. If a ΔX command occurs, and ΔY is not active, process ΔX.
2. If a ΔX command occurs and ΔY is active, then store ΔX command.
3. If a ΔX command is in storage and ΔY goes inactive, then process ΔX and reset ΔX storage.
4. If a ΔX and ΔY command occur at the same time, process ΔX and store ΔY.
5. If a ΔY command is in storage and ΔX goes inactive, then process ΔY and reset ΔY storage.

Notice that in line 4, preference is given to ΔX commands since only they can cause an output from the integrator. In this way, if there is to be an output, the next integrator is started processing sooner than if the ΔY (which never has an output) were processed first.

Each of the ΔX and ΔY command and storage flip-flops from each integrator is also decoded in the Master Timing and Control Section. When all flip-flops are zero, which indicates that no integrator is processing or has anything to be processed, the next ΔX generator or next iteration cycle is started — which is asynchronous timing.

*Counters*

The DDA contains three counters, 1T, 2T and 3T. Counters 1T and 3T are simple decoding counters of 2 bits and 1 bit, respectively, and 2T is a 4-bit double-rank forward and reverse shift counter. In each of the counters, since we cannot afford to wait for ripple time, a simple expediency is used. When a counter is one count below its output value the next count command will trigger the output even though the counter never really increments (or decrements). For example, suppose the counter was designed to overflow or output at a value of 16, when it reaches 15 the next count command triggers the output and resets the counter. In this way, the computer does not have to wait for the counter to increment but only for the time it takes the simple counter control to trigger.

It might appear that, since the counter ripple time has been bypassed as a problem, we could have used the simplest ripple counter, but this does

not work out. It is true that the counter time is important only in as much as it finishes its counting before the next count command is received. Under certain conditions in the DDA, the count commands could be very close together so the faster shift counter was required.

In order to avoid the annoying typical counter problems of two types of zero (plus and minus) and sign control, a simple expediency was used of making the counter one stage too big and resetting to the middle value. For example, if we want to count from 0 to + 7 and 0 to – 7, we make the counter 4 bits (instead of 3) and count from 8 up to 15 and 8 down to 1. The hardware difference between the two techniques is so small as to be unimportant.

*Master Timing and Control (MT&C)*

The Master Timing and Control is made up of a series of housekeeping and control functions which belong to the whole DDA rather than any one integrator. They are: (1) ΔX Generator, (2) Fault detection, (3) Rescale, and (4) Push button starter.

*ΔX Generator (ΔX Gen).* This flip-flop receives the output and status of all integrators to sense when they have completed their last operation and are waiting for another cycle. When this "nothing else to do" condition exists, the ΔX Generator initiates the start of the next iteration cycle. This scheme takes the place of a clock which would run the DDA at fixed clock periods and would have to be timed for the worst case pulse path or slowest possible cycle. By the very nature of a DDA not every iteration cycle will, and in fact many will not, have overflows from some integrators.

*Fault detection.* Certain conditions of the Y decode logic of each integrator are considered as a fault. These conditions, when they occur, are stored in flip-flops in the MT&C and the machine will stop or ignore these faults depending on a switch setting on the operator's console.

The fault conditions are:

- 1I Y register = 0.
- 2I Y register $> 2^7 - 1$ = overflow.
- 3I Y register $> 2^7 - 1$ = overflow.

*Rescale.* Two flip-flops are used to store the rescale conditions of 1I and 2I. When both conditions occur, the next ΔX Generator cycle is replaced by a rescale cycle which resets the initial conditions into the integrators and counters, and at the same time shifts left or right the contents of Y register in 1I and 2I. This shifting is done by using the adder network. At the completion of the rescale cycle, the next ΔX Generator cycle is initiated and the DDA starts its normal operation.

*Push button starter.* This is simply a means of filtering out the "Single-Cycle" push button noise when the DDA is operated in the "Step" mode. In the "Step" mode, the push button intiates the ΔX Gen cycles so that the machine iterates on command.

## HARDWARE DESCRIPTION OF DEMONSTRATION DDA

In the development phase of the DDA program it was unnecessary, and in fact burdensome, to package the computer for an aerospace environment. On the other hand, a "breadboard" package often turns out to mean "slapped together." Neither of these extremes was desirable and the result of this program is what we call a laboratory model.

For the logic hardware, it was advantageous to use aerospace approved circuits. In this way, the laboratry model would have speed and noise characteristics approximating those of a final packaged version. The logic chosen was the Fairchild commercial integrated circuits.

The front and back views of the resultant laboratory model are shown in Figs. 9 and 10. The chassis measures 23″ × 20″ × 5″ (exclusive of the operator's console) and can accommodate 100 logic cards of the type shown in Fig. 11. The connectors are 86 pin AMP tab-wired and require no soldering. The recessed tabs preclude wire shorting and also allow easy removal. Connector power (+ 4 v and ground) is tapped-off the bus bar with all grounds isolated from the chassis and brought back to a central point.

The logic cards measure 4½″ × 2″ and each contains twelve 8-pin, TO-5 header microelectronic logic elements. The only interconnections made on the printed circuit cards are the power bussing. The remaining logic interconnections are made on the wiring backpan (this is often referred to as "backpan logic"). This technique allows greater flexibility and ease of modification, and the logic partitioning is much easier since it can be done at a much later date in the design. There is also a significant cost saving in only requiring one printed circuit design.

The photographs in Figs. 9 and 10 were taken

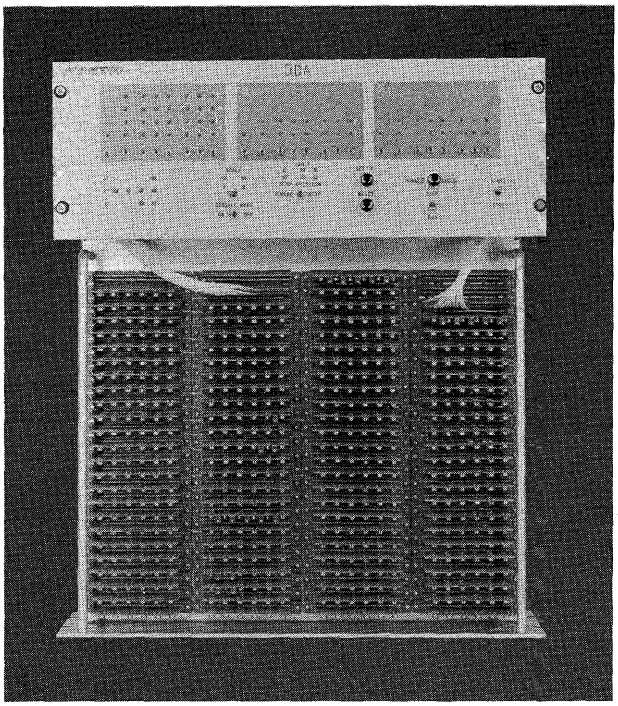


Figure 9. Front view of DDA.

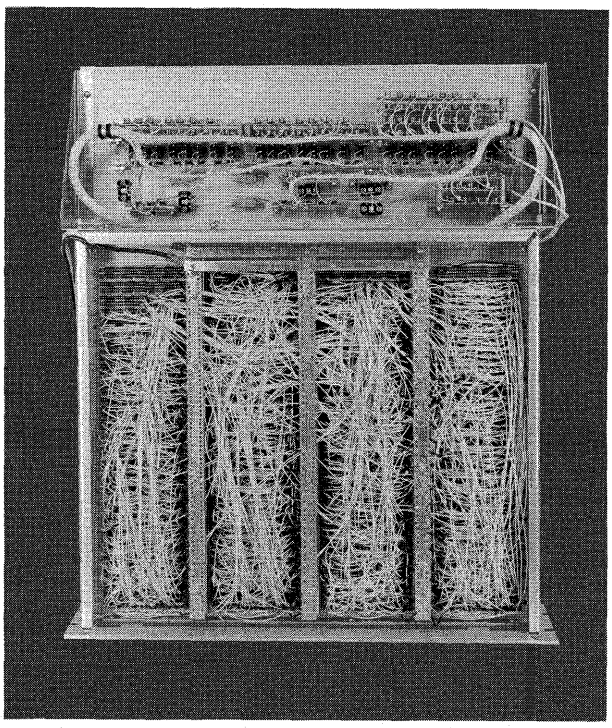


Figure 10. Back view of DDA.

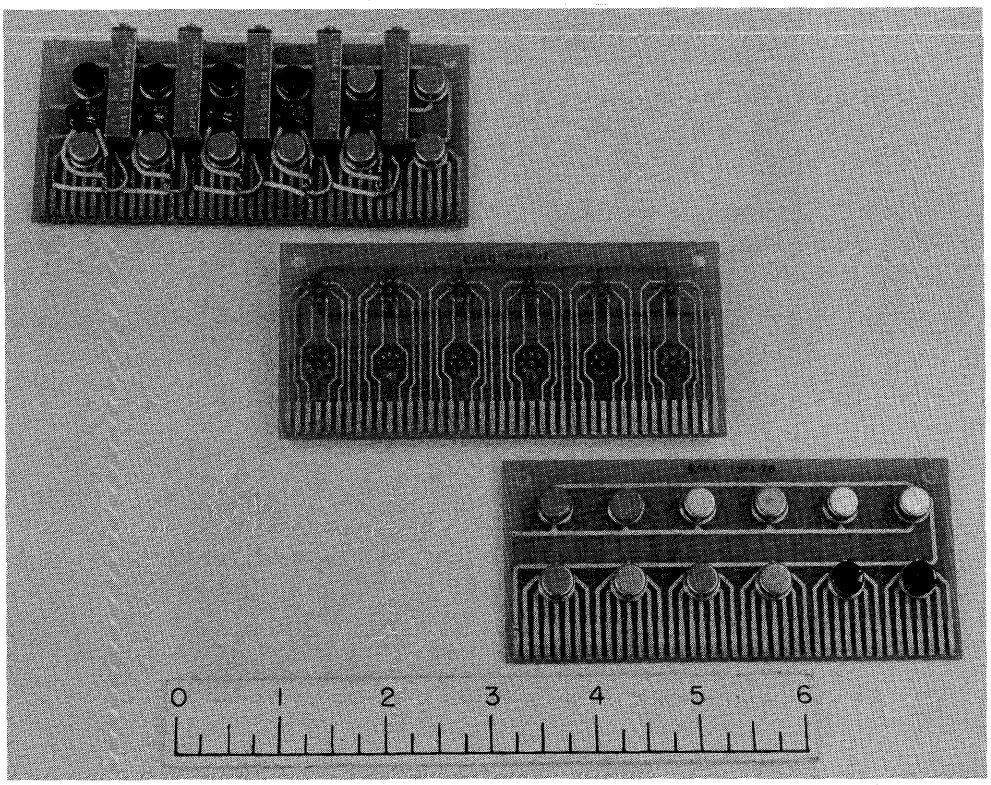



Figure 11

during the computer checkout which accounts for the chassis sitting vertically with the operator's console temporarily mounted on top.

In Fig. 11, the lower card is a normal 12-element logic card. The upper card is also a logic card but with the addition of capacitors and trimpots for single shots. The middle card shows the reverse side of all cards.

The demonstration DDA to solve $xy'' + \frac{1}{2} y' + y = 0$, contains 94 cards, each with 12 elements, for a total of 1200 TO-5 cans (approximately 1900 logic elements). Redesigned for aerospace, and using the conservative estimate of 17,000 flat-packs per cubic foot, the machine could be packaged in a volume of 0.07 $ft^3$ and weigh approximately 3.6 pounds.

## CONCLUDING REMARKS

The use of digital differential analyzers (DDA), because of their great speed in solving differential equations, appears to offer a promising future in aerospace applications over the pure general purpose (GP) approach. In brief, two very attractive applications for the high speed DDA are evident, (*a*) as part of a GP-DDA hybrid which would alleviate the loading of the GP computer for aerospace applications and ( *b* ) as the real time computer for a strap down inertial guidance system. A further detailed comparisn is necessary to justify the merits and economics of this approach, but we do feel that the vastly disproportionate design time given to the GP machines requires at least some attempt at equal design time for the DDA before the final comparison can be made. Potentially, the DDA can iterate a differential equation faster since the GP wastes some time doing housekeeping and memory transfer instructions, e.g., transfers to and from memory, and indexing. In contemporary DDA's this potential has generally not been realized for two reasons; ( *a* ) most of the DDA's built have been serial machines, and ( *b* ) the DDA uses a fixed independent variable increment while most of the more sophisticated GP programs for solving differential equations use a variable increment.

## ACKNOWLEDGMENT

The author wishes to express his apprieciation to Mr. M. F. Hutton of the Martin Company who contributed significantly to the details of this program.

## REFERENCES

1. D. R. Chapman, "An Approximate Analytical Method for Studying Entry Into Planetary Atmospheres," NACA TN 4276, 1958.

2. R. C. Wingrove and R. E. Coate, "Piloted Simulator Tests of a Guidance System Which Can Continuously Predict Landing Point of a Low L/D Vehicle During Atmosphere Re-entry," NASA TN D-787.

3. A. Gill, "Systematic Scaling for Digital Differential Analyzers," IRE PGEC, pp. 486-489, (Dec. 1959).

4. I. Flores, *The Logic of Computer Arithmetic,* Prentice-Hall, Inc., 1963.

5. R. S. Ledley, *Digital Computer and Control Engineering,* McGraw-Hill Book Co., p. 257.

6. R. K. Richards, *Arithmetic Operations in Digital Computers,* D. Van Nostrand Co., p. 303.

7. Huskey and Korn, *Computer Handbook,* Sect. 19 on DDA's, Mc-Graw-Hill Book Co. (1962).

## BIBLIOGRAPHY

1. "An Air-To-Surface Missile Guidance High-Speed Digital Differential Analyzer," Final Report Contract No. AF33(600)-31315, WADC TR-59-651, IBM Fed. Sys. Division (Jan. 1960).

2. C. G. Blanyer and H. Mori, "Analog Digital and Combined Analog-Digital Computers for Real-Time Simulation," *Proc. Eastern Joint Comp. Conf.,* Washington, D. C., Dec. 9 to 13, 1957, pp. 104-110.

3. R. E. Bradley and J. F. Genna, "Design of a One-Megacycle Iteration Rate DDA," Prog. SJCC AFIPS, 1962.

4. E. L. Braun, "Brief Introduction to the DDA Computer," Computers in Control, AIEE Control Comp. Session, 1960 to 1961, pp. 80 to 86.

5. E. L. Braun, "A Comparison of Integral and Incremental Digital Computers for Process Control Applications," *Control Engineering,* pp. 113-118 (Jan. 1960).

6. E. L. Braun, "Design Features of Current DDA's," *IRE Convention Record, Part 4,* N. Y., pp. 87-97, 1954.

7. E. L. Braun and G. Post, "Systems Considerations for Computers in Process Control," *IRE National Conv. Record, Part 4,* pp. 168 to 181, 1958.

8. V. Bush, "The Differential Analyzer," *Journal of Franklin Institute,* Vol. 212 (1931).

9. "Computers in Control," *AIEE* Pub. S-132 (Sept. 1961).

10. J. M. Crank, *The Differential Analyzer,* Green and Company, Ltd., London, 1947.

11. F. G. Curl, "A Comparison of Computers," Computers in Control, 1960 to 1961, AIEE Control Computer Sessions, pp 87-96.

12. "DDA," published by G. Forbes.

13. M. M. Dickinson, "A Comparison of DDA & GP Equipment in Guidance Systems," Computers in Control, 1960 to 1961, AIEE Control Computer Sessions, pp. 208-210.

14. "The Digitac Airborne Control System Trends in Computers: Automatic Control and Data Processing S-59," *Proceeding Western Joint Comp. Conf.,* Los Angeles, Feb. 11 and 12, 1954, pp. 38-44.

15. J. F. Donan, "The Serial Memory DDA," *Math. Tables and Other Aids to Comp.,* Vol. 6, No. 38, April 1952, pp. 102-112.

16. C. F. Edge, "Digital Differential Analyzers Versus General Purpose Digital Computers for Schuler-Tuned Inertial Navigation Systems," *IEEE Trans. on Military Elec.,* Vol. MIL-7, pp. 23-29, Jan. 1963.

17. E. E. Grabbe, *Handbook of Automation, Computation and Control,* Vol. 2, Computers and Data Processing, Wiley, 1959.

18. D. R. Hartree, *Calculating Instruments and Machines,* University of Illinois Press, Urbana, Ill., 1949

19. F. B. Hills, "A Study of Incremental Computation By Difference Equations," MIT Servomechanisms Lab., Rept. No. 7849-R-1 (May 1958).

20. E. G. Homer and W. Palmer, "Comparison of Computational Speeds of Digital Differential Analyzers and General Purpose Computers," *IEE PGEC,* June 1964, p. 307 (correspondence).

21. H. K. Knudsen, "The Scaling of Digital Differential Analyzers," *IEEE Trans. on Electronic Computers,* Vol. EC-14, pp. 583-589, Aug. 1965.

22. R. D. Lamson, "A Division Algorithm for a

Digital Differential Analyzer," *IEEE PGEC,* Feb. 1964, pp. 54-55.

23. F. Lesh, "Methods of Simulating a Differential Analyzer on a Digital Computer," *J. Assoc. of Comp. Machinery,* Vol. 5, July 1958, pp. 281-288.

24. "Maddida-DDA," Brochure No. 38, Northrop Aircraft, Inc. (Dec. 1950).

25. H. E. Maurer, "An Approximate Analysis of Error Propagation in a DDA," MIT Inst. Lab., Martin Company, MR-6140-A44 (March 1958).

26. H. E. Maurer, "Error Analysis of a DDA," MIT Inst. Lab., Martin Company, MR-6140-A-17 (May 1957).

27. M. J. Mendelson, "The Decimal Digital Differential Analyzer," *Aeronautical Engineering Review,* Vol. 13, pp. 42-54, (Feb. 1954).

28. K. Millington, "An Experimental Incremental Computer," *J. Brit. IRE,* Vol. 25, pp. 461-473 (May 1963).

29. L. M. Milne-Thompson, *Calculus of Finite Differences,* Oxford Press.

30. L. P. Mussner, "Real-Time DDA (DART) Trends in Computers: Automatic Control and Data Processing S-59," *Proc. Western Joint Comp. Conf.,* Los Angeles, Feb. 11 and 12, 1954, pp. 134-139.

31. D. J. Nelson, "DDA Error Analysis Using Sampled Data Techniques," *AFIPS SJCC,* 1962, pp. 365-376.

32. P. L. Owen, M. F. Partridge and T. R. H. Sizer, "Cosair, A Ditital Differential Analyzer," Royal Aircraft Establishment, England, TN IAP 1123; "The Use of Direct Coupled Logic in the Design of an Arithmetic Unit (for a DDA)," *Electronic Engrg.,* Vol. 34, pp. 540-545, Aug.; pp. 619-623, Sept. 1962; "A Transistor Digital Differential Analyzer," *J. Brit. IRE,* Vol. 22, pp. 83-96. (Aug. 1961).

33. M. Palensky, "The Design of the Bendix DDA," *Proc. IRE* 41, Oct. 1953, pp. 1352-1356.

34. M. Pavevsky and J. V. Howell, "Digital Differential Equation Solver," *Instr. and Control Sys.,* Vo. 36, pp. 118-121 (April 1963).

35. Z. Pawlak, "The Application of a Negative Base Number System to a Digital Differential Analyzer," *Bull. L'Acad. Polonaise Sci.,* Ser. Sci. Tech., Vol. 8, pp. 149-150, (Feb. 1960).

36. R. Rutishauser, "Litton-20 DDA," Litton Industries, 1955.

37. M. I. Schneider, "Logical Design of Integrators for Digital Differential Analyzers," MIT Instrumentation Lab., Rept. No. T-154 (May 1958).

38. R. G. Selfridge, "Coding a GP Digital Computer to Operate As a Differential Analyzer," *Proc. Western Joint Comp. Conf.,* Los Angeles, March 1 to 3, 1955, pp. 82-84.

39. G. T. Sendzuk, "Results of Simulation Comparison of Control Computers," Computers in Control, *AIEE Control Computer Sessions,* 1960 to 1961, pp. 97-103.

40. G. T. Sendzuk, "A Variable Increment Computer," *Computers in Control, AIEE Control Computer Sessions, 1960 to 1961,* pp. 112-120.

41. M. F. Sentovich, "Mechanization of SST Inertial Navigation Computations in All-DDA Computer," *20th Annual National Meeting, Navigation,* Autumn 1964, Vol. II, NVM 3, pp. 284-298.

42. S. M. Shackell and J. G. Tryon, "The Relative Merits of Incremental and Conventional Digital Computers in Air-Borne Real-Time Control," *Computers in Control, AIEE Control Computer Sessions, 1960 to 1961,* pp. 200-207.

43. C. E. Shannon, "Mathematical Theory of Differential Aalyzers," *J. of Mathematical Physics,* Vol. 4 (Dec. 1941).

44. D. E. Skabelund, "The Numerical Process of a Binary Differential Analyzer," University of Utah Report (Aug. 1953).

45. R. E. Sprague, "Fundamental Concepts of the DDA Method of Computation," *Math Tables and Other Aids to Comp. (6),* Jan. 1952, pp. 41-49. 49.

46. R. E. Sprague, "CRC-105 Computer," *Aero Digest (67)* pp. 48-55, (Aug. 1953).

47. R. H. Stotz, "Specialized Computer Equipment for Generation and Display of Three Dimensional Curvilinear Figures," Contract AD33(600)42859, Prof. DSR 8753, 154 pp., March 1963; U. S. Gov. Res. Rept., Vol. 38, p. 6 (A), September 20, 1963. AD 406 608 (OTS $12.00).

48. J. Tou, "Digital and Sampled-Data Control Systems," McGraw-Hill Book Co., 1959.

49. "The Trice—A High Speed Incremental Computer," *IRE Nat. Conv. Record, 1958, Part 4.*

50. O. C. Turtle, "Incremental Computer Error Analysis," *IEEE Trans. on Communication and Electronics,"* Vol. 82, pp. 492-495, Sept. 1963.

51. R. W. Waller and F. E. Brinckehoff, "A Comparison of Whole Value and Incremental Digital Techniques by the Use of Patch Panel Logic,"

*Computers in Control, AIEE Control Computer Sessions,* 1960 to 161, pp. 104-111.

52. C. J. Wayman, "The Airborne Digital Computer 'Dexan'," *Interavaia,* Vol. 16. pp. 1705-1706, (Dec. 1961).

54. E. Weiss, "Applications of CRC-105 Decimal DDA," *IRE PGEC* EC-1, Dec. 1952, pp. 19-24.

55. H. A. Whitted, "A High-Speed Rate Multiplier for Data Display Systems, Navy Electronics Laboratory, Report 1174, p. 70 (July 1963).

56. D. J. Winslow, "Incremental Computers in Simulation," Meeting of Southeast Simulation Council, Huntsville, Ala., Oct. 1958.

# ENGINEERING MATHEMATICS VIA COMPUTERS

John Staudhammer
*Arizona State University*
*Tempe, Arizona*

## THE NEED

In the last two decades the use of mathematics in engineering has increased considerably. Today, mathematical models make up the heart of many engineering disciplines. The rapid increase in scientific knowledge has brought about a staggering proliferation of technological applications. In order to equip students for this exploding technical environment, engineering colleges have turned to emphasizing fundamentals. It is physically impossible to include even a representative sampling-in-depth of today's technological developments without further crowding the already overburdened curricula. This crowding is due not so much to the new ideas the newer scientific applications require, but rather to the inordinate number of details and refinements they engender.

It is hardly a radical idea to suggest that today's curricula, developed in a more static scientific world, are not keeping pace with the accelerating pulse of the mainstream of scientific thought. But more importantly, the vast majority of curricula fail to prepare students in the philosophy and processes of engineering; rather, they stress a wide variety of special tricks and special procedures. Often the only explanation is that these tricks "solve problems." This is especially true of the construction of simple mathematical models, where the extent of abstraction may be dictated solely by the availability of simple solutions, as for instance, solutions of constant coefficient differential equations.

To ease some of the more routine calculations, computers are being used in some courses in various colleges. However, extensive use of modern computers by all instructors of engineering and engineering mathematics can eliminate many time-consuming rote calculations from the class presentations. Such an extensive use of computers requires a careful reevaluation of the structure of engineering education. Many new and unique computer programs need to be developed for each segment of the restructured engineering curriculum. The faculty and students need to understand how to use computers and what to use them for, and they need to be able to access the computer programs during normal classroom study. Computer use during classroom discussion can allow for the exploration of a larger number of problems of a greater degree of sophistication than is possible otherwise.[1]

A broader base of applied problems will afford an opportunity for students to gain a deeper understanding of the scope of modern engineering. Many more significant problems become vivid demonstrations of principles discussed; ramifications of changes in problem formulation become easily demonstrable,

and a wider variety of solutions can be supplied to the student upon which heuristic arguments may be based. It is even possible to explore intuitive approaches to the solution of given problems.

## THE APPROACH

One of the most subtle changes of the last decade and a half has been in the meaning of the word "solution." It is widely recognized that a mathematical closed form, while unquestionably pleasing aesthetically, may not necessarily be interpreted easily. Curves and graphs may constitute a more desirable solution format, but most often a solution merely consists of a procedure whereby an answer can be obtained. Categorically, today's good solution is one that can be calculated without undue difficulties. We do not shrink from solving 50 or 100 coupled simultaneous linear equations, or that many differential equations. Every major computer center has, or should have, an extensive subroutine library, often stored on an on-line library tape, for effecting extensive numerical calculation and/or simulation.

For statistical work, UCLA's BMD programs have become widely accepted industry standards; programs like RAND's ROCKET are used extensively in work in astrodynamics. Yet very few engineering instructors have been exposed to these powerful tools; fewer yet have introduced their students to the concepts, the frame of mind necessary for the effective use of these procedures. Many engineering educators are still preoccupied with methods for the solution of specific problems; they neither emphasize nor demand a mathematical environment that stresses concepts of problem formulation and problem evaluation. In such an environment, effort would be concentrated on *stating the problem correctly and adequately and estimating the expected answer.* The mechanics of obtaining an answer would be left to a computer (or a computer center), the computer then in effect would become an educated answer-generating device. Such a device may still be fallible: when an unexpected answer appears, the user of the device must know enough about the problem to be able to account for the discrepancy. Often it lies in the method used in arriving at the estimate, but sometimes it may be due to bad input data, or (rarely) to a true machine error.

## AN EXAMPLE: MATRIX APPLICATIONS

The main purpose of this paper is to discuss the articulation necessary for a set of computer programs, none of which is very interesting by itself, to be used for the teaching of a senior-graduate engineering course in the application of matrices. This course is conceived as a model for other engineering study areas, which must begin to employ more computer assistance.*

*For an outline of this course, see Fig. 1.

Matrix Methods in Electrical Engineering

1. Introduction: Examples of Matrix Formulations
2. Determinants and the Matrix Inverse
3. Linear Equations
4. Direct Numerical Procedures
   - 4.1 Computational Error Considerations
   - 4.2 Evaluation of Determinants
   - 4.3 Simultaneous Equations
   - 4.4 Inversion
   - 4.5 Programs for Very Large Matrices (Partitioning)
   - 4.6 Library Routines (Dimension - independent)
   - 4.7 Partial Double Precision
5. Iterative Numerical Procedures
   - 5.1 Simultaneous Equations
   - 5.2 Inversion
6. Non-Numeric Matrices (FORMAC)
7. Characteristic Value Problems
   - 7.1 Eigenvalue Calculation
   - 7.2 Eigenvector Calculation
   - 7.3 Repeated Eigenvalues
8. Diagonalization
   - 8.1 Non-repeated Eigenvalues
   - 8.2 Jordan Normal Forms
9. Functions of Matrices
10. Linear Differential Equations
11. Network Topology
12. Four Terminal Networks
13. Small Vibrations Problems
14. Continuous Systems
15. Operations Research Examples

Figure 1. Outline of course on matrix applications.

*Overall Criteria*

A set of computer programs was written conforming to the following criteria:

1. There is a variety of programs for each task to be performed.
2. The programs constitute a sequence of interrelated programs.
3. Successive programs are increasingly more economical, and/or powerful, and/or have a wider applicability.
4. Final programs are favorably comparable with good computer programs.
5. Selective querying is possible.
6. Uniformity of input formats.
7. Relatively uniform output formats.
8. Limited restart feature.
9. Machine independence.
10. Compatibility with current program systems.

In the design of the computer programs each of the above criteria must be considered concurrently. Thus the programming language used is FORTRAN II, with a FORTRAN IV version also developed. It should be pointed out that the matrix routines generated constitute a limited version of matrix interpretive systems, such as the ones in use at the Aerospace Corporation or in Lockheed's FAMAS system.

*Programs for Basic Operations*

After a relatively conventional introduction of matrix notation and of linearity, determinant theory is discussed. This is the first instance of computer work in the course.

The applications aspects of determinant theory are covered in approximately five hours, including the method of Gauss for the evaluation of numerical determinants. This method is then programmed, a main routine written for it, and an acceptable input data format is agreed upon. This format must conform to standard fixed format input of eight ten-digit numbers per card (8E10.0) preceded by two cards, the first specifying a title, the second the order of the determinant that is to be evaluated. Serial numbers and comments can also be placed on these cards. The input data is in row order.

By means of examples the program is shown to be faulty in some cases. The addition of a set of PRINT statements allows monitoring of this program, thus resulting in the second determinant evaluator. It is shown by the examples that it is the main-diagonal zero divisors that lead to the breakdown of the original routine.

At this point a cursory discussion of chopping errors is undertaken and the need for the pivot procedure demonstrated.[2] Implementing this pivot exchange leads to a fully acceptable routine, efficient and relatively accurate as demonstrated by a variety of numerical examples. The internal operation of this routine is checked by means of selective printing of intermediate results accomplished by setting control switches on the computer console.

For inclusion in the permanent library of the installation the last program is made dimension-independent for compilations in FORTRAN II; a version for complex matrices is also prepared. Extensive testing of these programs completes the discussion of numerical determinants.

Next the discussion of simultaneous equations is undertaken. The basic theory underlying linear vector spaces is covered in about three hours, culminating in the derivation of the Gauss method of elimination. Programming this method along the lines of the first determinant evaluator gives the first equation-solving routine. Although this program will also fail when zero main-diagonal elements are generated, the introduction of pivoting is delayed until after a discussion of the Gauss-Jordan method.

Pivoting used with the Gauss-Jordan method is then made dimension-independent for inclusion in the system library tape. Extensive numerical examples are used throughout.

The next topic handled is matrix inversion. Five different programs are discussed, paralleling the programs for the solution of simultaneous equations, and culminating in an efficient in-place inversion routine (using pivoting and dimension-independence) for use in FORTAN II and IV systems.

The derivation of this row-and-column-shuffling procedure is rather interesting. Instead of deriving the various algorithms needed for compacting the Gauss-Jordan inversion procedure,[3] the original matrix is reduced and its augmented unit matrix printed for several test cases until it becomes obvious how to eliminate the augmented part (thereby saving 50 per cent storage space.) Similarly the algorithms necessary for unscrambling the pivot-condensed, row-and-column-shuffled matrix are first obtained

heuristically from a series of examples; only then are they derived mathematically. This procedure gives life to otherwise dry and seemingly uninteresting derivations.

The discussion of the solution of linear operations is concluded with programs suitable for complex matrices and procedures used with partitioned matrices. A few tape-shuffling routines are demonstrated for solving a set of simultaneous equations too large to fit into the machine at one time.

The discussion of round-off errors and direct numerical procedures is "naturally" extended to iterative techniques such as the Gauss-Seidel procedure.

Throughout the discussions, a set of examples is used that demonstrates the limitations as well as the uses of the various methods. Without computer assistance most students would not be able to gain any insight into the why's and wherefore's of these various procedures.

*Eigenvalue Problems*

The next major topic discussed is the general eigenvalue problem. After conventional derivation of the Cayley-Hamilton theorem, the theorem is used to derive a computer-adaptable procedure for finding the characteristic equation. This process uses subprograms developed in earlier discussions; hence a pyramiding of subroutines starts at this point.

The solution of the characteristic equation is accomplished by a standard library routine, which is simply taken as given.

Properties of eigenvalues of special matrices are discussed with the aid of the above computer programs. The program is extended to complex matrices and properties of these matrices are studied. Applications to network analysis and vibration problems seem natural at this point.[4]

Application of the simultaneous equation routines discussed earlier leads to eigenvector problems; difficulties with repeated eigenvalues are easily demonstrated, and the idea of a minimum polynomial becomes obvious. Orthogonality of eigenvectors is demonstrated and their use in vibration problems illustrated by means of carefully contrived examples.

The complete eigenvalue problem constitutes the last problem discussed under this topic. Construction of the Jordan normal form leads directly to procedures (all programmed) for obtaining generalized eigenvectors.[5] Numerical examples showing these methods complete this part of the class presentation.

*Matrix Functions*

During the class discussion, the Cayley-Hamilton theorem is applied (repeatedly) to the construction

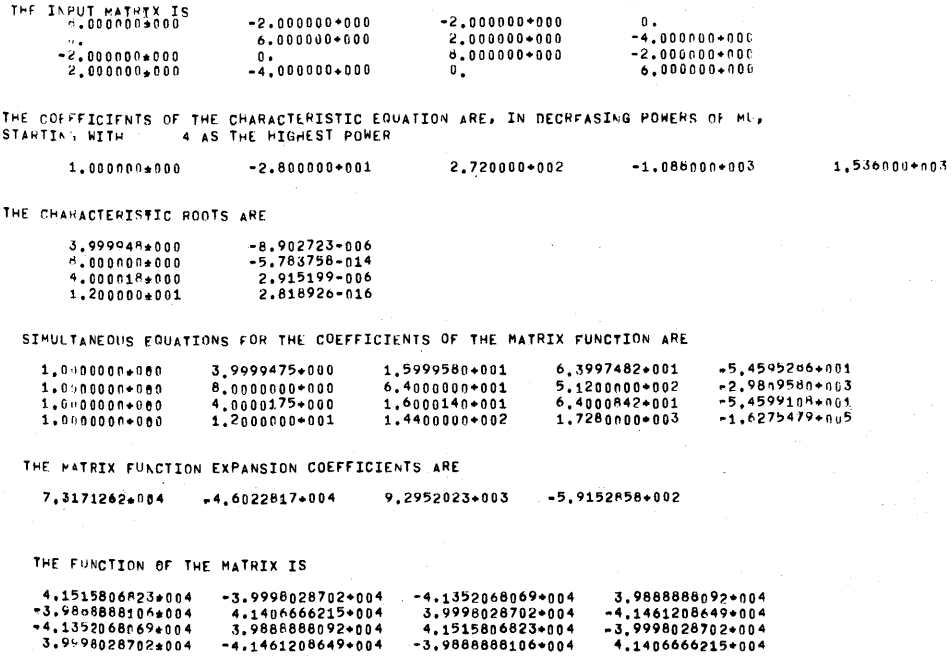



Figure 2. Calculation of $e^M$.

of matrix functions. Trigonometric and exponential functions having square matrices for arguments occur naturally enough in the solution of certain sets of linear differential equations. Several subprograms are constructed to carry out several methods for calculation of these matrices.

Various shortcuts in the above procedures lead to the surprising conclusion that the most time-consuming part of matrix function calculations is the raising of the original matrix to successively higher powers up to N-1, where N is the order of the matrix. A search for shorter procedures leads to the use of Jordan normal forms. Even though the logic is considerably more complex, the resultant program is approximately N times faster.

Once the programs are discussed they are checked out on short problems, suitably chosen to give easily checked results. (See Fig. 2.)

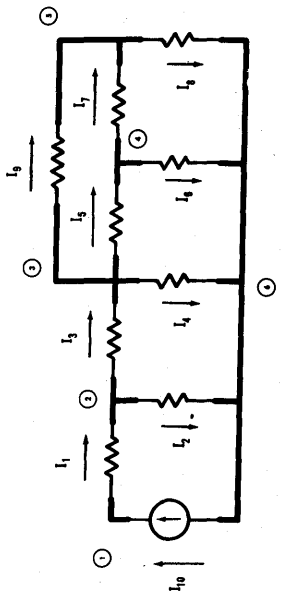
The ideas presented so far are then extended to the solution of simultaneous differential equations. Physical problems are formulated; the complexity of the mathematical model may be essentially unrestricted by considerations of computational involvement.

*General Linear Equations Package*

The program developed in the above section is one of two more extensive omnibus programs developed in the course. The other is a long routine for the solution of M linear equations in N unknowns. A check is made of the consistency of the set; consistent sets are further examined for unique solutions and parametric solutions. The program uses the Gauss method without pivoting to obtain a first approximation to the solution; by appropriate switch action the above process may be repeated using pivotal condensation. Next, unique solutions may be iterated using either the Gauss or the Gauss-Seidel iteration technique. After each complete pass the program may be restarted for a possibly different intermediate printing. Input data and intermediate results are stored on tape, thus enabling the handling of 80 by 80 coefficient matrices on a 16,000-word machine.

*Applications of Matrix Programs*

The reason for developing these computer programs is their wide applicability to various engi-



The interconnection matrix is:

$$\begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \\ I_5 \\ I_6 \\ I_7 \\ I_8 \\ I_9 \\ I_{10} \end{bmatrix} = 10^1_0$$

Order of elimination of Variables is

$$I_{10}, I_9, I_8, I_7, I_6, I_5, I_4, I_3, I_2, I_1 .$$

Figure 3. Example for finding independent currents.

neering fields. Since this course is taught within the electrical engineering curriculum, the lion's share of the examples are drawn from network topology, network analysis, and control systems. The programs typically facilitate analysis of various engineering problems rather than provide synthesis procedures.

Electrical network equations are now input in completely general form into the linear equations package; the computer results indicate the independent node voltages and the independent currents. Supplying separate element values to the programs gives the canonical equations for the network, which are then solved as a matrix differential equation. Throughout, many intermediate results may be printed, for class demonstration as well as possible checking, by appropriate console switch actions. (See Figs. 3 and 4.)

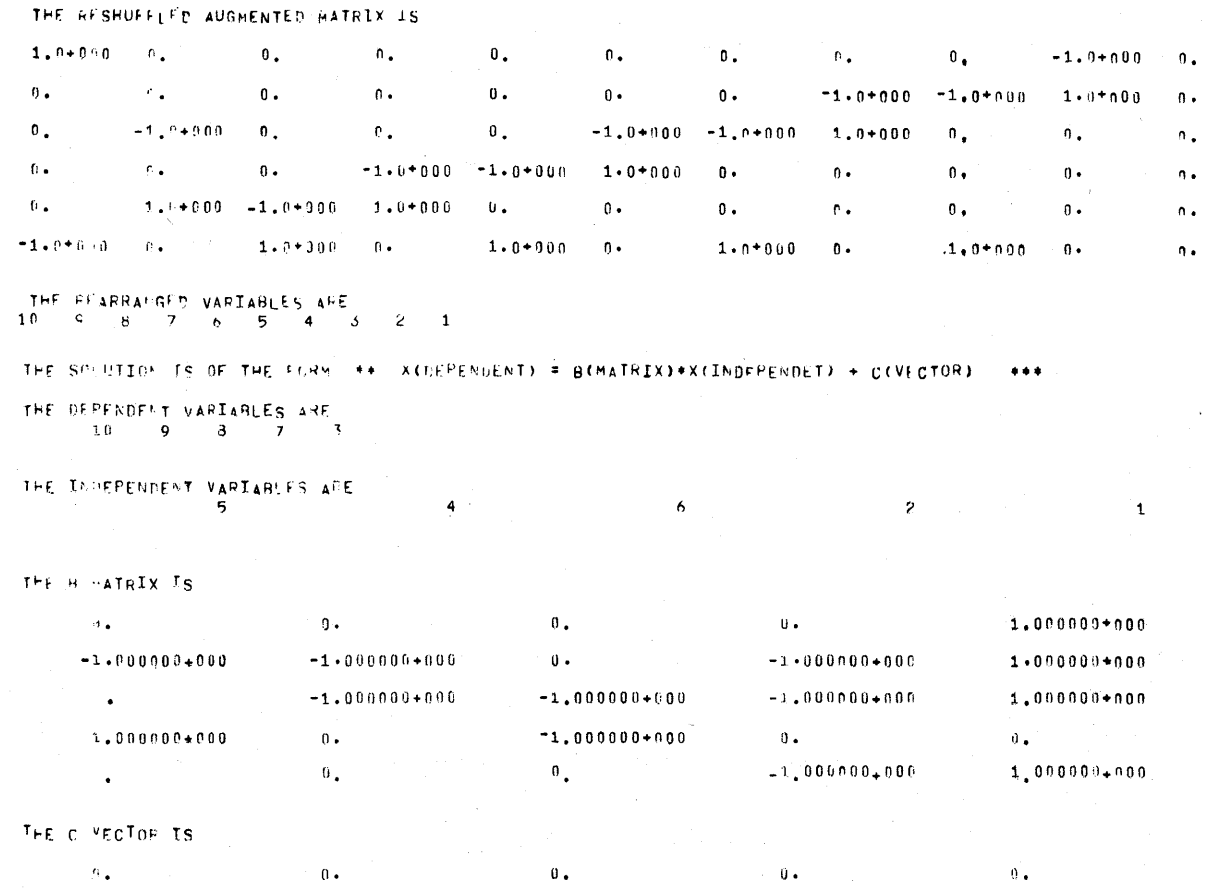
Four-terminal networks are treated in an analogous way; routines for the handling of various interconnections are prepared, and the mathematical network model is constructed on the computer, which then generates numerical answers to transfer function analysis. Several shortcut procedures are also illustrated.[6]

The matrix function routines are also used extensively in the state vector calculations of control system analysis. Particularly eigenvalues and Jordan normal forms are found very applicable in this work. Matrices of an order up to 80 by 80 are routinely handled in the course.

Not all applications are drawn from electrical engineering; rather, a good number of examples are taken from the subjects to be described next.

## OTHER EXAMPLES

The field of optimization techniques benefits greatly from expanded computer applications. Among the classical techniques of this field are linear progrmming and game theory. Most digital



```
THE RESHUFFLED AUGMENTED MATRIX IS

1.0+000   0.        0.        0.        0.        0.        0.        0.        0.        -1.0+000   0.
0.        0.        0.        0.        0.        0.        0.        -1.0+000  -1.0+000  1.0+000    0.
0.        -1.0+000  0.        0.        0.        -1.0+000  -1.0+000  1.0+000   0.        0.         0.
0.        0.        0.        -1.0+000  -1.0+000  1.0+000   0.        0.        0.        0.         0.
0.        1.0+000   -1.0+000  1.0+000   0.        0.        0.        0.        0.        0.         0.
-1.0+000  0.        1.0+000   0.        1.0+000   0.        1.0+000   0.        1.0+000   0.         0.

THE REARRANGED VARIABLES ARE
10   9   8   7   6   5   4   3   2   1

THE SOLUTION IS OF THE FORM  **  X(DEPENDENT) = B(MATRIX)*X(INDEPENDET) + C(VECTOR)  ***

THE DEPENDENT VARIABLES ARE
      10    9    8    7    3

THE INDEPENDENT VARIABLES ARE
                5                   4                   6                   2                   1


THE B MATRIX IS

      0.                  0.                  0.                  0.                  1.000000+000
     -1.000000+000       -1.000000+000        0.                 -1.000000+000        1.000000+000
      .                  -1.000000+000       -1.000000+000       -1.000000+000        1.000000+000
      1.000000+000        0.                 -1.000000+000        0.                  0.
      .                   0.                  0.                 -1.000000+000        1.000000+000

THE C VECTOR IS

      0.                  0.                  0.                  0.                  0.
```

Figure 4. Partial results for problem in Fig. 3.

computers have a standard linear programming package that requires little user concern for possible computational difficulties. The time gained may be spent more profitably on examining the philosophy of the procedures used and the significance of the answers provided. Solutions of game theory problems are also relieved of computational considerations. The way is cleared for setting up complicated pay-off matrices and obtaining answers with relative ease.[7]

The most recent general optimization technique is dynamic programming. The use of dynamic programming in control theory has penetrated to advanced undergraduate training. In its various specific applications the theory is a numeric process requiring the storage of a vast number of intermediate results ("decision tables"). Because of this voracious appetite for storage space, most of the present applications involve systems with but a few variables: however, with the widespread use of the new large-memory machines, we can expect more significant applications of this theory.

A few of the widespread uses of computers in the solution of essentially computational problems in engineering education will now be mentioned.

In the more conventional electrical engineering curricula there are several areas in which computers may be directly applied, notably network analysis and synthesis. In straightforward analysis it is possible to devise computer programs that take into account the network topology (with specific element values supplied separately) and compute the steady state as well as the transient response. Such a routine can be used early in the network theory courses to do cut-and-try designing to some specifications. While the results are not as graphic as a laboratory experiment, complicated networks may be analyzed rapidly, the effects of small parameter variations can be studied, and various parametric studies can be done in advanced training. In the synthesis of networks, the computer has proven itself invaluable in industry and should prove its worth in teaching as well; by preprogramming various synthesis procedures, the results of those procedures can be displayed rapidly after a discussion of the method rather than the mechanics. The additional bonus that these results are essentially error-free provides increased class time, which can be most effectively used for an examination of the underlying design philosophy, thereby strengthening the understanding of fundamentals of network theory.

In graduate courses where network analysis and synthesis are used as part of a larger subject area (such as control system courses), a computer can be of great help in automatically realizing compensator transfer functions. Such a program first has to determine physical realizability and the kind of network realizable; it then must carry out one or more alternate realizations, possibly deciding on an "optimum" network. One can build such a network synthesizer by programming each of the constituent subprograms separately in a course in network synthesis; also, many of these subprograms already exist in various computer facilities.

In aero- and hydrodynamics, a computer can mechanize the formulas of the Schwarz-Christoffel transformation, whose solutions in most nonartificial problems are numerically too complicated for hand computation. This particular mechanization allows the calculation of laminar incompressible flow streamlines, a subject occupying most of the first semester in an advanced hydrodynamics course. However, having obtained a powerful computational tool for such calculations, much time can be saved; this time can then be profitably spent on real fluids and methods of calculating their streaming behavior. The microscopic differential equations of fluid motion can be directly solved numerically (with some difficulty, to be sure) for given boundary conditions. This in turn leads to a deeper understanding of every phase of aero- and hydrodynamics, an understanding that presently is lost under a veritable avalanche of special methods for special problems and insurmountable numerical work.

Advanced dynamics also benefits from the computer revolution. Complex structural dynamics no longer needs to restrict itself to three-story coupled bents; one can discuss general methods powerful enough to describe multiple-loop coupled structures undergoing combined vibrations. Such procedures as the transfer matrix method, using 12 and 18-element state vectors can be discussed without the fear that they will not be usable in a given situation. As a matter of fact, the subject of elastomechanics has undergone a more marked metamorphosis than any comparable subject, as a direct result of digital computation. This is one of the areas in which matrix methods find their widest application.

Many of the methods of nonlinear analysis are essentially numerical procedures. A study of nonlinear phenomena is usually limited by the vast number of iterative calculations normally required even in a simple problem. Here also the removal of much numerical work, by the use of appropriately programmed computers, will allow concentration on method rather than procedure. In nonlinear control synthesis, the use of iterated matrices in significant problems becomes a perfectly acceptable procedure in a computer-assisted graduate course.

Finally, a short reflection on the newly emerged field of systems engineering in the computer revolution. Because of the extremely complex nature of the various problems that make up the basis of this discipline, no realistic systems engineering exercises can be demonstrated without a computer. It is in systems engineering that traditional industrial and electrical engineering overlap. The methods of the discipline are taken largely from applied mathematics and statistics. Since most students interested in the field have incomplete backgrounds in each of the areas from which the methods stem, thorough systems engineering education is hampered. With a computer, enough time can be saved so that criteria and their effects, and not merely the procedures of the techniques, can be critically examined in a systems engineering course.

## RESULTS

The course discussed in the section on Matrix Application was taught during the Spring 1965 semester at Arizona State University. Earlier, less computerized versions were taught over a period of four years at the University of California at Los Angeles and Arizona State University. The impact of computers on this course lies in the mechanization of routine numerical work enhanced by the introduction of the principles of more advanced numerical methods. Some of these methods were not discussed in class (e.g. the solution of polynomial equations) but their results supply numerical, nontrivial examples illustrating engineering thinking, not engineering drudgery. Concentration on problem *formulation* rather than *solution* unifies many facts of engineeing.

The use of the computer provides both faculty and students with valuable time, which may be used to develop intuition about the problems by exposing the student to a large number of allied numerical results. The time can be spent in exploring ramifications of second order effects, which so often can make the correlation of theoretical prediction and experimental results very inaccurate. A broader base and a larger number of examples can be used to explore a greater variety of applications of mathematical tools than is possible without computer aid. In the course already developed, approximately twice as much material and about five times as many examples are now covered without any significant degradation in the students' comprehension of the material.

More complex and elaborate problems were made feasible for classroom use. As the breadth of the problems handled adequately has been increased, a concomitant deeper understanding of the engineering method has been fostered. Newer and more advanced topics have been included in the undergraduate curriculum.

And finally, a facet of the curriculum has been changed to introduce the young engineer to methods used daily in modern engineering work.

## REFERENCES

1. W. J. Pelton and J. Staudhammer, "Computer Assistance in Teaching Engineering Mathematics," *The Journal of Engineering Education,* Dec. 1965. (See also Ref. 8.)

2. D. D. McCracken and W. S. Dorn, *Numerical Methods and FORTRAN Programming,* J. Wiley & Sons (1964).

3. D. K. Faddeev and V. N. Faddeeva, *Computational Methods in Linear Algebra,* Freeman, San Francisco, Calif. (1963).

4. R. A. Frazer, W. J. Duncan, and R. Collar, *Elementary Matrices,* Princeton University Press (1962).

5. S. C. Gupta, *Transform and State Variable Methods in Linear Systems, John Wiley &* Sons(1965).

6. J. Staudhammer, *Matrices in Electrical Engineering* (to be published).

7. J. Staudhammer and R. Kao, "Techniques of System Analysis and Design: Game Theory," TM-679/002/00, System Development Corporation, Santa Monica, Calif.

# THE COMPUTER — TUTOR AND RESEARCH ASSISTANT

Robert J. Meyer
*University of Rhode Island,*

The concept of *a library of statistical computer programs based on a single statistics text* can significantly reshape current teaching of statistical methods and can increase the use of statistical methods in research—especially if the library is quite easy to use. Such a library has changed the way we teach and use statistics at our university.

The library was developed in its present form because three years experience with existent program libraries convinces us that the old, but ever-present, communications problem was blocking extensive use of the computer in statistics courses and in quantitative empirical research.

We have in large measure removed that block by putting most of the Snedecor text[1] algorithms "in" the computer (i.e., on the 1410 systems tapes); and by making all the computer programs accessible through use of standard prepunched monitor system cards and a single, variable, control card. This control card "tells" the computer the number of columns of data in the two-dimensional data matrix and the number of items in each column.

This control card, along with a completely free form title card and data cards punched one item per card (while reading the data table columnwise), allows the student and researcher to process data with a minimum of instruction.

## DISSATISFACTIONS WITH EXISTENT COMPUTER PROGRAM LIBRARIES

The main difficulties associated with extensive classroom use of programs available through such organizations as SHARE, the IBM KWIC INDEX, the listings in Technometrics, etc., arise from the fact that the programs have been written by so many different people—people with rather diverse backgrounds and training and with, of course, differences in style. The three main problems that evolve are:

1. It is often difficult to know exactly what computations and analyses are being used in any given program without actually sending for the program and its associated documentation.
2. One must learn a rather large variety of, at times, quite complex data input and control card arrangements.
3. The symbolism and terminology used in the computer output is usually somewhat different from that used in the statistics course being given, or in the reference text usually used by the research worker. At times a major problem of the user is the determination of exactly what the program

writer meant by certain terms used in labeling or describing the output.

The principal difficulties then revolve around the large amount of effort needed to get programs to run, to get "into" the computer, and the effort needed to understand fully what has come out.

We believe that the library to be described goes a long way toward solving that problem. It solves the communication problem by building a program library around a standard statistics text—in this case the Snedecor, used in our course in Design and Analysis of Experiments.

## THE SNEDECOR SERIES — A PROGRAM LIBRARY

The prospective user of computer library programs typically asks the following questions:

1. What are the calculations performed by any particular available program?
2. What answers will be provided?
3. In what form will the answers appear; that is, what analysis technique will be used to portray the relationship existant in the data?
4. How do I use the program; that is, how do I prepare control cards and data cards?

All programs in the library are given a name including the appropriate Snedecor page listing; for example, SNEDECOR 122-126, Linear Regression. Therefore questions (1), (2), and (3) can be, and usually are, answered by simply referring the prospective user to the appropriate discussion in the Snedecor text.

When a student or researcher asks whether there is a program in the library that will do some particular statistical computation we ask whether such computation is described in the Snedecor text and where. Should he say, on page 160, we simply check the library listing. If there is a SNEDECOR 160 program listed, we can answer his query with a simple affirmative. When he asks for a description of the computation and output we simply state that all the answers shown on page 160 will be provided and each answer will be labeled with the actual wording used by Snedecor. Additionally, it is pointed out that the use and significance of each answer provided is discussed in Snedecor. Further discussion can be provided in the classroom and in the office.

Through these procedures the amount of time taken up by program writer-user, or library administrater-user, discussion and questioning is held to a minimum, encouraging, thereby, mutual confidence in the system.

The fourth question takes a little longer to answer. Input data decks contain one Title card, one Control card, and Data cards containing one data item per card.

The Title card can contain any identification information desired in any or all of its 80 columns. This information will print on each new page of output.

*Example:*

MEAN SYSTOLIC BLOOD PRESSURE OF 58 WOMEN IN 10-YEAR AGE CLASSES

The Control card contains a packed sequence of three-digit numbers which tell the computer: (1) how many columns of data there are in the data table, and (2) how many items there are in each column of the data table.

That is, the first three-digit number on the Control card specifies the number of columns of data there are in the data set used. Then, if all columns have the same number of items, the second three-digit number on the Control card specifies that equal columns length.

*Example* :

002005

These numbers punched in the first 6 columns of the Control card indicate that there are 2 columns of data with 5 items in each column.

Should each column in the data set have a different number of items, the second three-digit number of the Control card specifies the number of items in the first column of the variable length columnar data table, the third three-digit number on the Control card shows how many items there are in the second column of data, and so on, until all column lengths are specified.

*Example* :

00603012409013407009

These numbers punched in the first 21 columns of the Control card indicate that there are 6 columns of data with 30 items in the first column of data, 124 items in the second data column, 90 items in the third column, 134 in the fourth, 70 in the fifth,

and 9 in the sixth column of data in the two-dimensional data table.

In the Data cards the data items are punched one item per card, with a decimal point included, anywhere in the first 10 columns of each card. The data is punched and input to the computer column-wise. That is, the first column of the data table (as set up in the examples in the Snedecor text) is punched and input from top to bottom first, then the second column of data is punched and input from top to bottom next, and so on, until all data columns are so fed into the computer.

Any constants and parameters required are then added to the data input deck on separate trailing cards similarly punched, i.e., one item per card, with included decimal point, anywhere in the first 10 columns of the card.

This type of data input deck format—Title card, Control card, and one-item-per-card Data cards—is the only one used by beginners and casual users. (An across-the-card-punching-of-data, with row-wise reading, option is offered. There is no need, incidentally, to "tell" the computer that the data is coming in row-wise—the computer senses this by itself.)

This one package provides input to every program in the library. Consequently, once the student or researcher has successfully prepared one input deck he knows how to gain access to all of the other programs in the library.

With the input simplification and standardization problem solved, attention was next given to production of clear, easily understood, and specifically referenced output.

The first item printed out by the computer is the descriptive information the user punched in the Title card. All of it is presented. (See Item A in the example of computer output shown in Fig. 1.)

Next, the title of the library program, including the Snedecor page reference, is printed. (See Item B in Fig. 1.)

Thirdly, a paragraph listing of the items put out is provided. This information is especially useful when the reference text isn't handy. (See Item C in the figure.)

The paragraph listing is followed by a labeled multicolumn table of the actual data input. This table is used by the student or researcher to check his data deck punching. He knows exactly what data

MEAN SYSTOLIC BLOOD PRESSURE OF 58 WOMEN IN 10-YEAR AGE CLASSES (A)

SNEDECOR 122-126 LINEAR REGRESSION ANALYSIS (B)

THIS PROGRAM COMPUTES THE MEAN OF THE X VALUES,
THE MEAN OF THE Y VALUES, THE A TERM, THE B TERM,
THE SAMPLE STANDARD DEVIATION FROM REGRESSION,
THE SAMPLE STD. DEV. OF THE REGRESSION COEFFICIENT, (C)
THE T VALUE, THE ESTIMATED Y-S, AND THE DIFFERENCES
BETWEEN THE ESTIMATED Y-S AND THE OBSERVED Y-S.

THE DATA USED IN THIS PROGRAM IS AS FOLLOWS

| X VALUES | Y VALUES |
|---|---|
| 35.000 | 114.000 |
| 45.000 | 124.000 |
| 55.000 | 143.000 |
| 65.000 | 158.000 |
| 75.000 | 166.000 |

(D)

REGRESSION ANALYSIS

| | |
|---|---|
| THE MEAN OF THE X VALUES IS | 55.000 |
| THE MEAN OF THE Y VALUES IS | 141.000 |
| THE VALUE OF THE A TERM IS | 65.000 |
| THE VALUE OF THE B TERM IS | 1.380 |
| THE SAMPLE STD. DEV. FROM REGRESSION IS | 3.245 |
| THE SAMPLE STD. DEV. OF THE REGR. COEFF. IS | .102 |
| THE T VALUE IS | 13.446 |

(E)

| X | Y | ESTIMATED Y | DEVIATION |
|---|---|---|---|
| 35.000 | 114.000 | 113.400 | .600 |
| 45.000 | 124.000 | 127.200 | -3.200 |
| 55.000 | 143.000 | 141.000 | 2.000 |
| 65.000 | 158.000 | 154.800 | 3.200 |
| 75.000 | 166.000 | 168.600 | -2.600 |

(F)

Figure 1. An example of computer output.

was used in the calculations performed. (See Item D in the figure.)

Then, single number output of calculated statistics, along with sentence-like identification of each type of statistic, is provided. Here care is taken to use only those words used in the reference text, the Snedecor. (See Item E in Fig. 1.)

Lastly, tabular output, if pertinent, is presented. Again, the arrangements in the Snedecor text examples are followed as closely as it is possible to do so on a computer printer. (See Item F in the figure.)

## EXPERIENCE AT THE UNIVERSITY OF RHODE ISLAND

In the two years that we have been using this computer-programs library significant progress has been made towards reshaping our teaching and research consulting activities.

We can now offer a statistics course, taken by upper division and graduate students from many disciplines, in which the student moves much more quickly than he has in the past to quantitative empirical research in some field of his own interest, or

to the analysis of data gathered by others in such research.

And when outside researchers come for analysis suggestions or advice, we can suggest analytic approaches which will be implemented because calculations can so readily be accomplished at the computer lab.

We have reduced to a minimum the classroom time spent on those routine calculations exercises that often deaden the interest of beginning statistics students. (These are learned by doing simple homework problems.)

In the classroom there is more time, now, for extensive discussion of research in general, and of quantitative empirical research in particular. There is much more time to exemplify these by illustrative discussion of the proposals submitted by the students and by analysis of the finished projects submitted.

Still more importantly, we make it possible for the student to look at more problems more ways. It has been our experience that once students begin to use the program library they will run, say, a half dozen regressions at the same time, trying out different sets of independent variables, trying perhaps different types of equations, and even, at times, attacking the same questions and research projects with several different statistical techniques.

There are peripheral, but nonetheless important, benefits:

1. Students get started on their research projects much more quickly. There is no long period of hesitation while awaiting confidence in computational skills.
2. Research results can be analyzed with the confidence that the arithmetic is correct.
3. The student gets away from the lack of realism inherent in small, round numbered, statistically "clean" problems. He gets deeply involved in most of the real-life problems of empirical research.
4. It is possible to introduce statistical techniques not usually presented in introductory courses. It is possible to illustrate and use curvilinear regression, multiple regression, covariance analysis, two and three way analysis of variance and partial correlation. And data can be more often checked to see whether it meets the assumptions underlying the techniques used.

Not the least benefit of the availability of a computer and an associated library of statistical programs is the avoidance of the classic stance of the donkey between the two bales of hay. When the student asks which of two ways he should attack some statistical problem or research project we suggest that he do it both ways and see what happens. All that is required is that the data deck be submitted twice to the computer; or that the data deck be machine-duplicated and submitted to as many programs as the student wishes to try. When the question of whether apparently extraneous or unusual values should or should not be used is raised, we suggest that decks be submitted both ways, with and without the extreme values. The effect of extreme values can thereby be specified, presented openly, and clearly discussed for the consumer of the study.

In summary, then, bringing the computer into the statistics course increases the likelihood of heightening students' interest in the course, and, most importantly increases the probability of shifting the emphasis from historically distasteful "calculation" courses to the more exciting, more useful, and often more glamorous subject of empirical research.

Such objectives are not terribly new. The prospects discussed above have been articulated before. What is new here is the method of implementation —a computer-program library that is quite easy to use—a library that makes it easy to "talk" to the computer.

The library programs were, and are being, written by this author, assisted by one full-time graduate assistant programmer using some specially designed program production techniques.

We have also used "learning" and "full" program combinations in a start at having the computer act as a tutor. The covariance "learning" program, for example, provides only gross computations such as sums of squares and cross products (exemplified at the top of page 395 in Snedecor). The student is then asked to use the output from this program to carry out those computations leading to the final covariance analysis—computations he can carry out most readily if he knows and understands the concepts of covariance analysis. He then obtains a feedback check on his computations by resubmitting his original data deck to the "full" program.

Eventually we hope to provide error analysis in "tutorial" programs which will check hand-computed homework problem answers submitted on

punched cards. When this is done, the computer will act so like a tutor as to obviate the need of apostrophication of the word "learning" when describing the programs.

In any case we have, as one student recently remarked, "opened a whole new world" to the student —computerized empirical quantitative research.

Fully as importantly, though, we now have more adequate instruments for attacking and remedying a fundamental weakness in our students' backgrounds —their inability to frame and attempt to answer questions, especially those questions whose answering can lead to greater objective achievement in many areas.

## REFERENCES

1. *Statistical Methods,* 5th ed., Ames, Ia., Iowa State University Press, 1965.

# TRAINING FOR THE NO. 1 ESS

Matthew Raspanti
*Bell Telephone Laboratories, Inc.*
*Holmdel, New Jersey*

## INTRODUCTION

The first installation of the No. 1 Electronic Switching System[1] (No. 1 ESS), a stored-program electronic telephone switching system developed by Bell Telephone Laboratories, started commercial operation in Succasunna, New Jersey, on May 30, 1965. In the No. 1 ESS, call processing and maintenance functions are performend under the control of a program of some 100,000 words stored in a read-only type of memory. The 44-bit instructions are executed by a central processor operating on a basic cycle time of 5.5 microseconds. A read-write type of memory with 24-bit word locations is used for the storage of transient information such as the digits dialed by a subscriber.

The introduction of the No. 1 ESS has made it necessary to train Bell System personnel in an entirely new technology. To meet this need, the author was charged with the responsibilty of organizing an 8-month school on the No. 1 ESS to train a first group of 64 Bell System employees who would be involved in the installation or maintenance of the earlier central offices or in the training of personnel for subsequent installations. The school prerequisites were a high school education and a knowledge of basic electricity and electronics (typically provided by operating companies in a 4-week course).

In the early stages of planning for the school, considering the complexity of the No. 1 ESS and the electromechanical background of the class, it was recognized that significant difficulties would be experienced by the trainees in understanding the basic concepts of program control and in visualizing the overall operation of the system, particularly its information-processing aspects. Also, since the school and the debugging of the system would be conducted concurrently, the 64 trainees could be expected to have only limited access to the No. 1 ESS for training purposes. In any case, it was felt that, because of its size, speed and complexity, the system itself would not be a suitable aid in the initial stages of training. The design of WOSP a Word-Organized Stored-Program training aid, was then undertaken to achieve the following objectives:

1. To develop a machine whose organization would conceptually parallel that of No. 1 ESS.
2. To provide full display of internal operation so that the information being processed, the operations being performed and

the flow of information would be directly apparent to the trainees.

WOSP is a miniature information-processing machine capable of completely automatic operation under the control of a program stored in punched cards such as the one shown in Fig. 1. Up to 4 cards can be used to provide 44 word locations of

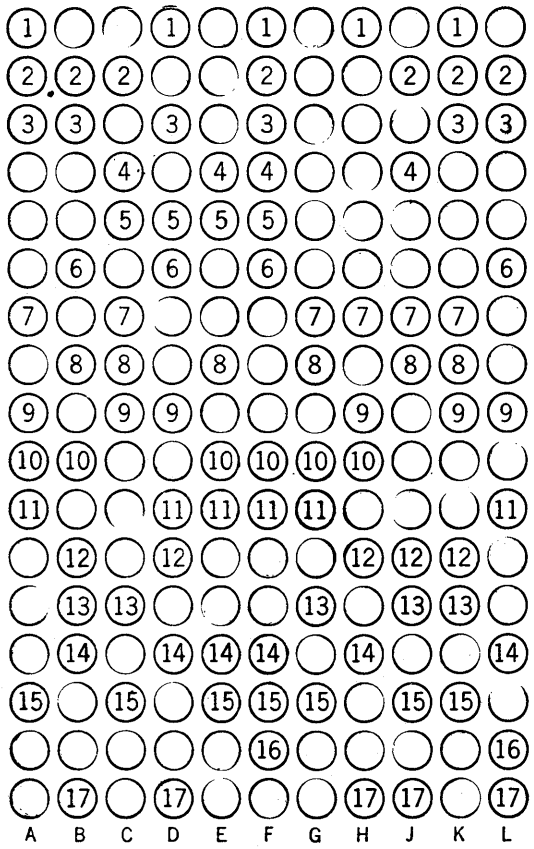



Figure 1. Program card.

17 bits each. Physically, WOSP consist of two units connected by a cable: a T-cart (Fig. 2) and a display panel (Fig. 3). The T-cart contains all the memory and logic circuitry which consists mainly of 4 card readers, about 170 wire spring relays, 48 reed relays and about 1,000 diodes. The four card readers and all the manual controls are mounted on the top of the T-cart. Relays and other electromechanical devices have been used because of their low cost and their readily understandable nature.

Five WOSP machines have been built so far, one by Bell Telephone Laboratories and four by Western Electric Company. These machines have been used very successfully to introduce program control to several hundred trainees. Because of the opportunity it offers for truly "hands on" training, WOSP could be a valuable aid for programming training of a more general-purpose nature.

## ORGANIZATION OF THE NO. 1 ESS SCHOOL

The No. 1 ESS school was set up as a special school to meet Bell System needs associated with the introduction of the No. 1 ESS. It was not intended to be a prototype for later schools to be conducted by the various telephone companies.

The 35-week trainnig, which started in January 1964, was conducted by a staff of 14 men, most of whom were also responsible for the preparation of maintenace manuals. Of the 14 men, 7 were Laboratories employees, 7 were on loan from Western Electric Company and 4 telephone companies. The 64 trainees came from 10 Bell System companies. The average age was 32, the average length of service was 10 years.

A typical day at the school included a 3-hour lecture delivered to all 64 trainees. During the remainder of the day, the trainees were divided into four groups, each permanently assigned to a study room under the supervision of a member of the staff. On the average, two hours were devoted to studying, one hour to class discussions and one hour to practice sessions in the school laboratory.

The general outline of the 35-week course is described below.

I—Fundamentals of Logic and Switching, *6 days*
(Boolean notation, solid state logic, binary arithmetic, functional blocks such as registers, counters, scanners, selectors, etc., memory systems, stored logic)

II—Functional Description of No. 1 ESS, *4 weeks*
(Principles of operation of the various system units, overall organization of the system program, handling of various types of telephone calls, maintenance facilities)
This part was paralleled by a series of sessions using WOSP, as described later.

III—Circuit Fundamentals, *2 weeks*
(Transistor circuits, pulse techniques, use of oscilloscopes, symbols and conventions of circuit drawings)

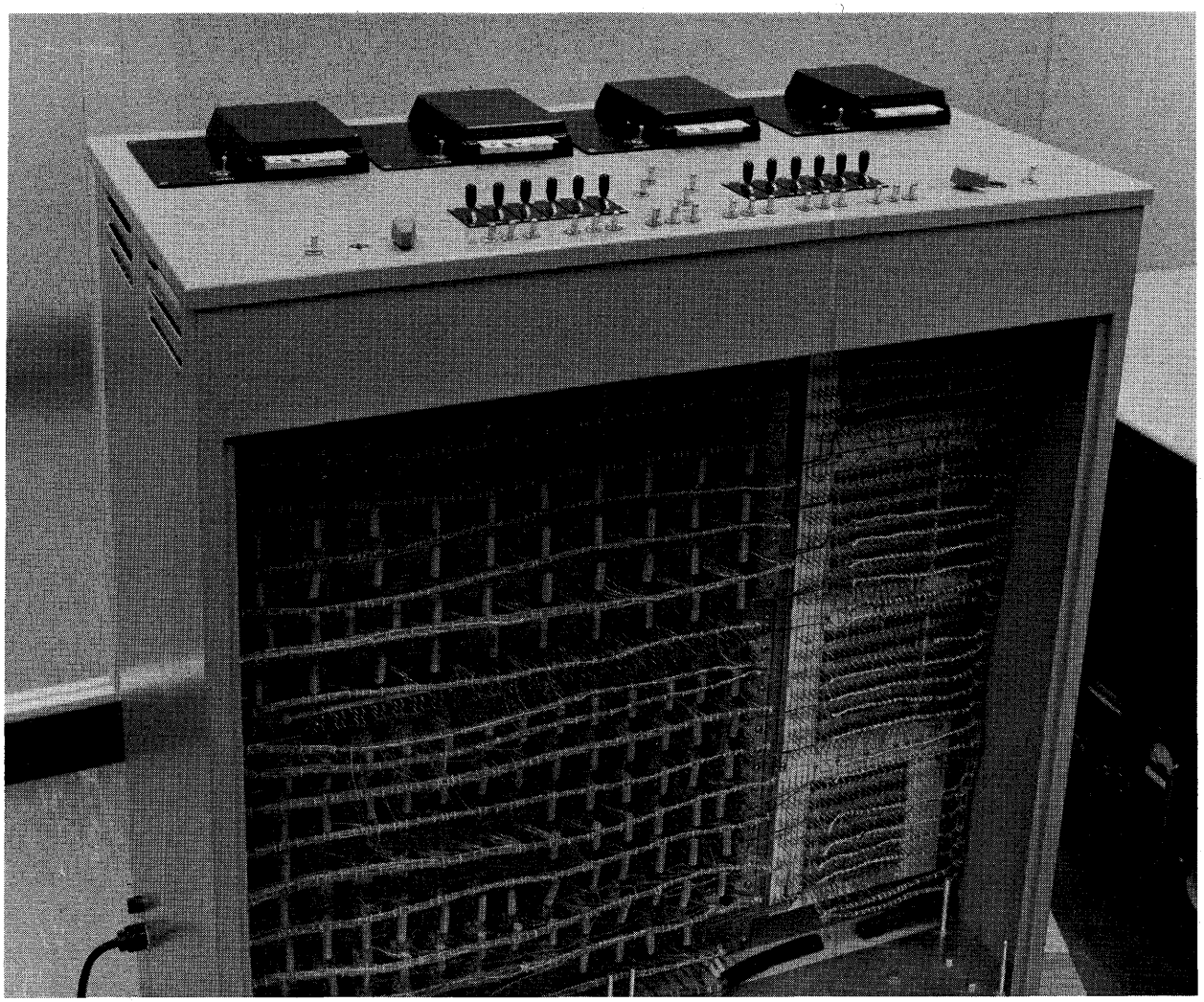


Figure 2. T-cart.

