

# DECUS PROCEEDINGS

# 1963

PAPERS AND PRESENTATIONS

of

The Digital Equipment Computer Users Society

Maynard, Massachusetts

**1963**

**PAPERS AND PRESENTATIONS**

**of**

**The Digital Equipment Computer Users Society**

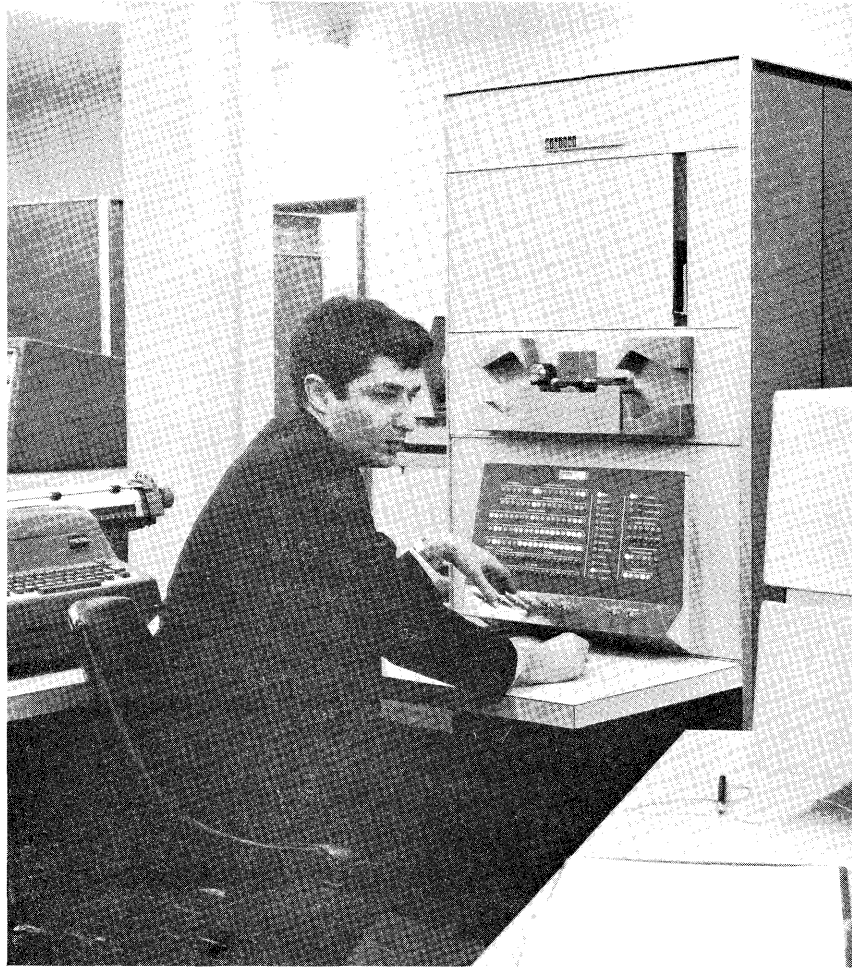
**Maynard, Massachusetts**

Copyright 1964 by Digital Equipment Computer Users Society

#### ACKNOWLEDGEMENT

On behalf of DECUS I gratefully acknowledge the help of the Technical Publications Department, Digital Equipment Corporation in the preparation of these Proceedings.

Elsa Newman, Editor



President Fredkin

## DECUS OFFICERS

### March 1961 - October 1962

Charlton M. Walter, President (AFCRL)  
John Koudela, Jr., Secretary (DEC)

### Committee Chairmen

Edward Fredkin, Programming (then BBN)  
Lawrence Buckland, Meetings (then Itek)  
William Fletcher, Equipment (BBN)  
Elsa Newman (Mrs.), Publications (DEC)

### October 1962 - November 1963

Edward Fredkin, President (I.I.I.)  
Elsa Newman, Secretary (DEC)

### Committee Chairmen

John R. Hayes, Programming (AFCRL)  
Eunice Cronin, Meetings (AFCRL)  
William Fletcher, Equipment (BBN)  
Elsa Newman (Mrs.), Publications (DEC)



## PREFACE

DECUS, Digital Equipment Computer Users Society, founded in March 1961 for the purpose of fostering the interchange of information, ideas, and programming materials of interest to users of Digital's Programmed Data Processor - PDP-1, and its successors, published its first Proceedings of meetings in March 1963. The volume called, "DECUS PROCEEDINGS 1962," comprised papers given at two DECUS meetings which took place that year.