IV—No. 1 ESS Programming, *6 days*
(Detailed description of No. 1 ESS instructions and their use in typical programming situations)

V—Detailed Description of No. 1 ESS Circuits, *15½ weeks*
At the completion of each type of system unit, the associated installation test program was described at a functional level. A total of 6 days was devoted to these programs.

VI—Function Description of Executive and Call Processing Programs, *5 Weeks*

VII—Functional Description of Maintenance Programs—Administrative and Maintenance Procedures, *4 Weeks*

During the last months, in teams of 4 men, the trainees spent about 120 hours each in circuit laboratory sessions. In addition to four oscilloscopes, the equipment used in these sessions included several laboratory setups that had been excerpted directly from various major system units to provide the trainees with a representative cross section of devices and printed-wiring boards. In addition, toward the end of the school, the trainees were given a 2-week field assignment at a number of No. 1 ESS installation sites.

Starting in October 1964, a second school for 69 trainees was conducted for 32 weeks. The format followed was essentially similar to the one described for the first school.

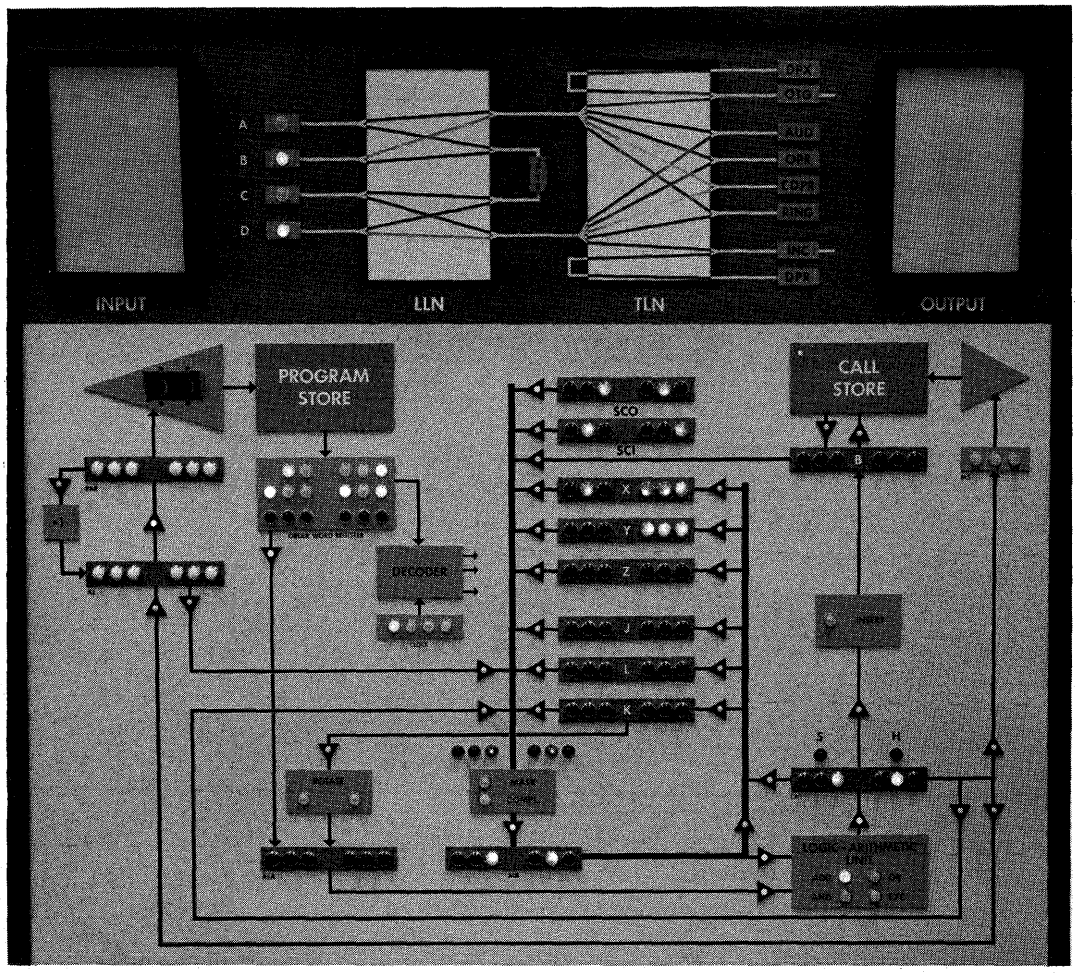



Figure 3. Display panel (side supports on casters not shown).

*Classroom Use of WOSP*

Two basically different approaches were adopted in the No. 1 ESS school with regard to hardware and software. The hardware was described in considerable detail to provide the necessary background for trouble-shooting. The software, on the other hand, was described mostly at a functional level except for a number of typical situations which were covered in complete detail. It was deemed necessary, however, to provide a good knowledge of program instructions and their use in order to develop, not a proficiency in programming, but a sound understanding of system operation and an ability to delve into program details as circumstances might require.

The most basic concepts of program control were introduced at the very end of Part I (Fundamentals of Logic and Switching). This was done with analogies and with a simple paper machine having only four basic instructions. Use was then made of WOSP in a series of 14 one-hour class discussion sessions, which were repeated for each of two groups of 32 trainees. A carefully selected sequence of sample instructions and programs of increasing complexity was presented on the machine in a step-by-step fashion, repeating backtracking as necessary. Each trainee had a copy of the programs on which he could readily follow the operation of the machine since the address of the instruction under execution is displayed in octal form. In this fashion, many concepts, features, and terms basic to No. 1 ESS were gradually presented in a manner that had immediate signifiance to the class. At the end of the demonstration sessions, the trainees were given an opportunity to write some programs and to debug them on the machine. Practically all trainees individually wrote and debugged two of the three

programs that were suggested for this purpose. Several successfully undertook the writing of the third program which entailed storing in memory eight numbers and then rearranging them in ascending order.

During the WOSP sessions and later during part IV (No. 1 ESS Programming), a symbolic notation of the type described in the appendix was extensively used to describe the execution of program instructions. This approach proved very successful, particularly in the second school, even though the trainees did not have an extensive mathematical background. Most trainees found symbolic expressions more convenient than the often lengthy and awkward verbal descriptions of the conventional programmer's manuals, which were also made available to them.

WOSP has been used so far in the two sessions of the No. 1 ESS school and in several sessions of a course organized by Western Electric Company for its installation people. The experience gained definitely indicates that, with the aid of WOSP, trainees with no prior background can acquire in less time a more meaningful understanding of programming than with conventional methods.

## WOSP HARDWARE

Functionally, WOSP consists of five major parts:

1. the Program Store—a random-access read-only memory for storing the program,
2. the Call Store—a random-access read-write memory for storing, updating, and retrieving data or control information,
3. the Processor which executes, one at a time, the instructions in the Program Store,
4. the input facilities,
5. the output facilities.

### *Program Store*

The Program Store consists of 4 motor-driven Hickok Cardmatic card readers combined with a 44 × 17 diode matrix to provide random access to any one of 44 word locations of 17 bits each. Each card reader contains an 11 × 17 array of switches, one for each hole position on a card of the type shown in Fig. 1. When a card is inserted into a card reader, it causes the switches to remain open where holes have been punched and to be closed where no holes have been punched.

The program memory described, beside being conceptually analgous to the program memory of the No. 1 ESS, offers the advantage of being easily and cheaply alterable in the classroom.

### *Call Store*

The Call Store consists of an 8 × 6 array of reed relays with controls to provide random access to 8 memory locations of 6 bits each for either reading or writing.

### *Processor*

The Processor, whose internal block diagram is fully depicted on the display panel (Fig. 3), contains:

(a) facilities for communicating with the program store and the call store,
(b) facilities for decoding and executing individual program instructions, and for determining the next program address,
(c) a number of 6-bit registers for detecting inputs, for controlling outputs or for general-purpose storage of information,
(d) facilities for performing arithmetic additions and logical operations (logical product, logical union, exclusive OR, complementation, and circular shift). The 6-bit words to be operated on may be stored in registers or in call store locations.

Regardless of type, the execution of an instruction requires a single machine cycle consisting of four clock phases. Five different modes of operation are available:

(a) Instructions are executed at the rate of two per second.
(b) Instructions are executed at the rate of one per second.
(c) Each instruction is executed during a 1-second interval. Upon completing the execution, the machine stops and waits for a command from the operator to proceed. When given this command through a cord switch, WOSP proceeds to execute the next instruction, upon completion of which it stops as before.

(d) Instructions are executed with the clock under the control of the cord switch. Thus, the duration of each of the four clock phases is under the direct control of the instructor or trainee operating the machine.

(e) In this mode, 17 toggle switches on the T-cart top panel are used for the manual setting of the instruction to be executed. The control of the clock is as in (d) above.

At any time during the execution of a program, the machine can be stopped at the completion of the instruction in progress; if desired, a different mode of operation may be selected before operation is resumed. Thus, for tutorial or testing purposes, the operation of the machine may be followed at different levels of detail.

*Output Facilities*

The OUTPUT section in the upper right corner of the display panel is a 6 × 2 array of windows and lamps. Through a slot, a transparent sheet with appropriate legends can be placed in front of the windows so that, when the lamp behind a particular window is lighted, a message is presented by WOSP. By means of appropriate display instructions, the programmer can request that certain OUTPUT windows be placed under the control of register X, Y or Z. The 6 windows on the left can be controlled by the 6 bits of register X; of the 6 windows on the right, 3 are controlled by the right half of register Y, 3 by the left half of register Z.

In the upper center portion of the display panel, the two units designated LLN (Line Link Network) and TLN (Trunk Link Network) make up the network display. This display is used as an output for programs processing simulated telephone calls through an elementary switching network. As a "call" progresses, approximate paths are displayed through the line link network and trunk link network. By means of appropriate network instructions, the programmer can request that certain network paths be illuminated under the control of register X, Y, or Z. Figure 4 shows the association between the network paths and the bits of registers X, Y and Z.

*Input Facilities*

In one mode of operation, WOSP has 12 inputs or scan points that can be independently controlled by means of 12 keys on the top panel of the T-cart. These keys are labeled SC00 through SC05 and SC10 through SC15. All inputs are detected internally through two 6-bit scanner registers, SC0 and SC1. The bits of registers SC0 and SC1 can be displayed by lamps in the 12 windows of the INPUT unit in the upper left of the display panel. A transparent sheet can be placed in front of the windows to indicate by means of appropriate legends the significance of the inputs for the particular program that is controlling the machine.

In another mode of operation, WOSP has only six scan points that can be independently controlled by means of the six keys SC00 through SC05. Of these, keys SC00 through SC03 control the state of the four telephone "lines" A, B, C, and D connected to the network (Fig. 3). There are six more scan points that are controlled by keys SC00 through SC03 via paths established through the network. The six scan points are associated with the six network terminals to the circuits labeled JCTR (Junctor Circuit), AUD (Audible Tone Circuit), OPR (Operator Trunk), CDPR (Customer Dial Pulse Receiver) and RNG (Ringing Circuit). For example, at the scan point associated with the CDPR, it is possible to detect the state of whichever of the four lines A, B, C, or D is connected to the CDPR via the network. Figure 4 shows the association between the bits of registers SC0 and SC1 and the scan points provided for the network terminals.

The Program Store, the Call Store, the Processor, the Line Link Network and the Trunk Link Network have direct conceptual counterparts in the No. 1 ESS. Although speed, capacity and logical complexity are vastly different, WOSP is fairly representative of the overall organization and operation of No. 1 ESS, particularly within the Processor.

## BASIC TYPES OF INSTRUCTIONS

The 17 bits that make up an instruction can be divided as follows:

(a) The 11 most significant bits specify the operation to be performed, including certain variations that can be optionally requested by the programmer.

(b) The remaining 6 bits, referred to as the data-address (DA) part of the instruction, usually specify data to be operated on, or the address of an instruction or of a data word.

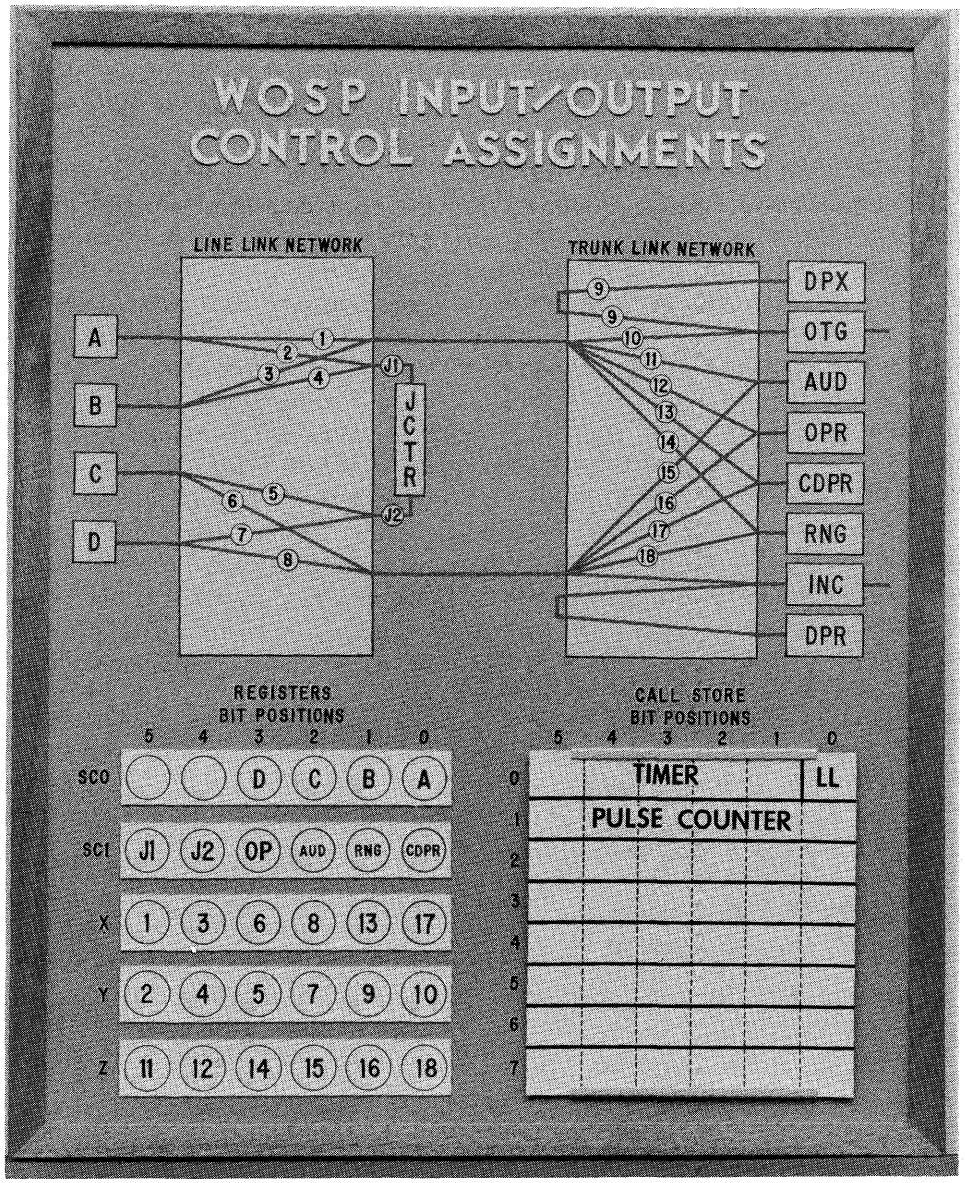



Figure 4. Intput/output assignments.

An instruction in which the programmer has made use of the indexing option is executed using an effective DA field obtained by adding the contents of the DA field and those of a specified index register.

Let W represent the effective DA field. Also, let M represent the memory (call store) location specified by W. The types of instructions available in WOSP can then be broadly described as follows:

1. W-to-Register—Move W into a specified register.

*2. W-to-Memory—Move W into a specified call store location.

3. Register-to-Memory—Move the contents of a specified register into M.

5. Add (Subtract)—
   (a) Add (subtract) W to (from) the contents of accumulator register K and store the result in K.
   (b) Same as 5(a) using the contents of M instead of W.

6. Logical Operations—
   (a) Determine the logical product

(AND) of W and the contents of register K and store the result in K.

(b) Same as 6(a) using the contents of M instead of W.

(c) Same as 6(a) and 6(b) for the operations of logical union (OR) and exclusive OR.

7. Shift—
   (a) Shift the contents of register K to the left (or right) by 1 or 2 positions.
   (b) Rotate the contents of register K to the left (or right) by 1 or 2 positions.

*8. Output Control—Control the network display or the OUTPUT display.

9. Unconditional Transfer—Transfer to program address W. If specified, remove part or all of a network or OUTPUT display.

10. Conditional Transfers—
   (*a) Transfer to program address W if a specified bit of register K is 0.
   (b) Transfer to program address W if, for the result of some previously performed operation, some specified condition is satisfied by the sign S (= 1 if the result is negative) and/or the homogeneity H (=1 if the bits of the result are all equal to 0 or all equal to 1).

Examples of transfer conditions in terms of S and/or H are:

| | | |
|---|---|---|
| P | Plus | S = 0 |
| M | Minus | S = 1 |
| AZ | Arithmetic zero (All zeros or all ones) | H = 1 |
| AU | Arithmetic "unzero" | H = 0 |
| LZ | Logical zero (all zeros) | S = 0 and H = 1 |
| LU | Logical "unzero" | S = 1 or H = 0 |
| LE | Less than or equal to arithmetic zero | S = 1 or H = 1 |
| GE | Greater than or equal to arithmetic zero | S = 0 or H = 1 |

The instructions described above have direct counterparts in the instruction list of No. 1 ESS except for the items marked by an asterisk. (There are several other types of instructions in No. 1 ESS of a general purpose nature or to control input, output, maintenance and other specialized functions.)

In addition to indexing, the programmer can avail himself of one or more of the following options.

*C Option (Complementing).* When this option is used, a data word is complemented before undergoing some arithmetic or logical operation.

*PL Option (Product Masking).* When this option is used, a data word is ANDed with the contents of register L. Some unwanted bits can thus be masked out before some logical or arithmetic operation is performed.

*EL Option (Insertion).* At times, it is necessary to insert information in some bit positions of a call store location without affecting in any way the remaining bit positions. For this reason, instructions for writing into the Call Store include as an option the insertion of a word A into a location B through a mask m. In each bit position where m has a 1, the bit of A replaces the bit initially in B. In each bit position where m has a 0, the bit initially in B remains unchanged. The mask m is provided by register L.

*J Option.* Whenever a transfer to some address X is specified at some address P, the programmer has the option of requesting that address P + 1 be saved in the J register. Thus, if the transfer is made to a subroutine, the use of the J option makes it possible to request at the end of the subroutine that the program return (jump back) to P + 1.

A complete description of the program instructions and their options is given in the Appendix in symbolic form.

## ILLUSTRATIVE PROGRAMS

### *Call Processing Program*

A 44-word program has been written to process "telephone calls" originated by any one of the four lines A, B, C or D (Fig. 4). Line A or B may call C or D and vice versa. The "directory numbers" of lines A, B, C and D are 1, 2, 3 and 4 respectively. Line changes associated with originations, dial pulses and disconnects are generated by manipulating keys SC00 through SC03 for lines A, B, C and D respectively. For instance, to simulate B calling D, key SC01 first is turned on to originate the call, then is turned off and on four times to "dial"

D's directory number and finally is turned off to disconnect. Only one call can be connected at any time.

The program performs the following functions:

1. By scanning periodically the four lines, identify which line, if any, has originated a call.
2. Connect the calling line through LLN and TLN to the customer dial pulse receiver (CDPR).
3. By scanning periodically the CDPR, detect and count the dial pulses generated by the calling line.
4. Recognize the completion of the single digit dialed.
5. Through the LLN and TLN, connect the calling line and the called line to AUD and RNG respectively. (In a real situation, AUD would return an audible tone to the calling line, RNG would apply ringing to the called line.)
6. By scanning RNG and AUD, detect an answer by the called line or an abandon by the calling line.
7. If the calling line abandons, release the network connections and start scanning again for an origination.
8. If the called line answers, release the network connections previously set up. Through the LLN establish a talking connection between the calling and called lines via the junctor.
9. By scanning both sides of the junctor, detect when either of the two lines disconnects. Release the network connections and start scanning again for an origination.

Two words in call store are used to process a call, see Fig. 4. Bit 0 of word 0 is used as a last look (LL) bit to record the previously observed state of the scan point for CDPR. Bits 1, 2 and 3 are used as a timing counter which is recycled whenever a dial change is detected at CDPR and is incremented whenever no dial change is detected at CDPR. Only after the completion of dialing will the counter be incremented until bit 3 becomes 1.

Call store word 1 is used as a pulse counter. Initially, bit 0 is made a 1. Whenever a dial change from 0 to 1 is detected, the contents of the pulse counter are rotated to the right by one position. At the completion of dialing, the pulse counter will have a 1 in bit position 5, 4, 3, or 2 depending on whether the dialed digit was 1, 2, 3 or 4 respectively.

Figure 5 shows that part of the program that controls the detection of dial changes, the counting of dial pulses and the timing to detect dialing completion. The program reaches octal address 14 after an originating line has been identified and connected to the CDPR through the LLN and the TLN.

For each instruction in Fig. 5, the first line shows the octal address and the instruction itself in symbolic form. The second line describes symbolically the processing actions caused by the instruction. The following notation has been used here:

| | |
|---|---|
| COMP(X) | = Complement of X |
| AND(X,Y) | = Logical product of X and Y |
| OR(X,Y) | = Logical union of X and Y |
| EXCL.OR(X,Y) | = Exclusive OR combination of X and Y |
| X→Y | = X is moved into Y |
| M1 | = Call store location #1 |
| K0 | = Bit 0 of register K |

The intent of each instruction is explained after its symbolic description.

*Call Store Test Program*

This program (Fig. 6) is representative of some of the techniques used in testing the No. 1 ESS. Its purpose is to test the ability to read and write properly at each of the eight memory locations of the call store. The operator can set up any two 6-bit words SC0 and SC1 on the 12 input keys of the T-cart. Test words SC0 and SC1 are alternately copied into the call store so that SC0 is written into locations 0, 2, 4, and 6 and SC1 is written into locations 1, 3, 5 and 7. Upon completion of this loading stage, the test proper starts. The contents of the call store locations are read out one at a time and compared with the appropriate test words. Whenever a discrepancy is found between the word from the call store and the associated test word, the machine stops and gives an OUTPUT display to indicate that a failure has been detected. Note that the bit(s) at fault can be precisely pinpointed since both the word read out and its address are shown on the display panel. The operator instructs the machine to resume the testing by pressing the cord switch. When the last call store location has been tested, the machine stops and gives an OUTPUT display to indicate that the test has been completed.

```
(14)  WM1   000 001
      000 001 → M1
Initialize pulse counter in
M1 (Pulses will be counted
by rotating the 1 bit to the
right by one position for
every pulse dialed)

(15)  WM0   000   001
      000 001 → M0
Reset the timer and write 1
in Last Look (LL) bit in M0

(16)  WK  0,SC1
      (0 + SC1) → K
Read scan point CDPR in SC1
and store in K

(17)  XMK  0
      EXCL. OR(M0,K) → K
Excl. OR combination of Last
Look bit in M0 and scan pt.
CDPR makes K0 equal to 1
whenever there is a dial change

(20)  TKB0  31
      Tra. to 31 if K0=0
Tra. occurs if no dial change
has occurred

(21)  WM0  0
      0 → M0
Reset the timer in M0 since a
dial change was detected.
Reset the Last Look bit for
a possible down change.

(22)  WJ  0,SC1
      (0+SC1) → J
Read scan pt. CDPR in SC1 and
store in J

(23)  PWK  0,J
      AND (0+J,K) → K
AND combination of scan pt.
CDPR and change bit in K0
makes K0 equal to 1 if dial
change was upward

(24)  TKB0  16
      Tra. to 16 if K0=0
Tra. occurs if dial change
was down (Last Look bit was
changed to 0 at step 21)

(25)  MK  1
      M1 → K
Read M1 into K to increment
the pulse counter

(26)  QC
Rotate K to the right by
one position

(27)  KM  1
      K → M1
Store incremented pulse
count back in M1

(30)  T  15
      Tra. to 15

(31)  MK  0
      M0 → K
Since no dial change has
occurred, read M0 into K
to increment the dialing
completion timer

(32)  AWK  2
      (2+K) → K
The timer is incremented by
adding 2 so as not to affect
the Last Look bit

(33)  KM  0
      K → M0
The incremented time count
is stored in M0

(34)  TKB3  16
      Tra. to 16 if K3=0
Tra. occurs as long as the
time count is less than 4
```

Figure 5. Telephone call processing program.

## ACKNOWLEDGMENTS

The author wishes to acknowledge the contributions of R. E. Osteen who designed the clock counter, the mode selection logic and the test program for debugging the machine.

## REFERENCES

1. *Bell System Technical Journal,* special issue on No. 1 ESS, Sept. 1964.

## APPENDIX

## A SYMBOLIC DESCRIPTION OF WOSP ORDERS

The general format of WOSP orders is similar to that of No. 1 ESS orders. It consists of an operation field, which specifies the type of processing action to be performed, and a variable field, which is divided into three subfields: DA, R, and LCJ. The DA subfield specifies a number to be interpreted as data or as an address. The R subfield specifies the

| | | | |
|---|---|---|---|
| (00) START | WX | 111 000 | Initialize index register X to –7 for loading. |
| (01) | UWDYS | 000 011 | Stop and display the message "TEST COMPLETED–SET KEYS." |
| (02) | TUD010 | LOAD | Remove the display and transfer to address (03). |
| (03) LOAD | SCOM | 7,X | Move test word from SCO to call store. |
| (04) | WX | 1,X | Increment index register X. |
| (05) | SC1M | 7,X | Move test word from SC1 to call store. |
| (06) | WX | 1,X | Increment index register X. |
| (07) | TCM | LOAD | Check for completion of loading. Transfer to (03) if X is still negative. |
| (10) | WX | 111 000 | Initialize index register X to –7 for testing. |
| (11) TEST | WK | 0, SCO | Move test word from SCO to K. |
| (12) | CMK | 7,X | Compare test word with memory readout. |
| (13) | TCAU | ERROR, ,J | Check for failure. Transfer to (23) if memory readout $\neq$ test word. Because of the J option, return address (14) is stored in register J. |
| (14) | WX | 1,X | Increment index register X. |
| (15) | WK | 0,SC1 | Move test word from SC1 to K. |
| (16) | CMK | 7,X | Compare test word and memory readout. |
| (17) | TCAU | ERROR, ,J | Check for failure. Transfer to (23) if memory readout $\neq$ test word. Because of the J option, return address (20) is saved in register J. |
| (20) | WX | 1,X | Increment index register X. |
| (21) | TCM | TEST | Check for completion of test. Transfer to (11) if X is still negative. |
| (22) | T | START | Return to START. |
| (23) ERROR | UWDYS | 000 100 | Stop. Display the message "FAILURE." |
| (24) | TUD010 | 0,J | Remove the message display and resume testing at either (14) or (20). |

Figure 6. Call store test program.

register to be used when indexing is called for. The LCJ subfield is used to specify program options other than indexing: product masking (PL), insertion (EL), complement (C), and save return address (J).

A complete symbolic description of all WOSP orders is given in Fig. 7. Particularly for complex machine languages such as that of No. 1 ESS, the type of symbolic notation used in Fig. 7 is very convenient for a description of instruction execution that is at the same time very precise and concise.

*Definitions*

| | |
|---|---|
| D | Contents of DA subfield |
| R | Contents of register specified by the R subfield |
| W | Effective DA subfield (= D+ R) |
| M | Contents of call store memory location identified by W |
| X | Contents of register X |
| X′ | Complement of binary word X |
| X + Y | Arithmetic sum of X and Y |
| X – Y | Arithmetic difference of X and Y |
| X · Y | Logical product of X and Y |
| X ⊥ Y | Logical union of X and Y |
| X × Y | Exclusive OR of X and Y |
| X ↙ N | Contents of register X shifted to the left of N bit positions. |
| X ↗ N | Contents of register X shifted to the right of N bit positions. |
| X N | Contents of register X rotated to the left of N bit positions. |
| X N | Contents of X rotated to the right by N bit positions. |
| X → Y | Word X replaces the contents of location Y. If X represents the contents of some register or memory location, it remains unchanged in its initial location. |

| | INSTRUCTION FORMATS WITH OPTIONS | | EXECUTION DESCRIPTION |
|---|---|---|---|
| NETWORK & DISPLAY | PWDX D, $R_2$ | (Y, Z) | IF i=1: D·$R_2$→X, CF } GIVE OUT- IF i=0: $R_2$→X, CF } PUT DISPLAY |
| | PWDXS D, $R_2$ | (Y, Z) | SAME AS PWDX. ALSO, STOP |
| | PWNX D, $R_2$ | (Y, Z) | IF i=1: D·$R_2$→X, CF } GIVE NET- IF i=0: D→X, CF } WORK DISPLAY |
| | UWDX D, $R_2$ | (Y, Z) | IF i=1: D⊥$R_2$→X, CF } GIVE OUT- IF i=0: D→X, CF } PUT DISPLAY |
| | UWDXS D, $R_2$ | (Y, Z) | SAME AS UWDX, ALSO STOP |
| | UWNX D, $R_2$ | (Y, Z) | IF i=1: D⊥$R_2$→X,CF } GIVE NET- IF i=0: D→X,CF } WORK DISPLAY |
| SHIFT / ROTATE | H | | (K ≠ 1) → K, CF |
| | Q D | | (K ≠ 1)·D → K, CF |
| | H2 | | (K ≠ 2) → K, CF |
| | Q2 D | | (Q ≠ 2)·D → K, CF |
| | HC | | (K ≠ 1) → K, CF |
| | QC D | | (K ≠ 1)·D → K, CF |
| | H2C | | (K ≠ 2) → K, CF |
| | Q2C D | | (K ≠ 2)·D → K, CF |
| TRANSFER | TCP D, R, J | (M,LZ,LU,AZ, AU,LE,GE) | IF CF=P: ┘→ W IF (CF=P)·(j=1): PAR+1→J |
| | TKB5 D, R, J | (0,1,2,3,4) | IF K5=0: ┘→W IF (K5=0)·(j=1): PAR+1→J |
| | TUNa,b,c D, R, J | | ┘→W; IF j=1: PAR+1→J REMOVE NETWORK DISPLAY. NOTE 1 |
| | TUDa,b,c D, R, J | | ┘→W; IF j=1: PAR+1→J REMOVE OUTPUT DISPLAY. NOTE 1 |
| | T D, R, J | | ┘→W; IF j=1: PAR+1→J |
| | | | |

NOTE 1.
IF a = 1 REMOVE DISPLAY CONTROLLED BY X
IF b = 1 REMOVE DISPLAY CONTROLLED BY Y
IF c = 1 REMOVE DISPLAY CONTROLLED BY Z

| | INSTRUCTION FORMATS WITH OPTIONS | | EXECUTION DESCRIPTION |
|---|---|---|---|
| MOVE | WX D, $R_1$, PLC | (J,K,L,Y,Z) | $[W\cdot L^p]^c$→ X, CF |
| | WM7 D | (0,1,2,3, 4,5,6) | W→ B, M7, CF |
| | XM D, R, PLC ELC | (SCO, SCI, J,K,L,Y,Z) | IF e=0 : $[X\cdot L^p]^c$→ B, M, CF IF e=1: $X^c\cdot L\perp B\cdot L'$→B,M; $X^c$→CF |
| | MX D, R, PLC | (J,K,L,Y,Z) | M→B; $[M\cdot L^p]^c$→ X, CF |
| | MC D, R, PLC | | M→B; $[M\cdot L^p]^c$→ CF |
| ADD ± | AWK SWK D, R, PL | | K ± $[W\cdot L^p]$ → K, CF |
| | AMK SMK D, R, PL | | M→B; K ± $[M\cdot L^p]$ → K, CF |
| COMPARE | CWK D, R, PL | | K−$[W\cdot L^p]$ → CF |
| | CMK D, R, PL | | K−$[M\cdot L^p]$ → CF |
| LOGICAL | PWK D, R, PLC | | K·$[W\cdot L^p]^c$→ K, CF |
| | UWK D, R, PLC | | K⊥$[W\cdot L^p]^c$→ K, CF |
| | XWK D, R, PLC | | K⊛$[W\cdot L^p]^c$→ K, CF |
| | PMK D, R, PLC | | M→B; K·$[M\cdot L^p]^c$→ K, CF |
| | UMK D, R, PLC | | M→B; K⊥$[M\cdot L^p]^c$→ K, CF |
| | XMK D, R, PLC | | M→B; K⊛$[M\cdot L^p]^c$→ K, CF |
| | EE | | NO OPERATION |

R = SCO, X, J, K  $R_1$ = SCO, SCI, X, Y, Z, J, K, L  $R_2$ = X, Y, Z, K

Figure 7. Symbolic description of WOSP orders.

$X \rightarrow CF$ The sign and homogeneity of word X determine the states of control (C) flip-flops S and H respectively.

W Transfer to program store address W.

IF a = 1:B If Boolean variable a is 1, then expression B applies.

i Boolean variable equal to 1 if the indexing option is used, that is, if a register is specified in the R subfield.

p,e,c,j Boolean variables which are equal to 1 when options PL, EL, C, and J respectively are used.

$R \cdot L^p$ If p = 1, $R \cdot L^p = R \cdot L$
If p = 0, $R \cdot L^p = R$

$X^c$ If c = 1, $X^c = X'$
If c = 0, $X^c = X$

CF = LZ The states of the control flip-flops S and H indicate a logical zero (LZ) condition (S = 0, H = 1). Similarly "CF = P" means that a plus (P) condition exists (S = 0).

As an example, in Fig. 7 the entry

WX D, R1, PLC (J, K, L, Y, Z)

serves to describe the set of similar orders WX, WJ, WK, WL, WY, and WZ. The execution description is given only for WX and is interpreted as follows: The result of indexing, W, possibly masked and possibly complemented is used to set the X register and the S and H flip-flops.

The LCJ subfield of an entry in Fig. 7 names the available options (other than indexing) for the class of orders described by that entry. For example, PLC in the LCJ subfield indicates that only the options of product masking (PL) and complementing (C) are available.

The orders grouped under the category Network and Display are the orders that selectively place the network display (LLN and TLN) or the OUTPUT display under the control of the X, Y, and registers, after performing an optional AND or OR operation as described in the execution description. ANDing makes it possible to remove part of a display previously established; O Ring makes it possible to add to a display previously established.

As an example, consider

PWNX 010010,K.

Since the R subfield is not blank, i = 1. Following the execution description of Fig. 7, the logical product 010010.K = $0K_4 00K_1 0$ replaces the contents of the X register and sets the C flip-flops. Since X appears in the operation field, network paths 1, 3, 6, 8, 13 and 17 (see Fig. 4) come under the control of bits 5, 4, 3, 2, 1 and 0 respectively of the X register. In this example, network paths 1, 6, 8, and

17 will not be illuminated, network paths 3 and 13 will be if bits $X_4$ and $X_1$ respectively are equal to 1.

Once an area of the network display or the OUTPUT display is placed under the control of the X, Y, or Z register, it remains under that control until removed therefrom by an order of the TUN or TUD type. An area of the network or OUTPUT display, when not under the control of its associated register, is all dark regardless of the state of the register.

# CASE: A PROGRAM FOR SIMULATION OF CONCEPT LEARNING

Frank B. Baker
*Research and Development Center for Learning and Re-Education*
*University of Wisconsin*
*Madison, Wisconsin*

The Learning Research and Development Center of the University of Wisconsin is engaged in a long-term multifacet study of concept learning, supported by the U.S. Office of Education. The Concept Attainment Simulation Experiment (CASE) is the facet of this overall effort which utilizes the technology of computer simultation as a vehicle for obtaining a better understanding of the psychological processes involved in the learning of concepts. The long-range goal is the utilization of the insights thus obtained to improve classroom learning. The study of concept learning has a long history within psychology and has received considerable attention in recent years due in part to the book by Bruner, Goodnow, and Austin which delineated strategies for learning concepts. The experimental materials used by Bruner consisted of a finite universe of objects each of which possessed $n$ dimensions; and each dimension could assume $k$ different values. A classification rule (a concept), consisting of a particular combination of dimension values, partitioned the universe into two mutually exclusive sets. In a typical experiment a subject was shown an object which was an exemplar of the set defined by the concept and told his task was to ascertain the classification rule. In order to attain the concept the subject chose objects from the universe and the experimenter indicated the set membership of the object chosen. The object selection-designation procedure continued until the subject could verbalize the correct classification rule and hence the concept had been attained. The experimental situation, the problem to be solved, and learning procedure involved appear reasonably simple and a number of persons have written programs to simulate this type learning experiment—Hunt and Hovland,[2,3] Hunt,[4] Allen,[5] Wickelgren,[6] and Baker. The book by Hunt provides an excellent review of much of the psychological literature relevant to concept learning as well as a discussion of his own simulation program. Unfortunately the existing programs leave one with the disquieting feeling that although they attain concepts, little has been added to our understanding of the psychological processes involved in concept learning. Most of these programs are at best watered-down algorithms and involve very little of psychological importance. Because of the shortcomings of the existing simulation programs a project was initiated to develop a program which hopefully will *eventuate* in something of psychological significance.

The basic approach was to use Bruner's notions about learning strategies, coupled with concepts regarding the structure of behavior from the book by

Miller, Galanter, and Pribam[9] to write a computer program which would attain concepts. This initial program based on semitheoretical grounds would then serve as a stepping-off point for a learning process on the part of the present author.

A system for collecting data was established which consisted of a closed feedback loop, with the simulation program at one end and protocol gathering during experiments involving human learning at the other end. Within the computer program, certain routines may be based upon a priori grounds or represent areas not clearly understood. In order to get better insights into such areas, questions are used during the protocol gathering which will elicit verbalizations relevant to those points. Thus, the computer program guides the production of information within the protocol which is subsequently used to modify the program itself. By making an extremely close connection between the computer program development and the learning experiments with human subjects the hope is to obtain a better understanding of the psychological processes involved. Having set the broad context within which the project operates, let us next turn our attention to the actual computer program involved.

## THE CASE PROGRAM

### *Memory Structure*

During the early phases in the evolution of the CASE program it became obvious that one of the keys to the problem was an adequate representation of the structure of human memory. The psychological literature contains a considerable body of material related to memory and much of this was studied to ascertain an appropriate structural form of memory. The result of this search was to design a memory consisting of three levels: Working memory (WM), short-term memory (STM), and long-term memory (LTM). The working memory is a unit which serves two functions. First, it holds all information received from the external environment until it can be analyzed and re-coded for transmission to a more permanent level of memory. Second, it serves as a buffer memory for holding information which is created within the subject and must be passed from one information-processing routine to another. In this buffer mode it provides certain higher-level routines with contextual information which is used to guide program flow. The short-term memory is semipermanent and retains information relevant to the current state in the learning of a particular concept. Short-term memory can receive inputs only from routines which re-code and transmit the contents of working memory or long-term memory. Long-term memory will contain information re-coded from short-term memory concerning concepts learned and how they were learned but at the present time only working memory and short-term memory have been programmed. Figure 1 illustrates the communication paths within the memory structure. The only means of communication from STM and LTM to the external world is via the output channel. For example, the subject tells the experimenter which object he has selected via this channel but the experimenters designation of the set membership of the object is received by the subject via working memory.

The internal structure of short-term memory consists of lists having a somewhat unusual IPL-V structure which has proven extremely useful. The

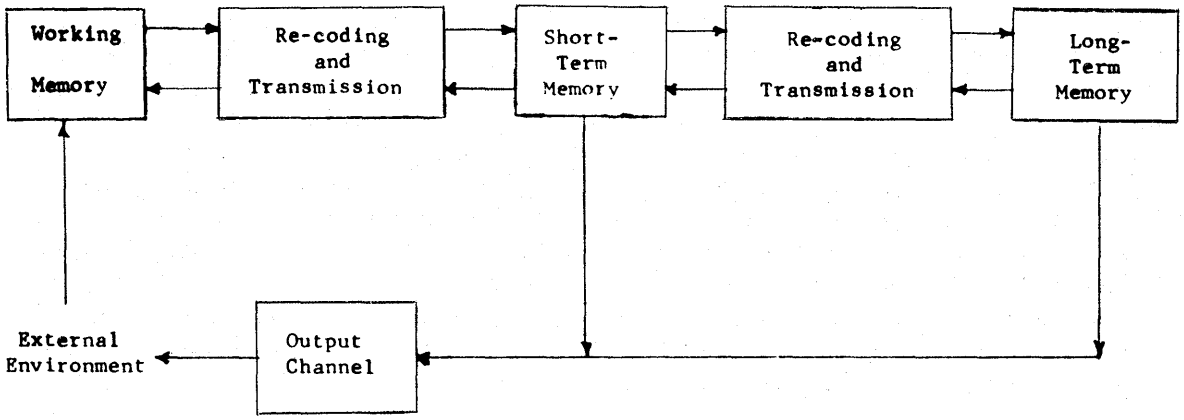



Figure 1. Communication paths within CASE memory structure.

structure employs two levels of attributes; the class attributes which represent a rather broad description such as the permanent characteristics of an object; and specific attributes such as an object's serial position in the external environment, thus providing a detailed description within the class attribute. Table 1 illustrates a typical list within STM. The description list 9-0 describes M13, description list 9-1 describes the symbol E0 on M13. The description list 9-1 contains class attributes, such as A2, and its attribute value list V2, which is merely a storage device for symbols whose function is to hold a description list containing the specific attributes and their values (A4, V4: A5, V5). The value list of the class attribute is a push-down list whose top symbol always represents current information.

Table 1. Typical memory list in short-term memory.

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| M10 | 9-0 | | | | | | | | | |
| | E0 | | | | | | | | | |
| | E1 | | | | | | | | | |
| | E2 | 0 | | | | | | | | |
| 9-0 | 0 | | E0 | 9-1 | | | | | | |
| | A1 | | | X1 | | | | | | |
| | V1 | 0 | | X2 | 0 | | | | | |
| | | | 9-1 | 0 | | | | | | |
| | | | | A2 | | | | | | |
| | | | | V2 | | | | | | |
| | | | | | | V2 | 9-2 | | | |
| | | | | | | | Y2 | | | |
| | | | | | | | Y1 | 0 | | |
| | | | | | | 9-2 | 0 | | | |
| | | | | | | | A3 | | Y2 | 9-3 | 0 |
| | | | | | | | V3 | 0 | 9-3 | 0 | |
| | | | | | | | | | | A4 | |
| | | | | | | | | | | V4 | |
| | | | | | | | | | | A5 | |
| | | | | | | | | | | V5 | 0 |

Notice in Table 1 that the list structure is symetrical in form to the upper left and lower right of the dotted line. The symmetry enables one to write simple routines which function for a module of memory regardless of the level at which the module occurs within the structure. There are four such basic memory routines which do all of the STM input and output:

1. Remember a name.
2. Remember something about that which has been named.
3. Recall a name.
4. Recall something about that which has been named.

At the present time we have not attempted to include LTM or to introduce forgetting or interference; however, we anticipate at some point building such mechanisms into the memory structure.

*Program Structure*

The CASE computer program has been designed with an expandable hierarchial structure whose depth depends upon the level of sophistication obtained in understanding the learning process. At the present time there are four levels, with each level being tested within the next higher level as an IPL-V list structure. The list structure representing the learning process is presented as input to a special interpreter,[9] which executes the symbols in the structure and performs a number of housekeeping functions. The upper level (S) specifies what Bruner et al[1] refer to as a strategy, and is a list of symbols which represent major procedures within a strategy. The next lower level is the procedure level (Z-D) which is a list of symbols representing the processes combined to accomplish a given procedure, such as searching the external environment for an object having certain characteristics. The next lower level (P-Q) consists of information-processing routines written in IPL-V and is the lowest level that the program can manipulate at run time. The fourth level (R) consists of basic information processing modules coded in IPL-V which a programmer can use to manually write new P-Q level routines. The R's are subroutines which do such things as compare, test for the presence or absence of information, etc. Within each level it is necessary to maintain a sharp distinction between routines which perform operations (Z's and P's) and those which provide decision-making information (D's and Q's). It has also been found necessary to defer decision making upward to the next higher level for action. It should be noted, that all levels above the P-Q and R levels merely consist of symbols, hence one writes IPL-V only at the lowest level—a fact which has many implications for how one studies the learning process.

In that the P-Q level routines are basic information-processing routines, they can be used in a wide variety of situations within the program where the processing is identical but the information, its source, and its disposition differ. Because of this characteristic of the P-Q level routines, one is faced with the problem of how to use the same routine in a number of different contexts without developing a significant amount of situationally dependent IPL-V coding. The solution devised uses a pseudo code whose description list contains the inputs, outputs, and characteristics of the routine. The inputs to a routine are determined by a higher-level routine called a *contexter* which in the case of the P-Q level routines uses the contents of working memory to determine what the inputs to the processing routine should be for a given situation. Currently the outputs are normally placed in working memory although other options are possible. Although we have not done so yet it appears feasible to put on the description list of the routine a specification of the kind of information processing the routine is capable of performing. Table 2 shows a P level routine before and after the context program has functioned. Once the inputs have been determined from the context, the routine is returned to the interpreter for execution. Although it has not been done, the concept of context routines which operate at all levels within the program structure appears feasible and the context routine at the highest level would be a plan to create plans such as suggested by Miller, Galanter and Pribam.[8]

Table 2. Pseudo Code System.

| Before Context Program | | | | | After Context Program | | | | |
|---|---|---|---|---|---|---|---|---|---|
| P31 | 9-0 | | | Pseudo routine | P31 | 9-0 | | | |
| | P30 | 0 | | Executable routine | | P30 | 0 | | |
| 9-0 | 0 | | | | 9-0 | 0 | | | |
| | A7 | | | Inputs | | A7 | | | |
| | V7 | 0 | 0 | | | V7 | M11 | | |
| | A8 | | | Outputs | | | M12 | | |
| | V7 | M1 | 0 | | | | A6 | 0 | |
| | | | | | | A8 | | | |
| | | | | | | V7 | M1 | 0 | |

*Case Mark 3 Mod 1*

The preceding discussion indicates the development of what might be loosely described as a programming system rather than a unique computer program such as LT, EPAM, or the previous concept attainment programs. The inherent flexibility in the system makes it difficult to discuss the CASE program per se. Because of this difficulty, periodically a particular configuration is set aside, given a mark and mod number, and documented. At the time this paper was written CASE Mark 3 Mod 1 was the operational program which implemented what Bruner et al[1] called the conservative focusing strategy. The implementation of this strategy required 9 routines at the Z-D level and 13 at the P-Q level. Table 3 lists some representative routines at these two program levels.

Table 3. Representative Routines Currently Implemented.

| | |
|---|---|
| | *Procedure Level* |
| Z1 | Form criteria for choosing an object |
| Z2 | Find object in external environment meeting criteria |
| D1 | Determine if concept should be offered |
| Z4 | React to class membership of object |
| | *Process Level* |
| P3 | Vary value of a dimension |
| P6 | Remember something about object named |
| Q1 | Test for specific value of a given item of information |
| P12 | Collect items having common attribute |

Fundamentally the program is still an algorithm as it very efficiently learns every concept attempted in a minimum number of object choices. Although it currently shares this fault with its published predecessors, we feel its internal structure is more psychologically oriented and the potential for nonalgorithmic behavior exists.

## SOME RETROSPECTS

*Program Characteristics*

When one reviews the history of the CASE program it becomes quite clear that a subtle process is in effect. Namely, as one's understanding of the learning process increases, the computer simulation program changes from routines which perform a

large block of the concept attainment process to a number of short routines which can be widely employed. In the CASE program such a change has been dramatic at the P-Q level from Mark 1 Mod zero to Mark 3 Mod 1. The cynic will counter that we are merely learning how to code IPL-V but I do not believe this is the only basis, as the change has been effected primarily on psychological grounds rather than coding considerations. In fact, separate symbols are used to designate routines which are the result of IPL-V rather than psychological considerations and the former routines are quite rare. The character of the subroutines in the CASE program have also changed from being highly specific to the Bruner type experimental situation to being reasonably independent of the experimental situation. They are, however, dependent upon the basic memory structures defined earlier. The situationally dependent tasks still get performed but the computer program is problem specific at a higher level than was previously true.

*Outcomes of the Case Program*

There have been a number of outcomes of the CASE effort which are as follows:

1. The hierarchial structure of the processing routines and what appears to be a parallel structure in context routines has led us into a continual search for logical units within the learning process. Behaviors which once seemed quite dissimilar have been decomposed and found to share a number of basic information processing modules. As a result we are slowly acquiring a better understanding of the psychological processes involved in concept learning.
2. The development of the CASE program has generated ideas for classical psychological experiments in a number of areas as a result of problems arising during the development of certain subroutines. Topics such as the role of dominant dimensions in a subject's learning behavior and the lack of independence among dimensions in the Bruner experimental materials have been elicited. It appears as though an important outcome of computer simulation is the generation of ideas which can be researched in the usual psychological experimental setting.
3. A completely unexpected outcome has been that we rarely make a production run on the computer, a fact which seems anomalous in a computer simulation project. What has happened is that enormous numbers of man hours have been devoted to gathering and studying protocols, to the development of programming techniques in order to implement the next level of sophistication within the system, and to analysis of the computer program itself. These activities plus the lack of variability in the learning behavior of the CASE program at this point in time have resulted in a relatively few production runs.

## CONCLUSIONS

The CASE project has *not* been conceived as an effort which will produce immediate spectacular results, rather we view it as a slow developmental process. The memory structure, the program structure, the interpreter, and the contexter are basic concepts which we feel will enable us to continuously improve the sophistication of the program as our understanding of the concept learning process improves. Although it is difficult to single out specific accomplishments of great psychological importance, a certain modicum of progress has been made along these lines. At the current time the computer program is primarily a medium for expressing and storing the insights and understandings of the concept learning process which we have acquired. Uhr[10] has previously indicated that psychological theories might be expressed in the form of computer programs and our experience to date tends to substantiate this point of view.

## REFERENCES

1. J. S. Bruner, J. J. Goodnow, and G. A. Austin, *A Study of Thinking,* Wiley and Sons, New York, 1956.

2. E. B. Hunt and C. I. Hovland, "Programming a Model of Human Concept Formation," *Proceedings of the Western Joint Computer Conference,* pp. 145-155 (1960).

3. ——— and ——— "Computer Simulation of Concept Attainment," *Behavioral Science,* vol. 5, pp. 265-267 (1960).

4. ———, *Concept Learning: An Information Processing Problem,* Wiley and Sons, New York, 1962.

5. M. Allen, "A Concept Attainment Program That Simulates a Simultaneous-Learning Strategy," *Behavioral Science,* vol. 7, pp. 247-250 (1962).

6. W. A. Wickelgren, "A Simulation Program for Concept Attainment by Conservative Focusing," *Behavioral Science,* vol. 7, pp. 245-247 (1962).

7. F. B. Baker, "An IPL-V Program for Concept Attainment," *Educational and Psychological Measurement,* vol. 24, pp. 119-127 (1964).

8. G. A. Miller, E. Galanter and K. H. Pribam, *Plans and the Structure of Behavior,* Holt, Rinehart, and Winston, New York, 1960.

9. F. B. Baker and T. J. Martin, "An IPL-V Technique for Simulation Programs," *Educational and Psychological Measurement* (in press).

10. L. Uhr, "Pattern Recognition Computers as Models for Form Perception," *Psychological Bulletin,* vol. 60, pp. 40-73 (1963).

# A 375-NANOSECOND MAIN MEMORY SYSTEM UTILIZING 7-MIL CORES

G. E. Werner and R. M. Whalen
*International Business Machines Corporation*
*Systems Development Division*
*Poughkeepsie, New York*

## INTRODUCTION

It has been postulated that ferrite and thin film memories should have approximately the same ultimate speed limits.[1] For this to be true in the case of conventional toroids, it is clear that the storage element must be made smaller and the storage array made more compact without increasing noise.

In the past, techniques such as partial switching and two cores per bit[2,3] have gone a long way toward reducing the time and energy associated with the switching of storage cores, and these combined with a word-organized storage array have enabled the technology to reach 0.5-microsecond cycles with core sizes of 0.020 inches inside diameter. At this point, however, a number of problems were encountered which became limiting factors when higher speeds were attempted.

Ferrite materials development has resulted in very little progress in reducing the switching coefficient of ferrite and there is little hope of reducing it, say by an order of magnitude, to make it comparable with thin films. Therefore, the alternative is to increase the coercivity of the material to obtain faster coincident-current switching. With 20-mil cores this increases power dissipation in the drive system beyond anything achievable with economical semiconductors. In addition, core heating reaches a point where reasonable temperature stability is difficult to maintain.

With bit densities of about 1000 bits per square inch, which is a typical density for 20-mil cores, the read, write, and digit wires can become excessively long for high-speed operation. Transmission time then becomes a factor which is significant in the cycle. It also becomes necessary to terminate drive lines which increase the power and voltage requirements. Digit and sense wires are restricted to as few as 1000 bits in order to avoid excessive transient recovery times and noise pickup at faster speeds. In addition, the open wire, stacked plane structure, which is an outgrowth of low-bit densities, presents cooling problems. It also has a form factor which maximizes the amount of cabling required to the drive and sense circuits.

The memory described in this paper incorporates a small, high-coercivity core in a new array geometry which circumvents many of the problems which previously limited memory speeds. Repetition rates of 2.66 megacycles have been achieved in a memory unit with a capacity of 8192 words of 72 bits each.

The access time with conventional SLT logic circuits is 200 nanoseconds. This is achieved by using

a two-dimensional organization. To optimize word drive and bit-sense circuit count, the memory was organized as 2,048 words of 288 bits each. Consequently on each readout, 4 72-bit words are set into the data register, one of which is selected and routed to the memory bus.

This paper discusses core operation and characteristics and the array, drive and sense systems; finally it comments on the design necessary to increase the repitition rate to 4 megacycles.

## CORE CHARACTERISTICS

The storage device used is a ferrite toroid, size 7.5 × 12.3 × 2.5 mils. This core has typical coincident current squareness and is operated in a one-core-per-bit mode. The total magnetic time was designed to be 250 nanoseconds, including rise and fall transitions of the drive currents. With the core fired for a d-c threshold of 250 milliamps, nearly full switching is achieved over the range of drive currents used.

The write operation is a "permit" type of selection. A one is stored by a summation of write and bit currents and a zero is stored by applying only a write current. Figure 1 shows a d-c *S* curve and two narrow pulse *S* curves for a typical core, which illustrates this operation. The pulse width notations of 100 and 50 nanoseconds on the narrow pulse curves denote the pulse duration at the 90 percent points and, therefore, do not include the rise and fall times. Since 25-nanosecond transitions were

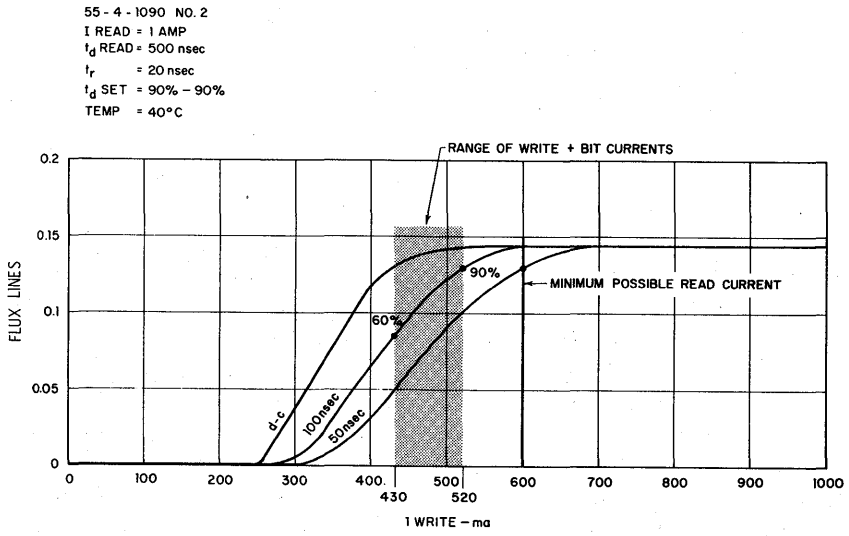



Figure 1. Flux versus set current.

designed for the memory, these translate into widths of 150 nanoseconds and 100 nanoseconds, respectively, at the 10 percent points. The d-c *S* curve determines the maximum bit current amplitude. The bit current is limited to a maximum of about 250 milliamps because many bit pulses must not disturb the core. The 100-nanosecond *S* curve determines the maximum amplitude of the write word current. The write current must be kept below the threshold of the 100-nanosecond curve but it can exceed the d-c threshold[4] because in a 2-dimensional memory a core can be disturbed by only one write pulse before it is reset. The maximum is about 280 milliamps.

The minimum readout current must be sufficient to completely reset the core from a maximum flux state achievable in writing. With a 100-nanosecond (at 10 percent points) readout pulse a 600-milliamp pulse is required.

The write current duration is made longer than the read current duration in order to incease outputs by maximizing the amount of flux switched in the core. One to zero ratios greater than 4 to 1 were achieved under the "worst-case" sequence of pulses. Figure 2 shows typical output signals over a range of operating currents. The gray zone designates the operating range of the memory for write and bit currents.

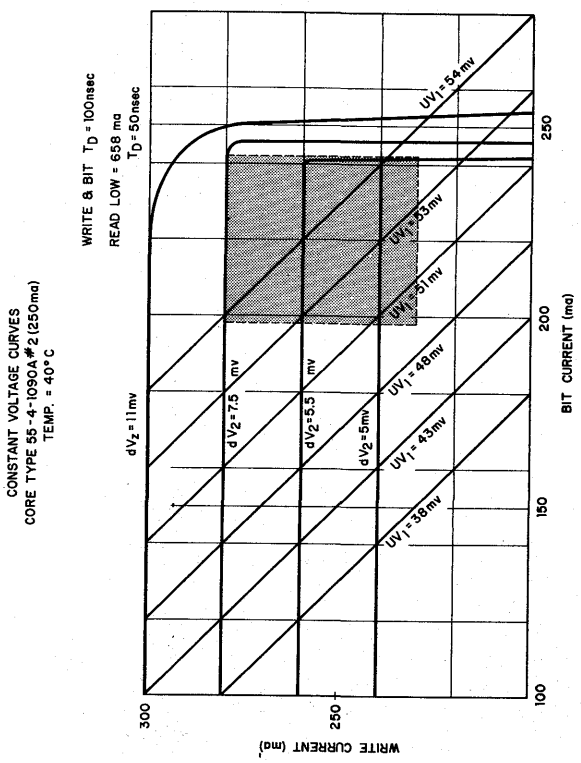



Figure 2. Constant voltage curves.

## ARRAY ORGANIZATION

The array is best described by referring to Fig. 3. Each core is threaded by two wires, a word wire and a bit-sense wire, and assembled on 15-mil centers. To facilitate assembly, 9,216 cores are assembled at one time to form a matrix 128 × 72. Four matrices are mounted on a partitioned core plane (Fig. 4). The partitions have printed lands to permit the matrices to be interconnected to form a continuous plane which is 256 × 144. Eight core planes are mounted, side by side, in a 2 × 4 matrix, on each side of a single aluminum ground plane and interconnected to form a continuous planar structure with is 1024 × 288.

The ground plane contains milled grooves, which permits the core plane frames to be countersunk sufficiently to allow the cores to rest on the surface of the ground plane. The depth is then finely adjusted so that the inner wire (bit-sense) is about 3 mil from the surface of the ground plane.

The core areas are coated with an epoxy resin with a plasticizer added to prevent hardening. This provides a thermal bond to the ground plane, increases the effective dielectric constant to about 1.7, and damps magnetostrictive ringing. To establish a constant array temperature, a temperature-controlled liquid is circulated through a tube imbedded in the ground plane. The control apparatus is remotely located and the liquid is fed to the ground plane through flexible tubes. Fully assembled, the array has a bit density of 4000 bits per cubic inch. Typical electrical characteristics are as follows:

| | Bit Wires | Word Wires |
|---|---|---|
| Length . . . . . | .21 inches | 9 inches |
| Resistance (20°C) . . | 3.8 ohms | 1.8 ohms |

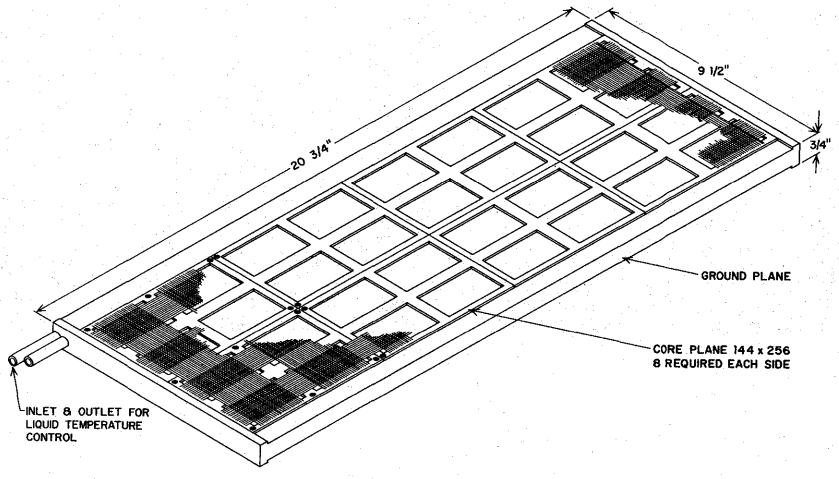



Figure 3. 8K Array assembly.

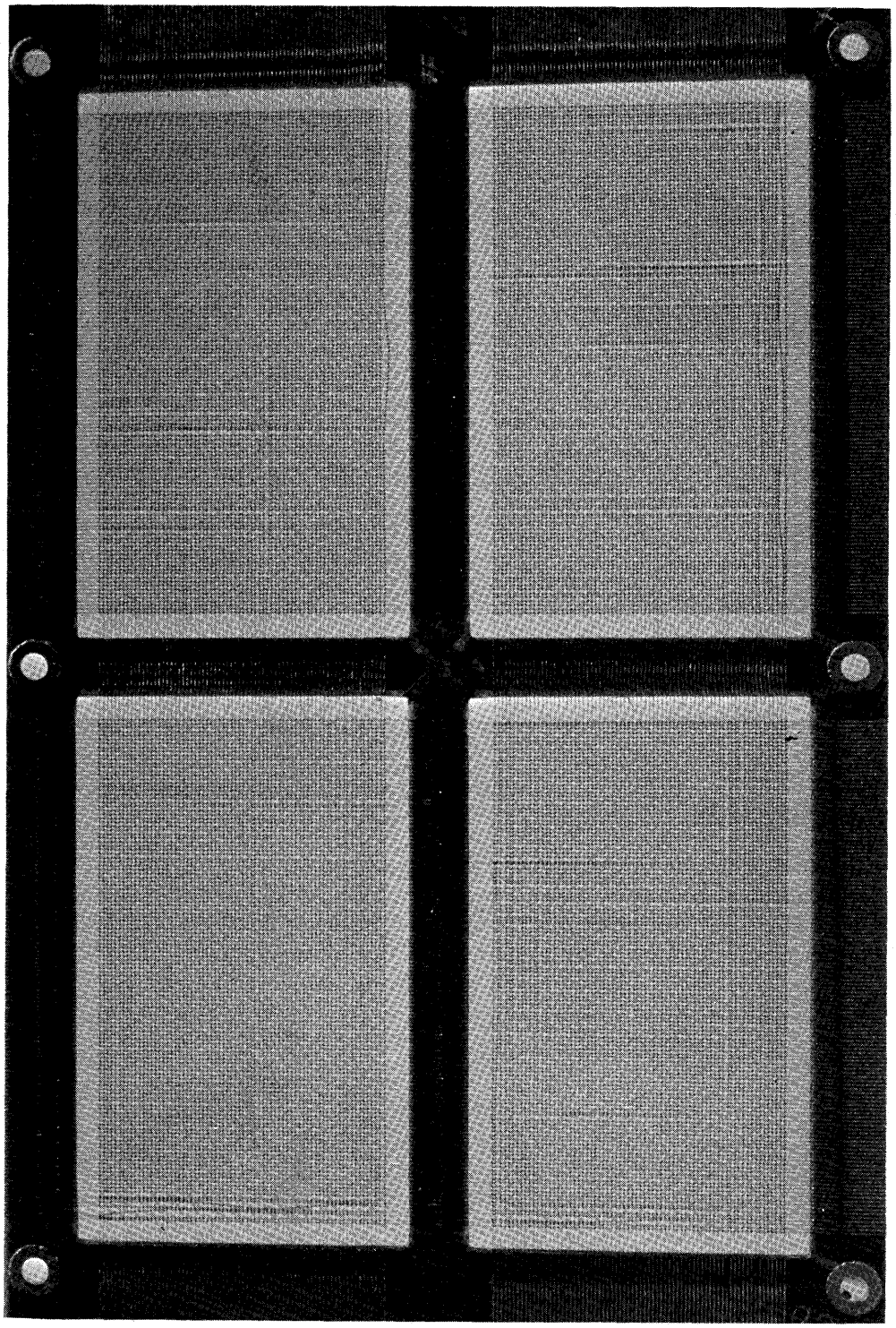


Figure 4. Plane frame.

| | | |
|---|---|---|
| Characteristic impedance | .94.5–102 ohms | 117 ohms |
| Inductance | .510–595 nh | 166 nh |
| Transmission delay | .5.4–5.8 nsec | 1.4 nsec |
| Mutual inductance to adj. wire | .60 nh | 25 nh |

Impedance, inductance and delay are a function of the flux state of the cores on a specific line. The

low values shown in the above table for the bit wires are for a condition of all zeros. The high values are for a condition of all ones. An additional variation in these characteristics can be expected due to geometrically caused variations in wire capacitance and inductance of the lines. In the case of impedance this was found to be about 2 ohms.

The cross-wire capacitance is on the order of 0.025 picofarads. Capacitive coupled noise which is due primarily to the time rate of change of read current has been determined to be 3 millivolts and is negligibly small. For slow time-rate of changes in word line voltage, such as that due to the cores switching, the amplitude of the capacitively coupled noise is not only lower but is mostly canceled as common-mode at the input to the sense amplifier because of the relatively short transmission delay on the bit-sense line.

The bit-to-bit coupling is limited to about 10 percent of the self inductance because of the close spacing of the bit-sense wires to the ground plane. This results in from 1.5 to 2 millivolts of induced noise on the sense line at read time when a core is switched on an adjacent line. Bit current on an adjacent line induces current pulses of about 5 milliamps on the line.

## WORD DRIVE SYSTEM

There are 2048 word lines (1024 on each side of the ground plane) in the array, one of which must be selected and driven with a bipolar current pulse during each cycle. Each line contains 288 cores and is electrically short enough to permit the line to be grounded at one end and driven as an inductive load. In order to achieve high performance and still maintain some degree of economy with a matrix selection system, the lines are divided into 8 groups of 256 lines each. Each group is driven by 256 linear transformers. A transformer is used for each word line primarily to permit common gating for read and write drives. Another advantage is that it reduces some of the noise voltages coupled into the memory array from the word drive system.

The primaries of the 256 transformers are connected in a diode matrix and driven by 16 gates, 16 read drivers, and 16 write drivers as shown in Fig. 5. The packaging is arranged such that the circuit boards butt up against the array ground plane (see Fig. 10), and the interconnections between the transformer outputs and the word lines are made with a short printed-wire strap. The ground plane in the circuit board is connected to the array ground plane in a similar manner.

In the quiescent state, the gate buses are maintained at a potential of +45 volts with the read and write buses at a slightly negative potential, so that the diodes are back-biased and nonconducting. When a word is selected, the appropriate gate bus is switched to ground potential leaving the diodes in a slightly reverse-biased condition. When a pair of read and write buses are selected and successively driven positive, the diodes of the transformer located at the intersection of the active word drive and gate buses will be forward-biased and conduct current. The amplitude and regulation of the word currents is controlled by a biased nonlinear transformer located in each drive circuit which assumes a high-impedance state when the current pulse reaches the proper amplitude. The back-to-back diodes, inserted in series with the secondary, reduces to a negligible level the d-c shift that would result from the difference between the read and write currents. The waveform in the top trace of Fig. 6 is an oscilloscope display of the read and write currents as they appear in the secondary of the transformer. The read current, which is about 50 percent loaded, is beginning to show the effect of core back voltage.

## BIT-SENSE SYSTEM

A common bit-sense system is used in the memory to minimize the number of windings of the core plane. Figure 7 illustrates one bit position of this system. Corresponding bit wires on either side of the ground plane are connected together at one end and driven as a pair. This results in a requirement for 288-bit drivers for an 8K system. Each bit driver must supply a maximum of 500 milliamps. The bit lines are short enough to be driven unterminated at the far end and still achieve the desired transition times. The sense signal is detected across the pair at the far end. Diodes shunt the bit currents past the sense amplifier during a write operation, but are essentially out of the circuit during sensing. A balancing transformer in series with the diodes forces a balance of the currents in the two lines. The terminating network at the output of the bit driver terminates the positive common mode signal generated by the bit driver turnoff. The sense amplifier input impedance terminates low-level

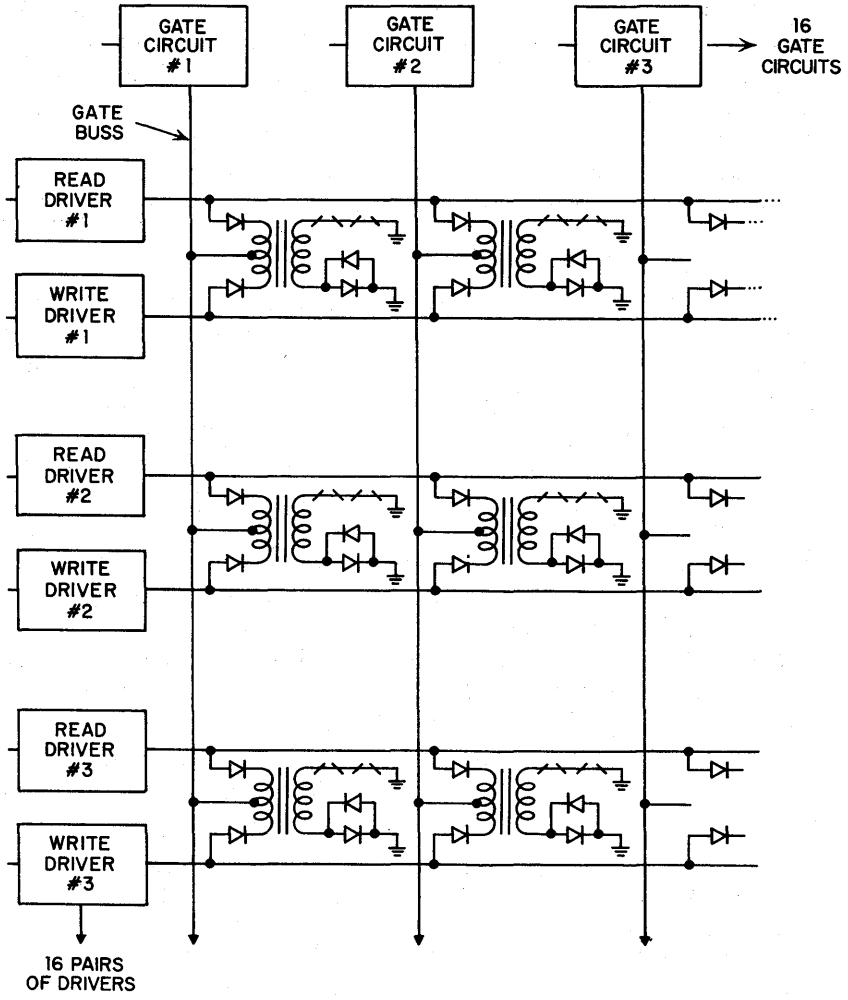



Figure 5. Word drive schematic.

common- and difference-mode noise on the sense line following the recovery of the shunt diodes and helps recover the line for sense time. The recovery time of the line under worst-case unbalance is about 90 nanoseconds.

The waveform in the bottom trace of Fig. 6 is an oscilloscope display of the bit current prior to splitting in the bit line pair. The step in the leading and trailing edge shows the reflection that occurs in the unterminated line.

The waveforms photograph (Fig. 8) shows the sense signal as observed at the input terminals to the sense amplifier. Ones and zeros are superimposed in this picture in order to depict an average signal-to-noise ratio.

The bit drivers and sense amplifiers are each connected to the array through a length of twisted pair and a "transition" board. The transition boards, which are butted to the array at either end, carry printed lines and provide a means for fanning out the bit-sense lines to socket pins into which the twisted pair is plugged (see Fig. 10). Twisted pair connects the transition board to the bit and sense circuits. The transition board lines and the array wires are interconnected through a short printed-write strap. The circuit-to-array wiring increases the length of the sense loop by 44 inches and the bit line by 28 inches. An attempt was made to match the characteristic impedances between array and circuits. Some mismatches were introduced

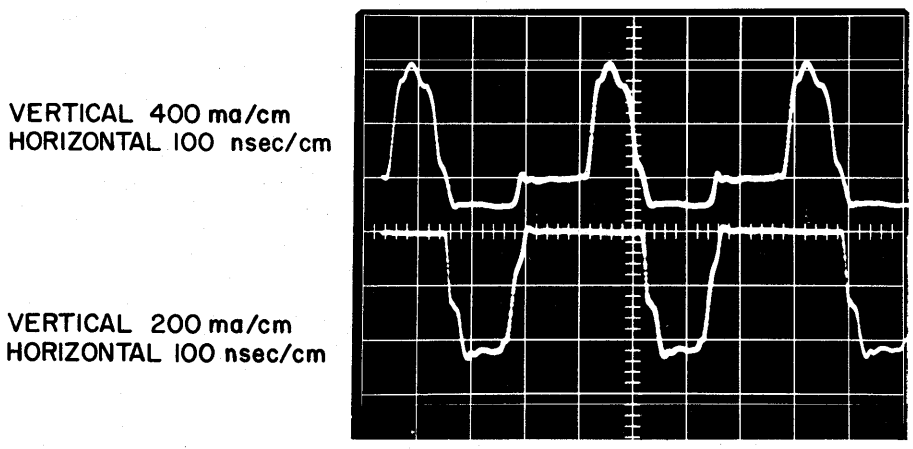



Figure 6. Drive current waveforms.

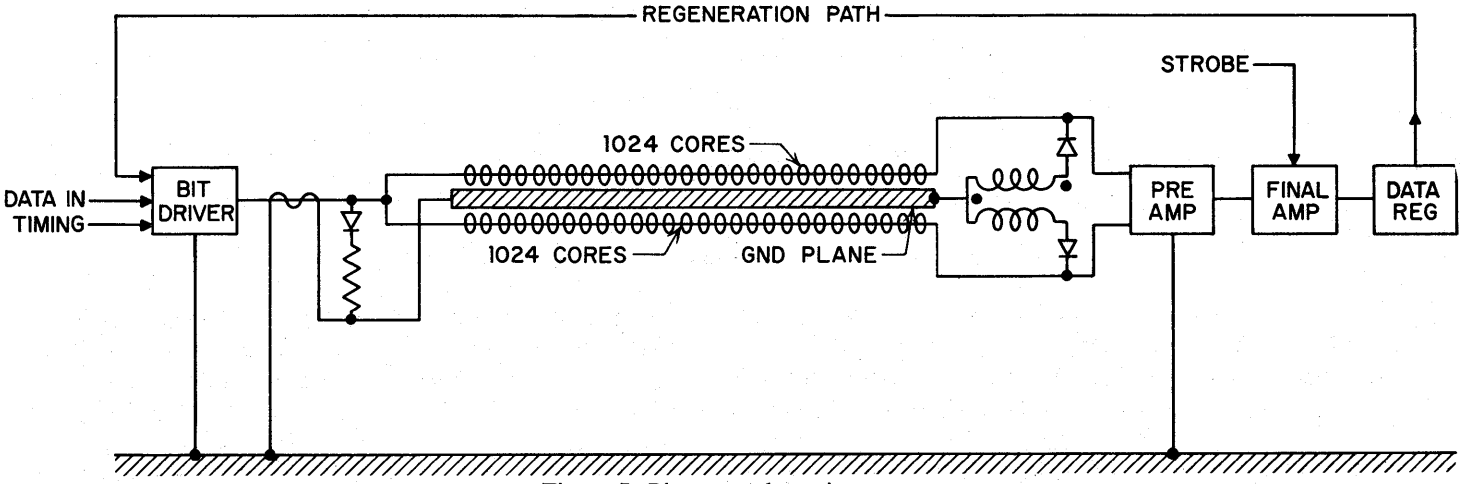



Figure 7. Bit-sense schematic.

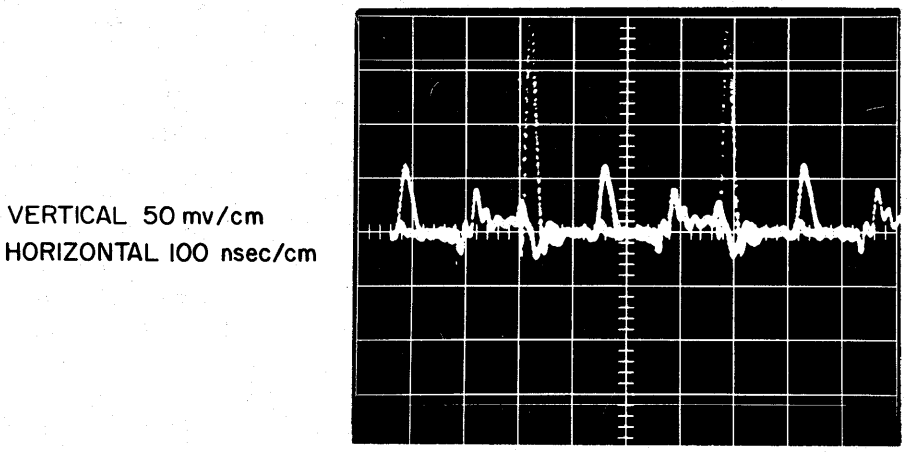



Figure 8. Sense signal waveforms.

but they are electrically so short they have negligible effect.

## MEMORY ORGANIZATION

The 8K memory unit is self contained in that it has its own clock, address registers, decoding, data register, data-in and data-out controls, and byte control. These are well known and will not be described. The computer need only provide address information and a select memory pulse to initiate a cycle. The internal timing of the memory can be traced from the timing chart in Fig. 9. With SLT circuits, an access time of 200 nanoseconds was achieved. In order to achieve a 375-nanosecond interval between accesses, an overlap technique is

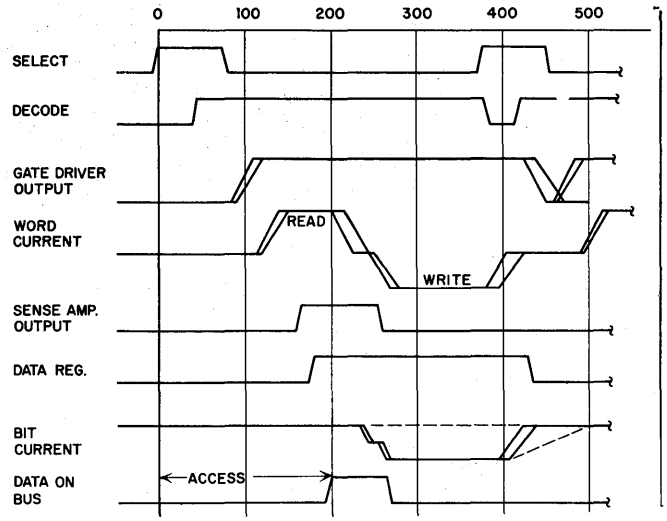



Figure 9. Ferrite memory timing cycle — 375 nanoseconds.

used. Because of preread delays, it is possible to begin a new cycle before the preceding one is completed. However, it is not possible to take full advantage of overlap in this memory because it is limited by the sense line recovery.

Figure 10 shows a gate which contains an 8K memory unit. Two such gates can be housed in a frame to make a 16K memory unit. One array temperature control unit is shared by two arrays and is also housed in the frame. All power supplies are external.

## CONCLUSION

The construction and operation of this memory demonstrated the feasibility of fabricating 7.5-mil cores and assembling them into a compact array. The array assembly eliminated many of the problems which previously limited ferrite speed. Most of the speed-limiting problems now seem to lie with the storage core, the circuits, and packaging external to the array.

It is clear at this time that several changes can be made which will significantly improve the memory performance. Separate tests conducted on the core showed that if the write pulse is shortened by 60 nanoseconds and the read pulse by 20 nanoseconds, the one signal under worst-case pulse sequence dropped to about 35 millivolts and the percent flux switched to about 40 percent. This is approaching the minimum acceptable limit for the sense amplifiers, but still appears acceptable. If this were done, it would reduce the memory cycle by 80 nanoseconds. To go beyond this would require some zero cancellation scheme such as 2 core per bit or a better core.[5] Faster logic circuits are now available, which were not available when the memory was designed. If these were used, the memory decode time could be reduced by 30 nanoseconds. The gate drive circuit now accounts for 60 nanoseconds in the access and 105 nanoseconds in cycle time due to turn-on delay and transition time. A new version of the circuit reduced the 105 nanoseconds by 55. The transition board and twisted pair cable increased the length of the bit-sense line considerably. If these were repackaged, the sense loop could be reduced by 16 inches and the bit line by 9 inches. It is estimated this would reduce sense line recovery by 30 nanoseconds. If all of these improvements were made it would reduce the access time to

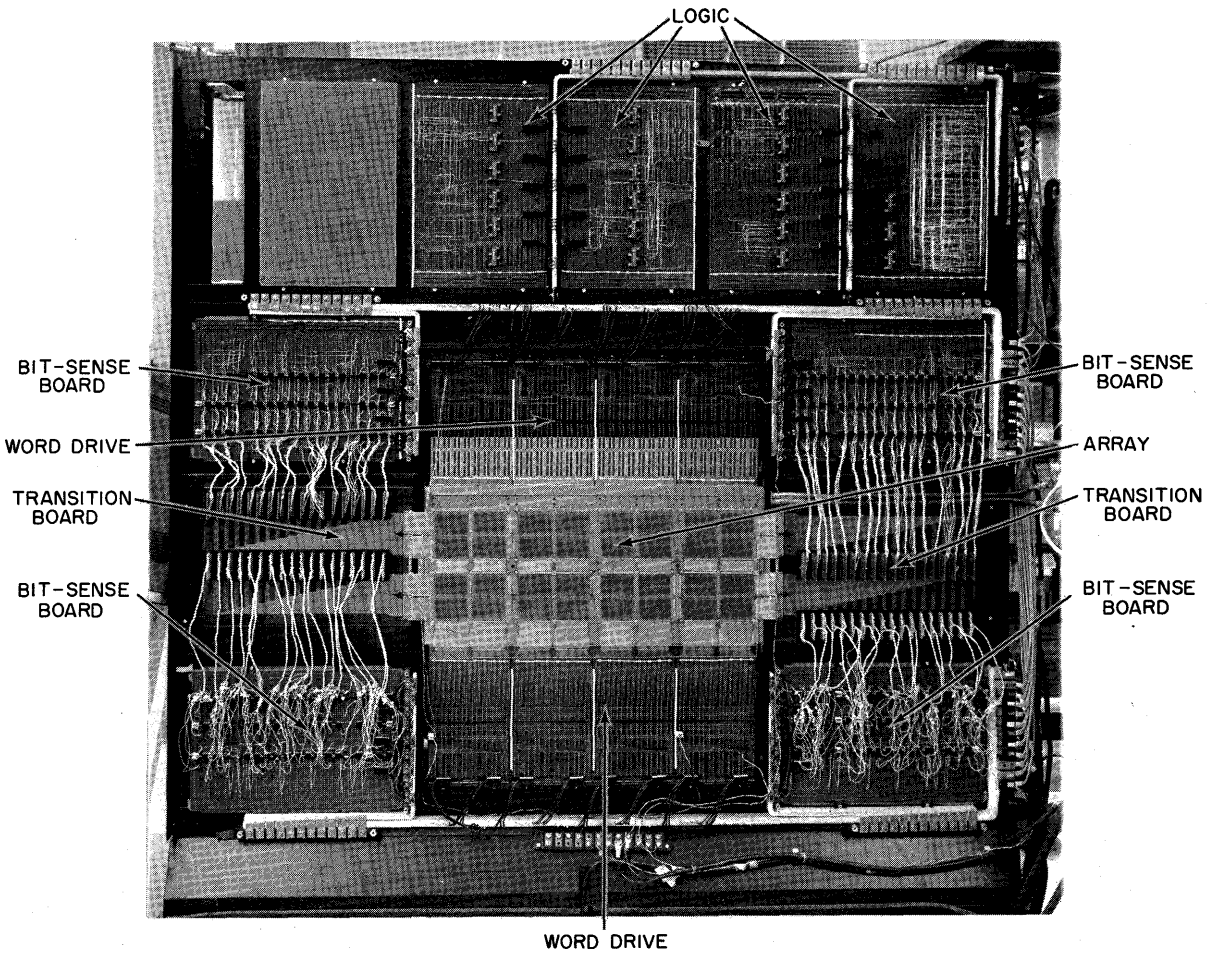



Figure 10. Memory unit — 8K words.

135 nanoseconds and the interval between accesses to 250 nanoseconds.

## ACKNOWLEDGMENTS

In a project of this magnitude and complexity, it is impractical to mention all the persons who contributed to its success. However, particular acknowledgment is given to the several engineers and technicians who worked directly on the project and were responsible for the design, construction, and testing of the model. In addition, grateful acknowledgment is given to the many supporting groups who contributed specialized talents so vital to the success of the program. In this category, the efforts of the circuits and device groups in the Poughkeepsie Memory Development organization and the core plane and array assembly groups of the IBM Kingston Manufacturing organization are acknowledged.

## REFERENCES

1. J. A. Rajchman, "Computer Memories—Possible Future Developments," *RCA Review* (June 1962).

2. C. J. Quartly, "A High Speed Ferrite Storage System," *Electronic Engineering,* vol. 31, pp. 756-758 (Dec. 1959).

3. W. H. Rhodes et al, "A 0.7 Microsecond Ferrite Core Memory," *IBM Journal of Research and Development* (July 1961).

4. V. L. Newhouse, "The Utilization of Domain Wall Viscosity in Data Handling Devices," *Proc. of IRE,* pp. 1484-1492 (Nov. 1957).

5. C. S. Holzinger, "Technique for Determining the Speed Capabilities of 2D Ferrite Core Memories," *Proc. Intermag Conf.,* April 1965, pp. 14.3-1 to 14.3-6.

# MONOLITHIC FERRITE MEMORIES

I. Abeyta, M. M. Kaufman, and P. Lawrence
*Radio Corporation of America*
*Camden, New Jersey*

## INTRODUCTION

Monolithic arrays of ferrite memory elements are being used to produce low-cost, high-speed memory stacks. These elements are made by the simple batch fabrication technique of laminating ferrite sheets with embedded conductors. This process, evolved at the RCA Research Center, Princeton, New Jersey, was selected for material development by the Electronic Components and Devices Division and for system development by DEP Applied Research.

This paper describes the construction, characteristics and system tests for a basic monolithic memory stack. The system operates in the word select mode and employs several types of selection matrices. Storage diodes and conventional diode systems will be described together with the employment of integrated circuits for a large number of system components.

## FABRICATION OF THE MEMORY STACK

### Ferrite Wafer

*Construction.* The ferrite wafer is constructed by sandwiching two groups of conductors between very thin sheets of ferrite to form closed-flux-path storage elements. The wafer is just over 1 inch square and less than 6 mils thick. Each group of

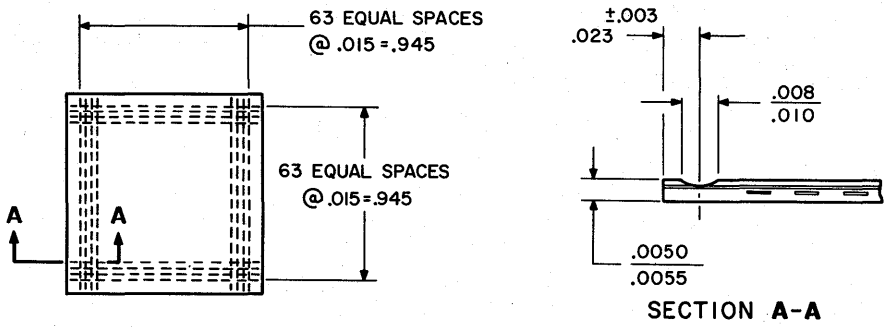



Figure 1. Monolithic wafer.

conductors consists of 64 straight parallel lines in a planar array, with center-to-center separations of 15 mils, as shown in Fig. 1. The conductors of one group are placed at right angles to those of the other. Their vertical separation is less than one mil.

The ferrite sheets are prepared by a technique known as "doctor-blading." In this process, a mixture of ferrite powder (an Fe-Mg-Mn-Zn composition), vinyl plastic and plasticizer dispersed in methyl-ethyl-ketone (MEK) is prepared in a ball mill. This slurry is poured onto a glass substrate and drawn to the appropriate thickness by passing a metal blade, called a "doctor blade," over the mixture. When the MEK evaporates, a sheet remains in which the ferrite particles are suspended and bound. The density of the sheet is related to the extent of dispersion of the ferrite powder in the slurry, and the binder system plays a very important role in stabilizing the dispersions. The doctor-blading technique enables us to form ferrite sheets of approximately 50 percent of the maximum density, with thicknesses ranging from 0.1 mil to 20 mils.

The laminated ferrite plate is made using three ferrite sheets. Two of the ferrite sheets have a pattern of 64 lines of metallic powder on one surface. Palladium, the metallic conductor, is squeegeed (in paste form) through a metal mask onto a glass substrate. The metal mask is removed and the ferrite is then doctor-bladed over the line patterns. The conductor lines obtained are 6 to 7 mils wide, 1 mil thick and 1.2 inches long.

On the green doctor-bladed sheet, the resistance of this line is 225 ohms ± 10 percent. This measurement is used as a quality control check on the metallic paste batches.

The three ferrite sheets form a wafer 1.2 inches square in the unfired state. The sheets are laminated at an elevated temperature and a pressure of about 10,000 pounds per square inch to form a monolithic body. The firing of these ferrite bodies is divided into two cycles: (1) a binder burnoff stage and (2) a sinter or densification stage. The binder burnoff cycle is extremely important, since it is here that such mechanical problems as cracking and warpage occur. The heating rate must be very slow and the proper atmosphere must be maintained to facilitate binder removal. After completion of this cycle, the laminates can be brought directly to sintering temperature and fired to yield the desired magnetic properties. The proper atmosphere must be maintained during sintering, not only for magnetic con-

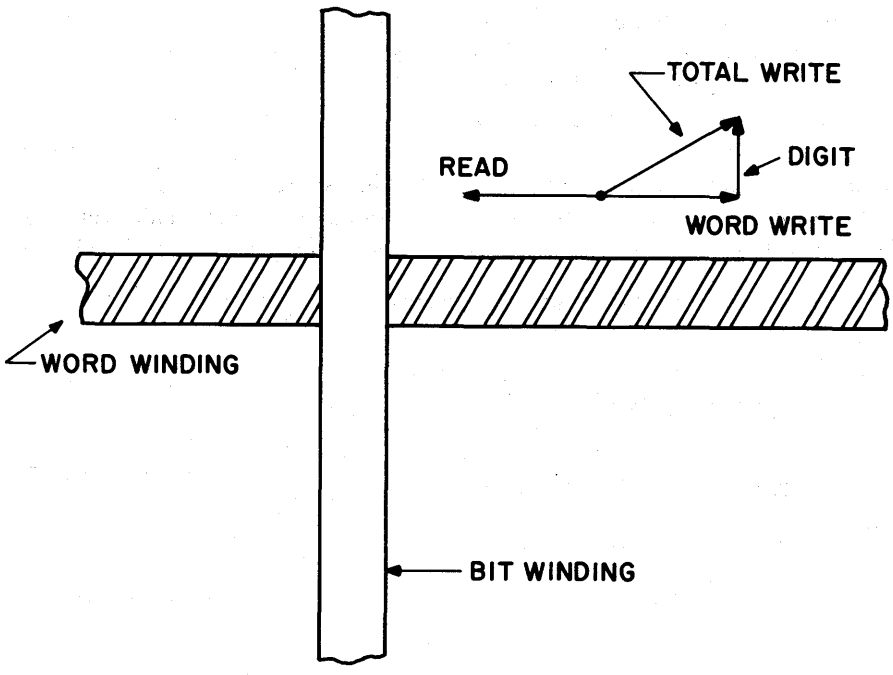



Figure 2. Memory operation.

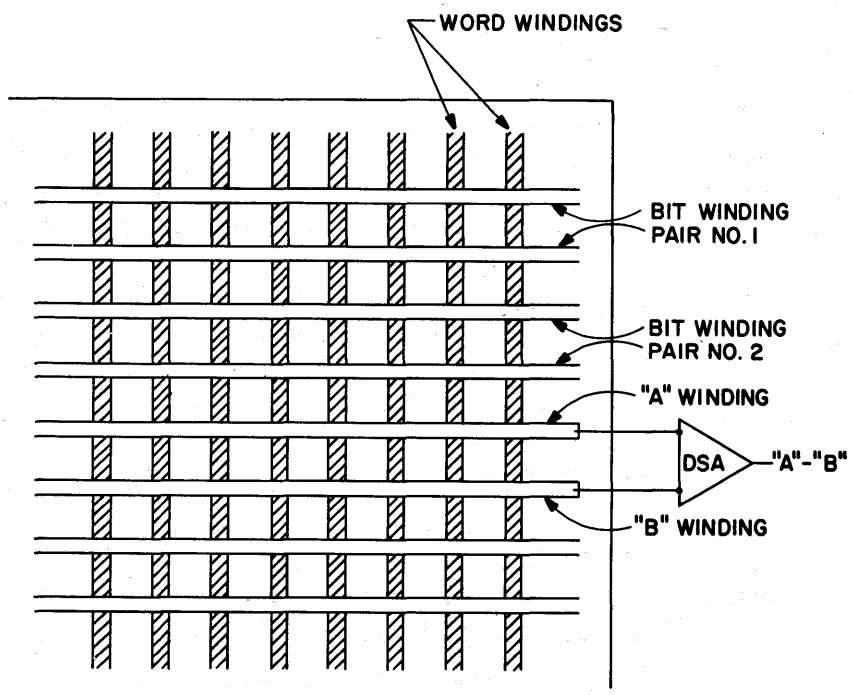



Figure 3. Laminate winding arrangement.

siderations, but also to keep the electrical resistivity of the ferrite high so that the ferrite layer between the palladium conductors can function effectively as an insulator.

After sintering, the ends of the palladium conductors are exposed for electrical connection. This is accomplished by using an airbrasive unit to erode away the ferrite above the conductors. The electrical resistance of the embedded palladium conductors is now 2.5 ohms.

*Operation.* Integrated ferrite wafers have been developed for linear select operation. Each wafer has 64 word windings and 64 bit windings. Each bit is composed of the crossover between the word winding and two adjacent digit windings, so that a wafer has 32 bits per word. By connecting bit lines of several wafers in series, a memory with some multiple of 64 words is formed. Similarly word lengths in multiples of 32 bits are made by adding wafers on the other axis.

Figure 2 illustrates the nature of magnetic flux switching at one crossover point with the application of word and digit pulses. As long as pulses are applied to the word winding only, there is no flux change around the bit windings, so no signal is coupled magnetically from the word to the bit windings. The vector diagram in Fig. 2 shows the addition of word write and digit driving fields. The respective components are drawn parallel to the driving currents, thus normal to the planes of their respective driving fields. With coincidence of word and digit currents, flux is switched to the plane to which the vector sum is normal. Upon application of a word read pulse, which is opposite in polarity to the word write pulse, all of the flux is switched to the planes normal to the word winding, with a direction consistent with the vector marked READ. Elimination of the component of flux which had linked the bit winding causes a magnetically coupled signal to appear on the bit winding. Analysis will show that the polarity of this signal depends only upon the polarity of the digit current. For the situation shown, the upper end of the bit winding has a positive voltage with respect to the lower end at read time.

Figure 3 shows the 2-crossover-per-bit storage technique. Each pair of bit windings has its own set of digit drivers and a sense amplifier. For those bits of the addressed word which are to store ones, a positive digit current is applied to winding "A" and a negative current to winding "B." At the

addressed word, flux switches in a fashion conforming to the explanation of Fig. 2. For those bits which are to store zeros, a positive digit current is applied to winding "B" and a negative current to winding "A." When that same word is next addressed with a word read pulse (opposite in polarity to a word write pulse), the explanation for Fig. 2 shows that the "A" lines of bits storing ones and the "B" lines of bits storing zeros have negative output voltages; conversely, the "B" lines of bits storing ones and "A" lines of bits storing zeros have positive output voltages. Hence, if the difference sense amplifiers yield A-B, one output signals from the sense amplifiers are negative and zero outputs are positive. Note that the total signal output of the sense amplifier is proportional to the sum of the absolute values of signals magnetically coupled at the contributing crossover points.

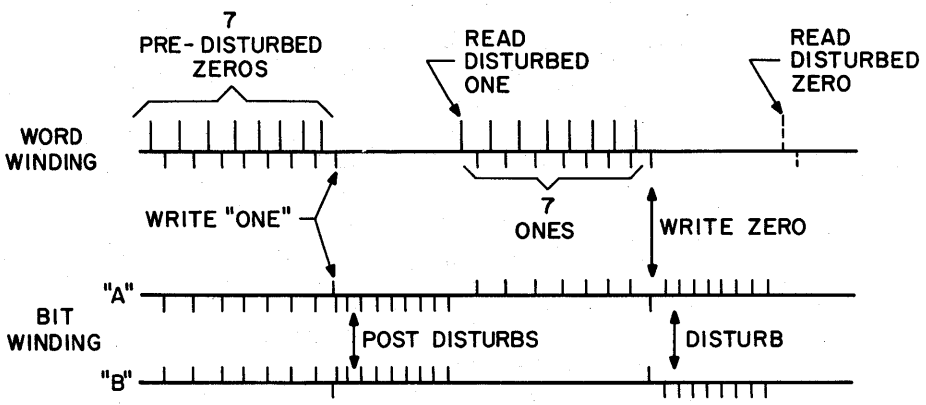



Figure 4. Disturb test pattern.

*Testing.* Extensive testing of ferrite wafers has been conducted under worst-case disturb conditions. The test pulse pattern applied to the ferrite wafers is shown in Fig. 4. Seven pre-disturb pulses and 8 post-disturb pulses are applied to the wafer. Tests made with a greater number of disturb pulses indicate that 7 pre-disturbs and 8 post-disturbs produce approximately the maximum disturb condition. From this test, a set of curves were plotted (Figs. 5 and 6) showing differential sensed

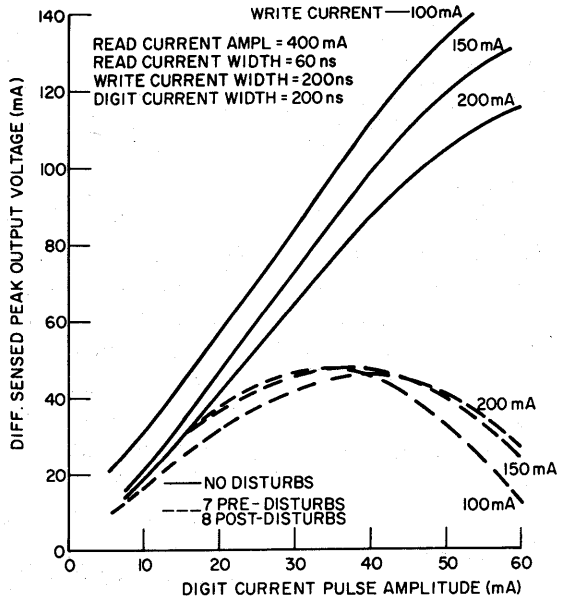



Figure 5. Signal output vs digit current (read current = 400 milliamps).

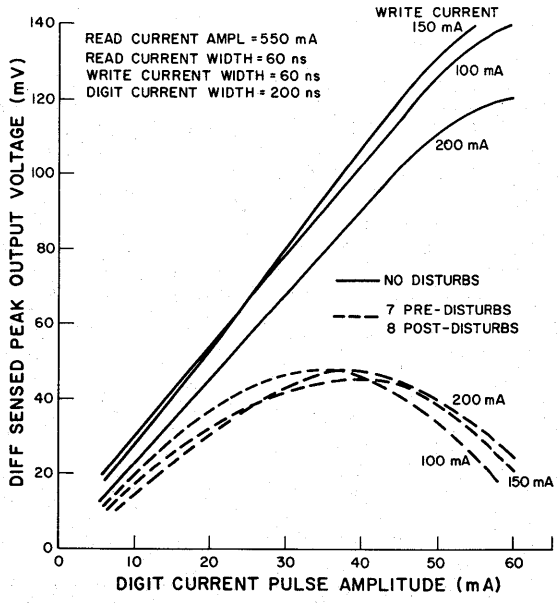



Figure 6. Signal output vs digit current (read current =550 milliamps).

creasing the magnitudes of the 60-nanosecond peak output voltage as a function of digit current. A desirable range of operating current values have been chosen on the basis of these curves. Disturbed signal output versus digit current, with write current magnitude as a parameter and read current magnitude fixed at 400 milliamps, is shown in Fig. 5. The same curves are plotted in Fig. 6, except the "read" current magnitude is fixed at 550 milliamps. The typical disturbed output of the ferrite wafer is 45 microvolts. Figures 5 and 6 indicate that inwide read and write current pulses above 400 milliamps for read and 100 for write will not increase the disturbed output signals.

The digit current indicated in Figs. 5 and 6 has an optimum amplitude of between 30 and 40 milliamps. It is apparent that the optimum word write and read pulse amplitudes are a function of the pulse widths chosen. A number of operating pulse values are shown in Table 1. The bit back-voltage is also a function of the pulse values and is listed in the table.

Table 1. Operating Pulse Values.

| Read Current | | | Write Current | | Digits | | Typical Outputs | BBV* |
|---|---|---|---|---|---|---|---|---|
| I *ma* | $T_d$(50%) *nsec* | $T_rT_f$ *nsec* | I *ma* | $T_d$(50%) *nsec* | I *ma* | $T_d$(50%) *nsec* | *mv* | *mv* |
| 400 | 110 | 45 | 100 | 120 | 30 | 200 | 45 | 250 |
| 400 | 60 | 30 | 150 | 30 | 30 | 100 | 25 | 320 |

*Bit back-voltage, or the word read voltage divided by the number of bits in the word.

As an indication of the spread of signal values on a ferrite wafer, a map of disturbed and undisturbed signal values across the wafer is given in Fig. 7. All locations of the wafer have been checked and the map in Fig. 7 indicates worst-case extremes.

Extensive testing of the transmission line properties of the wafers using nanosecond pulse techniques has been conducted also. A summary of the

| Word | DISTURBED/UNDISTURBED SIGNAL OUT AT DIGIT PAIRS (mv) | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pair 1 | Pair 2 | Pair 3 | Pair 4 | Pair 5 | Pair 6 | Pair 7 | Pair 8 | Pair 9 | Pair 10 | Pair 11 | Pair 12 | Pair 13 | Pair 14 | Pair 15 |
| 6 | 50/68 | 40/107 | 45/75 | 40/72 | 37/67 | 36/67 | 37/67 | 33/63 | 33/67 | 42/65 | 28/58 | 25/43 | 28/60 | 25/58 | 32/55 |
| 10 | 60/80 | 43/107 | 50/75 | 43/67 | 40/70 | 38/65 | 33/63 | 37/65 | 32/63 | 47/70 | 25/58 | 23/60 | 27/63 | 25/62 | 33/60 |
| 14 | 57/67 | 53/100 | 53/67 | 45/57 | 40/60 | 37/52 | 35/54 | 37/58 | 33/57 | 48/60 | 27/50 | 27/54 | 27/57 | 32/57 | 37/58 |
| 18 | 53/75 | 48/50 | 52/64 | 43/60 | 37/43 | 33/54 | 23/55 | 35/57 | 28/57 | 43/62 | 20/46 | 25/50 | 28/54 | 27/55 | 33/60 |
| 22 | 55/70 | 50/92 | 50/64 | 42/58 | 37/53 | 32/50 | 30/63 | 33/57 | 25/40 | 42/60 | 20/50 | 23/45 | 27/50 | 30/53 | 33/57 |
| 26 | 58/73 | 50/57 | 57/63 | 43/60 | 35/53 | 30/48 | 30/54 | 35/57 | 25/57 | 38/60 | 23/48 | 23/47 | 23/50 | 30/50 | 37/42 |
| 31 | 55/67 | 45/90 | 47/58 | 30/60 | 33/57 | 27/50 | 30/54 | 33/57 | 27/57 | 43/63 | 23/65 | 30/47 | 27/50 | 25/47 | 33/50 |
| 35 | 53/70 | 53/90 | 43/60 | 33/53 | 25/50 | 25/50 | 22/47 | 17/50 | 23/53 | 33/57 | 23/48 | 23/47 | 27/50 | 23/47 | 37/57 |
| 39 | 58/68 | 50/92 | 47/65 | 38/53 | 32/50 | 33/50 | 27/50 | 28/55 | 23/50 | 37/57 | 23/48 | 25/45 | 28/52 | 30/53 | 30/52 |
| 48 | 35/80 | 50/92 | 47/64 | 33/55 | 24/50 | 22/47 | 22/50 | 33/57 | 28/55 | 40/57 | 22/40 | 23/47 | 23/37 | 22/38 | 17/37 |
| 52 | 42/75 | 50/92 | 45/65 | 35/58 | 25/50 | 22/47 | 22/47 | 27/53 | 27/53 | 47/57 | 25/50 | 20/47 | 28/50 | 25/42 | 24/45 |
| 55 | 58/60 | 40/83 | 37/55 | 32/52 | 27/55 | 22/50 | 25/50 | 30/53 | 27/57 | 40/57 | 28/47 | 23/50 | 30/50 | 33/53 | 37/57 |
| 58 | 58/75 | 40/80 | 40/60 | 42/67 | 30/53 | 20/48 | 27/52 | 32/57 | 32/60 | 48/63 | 30/50 | 28/53 | 30/50 | 33/38 | 27/62 |
| 59 | 57/73 | 40/77 | 30/47 | 42/50 | 33/57 | 20/47 | 30/53 | 33/55 | 37/63 | 50/70 | 38/60 | 40/63 | 37/57 | 38/57 | 40/58 |

Note: Read current = 400 ma, 70 nsec; write current = 120 ma, 70 nsec; digits = 30 ma, 80 nsec.

Figure 7. Map of signals for a typical wafer. (This map shows typical and extreme signals.)

data obtained is shown in Table 2. The characteristic impedance of the transmission line is a function of frequency and coupling to the ground plane; however, the best terminating impedance is given. The attenuation shown in the table for pulses wider than the rise time is essential d-c attenuation, and is associated with the 2.5-ohm line resistance through each wafer.

Table 2. Digit Transmission Line Properties.

| Characteristic Impedance *Best termination* | Pulse Delay *Midpoint to midpoint for 1,024 words* | Pulse Rise Time *256 words* | Attenuation *For pulses wider than rise time, 512 words* |
|---|---|---|---|
| 150 ohms | 35 nsec | 15 nsec | 1.1 dB |

## Stack Construction

Monolithic ferrite stacks are assemblies of modular building blocks. The basic modules are assemblies of two ferrite wafers with diodes and bussing for word selection. There are two kinds of selection diodes available: conventional high-speed, and storage types. The module using conventional diodes requires two diodes per word, while the module using storage diodes requires one diode per word. The fabrication of a 1024-word 64-bit stack using 16 modules with conventional diodes will be described. A 256-word 64-bit stack was also assembled using 4 modules with storage diodes. Operation and testing of both stacks will also be described.

Figure 8 is an illustration of an integrated ferrite module with conventional diodes and Fig. 9 is a schematic of this module. For convenience, a 16-word array is shown. The modules actually have 64. One end of each word winding is connected to the anode of one diode and the cathode of another. The cathodes of the diodes whose anodes are connected to word windings are common in eight groups of eight diodes each. The anodes of the diodes whose

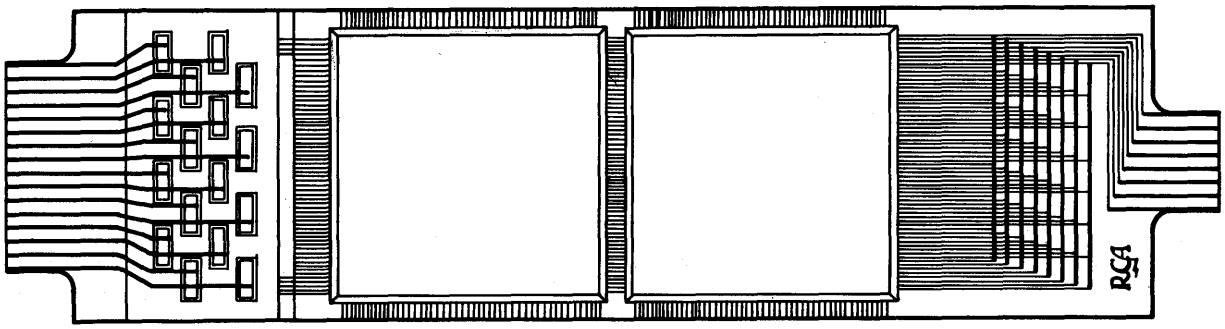



Figure 8. Construction of the basic integrated ferrite module.

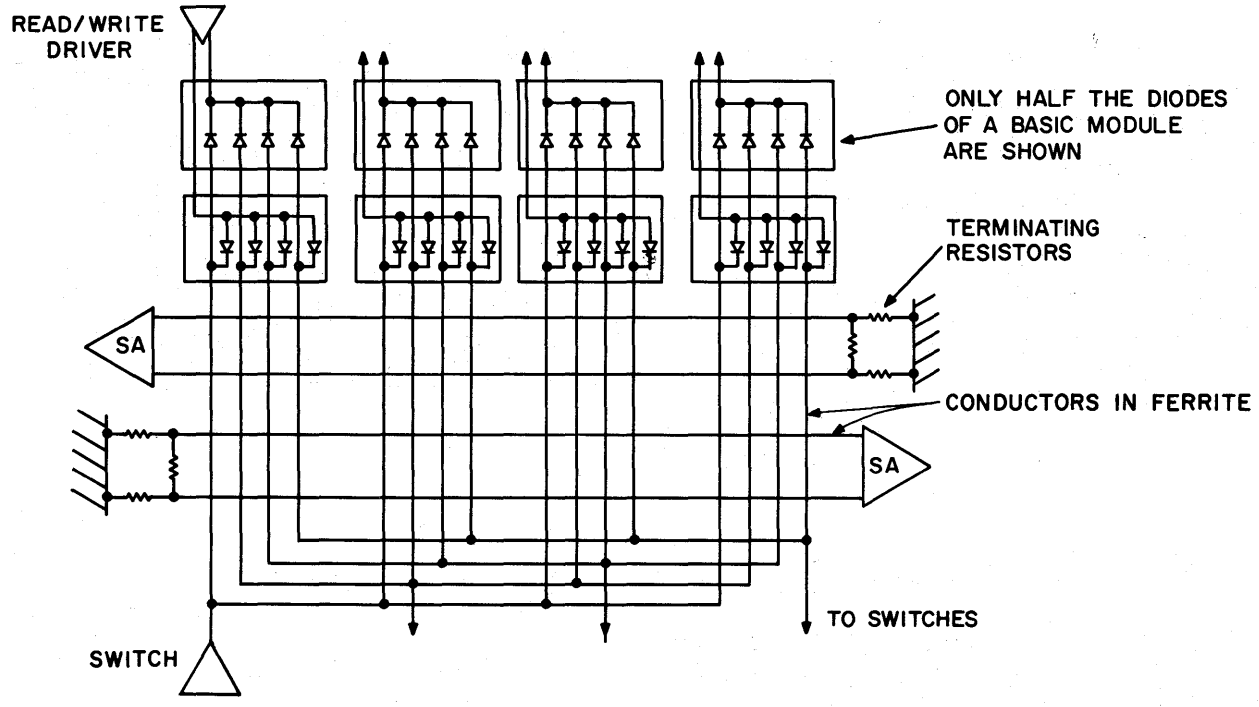



Figure 9. Schematic of the basic integrated ferrite module.

cathodes are connected to word windings are grouped identically.

The diode chips have 2 rows of 4 diodes, each on 30-mil centers. Notice that the number of connections required on the diode end of the words has been reduced from 64 to 16. The ends of the word windings remote from the diodes are connected directly together in eight groups of eight words each.

Each group has one word from each of the diode groups. The number of connections to this end of the module is reduced from 64 to 8.

Expansion of the diode selection matrix to arrays of more than 64 words merely requires expanding the sizes and numbers of the groups by connecting corresponding points of different modules together. For example, a 1024-word matrix would have 32 groups of 32 words each, as viewed from either end.

Line connections on the modules are "fingers" of etched combs leading to the winding ends on the four sides of a wafer. This assembly is soldered to a printed circuit pattern on the module board. After connection, the solid edge of the comb is sheared off. Tabs are left at the edges of the module for connection to the bit windings of another module.

The diode array is assembled as a separate unit and cemented onto the module tabs of previously mounted combs. Connection to the switch buses (see lower end of Fig. 8) is accomplished with combs appropriately etched from one-ounce beryllium copper. They are raised above the module surfaces as they cross lines which they must not contact. The wafer interconnection combs are etched from one-ounce copper sheets, subsequently plated with an electroless tin coating to facilitate soldering. The module substrate is constructed from 1/16-inch G-10 laminated glass epoxy board. The diode assembly comprises a baseboard with a printed circuit pattern and a spacer board of G-10 material with the common connections etched. Word drive connections are made with printed circuit plugs having contacts on 50-mil centers.

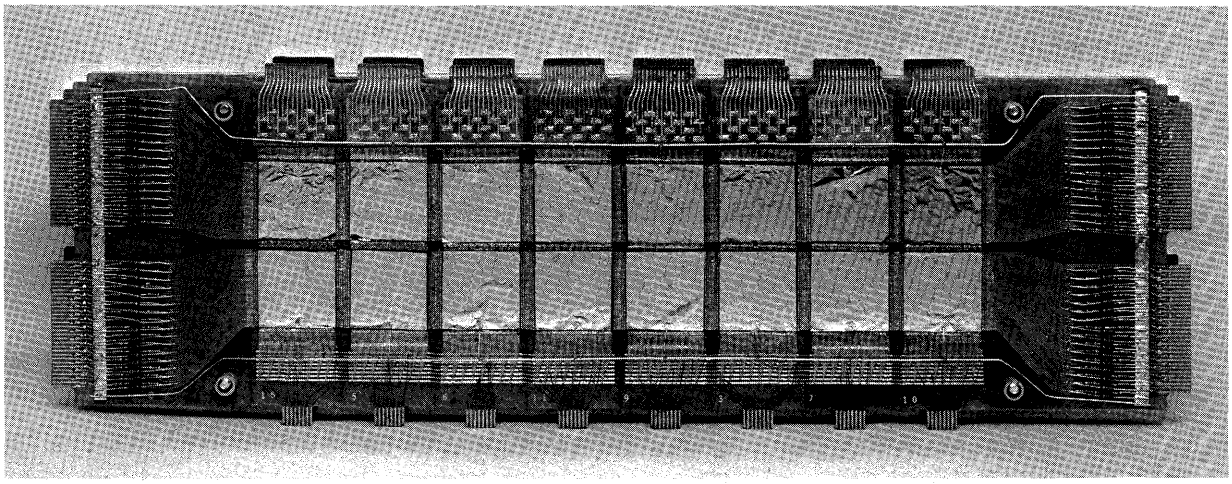


Figure 10. Typical arrangement of monolithic memory stack.

The conventional diode stack consisting of 1024 words of 64 bits is formed from 2 planes, each containing 512 words (Fig. 10). Overall dimensions for the unit, including the sense digit connector boards with terminating resistors, are 14 x 4.5 inches with 0.5-inch spacing between the planes. The sense digit connections are fanned out from the 15-mil centers to suitable plug connections on 50-mil centers. Module boards are cemented onto a backboard and interconnections between the modules are made by soldering. Bussing between planes is accomplished by soldering #30 wire between corresponding bit windings of the two planes.

## MEMORY SYSTEM USING CONVENTIONAL DIODES

### *Selection Matrix and Digit Drive*

This section describes the system design and operational details of the 1024-word, 64-bit memory. The word driving scheme is illustrated by the 4x4 diode matrix of Fig. 11. The physical dimensions and geometry of the matrix diodes have already been given.

These diodes have the following typical characteristics:

$C_0$ (function capacity at zero bias) = 4 picofarads
$V$ (at I=400 ma) = 2 volts
$V_B$ (at 10$\mu$a) = 60 volts
$T_v$ (reverse recovery time) = 15 nanoseconds

Under quiescent conditions, the matrix diodes are back-biased by a positive voltage applied at the write driver, a negative voltage at the read driver, and ground at the read/write switch. The read/write selection sequence is executed in the following manner:

1. A read command pulse turns on the selected read switch, driving the 32 words selected by the switch to a positive voltage, thus removing the back bias from 32 read diodes.
2. Sometime later, the read driver is turned on, forward-biasing one read diode on the selected word, while the drive line moves toward ground potential from a current source to complete the read operation.
3. Similarly, during the write part of the cycle a write command pulse turns on the write switch, driving the selected set of words negative.
4. The write driver is then turned on, driving the write diode on the selected word toward ground from a current source and thus enabling write current to flow.

All the drivers and switches of the word system are compatible at their inputs with the logic levels of the current steering integrated logic gates used throughout the system.

The parallel capacitance of the 32 word lines connected to each switch is about 1600 picofarads. The read channel of the switch supplies 1 ampere of current so that it can charge this capacitance to + 25 volts in about 40 nanoseconds. The write channel of the switch supplies about 500 milliamps to complete the transition from + 25 volts for read to –10 volts for write in about 100 nanoseconds.

As seen in Fig. 11, there are two clamp diodes at the output of each switch, one to + 25 volts and the other to –10 volts. Physically located at the stack, their purpose is to provide a sink for the switch current and thus to maintain a low impedance at the memory stack during the read and write pulses despite the cables needed to interconnect the switch lines in the stack with the circuit boards. This action helps to clean up waveforms and to reduce noise.

The digit drivers are bipolar. They store a one by delivering a positive pulse into one line of a digit

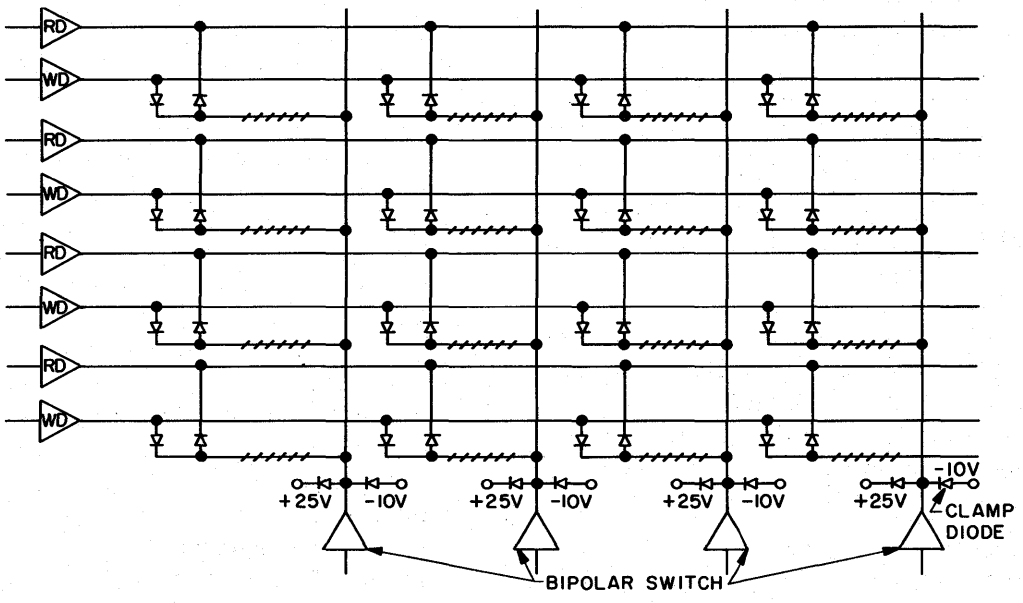



Figure 11. Schematic of the driving and matrix scheme for the two-diode-per-word laminated ferrite memory.

pair and a negative pulse into the other. A zero is stored by reversing these two pulse polarities. The digit driver outputs are voltage sources during the digit pulse, the amplitude of the digit current being controlled by the voltage and the termination resistors. The driver inputs have logic gates which are primed by the information register and activated by the digit timing pulse.

The sense amplifiers are constructed by cascading two current-steering logic gates and A-C coupling at the output of the second gate to the memory register. Strobing is accomplished at the input to the second gate. Power supply levels for the gates are raised to prevent gate saturation during the digit pulse. A positive pulse can appear at the sense output only during the strobe, and only if a one is sensed.

Operation

The test vehicle used to check the performance of the conventional diode memory consists of:

1. The timing and control generator
2. A word system
3. A digit system

The timing and control generator supplies all timing and control pulses. The word system supplies the proper switch voltages and read-write currents at the command of the timing generator. The digit sense system performs the dual function of sensing stored information and writing back into the memory.

The test vehicle has four different types of logic components, providing address scan, disturb patterns and error checks. These are:

1. Integrated current-steering gates of the emitter-coupled current-steered logic (ECCSL) type
2. Integrated current-steering flip-flops
3. One-shots with variable delay, made by adding a few external components to integrated current-steering gates
4. A free running multivibrator with variable frequency, also made from an integrated current-steering gate

These four devices have proven to be very stable and reliable, and have made the test unit highly flexible, coupled with high packing density.

The test system can be set to pre-disturb a word up to 35 times when running at 2 megacycles per second. The pre-disturbs end when the timing generator causes the digit drivers to change write information for one write time. The timing generation then causes up to 70 digit disturbs to be generated. At the end of the digit disturb period, the last written information is read out. Then this entire pattern is complemented. Finally, the address is changed to a new word. The disturb pattern is then repeated at the new word, etc. Information from one bit to the next along a word can be complemented by a mechanical switch in the information register. The timing diagram for the test vehicle is shown in Fig. 12.

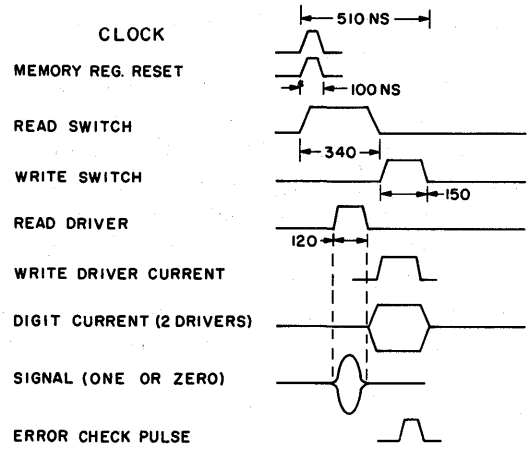



Figure 12. Timing diagram for conventional diode test vehicle.

The read switch is turned on approximately 150 nanoseconds ahead of the read driver to allow the noise transient coupled into the digit lines to decay to a level much less than the signal.

The noise is coupled into the stack through all of the words common to the selected switch. The magnitude of this noise depends on the number of words in the stack. For a switch driving 32 lines, the common-made noise injected into the digit lines is approximately 1 volt. However, as has already been pointed out, sensing is performed differentially; therefore, large common-mode noise will not be a problem provided the digit pair balance in the stack is sufficient. This balance minimizes conversion of common-mode noise to difference mode. The conversion to difference mode for this stack was approximately 1.5 percent, or 15 microvolts. With ideally

terminated digit lines, a read operation could immediately follow the end of the digit pulse. However, it has been found that a waiting period 6 to 9 times the delay on the dig line is required.

In addition to the switch and digit noise, which are not time-coincident with the signal, there are sources of noise which are coincident with the signal. One of these sources is associated with the fact that the impedance between the stack and the matrix switch ground cannot be made negligibly small. The problem is minimized by adding clamps at each switch line. Reference to Fig. 11 shows the location of these diodes on the stack. The second source of time-coincident noise is associated with the removal of back bias on the matrix diodes connected to the selected driver. Other things being equal, this noise can only be eliminated by the inherent common-mode rejection of the stack.

### Test Results

Waveforms of the tests on this 1024-word 64-bit stack operating at a 500-nanosecond cycle time are shown in Figs. 13 and 14.

Figure 13 (*top*) shows fully disturbed stack output signals with digit transients. Each of the three traces represents the outputs of a bit from 64 words superimposed. The first transient at the left is caused by read switch turn on which lasts 150 nanoseconds. Following this are signal outputs. The positive signals are ones and the negative signals are zeros. These last for about 70 nanoseconds. The remaining time in the cycle is occupied by the digit transient.

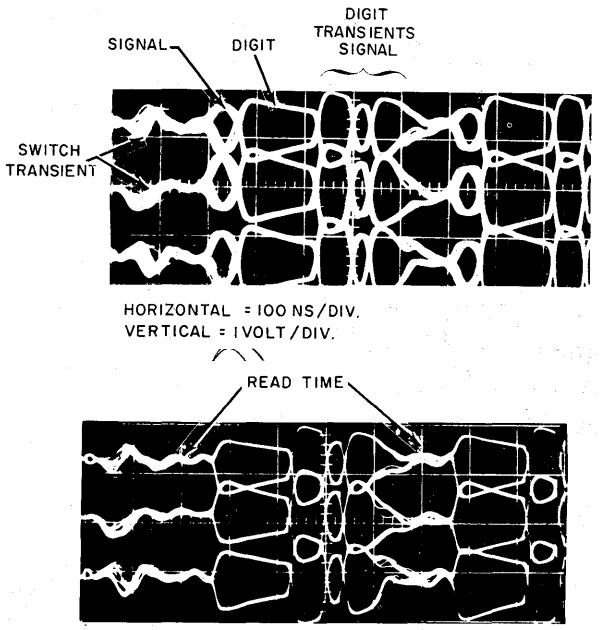



Figure 13. Comparison of the signal and total noise present at read time. *Top:* 64 Signals superimposed at each of 3 digit locations. *Bottom:* Same as above, except that write drivers are disconnected to show the noise present at read time.

Figure 13 (*bottom*) shows the stack output with the drive conditions the same as those above except for removal of word write current. The low noise level at signal time indicates good sense pair balance.

In Fig. 14 (*top*), typical read and write currents are shown at a point between the diode matrix and the drivers. In addition, typical signals are shown in coincidence with the read current. Again, these are disturbed signals. In Fig. 14 (*bottom*), read and write currents as well as switch voltages are shown.

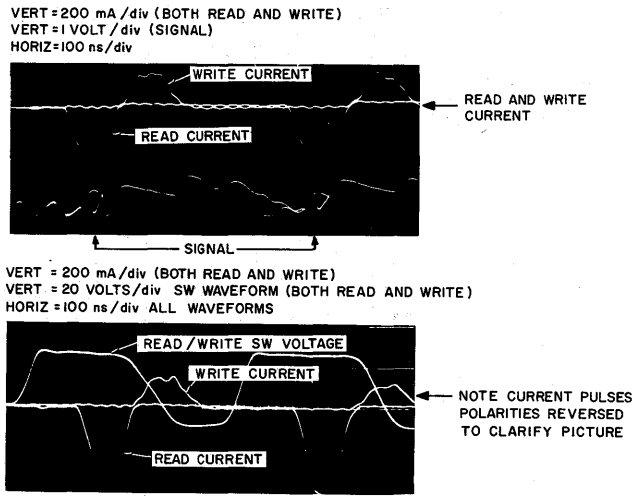



Figure 14. Typical time relationships. *Top:* Time relationships between read/write current and signal. *Bottom:* Time relationships between read/write driver circuits and read/write switch voltages.

## MEMORY SYSTEM USING STORAGE DIODE

### Selection Matrix

The storage-diode selection matrix uses one diode per word. The storage diode, because of its resistivity profile, has long-term minority carrier storage and with one diode allows construction of a selection switch that delivers bipolar currents. Storage diodes also exhibit the desirable property that in the absence of minority carrier storage, a high-resistance (megohms), low-capacitance (a few

picofarads) characteristic is maintained in the reverse direction. This ensures that below the half-select matrix voltage, the properties of a storage diode will be identical to those of a high-speed diode.

A schematic diagram of a 4-word portion of the 256-word matrix that was built and tested is shown in Fig. 15. Half-select operation of the matrix is identical to the half-select operation of a conventional diode matrix, all diodes appearing as approximately 2-picofarad capacitors. Activation of a driver alone or a switch alone does not forward-bias any diode in the matrix.

Activation of a switch followed by a driver drives the selected diode sufficiently beyond its voltage gap to permit the flow of the required read current. The storage diodes in the matrix have minority carrier lifetimes of approximately 170 nanoseconds. Storage-diode lifetime is the parameter that defines the diode's stored-charge recombination rate[2] and, therefore, is directly proportional to the maximum charge the diode can store. The 170-nanosecond lifetime allows the diode to return 20 percent of the charge flowing into it during the read pulse as a write pulse. The write pulse is generated when the read channel of both the switch and the driver are deactivated and the write channel is activated. The write channel of the switch is activated during the fall time of the read clock pulse. The previously selected storage diode is then forced by the voltage that eventually reverse-biases the diode to give up its stored minority carriers as write current. The write current terminates when the charge in the diode is depleted and the fall time of the write current is approximately equal to the snap-off time of the diode.

The diodes used in this matrix are fabricated in integrated strips of eight diodes each. They have the following typical characteristics:

| | | |
|---|---|---|
| $C_0$ | = | 2 picofarads |
| $V_+$ (at $I_f$=400 ma) | = | 1.2 volts |
| $I_R$ (at $V$=100 v) | = | 1 microamps |
| $B_v$ (breakdown $I_R$=10$\mu$a) | = | 150 volts |
| Transition time (snap-off time) | = | 4 nanoseconds |
| Lifetime | = | 170 nanoseconds |

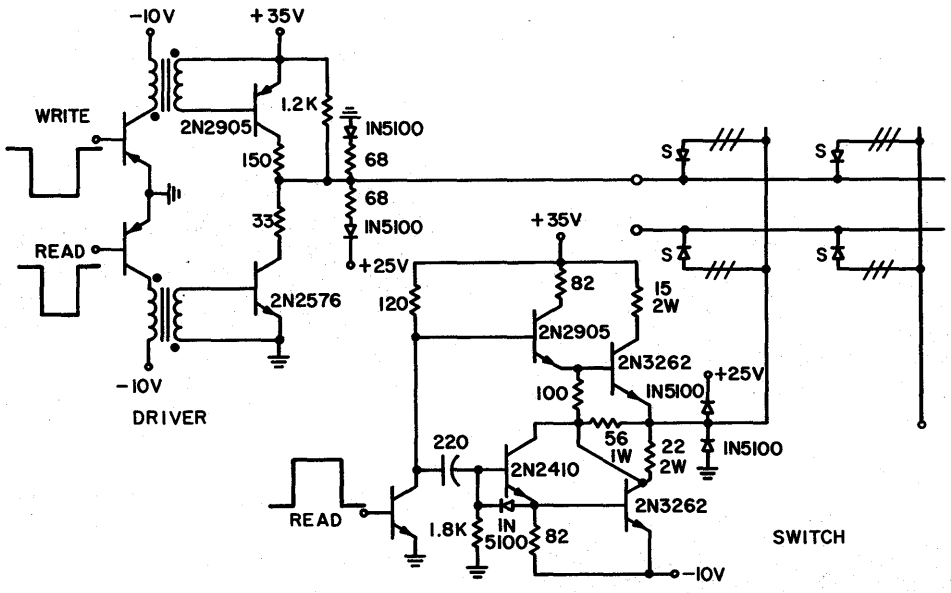



Figure 15. Storage-diode selection matrix.

The read/write current and the word back voltage generated in the matrix for the 170-nanosecond lifetime diode are shown in Fig. 16. The read current peak magnitude is 600 milliamps with a 300-nanosecond base width; write current peak magnitude is 200 milliamps with a 100-nanosecond base width. The word current has a front porch caused by the voltage associated with switching the ferrite. Comparison of the back voltage waveform across the word when all the ferrite is switched with the back voltage under the above condition indicates that 80 percent of the ferrite is

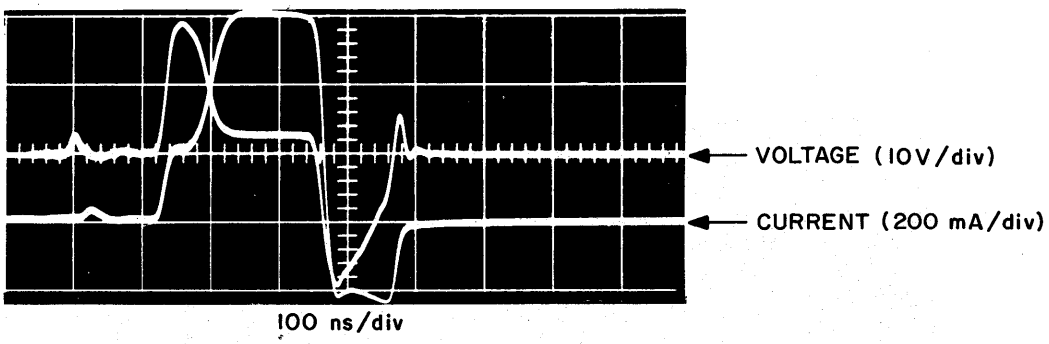



Figure 16. Read/write current and word back-voltage generated for 170-nanosecond lifetime diode.

switching. The percentage of the ferrite that switches is a function of the write current pulse, width as well as amplitude; therefore the duration of the read current front porch is also a function of the write current pulse width. The front porch limits the amount of read charge going into the diode and reduces the percentage change of write current pulse width as a function of storage diode lifetime. A 30 percent change in storage diode lifetime produces only a 20 percent change in write current pulse width.

## Operation

A block diagram of the 256-word 64-bit test vehicle is shown in Fig. 17. The memory was addressed by scanning the 256 words sequentially. There were 16 switches and 16 drivers in the system and all 256 words were addressed. However, there were only 4 sense amplifiers and 4 digit drivers in the system, operating only 4 out of 64 digit locations at one time. The circuits were built on plug-ins and placed in a rack, as indicated in Fig.

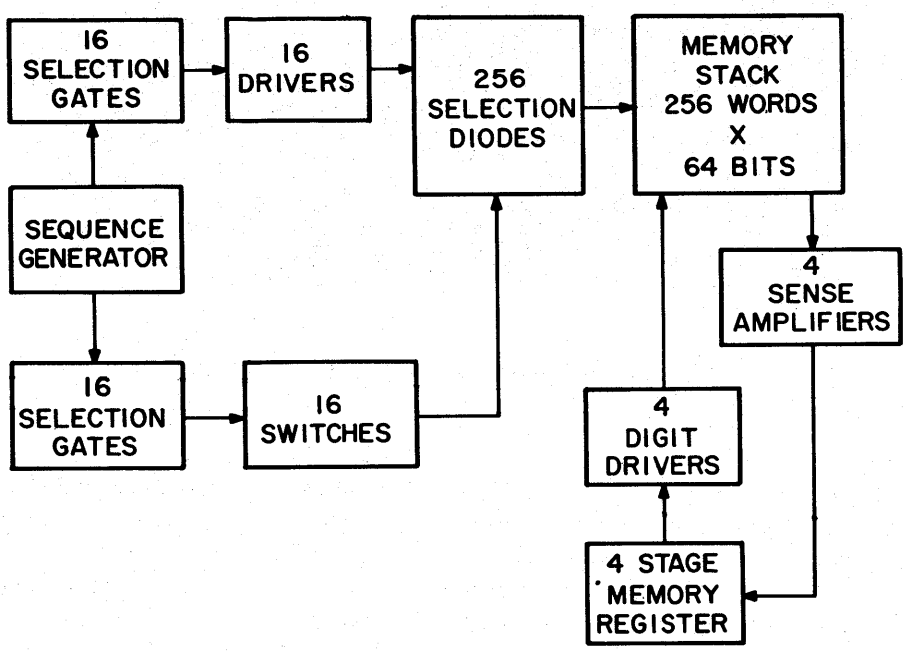



Figure 17. Block diagram of test vehicle for storage-diode matrix.

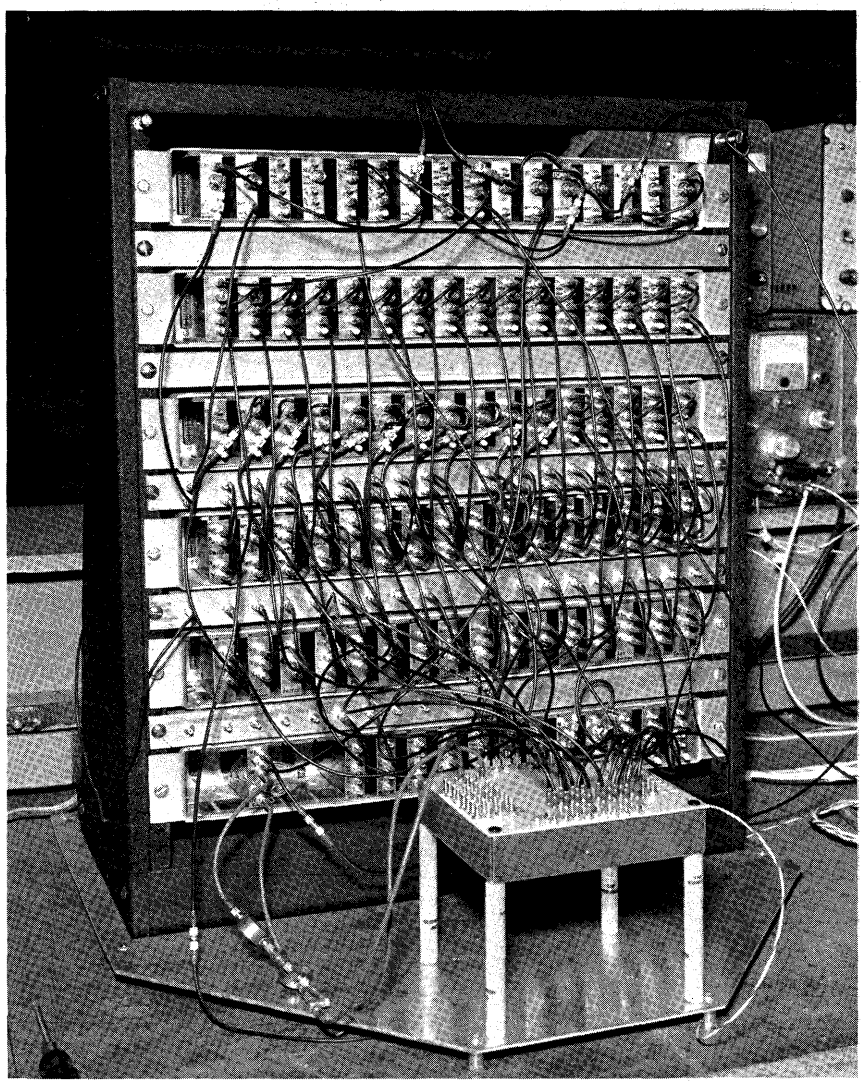


Figure 18. Construction of test vehicle for storage-diode matrix.