The present volume, "DECUS PROCEEDINGS 1963," concerns papers given at the Spring Symposium and the Annual Meeting held in 1963. The programs of these meetings are given in the Appendix. The twenty-eight papers deal with the PDP-1. Papers on the Programmed Data Processor - PDP-4, given at a special PDP-4 Symposium held on 24 January, 1964, will be reported in a separate publication.

The subject of the 1963 Spring Symposium was TIME SHARING. After papers were presented, the M.I.T. and Bolt Beranek and Newman Time-sharing Systems were demonstrated. The topic, Time Sharing, continued to be of interest at the Annual Meeting where a variety of subjects was covered.

"DECUS PROCEEDINGS 1963" is a testimonial of the continued growth of Digital Equipment Computer Users Society and is a tribute to the authors. DECUS is deeply grateful to all who have contributed to its substance. We give special thanks to the hosts of our meetings and to Dr. J.C.R. Licklider for his inspiring address.

Elsa Newman



# TABLE OF CONTENTS

A USER'S FOREWORD	
C. M. Walter . . . . .	xi
ADDRESS	
Dr. J.C.R. Licklider . . . . .	1
Section I TIME SHARING	
JUST NINE PACKAGES BETWEEN YOU AND TIME SHARING	
S. Boilen and L. Clapp . . . . .	11
STANFORD TIME-SHARING SYSTEM	
J. McCarthy . . . . .	13
REPORT ON A LARGE-SCALE TIME-SHARING SYSTEM	
J. I. Schwartz . . . . .	19
M.I.T.'S PROJECT MAC: CURRENT STATUS	
R. Mills . . . . .	33
Section II UTILITY PROGRAMS AND TECHNIQUES	
RECENT IMPROVEMENTS IN DDT	
D. J. Edwards and M. Minsky . . . . .	41
AN INVISIBLE DEBUGGING PROGRAM FOR A PDP-1 TIME-SHARING SYSTEM	
M. Wolfberg . . . . .	43
MODIFICATION OF A PROGRAM SYMBOLIC AT COMPILE TIME	
J. B. Goodenough . . . . .	51
A VERSATILE PROGRAMMING SYSTEM FOR LARGE PDP INSTALLATIONS	
T. Stollo . . . . .	57
FLINT 36 A3D	
J. Baker and D. Isenberg . . . . .	61



MIDAS ASSEMBLY PROGRAM AND THE PDP-1	
R. Saunders . . . . .	71
Section III PROBLEM ORIENTED TECHNIQUES	
THE PDP-1 AS A VERSATILE RESEARCH TOOL	
W. Fahle and D. Brand . . . . .	75
TIME SHARING IN THE PROCESSING OF NUCLEAR RESEARCH DATA	
A. J. Ferguson, B. Miles and J. Leng . . . . .	83
PDP-1 SCANNING AND MEASURING OF NUCLEAR PARTICLE TRACK PHOTOGRAPHS	
M. Deutsch . . . . .	89
PIP: A PHOTO-INTERPRETIVE PROGRAM FOR THE ANALYSIS OF SPARK-CHAMBER DATA	
H. Rudloe . . . . .	91
THE DIGIGRAPHIC DISPLAY PROGRAM FOR THE DX-1 COMPUTER	
J. T. Gilmore, Jr. . . . .	107
SIGNAL REPRESENTATION AND MEASUREMENT DATA MANIPULATION IN N-SPACE USING AN ON-LINE PDP SYSTEM	
C. M. Walter . . . . .	131
STEPS TOWARD COMPUTER SIMULATION OF SMALL GROUP BEHAVIOR	
T. Roby and R. Nickerson . . . . .	139
A HYBRID PDP-1 FOR SPEECH RESEARCH	
D. L. Hogan and R. J. Scott . . . . .	183
COMPUTER AIDS TO NUMBER THEORY	
M. Pivar . . . . .	193
REQUIREMENTS OF A TIME-SHARED COMPUTER SYSTEM FOR PUBLISHING APPLICATIONS	
L. Buckland . . . . .	201
THE PDP-1 AS A TEACHING AID FOR PROBLEM SOLVING	
W. Feurzeig . . . . .	203