18, a photo of the test system.

A timing diagram is shown in Fig. 19. The switch is initiated 100 nanoseconds before the driver to isolate the switch noise coupled into the matrix when the half-selected word lines are driven to 25 volts. The switch rise time is 50 nanoseconds for driving the 256-word stack. When driving a stack of 4096 words (2000-picofarad line capacitance) the rise time is 100 nanoseconds. The word driver rise time is only 25 nanoseconds. The maximum selection line capacitance it drives, considering a 4096-word stack, is 128 picofarads. The digit current is initiated at the end of the read current and has a 30-milliamp peak magnitude. It is 200 nanoseconds wide at the base. The digit lines are driven in a push-pull mode, as shown in Fig. 20. The lines are terminated in their characteristic impedances of approximately 120 ohms.

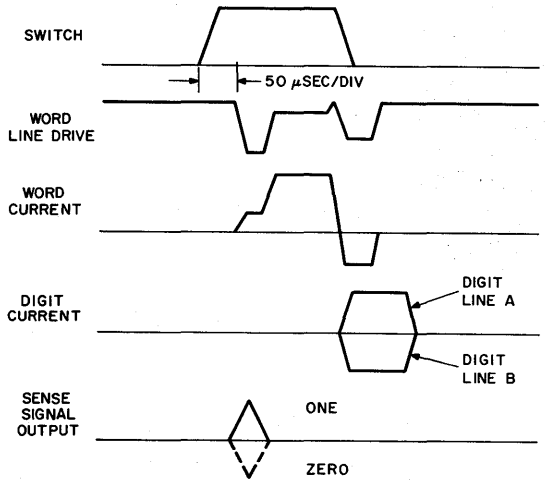



Figure 19. Timing diagram for storage-diode test vehicle.

### Test Results

The common-mode noise on the digit sense lines, generated during the switch rise time, is primarily electrostatic coupling from the half-selected word lines, and is approximately 150 millivolts. Approximately 1.5 percent of the common-mode signal is converted to a difference mode yielding 2 millivolts. The common-mode signal generated during the rise time of the word driver is approximately 20 millivolts, and the signal converted to difference mode cannot be detected.

A group of one and zero outputs out of the stack, as amplified by a linear difference amplifier, is shown in Fig. 21. The lowest disturb output observed is 5 microvolts. The lack of symmetry in the one and zero outputs is due primarily to the noise generated by the strobe circuit in the sense amplifier plug-in.

## CONCLUSIONS

Monolithic ferrite elements have now been sufficiently developed to be competitive with other basic memory elements. They offer the following advantages:

1. These elements can be combined to make basic blocks for high speed memories.
2. The ferrites can be combined with integrated circuits and diodes to make an economical and compact package.
3. Transients in the stack are readily controlled and noise levels are well below signal levels during read time.
4. The use of special storage diodes reduces the number of decoding diodes required, and simplifies the electronics to permit the use of single polarity drivers for all word selection functions.
5. The use of storage diodes and integrated circuits make the linear select monolithic ferrite system cost competitive with coincident current core systems, while offering considerably higher speed capability.

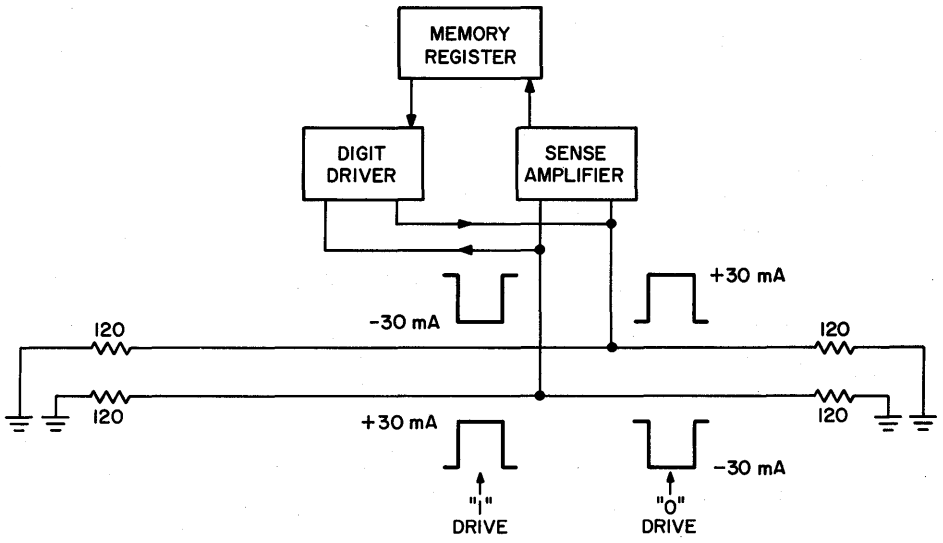



Figure 20. Digit-sense system for storage-diode memory.

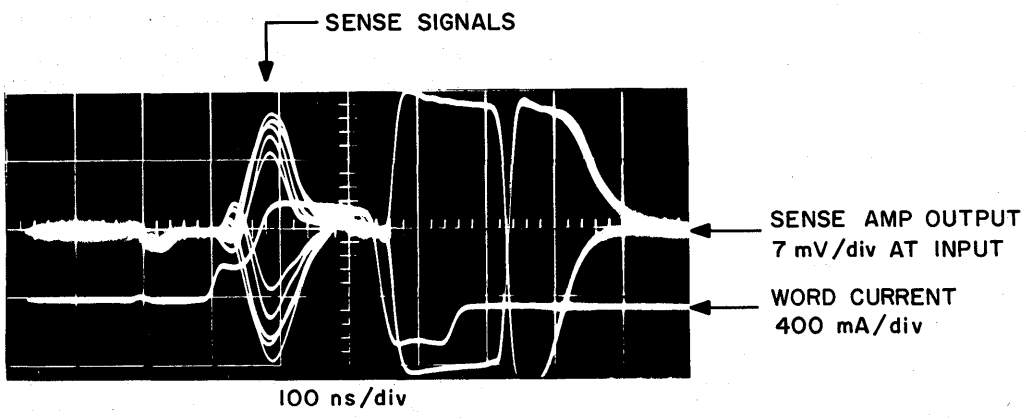



Figure 21. A group of one and zero signals from the storage-diode selected stack.

## REFERENCES

1. R. Shahbender et al, "Laminated Ferrite Memory," *Proc. Fall Joint Computer Conf.,* 1963, p. 86.

2. G. A. Dodson and J. A. Ruff, "Charge Storage Diode for Magnetic Memory Application," *Proc. ISCC,* vol. 7, p. 104, (1964).

3. J. L. Moll, S. Krakanes and R. Shen, "P-N Junction Charge Storage Diodes," *Proc. IRE,* pp. 43-53, (Jan. 1962).

4. H. Amemiya, F. R. Mayhew and R. L. Pryor, "A $10^5$-Bit High-Speed Ferrite Memory System—Design and Operation," *Proc. Fall Joint Computer Conf.,* 1964, pp. 123-144.

# HIGH SPEED FERRITE 2½D MEMORY

Thomas J. Gilligan, Perry B. Persons
*Electronic Memories Incorporated*
*Hawthorne, California*

## INTRODUCTION

Main core storage in digital computers has been getting both larger and faster with each generation. In view of this a design was undertaken which would be inherently faster, and inherently less expensive in the large sizes. Other significant inputs to the design approach chosen was that electronics, that is semiconductors, were becoming less expensive; also, to operate at the higher speeds smaller cores must be used. Since it is progressively more difficult to put additional wires through these smaller cores, system approaches using fewer wires through the core were studied.

The system chosen has basically a two-dimensional magnetics array similar to linear select; however, it has the advantage that it has a level of decoding in the array. Therefore a significant saving in electronics over the liner select may be affected, while maintaining the inexpensive magnetics array. Shown in Fig. 1 is how the relatively inefficient aspect ratio of linear select arrays may be improved by dividing the word dimension by some number and, while multiplying the bit dimension by the same number, get a squarer array.

The chosen system has a coincident current read cycle, and a linear select write cycle, that is, the digit current is additive rather than subtractive. Since

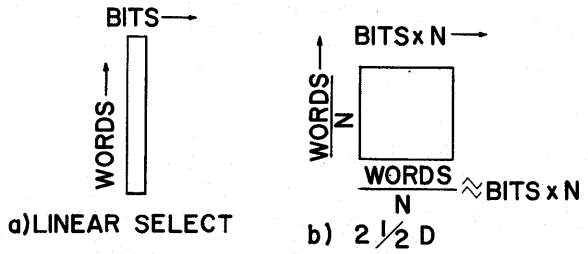



Figure 1. Comparison of linear select and 2½D planar arrays.

this system is not as efficient in decoding as a cubic three-dimensional coincident current system, but far more efficient than a planar two-dimensional linear select system, it became known as a 2½-dimensional system. The inherent power in the 2½D system organization may be appreciated by noting that the system reported here is capable of a 900-nanosecond cycle time using a 30-mil

core. In order to achieve a comparable cycle time, a standard coincident current organization must use a 20-mil core.

## MEMORY ORGANIZATION

In the "pure" memory organizations, coincident current and linear select, the dimensions of the system may be clearly identified as being either address or data dimensions. In the 2½D organization, one dimension serves a dual function as a data and address dimension. This requires that data be inserted on the address lines in one dimension during the write cycle, requiring an independent drive system in one dimension for every bit. We will call this the bit dimension and the other will be the word dimension.

In Fig. 2, a functional diagram of a 2½D, 16K by 3 bit system is shown. It should be noted that a

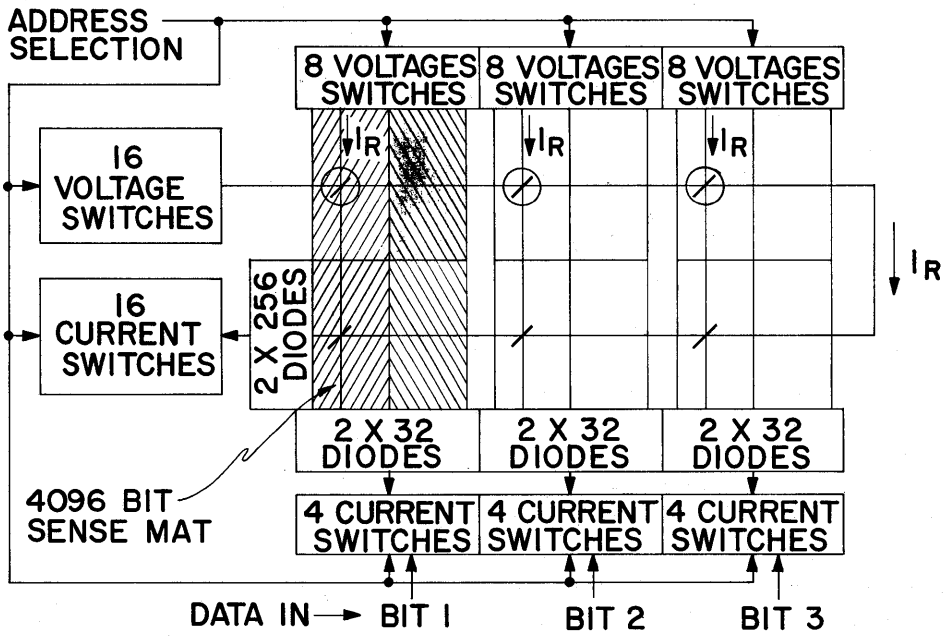



Figure 2. 2½D memory organization — 16,384 words, 3 bits.

complete 32-way diode decode matrix is used for every bit. To select one core of 16,384, one bit line of 32 is selected and one word line of 256 is also selected. In addition the phasing of the word current is chosen to be either positive or negative, and therefore affect coincidence in one of two cores on a common intersection, and anticoincidence in the other. Using the phase of the word current as part of the selection allows the word electronics to be effectively divided by two. It should also be noted in Fig. 2 that there is great redundancy in the bit selection circuitry. That is, the same relative voltage switch, and the same relative current switch, is selected in each bit position. The only difference between bits is that the data will gate current to the matrix or not, depending on whether it is desired to write a one into that bit. Cost will be minimized as the aspect ratio of the system approaches a square. This forces the aspect ratio of a particular bit to be quite "unsquare." The aspect ratio per bit is 512 in the word dimension times 32 in the bit dimension. The aspect ratio of the 4096 sense mat is 256 by 16. A word is read out of the memory by selecting one line in each bit matrix, and then turning on current in the appropriate word line.

A block diagram of the 2½D system is shown in Fig. 3. Included in each block is the number of components in terms of memory capacity $C$, and the number of bits per word $B$. Also included is a weighting factor, $K_n$, which when multiplied by the number of elements in that box, and then summed over all boxes will give the total cost of the system as a function of memory capacity and number of bits per word. From this equation it can be determined that the cost per bit is strongly a decreasing function of the number of words.

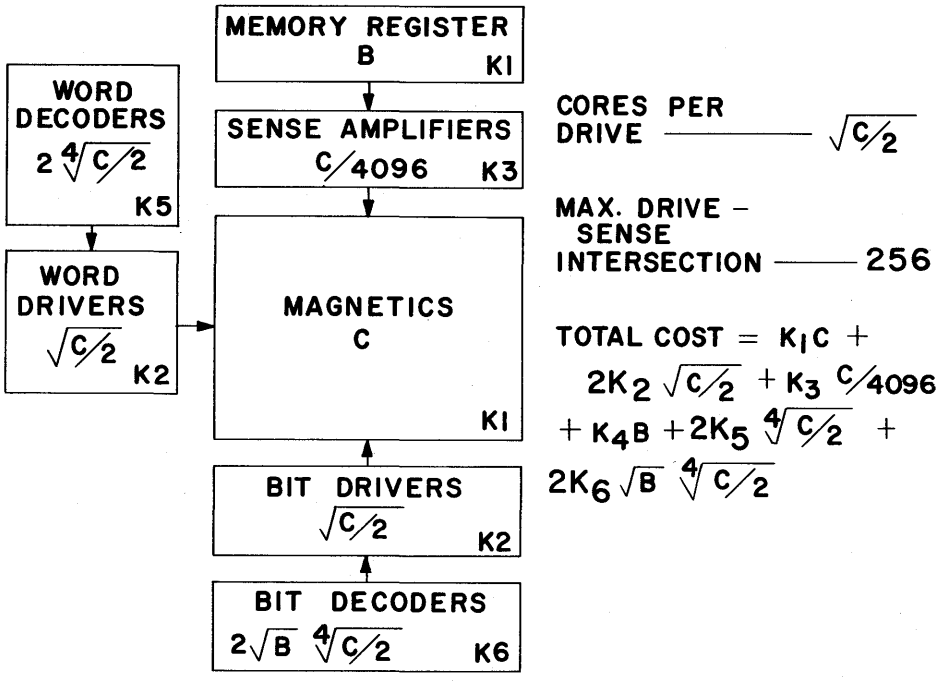



Figure 3. Three-wire 2½D cost breakdown.

## SYSTEM CONSIDERATIONS

As pointed out previously, the 2½D system has many advantages of the 3D and the 2D systems, as well as many advantages in its own right. The most important of these advantages is that there is no inhibit recovery problem, since there is no inhibit current. As a matter of fact in a typical 16K word system, the maximum drive-sense intersection is 256 cores. The typical drive-sense intersection in a coincident current system is 2048 cores. The effect of this is that there is no recovery problem from the previous memory cycle. Another big advantage of the 2½D system is that no drive line is longer than 900 cores, and therefore all system resonances will be significantly higher than the frequency spectrum of a core switching, and thereby simplify the sensing problem. The short drive lines also allow fast rise times on the current pulses, with relatively low drive voltages. This in turn will allow the use of less expensive semiconductors. Since digit current adds to write current to write a one, no time has to be allowed between read and write currents to insure overlap of the write current by the digit current.

The bit drive system accounts for the largest part of the power used in a memory system.

The power required to establish and then hold current in a line may be expressed as: $P = \frac{1}{Tc}(2LI^2 + IRTS)$, where the assumed driver is a saturated switch, $Tc$ is cycle time, $L$ is line inductance, $R$ is line resistance and $Ts$ is the switching time of the core.

The 2½D will have a lower minimum power required since the bit line is shorter by a factor of four or eight than a linear select or a coincident current memory. It must be remembered that current must be driven in the bit line twice during a 2½D cycle and therefore the gain isn't as great as expected.

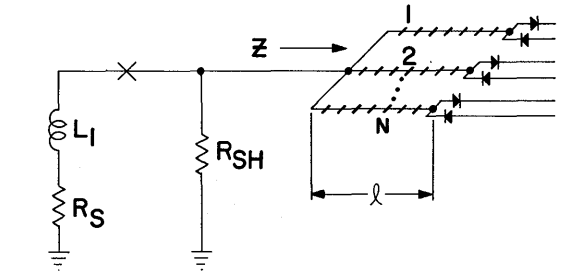
In practice, the power used in a commercial 2½D system will be about the same as in a coincident current system; however, the lower voltage required will allow the drive currents to be derived from logic voltages, i.e., ±12 volts.

The core chosen for this system has the following characteristics: 30 mil O.D., 22 mil I.D., switching time 330 nanoseconds, half select current 380 mils. Since almost the full cycle time in this system is occupied by core switching time, the most desirable core for a faster system of this type would be a high drive (500 milliamps half select current), low flux 22-mil O.D. core. The high drive core may be used effectively since the increased drive requirements will not slow cycle time significantly as the

rise and fall times are but a small percentage of the cycle time.

## TWO DIODE PER LINE DIODE DECODE MATRIX

A two diode per line decoding scheme was chosen for the 2½D system since it is the least expensive of any high speed access system. The important points of this two diode per line system are shown in Fig. 4. This decode system has not been too popular in high-speed systems since when a voltage switch is selected, the selected group of lines tend




$L_1$ — Parasitic Inductance of Voltage Switch Circuit
$R_S$ — Series Resistance of Voltage Switch Circuit
$R_{SH}$ — Shunt Damping and D.C. Restoring Resistor
$Z$ — Impedance Looking Into Magnetics Array
$\ell$ — Electrical Length of Drive Line
$L_2$ — Drive Line Inductance
$C$ — Drive Line Capacitance
$Z_0$ — Drive Line Characteristic Impedance

| | $f$ | $Z$ | Critical Damping |
|---|---|---|---|
| $L_1$ LARGE | $\frac{1}{2\pi\sqrt{L_1 CN}}$ | $\frac{1}{SCN}$ | $R_{SH} = \frac{1}{2}\sqrt{\frac{L_1}{NC}}$ |
| $L_1$ SMALL | $\frac{1}{2\pi\sqrt{L_2 C}}$ | $\frac{Z_0}{N}$ | $R_S = \frac{1}{N} Z_0$ |

Figure 4. General two diode per line decoding; voltage switch resonance.

to resonate, either as a quarter wavelength line with zero impedance at the voltage switch end, and open circuited at the far end, or as a capacitor with the parasitic inductance in series with the voltage switch. In the 2½D system being described, both these effects were minimized by mounting the voltage switches in close proximity with magnetics array. Since the drive lines are so short, the frequency of resonance is high, and therefore the lines do not have to be critically damped. Note that the basic philosophy here is different than previously reported[1] since no attempt was made to minimize the voltage switch to sense line coupling, but instead the resonant frequency and damping were controlled.

## WORD DRIVE SYSTEM

The schematic of the word drive system is shown in Fig. 5, and the photographs of its operation are shown in Fig. 6. It is a simple 256-way diode decode, having the following characteristics: sink capacitance 700 picofarads, drive line inductance 1.3 microhenrys, propogate time 9 nanoseconds.

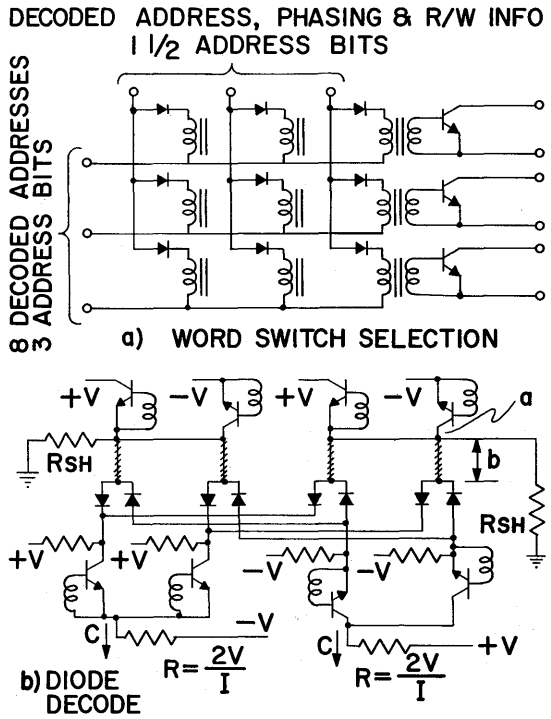



Figure 5. Word drive circuitry.

Floating switches are used and the current is determined by a fixed resistor providing an $L/R$ time constant of 30 nanoseconds. The phasing information is present in the logic input to the decode matrix.

## BIT DRIVE SYSTEM

Each bit in this system has an electrically independent drive system. Because of the redundancy the same relative switch in each matrix is selected at one time. This is accomplished by placing all the transformer primaries in series. The 32-way output is obtained from 8 bipolar voltage switches and 4 bipolar current switches. The read current is always present when the appropriate read switches

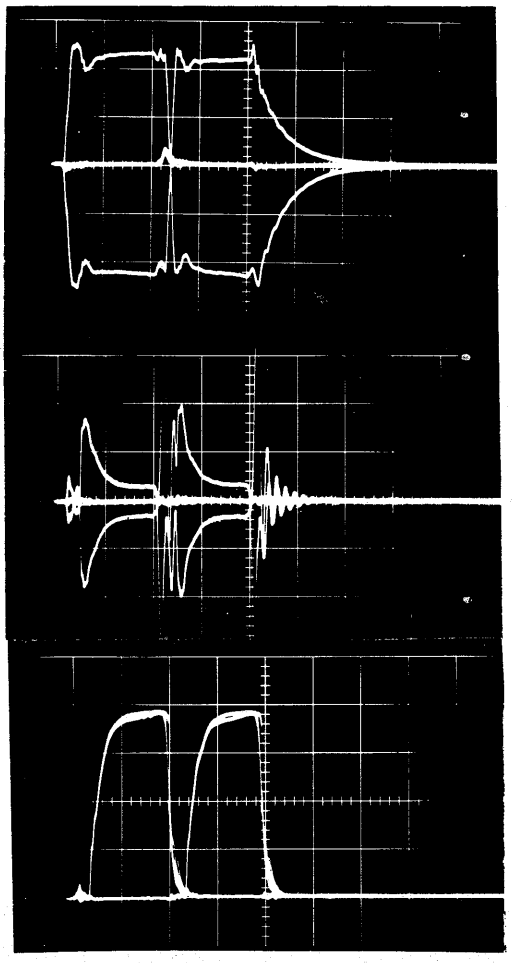


Figure 6. Word drive system waveforms, 200 nsec/cm. *Top:* Voltage switch 5v/cm. *Center:* Voltage across drive line 5v/cm. *Bottom:* Drive current 100 ma/cm.

are selected. The write current, on the other hand, is conditional and will only be present when a one is to be written into memory. The schematic of the bit drive system is shown in Fig. 7. Photographs of the bit drive system operating are shown in Fig. 8. Pertinent characteristics are sink capacitance 125 picofarad, bit line inductance. .70 microhenry, propogate time 5 nanoseconds.

## SENSING SYSTEM

The 16,384 cores per bit are broken up into four 4096 sense lines. The aspect ratio of the sense line was chosen to be 256 × 16 for ease of fabrication. The penalty paid was that twice the delta noise[2, 3] must be handled. Fortunately the system resonances are high, and a wide-band sense amplifier may be large output from the core and the absence of large

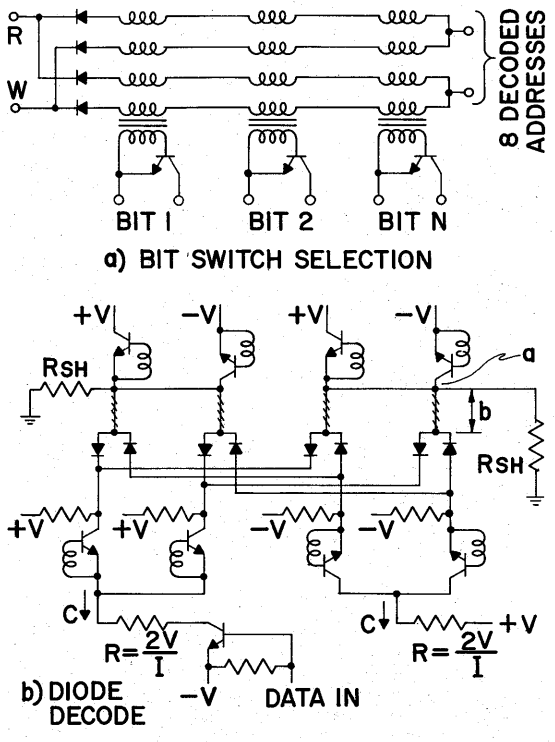



Figure 7. Bit drive circuitry.

used effectively to achieve time discrimination. The inhibit noise (see Fig. 10) allow a less expensive sense amplifier to be designed. The 30-mil cores may be strung on 25-mil centers since a parallel core pattern is used. The sense winding is rectangular,[4] and therefore the cores are on 25-mil centers on the sense winding. This compares to a core spacing of 42 mils on the normal diagonal sense winding with a box pattern. Since the cores are on such close centers on the sense line the impedance of the sense line is somewhat higher. It is approximately 170. The wiring pattern of the sense matrix is shown in Fig. 9. Worst-case outputs from the array are shown in Fig. 10.

## CONSTRUCTION

The magnetics module is constructed by mounting two magnetics arrays back to back and interconnecting them. Each array is 256 × 448 cores. Around the periphery are mounted the drive diodes. This complete assembly is then sandwiched between the bit drive and the bit sink circuitry. The drive and sink circuitry are connected to the magnetics module with flexible wiring. This allows the boards

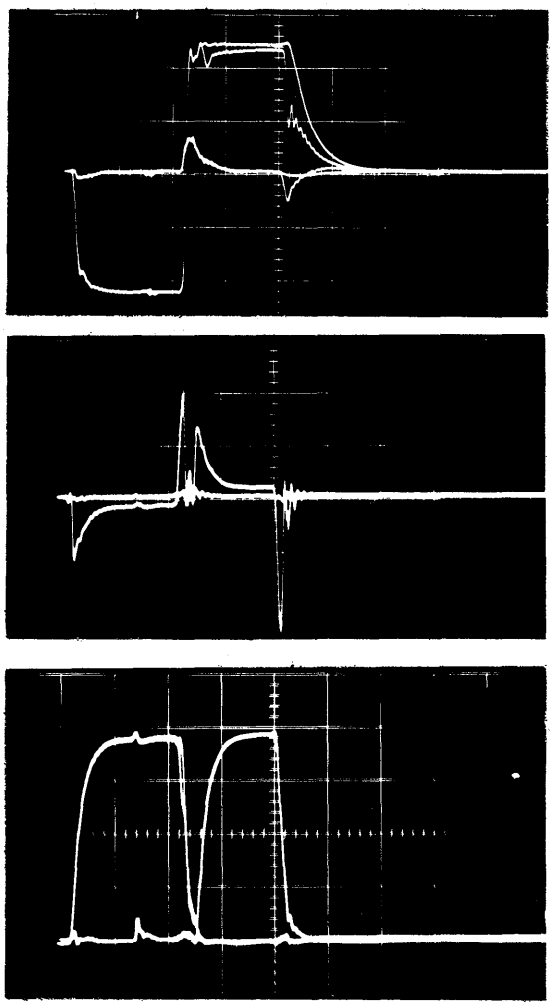


Figure 8. Bit drive system waveforms, 200 nsec/cm. *Top:* Voltage switch 5v/cm. *Center:* Voltage across drive line 5v/cm. *Bottom:* Drive current 100 ma/cm.

to be swung open for repair. The drive boards are hard wired to the array to eliminate a large number of connectors.

The sense amplifiers are of cordwood construction to get a high component density. This is done since it allows the sense amplifiers to be mounted with the drive and sink circuitry near the magnetics array. The package as described to this point is effectively a complete 14-bit memory system, having logic inputs and logic outputs. A system having a larger number of bits may be constructed by joining any number of up to 5 of these smaller 14-bit arrays in parallel. Photographs of this system are shown in Figs. 12, 13, and 14.

## OPERATING MARGINS

In Fig. 15 a 3-dimensional Schmoo diagram is shown similar to those reported by Womack.[5] The

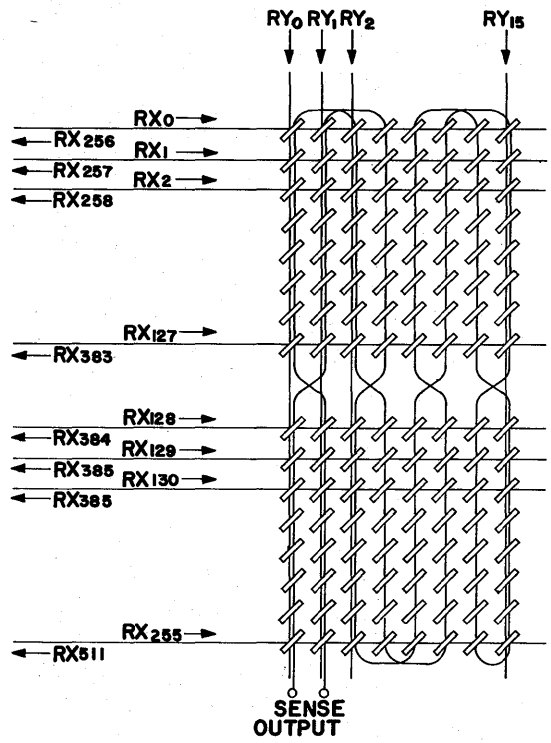



Figure 9. Sense matrix wiring.

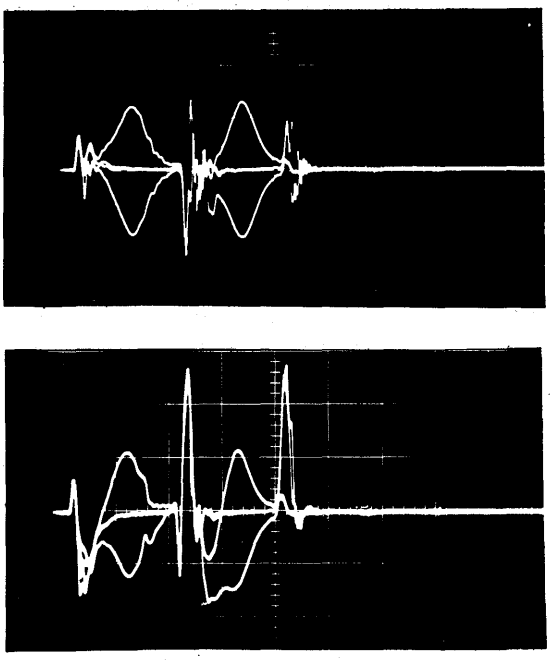


Figure 10. Sensing waveforms, 200 nsec/cm, 40 mv/cm. *Top:* Ones and zeros. *Bottom:* Worst pattern.

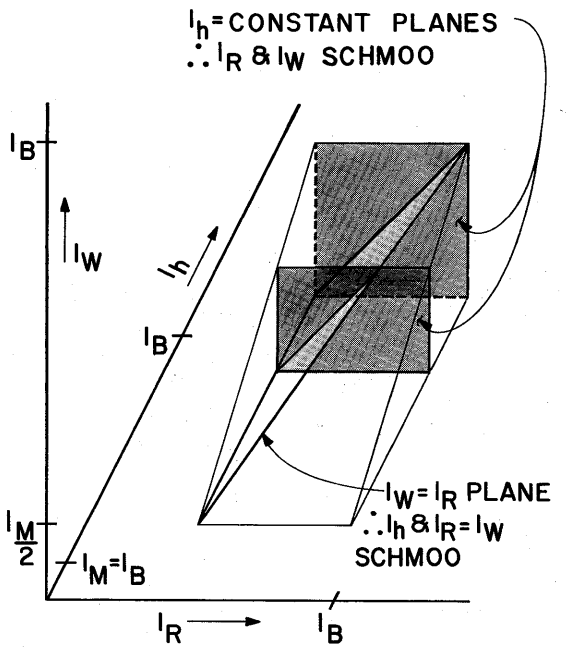



Figure 11. EMI "Nanomemory" — 16,384 words, 56 bits.

operating margins may be compromised somewhat by increasing the write current. This in turn will commonly encountered Schmoo diagrams are shown as planes in this 3-dimensional representation. It must be appreciated that in the 2½D system, since it is a planar array the Schmoo diagram will, of necessity, be a 2-dimensional figure. In the 2½D system being reported, all the currents were derived from plus and minus 12 volts. This makes it difficult, in the 2½-dimensional approach, to vary read and write currents independently. As an alternative, another method is suggested. It is a relatively simple matter to drive the word and bit drivers from separate plus and minus 12 volt power supplies, and to vary these independently to check system margins. That this is a meaningful check on system margins can be understood in that the word and the bit currents are orthogonal, that is, one of these currents may be increased without significantly increasing the knee of the core and therefore check the knee. It is readily apparent from the 3-dimensional Schmoo diagram shown in Fig. 15 that the 2½D I read versus I write Schmoo is the largest plane there. This says that the 2½D system has inherently broader operating margins. These cause a faster cycle time since the core may be switched faster, and we may be operating on a more favorable portion of the loop.

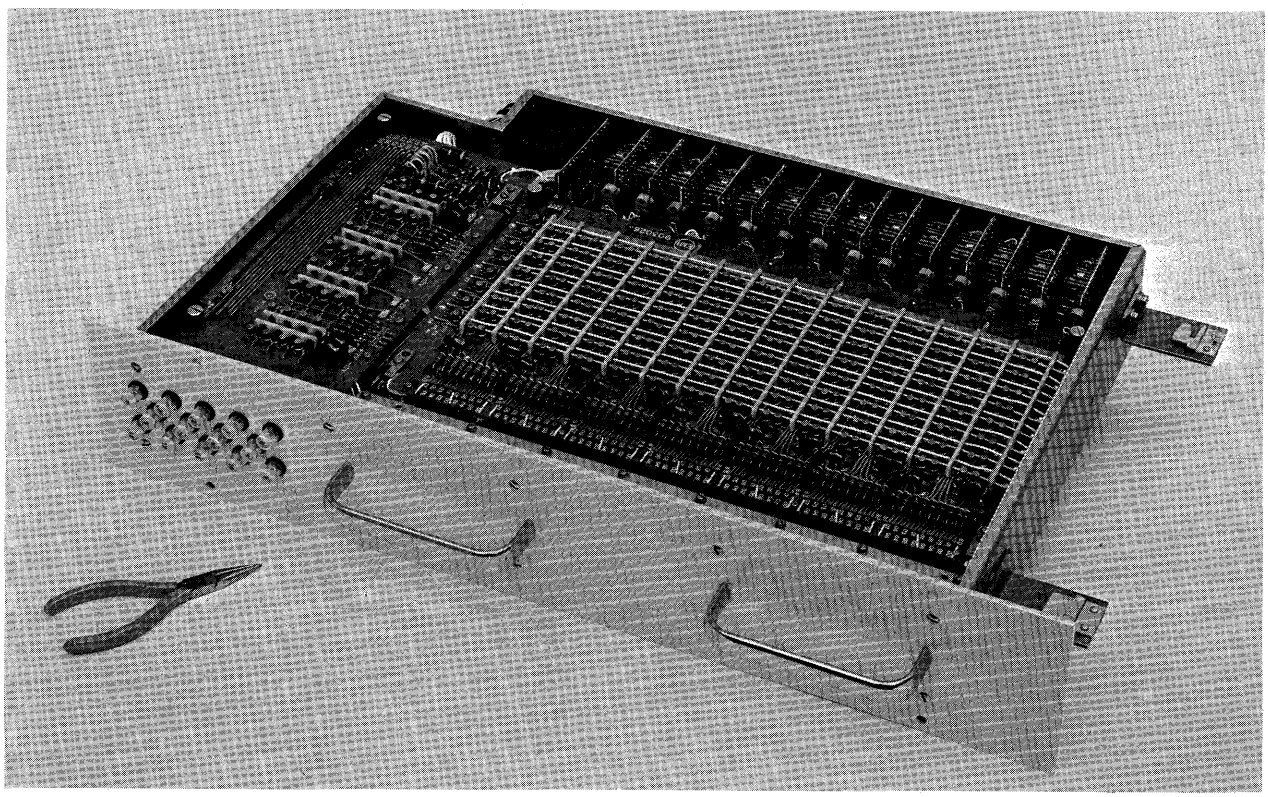


Figure 12. Memory drawer, voltage switch side.

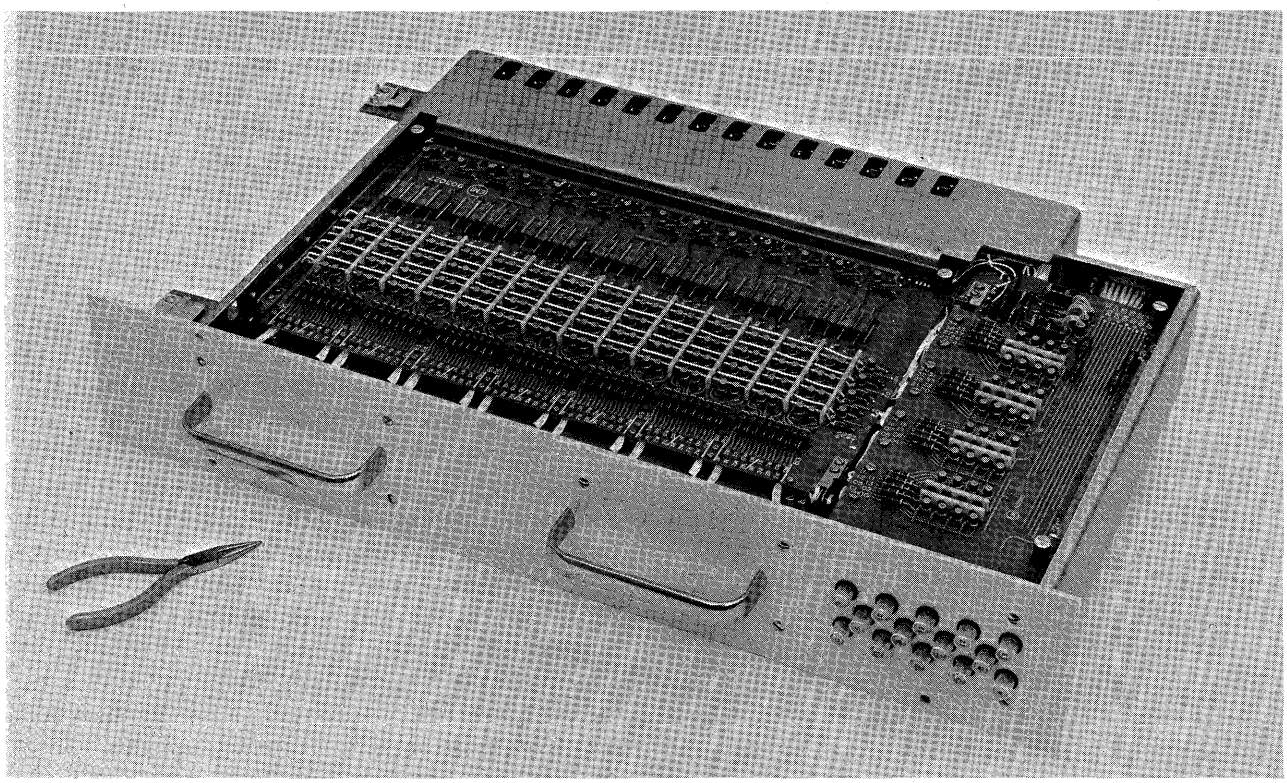


Figure 13. Memory drawer, current switch side.

The operating margins of the system are shown in Fig. 16. We would expect it to be symmetrical in word and bit currents; however, it is seen that we are far more sensitive to the bit currents.

The reason for this is the nonsquare aspect ratio of the sense line, having far more cores on the selected bit line.

Since the Schmoo is elliptical, the operating margins of the system may be measured by varying the word and bit currents together. This would give a

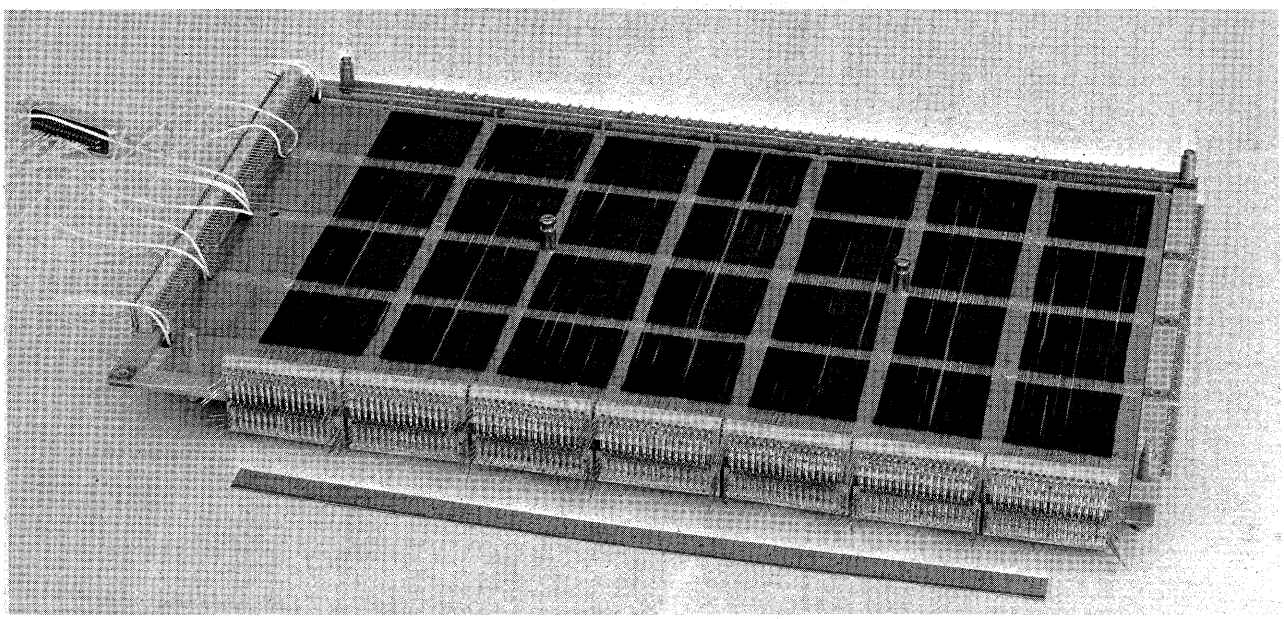


Figure 14. Magnetics array, decode diodes on periphery.

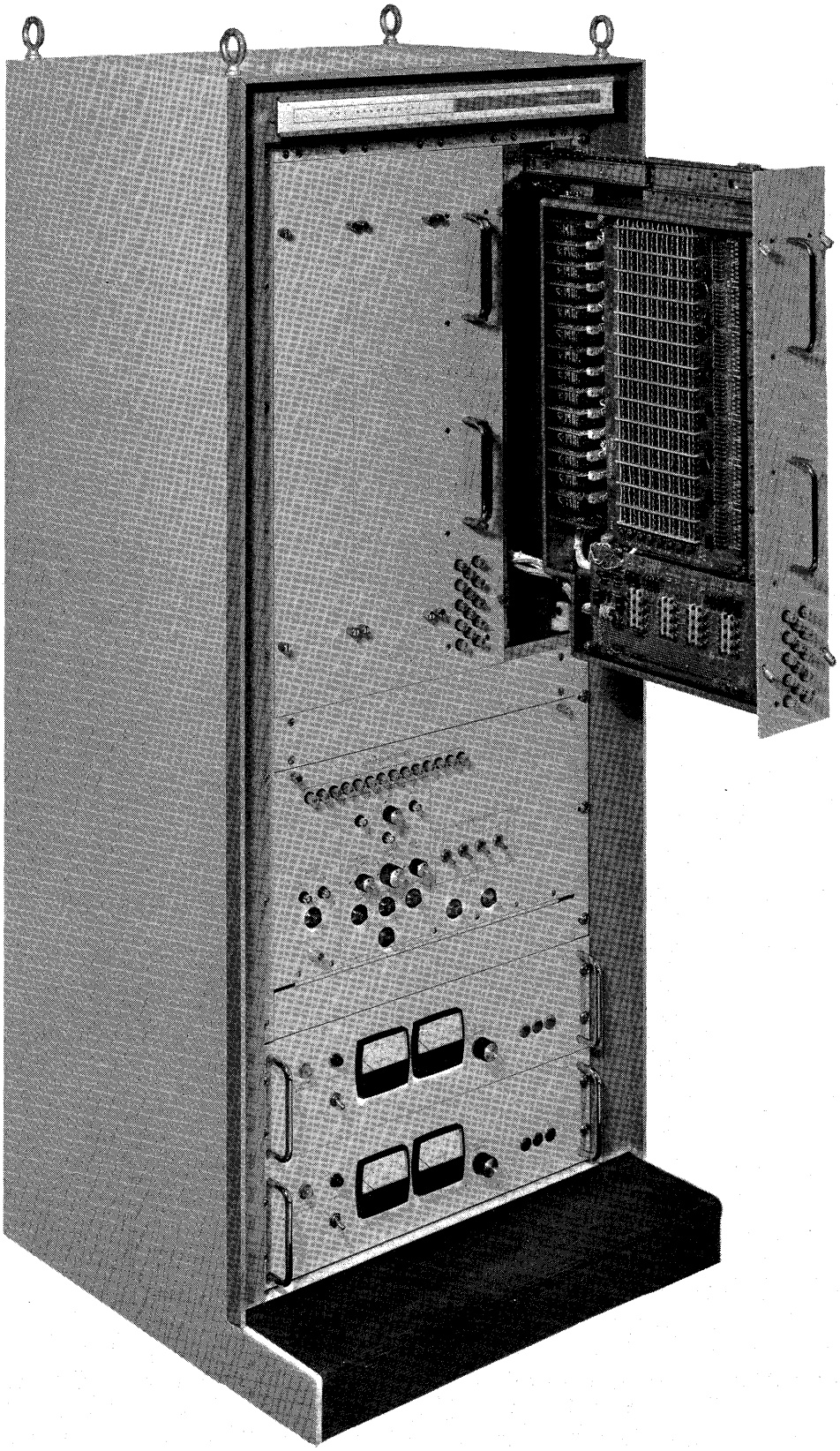


Figure 15. Coincident current three-dimensional Schmoo.

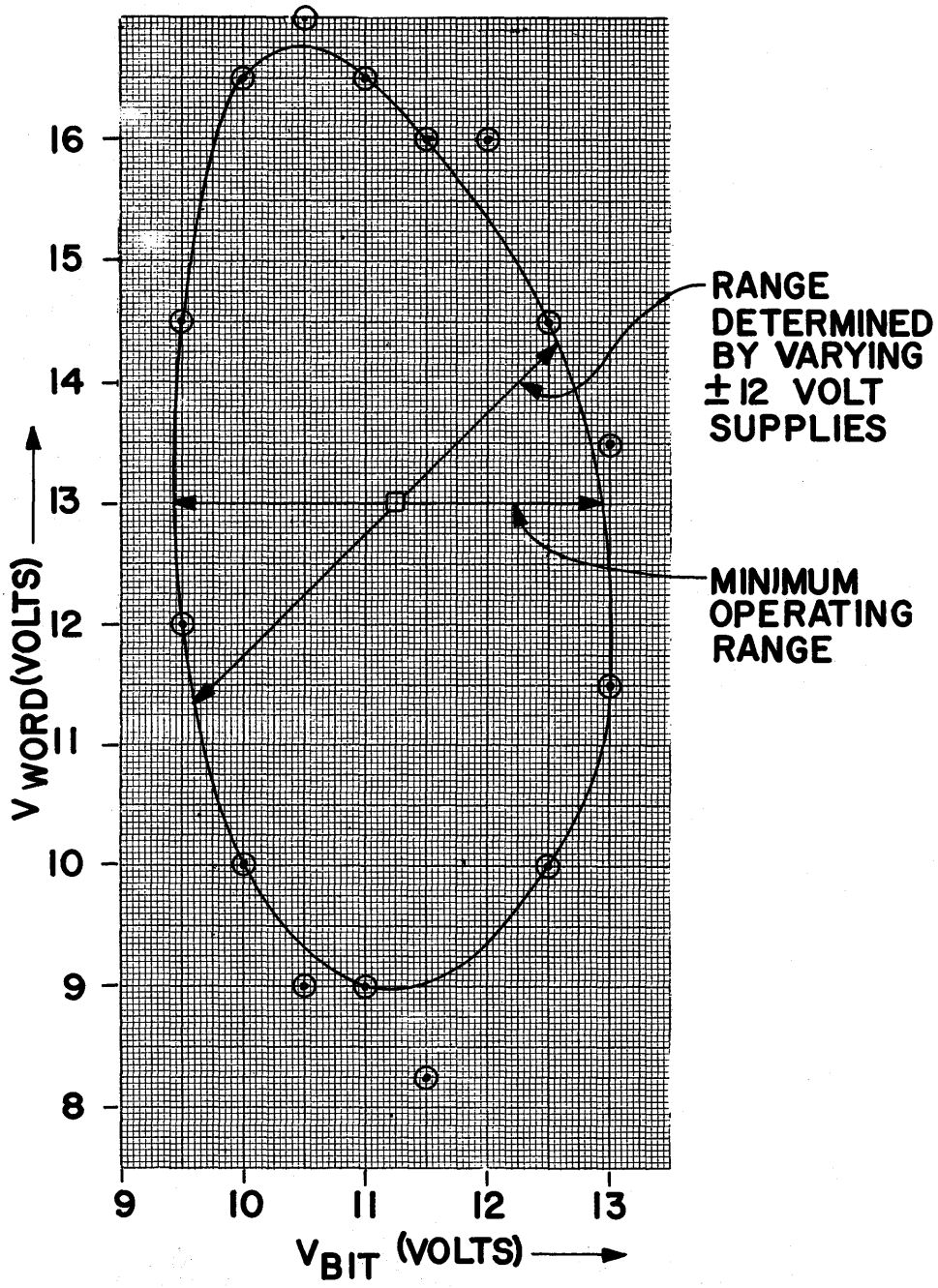



Figure 16. Operating margins, I word versus I bit with worst pattern.

one-dimensional Schmoo, which would quite adequately define system tolerance.

All drive currents are derived from ±12 volt supplies which also is used throughout the system. In margining the logic voltage, therefore, the drive current would also be margined by the same percentage (±5 percent).

REFERENCES

1. H. Amemiya, T. R. Mayhew and R. L. Pryor, "A $10^5$ Bit High Speed Ferrite Memory System—Design and Operation," *Proceedings, Fall Joint Computer Conference,* Spartan Books, Washington, D.C., 1964 pp. 123-145.

2. Meyerhoff, *Digital Applications of Magnetic Devices,* John Wiley & Sons, New York, 1960.

3. J. R. Freeman, "Pulse Responses of Ferrite Memory Cores," *IRE Wescon Convention Record,* 1954, pp. 50-61.

4. P. Cooke and D. C. Dillistone, "The Measurement and Reduction of Noise in Coincident Current Core Memories," Institute of Electrical Engineers, Sept. 1962. pp. 383-389.

5. C. P. Womack, "Schmoo Plot Analysis of Coincident Current Memory Systems," *IEEE Transactions on Electronic Computers,* Feb. 1965, pp. 36-44.

# DESIGN AND FABRICATION OF A MAGNETIC, THIN FILM, INTEGRATED CIRCUIT MEMORY*

T. J. Matcovich and W. Flannery
*Sperry Rand Corporation*
*UNIVAC Division*
*Blue Bell, Pennsylvania*

## INTRODUCTION

Techniques have recently been developed for using uncased integrated circuits in electronic systems. The use of uncased integrated circuits in computers will lead to the development of more reliable, more economical, and physically smaller computers than can be fabricated from discrete components or packaged integrated circuits. Before these advantages can be realized, practical techniques must be developed in the following areas:

1. Memory design
2. Logic design
3. Chip testing
4. Chip bonding
5. Chip passivation

All these areas are under study, and the initial results are promising. UNIVAC has been concentrating on developing techniques for achieving practical memory design, logic design, chip testing, and chip bonding. Several integrated circuit manufacturers are studying techniques for chip passivation, and economical processes should be developed within a few years. The design and fabrication of an integrated circuit, thin magnetic film memory system is described in this paper.

In addition to using uncased integrated circuits, the memory to be described also makes use of evaporated wiring and insulating layers. When fast rise times are required, the usual requirement of hundreds of milliamperes of drive current for operating magnetic memory elements is difficult to achieve with integrated circuits. The current requirements can be reduced by decreasing wire size to obtain a larger magnetic field per ampere or by using a storage element which operates with smaller magnetic fields. Both of these approaches are used in the memory to be described. The wire size is reduced to the point where evaporatd conductors and insulators must be used to obtain tight coupling between storage element and wires. The use of evaporated wiring has other advantages which include compatibility with uncased integrated circuits and excellent reproducibility.

The memory system design is described in the next section. Emphasis is placed on the electrical characteristics of the system. The advanced fabrication techniques are described in the following sec-

*Supported in part by Air Force Material Laboratory, Research and Technology Division, Air Force Systems Command, United States Air Force.

tion. Production vacuum evaporation equipment and the chip testing and bonding process are also described. The physical layout of the memory and fabrication steps are given in the next to the last section, and the results are discussed and conclusions presented in the final section.

## MEMORY SYSTEM DESIGN

The memory element configuration is shown in Fig. 1. The element is a thin film of electroplated magnetic material and is wired in a conventional

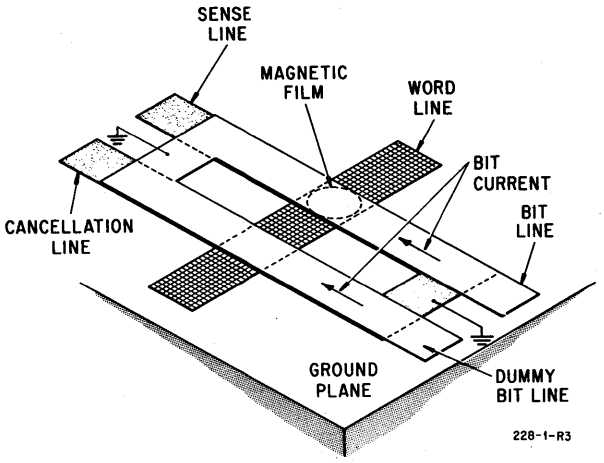



Figure 1. Memory element configuration.

manner with bit, dummy bit, and sense and cancellation lines. The word lines are 0.005 inch wide and are located less than 0.0005 inch from the magnetic element; drive fields of about 100 oersteds per empere are produced by the word drive currents. The memory operates with word drive fields of about 5 oersteds, and, consequently, requires only 50 milliamperes of word current. Bit and sense lines are 0.004 inch wide, and the memory requires only 20 milliamperes of bit current. The magnetic film is 800 angstroms thick and has an anisotropy field ($H_k$) of 1.5 oersteds and coercivity ($H_c$) of 2.0 oersteds.

The memory system consists of a 64-word, 24-bit-per-word rectangular array of storage elements, octal decoders, address registers, a selection matrix, word drivers, sense amplifiers, data registers, and bit drivers. All of these units are mounted on a common substrate. The circuits are all integrated and mounted (unpackaged) face-down onto the evaporated aluminum wiring on the substrate.

The memory operates in a linear-select, destructive-readout mode with a complete read-write cycle time of 250 nanoseconds. The memory element output is about 1 millivolt in amplitude and 5 nanoseconds in width at the base. Output from the data register is at standard logic levels, and power dissipation is about 4 watts.

The memory system circuitry is contained on 178 integrated circuit chips. There are 10 different chip types of which 2 are of standard and 8 of custom design.

The word address and driver circuitry is shown in Fig. 2. The address register consists of 6 Motorola MC 302 flip-flops, and octal decoder A consists of 8 Motorola MC 306 3-input gates. The remaining circuitry is custom designed and is divided into the three chip types illustrated in Fig. 3. All of the PNP devices are grouped on a single chip so that the best characteristics of both NPN and PNP devices can be realized. Octal decoder B consists of 8 of the PNP chips, and the 64 matrix transistors are contained on 16-word switch chips. Word current pulses with a 50-milliampere amplitude and a rise time of less than 3 nanoseconds are generated by this circuitry. The amplitude of sneak currents in unselected lines is less than one milliampere.

A block diagram of the recirculation loop is shown in Fig. 4. The differential amplifier is designed for low noise and low power operation. The amplifier pulse gain is 12, its rise time is 3 nanoseconds and its common mode rejection ratio is 320. The amplifier stages are capacitor-coupled, and the coupling capacitors are included on the single-ended amplifier chips. The design shown in Fig. 4 makes it possible to use identical chips for the two single-ended stages. The pulse gain of the 2 cascaded single-ended stages is adjustable from 100 to 200, and the rise time is 3.5 nanoseconds. A strobe circuit which is used to gate the amplifier off except during the read cycle is on a separate chip. The data register is a standard Motorola MC 302 flip-flop. The bit driver circuitry requires both NPN and PNP devices. The integrated bit driver is fabricated on two chips, one containing all NPN devices and the other all PNP devices. The chips are designed so that one NPN chip and one PNP chip connected together form two bit drivers.

A typical chip is shown in Fig. 5. This is the single-ended amplifier chip, and the coupling capacitors can be clearly seen. All chips are

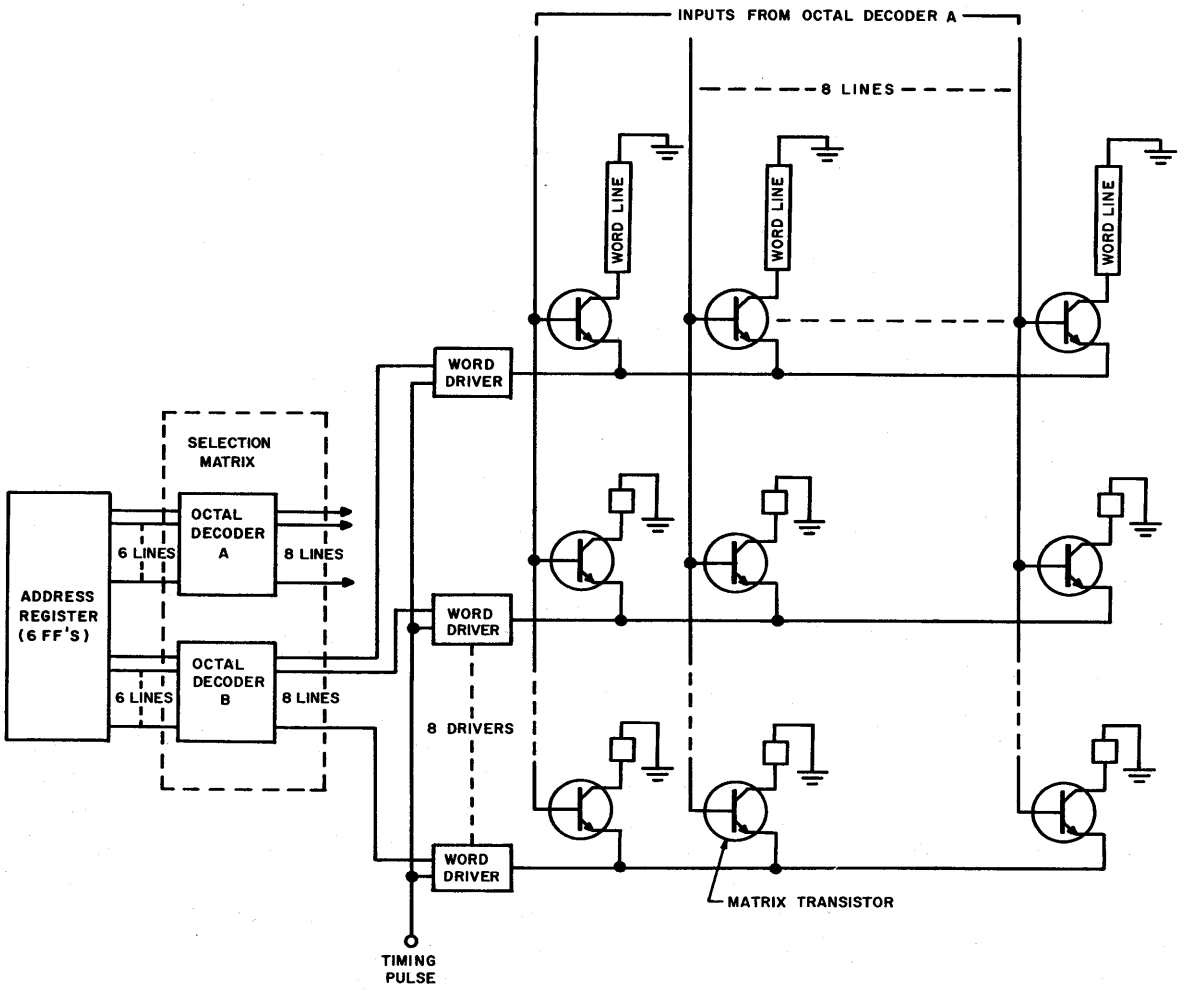



Figure 2. Word address and driver circuitry.

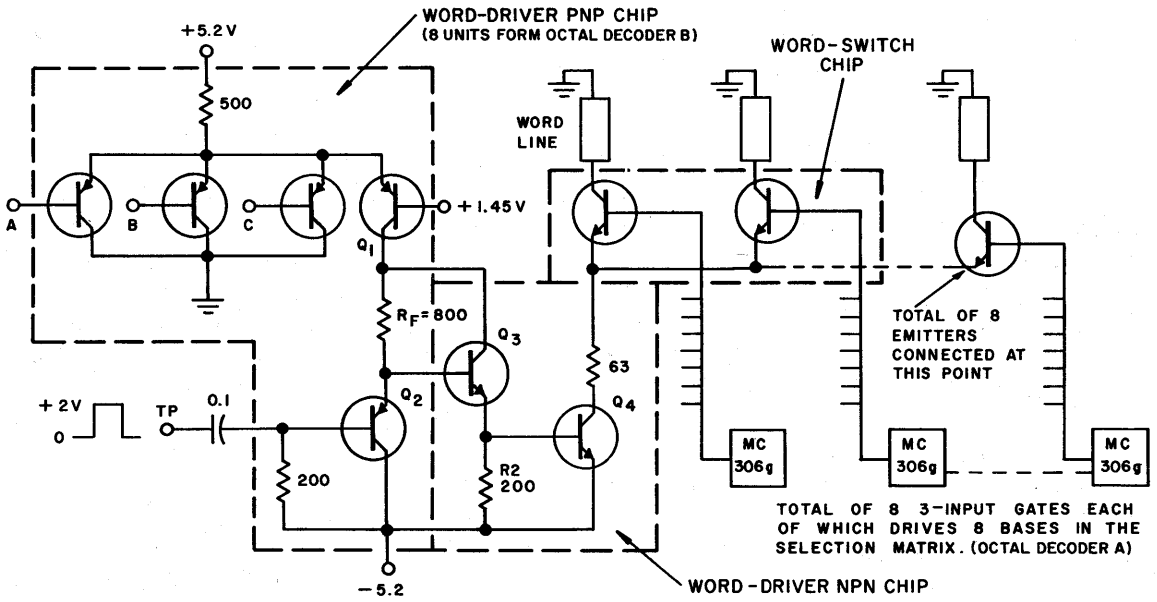



Figure 3. Word driver and selection matrix.

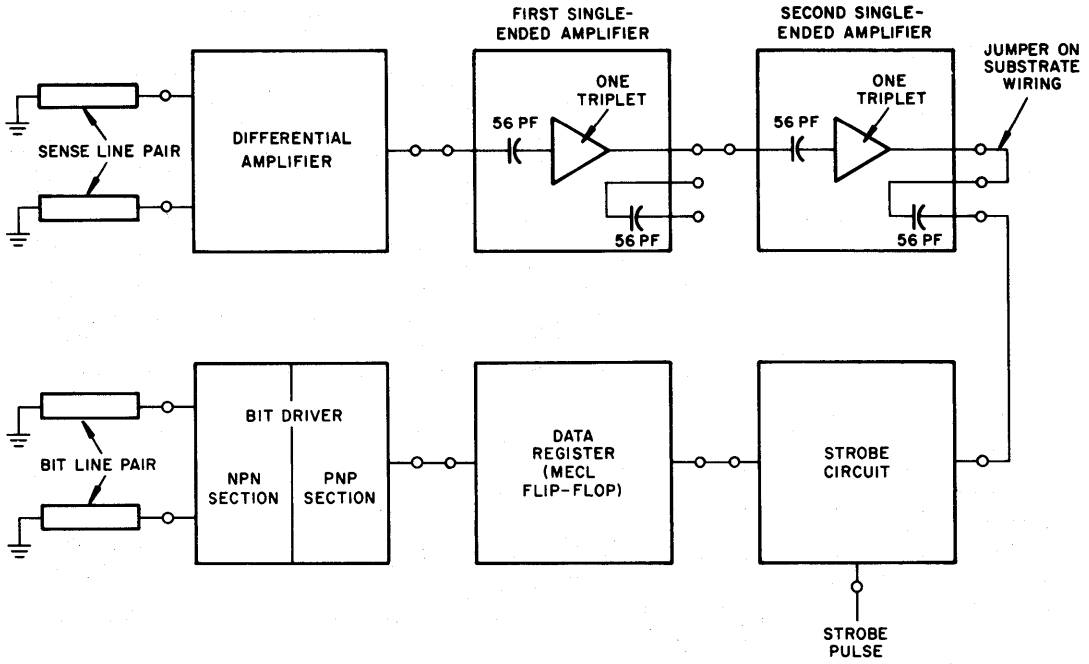



Figure 4. Recirculation loop.

0.050-by-0.050-by-0.006 inch in size and are glass passivated for enviromental protection. Critical circuits contain custom designed transistors. Use is made of evaporated nichrome for close tolerance resistors.

## ADVANCED MANUFACTURING TECHNIQUES

The most serious problem resulting from the elimination of the integrated circuit package is that of providing environmental protection for the chip. The commonly employed passivation technique is to grow a silicon dioxide layer on the surface; this passivation coating has proved inadequate in environmental and life tests. Other techniques, which include the use of thin layers of low-temperature glass, thick layers of high-temperature glass, epoxy encapsulation, and silicone rubber encapsulation, have been tried with encouraging degrees of success. A satisfactory passivation technique will probably become available within a few years.

Other problems encountered with uncased chips are handling, testing, and bonding. A number of micromanipulators with vacuum pick-up devices are commercially available and provide adequate handling facilities. The problem of testing the chips is under study at UNIVAC. Chips are usually tested before wafers are diced or after they are packaged. Since the uncased chips are never packaged, only the wafer probing techniques are applicable to chips. Wafers are usually probed with long, needle-point, metal probes attached to small manipulators. These probes damage the probed area and are awkward to manipulate. More seriously, the size and shape of the probe make high-frequency testing of the chips impossible due to the inductance of the probe leads.

A high-frequency chip testing device has been developed at UNIVAC. A schematic drawing of this test equipment is shown in Fig. 6. The chip to be tested is held by a vacum pick-up and positioned on a test card. The test card has a set of pedestals which correspond to the pad locations on the chip. The test circuitry is located adjacent to the pedestals on the test card, and no long, high-inductance leads are required. Consequently, high-frequency testing of the chip as well as low-frequency testing, is possible. A photograph of the chip test equipment is shown in Fig. 7.