Section IV HARDWARE AND INPUT-OUTPUT TECHNIQUES

HARDWARE PROVISIONS FOR EFFICIENT TIME-SHARING  
OPERATION OF A PDP-1

N. Kerllenevich . . . . . 217

MICROTAPE: ITS FEATURES AND APPLICATIONS

L. Hantman. . . . . 221

ON-LINE INPUT OF GRAPHICAL DATA

W. Fletcher . . . . . 249

THE HYBRID COMPUTATION FACILITY AT UNITED  
AIRCRAFT CORPORATION RESEARCH LABORATORIES

R. Belluardo, R. Gocht and G. Paquette . . . . . 261

USES FOR THE PDP-1 AT LIVERMORE

N. Hardy. . . . . 271

THE PDP-1 AS A DISPLAY MAINTAINING CONSOLE

A. Kotok . . . . . 273

Section V APPENDIX

SPRING MEETING PROGRAM . . . . .	A-1
ANNUAL MEETING PROGRAM . . . . .	A-3
ATTENDANCE - SPRING MEETING . . . . .	A-5
ATTENDANCE - ANNUAL MEETING . . . . .	A-7
AUTHOR INDEX . . . . .	A-9



## A USER'S FOREWORD

Seldom in the rapidly evolving field of computer technology has there been so succinct an expression of key concepts - likely to dominate the shape of things to come - as that outlined by J.C.R. Licklider in the DECUS Annual Meeting keynote address: COMMAND OF PROCEDURES.

The concept of coherent programming thrusts to the heart of the principal difficulty which thwarts our advance into the third decade of the first era of electronic information processing systems. This difficulty is exemplified by the proliferation of programs which are conceived in too narrow a context to mesh in any coherent manner with other programs. It is unfortunate that this course of action is often forced upon the user (who must get results in a hurry) by those systems programmers who are out to write the ultimate utility system, and who show an enormous disdain for all that has gone before. Only modest and tentative agreement can be expected on a "general" meta language. Even here, there is much basic disagreement among the theoreticians. Under such circumstances, enlightened pragmatism, and a willingness to achieve some reasonable degree of coherence, appears the best one can hope for.

The explicit identification by Dr. Licklider of "memory sharing," rather than "time sharing", as the key to any fundamental advance in information processing systems places the emphasis where it rightly belongs. The phantom delusion of timesharing a single central processor - at its fastest, orders of magnitude too slow to simulate the most elementary filter structures - like many a tired TV commercial, has been pushed just a little too far. The utility of central-processor time sharing, for many routine jobs of an elementary nature, is of undisputed value. Some of the papers in this volume, and those discussed at the May Symposium, show the value of this kind of time sharing. The extension of this notion to the "on-line" solution of all problems, with everyone promised a highest priority channel is, however, somewhat questionable. The concept of numerous, quasi-autonomous processors, sharing a hierarchy of both private and communal memories, certainly offers the user an important measure of control over his own destiny, and valuable insulation from the dictatorship of a system which purports to be all things to all users.

C. M. Walter

## ADDRESS

### COMMAND OF PROCEDURES

Dr. J. C. R. Licklider  
Advanced Research Projects Agency  
Washington, D. C.

The context to be assumed in this discussion is that of a large-scale coherently-programmed, memory-shared computer system used cooperatively by several or many people. This context is appropriate to a DECUS meeting because the PDP-6 seems to be well designed to be the heart of, and PDP-1, PDP-4, and PDP-5 to fulfill peripheral or utility functions in such a system and because some of the most important problems that must be solved in the process of developing such a system are problems of cooperation among users in the preparation and application of software.