The problem of bonding the chips to the circuit assembly has been solved by the development of an ultrasonic bonding technique. Pedestals are evaporated on the substrate at locations corresponding to the pad locations on the chip. Interconnect wires are evaporated onto the substrate to interconnect pedestals, memory elements, and external connec-

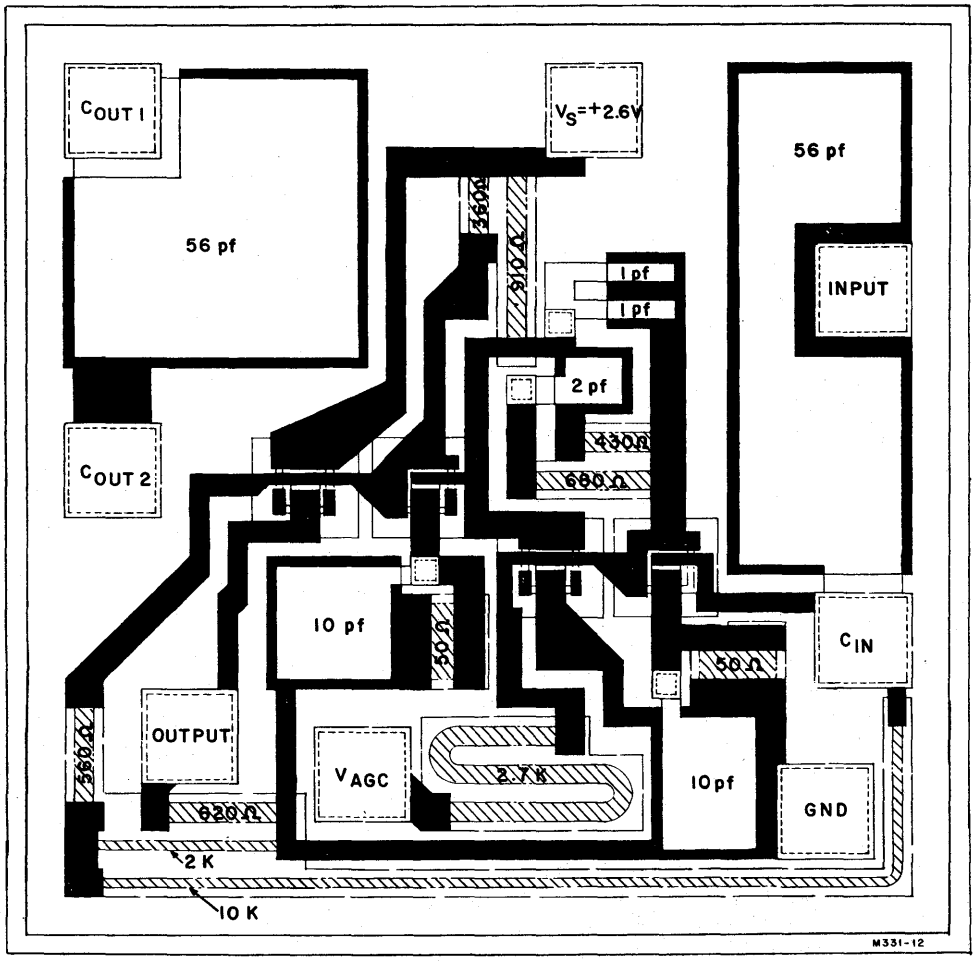
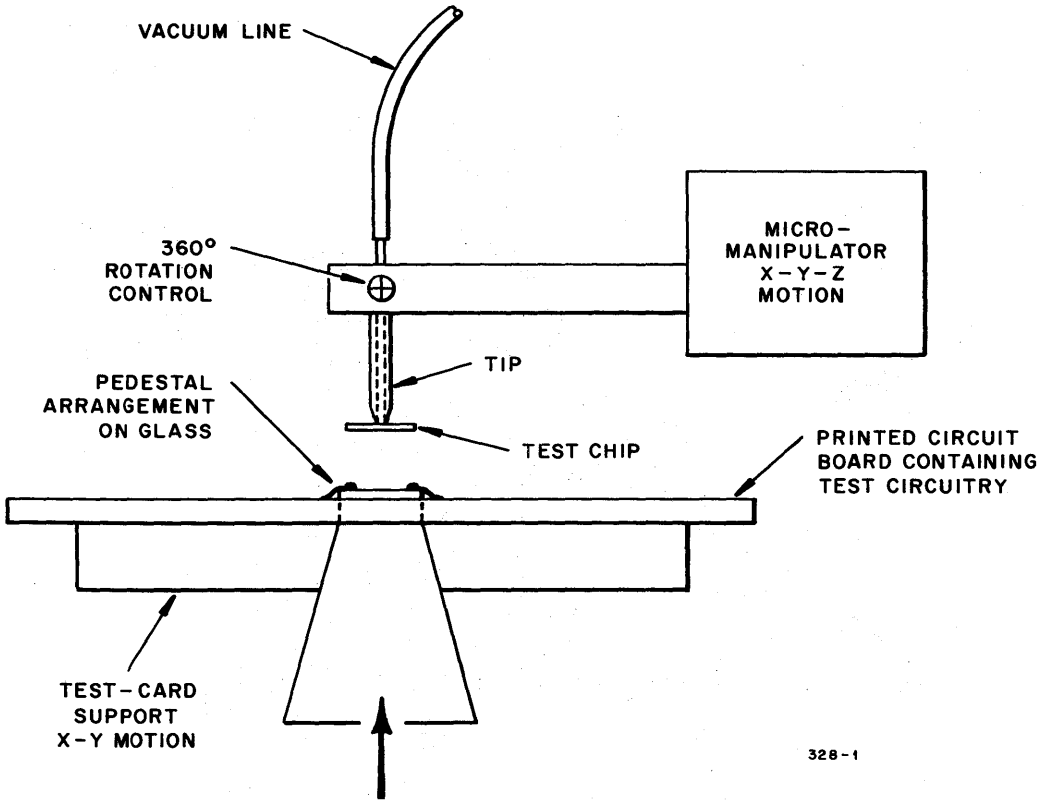



Figure 5. Single-ended amplifier chip layout.

tion pads. The chip is located over the pedestals, and ultrasonic energy is applied. As many as 14 bonds are made simultaneously on a single chip. The apparatus used for performing the bonding operation is shown in Fig. 8, and a photograph of a bonded chip (viewed through the bottom of the transparent substrate) is shown in Fig. 9.

The problem of wiring substrates by evaporation appears formidable; however, practical production equipment has been developed. A typical 64-by-24 wired array of elements is shown in Fig. 10. This array has 3 layers of conductors and 3 layers of insulators and contains over 5,000 conductor crossovers. Units of this type have been produced in conventional vacuum system bell jars during a single pumpdown. Materials are evaporated through masks; masks, sources, and substrates are manipulated externally while the system is pumped down. Registration between masks is held to 0.0002 inch over a 1-by-2-inch substrate area. Although only a single pumpdown is required for completing the wiring of a memory system, production is limited to one or two per 8-hour day.

A production vacuum system has been developed for evaporating the conductors and insulators. An artist's sketch of the system is shown in Fig. 11, and a photograph of the prototype system is shown in Fig. 12. The input and output boxes contain substrate holders, each of which has a 13-unit capacity; these boxes are shown in Fig. 13. A trolley carries a substrate from the input chamber to the main chamber where all of the wiring is deposited in a sequence of evaporations through appropriate masks. The mask changer and operating mechanism



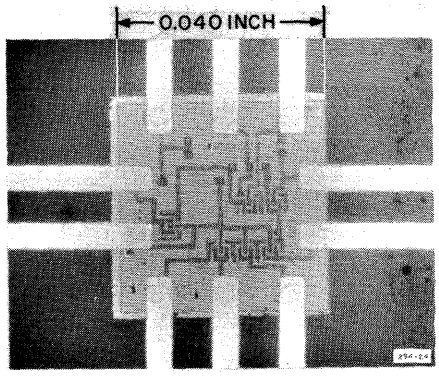




Figure 6. Schematic diagram of chip test equipment.

are shown in Fig. 14. When the evaporations are completed, the trolley carries the substrate to the output box and delivers a new substrate from the input chamber to the main chamber. When the substrate supply is exhausted, the input and output boxes are isolated from the system, new substrates are added, and completed substrates are removed. These boxes can be pumped down to operating pressure in 10 minutes. Consequently, the evaporation processes are not delayed for lack of substrates. The prototype system shown contains a single main chamber. This system can be easily expanded to include several main chambers. The following are some of the advantages multichamber systems provide:

1. Increased production by parallel operation.
2. Continuous production by sequentially isolating single chambers from the system for routine maintenance.
3. Provisions for performing low-vacuum and high-vacuum deposition techniques in different chambers.

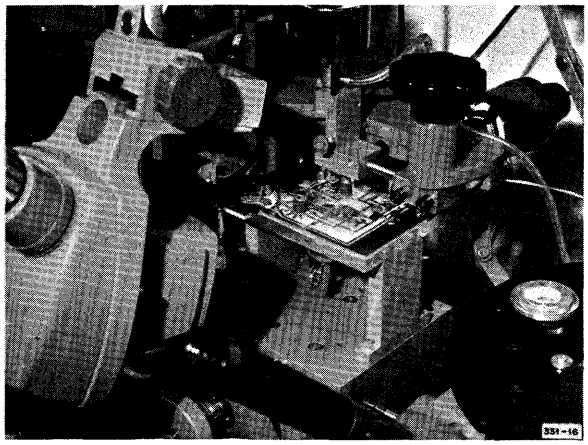


Figure 7. Photograph of chip test equipment.

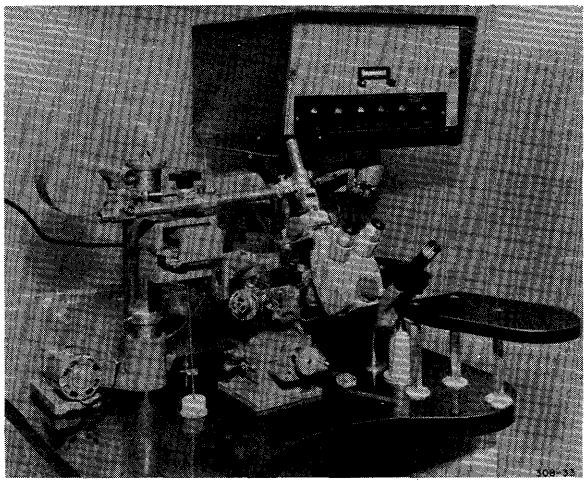


Figure 8. Ultrasonic face-down-bonding equipment.

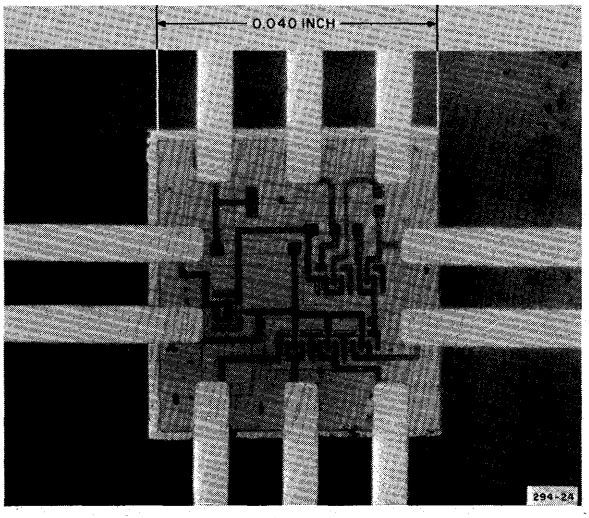



Figure 9. Photograph of face-down-bonded chip.

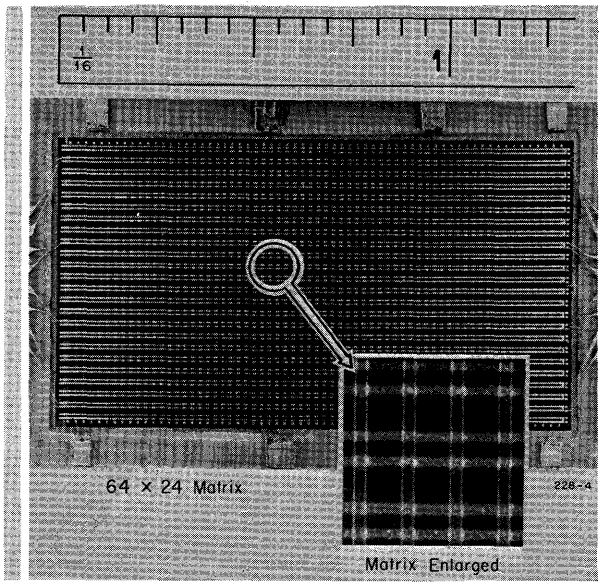



Figure 10. Photograph of 62-by-24 matrix.

The system shown can be used to produce 3 fully wired 6-by-3-inch memory planes per hour.

## MEMORY SYSTEM FABRICATION

A plane view of the memory system is shown in Fig. 15. Word lines are on 0.020-inch centers, and bit and sense line pairs are on 0.060-inch centers. All lines are made of evaporated aluminum 20,000 angstroms thick; word lines are 0.005 inch wide, and bit lines are 0.004 inch wide. A cross section view through the plane is shown in Fig. 16.

The recirculation loops are equally divided on the left and right sides of the memory matrix. The entire word address circuitry and the word drivers are located at the top of the matrix. Two spring-type connectors provide all voltage and signal leads for the plane.

The fabrication steps for manufacturing the memory system are shown in Fig. 17. A glass substrate is covered with a copper ground plane formed by a combination of evaporation and electroplating steps. The copper serves as the conductor for electroplating the film of magnetic alloy, which is applied next. Following this step, the 64-by-24 array of storage elements is photoetched, and the magnetic memory properties of the resulting elements are measured in a test setup which simulates memory operation. Planes with defective bits are rejected. After testing, the plane is installed in the production vacuum system where all the pedestals,

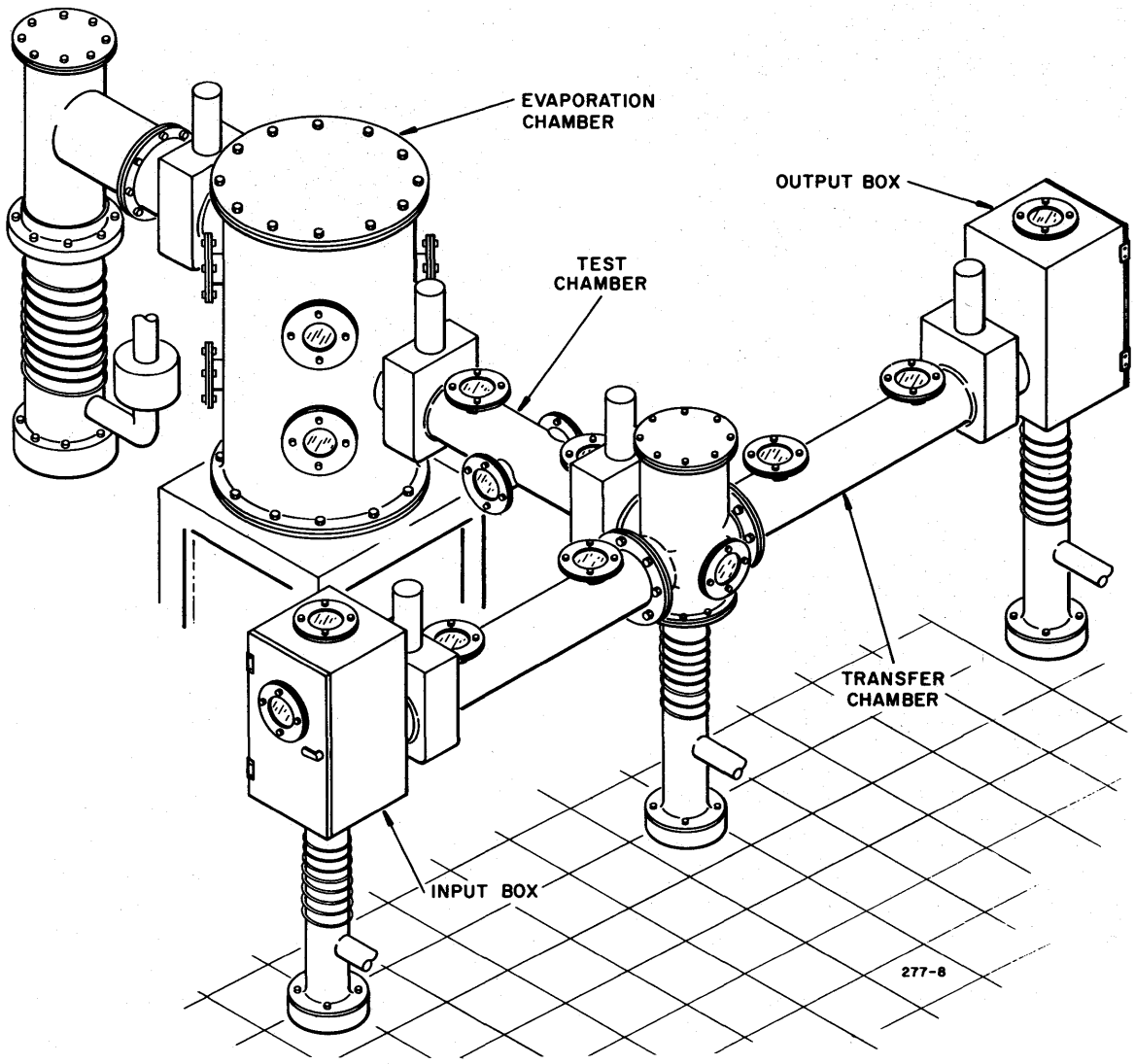



Figure 11. Artist's sketch of production vacuum system.

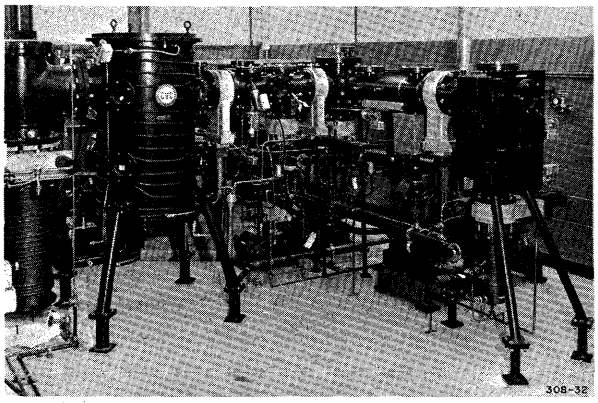



Figure 12. Production line vacuum system.

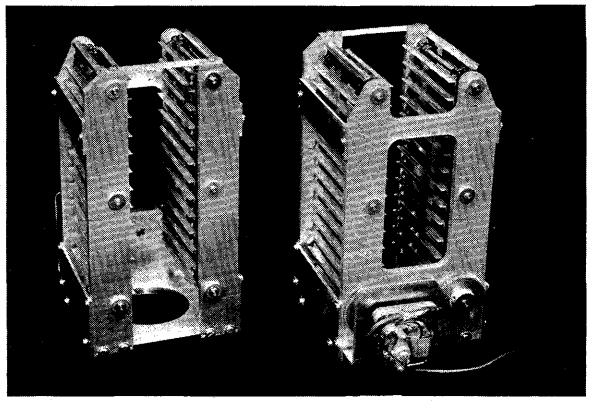


Figure 13. Substrate holder.

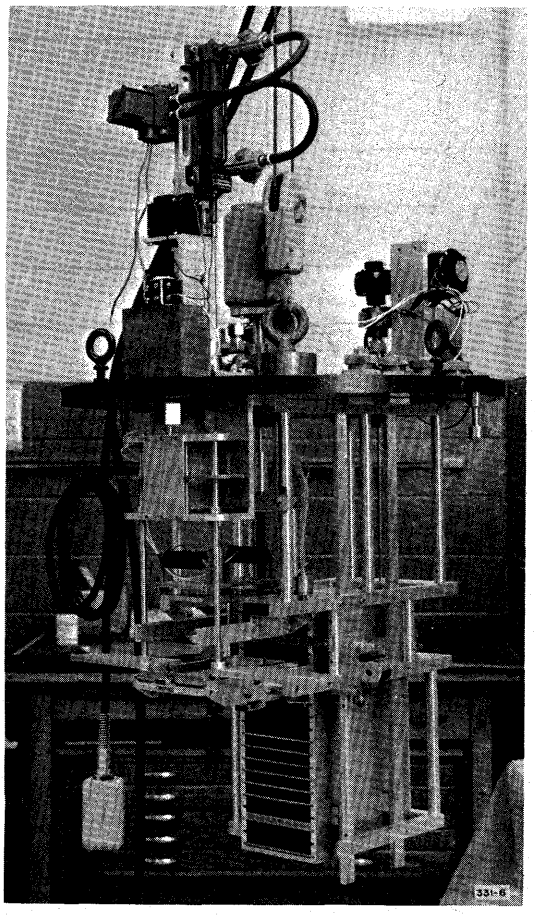


Figure 14. Mask changer and operating mechanism.

wires, and insulating layers are sequentially deposited through appropriate masks. All wiring is completed in a single cycle through the vacuum system, and no etching steps are required. The wiring on the planes is next tested for continuity, shorts, and resistance. The pretested integrated chips are attached to complete the system. The memory is then given an operational test, encapsulated, and finally retested.

## DISCUSSION AND CONCLUSIONS

The memory described does not represent the best that can be produced by the new techniques developed, but it does establish the applicability of the techniques to the fabrication of thin film memory systems.

The system could be made denser. The bit spacing and chip spacing were set by energy dissipation considerations. The bit spacing in the array shown in Fig. 10 was 0.030 inch; the 0.060-inch spacing was used in the final model so that the recirculation loop circuit chips could be spread over a larger area without fanning out. If the chips are spaced as shown in Fig. 15, temperature rise during operation is limited to a practical value. When the energy dissipation problem is solved, the chips can be placed as close as 0.010 inch apart by using the fabrication techniques already developed.

The system capacity is adequate only for scratch-pad applications. Larger systems can be made, but there is a limit to the length of line that can be vacuum evaporated through masks. A practical upper limit is presently considered to be 6 by 6 inches. When the energy dissipation problem is solved, 138,000 bits, including circuitry, could be placed on the substrate. This is still not adequate for large systems; however, larger systems could be fabricated by stacking planes.

The cost per bit of the memory system could be reduced by making higher-capacity planes. However, the most substantial decrease in cost will re-

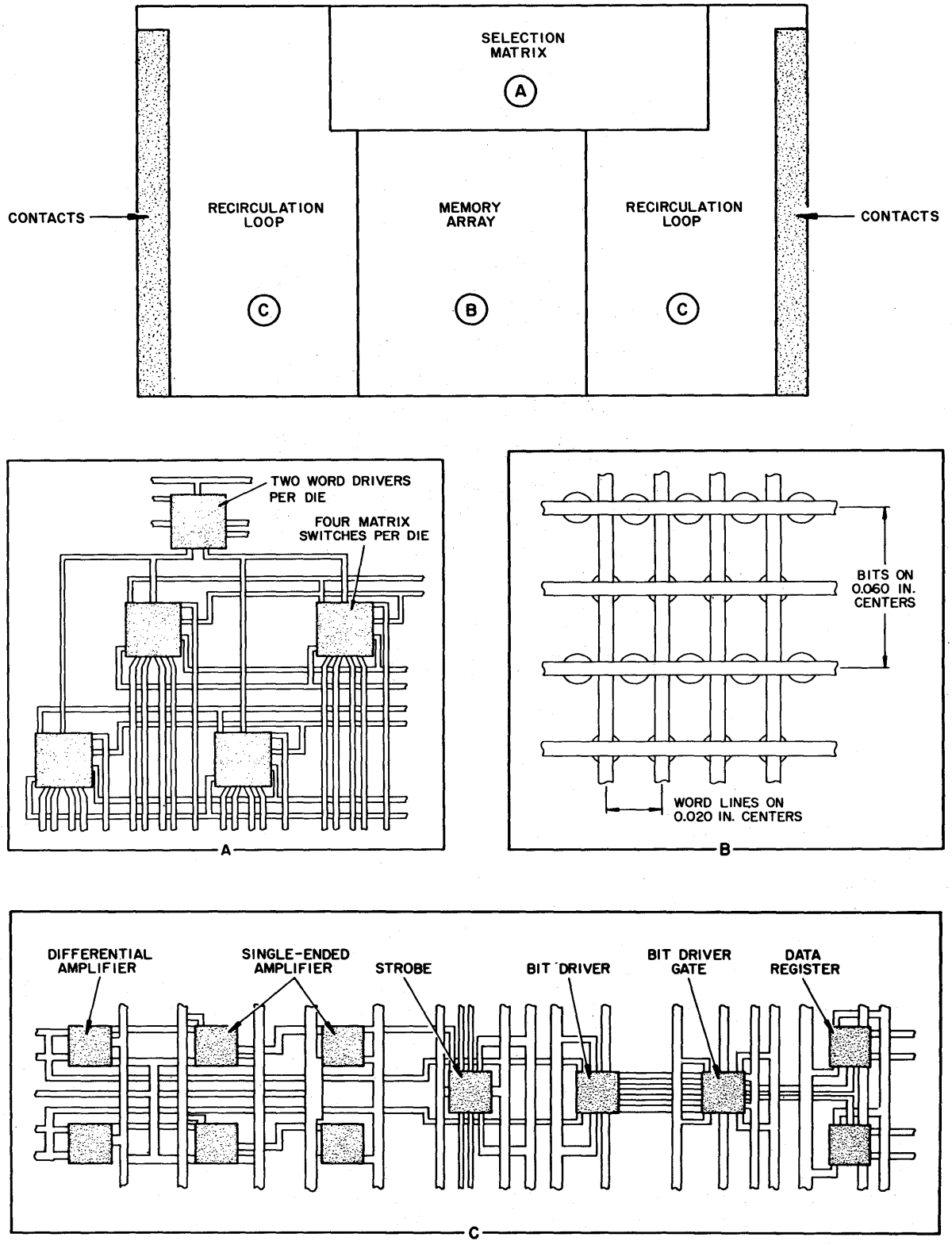



Figure 15. Plane view of memory system.

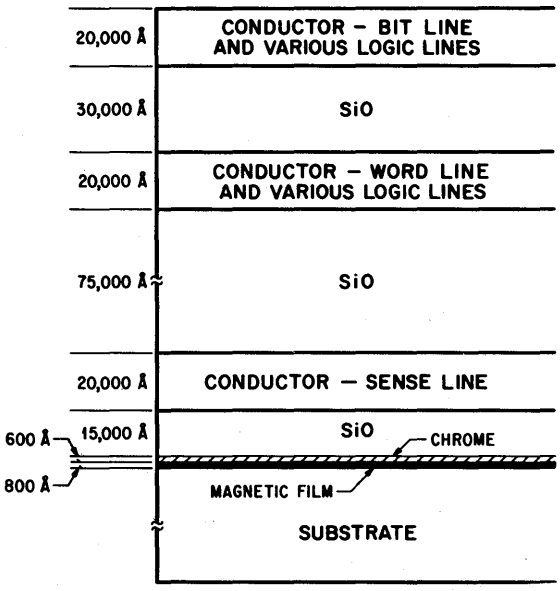



Figure 16. Cross section through memory plane.

sult from reduced integrated circuit prices. The cost of integrated circuits has decreased substantially in the last year, and further reductions are anticipated. The use of uncased integrated chips should result in additional substantial reductions in the cost per circuit. The cost should reach such a low level that minimizing circuitry will not be a significant system design criterion. A parallel situation existed when integrated circuits were introduced with the result that transistors became less expensive than resistors. The estimated cost of 4,096-word memories of the type described is $0.02 per bit based on a $2.00-per-chip price. Since chip prices eventually fall below $1.00 per chip, memory systems costing less than $0.01 per bit are anticipated.

The reduced chip price and the fabrication techniques described may lead to a new concept in memories. Memories are presently formed by interconnecting components, such as storage arrays, word drivers and sense amplifiers. Rugged, compact, and inexpensive memory modules containing all the system circuitry may be made available in standard sizes for use as computer system components. This advance, which is potentially as significant as was the introduction of integrated circuits, which changed the component unit from the resistor, capacitor, and transistor to the entire circuit, could substantially alter the present concept of computer design.

## ACKNOWLEDGMENTS

The design and fabrication of the memory system described could only be accomplished through the close cooperation of specialists in several fields. These people constitute the Molecular Systems Group at the Blue Bell Laboratories of UNIVAC, and the authors wish to acknowledge their indebtedness to all members of the group for their contributions in the development of the memory system.

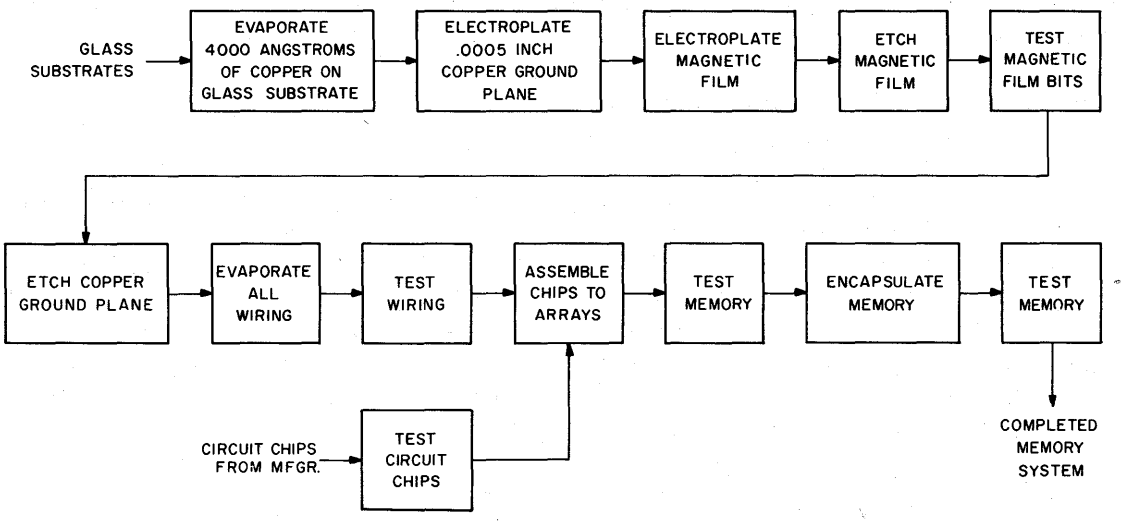



Figure 17. Production flow chart.

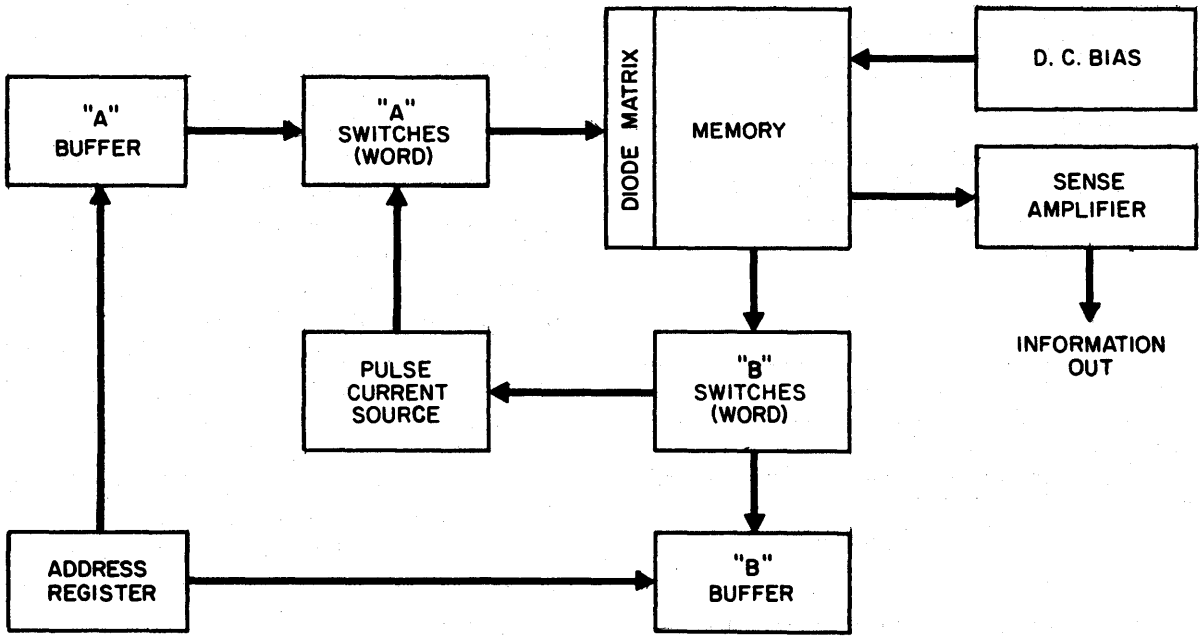



Figure 16. Memory block diagram.

Considering all sources of delay, a memory cycle of 100 to 150 nanoseconds is easily obtainable in a memory consisting of a few thousand 32-bit words.

## CONCLUSIONS

The experimental results have shown that the woven read-only memory array has the required properties for achieving very short memory cycle times. Another attractive feature of the memory is the high packing density which can be achieved; the results quoted on area packing density indicate that over 50,000 bits per cubic inch are obtainable.

The description of the fabrication method for the memory has shown that stacks can be assembled at very low cost per bit due to the highly automated nature of the memory production process. If the costs become as low as anticipated, it may even be possible to consider the woven memory array as a very high-speed semipermanent memory by using "throw-away" planes.

Finally, the results show that the memory element is insensitive to mechanical stresses and temperature changes. The woven array will therefore be applicable in those areas where highly reliable operation is required under extreme environmental conditions.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the help and encouragement of A. Matsushita of TOKO, Inc., and Dr. R. H. Fuller of General Precision Inc.

## REFERENCES

1. H. Maeda and A. Matsushita, "Woven Thin Film Memories," *IEEE Trans. on Magnetics,* vol. 1, p. 13 (Mar. 1965).

2. P. Kuttner, "The Rope Memory: A Permanent Storage Device," *AFIPS Conference Proceedings,* p. 45, 1963.

3. D. M. Taube, "A Short Review of Read-Only Memories," *Proc. IEE,* vol. 110, no. 1, p. 157 (Jan. 1963).

# BATCH FABRICATED MATRIX MEMORIES*

Thomas L. McCormack, Claude P. Battarel and
Harrison W. Fuller
*LFE Electronics*
*Boston, Massachusetts*

## INTRODUCTION

Present-day matrix memory fabrication techniques are relatively expensive since discrete binary memory elements are individually made and then assembled into a matrix array by means of manual or semimanual wiring. The assembly of individual matrix planes is usually followed by another expensive step wherein the planes of a memory stack are interconnected. The key to low cost matrix memories lies in integrated or batch fabrication of the memory elements and wiring structure of a plane, and also batch forming the interconnections between planes in a memory stack. Additional economy results from making the bit capacity of a plane as large as possible. The need for batch fabricated memory planes appears to be generally recognized, considering the number of suggestions that have been made for achieving the goal. Thin magnetic film memory elements were initially of interest because of their high switching speed, which made very fast memories possible, but today thin magnetic film memories are of interest principally because they offer one approach to batch fabrication, since in the meantime ways have been found for achieving high speed with ferrites.

Although low cost is the principal motivation for developing batch fabricated memory planes, the result is that such manufacturing methods also make small physical size and low power practically achievable. The small physical size of memory elements makes large-capacity planes possible, and this, together with the low-power requirements, works to reduce the cost of selection, drive and sense electronics. This reduction in electronics cost possible with batch fabricated memory planes is very important to the objective of low cost memory systems since the cost of electronics in present-day memory systems is a substantial and sometimes dominating fraction of the total.

A new approach to batch fabrication of memory planes based on the use of permalloy-sheet toroids as memory elements is employed. The etched permalloy-sheet toroid approach is attractive because the fabrication techniques are applicable to a variety of memory systems.

*This program is receiving support from the Air Force Materials Laboratory, Research and Technology Division, Air Force Systems Command, United States Air Force, under contract no. AF 33(615)-3018; and Rome Air Development Center, Research and Technology Division, Air Force Systems Command, Griffiss Air Force Base, N. Y., under contract no. AF 30(602)-3826, as well as LFE Electronics, a division of Laboratory For Electronics, Inc., Boston, Mass.

## BATCH FABRICATED MEMORY PLANES

The batch fabricated memory plane consists of flat toroids etched from sheet permalloy with the necessary control wiring formed by etching and plating copper. The wiring pattern required for planes to operate in the coincident current mode is shown in Fig. 1. A slightly modified version is used for the linear select mode. The dark circles represent the etched toroids; wiring, indicated by solid

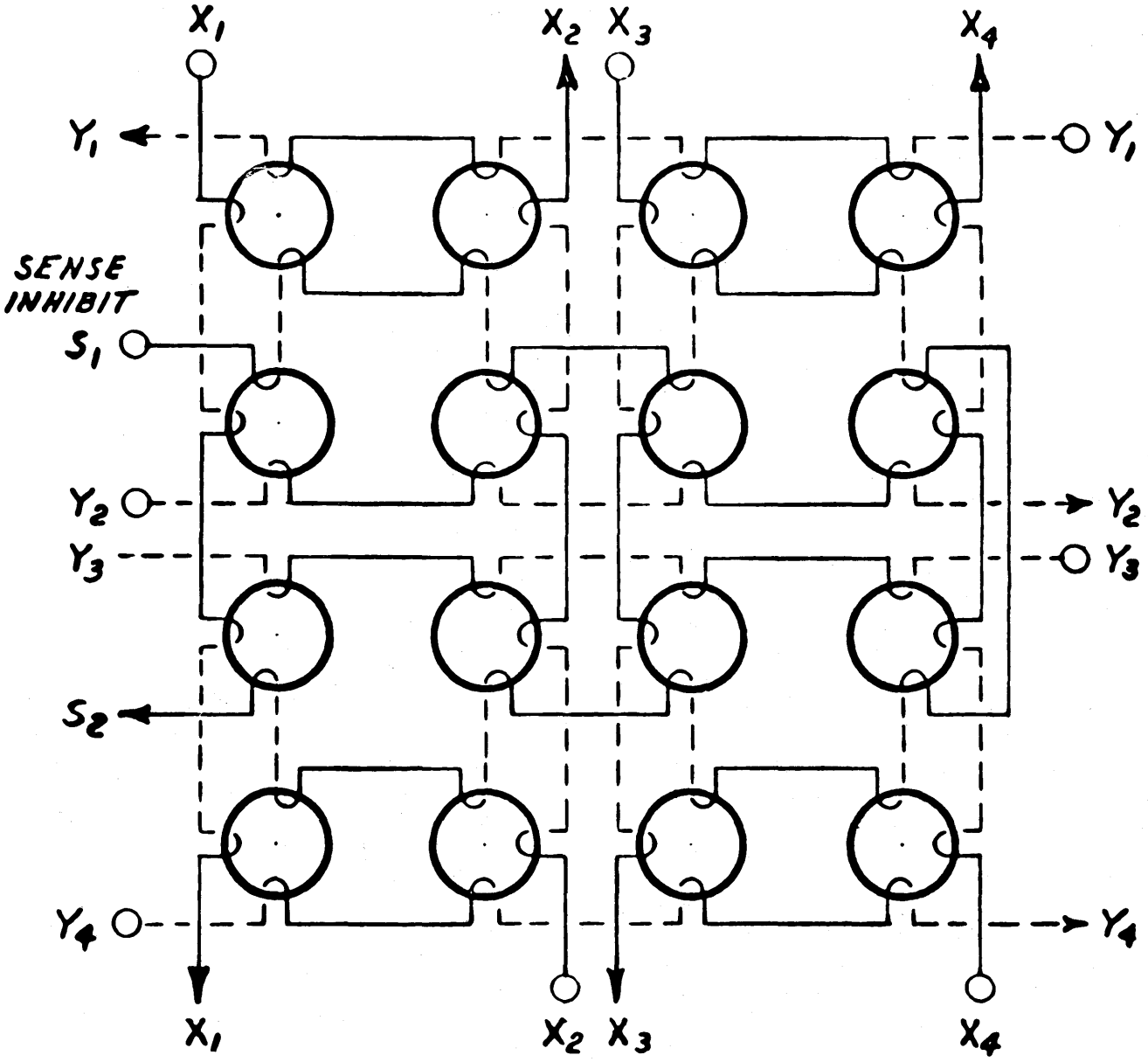



Figure 1. Wiring pattern for 4 × 4 memory matrix.

lines, is in the plane above the toroids, while the wiring indicated by dotted lines is in the plane below the toroids. Connections between the two wiring layers are made through the interior of the toroids. The topology is such that wires never cross on the same plane, thus allowing the wiring pattern to be formed by two layers of etched copper insulated from each other and the toroids but connected by means of plated regions through the interior of the toroids. Larger arrays are made by repeating this basic pattern. Figure 2 shows an early test model with toroids on 25-mil centers prepared by bonding sheet permalloy to a plastic sheet, coating it with photo resist and exposing it to the negative

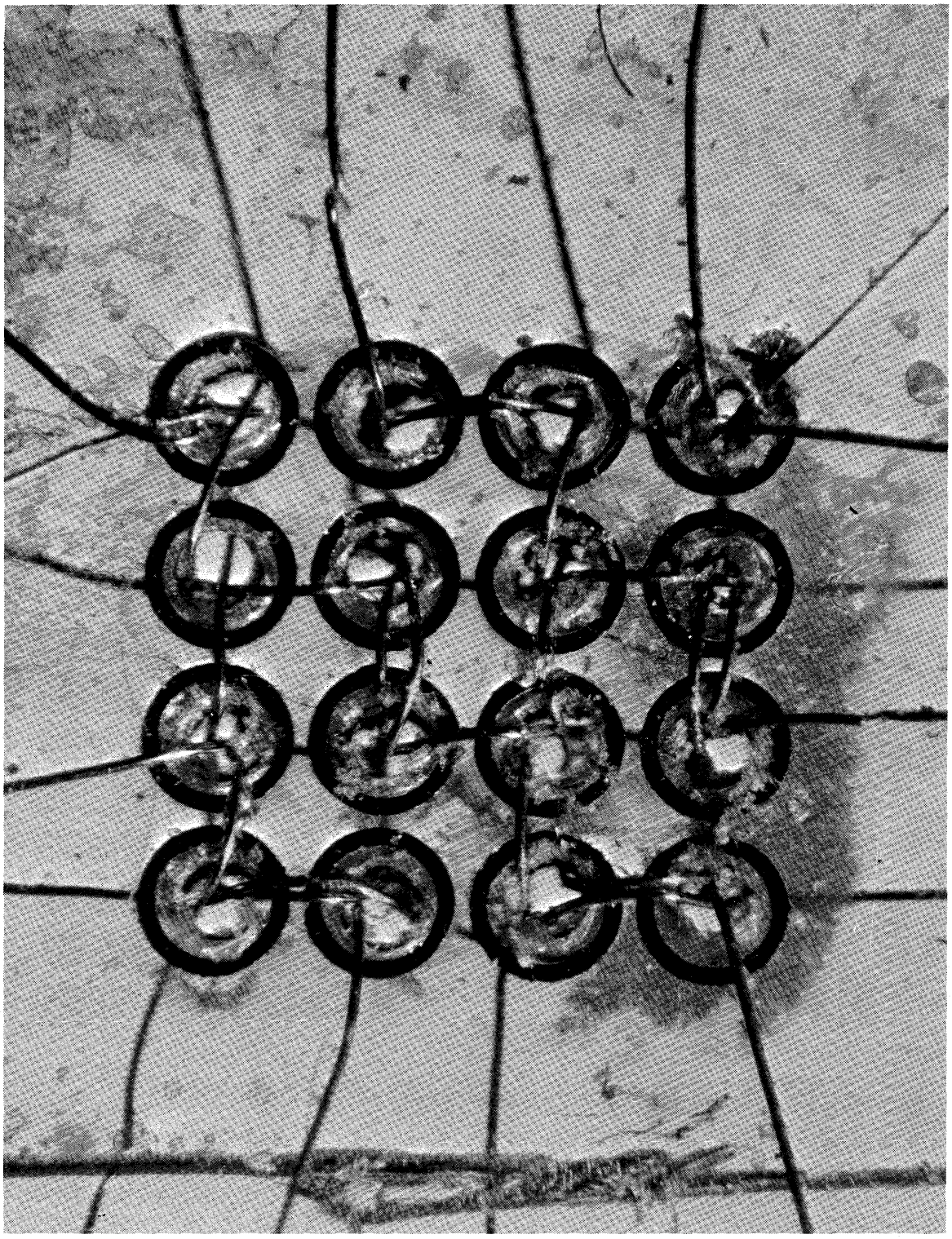


Figure 2. Hand-wired 4 × 4 matrix model. 23-Mil O.D. on 25-mil centers.

of the toroid pattern. The model was then developed and etched, and holes were burned in the plastic in the middle of the toroids. It was then hand-wired.

In Fig. 3 are shown the photo masters which are required to batch fabricate the memory array shown in the previous figures. The two lower patterns are the top and bottom halves of the wiring pattern. At

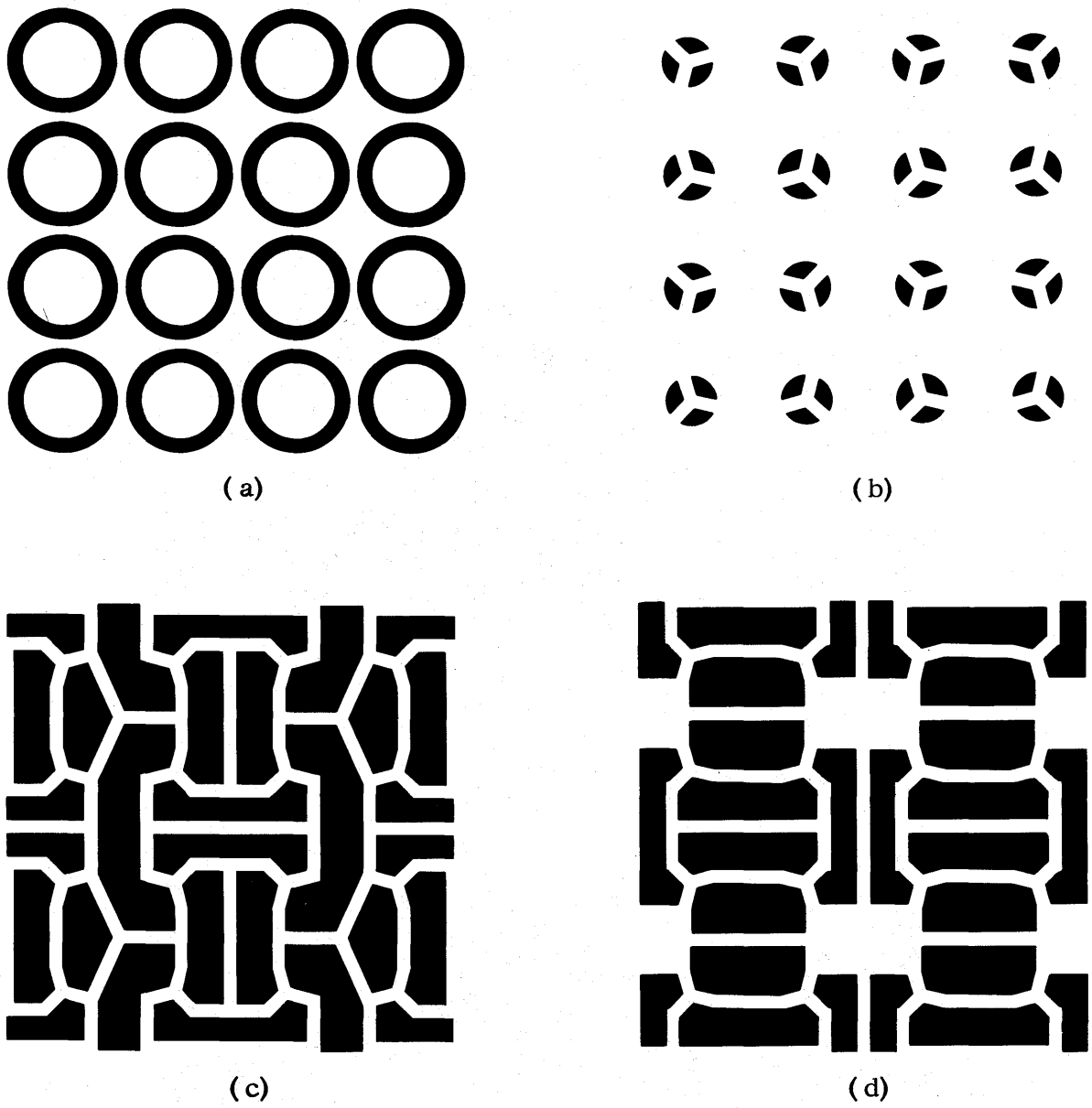



Figure 3. Partially reduced photo-etched masters for a 4 × 4 matrix model.

upper left is shown the toroid pattern, while to its right is shown the mesa pattern which produces the connection between the two wiring layers through the interior of the toroid.

The steps involved in batch fabricating a memory plane of the design shown will be described, briefly, with reference to Fig. 4 (a-e), which show cross sections of a plane in various stages of fabrication. In Fig. 4a, a 1.3-mil-thick copper sheet has been bonded to a temporary substrate via a thermoplastic adhesive. The copper has subsequently been coated with a photo resist, the mesa pattern exposed, and the resist developed. This layer has been baked and a subsequent coating of photo resist applied and

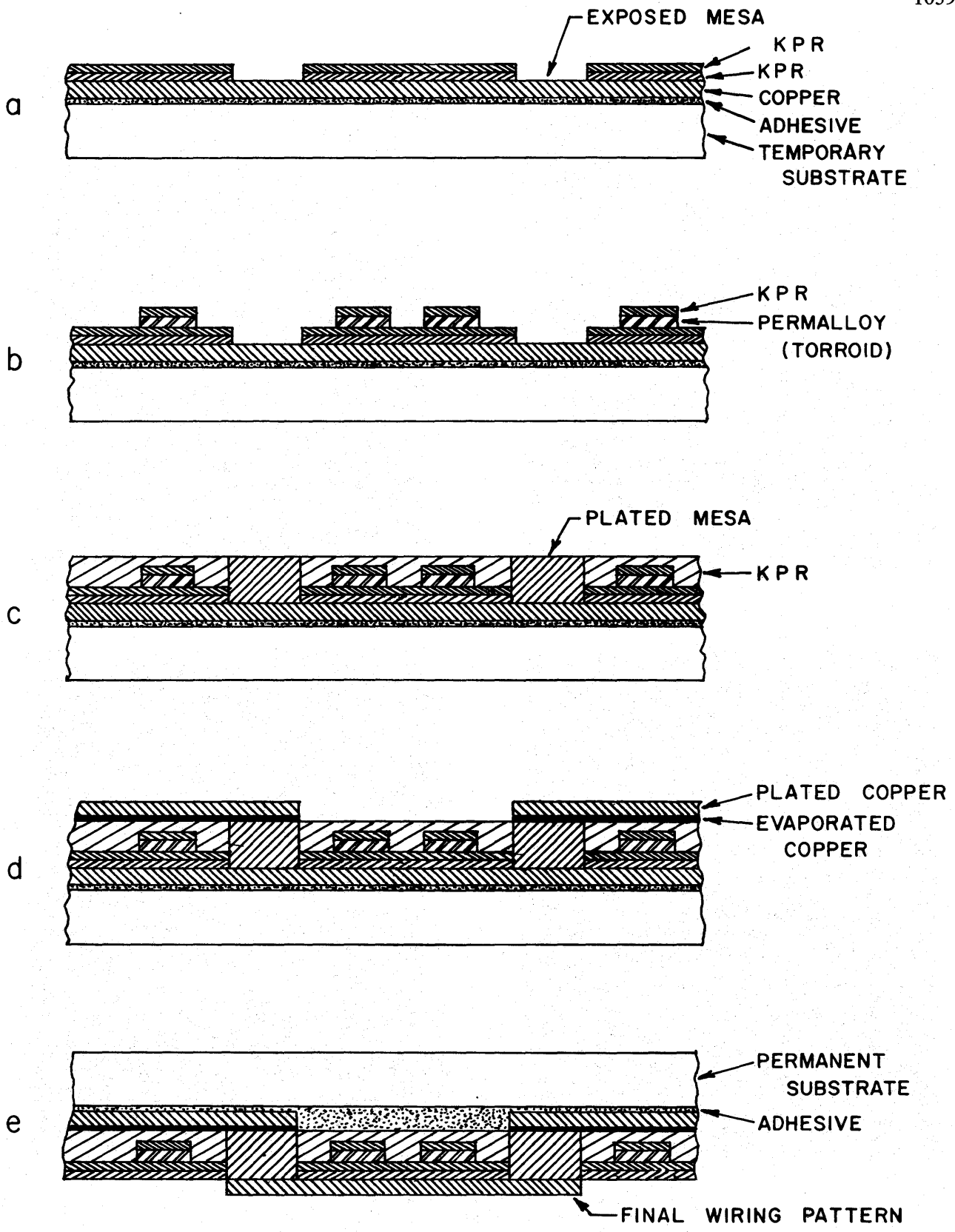



Figure 4. Steps in batch fabricating technique of memory plane manufacture.

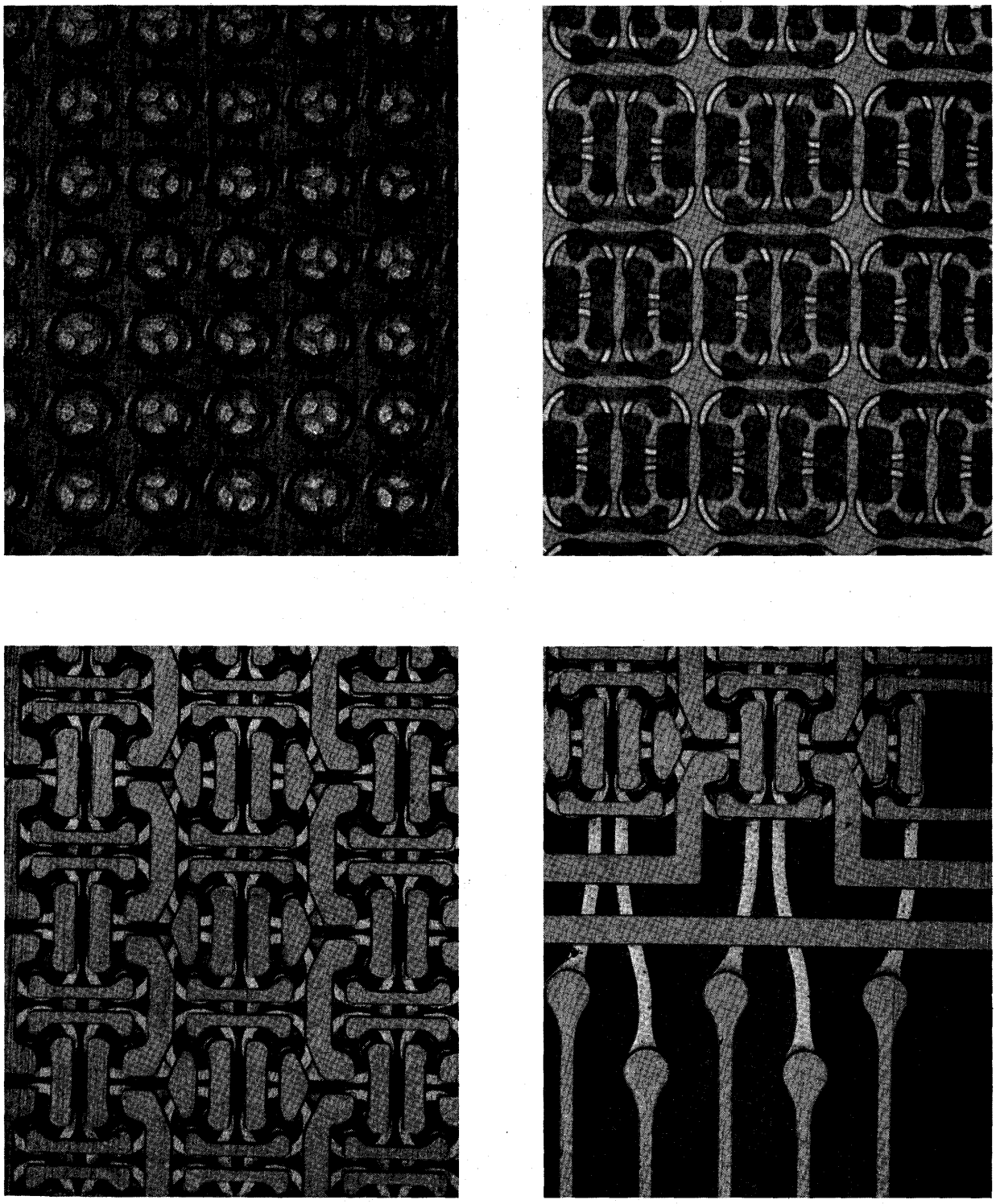


Figure 5. Fabrication stages of 64 × 64 models. *Upper left:* Toroids with mesas plated up. *Upper right:* Top wiring pattern completed. *Lower left:* Top and bottom wiring completed. *Lower right:* Lead-in from edges and top to bottom wiring feed through technique for edge connectors.

exposed to the mesa pattern as before, then developed. This layer of resist serves as the adhesive which bonds the permalloy. In Fig. 4b the permalloy has been bonded, coated with photo resist, exposed to the toroid pattern and developed, and the toroids have been etched. In 4c the specimen has been recoated with photo resist, the mesa pattern exposed and developed once more (this completely surrounds the permalloy toroid with an insulating layer of photo resist), then the specimen is placed in an electroplating bath and the mesas plated up level with the top layer of photo resist. In 4d a coating of copper has been evaporated onto the surface and electroplated up to a thickness of 1.3 mils; this surface is then coated with photo resist and exposed to one wiring pattern, developed, and etched. In Fig. 4e the specimen has been transferred to the permanent substrate, the temporary substrate removed, the newly exposed copper surface cleaned, coated with photo resist, exposed to the final wiring pattern, developed and etched.

The basic 4×4 pattern has been developed into larger models; Fig. 5 shows 64×64 models at various stages of fabrication, upper left, toroids with mesas plated; upper right, top wiring pattern completed; lower left, top and bottom wiring completed; lower right, shows details of lead-ins. Figure 6 shows a completed 64×64 memory plane. This plane is 1.6 inches square and has 4,096 toroids on 25-mil centers.

The completed planes may be stacked; the manner of accomplishing this is indicated in Fig. 7. These connections are formed by stacking and potting the planes in register, lapping off the surfaces to be interconnected, coating the surface with copper, photo resist, exposing through a mask and finally etching. In this way the plane-to-plane interconnections are formed in the same manner as the memory plane wiring. Figure 8 shows a section of a 64×64 stack of 5 planes where the edge of the copper lines is clearly visible. A stack of 5 16×16 planes is shown in Fig. 9.

Some essential features of the technique described are: (a) no new materials need to be developed; (b) the magnetic material, thin permalloy sheet, is inexpensive, is available from several sources, and rigid quality control has been in hand for many years; (c) closed flux structure memory elements allow close coupling to wiring and close spacing of elements without interaction and with low sensitivity to external fields; (d) the chemical, photochemical and electrochemical materials and processes are individually inexpensive, well-known and well-established ones; (e) the processing methods are relatively simple and allow large planes to be fabricated with simple apparatus; (f) the tolerance and resolution required in individual steps of the fabrication process are well within the present state-of-the-art; (g) the batch fabrication processes used to make interconnection wiring between planes of a memory stack are identical to those used to manufacture memory planes. These properties of the fabrication techniques give good assurance that a high yield of memory planes at low cost can be dependably and reproducibly expected, once all the steps in the manufacturing sequence are brought under control. By reason of the nature of the batch fabrication techniques, the same expectations follow for connection of individual planes to automatic test equipment and interconnection of planes in a stack.

## BATCH FABRICATED PLANE RESULTS

Early models consisted of an array of 16×16 toroids on 62.5-mil centers. These were used primarily as a demonstration of the feasibility of the process. Subsequently arrays of 16×16 toroids on 25-mil centers have been made, and the development of a satisfactory fabrication process has been a part of the program. Planes are generally fabricated in lots of 10 and as many as 9 out of 10 planes are being carried successfully through fabrication with an average of 6 out of 10 for the last 8 lots fabricated. In additions, the yield of testable planes has improved so that if a plane was processed completely through fabrication it was also a testable plane. Further, the yield of planes has steadily improved, despite the fact that planes are actually being made on an experimental basis where it is understandable that the yield may temporarily drop from the introduction of changes in the process which are expected ultimately to improve planes. In the last 3 lots of 16×16 planes fabricated, one lot had 3 out of 10 planes perfect, the next 3 out of 7 perfect, and in the last 5 out of 10 were perfect. Fabrication is now in progress on 64×64-bit planes. Of the 90 processed to date, 50 have been completed, and of these 37 were testable. The first 7 lots have been tested, with 6 planes having perfect wiring (no shorts or opens). Spot switching tests have shown $S$ curves similar to those for the 16×16 planes.

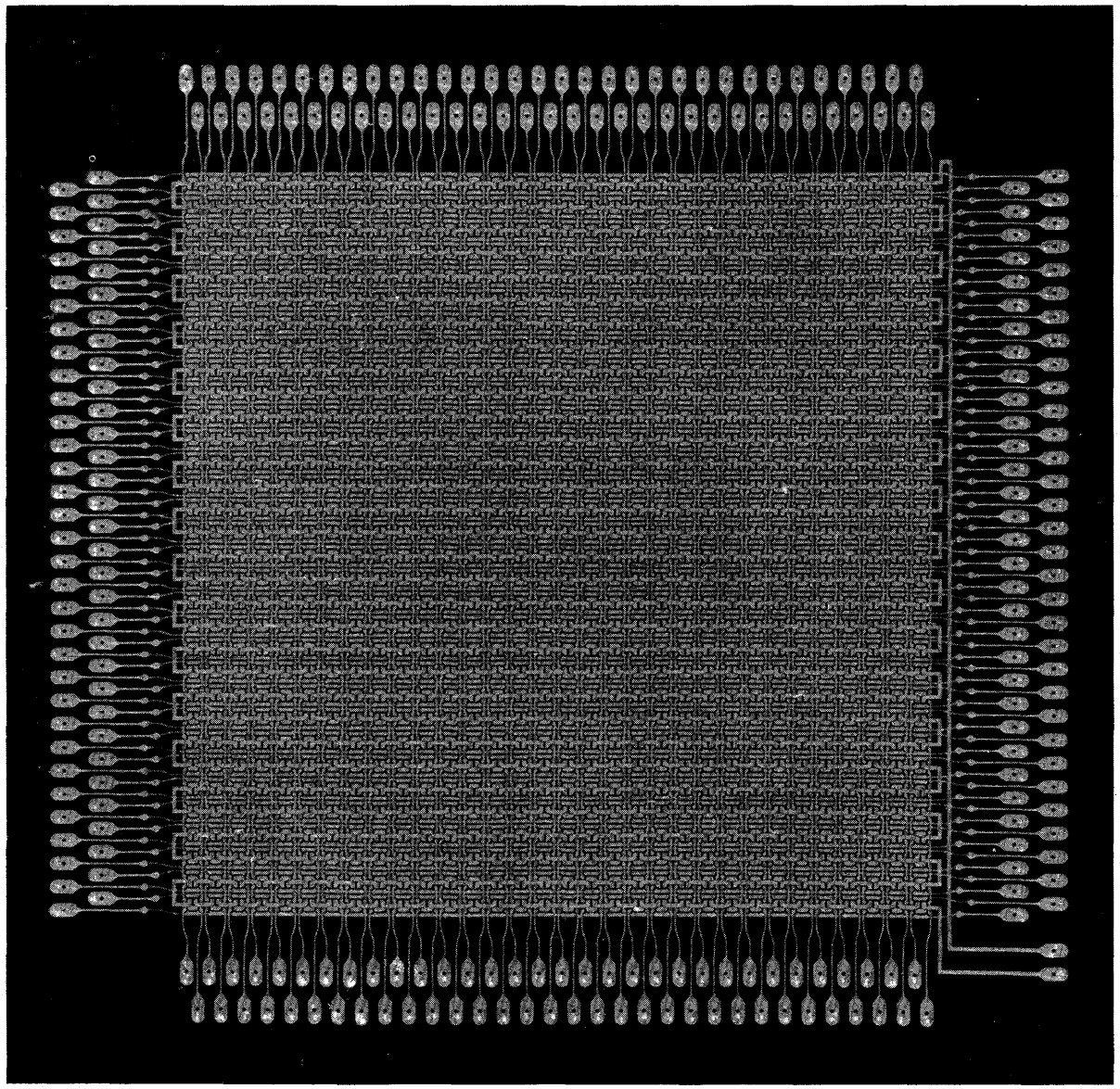


Figure 6. Completed 64 × 64 bit memory plane.

## LOW-POWER LINEAR SELECT OPERATION

The memory planes presently being fabricated are suitable for use either in the linear select mode or the coincident current mode. When operated in the linear select mode, reasonably short cycle times can be realized since the switching constant is less than 0.5 oersted-microseconds. Figure 10 is a switching curve taken using one of the planes and it may be seen that a 120-milliamp, 0.5-microsecond pulse should switch the toroid fully. In Fig. 11d it may be seen that a 2-microsecond read-write cycle time may be readily achieved with read word pulses of 0.5-microsecond at 150 milliamps and a write word pulse of 1.0 microsecond at 67 milliamps with bit current at ±33 milliamps. These currents are less than one-third the values generally required with ferrites at the same cycle time.

At present a system design using these memory planes in a linear select memory of 8,000 30-bit words is under study. Present estimates are (for 2-microsecond cycle time): power — less than 10 watts; volume — approximately a 4-inch cube; cost — 7 to 14 cents per bit depending upon the number of units. Figure 12 shows an artist's sketch of this memory.

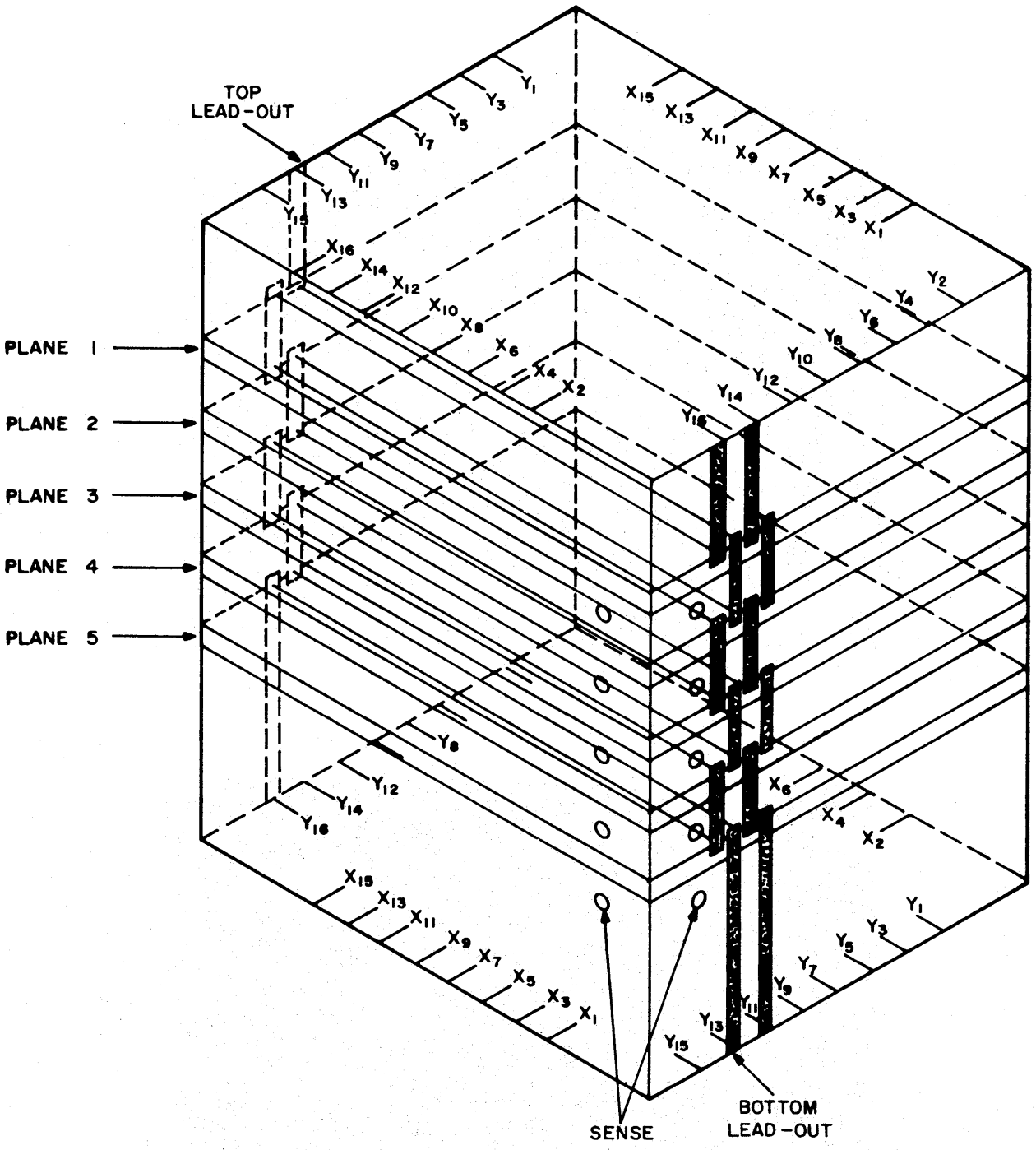



Figure 7. Wiring diagram of completed stack.

## MASS MEMORY

The low cost per bit resulting from the method of batch fabricating memory described, typically 0.01 to 0.02 cents per bit, combined with key system techniques make it possible to build very large memories with a capacity in the order of $10^8$ bits at a cost in the order of 0.1 cents per bit including electronics.

The present mass memory system approach is based on the use of coincident-current memory organization, since for a large memory the drive and selection electronic circuits are many fewer for a coincident-current memory compared with those in a

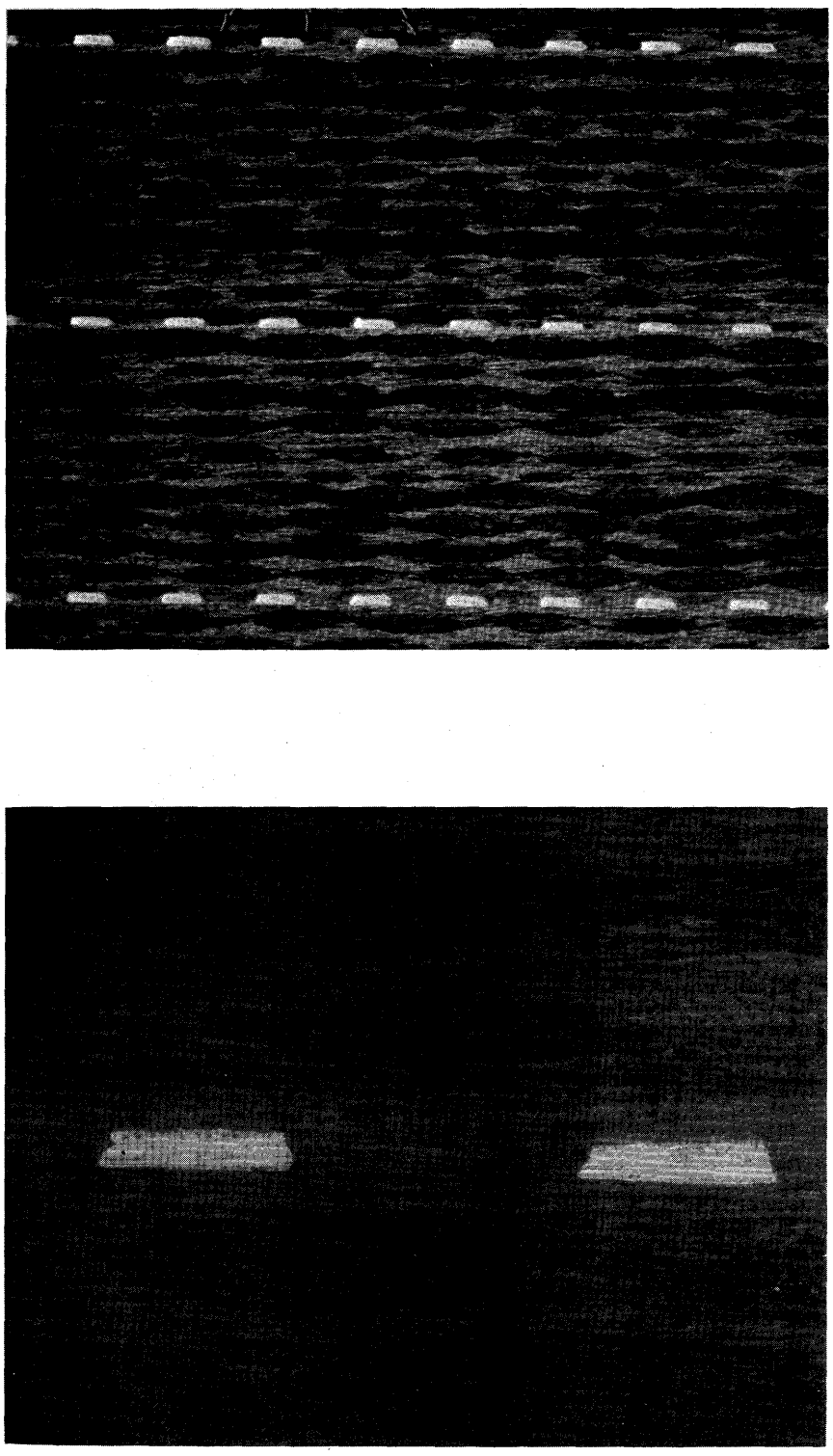


Figure 8. Magnified views of end cross sections at 20× (top) and 100× (bottom).

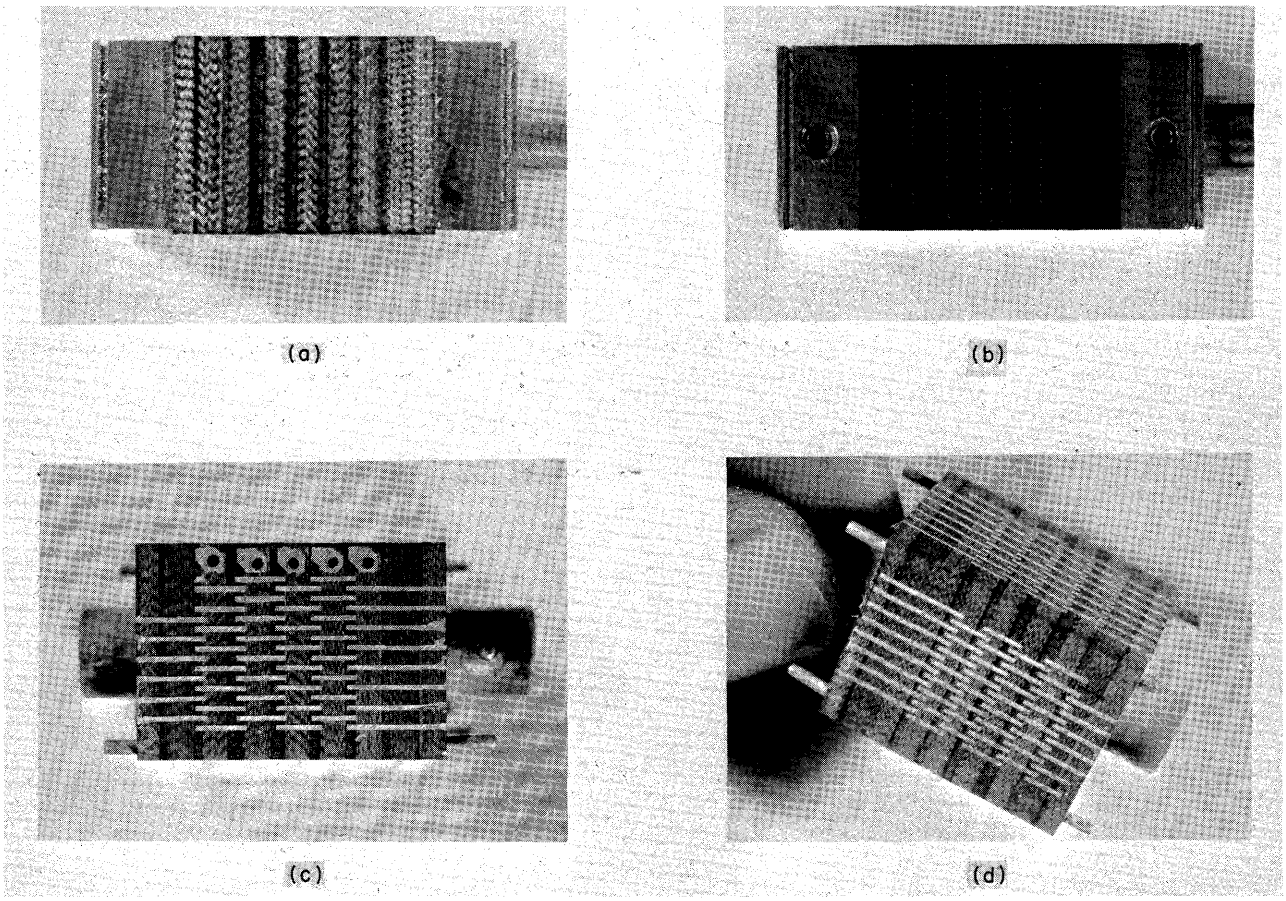



Figure 9. Views of five 16 × 16 planes stacked with batch fabricated plane to plane interconnections.

linear-select memory. It is next considered important in a low-cost memory that the matrix plane size be as large as possible, e.g., at least 256×256, since drive, selection and sense costs are again reduced as a result. Matrix planes in coincident-current memories are normally restricted to much smaller sizes, e.g., 64×64, and even then the planes may require partitioning of the sense line, and the use of additional sense amplifiers, to reduce delta noise. To eliminate delta noise and make large planes possible, an unconventional two-frequency c.w. selection scheme[1] was adopted for the reading operation, while conventional coincident-current writing is employed. Small toroidal storage elements are used to reduce the length and losses of drive and sense lines. Closed flux storage memory elements allow close coupling to wiring and result in low sensitivity to external fields. Sense signals are relatively small, but the sinusoidal nature of sense signals provides system versatility by permitting narrow-band filtering prior to sensing for acceptable signal-to-noise ratio. The reading method is nondestructive, which enhances reliability in large memories, and which reduces access time in the frequent case where read accesses significantly outnumber write accesses. The reading signals are sinusoidal while the write signals are d-c pulses; this difference provides a degree of electrical isolation between reading and writing operations. In Fig. 13 are shown the characteristic *S*-curves (set pulse amplitude as the variable) traced from all 256 toroids on one 16×16 plane read in this manner. Outputs from 64×64 planes are similar. The majority of experimental fabrication work has been done using permalloy toroids that are 0.025 inch on centers. Small toroid size and low coercive force (0.2 oersteds for some permalloy) make possible a low, 30 milliampere, half-select write current for 0.025-inch toroids, read drive currents are approximately the same peak magnitude. Memories using these toroids are potentially low-power ones, and the memory fabri-

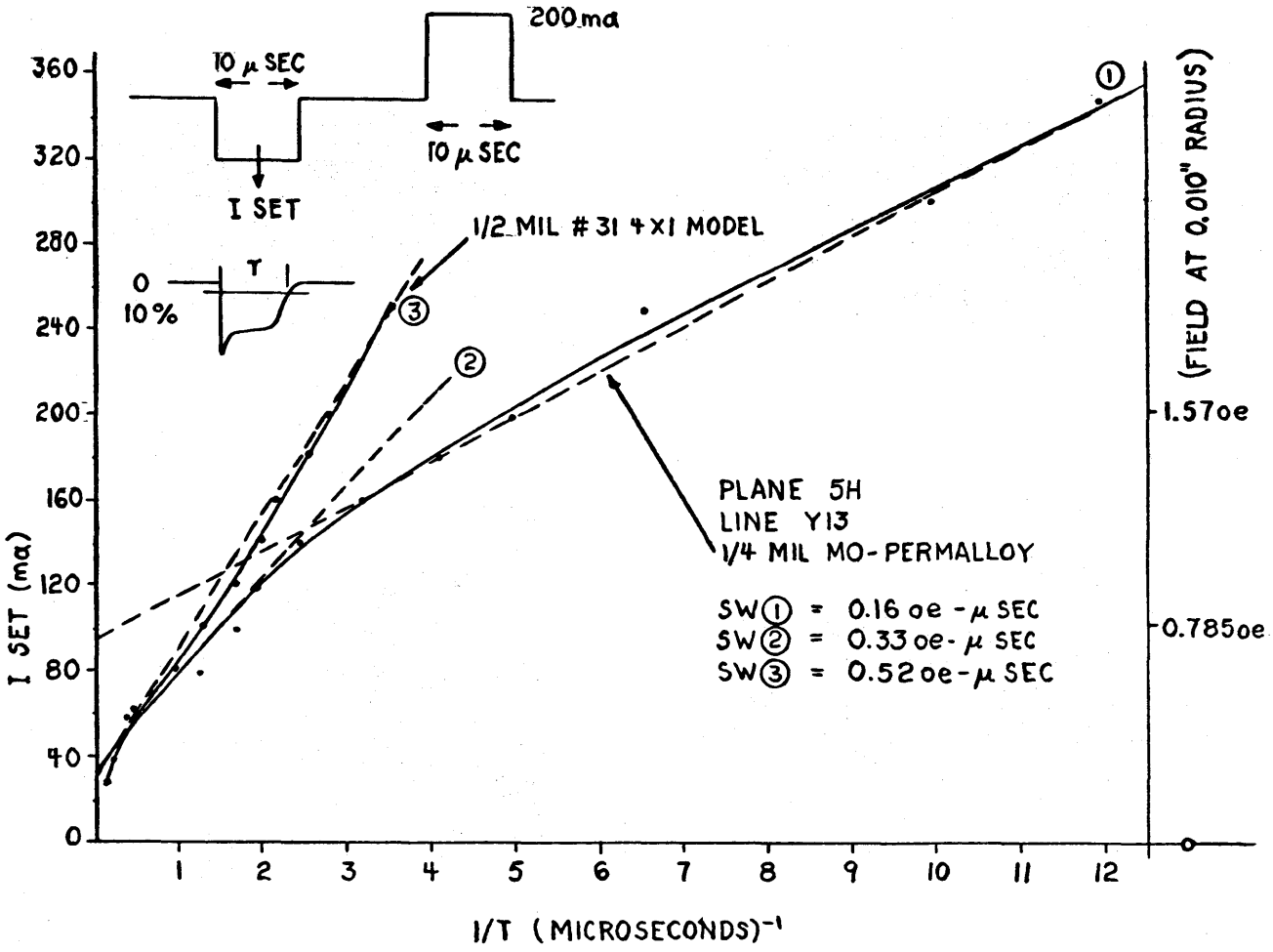



Figure 10. Switching curve for mo-permalloy.

cation technique can be compatible, therefore, with present capabilities of integrated semiconductor microcircuits.

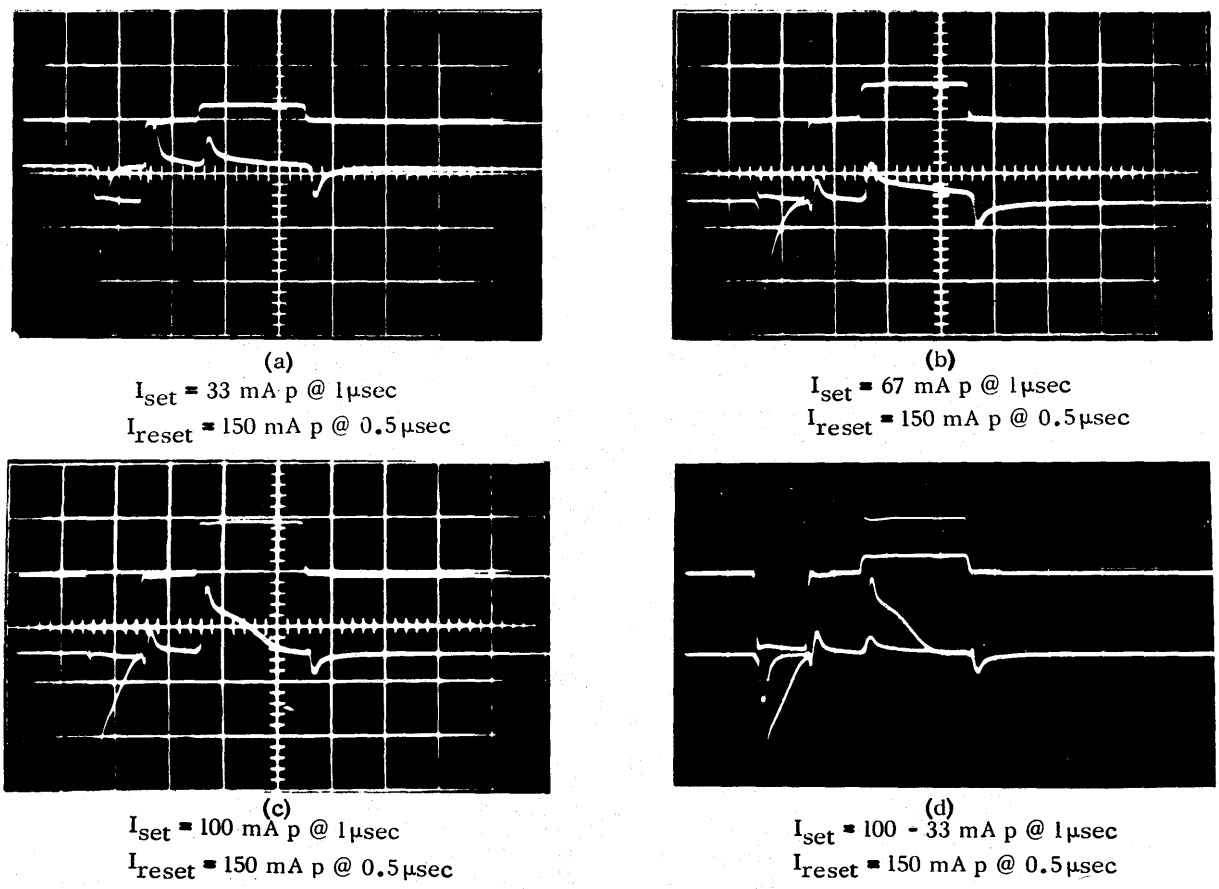
Figure 14 shows a simplified block diagram for a $10^8$-bit mass memory. The system consists of 16 modules of $6.5\times10^6$ bits showing common electronics organized as in the CCM. The basic system design has been described previously.[2] A model of this memory is shown in Fig. 15. The model has a volume of 4.85 cubic feet.

## FABRICATION FACILITY

It is apparent that yield is a critical factor in any batch fabrication process and this is no exception. Present results in a normal laboratory environment are quite good for the $64\times64$ planes now being fabricated, but the next step is to prepare $256\times256$-bit ($65\times10^3$ bits total) planes. In order to assure good yield the total fabrication process has been developed carefully, including the provision for a clean production area. A series of 10 clean work stations have been designed and are now being installed to implement the process on a pilot plant production basis. Figure 16 shows two of the clean stations which are designed for class 100 conditions. The fabrication facility includes in contiguous areas the pilot plant, photo master preparation and electrical test areas.

## ACKNOWLEDGMENTS

The authors would like to acknowledge the assistance of the staff of the Solid State Electronics Laboratory at Laboratory For Electronics, Inc., for the work reported here.



(a)
$I_{set}$ = 33 mA p @ 1 μsec
$I_{reset}$ = 150 mA p @ 0.5 μsec

(b)
$I_{set}$ = 67 mA p @ 1 μsec
$I_{reset}$ = 150 mA p @ 0.5 μsec

(c)
$I_{set}$ = 100 mA p @ 1 μsec
$I_{reset}$ = 150 mA p @ 0.5 μsec

(d)
$I_{set}$ = 100 - 33 mA p @ 1 μsec
$I_{reset}$ = 150 mA p @ 0.5 μsec

Figure 11. Test of write operation of ¼-mil mo-permalloy. Plane 5H line Y-13. Vertical voltage sides are 10 mv/cm for (b), (c) and (d), and 5 mv/cm for (a). (a): Readout from 1/3 selected elements (0). (b): Readout from 2/3 selected elements. (c): Readout from selected elements (1). (d): Combined 1-0 read write signals. *Note:* $I_{set}$ corresponds to the write signal while $I_{reset}$ corresponds to the read signal.

## REFERENCES

1. B. A. Widrow, "A Radio-Frequency Non-destructive Readout for Magnetic Core Memories," *Trans. IRE, PGEC,* vol. EC-3, p. 12 (Dec. 1954).

2. H. W. Fuller, T. L. McCormack and C. P. Battarel, Paper 5.5, *Proc. of the 1964 Intermag. Conf.,* IEEE Inc., New York.

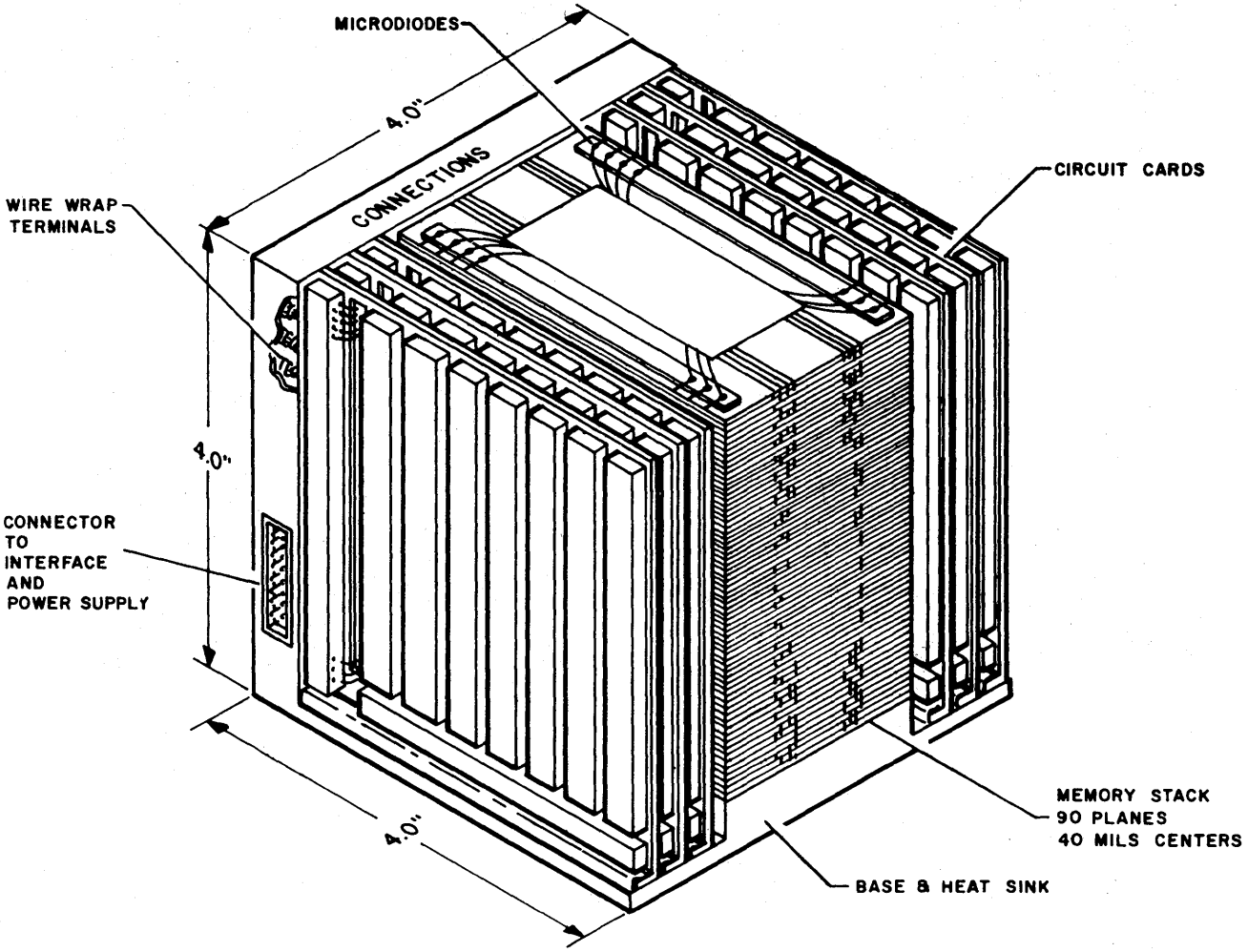



Figure 12. Proposed low-power memory packaging layout, 8,010 words, 30 bits per word.

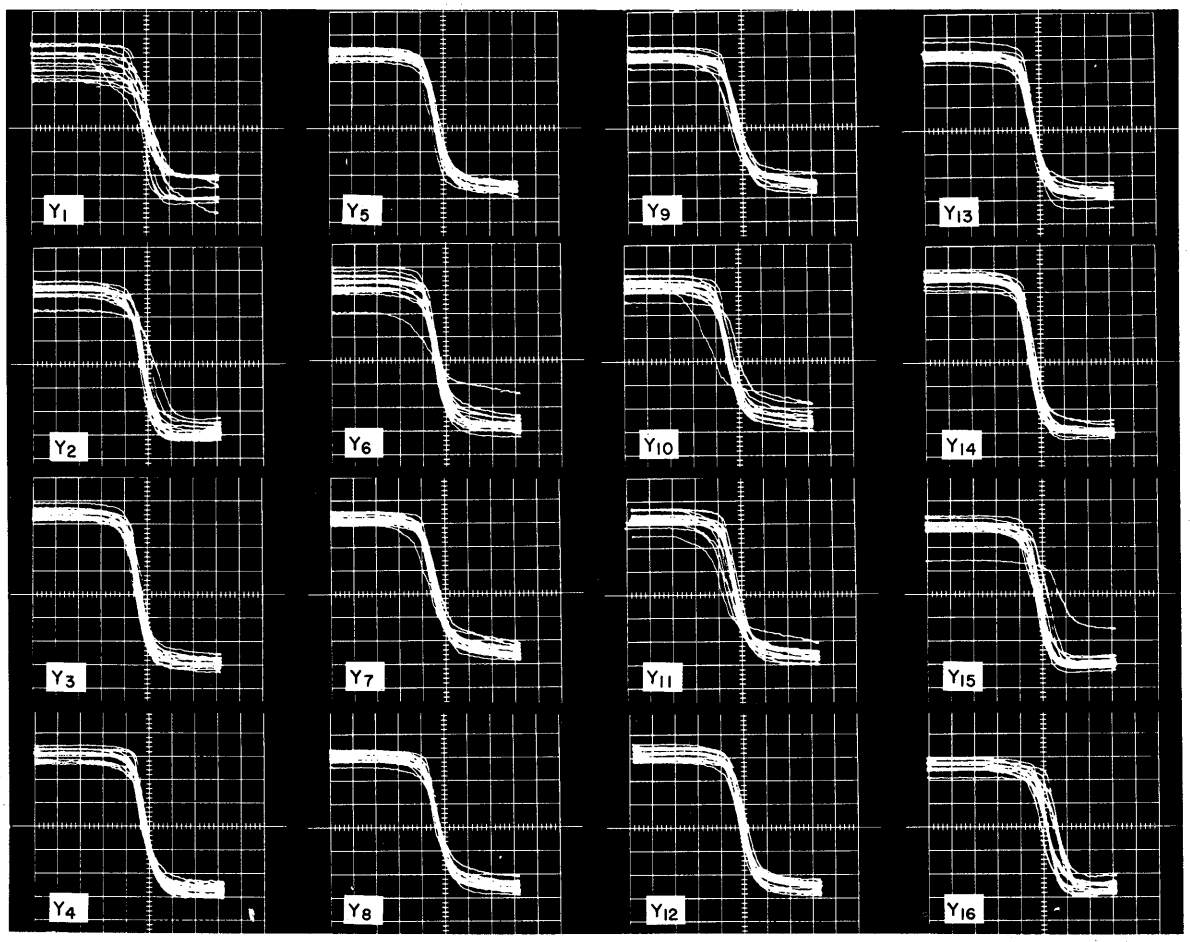



Figure 13. *S* curves for all 256 elements on plane 2N. Reset pulse –80 ma peak, 5 μsec wide; set pulse varied 0 to +80 ma peak, 5 μsec wide. Read currents 40 ma pp. Vertical scale 10 μv/main div. Horizontal scale 10 ma/major div. Each photo shows 16 *S* curves associated with one Y line. Material is No. 43 — ½ mil.

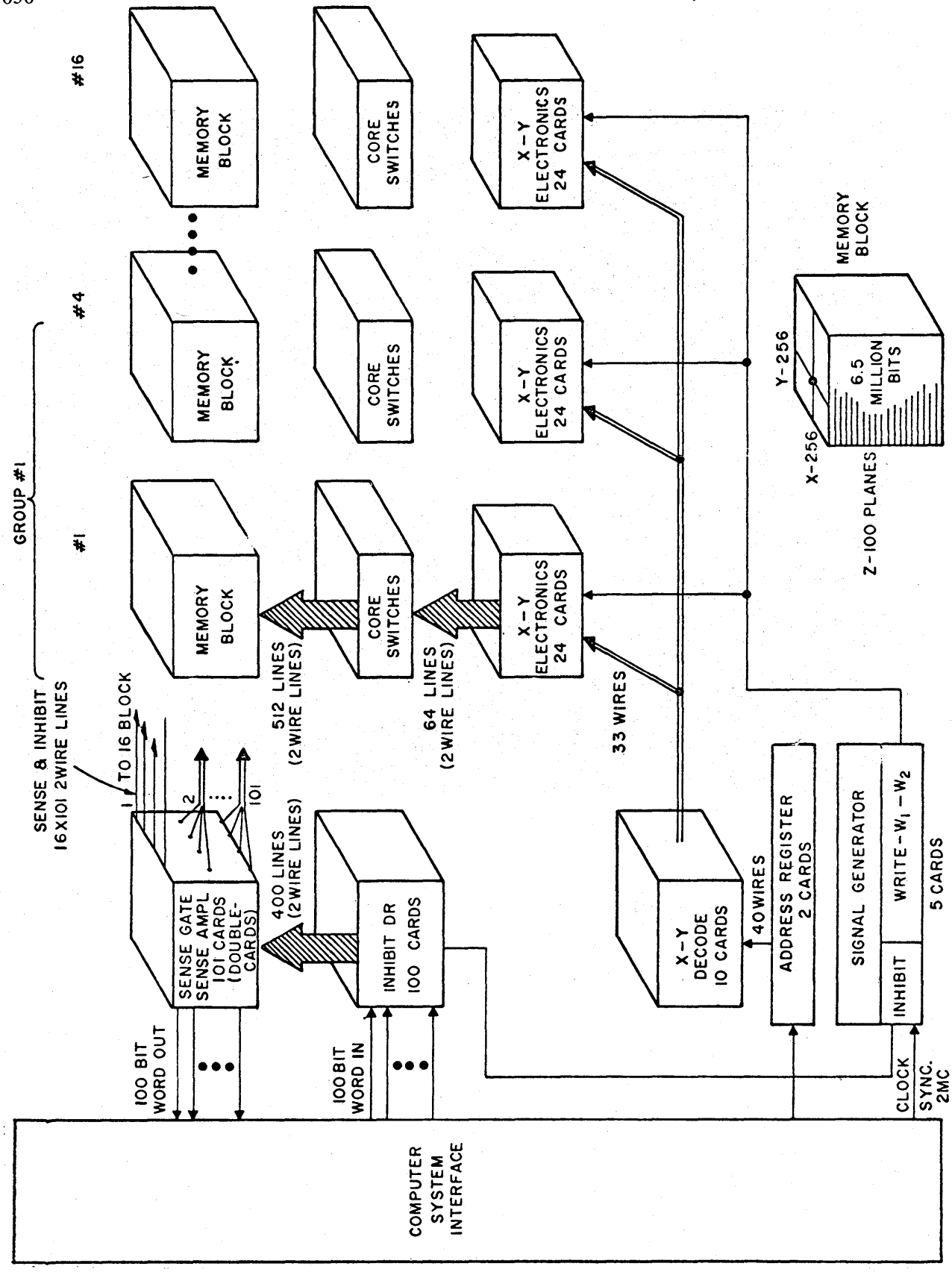



Figure 14. Mass memory block diagram.

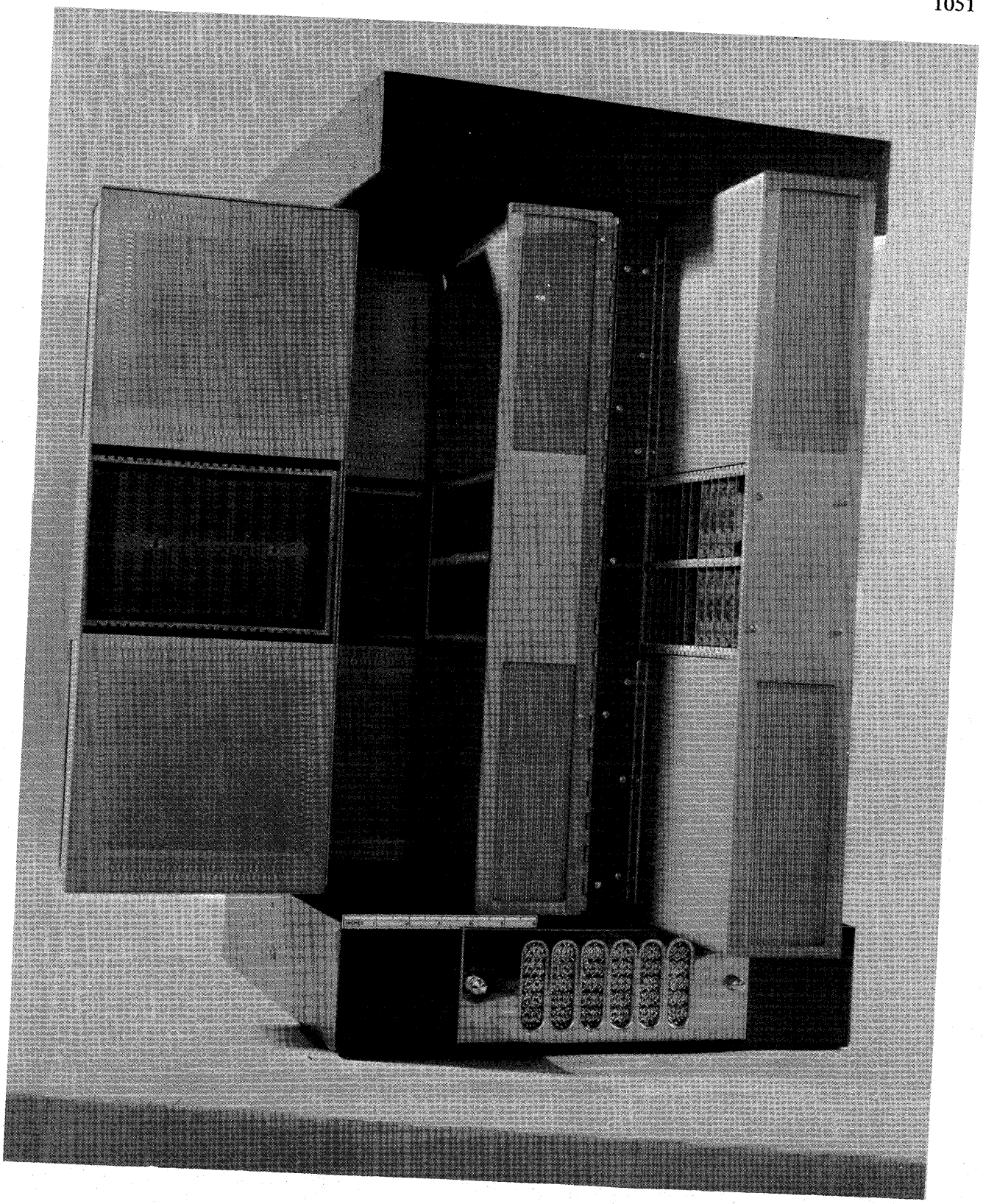


Figure 15. Model of $10^8$-bit memory. Outside dimensions are 17 × 17 × 29 inches.

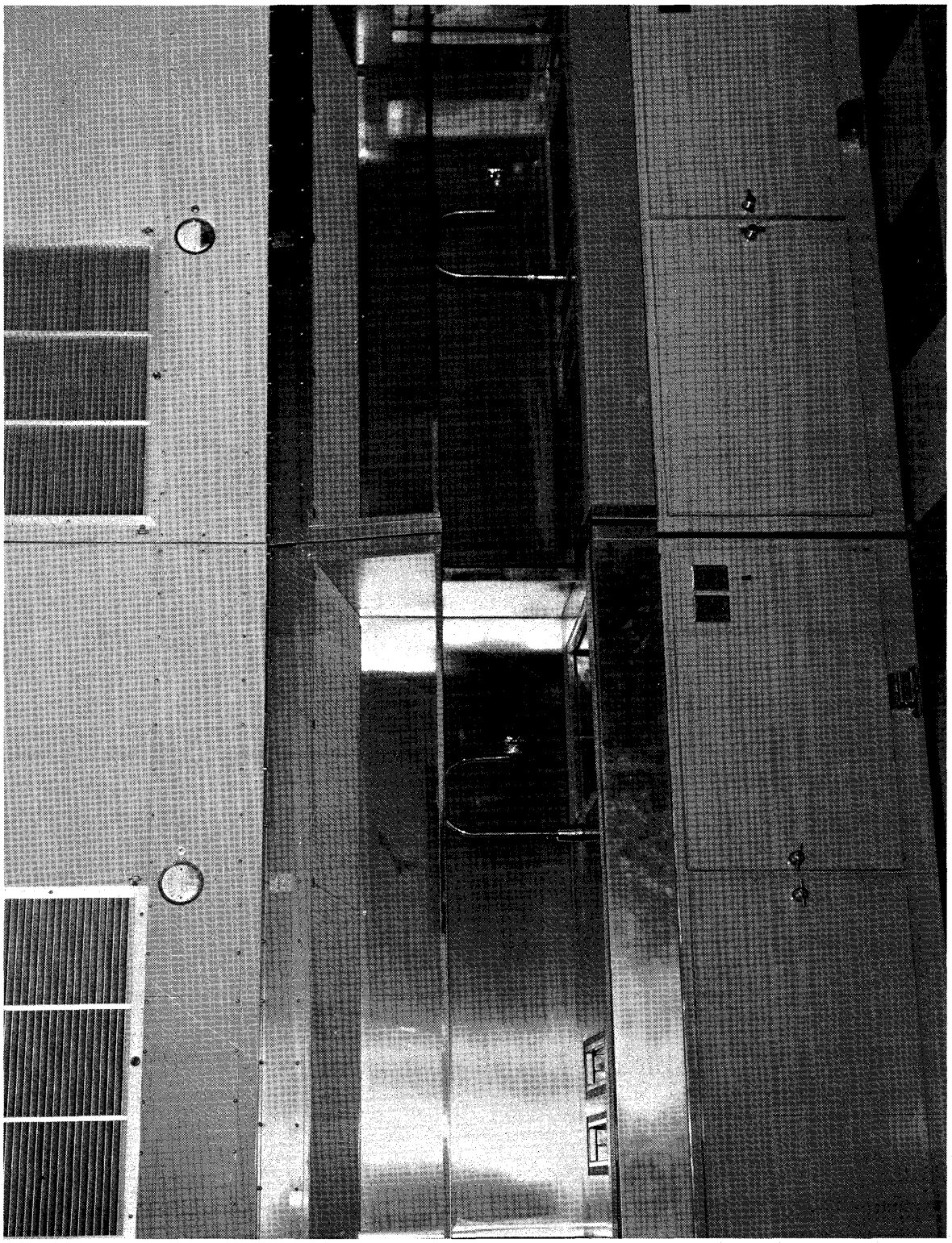


Figure 16. Clean work stations used for batch fabricated memory plane manufacture.

# AN INTEGRATED SEMICONDUCTOR MEMORY SYSTEM

H. A. Perkins and J. D. Schmidt
*Fairchild Semiconductor Research and Development Laboratory*
*Fairchild Camera and Instrument Corporation*
*Palo Alto, California*

## INTRODUCTION

The concept of active circuit data storage (flip-flop) is as old as electronic data processing systems. The attributes of highest access speed, steady state nondestructive readout and flexibility of application have been partially offset by higher costs and higher standby power per storage bit. As a result, flip-flop storage has until recently been really only feasible for registers.

As integrated circuits have been perfected to provide a multiplicity of gates or flip-flops on each monolithic die at lower costs than traditional discrete component circuits, the always interesting possibility of an integrated all-semiconductor memory of reasonable capacity becomes exciting. Simply integrating the circuits is not enough. A maximum number of storage positions in a given device package should also require a minimum of leads. Since a memory system is the objective, the need for the usual logic level compatible interface is waived. The storage device is optimized for minimum complexity per bit of storage and greatest electrical tolerance allowance at its terminals. Interface (peripheral) circuits are required such as word bit drivers and sense amplifiers much as for magnetic core or film memory systems. However, the performance required from the peripheral circuits is much less stringent for the semiconductor memory described.

The following sections will describe the circuits, devices, packaging and system design for a random access 256-word memory of 72 bits per word.

## SYSTEM DESCRIPTION

### System Design Goals

The main goal to be achieved is a memory system competitive in cost and superior in performance for a certain range of applications. To achieve the goal, the storage device's complexity must not push processing technology toward low yields, must use fairly standard, easily installed packages and require a minimum number of packages for the system complement. To meet these needs, a 16-lead dual in-line package (similar to Fairchild CT$\mu$L logic family) containing 4 words of 9 bits per word (36 bits) was selected. An 8x10-inch double-sided printed circuit card readily holds 160 packages. Of this number, 128 are arranged in an array of 16x8 to provide 64 words of 72 bits each, with the balance of the packages containing word drivers

at 2 circuits per package. Thus the 256-word memory requires 4 such cards (storage). In addition, 2 data circuit cards at 36 bits per card (bit driver and sense amplifier) and a single address register and control card are needed to complete the card complement. A power density of approximately 5 milliwatts per bit was chosen with a view to providing high speed (150-nanosecond cycle time) but moderate system power required (130 watts). The basic storage cell has wide application flexibility and the specifications which follow are one compromise to several conflicting needs. Minor modifications to the design permit optimization for higher speeds, larger capacity for one set of peripheral circuits or reduced power. The capacity of the storage device itself (36-bit package) is a function of present device and package technology and should be regarded as a first expedient.

Table 1.

| | |
|---|---|
| Capacity | 256 words, 72 bits/word, 18,432 bits total |
| Repetitive access or write cycle time | 150 nanoseconds |
| Read mode | nondestructive |
| Write mode | jam set to one or zero (at storage device) |
| Read access time | 120 nanoseconds |
| Data flow rate | 480 bits/microsecond |
| Interface signal levels | + 2 volts = one, –0.5 volt = zero (compatible with Fairchild CT$\mu$L family) |

## System Organization

The functional components of the semi-conductor memory are similar to most random access memories as shown in the block diagram of Fig. 1. A nondestructive steady state output is obtained from the storage cell. Strictly speaking, buffer registers are unnecessary. In the memory described, registers

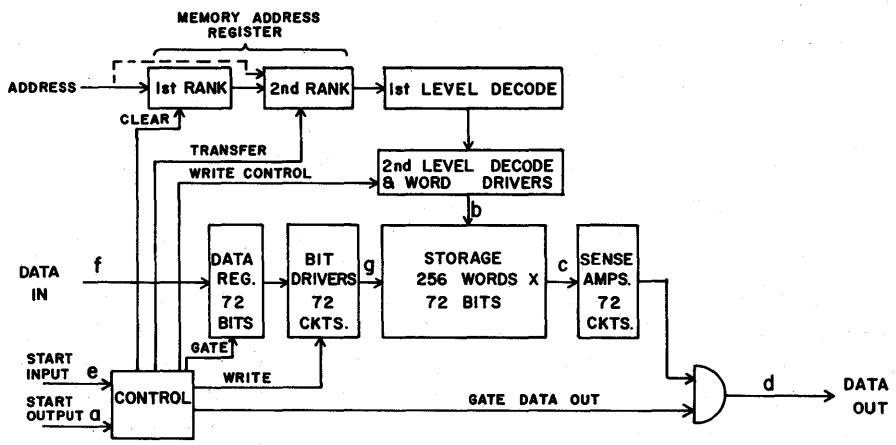



Figure 1. Block diagram.

have been included primarily as a convenience for writing in data so that the computer need not be tied up any longer than necessary. Instead of having an output level directly compatible with logic level signals, a relatively low 30-millivolt signal is sensed as a one. Although a sense amplifier is required, the advantage of relatively low-impedance sense/bit line (50 to 150 ohms) justifies its use. Such an impedance level is compatible with interconnection techniques such as printed wiring, twisted pair and coaxial cable. Yet only about 400 microamperes are required from the storage cell tending to minimize standby power needs for high speed capability.

Logic elements for the memory are standard complementary transistor micrologic units. The memory address register is formed from Dual Rank Flip-Flop circuits (Fairchild CT$\mu$L-957). The first level decoder uses dual, 4-input positive "and" gate (Fairchild CT$\mu$L-954) followed by inverters (Fairchild CT$\mu$L-952) to provide the correct signals for the second-level decoders and Word Drivers. On the first memory cycle after the memory has been idle, the address with a start command is loaded into both ranks and decoding begins immediately. After the correct word has been selected and delay through the sense amplifier completed, a gating signal transfers the output to the computer system. If an input command (write-in) is presented to the memory, data is gated into the Data Register (a Latch circuit Fairchild CT$\mu$L-968), Bit Drivers corresponding to zeros are energized and after decode delay time a Write Control signal causes the Word Driver to change the state of the storage cells to correspond with Bit Driver outputs.

For repetitive memory cycles, the earlier address is retained in the second rank of the memory address register until the first cycle is complete. The new address, if a start command is also present, is loaded in the first rank only, pending completion of the preceding cycle. Near the end of the first cycle, the new address is transferred to the second rank and, soon after the new cycle begins, the first rank is cleared. By storing the start command also in dual rank register, an asynchronous input can keep the memory cycling at maximum speed.

Special integrated circuits are the 36-Bit Storage Cell, Decoder-Word Driver, the Bit Driver, and the Sense Amplifier. (The Sense Amplifier is a Fairchild $\mu$A-710 Comparator.)

Typical voltage and current levels at the storage array interface are given in Table 2.

Table 2.

| Function | $V$ | $I_{max}$ |
|---|---|---|
| Word drive, read | 1.5–2$v$ | 25 ma |
| Word drive, write | $> 3.0v$ | 55 ma |
| Bit drive | $-0.7v$ | 10 ma |
| Sense output | 50 mv | 0.5 ma |

A timing diagram is shown in Fig. 2. Times correspond to functional blocks in Fig. 1. Response of the storage bit after word drive is applied is about 40 nanoseconds. The time required to store information in a storage bit is about 50 nanoseconds. A

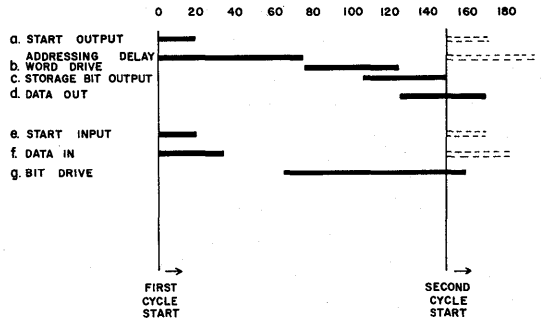



Figure 2. Timing diagram.

150-nanosecond repetitive cycle time is obtained by allowing overlap into the subsequent cycle. The sense amplifier is not limited to 256 bits on a sense line. Using a multiplexed sense amplifier (4 segments of 1024 words) a 4096 word memory of 72 bits per word should easily achieve a repetitive cycle time of 500 nanoseconds. A cycle time of 100 nanoseconds is obtained at 64 words of 72 bits. These timing estimates are projected on the basis of 5 milliwatts per bit for the particular partitioning scheme described. A higher redundancy of peripheral circuits obviously tends toward maximum speed. The other variable is the power density in the storage package. This may be modified readily by changing the applied power supply voltages or by changing the internal resistances within the device either to optimize for speed, cost or power consumption. The most significant point is that basically a single device design in the storage package will serve a large spectrum of application requirements. The goal of competitive costs is achieved mainly by producing a very large volume of essentially a single device with the attendant cost minimization realizable with microcircuitry.

## CIRCUIT DESIGN

### Basic Memory Cell

The particular circuit configuration that one chooses for an integrated semiconductor memory is

dependent on a number of considerations such as chip complexity and size, number of leads, power dissipation, speed, logical flexibility, fabrication capability, etc. After analyzing the complete memory system costs and considering the type of packaging to be used, we concluded that the most important criteria for selection of the circuit configuration is to minimize the number of leads required to the memory cell. This approach permits achieving the highest functional complexity possible for any given package chosen. Consistent with the above statements is the belief that the memory cell circuit should be as simple as possible even at the expense of more complex peripheral circuits if required.

After investigating both linear selection and coincident selection circuits as well as separate data in and data out buses, we decided on a linear selection cell with a common data in–data out bus as shown in Fig. 3. The circuit is a conventional emitter-coupled binary except that the right-hand

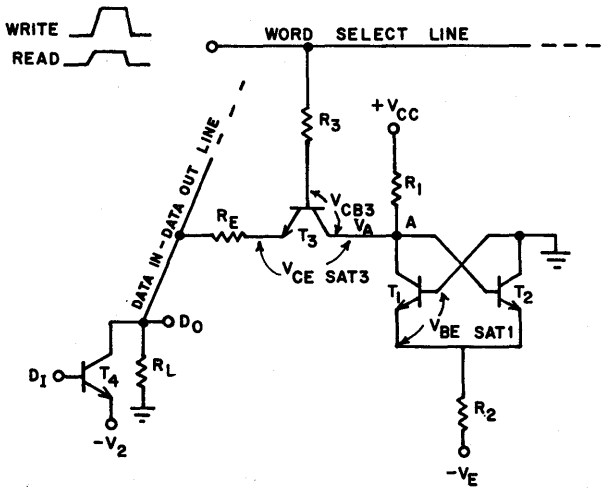



Figure 3. Basic memory cell.

load resistor has been omitted and this node tied to ground. This forces the left-hand collector output node (point A) to vary from –0.7 volt (defined as a zero) to +0.7 volt (one state) under quiescent conditions. These voltages are determined only by the transistor parameters and are as follows:

$$V_{\text{one}} = +V_{be}\,sat - V_{ce}\,sat \text{ and}$$
$$V_{\text{zero}} = -V_{be}\,sat + V_{ce}\,sat$$

Transistor $T_3$ acts as an input/output switch and is driven by a bi-amplitude positive pulse as shown on the word select line. Consider first the read operation. A lower amplitude positive pulse forward biases the $T_3$ base to emitter turning the transistor on. If a one is stored (+0.7 at A), $T_3$ acts as a normal transistor and a *positive* output appears across $R_L$. Its amplitude is determined by the ratio of $R_L$ to $R_1 + R_E$ and the magnitude of $+V_{cc}\,V_{ce}sat$. The purpose of $R_E$ is to prevent point $A$ from going closed to ground that +200 mv even with $D_0$ shorted to ground and minimum $V_{ce}sat$ on $T_3$.

If a zero is stored, $T_3$ acts as an inverse transistor and a slight current (dependent on inverse beta) flows into node A, tending to produce a *negative* output signal. Thus, a zero or a negative level at $D_0$ corresponds to a zero and a positive level corresponds to a one. When reading a zero, the base collector junction of $T_3$ is forward biased forcing base current into node A. In order not to write a one into the cell, this current must be limited and this in turn is the reason for the bi-amplitude word select pulse. To write a one, transistor $T_4$ is left off and the higher amplitude word select pulse is applied. This forces sufficient current into node A to cause $T_1$ to turn off and $T_2$ to turn on, thus writing in a one. To write a zero, $T_4$ is saturated, pulling the data in–data out line negative and thus pulling node A negative when $T_3$ is saturated.

In designing the circuit, three primary constraints must be met to satisfy the conditions that the cell performs correctly at d-c or low frequency. These design constraints are:

1. Minimum write current is sufficient to write a one.
2. Maximum read current *does not* write in a one if the F/F is in the zero state.
3. Minimum read current is sufficient to saturate the gate transistor ($T_3$).

In considering the design problem, it is soon realized that there are many more degrees of freedom in the design than there are constraints on design conditions which must be met. Thus one must use many secondary design considerations and some judgment in order to arrive at a unique solution. Our approach was to consider first the word driver circuit and determine roughly what read and write voltage levels and spreads could be realized. From this preliminary work, all terminal voltages were specified for the cell (i.e., word line and data line voltages for both reading and writing). The next step is to define the power supply voltages and the approximate power level of the cell. From the standpoint

of wide tolerances in the cell, it is desirable to have $+V_{cc}$ and $-V_E$ both large in magnitude compared to $V_{BE}$, and $R_1$ and $R_2$ large so that the node A and the emitter node are fed from a constant current source. However, this increases both the power dissipation and the size of the circuit in addition to yielding a somewhat slower circuit due to the RC time constant at node A. Consideration of these tradeoffs lead to fixing $+V_{cc}$, $-V_E$ and $R_2$.

The three worst case equations for the three primary constraints are:

$$1.\quad V_A = \left[\frac{\overline{R_1}\,\overline{R_3}}{\overline{R_1}+\overline{R_3}}\right]\left[\frac{\overline{V_W} - \overline{V_{cb3}}}{\overline{R_3}} + \frac{\overline{V_{cc}}}{\overline{R_1}} - \frac{(\overline{V_E} - \underline{V_{be}\,sat_1})}{\overline{R_2}}\right]$$

(Use high temperature values for transistor parameters)

$$2.\quad V_A = \left[\frac{R_1 R_3}{R_1+R_3}\right]\left[\frac{\overline{V_R} - \underline{V_{cb3}}}{R_3} + \frac{\overline{V_{cc}}}{R_1} - \frac{(V_E - V_{be}\,sat_1)}{\underline{R_2}}\right]$$

(Use low temperature values for transistor parameters)

$$3.\quad V_A = (R_E+R_L)\left[\frac{V_R - \overline{V}_{be}\,sat_1}{\overline{R_3}} + \frac{\underline{V_{cc}}}{\overline{R_1}}\right] + V_{ce}\,sat_3$$

where the bar above a parameter indicates the maximum value and a bar below indicates the minimum value. $V_R$ is the "read" voltage on the word select line and $V_W$ is the "write" voltage at the same point. All other notations are defined in Fig. 3.

The following values were used in the design equations:

| | |
|---|---|
| Resistor tolerance | $\pm$ 30% |
| Power supply tolerance | $\pm$ 5% |
| "Read" word select voltage tolerance | $\pm$ 0.25 volt |
| Minimum difference between "read" and "write" word select voltage | 1 volt |
| $V_A$ (write 1) | = 0 volt |
| $V_A$ (read 0) | = –200 mv |
| $V_A$ (read 1) | = +200 mv |

The voltage levels at node A were chosen so that there would be no change in the state of the cell during reading. By using the above values in Eqs. (1) and (2), $R_1$ and $R_3$ can be uniquely determined in terms of $R_2$ which was previously chosen. The values of $R_1$ and $R_3$ along with $R_L$ were then used in Eq. (3) and $R_E$ was determined. The analytical results were verified by breadboarding *all* worst configurations using kit-integrated parts and determining the points at which failures occurred.

The decision was made to package the memory circuits in Fairchild's new dual in-line configuration having 14 or 16 leads available. Since the memory cell requires . . . . power supply contacts ($+V_{cc}$,$-V_E$, and ground), 13 signal leads are available with the 16 pin package. These 13 pins can be used in a near optimum fashion by organizing the memory chip as a 4-word array of 9 bits per word. This gives 36 bits of storage with 4 word select lines and 9 data in–data out lines required. The chip size is 60 by 80 mils and utilizes two layers of metal to solve the crossover problem. Fig. 4 is a photomicrograph of the 36 bit memory array.

## Peripheral Circuits

*Final Decoder and Word Driver.* In order to minimize the number of gates required for word decoding, the decoding has been broken down into *two* levels called a first decoder level and final decoder. The final decoder is a two-input "and" gate whose output ties to the word driver circuit. Thus for a memory of $2^n$ words, there are $2^n$ of these circuits required. The first level decoding is performed by $2\times2^{n/2}$ "and" gates with n/2 inputs each. For n=8, there are 256 final decoder and driver circuits and 32-4 input "and" gates for the first level decoder. The fan out required of the first level decoder is simply $2^{n/2}$ with this decoding scheme.

The final decoder and word driver circuit is shown in Fig. 5. It consists of a current mode gate providing a "negative and" function with the non-inverting output directly coupled to a common emitter inverter stage which in turn drives an emitter follower output stage. To make the input levels compatible with the first decoder output, the reference transistor in the C.M. stage ($T_3$) is tied one diode drop above the group. This also gives some temperature compensation to the circuit. The input resistors are used to suppress any tendency of the circuit to oscillate. The bi-amplitude output is achieved by controlling the voltage on the write control bus (WCB). If this line is tied to ground, a voltage divider is created and the lower amplitude output (read) results. If the bus is opened, the high-

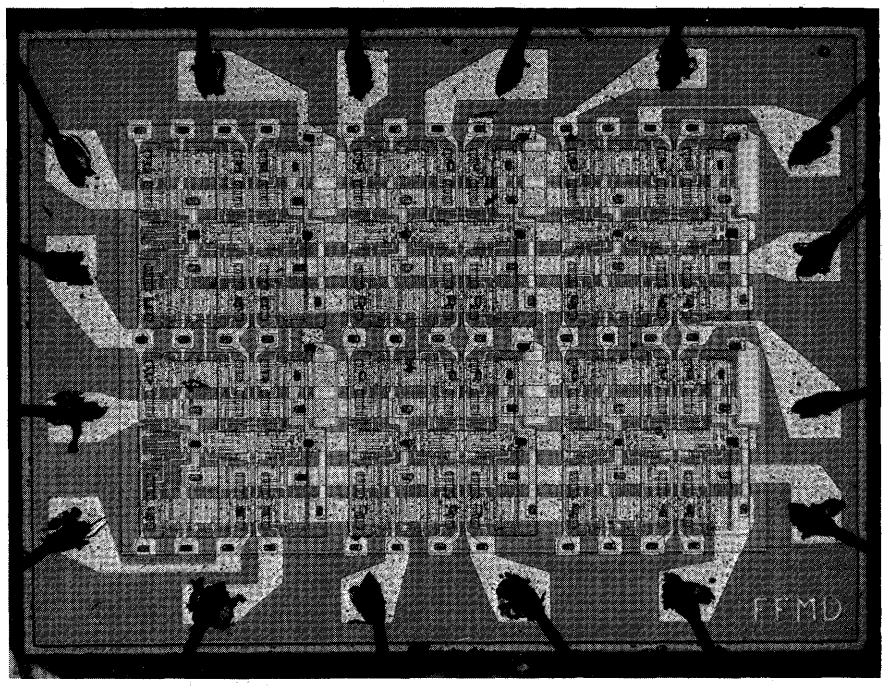


Figure 4. 36 Bit memory circuit (60 x 80 mils).

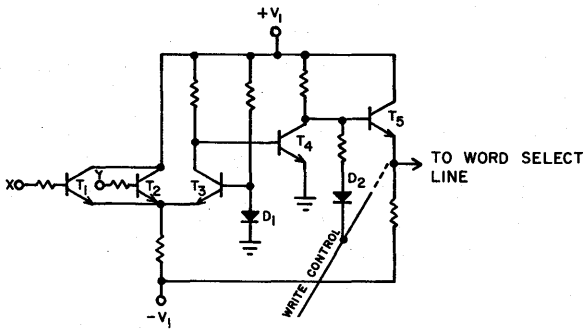



Figure 5. Final decoder and word driver.

er amplitude output (write) results. Again the purpose of diode $D_2$ is to provide temperature compensation. The output stage is designed to drive a 72 bit or less length word which requires about 50 ma current for writing.

Again the circuit was worst-case designed using the same tolerances as previously listed for the basic memory cell. In addition, the output transistor was designed to withstand momentary shorting to the minus supply voltage. The circuit was designed to operate within the correct levels at a junction temperature of 110°C and was actually tested at this ambient temperature with no special heat sinks on the transistors.

In order to keep the integrated circuit relatively simple, only two such circuits were put on one chip and single layer metal was used. The chip is 45 mils on a side, has a nominal power dissipation of 250 mw and is shown in Fig. 6. The chip is packaged in the 14 lead dual in-line header and two such packages are required to service the four words contained in the memory package.

*Data Circuits.* The sensing problem with this memory cell is strictly a compromise solution based on system size, speed and complexity. Basically, the output of the memory cell during reading is a given current (about 0.5 ma) and this current may be driven into any impedance from zero on up as long as it is referenced to ground. Thus an ideal method of sending would be to use a common base stage which would terminate the line in a low impedance and provide voltage gain with good bandwidth. Unfortunately, the polarity is such that a PNP transistor is required and good PNP common base stages are not easily integrated. For this reason, and also to limit the number of special integrated circuits that have to be designed, we decided to build the prototype system using a convential integrated amplifier, the $\mu$A710. All 256 words in the memory system are tied to a common data line. This can be done since the leakage current of the gating transistor in the memory cell and its emitter capacitance are both very low. The data lines are then termi-

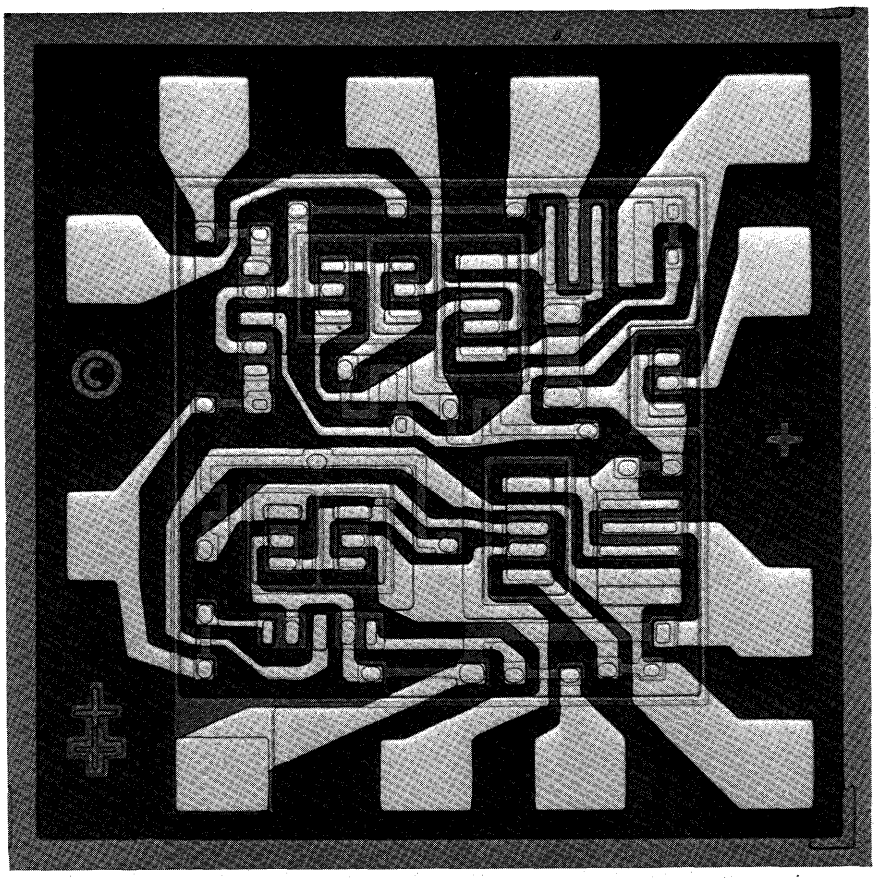


Figure 6. Dual final decoder and word driver (45 mils square).

nated in a 150Ω resistor and the 0.5 ma sensing current produces about 75 mv output signal nominally. This signal is amplified to the logic level by the $\mu$A710 and a CT$\mu$L-956 buffer is used as an output gate and line driver as shown in Fig. 7.

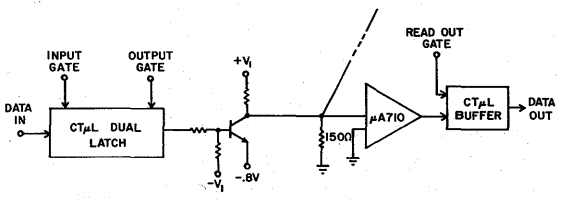



Figure 7. Data circuits.

A CT$\mu$L dual latch element is used as a data input register. This circuit is simply a flip-flop with single rail input and output and a gate on both the input and output. Its output cannot pull negative enough on the data line to write in a zero and thus a single common emitter transistor tied to –0.8 volt is required on the output of the latch. Saturating this transistor when a "write" pulse is present on the word select line will write in a zero and, if the transistor is turned off, the data line will be referenced to ground and a one will be written in.

In the preliminary design, data line recovery time was recognized as an important system parameter. The capacity associated with the line was estimated as 0.2 pf per emitter times 256 plus about 20 pf wiring for a total of 70 pf. Thus the RC time constant would be about 10 nanoseconds with a 150 ohm termination. This appears to be a reasonable number.

With regard to packaging, all circuits are housed into dual in-line packages with two latches, two data drivers, two $\mu$A710's and two buffers per package. In addition, a special interface circuit will be provided so that the memory will be compatible with any logic levels commonly used. Thus it require five of the dual in-line packages to service two bits of data.

## PACKAGE DESIGN

### Printed Circuit Boards

All the circuits are packaged in the dual in-line 14 or 16 lead configuration which, in turn, are inserted and flow-soldered in a two-sided printed circuit board. Fig. 8 is a photograph of the storage board which is 8″ x 10″. The boards are standard 1/16″ G-10 material with 1 oz. copper. The plated through holes are 30 mils in diameter with a 40 mil land. The copper runs are 20 mils wide with a minimum clearance of 20 mils. The power supply voltages and ground are interconnected on the two sides of the board to form an interleaved grid pattern. This helps to reduce the self inductive and resistive drops in the lines.

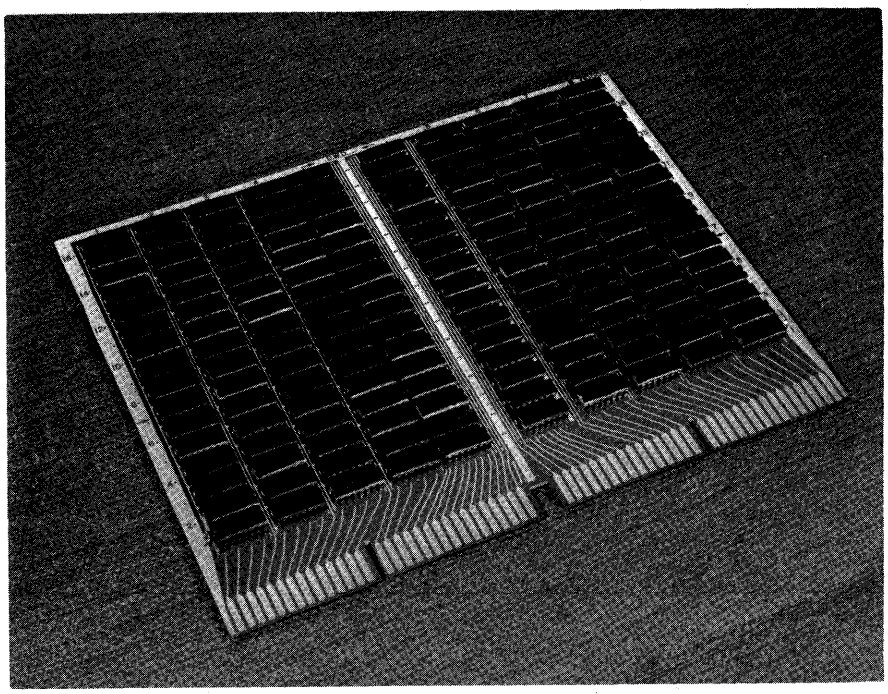


Figure 8. Storage printed circuit board (64-72 bit word).

### Interconnection Techniques

The 7 pc cards are interconnected by means of a two sided mother board which also functions as the base of the specially developed connector. Fig. 9 shows a photograph of the top side of the mother board with some of the contacts installed. The signal interconnection pattern between boards is contained on this side of the board. This is also the side into which the contacts and pc boards are inserted. The other side of the mother board contains power supply busing and solder contacts.

Fig. 10 shows a photograph of the connector assembly. It consists principally of three parts: the mother board, the contacts, and a set of cams. When the cams are in the relaxed position, the pc cards may be freely inserted or withdrawn since they do not touch the contact fingers on the board. After a board is inserted the cans are rotated, causing the contacts to exert a wiping action on the fingers and thus make contact. The contact pressure is designed to wipe through any oxide films that may be built up on the pc card.

The same mother board and connector is designed to be used as the equipment interface connector in order to minimize hand wiring required in the memory system. Fig. 11 shows a photograph of the interface connector card with sub-miniature coaxial cable connected to it. Two such cards are required to complete the approximately 160 connections to the system. The cables are secured with a hood and clamp piece that feeds them out the back end of the card cage in a bundle.

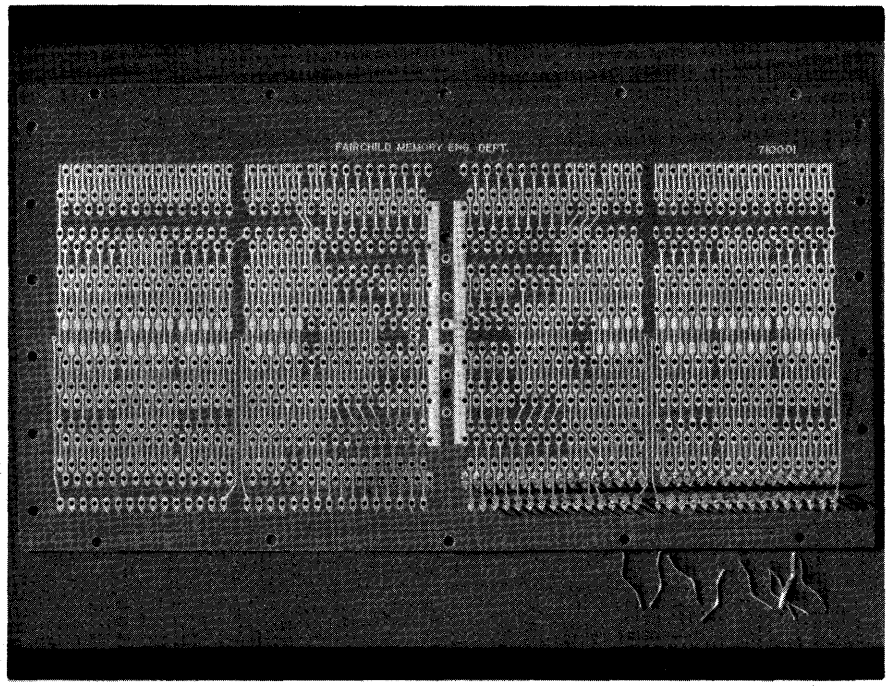



Figure 9. Mother board — top side.

### Card Cage

Figure 12 shows a photograph of the partially assembled card cage. This mechanical assembly provides two primary functions. They are mechanical support and guiding of the individual boards into the mother board and connector assembly, and providing proper cooling for the assembly via a small fan and a controlled air flow path. In addition, the card cage will house the right angle drives that allow rotating the cams from the front of the unit. Thus it will be possible to completely service the unit from the front, i.e., withdraw and insert cards or insert an extender board and contact assembly.

### Overall Unit and Power Supply

The card cage is bolted to a standard 7" rack panel. The panel has a cutout to permit servicing the unit. The depth of the assembly will be about 11", allowing the power supply to be placed directly behind the memory unit in the same rack space. The power supply requirements are approximately as follows:

| | | |
|---|---|---|
| Total d-c Power | | 132 watts |
| + 12 V | 0.7 amps | ± 10% |
| + 4.5V | 7 amps | ± 5% |
| + 2 V | 5 amps | ± 5% |
| − 0.8V | 1 amp | ± 10% |
| − 2 V | 3 amps | ± 10% |
| − 5 V | 15 amps | ± 5% |

## OPERATING CHARACTERISTICS

A pulse program consisting of write one, read one, write one, read zero, and then repeat has been used for the testing. The oscillographs of Fig. 13 show some of the waveforms for this system. A is the negative output pulse from the first decoder and inverter as it appears on one of the decode input lines on the storage card. These are all essentially "open" lines so the reflections present are reasonable.

B shows the output of the final decoder and word driver as it appears on the word select line. The first pulse is the read zero, followed by write one, etc. Mid-amplitude delay through the circuit is observed to be about 20 nanoseconds. The long tail off (about 50 nanoseconds) on the trailing edge of the pulse is due to stored charge in the base of the gating transistors. The cycle time shown here is 150 nanoseconds.

C shows the output of the sense amplifier. As ex-

pected, a signal is produced by the write "1" operation as well as the read "1" operation. Mid-amplitude delay from word line to data out is about 30 nanoseconds.

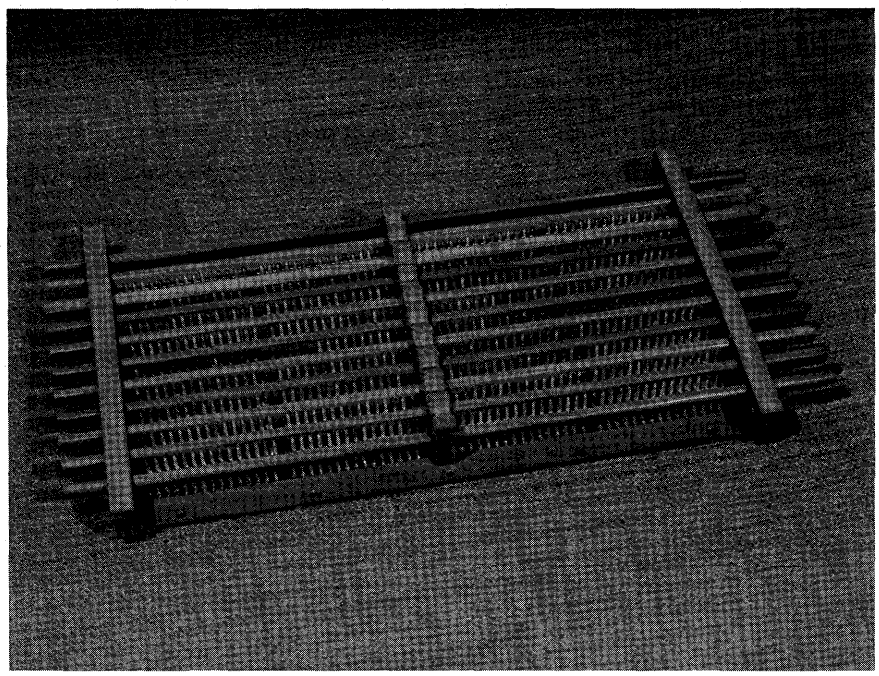


Figure 10. Connector assembly.

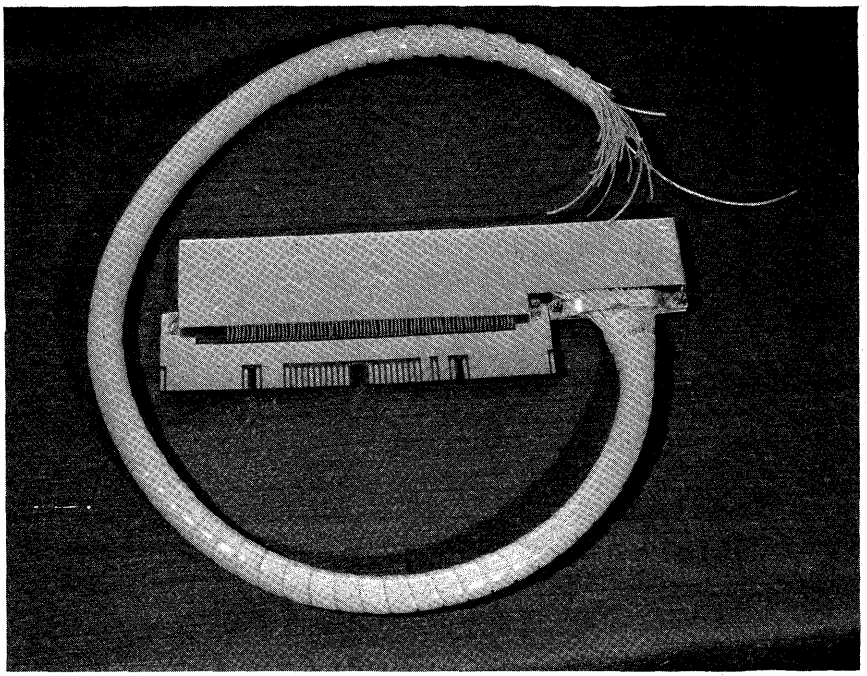


Figure 11. System interface connector assembly.

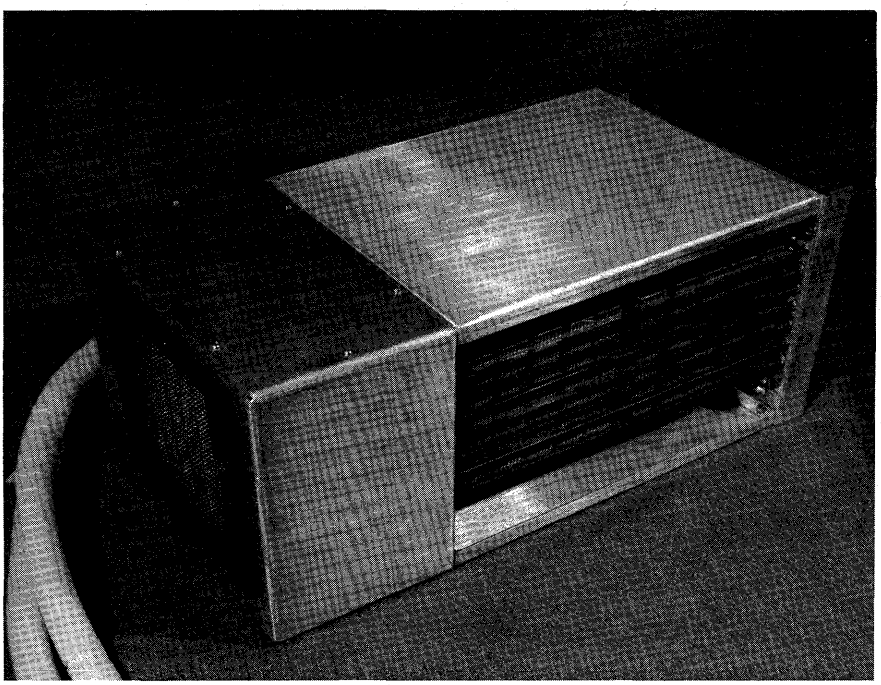


Figure 12. Card cage.

To demonstrate possible crosstalk that exists on the bit line, the repetition rate was slowed down and a zero and one superimposed. Figure 14(a) shows the word line and 14(b) the bit line waveforms. For this case, the two adjacent bit lines of both sides of the central bit had complement data written in and read out at the same time as the central bit line was being operated on. The photo is of the central line and the small amount of crosstalk at the leading and trailing edges is apparent.

## CONCLUSIONS

The results obtained have shown the feasibility of producing integrated circuit memory systems which offer advanced performance and design simplicity. Such memories may be expected to have an important place in applications where high speeds at low costs for moderate storage capacity are needed. Our experience has shown that the engineering of an integrated semiconductor memory system is much easier than that required for a thin magnetic film memory of comparable performance.

As semiconductor fabrication techniques continue to advance, further improvement in semiconductor memory systems is certain.

## ACKNOWLEDGMENTS

Many persons have contributed in one way or another to this project and all cannot possibly be named. However, special thanks are due to the Digital Integrated Circuit Section (R. Seeds) for fabricating the storage array and the Digital Systems Research Department (R. Rice) for many of the packaging concepts and, in particular, for the design of the mother board connector assembly. We are also indebted to J. Friedrich for the design and fabrication of the word driver circuit, and H. Zinschlag for the design of the data circuits.

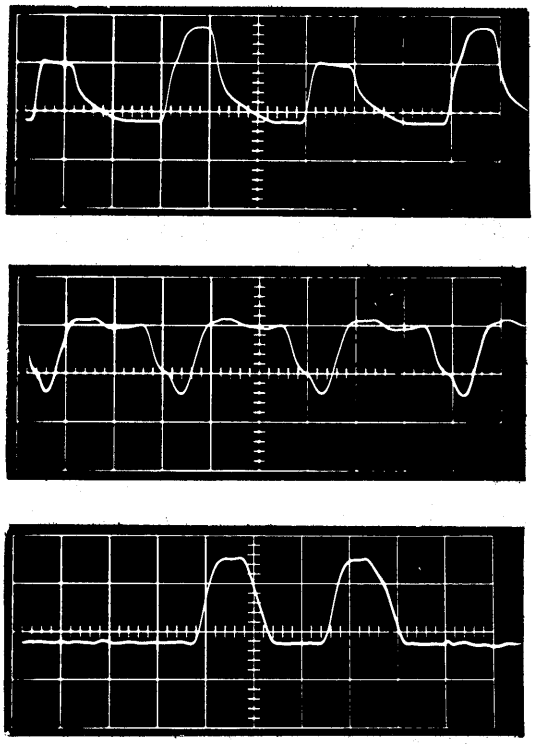


Figure 13. Memory system waveforms (50 nanoseconds/square, 2 volts/square — execept as noted).
a. Decode input pulse.
b. Word line.
c. Sense amplifier output.

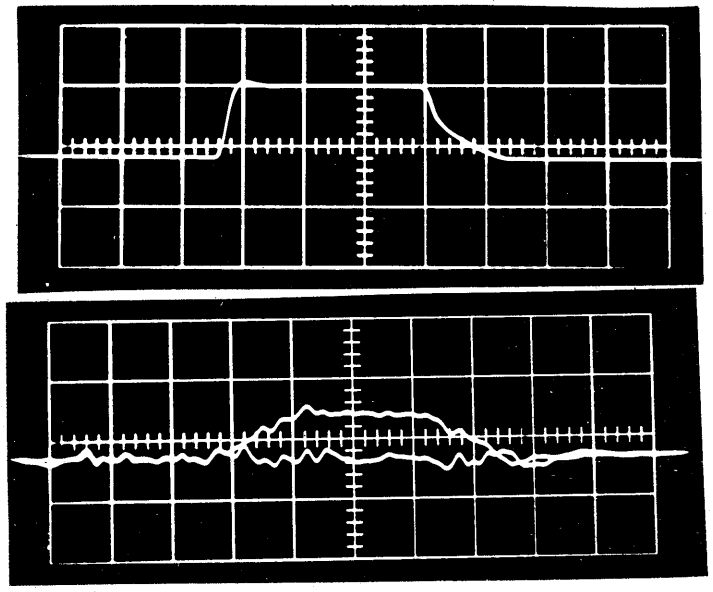


Figure 14. Superimposed one and zero outputs (40 nanoseconds/square).
a. Word line (2 volts/square).
b. Bit line (100 mv/square).

# STROBES: SHARED-TIME REPAIR OF BIG ELECTRONIC SYSTEMS*

Jesse T. Quatse
*Computation Center*
*Carnegie Institute of Technology*
*Pittsburgh, Pennsylvania*

## INTRODUCTION

The trend towards larger, more elaborate, and more complex computer systems is gradually modifying many programming and engineering techniques. The sheer bulk of equipment required by contemporary computer centers, in itself, creates new and severe problems in many areas. In particular, the maintenance of big systems has been complicated by three important characteristics: redundancy, complexity, and expense.

Redundancy has adversely affected system reliability. Devices such as memory modules are respectably reliable when administered in small doses—but memory banks consisting of 32 or more modules are currently available. In these quantities, even the most reliable modules can become collectively significant as a source of down-time. One popular, but expensive, approach to the problem is the refinement of packaging techniques so that entire modules, including addressing circuits, may be unplugged and replaced as units.

The complexity of a big system imposes a heavy burden on maintenance personnel. As a rough comparison of complexity, a maintenance engineer who is responsible for a single unit in a big system may be required to retain an understanding of a device two orders of magnitude larger, by circuit count, than an equivalent unit produced six years ago. Maintenance crews have been growing, in response to the problem, so that each engineer need be familiar with only a fragment of the entire system. Again, the approach appears to be more expensive than effective.

Big systems mean big cost in down-time. Although the maintenance engineer can think no faster than he did six years ago, the hourly cost of down-time has grown considerably. Proportional increases in maintenance crew sizes is an unsatisfactory approach to the problem. Thinking speed is not linearly accumulative.

STROBES (Shared-Time Repair Of Big Electronic Systems) is an attempt to ease these problems. It is based upon two assumptions. First, a big computer system is an unnecessarily overpowered device for regenerating the trace on a CRT console —even if the console is a maintenance engineer's oscilloscope. Flicker-free wave shape display requires no more than 32 uniformly spaced runs of the maintenance program during each second of use. Second, the bigger the system, the lower the

*This work was supported by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-146).

probability that any isolated facility is crucial to system operation. A redundant system facility, such as a core memory module, a magnetic tape drive, a telephone channel, etc., can be removed from general use without destroying the integrity of the computer system. The two assumptions lead to the STROBES approach of isolating the faulty unit, so that standard time-shared programs are denied its use, and of blanking the maintenance CRT cathode whenever standard programs are running. STROBES thereby achieves a "stroboscopic" effect in which the maintenance engineer is able to monitor the faulty equipment, at cyclic intervals, during the time-shared running of his maintenance routines. In typical trouble-shooting situations, the standard users are hardly aware of the presence of the maintenance engineer, and he is totally unaware of them.

## TIME-SHARED TROUBLESHOOTING

The STROBES equipment and programs treat the maintenance engineer as a nonstandard time-shared system user. The equipment consists of a modified oscilloscope, a small control panel, and central processor circuits for communicating with the oscilloscope and control panel. The control panel offers a few basic operating options and is mounted on the oscilloscope frame. The oscilloscope is modified to accept blanking and unblanking commands from the STROBES programs.

Three modes of operation are provided by the STROBES programs. A "log-in" mode is furnished for identifying the maintenance engineer and the faulty unit and for initiating the "conversational" mode of troubleshooting. In conversational mode, the maintenance engineer can monitor and control the system by means of tasks called from the permanent library. The tasks can be used to exercise faulty units, assemble other tasks, inspect and alter the permanent library, terminate operation, and initiate or terminate a "stroboscopic" mode without terminating the conversational mode. The stroboscopic mode serves the purpose of regenerating the trace on the modified oscilloscope. A real-time interrupt, of the frequency selected at the control panel, calls a previously specified maintenance routine which unblanks the CRT, exercises the faulty unit, blanks the CRT, and returns control to the interrupted program.

The primary advantage of this approach to troubleshooting is the drastically reduced cost of down-time. Each run through a typical stroboscopic mode program, for example one which reads and writes into a faulty memory module, requires less than 1 millisecond including interrupt overhead. At 15 repetitions per second, near flicker-free display is obtained for the cost of 1.5 percent degradation in system speed. (During extended periods of independence from the CRT, the maintenance engineer can reduce this time degradation to zero by disenabling the STROBES interrupt. A switch on the oscilloscope control panel is provided for this purpose.) Ignoring for the moment the cost of configuration degradation, the cost of down-time apparently drops to a negligible value.

In addition to its more obvious value, virtually free troubleshooting time should lower the requirements imposed upon maintenance crew sizes, individual skills, and equipment familiarity.

The effect of configuration degradation during STROBES troubleshooting is not easily related to the cost of down-time. In special cases (for the example given, those in which the system programs handle a "virtual memory" which far exceeds the capacity of the "actual memory") system performance is relatively insensitive to small changes in the system configuration. System performance degrades as some function of lost capacity when configuration changes are sufficiently large; but in the example, large changes reduce multiple faults and are rare. In general, configuration degradition has a binary effect on programs. Programs which reduce service from the faulty unit cannot be run. For them, the entire system can be considered inoperative during the repair period. For those which do not require the faulty unit, the entire system can be considered fully and continuously operative. The system is more reliable in effect, if not in fact. This increase in effective reliability is of particular signifficance to users who interact with the system in real time and are therefore acutely aware of inactive periods.

The advantages of the STROBES system are not obtained without cost. System programs must be rewritten to accept the maintenance engineer as a nonstandard user. More important, they must function in a system which can change configuration upon operator command. System programs having similar properities are already available from manu-

facturers who support many configurations of one basic system. Systems which make use of paging and segmenting hardware are particularly adaptable to the STROBES troubleshooting of memory modules. However, a sizable cost can be expected if these systems are to be adapted to permit the isolation of any nonessential system facility during STROBES troubleshooting. The obstacles are greater, but not insurmountable, for system programs which are rigidly fixed to one system configuration.

A worthwhile gain can be obtained by relegating one facility at a time to the nonessential status. For example, an essential line-printer can be buffered by tape or disk so that images are loaded on an intermediate storage during printer down-time, then unloaded and printed later. (Because of the image scrambling problem, elaborate printer buffering is already mandatory in time-shared systems.) Buffered line printers can be efficiently isolated for STROBES troubleshooting because of their normally low duty cycles. At Carnegie Tech, a 900-line-per-minute-printer produces an average of about $2 \times 10^{-5}$ lines per 24-hour day. At that rate, about 20 hours each day can be devoted to line-printer maintenance without affecting system performance.

Another important cost of implementing STROBES results from the rewriting of maintenance programs. They are subject to the same versatility requirements as system programs. They must be parametric so that they may be confined in extent, and non-iterative so they can be run in stroboscopic mode. They must be cataloged and assembled in a maintenance library so that small relevant subroutines may be called conveniently. Finally, the conversational mode requires an appropriate interpretation and assembly system.

Other costs and disadvantages of the STROBES system are less tangible. A psychological reaction can be expected on the part of both the maintenance engineer and the standard time-shared user. Both object to trouble shooting during production runs. The objections are based upon the fear that the maintenance engineer will interfere with the standard user's program by accidently touching a circuit component with an electrically active object. Such accidents might cause program errors which are difficult or impossible to detect. Although standard maintenance procedures rarely endanger the standard maintenance programs in this way, insufficient data are available to determine the extent to which the objections are valid for production runs where mishaps are more costly.

In conclusion, it is worth noting that the disadvantages of STROBES may not survive installation and general user acceptance.

## STROBES OPERATION

### *Mandatory Equipment*

The stroboscopic mode requires a nonstandard oscilloscope, a small control panel, and appropriate communicating circuitry located in the central processor. The circuit shown in Fig. 1 is used at Carnegie Tech to adapt a Model 535A Techtronix oscilloscope. It is housed in the preamp storage bay of the oscilloscope and mounted on the control panel which covers the front of the bay. No modifications to the oscilloscope proper are required other than the addition of the CATHODE BLANK and TRIGGER INHIBIT wires shown in Fig. 1. The switches, indicators, and connectors shown in the figure constitute the control panel. Repetition rates of 1, 15, and 60 per second are selected by the 3-position REP RATE switch. Repetitive traces are easily visible on the standard 535A CRT at the 1 per second rate. Traces are near flicker-free at 15 per second and entirely free at 60 per second. The reptition rate selector wires run to AND gates in the processor which enable interrupts generated by a real-time clock. The connecting cable is 100 feet in length so that the oscilloscope may be wheeled to any unit in the system. The four-position MODE switch must be positioned at AUTO to obtain automatic trace repetitions at the selected rate. Interrupts are generated manually, for one-shot cycling, by depressing the ENABLE switch while MODE is in the MAN position. The manual interrupt line circumvents the real-time clock at the processor. INHIBIT TRACE is the only signal received from the processor. It is a d-c level supplied by a programmable processor flip-flop. In either AUTO or MAN mode, INHIBIT TRACE blanks the CRT cathode and inhibits further triggering. The TEST position of MODE substitutes a 1-kc square-wave oscillator for the inhibit trace signal. The NORM position returns the oscilloscope to normal operating condition.

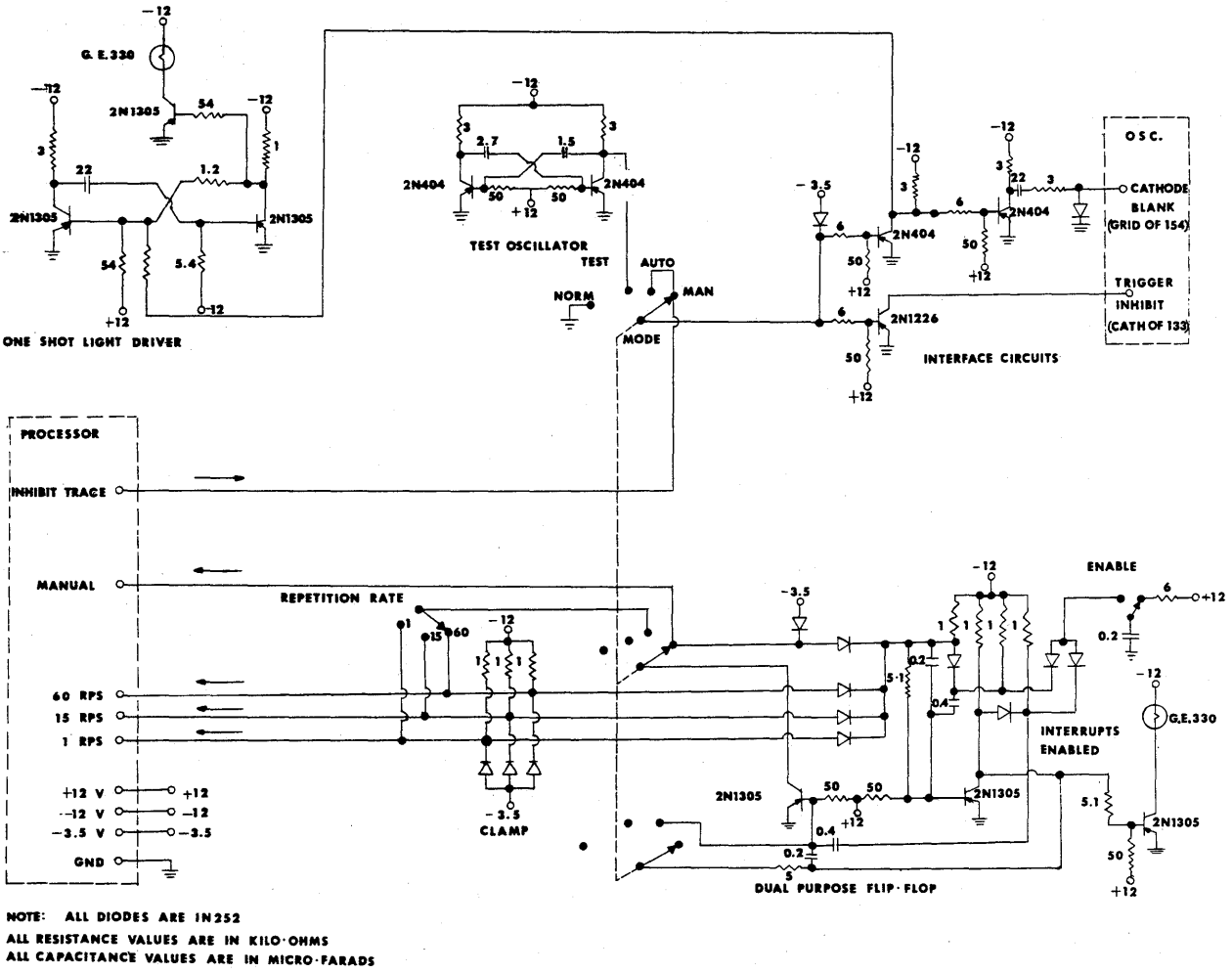



Figure 1. The oscilloscope circuits.

*The STROBES Assembly Language*

A version of the standard machine language has been augmented by special STROBES macros and pseudos so that closed relocatable subroutines can be assembled in conversational mode. Statements in the language are of the form "LABEL, OP-CODE, PARAMETER, PARAMETER; COMMENTS" and are assembled as they are typed. The relative address of each statement is automatically typed at the beginning of each input line so that it may be used as a parameter. Relative addresses are distinguished from numeric values by a preceding decimal point.

At the completion of an assembly, a debugging mode is automatically entered, in which error comments are typed-out and corrections are typed-in. When errors are corrected to the satisfaction of the assembler, a request for a unit name is automatically typed-out. After the name is typed, the subroutine is automatically dumped onto a temporary library and classified as a "task" for the specified unit. The task can be called and executed in either the conversational or stroboscopic mode, only if the specified unit was reserved during log-in. If no unit is specified, the task is assigned to all units. A permanent task is available which copies tasks from the temporary library to a permanent master library which remains intact when STROBES is terminated.

A task call is of the form "NAME, FIRST PARAMETER, SECOND PARAMETER,..., $N^{th}$ PARAMETER, STROBES MODE; COMMENTS" where NAME can be any alphanumeric string and $N \leq 9$. The task can be run in stroboscopic mode only if indicated by the STROBES MODE parameter. The actual parameters are passed by means of a macro

called "Par." The PAR macro requires one parameter: a digit which represents the position, in the task call, of a parameter which is to be brought to the accumulator. For example, the occurrence "PAR,2;" in the task called by "LOAD, 100, 200, 300;" will cause the assembly of code which brings the number 200 to the accumulator. A list of pseudos and macros are given in Table 1 and Table 2. Angle brackets are used in the tables to denote class names.

Table 1. The STROBES Assembly Language Pseudos.

ENT;
Defines the automatically typed relative address as the entry point of a library task.

DEF, < LABEL >, < INTEGER >;
Defines < INTEGER > as the value of < LABEL >.

END;
Terminates assembly and enters the task being assembled into the temporary library.

PRT;
Causes each used label and its defined value, if any, to be typed-out in tabular form.

PRT, < INTEGER >;
Executes the PRT pseudo for labels L0 through < INTEGER > only.

DMP, < INTEGER >, < LOCATION >;
Types the contents of the < INTEGER > memory locations beginning at < LOCATION >.

Table 2. The STROBES Assembly Language Macros.

RTM;
Returns control to the program marked in the ENT location.

TON;
Turns the oscilloscope trace on (unblanks the CRT).

TOF;
Turns the oscilloscope trace off (blanks the CRT).

TPI, < INTEGER >, < LOCATION >;
Enables type-in of < INTEGER > octal numbers beginning at < LOCATION >.

TPO, < INTEGER >, < LOCATION >;
Types-out the octal contents of the < INTEGER > words beginning at < LOCATION >.

PAR, < INTEGER >;
Brings the parameter in call position < INTEGER > to the accumulator.

CLL, < NAME >, < PARAMETER >, < PARAMETER >, . . .;
Calls the library task specified by < NAME > and supplies the ensuing parameters.

*The Operation Modes*

The first phase of the log-in mode identifies the maintenance engineer to the general time-shared system programs and registers his request for STROBES. In the second phase, any tasks accessible in log-in mode may be called. Of the log-in tasks shown in Table 3 only "DIRECTORY" leaves the system in log-in mode. The third phase terminates log-in by returning to a previously aborted state, by terminating STROBES, or by progressing to the conversational mode.

Table 3. The Log-In Tasks.

DIRECTORY;
Types the unit, name, number of parameters, STROBES mode, and length of each task registered in the permanent library.

DONE;
Terminates STROBES.

RESTORE;
Establishes the conditions which existed when a SAV task was last executed. (See Table 4.)

UNIT, < ALPHA-NUMERIC >, < ALPHABETIC >;
Declares the unit or units specified by the name < ALPHA-NUMERIC >, or the subunit delineated by the < ALPHABETIC > descriptor, as reserved for exclusive use of the maintenance engineer, loads all of the relevant subroutines from the permanent library, creates an appropriate temporary directory, and initiates the conversational mode.

Once initiated, the conversational mode remains in effect until STROBES is terminated. Successful initiation is verified by the automatic type-out of the message "STROBES IS NOW PREPARED FOR" followed by the name of the unit requested. After an automatic carriage return, the letter "T" is typed-out to request input of a task call. When the task call is received, either a standard message, in-

formation particular to the called task, or a "T" is typed-out. In this way, the reception of typed-in information is always verified by a subsequent type-out. The conversational mode tasks are listed in Table 4.

Table 4. Conversational Mode Tasks.

TSK, < NAME >, < INTEGER >, < MODE >;
Begins assembly of the temporary library task called < NAME > requiring < INTEGER > parameters and operable in stroboscopic mode if < MODE > = 1.

STT, < NAME >, < PARAMETER >, < PARAMETER >, . . . ;
Initiates stroboscopic mode and calls the task specified by < NAME > with the ensuing parameters.

STP;
Stops the stroboscopic mode by inhibiting the STROBES interrupt.

DIRECTORY;
Identical to the log-in mode task by the same name.

INI;
Clears the temporary library.

DLT, < NAME >;
Deletes the task specified by < NAME > from the temporary library.

RMV, < NAME >; Deletes the task specified by < NAME > from the permanent library.

CPY, < NAME >;
Copys the task called < NAME > from the temporary library to the permanent library if it is not already permanent.

MRG, < NAME >;
Merges the entire temporary library with the permanent library.

TPI, < INTEGER >, < LOCATION >;
Enables type-in of < INTEGER > octal numbers beginning at < LOCATION >.

TPO, < INTEGER >, < LOCATION >;
Types-out the octal contents of the < INTEGER > words beginning at < LOCATION >.

ATI, < INTEGER >, < LOCATION >;
Enables type-in of < INTEGER > alpha-numeric characters, one per word, beginning at < LOCATION >.

ATO, < INTEGER >, < LOCATION >;
Types-out the alpha-numeric contents of the < INTEGER > words, one character per word, beginning at < LOCATION >.

SAV;
Completely copies the state of the STROBES system so that restoration can be made by the log-in task "RESTORE" and terminates STROBES.

OUT;
Terminates STROBES unrecoverably.

Entrance and exit to the stroboscopic mode is controlled by the conversational mode tasks STT and STP. The STP task is equivalent, in effect, to disenabling the STROBES interrupt at the control panel. Thus, the stroboscopic mode can be considered to be the "iterative" form of conversational mode tasks.

## AN EXAMPLE OF USE

The coordination of conversational and stroboscopic modes, during the trouble-shooting process, is best described by example. What follows is a contrived protocol intended to illustrate the effect that the maintenance engineer is the sole system user while he absorbs an estimated 2 percent of total system time. Typing performed by the STROBES system programs is underlined for clarity.

The maintenance engineer is required to eliminate recurrent parity errors in an 8K memory module having the octal address range 120000 to 140000. He initiates log-in by the standard identification procedure required of all time-shared users. He then requests all library procedures relevant to the faulty memory module (module MM11, E) and reserves it for his own exclusive use.

UNIT, MM11, E;

After a brief pause in which the requested routines are prepared for use, a type-out informs the maintenance engineer that conversational mode has been established.

STROBES IS NOW PREPARED FOR MM11 E

The letter "T" is typed-out to request a conversational mode task. The maintenance engineer then

types a small routine, by means of the task "TSK", which simply clears and then sets every bit in a word of the defective module. Three parameters are required by TSK: the task name (MEM), the number of parameters (2), and the mode of operation desired (1 for stroboscopic). The maintenance engineer intends to use the oscilloscope to trace the information path to and from the defective module.

```
T TSK, MEM, 2, 1;
0000 ENT; SPACE FOR THE MARK
0001 PAR, 1; BRING THE FIRST PARAMETER
0004 STL,, LO; STORE IT IN LOCATION LO
0005 PAR, 2; BRING THE SECOND
     PARAMETER
0010 STL,, L1; STORE IT IN LOCATION L1
0013 STZ, 1, L0; STORE ZEROS IN THE LOCA-
     TION SPECIFIED BY L0
0014 CAL,, L1; CLEAR ADD THE PATTERN
     IN LOCATION L1
0015 STL, 1, L0; STORE THE PATTERN IN
     THE LOCATION SPECIFIED BY L0
0016 TOF; TURN OFF THE OSCILLOSCOPE
     TRACE
0017 RTM; RETURN TO MARK
0020 L0, 0; DECLARE L0 AND ITS CONTENTS
0021 L1, 0; DECLARE L1 AND ITS CONTENTS
0022 END; TERMINATE ASSEMBLY
```

If assembly is unsuccessful, special debugging comments and requests will be typed-out. The maintenance engineer may then debug and correct his code. When the task is acceptable to the asembler, a type-out requests the name of the unit for which the task is intended.

```
SPECIFY UNIT
MM11
```

The task is added to the MM11 category of the temporary library. Successful assembly and library updating is then reported by a type-out and the next task is requested. The maintenance engineer enters stroboscopic mode by calling the newly created MEM task with STT. He supplies the address 120000 and the pattern 77777777777 as parameters to MEM.

```
LIBRARY UPDATED
T STT, MEM, 120000, 77777777777;

T
```

The stroboscopic mode is now in effect. The maintenance engineer commences trouble-shooting with the oscilloscope. After an hour of unsuccessful labor, he decides that a call to his supervisor is appropriate. The SAV task will recoverably remove STROBES from the operating system. After recognizing the call for SAV, STROBES types a recognition message, preserves the temporary library and the operating state, and terminates.

```
T SAV
STROBES SAVED
```

After consulting with his supervisor, the maintenance engineer must log-in once again. He can recover from the SAV operation by means of the RST task.

```
RST
STROBES IS NOW PREPARED FOR MM11, E
T STT
T
```

Troubleshooting with the oscilloscope resumes. As other tasks are found necessary, they may be written or called from the library.

```
T TSK, SEQ, 1, 1;
0000 ENT
0001 }
...  } code which sequences the MEM task
0016 } through memory
0017 END
SPECIFY UNIT
MM11
T STT, SEQ, 77777777777;
T STP
T
```

When the faulty component is found, stroboscopic mode is terminated by STP, the memory modules is powered down, the component is replaced, the memory modules is again powered up, and an acceptance routine MTA is called. The routine reports faultless operation so the maintenance engineer terminates operation.

```
T MTA, 'E';
MEMORY MODULE E O.K.
T OUT;
```

## CONCLUSION

The work described in this paper serves more to introduce than to exploit the STROBES approach. The central theme has been efficient troubleshoot-

ing in emergency conditions. No mention is made of the many periodic "preventive maintenance" formance. These procedures are particularly adaptable to the STROBES approach and should be a profitable realm of application. As systems increase in size, speed, and modularity, they should become more adaptable to the STROBES approach. An attempt to incorporate STROBES at the equipment and program design level may lead to significantly reduced maintenance costs in future systems.

## ACKNOWLEDGMENTS

The author is grateful for the assistance of his colleagues Arthur Yaffe, John Schlotterer, and Beau Brinker.

# A SELF-DIAGNOSABLE COMPUTER

R. E. Forbes, D. H. Rutherford, C. B. Stieglitz
*International Business Machines Corporation*
*Federal Systems Division*
*Owego, New York*
and
L. H. Tung
*International Business Machine Corporation*
*Systems Development Division*
*Endicott, New York*

## INTRODUCTION

A self-diagnosable computer is a computer which has the capabilities of automatically detecting and isolating a fault (within itself) to a small number of replaceable modules.

This paper presents a new concept which, in mid-1962, resulted in the design of a self-diagnosable computer called the DX-1. The design effort represented one of a number of related and continual efforts aimed at finding the best way to achieve maximum operational availability of a computer to do useful work. The design objectives were:

1. Maximum capability for self-diagnosis.
2. Minimum mean-time-to-diagnosis (MTTD). (MTTD).
3. Minimum additional hardware to make it self-diagnosable.
4. Minimum hard core (that portion of logic which must function correctly).

The DX-1 was logically implemented and simulated on the IBM 7090 during 1963. The results of the logic simulation indicated that the basic philosophy of self-diagnosis is technically correct. It also confirmed two facts, namely: (1) the design of a self-diagnosable computer must originate with the system architecture and must be treated as a principal design parameter, and (2) it is mandatory that the diagnostic programs be automatically generated by a computer.

This paper describes *one* way a computer can be designed to be self-diagnosable. Although much of what is discussed here has undergone substantial modification and advancement, it remains fundamental and valid.

Before discussing the design techniques used to make the DX-1 a self-diagnosable computer, we present an abstract model as a basis for design and then describe the DX-1's organizational structure and operation in its normal mode. We conclude with certain specific results.

## A MODEL OF SELF-DIAGNOSABLE DIGITAL SYSTEMS

It is well known that a digital system $S_1$ possessing a certain configuration is capable of diagnosing single solid faults of another digital system $S_2$, provided that all the nonredundant inputs and outputs of system $S_2$ are accessible to $S_1$. In particular, where system $S_2$ consists of combinational logic, this method has been demonstrated for various systems.[1,2,3]

When a digital system is partitioned under certain restrictions into subsystems it is possible to achieve self-diagnosis of the system through the mutual diagnosis of its subsystems.

### *Diagnosis and Diagnostic Systems*

We state briefly, without proof, a method of diagnosis of logical malfunctions of a digital system and the requirements of a diagnostic system to be used for such diagnosis.

1. Let $S_1$ and $S_2$ be two examples of a digital system S, where $S_1$ is the system to be diagnosed and $S_2$ is known to be nonmalfunctioning. The diagnostic method consists of applying a sequence of stimuli to the inputs of $S_1$ and $S_2$ and of comparing the outputs of $S_1$ to the corresponding outputs of $S_2$.
2. When S contains combinational circuits only, the sequence of stimuli may consist of all distinct combinations of input signals or some subset thereof. By utilizing the results of the comparison previously referred to in (1), it is possible to decide whether $S_1$ is malfunctioning, if the sequence of stimuli is properly chosen. In most cases, it is possible to obtain better diagnostic resolution, in the sense that the malfunction can be isolated to a relatively small portion of $S_1$.
3. When S contains sequential circuits, a suitable sequence of stimuli may be derived from the original design information. However, it is difficult to obtain as good a resolution as is possible with combinational circuits.
4. Applying the diagnostic procedure[4] and implementing the above-described method, a digital system R can be used to diagnose another malfunctioning digital system S, provided that R has the following capabilities:

   $F_1$: Supply the linkage to enable execution of a given sequence consisting of the operations $F_2$, $F_3$, $F_4$ and $F_5$ appearing repeatedly in any order.

   $F_2$: Transmit predetermined stimuli to all, or any selected portion, of the inputs of the system S.

   $F_3$: Observe the outputs of the system S and compare them with given patterns.

   $F_4$: Proceed to the next operation in the sequence or branch to a different point of the sequence of operations, depending on the result of the last $F_3$ operation.

   $F_5$: Stop and communicate to the human operator the place in the sequence where the stop occurred or communicate other chosen information such as the location(s) of replaceable unit(s) that have been diagnosed as being faulty.

### *A Model of Self-Diagnosable Digital Systems*

A system S is self-diagnosable if S can be partitioned into $n$ mutually exclusive subsystems, $S_1, S_2, \ldots, S_n$ for $n \geq 2$; and, the following conditions hold:

(a) Diagnosability of subsystems $S_i$:
Each $S_i$, $i = 1, 2, \ldots, n$, must consist of either combinational circuits only; or, if it contains sequential circuits, a sequence of stimuli for its diagnosis must be possible and known.

(b) Existence of diagnostic subsystems:
Among the subsystems $S_1, S_2, \ldots, S_n$, there exist subsystems, each of which can perform all the five operations $F_1, F_2, \ldots, F_5$. These subsystems form the class of the diagnostic subsystems which we will denote by $C_{DS} \cdot C_{DS} = \{S_j\}$, $j = i_1, i_2, \ldots, i_m$, $m \leq n$.

(c) Communication between $S_i$ and $C_{DS}$:
The inputs and outputs of each subsystem $S_i$, $i = 1, 2, \ldots, n$, are accessible to at least one of the diagnostic subsystems (other than itself if it is in $C_{DS}$).

Following the discussion in the preceding section, we assume that a subsystem $S_i$ is diagnosable if its input and output terminals are accessible to a nonmalfunctioning subsystem $S_\alpha \in C_{DS}$. The word "diagnosable" implies that the action of diagnosis

described in the preceding section is possible by using $S_\alpha$ as the diagnostic digital system where $S_i$ is the digital system being diagnosed. We assume also that S is diagnosable if all the $S_i$ are diagnosable. Furthermore, we assume that when a malfunction subsystem is diagnosed, it is immediately repaired.

*Classes of Subsystem Failures for Which the Model is Self-Diagnosable*

Certain conditions of failure are inconsistent with self-diagnosability, the extreme case being simultaneous malfunctioning of all subsystems. We will examine the classes of failures for which the model defined above is self-diagnosable.

1. It is possible to construct a self-diagnosable system S if at least two subsystems among $S_1, S_2, \ldots, Sn$ belong to $C_{DS}$.
2. S is self-diagnosable, independent of the number of malfunctioning $S_i$ ($i = 1, 2, \ldots, n$), if no more than one of the malfunctioning $S_i$ is in $C_{DS}$.
3. S is self-diagnosable no matter how many of the $S_i$ are malfunctioning if for each of the malfunctioning $S_i$ in $C_{DS}$, there exists at least one known nonmalfunctioning $S_j \in C_{DS}$, and the inputs and outputs of $S_i$ are accessible to $S_j$.

4a. Let $S_{i_1}, \ldots, S_{i_m}$ be a subset of $S_1, S_2 \ldots, S_n$. $S_{i_1}, S_{i_2}, \ldots, S_{i_m}$ is said to form a diagnostic sequence from $S_i$ to $S_{i_m}$ if the input and output terminals of $S_{i_p}$ are accessible to $S_{i_{p-1}}$ for $p = 2, 3, \ldots, m$, while $S_{i_1}, S_{i_2}, \ldots, S_{i_{m-1}} \in C_{DS}$ and $S_{i_2}, S_{i_3}, \ldots, S_{i_m}$ are malfunctioning subsystems. Then, $S_{i_m}$ is diagnosable if $S_{i_1}$ is known to be nonmalfunctioning.

4b. S is self-diagnosable no matter how many of the $S_i (i = 1, 2, \ldots, n)$ are malfunctioning, if there exists at least one nonmalfunctioning $S_j \in C_{DS}$ and a diagnostic sequence can be formed from a nonmalfunctioning $S_j$ to each malfunctioning $S_i$.

*Extensions of the Model of Self-Diagnosable Digital Systems*

In this subsection, we will consider the extensions of the self-diagnosable digital system model, taking into account realistic situations as found in actual system design. We shall define S; $S_1, S_2, \ldots, Sn$; and $C_{DS}$ as above (see A Model of Self-Diagnosable Digital Systems).

1. Let $S\alpha$ be any of the $S_i \in C_{DS}$. Let $S\alpha$ be further partitioned into subsystems $S\alpha_1, S\alpha_2, \ldots, S\alpha_k$. The partition possesses the following properties:
   (a) $S\alpha_i$, $i = 1, 2, \ldots, k$, are also subsystems of S.
   (b) $S\alpha$ is nonmalfunctioning if each of the $S\alpha_i$ ($i = 1, 2, \ldots, k$) is nonmalfunctioning.
   (c) When $S\alpha$ contains no redundant or irrelevant elements, each $S\alpha_i$ ($i = 1, 2, \ldots, k$) is nonmalfunctioning if $S\alpha$ is nonmalfunctioning.
2. *Forming new diagnostic subsystems during diagnosis:* Let $S\alpha_1, S\alpha_2, \ldots, S\alpha_k$ be the subsystems of a diagnostic subsystem $S\alpha$ resulting from a partitioning of $S\alpha$. Let $S_{i_1}, S_{i_2}, \ldots, S_{i_p}$ be some of the subsystems $S_i$ of S.
   (a) If a new diagnostic subsystem $S_\beta$ can be formed by combining $S_{i_1}, S_{i_2}, \ldots, S_{i_p}$ and some or all of the $S\alpha_i$, say $S_{j_1}, S_{j_2}, \ldots, S_{j_m}$, then we can consider $S_\beta$ as the result of a new partitioning of S.
   (b) If all the $S_{i_1}, S_{i_2}, \ldots, S_{i_p}$ and $S_{j_1}, S_{j_2}, \ldots, S_{j_m}$ are diagnosable, then $S_\beta$ is diagnosable.
3. (a) $S_{i_1}, S_{i_2}, \ldots, S_{i_m}$ are said to form a chain diagnostic sequence from $S_{i_1}$ to $S_{i_m}$ if they possess the properties of a diagnostic sequence and at least one of the $S_{i_j}$ is formed by a new partition of S immediately after the $S_{i_{j-1}}$ is diagnosed and repaired, and each of these $S_{i_j}$ is a combination of known nonmalfunctioning subsystems of S in the new partition.
   (b) $S_{i_m}$ is diagnosable if $S_{i_1}$ is known to be nonmalfunctioning.
   (c) S is self-diagnosable no matter how many $S_i$ ($i = 1, 2, \ldots, n$) are malfunctioning if there exists at least one nonmalfunctioning $S_j \in C_{DS}$, and a chain diagnostic sequence can be formed from a nonmalfunctioning $S_j$ to each malfunctioning $S_i$.

4. *Indirect access in diagnosis:* Consider Fig. 1 where a diagnostic subsystem $S_\alpha$ has access to the inputs but not the outputs of $S_1$ and the outputs of $S_2$, $S_3$, and $S_4$; while at the same time, the output of $S_1$ can control the input of $S_2$, $S_3$, and $S_4$. $S_1$, $S_2$, $S_3$, and $S_4$ can be diagnosed by using a majority technique if only one of these four is assumed to be malfunctioning. This can be done as follows: (a) apply stimuli to $S_1$ and observe the output indirectly through the outputs of $S_2$, $S_3$, and $S_4$ individually; and then (b) accept only the majority of the three results. It is possible to use this technique in a variety of ways including the joining of diagnostic sequences and chain diagnostic sequences.

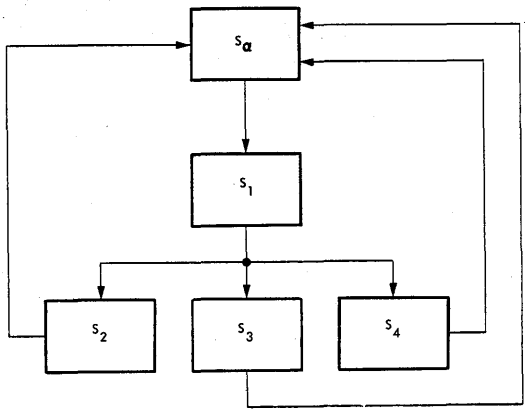



Figure 1. Indirect access.

5. *Partition vs. resolution in diagnosis:* Even though each $S_i$ ($i = 1, 2, \ldots, n$) is diagnosable, the resolution of diagnosis depends on the configuration of $S_i$. However, if S is partitioned such that each of the $S_i$ is small enough and preferably packaged in a single replaceable unit, it is not necessary to obtain better resolution than diagnosis to an $S_i$.
6. *Hard core:* When it is not possible to partition S into mutually exclusive diagnostic subsystems, and additional hardware cannot be added for economic or other reasons, some elements, say H(S), are then necessary to be in common to two or more diagnostic subsystems, say $S_{j1}$, $S_{j2}$, . . . , $S_{jm}$. If the nonmalfunctioning of at least one of these $S_{ji}$ is necessary for S to be self-diagnosable, we must be certain that H(S) is not malfunctioning prior to the commencement of diagnosis. In this case, H(S) is called the "hard core" of S.
   When a hard core H(S) exists, S is not completely self-diagnosable. We can, however, design H(S) to utilize self-checking hardware, and we can employ manual methods to diagnose H(S). When H(S) is tested and is known to be nonmalfunctioning, we may leave H(S) alone and treat S as a self-diagnosable system.
7. *Use of additional hardware during diagnosis:* Certain parts of a self-diagnosable digital system might not be necessary for the normal operation of the system. Such parts would therefore not be required to be connected to the system except when the system is to be used for self-diagnosis.

*Diagnostic Procedure*

For a given self-diagnosable digital system, there are various ways of constructing diagnostic procedures. We will present a method which will aid in determining whether a system is self-diagnosable. If this is the case, this method will also assist in the construction of a diagnostic procedure. A simple example is given in conjunction with this discussion.

1. List all the mutually exclusive subsystems $S_1, S_2, \ldots, S_n$ which are the result of all the expected partitions of S. For each subsystem $S_i$ ($i = 1, 2, \ldots, n$) list all the diagnostic subsystem or subsystems (denoted by $DS_j$) by which $S_i$ is diagnosable, either by direct or indirect access.
   In the example, there are eight subsystems. They are given in Table 1.

Table 1. Subsystems

| Subsystems | Diagnosed by |
|---|---|
| $S_1$ | $DS_2$ |
| $S_2$ | $DS_2$ |
| $S_3$ | $DS_1$ |
| $S_4$ | $DS_4$ |
| $S_5$ | $DS_3$ |
| $S_6$ | $DS_1$ |
| $S_7$ | $DS_2$ |
| $S_8$ | $DS_4$ |

2. List all diagnostic subsystems in the same manner. Furthermore, for each diagnostic subsystem $DS_j$, $j = 1, 2, \ldots, k$, list all the subsystems resulting from its partition. There are four diagnostic subsystems in the example. They are given in Table 2.

Table 2. Diagnostic Subsystems

| Diagnostic Subsystems | Diagnosed by |
|---|---|
| $DS_1$ $(S_1, S_2)$ | $DS_2$ |
| $DS_2$ $(S_3)$ | $DS_1$ |
| $DS_3$ $(S_1, S_6)$ | $DS_2, DS_1$ |
| $DS_4$ $(S_1, S_2, S_3, S_5)$ | $DS_2, DS_1, DS_3$ |

The method of partitioning and the diagnostic system requirements are necessarily interacting. Therefore, Tables 1 and 2 should be constructed simultaneously. Several trials are likely to be required before a satisfactory list can be constructed. If a list cannot be constructed, the system is not self-diagnosable.

3. Denote $A \rightarrow B$ to mean that subsystem B is diagnosable by subsystem A, $A\ B \rightarrow C$ to mean that subsystem C is diagnosable by the combination of subsystems A and B, etc. By examining Tables 1 and 2, using the relationship that a diagnostic subsystem is diagnosable if all the resulting subsystems from its partition are diagnosable, construct a *diagnostic diagram* as in Fig. 2. The diagnostic diagram shows how each of the diagnostic subsystems may be diagnosed.

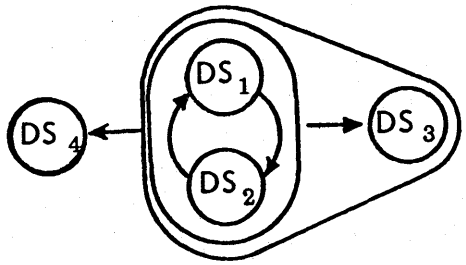



Figure 2. Diagnostic diagram.

## DX-1 AS A COMPUTER

Figure 3 presents the data flow within DX-1. In its normal mode, the DX-1 is a binary, fixed point, parallel, microprogrammed digital computer. It includes a single bus, six arithmetic registers (R1

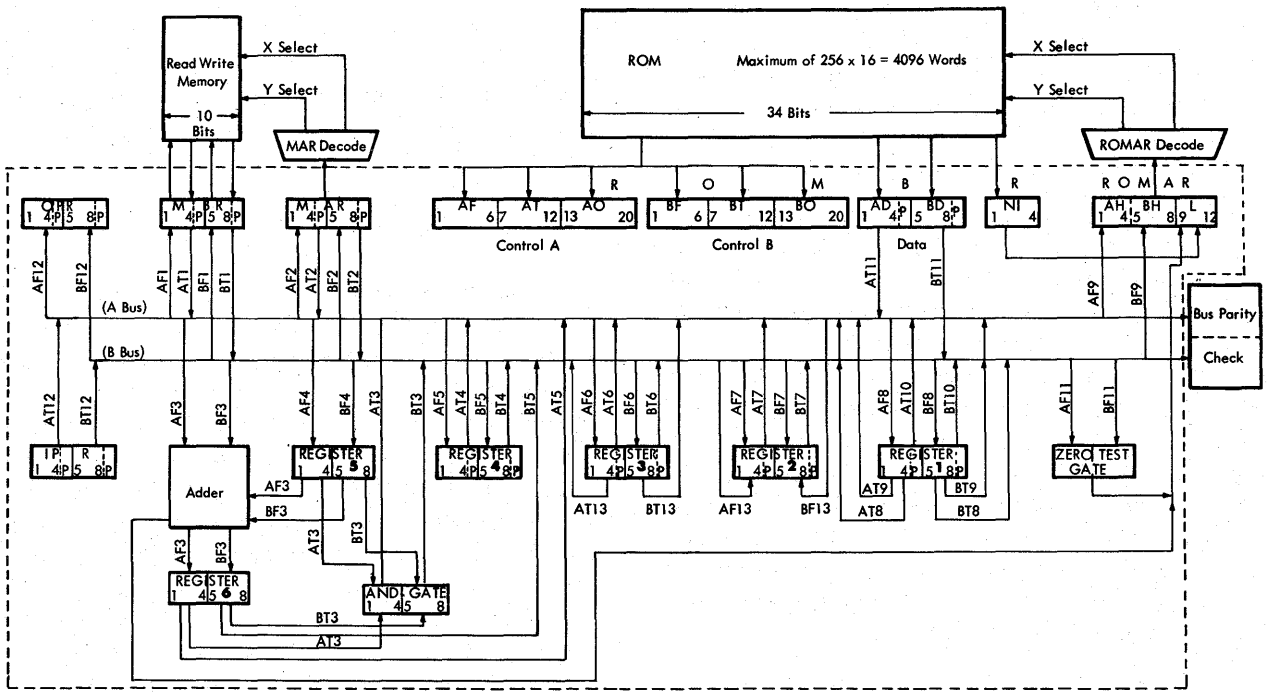



Figure 3. DX-1 data flow.

through R6), an ADDER, gates for logical AND and ZERO TEST, buffer registers IPR and OPR for communication with input and output devices, a read-write memory (RWM) addressed by the MAR and buffered by the MBR, and a read-only memory (ROM) addressed by the ROMAR (which is divided into H and L parts) and buffered by the ROMBR. The standard word length is eight data bits with two parity check bits and no sign bit. The ROM word is 34 bits long.

Micro operations are stored in the ROM. One field (O) of the ROMBR acts as the micro operation code register. Two other ROMBR fields (F and T) hold codes specifying which registers the micro operation must connect, respectively from and to the bus. The F, T, and O fields appear in both A and B control registers within the ROMBR because register control is divided into A (left or high order) and B (right or low order) parts. In normal operation, CONTROL A and CONTROL B contain identical bit configurations. The lengths of fields F, T, and O are six bits, six bits, and eight bits, respectively; and each must always contain exactly two binary ones in normal operation. They provide addressing capability for 15 registers, and codes for 28 micro operations.

The two remaining fields in the ROMBR can contain arbitrary binary numbers. The data (D) field holds one word and is 10 bits long. The next instruction (NI) field holds four bits. Its role in determining the next micro operation's address in ROM will be detailed later. Note that one ROM word is required for each micro operation.

Each micro operation is executed in one clock cycle. At time $t_1$, the micro operation is fetched into the ROMBR from the location in ROM addressed by ROMAR. Transfer (by micro operation XFER) of a number from the register addressed by the T field [R(T)] to the register addressed by the F field [R(F)] requires only two more clock times: $t_4$ for clearing R(F) and $t_5$ for connecting to the bus both the input of R(F) and the output of R(T). Other micro operations which require more timed steps are interleaved with the XFER micro operation using times $t_2$, $t_3$, and $t_6$. At time $t_7$, the ROMBR is always cleared to prepare it to accept the next micro operation (Fig. 4).

The DX-1 uses its micro operations to follow a sequence of macro operation codes stored in the RWM. Each macro operation is the high order eight bits of the addresses of 16 ROM locations. These locations, and more, if necessary, are used to store the sequence of micro operations that constitutes that particular macro operation. Each such sequence starts, by convention, in a ROM location whose four low order address bits are zeros. Consequently, each macro operation can start the next macro operation by READing its macro operation code from RWM into ROMAR H and simultaneously placing zero in ROMAR L.

Within each block of 16 ROM locations, the NI field of each micro operation specifies the address of the next micro operation. The ROMAR L is cleared at time $t_4$ and then loaded from the NI field at $t_5$ so the specified micro operation will be fetched at the next $t_1$. This is the ordinary sequencing within a macro operation. Continuing a macro operation beyond the 16 ROM locations addressed by its macro operation code is easily effected.

The last micro operation in the first group of 16 is made a XFER from its own D field to ROMAR H. The next micro operation executed will be in the group of 16 specified by D, and the low order four bits of its address will be determined by its predecessor's NI field as usual.

### *Micro Operations*

This completes the general description of DX-1. The various micro operations are now described in detail. How they may be combined to perform the ordinary functions of a digital computer is also shown.

#### 1. *XFER: Data Transfer Micro Operation*

As has been mentioned, the micro operation code for XFER appearing in the O field of the control registers causes the computer to clear R(F) and transfer R(T) into R(F) by way of the bus. Micro operation XFER has special uses in addition to its obvious applications in shuffling data. First, any XFER specifying the AND GATE by its T field connects registers R5 and R6 to the inputs of the gates so that the bit-wise AND of the contents of these registers is transferred to R(F). Second, special F and T codes are provided for registers R2 and R3, respectively, which flip (interchange high and low halves of) their contents on XFER'S into and out of them, respectively. Third, special T codes are provided for register R1, which shifts its contents either

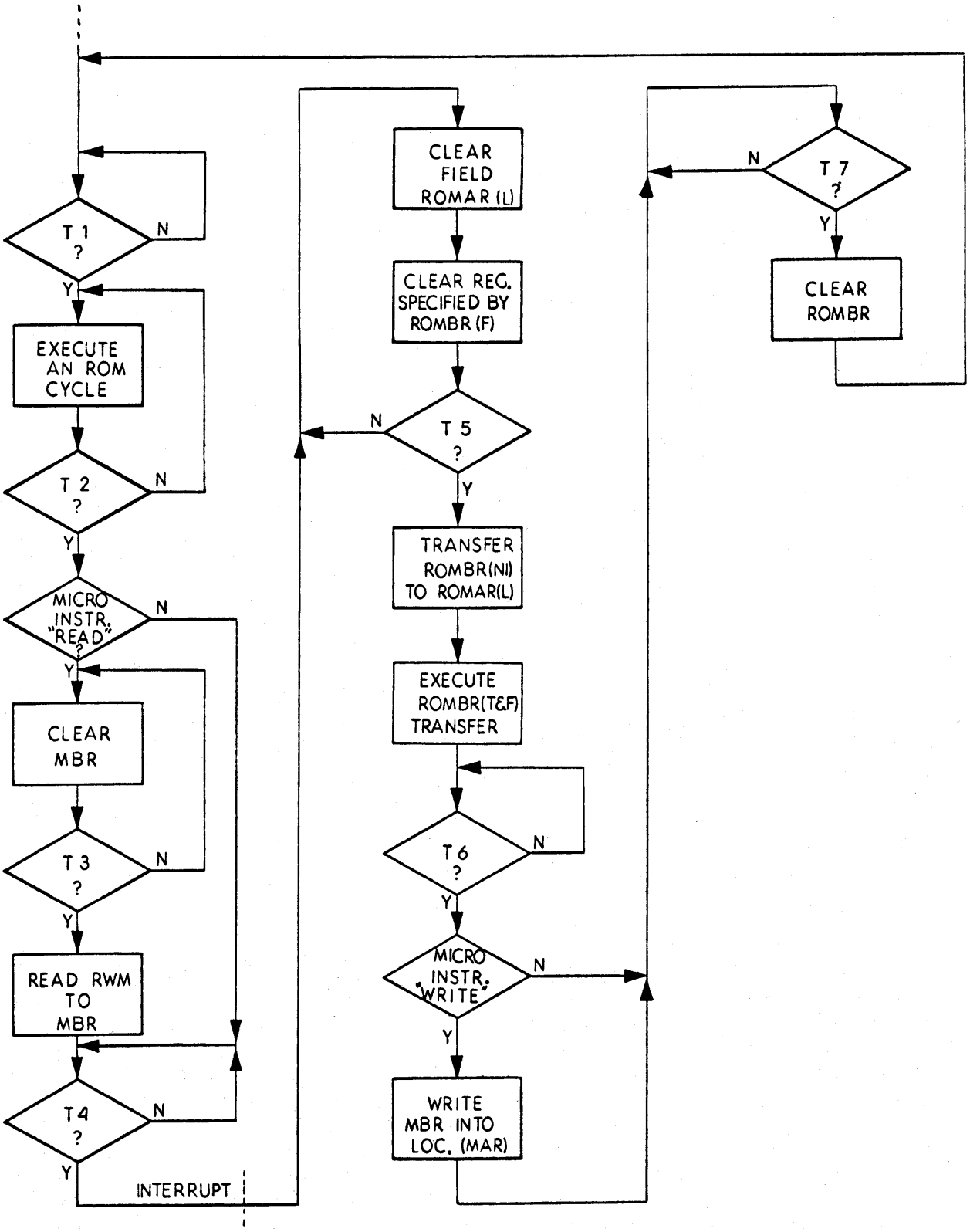



Figure 4. Normal mode simplified flow diagram.

left or right when it is XFERed to the bus. Of course, normal inputs and outputs are also provided for registers R3, R2, and R1. Fourth, a XFER with the ZERO TEST GATE specified by its F field can be used to provide a 1 output from these gates if R(T) is zero. In normal operation, this output is connected to the input of ROMAR bit 9 so that, if the NI field of the XFER is between zero and seven, the address in ROM of the next micro operation is effectively increased by eight. This conditional branch can be used to construct most conditional operations that normally appear in a computer's operation code list.

2. *XFC5: Transfer, and Clear Register R5, Micro Operation*

The ordinary transfer micro operation, XFER, never clears register R5, even when field F specifies this register, so the function of logical OR may be performed by successive XFER's into R5. For normal transfer of data into R5, a special micro operation XFC5 is provided to clear R5 whenever it is executed, whether or not the transfer which it also performs is in register R5.

3. *CLAD: Add Micro Operation*

The most elementary arithmetic micro operation, CLAD, clears R6 at time $t_4$ and, if field F specifies the ADDER, connects R5 and the bus to the two sets of adder inputs at time $t_5$. Hence, the CLAD micro operation places in R6 the sum of the contents of R5 and R(T).

4. *CLHA: Half Add Micro Operation*

Micro operation CLHA is like CLAD except that CLHA provides a carry inhibit signal to the ADDER. This causes the ADDER to perform the EXCLUSIVE OR logical operation that is useful for comparing the contents of a register with the contents of R5. The result in R6 is zero if the two registers contain identical bit configurations.

5. *CATO: Add, and Transfer Control on Overflow, Micro Operation*

Another modification of the CLAD micro operation is CATO. It forces bit 9 of the ROMAR to a 1 if the ordinary addition performed by CATO causes an overflow. This micro operation is a conditional branch that is necessary in subtract and multiply macro operations.

6 and 7. *READ and WRIT: Read and Write Micro Operations*

Communication with the RWM location addressed by the MAR is effected with the READ and WRIT micro operations. The WRIT micro operation executes an ordinary XFER into the MBR at times $t_4$ and $t_5$ and writes the MBR into the RWM at time $t_6$. The READ micro operation clears the MBR at time $t_2$ and reads into it from RWM at time $t_3$. Then it executes an ordinary transfer from the MBR at times $t_4$ and $t_5$.

8 and 9. *IPUT and OPUT: Input and Output, Micro Operations*

Communication with the external world is through the input and output registers, IPR and OPR. When the IPR is addressed by the T field of an IPUT micro operation, the machine waits at time $t_3$ until it receives an input ready signal (or IR), from the input reader, executes the ordinary transfer into R(F), and, at times $t_5$ and $t_6$, signals the input reader to begin reading in the next 10-bit word. It is assumed that the input reader withholds its IR signal until this operation is complete. In an analogous way, the OPUT micro operation causes the output writer to write out R(T) and inhibit further OPUT micro operations until its buffer is free to receive new data as indicated by the presence of an output ready, or OR, signal.

10. *PAUS: Pause Micro Operation*

PAUS waits for a restart signal, or RS, and then executes an XFER micro operation.

*Macro Operations*

Fifty-two macro operations were constructed using the 10 micro operations. Most of the macro operations were single address instructions in which the address used by each macro operation was stored in the location immediately following its operation code. Executing such an instruction involved adding one to the MAR to get the address of the operand, and adding two to its original value to get the address of the next macro operation.

## ORGANIZATION AND DESIGN OF THE DIAGNOSTIC SUBSYSTEMS AND DIAGNOSTIC PROCEDURES

### *Diagnostic Subsystems*

We shall now discuss design facets of the computer system just presented (Fig. 3) relating to the design of the diagnostic subsystems. A *diagnostic subsystem* is that portion of a digital system capable of effectively diagnosing another portion of the digital system. It has been shown that at least two mutually exclusive diagnostic subsystems are needed in self-diagnosable systems.

### *Optimization Criteria*

The DX-1 was made almost completely self-diagnosing (in the sense that it contained a small percentage of hard core) by choosing two diagnostic subsystems (A and B). The DX-1 was designed with the objective of minimizing hardware cost and hard core. In this example, hard core means those parts of the system common to all diagnostic subsystems; for example, the clock and power supply. The diagnostic subsystems were made identical for reasons of simplicity.

### *Partitioning the CPU*

The process of dividing the CPU into subsystems was quite straightforward because of the organization and implementation of the DX-1 with diagnosis as a principal design parameter and because of data flow simplicity inherent in microprogrammed machines. A simple visualization of the concept of partitioning the machine is given in Fig. 5.

All registers were divided by assigning the left halves to subsystem A and the right halves to subsystem B. The control was also divided into A and B halves controlling the A and B halves of all registers. The data bus consisted of two identical portions: the high or A bus which affected the left halves of all registers and the low or B bus which affected the right halves of all registers. Timing was treated in a similar manner by driving two timing rings: one for each subsystem. Arithmetic and Parity circuitry was divided equally between subsystems A and B. From these two subsystems (exclusive of the memories and I/O), we defined two diagnostic subsystems for the CPU. Each was associated with one subsystem and performed diagnosis on the other, or object, subsystem.

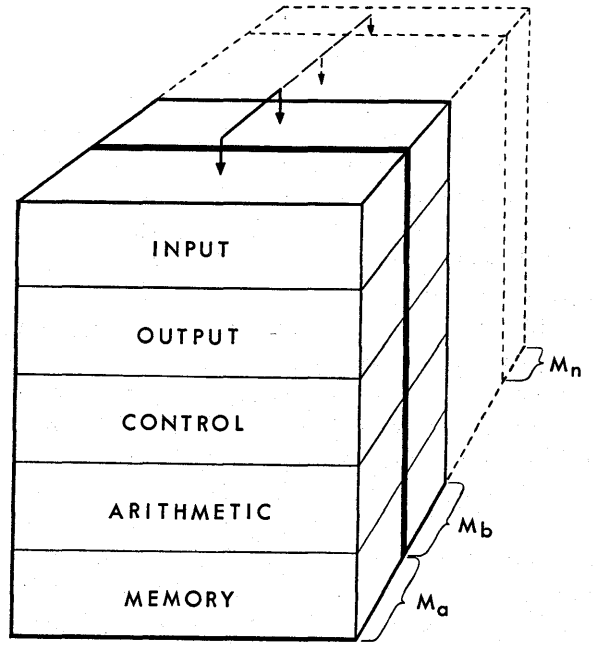



Figure 5. Concept of machine partition(s).

### *Diagnostic Subsystem Requirements*

The functional requirements of each diagnostic subsystem are restated as follows.

1. It must be capable of reading a set of identifiable numbers from some medium.
2. It must be capable of directing these sets of numbers appropriately.
3. It must be able to interpret the stored numbers to:
   (a) control switching between the diagnostic subsystem and the object subsystem;
   (b) make decisions affecting the next diagnostic subsystem step, based on the result of the last operation;
   (c) apply stored stimulus data to the object subsystem;
   (d) receive response data from the object subsystem;
   (e) initiate and sense the completion of each process in the diagnostic subsystem.

4. The diagnostic subsystem must have provision for indicating the physical location of suspected object subsystem malfunctions.

The ideal situation is that as much as possible of each diagnostic system be derived from each partition. This reduces the amount (and cost) of hardware that must be added; and also reduces hard core because the parts of the diagnostic system included in the partition are checked when the partition is checked.

This ideal was compromised in our example because our diagnostic systems were externally programmed. This tradeoff placed an input/output device in hard core but removed from hard core the ROM, RWM and all of their associated circuitry.

We assumed that the diagnostic subsystem (DSA) associated with subsystem A was diagnosing subsystem B, so $S_B$ was the object machine in this case. It must be kept in mind that the A and B machines were perfectly symmetrical so diagnosis of faults in A by B could be described in exactly the same way in which we describe the diagnosis of faults in B by A.

*Gating Requirements of DSA*

Additional paths were added to the data flow from the data bus of $S_A$ to allow the gating of bus data to:

1. CONTROL A
2. CONTROL B
3. The timing ring of $S_B$.

Only three paths were added to control register A since a single byte (equal to four information bits) for each of the F, T, and O fields would satisfy the needs of DSA. CONTROL B and $T^B$ required seven transfer paths from the A bus since the full contents of these registers were needed in the diagnosis of $S_B$.

In the diagnostic mode, the memories were disabled until they were to be diagnosed. That is, the ROMAR DECODER and MAR DECODER were disconnected from the ROMAR and MAR respectively; and the buffer registers were freed from interference by the memories. The clock (oscillator) was disconnected from the B timing ring.

*Diagnostic Commands*

The $IPR^A$ acts as the command code register when DX-1 is in its diagnostic mode. It is loaded from the A BUS at time $t_6^A$ and determines the destinations of the bytes placed on the A bus by the input device at times $t_7^A$, $t_8^A$, $t_1^A$, $t_2^A$ and $t_3$ which follow in sequence from the eight-place timing ring. Only three commands were needed, so a one-out-of-three command code was used; the left bit of $IPR^A$ being ignored in the diagnostic mode.

Command LA loads the CONTROL A register and the B timing ring $T^B$. Because the diagnostic mode needs to address only a few A register halves and execute only a few micro operations, only the four low order bits (or first byte) of the AF, AT, and AO fields of CONTROL A need to be loaded. The $T^B$ ring expects a bit of one of its eight positions, so it can be loaded with two bytes. Thus, command LA connects to BUS A four bits of AF, AT, AO, $T^B$ (first byte), and $T^B$ (second byte) at times $t_7^A$, $t_8^A$, $t_1^A$, $t_{2A}$ and $t_2^A$ and $t_3^A$ respectively.

Command LB loads the CONTROL B register. Control B's 20 bits are divided into five bytes of four bits each. Command LB connects the five bytes to the A BUS sequentially at times $t_7^A$ through $t_3^A$ inclusive. Each LA or LB code appearing on the A BUS at time $t_6^A$ must therefore be followed by five bits of binary information appearing on A BUS at the five succeeding clock times.

Command XQ executes a micro operation. Its code is read into IPR at $t_6^A$. Nothing happens at times $t_7^A$ through $t_3^A$; but at $t_4^A$ the half-register addressed by AF is cleared, and at $t_5^A$ the input of this half-register and the output of the half-register addressed by AT are connected to a bus. Late in time $t_5^A$, the XQ command clears not only the IPR but also CONTROL A, $T^B$, and CONTROL B. Other micro operations, of which CLHA (an exclusive-OR of some half-register and $R5^A$) is the most useful, are modifications of the fundamental XFER using only times $t_4^A$ and $t_5^A$.

Three simple abbreviations were used in our diagnostic programs. The mnemonic of an ordinary micro operation (XFER and CLHA for examples) means load its addresses and code with LA and LB commands and execute it with an XQ command. Compare $RN^A$ with $x$ means load $x$ in$R5^A$, CLHA $RN^A$ and $R5^A$, and XFER the result from $R6^A$ to the OPR. Branch on (non)zero to N means transfer external diagnostic programmer control to step N if the contents of OPR is (not) zero.

*Diagnostic Programs*

The diagnostic procedure was dictated by two fundamental principles. They were (1) diagnose faults first in those portions of the system having widest utilization in the system, and (2) the result of each diagnostic step should depend as little as possible on untested portions of the system and should depend as much as possible on the portion currently being tested. The first principle suggested that we test portions in the following order: (a) B BUS, (b) $T^B$ positions $t_4^B$ and $t_5^B$, (c) CONTROL B fields BF and BT, (d) arithmetic registers, (e) special gating, (f) ROM system, (g) RWM system (see Fig. 6). Some programs (suggested by principle two) are given for all but the last portion of the machine.

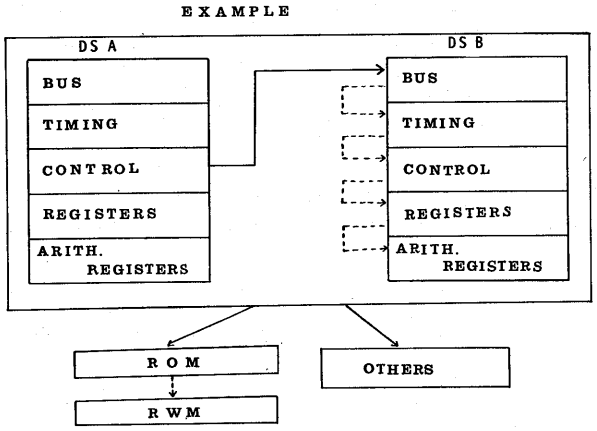



Figure 6. Example of bootstrapping.

(a) *B BUS Diagnosis*

A diagnostic program (Fig. 7) for detecting a stuck B BUS bit follows.

1. Load 11111 in $R3^A$.
2. XFER $R3^A$ into $R2^A$ by way of the B BUS.
3. Compare $R2^A$ with 11111.
4. Branch on non zero to 8. (B BUS struck to zero.)
5. Repeat 1 through 4; loading and comparing with 00000.
6. Branch on non zero to 9. (B BUS stuck to one.)
7. Branch to (b) Timing diagnosis. (B BUS not stuck.)
8. Stop and write out OPR. B BUS position indicated by a 1 is stuck to zero. Write out locations of cards that could cause this failure.
9. Stop and write out OPR. B BUS position indicated by a 1 is stuck to one. Write out locations of cards that could cause this failure.

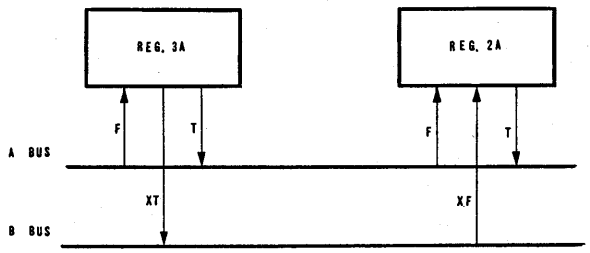



Figure 7. Bus diagnosis.

(b) *Timing Diagnosis (of $t_4^B$ and $t_5^B$)*

The sample diagnostic program for this portion of the machine indicates that we cannot distinguish among two failures (namely a permanent clear condition set up by $t_4^B$ and a missing signal to open the input gates on $t_5^B$); and this is the worst resolution obtained by any diagnostic routine. Note that diagnosis of failures assumes explicitly that only a single fault has occurred. A part of the timing diagnosis program follows:

1. Load each $RN^B$ (i.e., all B half-registers) with 11111.
2. Compare $R1^B$ and $R2^B$ with 00000.
3. Branch on both non zero to 9. (If and only if the branch does not occur, failures 8, 10, 1, 3 are possible.)
4. Compare $R3^B$ and $R4^B$ with 00000.
5. Branch on both non zero to 8. (If and only if the branch does not occur, failures 8, (10 and 12), 1, (3 and 5) are possible.)
6. Stop. The failure is number 8 or 1. Write out location of card(s) causing these failures.
7. Branch to 9.
8. Stop. The failure is number 12 or 5. Write out locations of card(s) causing these failures.
9. Compare $R5^B$ and $R6^B$ with 00000.

10. Continue to diagnose other failure numbers.

It should be noted that the diagnostic program used a "serial voting" technique to make the tests insensitive to a single bit of a register being stuck to 1.

(c) *CONTROL B Diagnosis*

Because each BT bit was connected to five AND gates and each BF bit was connected to ten AND gates, a serial voting technique was used to diagnose these fields, independent of a possible failure in the gates or registers they control. We placed 1 in a field's four bit positions successively, each time attempting to execute an XFER micro operation.

If three or more of the XFER's actually transfer data, we knew that the bit position common to the XFER's in that field was stuck to 1. Similarly, we tested a field for a bit position stuck to zero by trying each allowed bit configuration as an address of a XFER. If two or more of these XFER's failed to work, their common bit position must have been at fault.

The diagnostic programming example for the control portion of the machine diagnoses BT for a bit stuck to 1.

1. Load each $RN^B$ with 11111.
2. XFER $R1^B$ into $R2^A$ by way of the B BUS.
3. Compare $R2^A$ with 00000.
4. Branch to 7 on non zero.
5. Repeat 2 through 4 taking $R2^B$, $R3^B$, etc., in sequence.
6. Branch to BF one-stuck diagnosis.
7. Write out number of last register read from.
8. Stop if this is the third execution of this instruction and write out location of card causing this failure.
9. Branch to 2, taking next register in sequence.

Note that if the output record contains the numbers of three registers with a common address bit, then this bit of BT must be stuck to 1 even if one or two failures exist in the registers and gating involved.

(d) *Arithmetic Registers Diagnosis*

Knowing that the B BUS, control, and timing were operational, we proceeded to test and/or diagnose faults in individual bits of individual arithmetic registers. In the program for diagnosing this portion of the system, "AND $R6^A$ with $x$" means load $x$ into $R5^A$, pass $R6^A$ and $R5^A$ through the A half of the AND GATES on an XFER which puts their logical AND in the OPR. For example, if $x$ is 00001, then AND $R6^A$ with $x$ effectively places only the low order bit of $R6^A$ in the OPR. The complete diagnostic program for arithmetic register diagnosis follows.

1. Load each $RN^B$ with 11111.
2. XFER $R1^B$ into $R2^A$ by way of the B BUS.
3. Compare R2 with 11111.
4. Branch to 8 on non zero. (A bit is stuck to zero.)
5. Repeat 2 through 4 for $R2^B$, $R3^B$, etc., in sequence.
6. Repeat 1 through 5 loading and comparing with 00000.
7. Branch to ROM Diagnosis routine.
8. AND $R6^A$ with 11100.
9. Branch to 14 on zero.
10. AND $R6^A$ with 11110.
11. Branch to 13 on zero.
12. Stop. The fourth bit of the last register read from is stuck. Write out the location of its card.
13. Stop. The fifth bit of the last register read from is stuck. Write out the location of its card.
14. AND $R6^A$ with 11011.
15. Branch to 17 on non zero.
16. Stop. The third bit of the last register read from is stuck. Write out the location of its card.
17. AND $R6^A$ with 01111.
18. Branch to 20 on non zero.
19. Stop. The second bit of the last register read from is stuck. Write out the location of its card.
20. Stop. The first bit of the last register read from is stuck. Write out the location of its card.

Steps 14 through 20 of this program simply determine which bit of a register is stuck.

(e) *Special Gating Diagnosis*

The ADDER, AND GATE, and ZERO TEST GATE in $S_B$ in our example were checked and diagnosed by exhaustion techniques. Since we were deal-

ing with a four-bit device, this was practical. In the general case, where the number of tests required by exhaustion is unreasonable, we would have used tests which were generated through a process of computer analysis and simulation.

Diagnosis of the parity checking circuitry in $S_B$ was also done by exhaustion. Here, however, the output of the parity checking circuitry of $S_B$ was observable by DSA. This required that a gate be provided to direct the parity check output in $S_B$ to a bit position of the A BUS.

## SIMULATION RESULTS AND SUMMARY REMARKS

The DX-1 logic was built into a simulator which was designed to be processed on the IBM 7090. The simulator is capable of operating the machine in two distinct modes—normal and self-diagnostic. The normal mode was used to debug the original computer design, read-only memory programs, read-write memory programs and simulator design. It was also used to prove that the DX-1 is a computer. The normal mode simulation served its purpose in that it accomplished the above objectives and also proved that the DX-1 can perform all of the operations as specified in the earlier sections of this paper. The diagnostic mode was used to evaluate the philosophy of self-diagnosis and to investigate problems encountered in the application of this principle.

It was assumed that only solid single faults can occur. The diagnostic inputs and procedures were manually generated. Several changes in the design resulted from the logic simulation. The simulation did not consider the failures of ROM and/or RWM and their corresponding addressing systems.

The results of the simulation are as follows:

1. 94 percent of the DX-1 CPU logic was self-diagnosable.
2. 14.6 percent of the DX-1 CPU logic was added to achieve self-diagnosis.
3. 1 percent of the DX-1 CPU logic was hard core.
4. MTTD was less than 2 minutes for an input rate of 600 cards per minute.

Experience in simulation showed that the manual effort of preparing a diagnostic program is a long and tedious task and errors in judgment are always possible. Moreover, any modification in the program, because of a change in load assignment or control register coding, can become very formidable. Engineering changes could result in the scrapping of a computer program. These problems would become almost trivial if a computer could be used to generate the required diagnostic procedures or programs. Therefore, if the theory of self-diagnosing computers is to become practical for a family of machines, further study and development of machine generation of diagnostic procedures is necessary.

The degree of discrimination between the diagnostic results is highly dependent upon the ground rules followed in packaging the equipment. Packaging also has a major influence on the total time (MTTD) required to process the diagnostic program. This emphasizes the important fact that the design of a self-diagnosable computer must originate with the machine specification and must be treated as a principal design parameter.

The simulation study of the DX-1 has shown that both its design and the basic concept and philosophy of self-diagnosis are technically correct. However, many questions must be answered before the concepts of self-diagnosis and self-repair can become physical realities.

In conclusion, consider the following questions as samples:

1. How do we automate the generation of the diagnostic procedure?
2. How do we automatically initiate the diagnostic procedure?
3. What techniques applied singly and/or in combinations provide the great improvement in reliability and/or availability?
4. How can the design be optimized with respect to performance, cost, speed, and/or complexity? That is, what tradeoffs are possible?
5. Can the concept be extended to encompass other CPU's and systems as related to the concept of a "Central Diagnostic Computer"?
6. What is the effect on diagnosability, cost, and maintenance of partitioning into more than two subsystems?
7. What are the packaging ground rules?
8. What design notation is necessary for automation?

9. What will be the programming requirements?
10. To what extent could our assumptions be relaxed (e.g., types and number of assumed failures)?

## ACKNOWLEDGMENTS

The authors are indebted to: (1) P. W. Agnew, who was a principal contributor to the DX-1 design effort and coauthor of two of the three IBM technical reports which constitute this paper; (2) J. R. Belford for his active and sustaining support; (3) K. C. Dickerson, J. S. Jephson and V. Y. Lum for their simulation and evaluation of the DX-1; and (4) Prof. D. E. Muller for his illuminating discussions and comments on the theory of self-diagnosing and self-repairing computers.

## REFERENCES

1. R. E. Forbes, "The Case for Computers in Automatic Checkout Equipment," *Proceedings of a Seminar on Automatic Checkout Techniques,* Sept. 5-7, 1962, Battelle Memorial Institute, pp. 51-65.

2. K. Maling and E. L. Allen, "A Computer Organization and Programming System for Automated Maintenance," *IEEE Trans. on Electronic Computers,* vol. EC-12, no. 5, pp. 887-895 (Dec. 1963).

3. W. C. Carter, et al, "Design of Serviceability Features for the IBM System/360," *IBM Journal of Research and Development,* vol. 8, no. 2, pp. 115-126 (April 1964).

4. R. E. Forbes, D. E. Muller, and C. B. Stieglitz, "Automatic Fault Diagnosis," *AIEE Conference on Diagnosis of Failures in Switching Circuits,* Michigan State University, May 15-16, 1961.

# AN AUTOMATED INTERCONNECT DESIGN SYSTEM

W. E. Pickrell
*Automation Systems, Incorporated*

## INTRODUCTION

This paper describes a system for automatically designing and producing artwork for interconnect surfaces. This system consists of a number of computer programs which can be a subsystem of a general design automation effort. The interconnect design programs deal with the problems of artwork production and interconnection of electronic components frequently experienced in computer system design and construction.

One such system was developed for a former employer and spanned a period of about 18 months. During this period of development numerous experiments comparing the automatic interconnect design method to the manual method were performed. This paper discusses the results of these experiments along with some details of the programs and their operation. Before a discussion of automated interconnect design can proceed, some preliminary definitions are required.

The laminate of this paper is a series of single-sided etched boards or layers. Each layer has its own series of conductive interconnect paths. Layer-to-layer communication is accomplished through plated holes drilled through the layer. A group of layers are stacked and bonded together under heat and pressure to form a *laminated* interconnection board. Electronic components which require interconnection are mounted on the surface (or both sides) of the laminated board according to the scheme of the logic.

The "feed-thru" or "drill-thru" implies a second form of layer-to-layer or side-to-side communication in addition to component terminals. The system accepts either fixed or "floating" locations on the interconnect media.

## MANUAL INTERCONNECT DESIGN AND ARTWORK

As with many computer applications the beginning of the analysis is the "present" manual method of how things are done, since this situation is usually the one that requires some improvement in speed, accuracy, or cost. This holds true equally well for the interconnect design effort. A typical circuit board design task can be divided into the following steps:

1. Prepare a schematic wiring diagram.
2. Prepare a cover (master design pattern).
3. Place components in the best pattern.
4. Assign the external pins.
5. Check the artwork design.
6. Tape the artwork.
7. Check the artwork against the design layout.

8. Photographically reproduce (chronoflex) and reduce the artwork to desired size; produce necessary negatives and positives.
9. Deliver finished negatives to laboratory for fabrication.

Analysis of these steps shows that some of the work is particularly suited to automation with computer programs. If some or all of the functions could be performed for less cost, time and greater accuracy with computer programs, a significant gain in computer technology would be realized. The following section describes the program system and programs developed to meet that end, concluding with a section of remarks and statistical results for review.

## THE PROGRAM SYSTEM

The system includes the following general program areas:

1. *Generation* of the simulated physical composition of the surfaces in digital form (mapping or cover layer generation).
2. *Organization* of the data to be interconnected (routed). This may include: selection processes, determination of a minimal connection tree for a signal net, construction of various sort keys and selected sequences of sorts. These are all designed to facilitate and increase the yield of the routing phase which normally follows.
3. *Routing* of the organized data strings against the simulated physical environment of the surfaces taking into account any special constraints imposed by the hardware system being processed.
4. *Editing* and generating inputs to graphic devices such as plotters for covers and routed surfaces.
5. *Auxiliary* program providing secondary passes or continued processing such as:
   (a) routing update — to allow manual intervention,
   (b) net change — to pass first run failures onto alternate paths,
   (c) drill through — a subsequent run to the router phase if further laminate interlayer communications are required.

## THE PROGRAMS

This interconnect design system is modular and independent of other systems and, as such, operates on a *fixed placement* of components on the board as provided by a *Logic Assignment and Placement Subsystem* which precedes it. The programs can be best described by *function, input* and *output,* in the sequence they normally retain in operation.

1. *Title:* Cover Layer

   *Function:* Simulate the environment in digital form for interconnection (grid, size of board, obstacles, number of layers, external connectors, component pin arrays).

   Apply grid coordinates to all necessary data.

   Provide for special requirements such as clock, voltage singularities, deletion of grounds, etc.

   *Inputs:*
   (1) String list (signal nets with component-pin identifications) as assigned and placed.
   (2) External connectors available.
   (3) Obstacle descriptors, drill-thru locations, module or chip arrays.
   (4) Signals to be deleted and signals to special layers.

   *Outputs:*
   (1) Cover layer tape (include all routine obstacles).
   (2) Special layers signal nets tape.
   (3) General layers signal nets tape.
   (4) External connectors mesh.
   (5) Printer plot of cover layer; errata list.

2. *Title:* Organizer

   *Function:* Inspect each signal net. Select an external where required. Produce the minimal tree for inter-connect based on straight-line distances.
   Establish sort keys for data organization for the Router. Compute slope class, distance, number of pins/net and signal priority.

   Sort as directed.

*Inputs:*
(1) Signal nets tape.
(2) Control data for external selection.
(3) Slope class criteria.
(4) Sort criteria (sequences, keys).

*Outputs:*
(1) Connect input tape(s) in specified sort.
(2) Class statistics, errata, external connection list.
(3) External connection punched cards (optional).

3. *Title:* Router

*Function:* Set the environment to memory. Route the interconnect of two points according to the organization, constraints and bounded routing area. Record the path if successful. Record the event of a failure for further routing trials on subsequent layers or update processing. Set a coordinate tape for a plotter editing. Record path in core as a layer history.

*Inputs:*
(1) Cover layer tape.
(2) Connect input tape.
(3) Media parameters (grid, number layers, etc.).

*Outputs:*
(1) Coordinate lists (for each layer) tape.
(2) Fail list (unconnected pairs of pins) tape.
(3) Layer or side history (cover + paths for each) tape.
(4) Statistics and layer listing.

4. *Title:* Plot Editors

*Function:* Translate data from cover layer tape and coordinate list tape into formats required for the plotting devices (i.e., CALCOMP, GERBER).

*Inputs:*
(1) Cover layer tape.
(2) Coordinate list tape.
(3) Control data (scaling, conversion factors, etc.).

*Outputs:*
(1) Plotter tapes (cover + layers) either magnetic or paper, as required.
(2) Signal name tape (optional).
(3) Line length lists (by layer and by signal).

## AUXILIARY PROGRAMS

1. *Title:* Router Update

*Function:* Reconstruct cover layer and routed history from the initial routing run. Process manually derived inputs through erasure and rerouting to attain total interconnect for plot editing and artwork.
Provide sufficient errata reports on fails or invalid conditions to allow feedback through the man-machine loop.
Provide outputs for checking continuity of signal interconnection.

*Inputs:*
(1) Cover layer tape.
(2) Coordinate list (1st run) tape.
(3) Layer parameters
(4) Update list (deletions, insertions).

*Outputs:*
(1) Coordinate list tape (updated).
(2) Listing of connections updated, fails and errata.

2. *Title:* Net Change

*Function:* Reconstruct layer history (1st run) and try to route from a list of alternate paths. Use only original failures where alternate paths are available to interconnect the signal net.

*Inputs:*
(1) Layer history tape (1st run).
(2) Coordinate list tape (1st run).
(3) Net prep tape (a list of alternate routes).

*Outputs:*
(1) Layer history tape (updated).
(2) Coordinate list tape (updated).
(3) Fail list tape.

3. *Title:* Drill Thru

*Function:* Apply a subsequent run to the router program with the capability of using "drill thru" locations for layer-to-layer communication.

*Inputs:* (1) Layer history tape (1st run).
(2) Coordinate list tape (1st run).
(3) Fail list tape (1st run).

*Outputs:* (1) Layer history tape (updated).
(2) Coordinate list tape (updated).
(3) Fail list tape.

## RESULTS

The following table illustrates the degree of success in using the design system. Five different laminate boards (in five different hardware systems) were designed. In all cases, no drill-thru pass was used. In one case the manual update feature was employed to attain 100 percent interconnection.

| Components | Board Size | Layers | Inputs | Paths | Fails | Percent |
|---|---|---|---|---|---|---|
| Modules | 346 × 139 | 6 | 593 | 518 | 75 | 87 |
| Modules | 268 × 168 | 7 | 1006 | 965 | 41 | 96 |
| Chips | 70 × 84 | 6 | 685 | 670 | 15 | 97 |
| Chips | 88 × 105 | 5 | 712 | 659 | 53 | 91 |
| Chips | 98 × 78 | 4 | 312 | 304 | 8 | 97 |

Additional items of importance derived from this study are:

1. "Chips" on a board result in a higher yield of interconnect in a shorter period of time. This physical configuration offers a better distribution of pins about the board, simplifying the interconnect problem.
2. Presentation of the data, pairs of points, in the following sequence is the most optimal for the routing phase:
   (a) Classification by slope of the straight line connecting the two points.
   (b) Straight line distance between the two points.
   (c) Minor sorts on signal priority and number of pins in the signal net.

Since the deterministic method used in the routing phase does not yield 100 percent interconnect design, in most cases, manual intervention is required. This phase involves an analysis of the failures against the plotted layers. Subsequent introduction of the failures into the machine solution is achieved through the preparation of *update* and *insertion* inputs to the Router Update Program.

Direct comparisons of manual design versus the man/machine method for multilayer laminates have shown the following results:

1. A calendar time compression to 5-10 days for the design cycle.
2. Cost reductions to the project of 50 percent or more.
3. An accuracy or reliability factor unobtainable by manual approaches.
4. Shorter line lengths for etched paths.
5. By-product provisions for automated tooling for board manufacture.

*Figure* 1 illustrates in flow form the programs (input/output) comprising the basics of a design system.

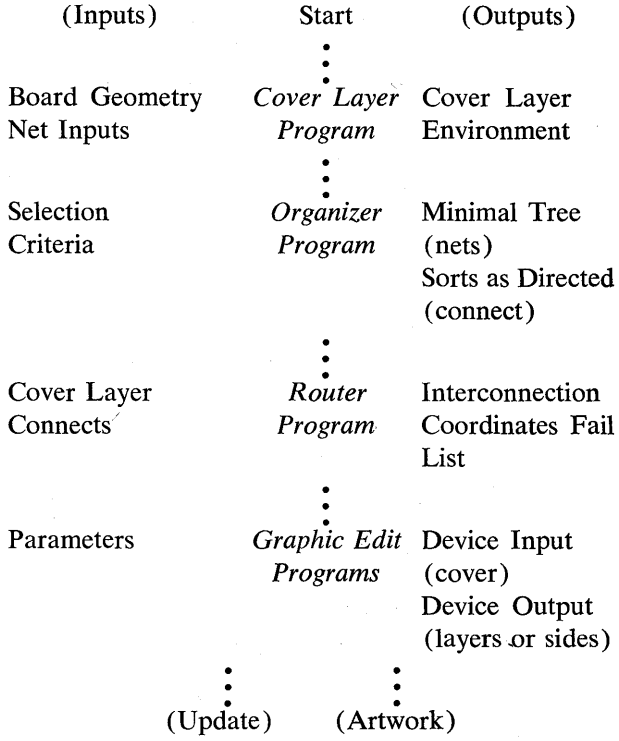



Figure 1

*The environment* of the interconnect medium must be completely described in the following areas: module or chip pin arrays, drill-thru or via locations and identifications, dimensions of the layer or board ( *it is highly desirable* to design the board to an equally-spaced grid in both directions), cutouts, mounting, or fabrication areas where routing is not allowed, and the external connector array where signals enter or leave the board.

In general, for a multilayered laminate, the cover

layer (environment) is constant for all cases. However, some newer manufacturing techniques are being used ("Post" or "build-up") that permit deletion of pins which are net terminals in all layers below that of the interconnection. The concept here is that this deletion "opens up" further areas for interconnect routing as the process proceeds. This is, of course, highly desirable for boards of high interconnect density and tightly packed pin arrays.

This process requires "customizing" the router program to reflect the deletions. In additon, if one desired pen plots as design or update tools, it would require the generation of $n$ cover layers for an $n$-layer laminate.

For the two-sided board, one would normally have two cover layers reflecting the environments on the respective sides.

The function of a *Cover Layer Program* is to generate this environment and append all coordinates to the net input list for subsequent program processing.

Organization of the data to meet changing requirements of board design, external usage, size, singularities of environment, and so on can very well be the key to the degree of success attained in the routing phase.

The methods employed in the router phase of a system such as this are *not* iterative. Thus the sequence of presentation of inputs to the router program is extremely important.

Experience in several different environments indicates that so far a typical, practical approach might include some of the following:

1. Inspect the environment for singularities which can prove useful in selecting a set of criteria. For example, a well-distributed pin population on a square board might suggest the use of slope classes in equal angle segments.
2. Also with respect to slope, for laminates it is generally good to equate the number of classes with the number of layers.
3. Specification of pairs of pins selected for interconnection of an entire signal (net) is normally the result of a *minimal tree* calculation based on straight line distances pin-to-pin.
4. Selection of external connectors available for signals entering or leaving the board can also be accomplished during the *tree* phase.
5. Other considerations in this stage prior to final sorting are number of pins in the net (this can be important in an update process), and special priorities to be considered in the routing phase.

   At this point, an additional function of an *Organization Program* is to monitor the soring of the data for which the various keys mentioned above have been included. For example, a typical sort sequence preparatory for routing might be:
   (a) Slope class (numeric).
   (b) Distance (in grid units).
   (c) Number of pins/net.
   (d) Priority.

The *routing* phase of the design system generally has little freedom in changing the sequence of processing the input data. However, a moderate amount of override capability should exist. For example, due to statistics from the organizer phase, it is indicated that several classes should be attempted on layer 1, rather than the single class originally planned. At this point it would be desirable to have the capability to modify the presentation with, say, input cards rather than returning to the organizer phase.

A rather important aspect of a routing program is the ability to "customize" the routing algorithm routines to meet particular requirements of a board or laminate. Certain restrictions or constraints can be imposed in this area which would be germane to a certain type of environment. For example, spacing between etched path and adjacent pad areas (a terminal where an interconnect already exists) might be critical on a board and require a prohibitive action. To reduce computer processing time, only the required logical inspections of the routing space are performed and these are functions of the particular board.

Outputs from a router program are normally:

1. Descriptions of the interconnect paths which can be edited for a plotting device. In previous systems these were in the form of a "from-to" terminal or pad identifications accompanied by a coordinate chain which represented every unit cell in the interconnect path.

2. A listing of all input pairs which the program failed to layout.

Finally, programs which perform the necessary translation of data into plotting device input format are required. There are several ways to develop the graphic output; one of these is to edit the cover layer (environment) and router output separately. If a pen plot is used, the cover and interconnect can be plotted in superimposition using contrasting colors for clarity. Size or scale of the plot may be as desired within the limits of the device.

The "ultimate" at the moment is production of final artwork for the interconnection surfaces. This requires high resolution and optical capabilities in the plotting device.

Within the realm of design automation, automatic interconnect design is in its infancy. Many new things are being done presently, and many more will follow in the months ahead; some of these will include: automatic board design, refinements in the analysis of data organization, and new "customized" router algorithms to accommodate advanced manufacturing techniques.

# SYSTEMATIC DESIGN OF AUTOMATA

J. P. Roth
*IBM Watson Research Center*
*Yorktown Heights, New York.*

The subject of this paper is a system of programs to aid in the logical design of automata. Figure 1 depicts the experimental system as it presently exists within IBM. There are essentially two internal

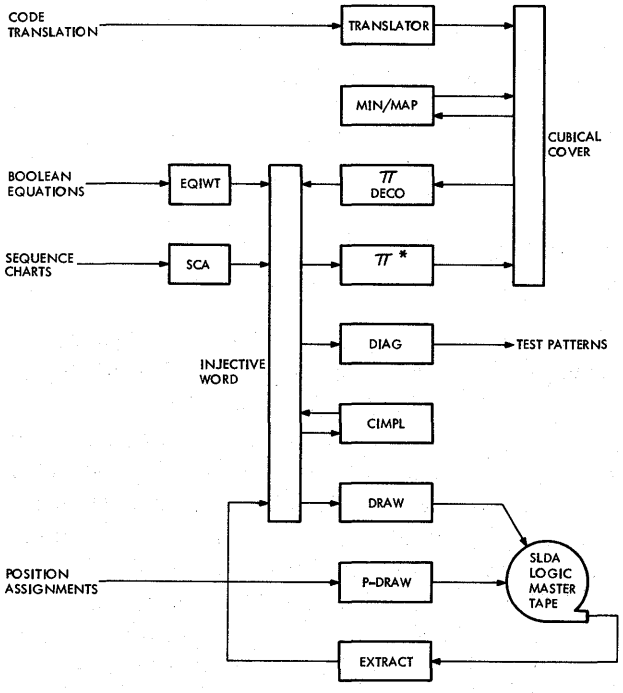



Figure 1. The logic automation complex.

formats for the system, one called the injective word as shown in Fig. 3. which in essence specifies all the logic blocks in a circuit and how they are linked together. The other is termed a cubical cover (or ON-OFF-ARRAYS) which is a means of describing the behavior of the circuit as if it were a two-level circuit consisting of ANDs followed by ORs (or vice versa). See references 1 and 2 for a more detailed description of these notations.

The programs of the system, the principal ones being shown in boxes (Fig. 1), may be thought of as transformations of these formats into themselves. There are two exceptions. They are the Sequence Chart Analyzer SCA and the program EQIWT, the equation-to-injective-word translator.

We shall first discuss the sequence chart analyzer; a sequence chart is shown in Fig. 2. This is a sequence chart for the operation of the instructions Floating Point Add, Subtract or Compare for an early version of MODEL 60 for System/360. It will be observed that across the top of the chart appear intervals labelled T1, T2, . . . , T10. These refer to time intervals and admit of many interpretations. Operations are written above horizontal line segments in one of the "time columns". Immediately to the left are written the "immediate" conditions necessary for its execution. For example in column 5 the operation Set Condition Register is performed,

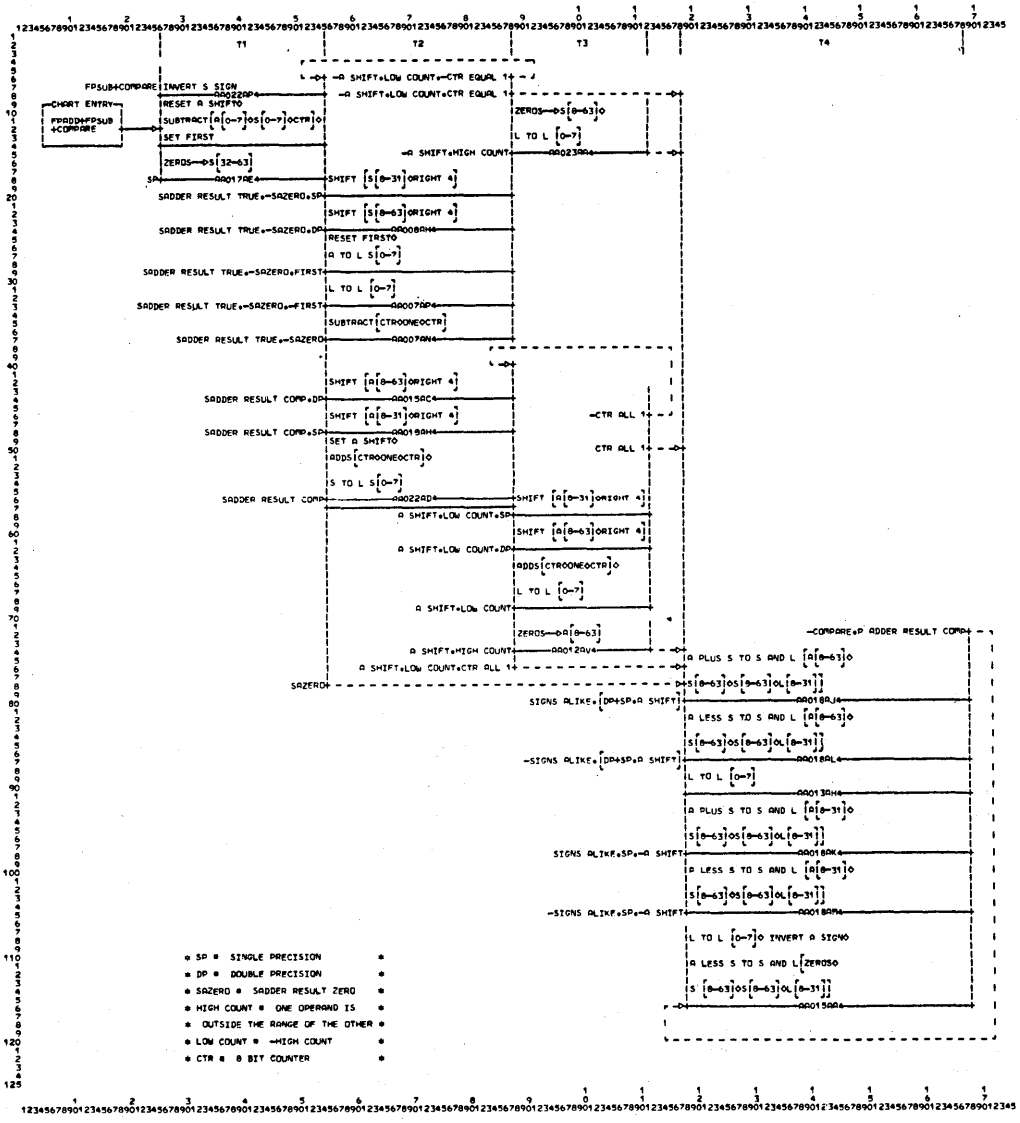
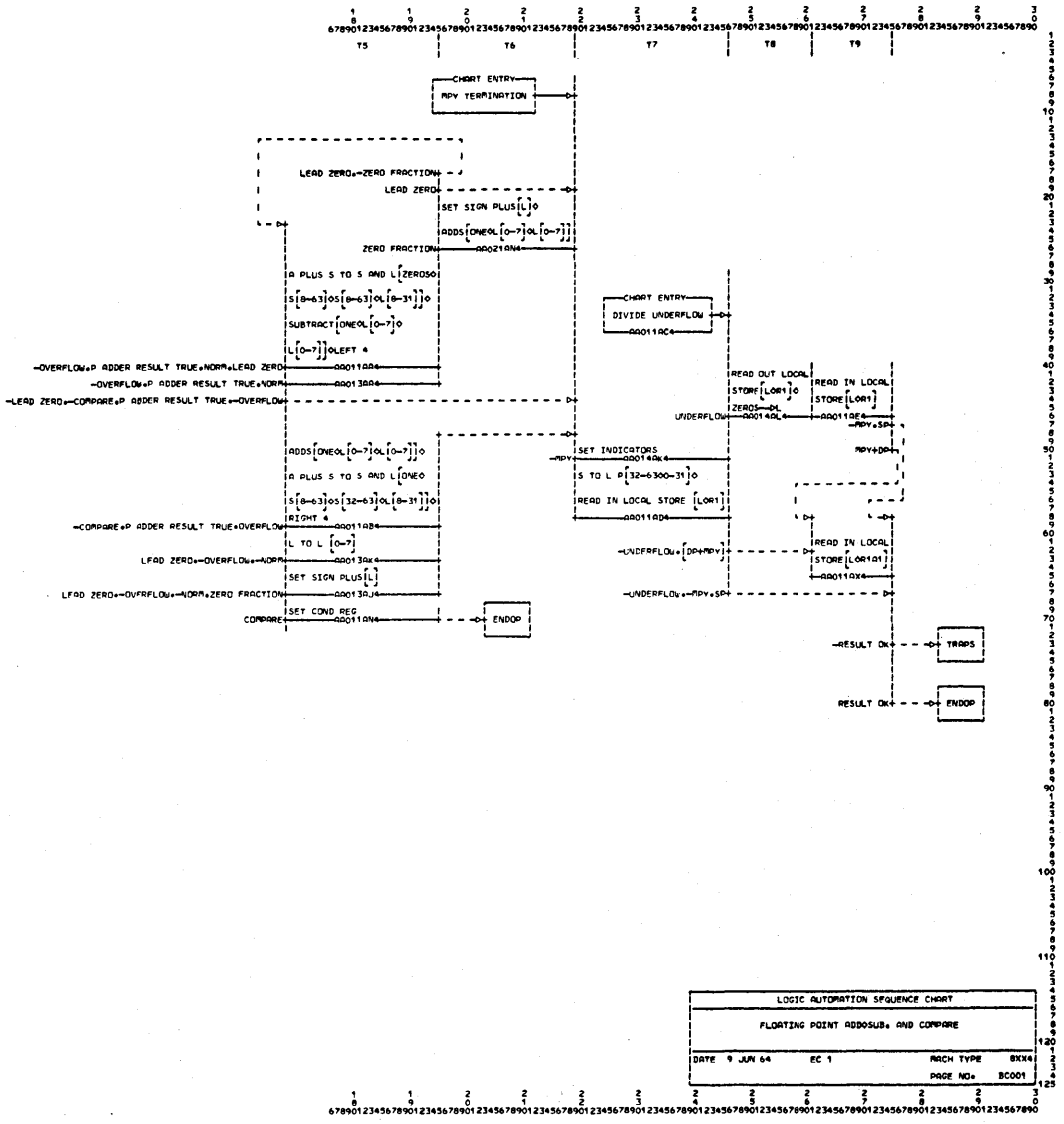



Figure 2.

when the sequence chart is "operative" in this area, when the condition holds. The given sequence chart becomes operative at "Chart Entry Conditions" such as (on the far left of the chart) FPADD + FPSUB + COMPARE. Thence in time T1, in the T1 column some, or all, of the operations such as INVERT S SIGN are performed provided the immediate condition written to its left, such as FPSUB + COMPARE, are satisfied. If no immediate condition is there written then the operation is automatically performed. Thus several operations may be simultaneously executed. Thence one moves to the next time interval or else to the next specified interval. For example if after performing any of the operations at time T1 if SAZERO (Serial Adder is Zero) then one jumps to time interval T4. Whichever is the case one proceeds into other phases of the chart. For example after an operation in time T4 has occurred then the next operation and the next time are determined by the immediate conditions written to the left of the line segments emanating from the right half of the T5 column. For example if LEAD ZERO—OVERFLOW and —NORM and ZERO FRACTION is true then the operation SET SIGN PLUS [L] is performed. Thus any set of conditions are prescribed a "path of operations" on the sequence chart ending ultimately in an ENDOP which means end of operations.

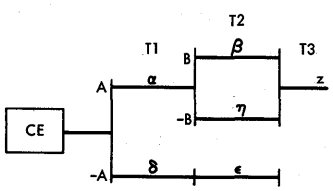
It is thus seen that the sequence chart is kind of sequenced flow-chart of machine operations as de-




scribed for some particular data flow structure of a machine.

The Sequence Chart was originally designed by Stanley Pitkowski and subsequently formalized by the Logic Automation Group mainly by J. M. Galey, P. N. Sholts and Jere Sanborn. Numerous abbreviations for indirect addressing are employed in the description.

The Sequence Chart Analyzer (SCA) first produces a hard copy of the sequence chart itself as illustrated by Fig. 2. The form of this chart is subject to updating and can be changed at will. The second output from SCA is a injective word which prescribes for each gate the total conditions for which it is to be open or to be closed. Figure 3 is one page of several pages of the injective word produced by SCA for the sequence chart of Figure 2. The meaning of this form may be understood by the following example: at left is a circuit and at right is the corresponding injective word.




$\alpha^* = A, CE$ AND
$\alpha = \alpha^*, T1$ AND
$\beta^* = B, \alpha^*$ AND
$\beta = \beta^*, T2$ AND
$\delta^* = -A, CE$ AND
$\delta = -A, T1$ AND
$\epsilon = \delta^*, T2$ AND
$\eta^* = A, -B, CE$ AND
$\eta = \eta^*, T2$ AND
$z = A, T3, CE$ AND

```
FLOATING PT. ADD, SUB, COMPARE  - ALIGNMENT OF FRACTIONS -

INJECTIVE WORD

-001BC0 = 5A1E,T2,LOWCNT                                    A   3
-002BC0 = 9T,9A1E                                           A   2
 003BC0 = T1,9Z0007,SAZERO                                  A   3
 003BC1 = 5A2E,T2,LOWCNT,CTREQ1                             A   4
 003BC2 = 7A6E,T2,LOWCNT,CNTAL1                             A   4
 003BC4 = 9T,CNTAL1                                         A   2
-1A1E   = FPSUB,COMPAR                                      OR  2
-1A1    = 2A1E,1A1E,9Z0005                                  OR  3
-1A2    = 2A1E,9Z0005                                       OR  2
 1A3E   = -SP                                               N   1
-1A3    = 2A1E,1A3E,9Z0005                                  OR  3
-2A1E   = FPADD,FPSUB,COMPAR                                OR  3
-3A1    = 3A1X,9Z0006                                       OR  2
-3A1X   = SADRT,SP,9Z0028                                   A   3
-3A2    = 3A2X,9Z0006                                       OR  2
-3A2X   = SADRT,DP,9Z0028                                   A   3
-3A3    = 3A3X,9Z0006                                       OR  2
-3A3X   = SADRT,FST,9Z0028                                  A   3
-3A4E   = SAZERO,FST,2A1E                                   OR  3
-3A4    = 3A4X,9Z0006                                       OR  2
-3A4X   = 3A4E,SADRT                                        A   2
-3A5    = 3A5X,9Z0006                                       OR  2
-3A5X   = SADRT,9Z0028                                      A   2
-3A6    = 3A6X,9Z0006                                       OR  2
-3A6X   = 9Z0007,SADRC,DP                                   A   3
-3A7    = 3A7X,9Z0006                                       OR  2
-3A7X   = 9Z0007,SADRC,SP                                   A   3
-3A8    = 3A8X,9Z0006                                       OR  2
-3A8X   = 9Z0007,SADRC                                      A   2
-5A1E   = CTREQ1,5                                          OR  2
 5A2E   = -5                                                N   1
 5A3E   = -HICNT                                            N   1
-5A3    = 5,5A3E,9Z0012                                     OR  3
-5      = 9Z0013,9Z0014,9Z0015,9Z0016,9Z0017                OR  5
-7A1E   = LOWCNT,9A1E,SP                                    A   3
-7A1    = 7,7A1E,9Z0012                                     OR  3
 7A2    = T3,7A6E,LOWCNT                                    A   3
-7A3E   = LOWCNT,9A1E,DP                                    A   3
-7A3    = 7,7A3E,9Z0012                                     OR  3
-7A4E   = LOWCNT,9A1E                                       A   2
-7A4    = 7,7A4E,9Z0012                                     OR  3
-7A5    = 7,5A3E,9Z0012                                     OR  3
 7A6E   = -7                                                N   1
-7      = 9Z0018,9Z0019,9Z0020                              OR  3
 9A1E   = -CNTAL1                                           N   1
 9T     = -9Z0034                                           N   1
 T2J    = -001BC0                                           N   1
 T3J    = -002BC0                                           N   1
 T4J    = -9Z0035                                           N   1
 9Z0005 = -T1                                               N   1
 9Z0006 = -T2                                               N   1
```

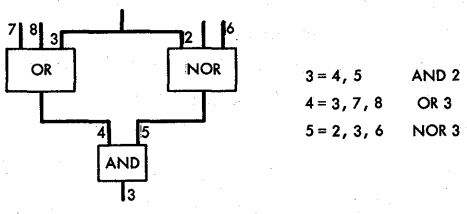
Figure 3.

Each line or argument is given a label; each line, for example line 3, emanating from a box is expressed as a function of the lines entering the box, together with the name of the logical function which the box performs, and the number of inputs, e.g.

$$3 = 4,\ 5 \qquad \text{AND } 2$$

This method of description of course allows feedback in the circuit.

The method of analysis for SCA will be illustrated in the next figure.




The injective word output of SCA is fed directly into the program CIMPL, an acronym for circuit implementation. This program accepts an injective

word described by any set of logic blocks and converts it into an SLT injective word, i.e. one composed of SLT logic modules and obeying fan-in, fan-out powering, etc. of this particular technology (SLT stands for Solid Logic Technology.)

This program was outlined in architectural form by John Earle and programmed by Peter Schneider, now of the University of Wisconsin; Michael Galey, IBM San Jose; Jere Sanborn, IBM Poughkeepsie; and others.

The first operation which CIMPL performs is to search for identical gates or inverters. It eliminates all but one of these and fans out from it. It then does the SLT implementation in the low, medium or high-speed circuit families so as to preserve the general logical structure specified by the designer on the Sequence Chart as well as the delays inherent in the structure. It does this implementation within the SLT circuit constraints manipulating the logic locally so as to satisfy the fan-in constraints and inserting powering whenever called for by the loading equations. It does the fan-in and powering manipulation so as to add the minimum amount of additional hardware to satisfy the constraints while preserving the general logical structure. The program then assigns the circuit types in the specified circuit family and produces an output which can feed directly into DRAW, the program which partitions the injective word produced into pages and assigns print positions for each page, in sum, in effect to draw the ALD sheets (ALD automated logic diagram) for Solid Logic Design Automation; it in fact generates the SLDA logic master tape.

Another input to CIMPL is Boolean equations: a preprocessor called EQIWT (equation to injective word translator) accepts Boolean equations and converts it to the injective word format. This gives the designer the ability to convert his Boolean equations directly into hardware. This path within the logical automation complex was used in Hursley, England, particularly by K. A. Duke, in the design of the Model 40 of System/360. Specifically the ALU, the Arithmetic and Logic Unit was so designed.

An alternate program called Position Draw or P-DRAW was written by Galey: this accepts an injective word in which the pages and print positions have been specified by the designer in a list structure. It generates the ALD sheets in accordance with the pattern prescribed by the engineer and also generates the Logic Master Tape of Solid Logic Design Automation. It thus saves the designer from making careful drawings for the keypunching operation and substantially cuts down on the magnitude and difficulty of the keypunch operation itself.

Figure 4 shows one page of 24 ALD sheets produced by SCA followed by CIMPL followed by DRAW. These ALD sheets would have, in the manual version, been originally computed and hence drawn by hand for insertion into the Solid Logic Design Automation System. Total IBM 7094 time for this operation was about 15 minutes.

Another entry into the system of logic automation programs besides the sequence chart and Boolean equation is through a translation code, as shown in Figure 5. This is the translation from NPL to typewriter coding. The code translation program assembles this into a set of Boolean functions, one for each output. This is then formulated as a two-level minimization problem with many outputs.

MIN is a program of the extraction algorithm[2, 3] which works with cubical covers and has a domain of applicability much wider than those algorithms using canonical terms. For the NPL-typewriter code translation a minimum was obtained. This minimum two-level solution was then converted by a decomposition algorithm DECO[2, 4, 5] applied by a human computer (the program was unavailable at the time). This result was then fed into CIMPL and DRAW to produce on 13 ALD sheets an SLT design of the code translator. Total machine time did not exceed 10 minutes on the IBM 7094.

Two more programs of the system will be mentioned. These programs can be run in conjunction with the other programs of the system. The first is called $\pi^{*}$[2, 3] and is a program for analyzing circuits; precisely it accepts an injective word and translates it to a cubical cover, i.e. a normal form expression, for each output of the circuit in terms of its primary inputs. This program requires that all feedback loops be cut. It enables us to apply minimization procedures to already designed circuits in an effort to achieve cost reduction.

This approach is especially useful in the case of design of low-cost circuits for which many copies will be made and wherein the savings in diodes or transistors are particularly significant. These programs have had extensive use in IBM Endicott, e.g. on the MICR (magnetic ink character recognition)

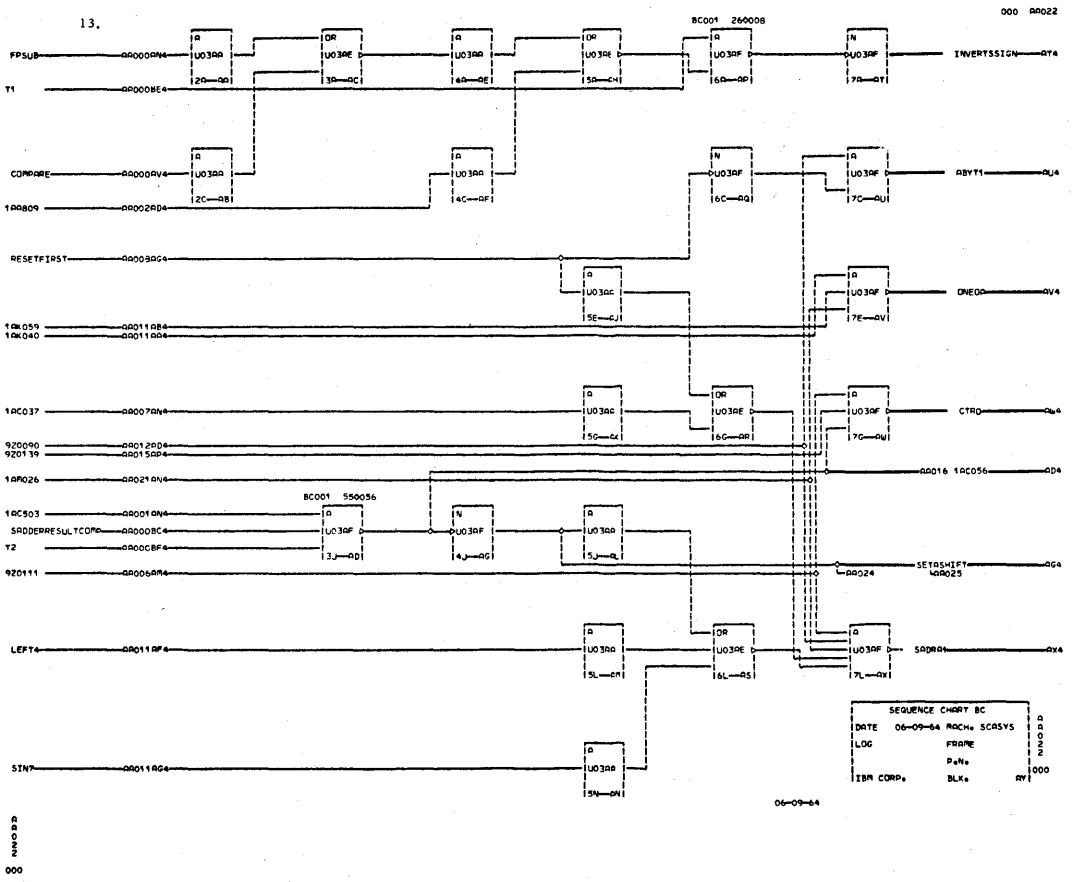



Figure 4.

circuitry by Billy N. Carr, on the IBM 1030, a device to transmit data from terminals, and on the Serial Wire Printer. In each case, $\pi^*$ was followed by MIN followed by factorization and thence (in two cases) by CIMPL. Incidentally mistakes by the engineer were detected in the design and corrected. In each case a very small amount of machine time was used.

Another program which has had extensive usage is a set of diagnostic programs, labelled DIAG in Fig. 1 This is a program which accepts a circuit in injective word format and produces a set of input patterns termed tests which when applied to the circuits is capable of determining whether or not a failure has occurred of a given variety. The variety treated by the program basically is a failure for each line wherein the line may be fixed in value, either stuck-at-1 or stuck-at-0. This program was used specifically in the design of the Memory Protect and Relocation MPR for the 7094 configuration going to M. I. T. By an addition of only 2.5 percent extra hardware it was possible by means of the tests generated for this piece of hardware to detect *any* single failure of the stuck-at-1 or stuck-at-0 variety. Approximately one hour was required of 7090 time to compute the tests. DIAG was a predecessor of the SLT set of programs FLT (Fault Location Technology) used on some models of System/360, written principally by Frank and Martha Evans.

A new program is being planned[8] for an improved algorithm called d-DIAG for generating diagnostic tests. This algorithm is substantially faster than other known methods and is based essentially on a calculus of injective words rather than by a calculus of cubical covers, as is DIAG. Reference 8 describes this algorithm in considerable detail and makes comparisons with previous methods.

A profitable way to utilized this complex of design programs is in the redesign mode: A computer system is designed for one technology and it is desired to redesign it in another. For this purpose a program EXTRACT is needed to take from the SLDA Logic Master Tape the logical essentials to

NPL Code

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | P |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |

BCD Code

| 1 | 2 | 4 | 8 | B | A | C | W/M |
|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 0 | 1 | 0 |

Figure 5.

be fed into the analysis and synthesis programs. This process has been used partially automated, that is to say, the EXTRACT program as it now stands requires another routine before being completely automatic. This program is in an experimental state.

## FUTURE SYSTEM

The Iverson notation has been used to present a very complete specification of the architecture of the IBM System/360 (Ref. 9) and this immediately opens the possibility of adjoining this description to our Logic Automation Complex. The design for the machine would thus be specified initially by means of a program or system of programs written in the Iverson notation. For each model of System/360 the data flow would also be given, essentially as a list of facilities available plus their interconnections. This part of the experimental design system remains to be done.

## REFERENCES

1. J. P. Roth, "Minimization Over Boolean Trees," *IBMJ of Res. & Dev.,* vol. 4 no. 5, pp. 543-558, (Nov. 1960).

2. R. E. Miller, *Switching Theory,* John Wiley & Sons, Inc., New York, 1965.

3. J. P. Roth, "Algebraic Topological Methods in Synthesis," *Proc. of an Inter. Symp. on the Theory of Switching,* 2-5 April 1957, Part 1, Harvard University Press, Cambridge, Mass., 1959.

4. J. P. Roth, and R. M. Karp, "Minimization over Boolean Graphs," *IBMJ of Res. & Dev.,* vol. 6 no. 2, pp. 227-238, April 1962.

5. R. M. Karp, F. E. McFarlin, et al, "A Computer Program for the Synthesis of Combinational Circuits," *Proc. of 2nd Annual Symp., Switching Circuit Theory and Logical Design,* AIEE, Oct. 17-20, 1961, New York.

6. J. M. Galey, R. E. Norby, and J. P. Roth, "Techniques for the Diagnosis of Switching Circuit Failures," *IEEE Transactions on Communications and Electronics,* vol. 83 no. 74., Sept. 1964, pp. 509-514.

7. W. C. Carter, R. Preiss, et al, "Design of Serviceability Features of IBM System/360," *IBMJ of Res. & Dev.,* vol. 8 no. 2, April 1964, pp. 115-126.

8. J. P. Roth, "Algorithms for the Mechanization of Design I," Diagnosis IBM Resarch Paper RC-1924 (Oct. 1964). To appear in *IMBJ of Res. & Dev.*

9. A. D. Falkoff, K. E. Iverson, and E. H. Sussenguth, "Formal Description of System/360," *IBM Systems Journal,* vol. 3, no. 3, pp. 198-262 (1964).

# AMERICAN FEDERATION OF INFORMATION PROCESSING SOCIETIES (AFIPS)

211 E. 43rd Street, New York 17, New York

*Officers and Board of Directors of AFIPS*

*Chairman*

DR. EDWIN L. HARDER*
1204 Milton Avenue
Pittsburgh 18, Pennsylvania

*Treasurer*

MR. FRANK E. HEART*
Lincoln Laboratory—MIT
P. O. Box 73
Lexington 73, Massachusetts

*Secretary*

MR. MAUGHAN S. MASON
Thiokal Chemical Corp.
Wasatch Division—Mail Stop #150
Brigham City, Utah 84302

*Chairman-Elect*

DR. BRUCE GILCHRIST*
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York

*ACM Directors*

MR. J. D. MADDEN
ACM Headquarters
211 East 43rd Street
New York 17, New York

MR. HOWARD BROMBERG*
CEIR, Inc.
Benson East, Twp. Line & York Roads
Jenkintown, Pennsylvania

MR. EUGENE H. JACOBS
System Development Corp.
2500 Colorado Avenue
Santa Monica, California

DR. BRUCE GILCHRIST
IBM Corporation
Data Processing Division
112 East Post Road
White Plains, New York

*IEEE Directors*

MR. WALTER L. ANDERSON*
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

DR. R. I. TANAKA
3427 Janice Way
Palo Alto, California

MR. L. C. HOBBS
4701 Surrey Drive
Corona Del Mar, California

MR. T. J. WILLIAMS
Monsanto Chemical Co.
800 N. Lindburgh
St. Louis, Missouri 63166

*Simulation Councils Director*

MR. JOHN E. SHERMAN
Lockheed Missiles & Space Corp.
D-59-15-15, B-102
P. O. Box 504
Sunnyvale, California

*American Documentation Institute Director*

MR. HAROLD BORKO
System Development Corp.
2500 Colorado Avenue
Santa Monica, California

*Association for Machine Translation and Computational Linguistics-Observer*

PROFESSOR WINFRED P. LEHMAN
Department of Germanic Languages
The University of Texas
Box 7939, Austin 12, Texas

*Executive Secretary*

MR. H. G. ASMUS
AFIPS Headquarters
211 East 43rd Street
New York, New York 10017

* Executive Committee

*Standing Committee Chairmen*

*Admissions*

MR. WALTER L. ANDERSON
General Kinetics, Inc.
2611 Shirlington Road
Arlington 6, Virginia

*Award*

MR. SAMUEL LEVINIE
Bunker-Ramo Corporation
445 Fairfield Avenue
Stamford, Connecticut

*Conference*

MR. KEITH W. UNCAPHER
The RAND Corporation
1700 Main Street
Santa Monica, California

*Constitution and By-Laws*

MR. EUGENE H. JACOBS
System Development Corp.
2500 Colorado Avenue
Santa Monica, California

*Finance*

MR. WILLIAM D. ROWE
Sylvania Electronics Systems
189 B. Street
Needham Heights, Massachusetts

*International Relations*

PROFESSOR JOHN R. PASTA
Digital Computer Lab.
University of Illinois
Urbana, Illinois

*Planning*

DR. JACK MOSHMAN
CEIR, Inc.
One Farragut Square, S.
Washington, D. C.

*Publications*

MR. STANLEY ROGERS
P. O. Box 625
Del Mar, California

*Public Relations*

MR. ISAAC SELIGSOHN
IBM Corporation
Old Orchard Road
Armonk, New York

*Social Implications of Information Processing Technology*

MR. PAUL ARMER
The RAND Corporation
1700 Main Street
Santa Monica, California

*Education*

DR. DONALD L. THOMSEN, JR.
IBM Corporation
Old Orchard Road
Armonk, New York

*Technical Program*

MR. BRIAN POLLARD
Burroughs Corporation
41100 Plymouth Road
Plymouth, Michigan

*Harry Goode Memorial Award*

DR. JERRE D. NOE
Engineering Sciences Division
Stanford Research Institute
Menlo Park, California

*General Chairman—66 SJCC*

DR. HARLAN E. ANDERSON
Digital Equipment Corp.
146 Main Street
Maynard, Massachusetts

*General Chairman—66 FJCC*

MR. R. GEORGE GLASER
McKinsey & Co.
100 California Street
San Francisco, California

# 1965 FALL JOINT COMPUTER CONFERENCE

# EXHIBITORS

*as of October 31, 1965*

ADAGE, INC., Boston, Mass.
ADDISON-WESLEY PUBLISHING COMPANY, INC., Reading, Mass.
AMERICAN TELEPHONE & TELEGRAPH COMPANY, New York, N.Y.
AMPEX CORP., Redwood City, Calif.
AMPEX CORPORATION, Redwood City, Calif.
ANELEX CORPORATION, Boston, Mass.
APPLIED DYNAMICS, INC., Ann Arbor, Mich.
BECKMAN INSTRUMENTS, INC., Fullerton, Calif.
BENSON-LEHNER CORPORATION, Van Nuys, Calif.
BRYANT COMPUTER PRODUCTS, A Div. of Ex-Cell-O Corp., Walled Lake, Mich.
CALIFORNIA COMPUTER PRODUCTS, INC., Anaheim, Calif.
CALMA COMPANY, Los Gatos, Calif.
COMCOR/ASTRODATA, INC., Anaheim, Calif.
COMPUTER ACCESSORIES CORPORATION, Santa Barbara, Calif.
COMPUTER CONTROL COMPANY, INC., Framingham, Mass.
COMPUTER DESIGN, West Concord, Mass.
COMPUTER PRODUCTS, INC., Braintree, Mass.
COMPUTERS AND AUTOMATION, Newtonville, Mass.
COMPUTER SCIENCES CORPORATION, El Segundo, Calif.
CONDUCTRON CORPORATION, Ann Arbor, Mich.
CONRAC, A DIV. OF GIANNINI CONTROLS CORPORATION, Glendora, Calif.
CONSOLIDATED ELECTRODYNAMICS CORPORATION, Pasadena, Calif.
CONTROL DATA CORPORATION, Minneapolis, Minn.
CORNING GLASS WORKS, Bradford, Pa.
CYBETRONICS, INC., Waltham, Mass.
DATA DISC, INC., Palo Alto, Calif.
DATA EQUIPMENT COMPANY, A DIV. OF BBN CORPORATION, Santa Ana, Calif.
DATA MACHINES, INC., Newport Beach, Calif.
DATAMATION/F.D. THOMPSON PUBLICATIONS, New York, N.Y.
DATAMEC CORPORATION, Mountain View, Calif.
DATA PROCESSING MAGAZINE/NORTH AMERICAN PUBLISHING CO., Philadelphia, Pa.
DATA PRODUCTS CORPORATION, Culver City, Calif.
DIGITAL ELECTRONIC MACHINES, INC., Kansas City, Mo.
DIGITAL EQUIPMENT CORPORATION, Maynard, Mass.
DIGITRONICS CORPORATION, Albertson, L.I., N.Y.
ELCO CORPORATION, Willow Grove, Pa.
ELECTRONIC ASSOCIATES, INC., West Long Branch, N.J.
ELECTRONIC MEMORIES, INC., Hawthorne, Calif.
FABRI-TEK, INC., Amery, Wis.
FAIRCHILD SEMICONDUCTOR, Mountain View, Calif.
FERROXCUBE CORPORATION, Saugerties, N.Y.
GENERAL COMPUTERS, INC., Los Angeles, Calif.
GENERAL ELECTRIC COMPUTER DEPT., Phoenix, Ariz.
GENERAL PRECISION, INC., Librascope Group, Glendale, Calif.
HEWLETT-PACKARD DYMEC DIVISION, Palo Alto, Calif.
INDIANA GENERAL CORPORATION, Valparaiso, Ind.
INTERNATIONAL BUSINESS MACHINES CORPORATION
DATA PROCESSING DIVISION, White Plains, N.Y.
INDUSTRIAL PRODUCTS DIVISION, White Plains, N.Y.
ITT
INDUSTRIAL PRODUCTS DIVISION, San Fernando, Calif.
ITT DATA SERVICES, Paramus, N.J.
KLEINSCHMIDT DIVISION OF SCM CORPORATION, Deerfield, Ill.
LOCKHEED ELECTRONICS COMPANY—A & IP DIVISION, Los Angeles, Calif.
MAI EQUIPMENT CORPORATION, New York, N.Y.
McGRAW-HILL BOOK COMPANY, New York, N.Y.
MEMOREX CORPORATION, Santa Clara, Calif.
MICRO SWITCH—A DIVISION OF HONEYWELL, INC., Freeport, Ill.
MIDWESTERN INSTRUMENTS, INC., Tulsa, Okla.
MILGO ELECTRONIC CORP., Miami, Fla.
3M COMPANY MAGNETIC PRODUCTS DIVISION, St. Paul, Minn.

MONROE DATA/LOG DIVISION OF LITTON INDUSTRIES, Beverly Hills, Calif.
THE NATIONAL CASH REGISTER COMPANY, Dayton, Ohio
OLIVETTI UNDERWOOD CORPORATION, New York, N.Y.
PHILCO COMMERCIAL ELECTRONICS OPERATION, Willow Grove, Pa.
POTTER INSTRUMENT COMPANY INC., Plainview, L.I., N.Y.
PRENTICE HALL, INC., Englewood Cliffs, N.J.
RADIO CORPORATION OF AMERICA
ELECTRONIC COMPONENTS & DEVICES, Harrison, N.J.
ELECTRONIC DATA PROCESSING, Camden, N.J.
RAYTHEON COMPUTER, Santa Ana, Calif.
RECORDAK CORPORATION, New York, N.Y.
REEVES INSTRUMENT COMPANY, Garden City, N.Y.
REEVES SOUNDCRAFT DIVISION, REEVES INDUSTRIES, INC., Danbury, Conn.
REMEX/RHEEM ELECTRONICS, A DIV. OF EX-CELL-O CORP., Hawthorne, Calif.
ROTRON MANUFACTURING COMPANY, INC., Woodstock, N.Y.
ROYAL TYPEWRITER CO.—A DIV. OF LITTON INDUSTRIES, New York, N.Y.
SANDERS ASSOCIATES, INC., Nashua, N.H.
SCIENTIFIC DATA SYSTEMS, Santa Monica, Calif.
SCM CORPORATION, New York, N.Y.
SOROBAN ENGINEERING, INC., Palm Bay, Fla.
SPARTAN BOOKS, INC., Washington, D.C.
STROMBERG-CARLSON CORPORATION—DATA PRODUCTS DIV., San Diego, Calif.
SYSTRON-DONNER CORPORATION, Concord, Calif.
TALLY CORPORATION, Seattle, Wash.
TECH-MET INC., Sunnyvale, Calif.
TELETYPE CORPORATION, Skokie, Ill.
TEXAS INSTRUMENTS, INC., Dallas, Texas
TRANSISTOR ELECTRONICS CORPORATION, Minneapolis, Minn.
UGC INSTRUMENTS, INC., Houston, Texas
UPTIME CORPORATION, Golden, Colo.
U.S. MAGNETIC TAPE COMPANY, INC., Huntley, Ill.
WEST ELEVEN, INC., Los Angeles, Calif.
JOHN WILEY & SONS, INC., New York, N.Y.
ZELTEX, INC., Concord, Calif.

# REVIEWERS, PANELISTS, AND SESSION CHAIRMEN

*SPECIAL ACKNOWLEDGEMENT. The 1965 Conference Committee is indebted to Russell Bennett, Leonard Cotton, Robert Davies, Ben Ferber, Robert Gray, Robert Kiel, Donal Meier, James Mihalik, John Piontek, Lynn Yarbrough, and David Yetter for applying themselves so diligently to make the Preprint Volume for the Discuss-Only Sessions possible.*

## SESSION CHAIRMEN

D. ACKLEY
L. D. AMDAHL
P. ARMER
D. G. BOBROW
E. E. BOLLES
P. BROCK
J. R. BROWN, JR.
E. BRYAN
J. W. CELLAR
C. W. CLEWLOW
J. S. CRAVER
A. J. CRITCHLOW
C. DAVIDSON
J. B. DENNIS
W. A. FARRAND
M. T. FISCHER
E. GLASER
D. L. GOLDY
M. H. HALSTEAD
D. HARATZ
W. H. HARTWIG
D. G. HAYS
L. C. HOBBS
R. E. KAYLOR
W. B. KEHL
K. KOLENCE
R. L. KOPPEL
W. S. MCCULLOCH
D. MEIER
R. C. MINNICK
G. S. MITCHELL
W. D. ORR
J. B. RHINE
J. A. RICCA
R. RICE
A. ROSENBERG
M. ROSENBERG
W. SANGREN
G. M. SILVERN
I. R. WHITEMAN

## 1965 FJCC PAPER REVIEWERS

C. W. ADAMS
J. C. ALRICH
L. D. AMDAHL
J. P. ANDERSON
P. L. ANDERSSON
R. M. ANNIS
M. ARBAB
A. AVIZIENIS
L. AYRES
R. S. BARTON
D. BEELER
D. W. BERNARD
D. L. BICKEL
L. BISCOMB
R. M. BLOCH
Q. BONNESS
R. E. BRADLEY
P. BROCK
D. R. BROWN
H. D. BROWN, JR.
E. BRYAN
D. J. CHESAREK
J. G. CLARK
D. CLUTTERHAM
M. COHEN
T. A. CONNOLLY
L. W. COTTON
R. DAVIS
O. R. DEACON, JR.
C. F. DEDO
P. DENNING
D. L. DIETMEYER
S. M. DREZNER
T. J. DUDLEY
A. I. DUMEY
F. G. DUNHAM
H. P. EDMUNDSON
R. A. ELLIOTT
H. S. ENGLANDER
F. ESS
B. FERBER
P. J. FOY
R. J. FREEMAN
S. K. FREEMAN
R. H. FULLER
L. GALLENSON
C. L. GERBERICH
J. GOLDBERG
H. GOODMAN
W. F. GOODYEAR
N. GORCHOW
J. W. GRANHOLM
F. S. GREENE
C. H. GUTZLER
E. HABIB
N. HAUSNER
D. HAYS
W. H. HOWE
C. A. IRVINE
B. JACKSON
W. W. JACOBS
R. KAIN
D. KIELMEYER
P. KIVIAT
J. KLEINBARD
K. W. KOLENCE
N. R. KORNFIELD
I. J. KUBASAK
S. M. LAMB
C. J. LAMPE
E. L. LAWLER
J. A. LEE

## 1965 FJCC PAPER REVIEWERS—Continued

S. Y. Levy
D. Liu
P. A. Lunday
F. Magness
R. N. Mathur
R. E. Matteson
J. J. Mihalik
M. Minsky
T. J. Moffett
J. Moxley
G. G. Munay
T. W. Murphy
E. Ostrowsky
J. J. Pariser
D. B. Parker
C. L. Perry
S. N. Porter
R. Richman
V. C. Rideout
B. Roos
A. Rosenberg
M. Rosenberg
J. Saltzer
A. D. Scarbrough
W. J. Schart
P. N. Scheid
I. B. Schneiderman
J. Schwartz
B. Sear
D. Shaefer
N. Shapiro
C. Shelton
T. L. Sides
G. M. Silvern
L. C. Silvern
J. W. Smith
P. B. Smith
F. G. Snyder
L. M. Spandorfer
C. F. Sprague, III
T. Stockham
J. F. Sweeney
D. Tocher
Q. T. Trostund, Jr.
F. O. Underwood
E. Van Horn
I. Warshawsky
C. Weissman
I. R. Whiteman
R. L. Wigington
J. M. Willard

## 1965 FJCC PANELISTS

L. Andrews
G. B. Beitzel
R. W. Beveridge
J. R. Brown, Jr.
A. Burns
R. W. Burt
E. U. Cohler
V. M. Corrado
L. W. Cotton
J. F. Cubbage
C. Dedo
J. J. Digiacomo
J. P. Echert
R. Elfant
D. C. Englebart
D. C. Evans
L. Fein
A. Feller
G. W. Fennimore
E. Forbes
J. W. Forgie
K. Frederiksen
H. Fuller
R. Fuller
W. J. Gallagher
M. W. Goldman
J. W. Gratian
L. C. Hobbs
W. H. Howe
W. S. Humphrey, Jr.
W. J. Karplus
H. R. Kaupp
A. J. Kolk
D. H. Kramer
W. Kuzmin
T. G. Lesher
M. H. Lewin
J. Markus
M. May
D. F. Orr
R. J. Petschauer
J. A. Rajchman
A. P. Sage
F. X. Scafuro
H. Schmid
R. D. Schmidt
P. E. Shafer
R. Shahbender
P. S. Sidhu
L. Slattery
R. W. Smiley
L. M. Spandorfer
K. H. Speierman
T. Steel
L. M. Terman
T. L. Thau
D. Tocher
K. Uncapher
J. A. Ward
W. Wingstedt
R. F. Zeith

Conferences 1 to 19 were sponsored by the National Joint Computer Conference, predecessor of AFIPS. Conferences 20 and up are sponsored by AFIPS. Copies of volumes 1-26, Part II may be purchased from SPARTAN BOOKS, scientific and technical division of Books, Inc., 432 Park Avenue South, New York, N. Y.

| *Volume* | *Part* | *List Price* | *Member Price* |
|---|---|---|---|
| 1–3 | | 11.00 | 11.00 |
| 4–6 | | 9.00 | 9.00 |
| 7–9 | | 9.00 | 9.00 |
| 10,11 | | 7.00 | 7.00 |
| 12,13 | | 7.00 | 7.00 |
| 14,15 | | 8.00 | 8.00 |
| 16,17 | | 6.00 | 6.00 |
| 18 | | 3.00 | 3.00 |
| 19 | | 3.00 | 3.00 |
| 20 | | 12.00 | 12.00 |
| 21 | | 6.00 | 6.00 |
| 22 | | 8.00 | 4.00 |
| 23 | | 10.00 | 5.00 |
| 24 | | 16.50 | 8.25 |
| 25 | | 16.00 | 8.00 |
| 26 | I | 18.75 | 9.50 |
| 26 | II | 4.75 | 2.50 |

Cumulative Index to Vols. 1-26, Part II $3.00

Complete Set: $150.00 Price per set to members: $100.00

27. 1965 Fall Joint Computer Conference, Las Vegas, Nevada, November, 1965.