Within the stated context, I should like us to examine an idea that we may call "command of procedures." If we think of computer programming as being the "preparation of procedures," then "command of procedures" has to do with the organization and application of procedures that have already been prepared. The term has connotations of on-line operation and intimate interaction between the user ("commander") and the running program. Command of procedures depends heavily upon memory sharing and coherent programming. We should therefore review those two concepts as a preliminary.

The central idea of memory sharing, of course, is to make available to many users and many programs 1) essentially continuous read-only access to memory containing a common pool of procedures and data and 2) intermittent read-write access to memory containing individual programs and working data -- particularly to a larger amount of high-echelon memory than the users could otherwise afford. The central idea of coherent programming is to make it easy for users and the high level parts of their programs to take advantage of the common pool of prepared procedures and the common data base.

#### Memory Sharing

The reason for speaking of "memory sharing" instead of "time sharing" is that, as many of us have recognized, it is the sharing of memory, more than the sharing of processor or of time itself, that is fundamental. Time sharing is presently, indeed, the most practicable technique with which to achieve memory sharing, and it may achieve some

additional advantages through the sharing of other computer subsystems as well, but it is nevertheless merely a technique. Memory sharing, on the other hand, is a fundamental advance toward coordination and mutually facilitory interaction among the efforts of many people, working together either on particular tasks or in the general intellectual endeavor of mankind.

### Coherent Programming

The other term, "coherent programming," is intended to cover a general approach to the software problem, almost a philosophy. It is to be contrasted with the approach in which each user or programmer prepares his programs without regard for the possibility that they might be useful later, either to himself or to others. Coherent programming emphasizes the generation and/or use of procedure libraries, library indexes, adaptable routines, general routines, procedure documentation, communication pools, procedure-calling conventions, etc. It is not opposed to diversity of languages, not strongly opposed even to diversity within such subclasses of language as procedure-oriented language, but it is opposed to the prevailing inter-linguistic incoherence which makes routines written in one language usually very difficult to link with routines written in another. The concept of coherent programming includes arrangements for the linking of subroutines at the time of compilation, or at the time of loading into the information base, or at the time of transfer from one memory echelon or location to another, or at the time of execution. The latter two are envisioned as being the most useful within the context we have assumed.

As a final note of introduction, let me try to relate the concept I am here calling "coherent programming" to the concept recently dubbed "implicit programming" by Fred Frick and his associates in a technological forecasting group. "Implicit programming" is aimed at the problem of making computers useful to people who need information-processing help on substantive tasks but who are not skilled in the arts ordinarily subsumed under computer programming, and it is characterized by its rather definite, integrated approach. An implicit programmed computer is envisioned as having within it a large executive program--in fact, as being delivered with a large executive program--that includes: the language facilities, such as translators, compilers, and interpreters; the display facilities, such as display-formatting and display-generating routines; the information-base management facilities, such as file-handling and list-processing routines; and an extensive set of algorithms covering the field of application of the system. "Coherent programming" is intended, also, to solve the substantive-user problem, and it makes use of all the techniques just mentioned, but it does not insist upon - nor,

for that matter, does it wholly reject - having everything incorporated into a monolithic executive program. Coherent programming simply requires coherence among the component languages, routines, displays, techniques, conventions, and formats of a large software system no matter whether the system is fully developed before delivery to the users by a programming organization or whether, after being launched as a fairly simple "boot-strapping" program, it evolves into mature operational form through coordinated efforts of substantive users who program or substantive users who do not program but have programmers closely associated with them.

In the first decade of electronic digital computing, the development of software was mainly a matter of analyzing substantive problems and preparing integral programs (what old aviation enthusiasts might call "monocoque" programs ) to solve them. In the second decade, basic algorithms corresponding to the common functions of algebra and analysis were available as ready-made subroutines, and assembler and compiler languages were devised to facilitate incorporation of the basic algorithms into ad hoc programs. Typical programs of the second decade were hierarchal, two or three echelons deep, and about half library subroutine, half ad hoc programming.

As we look back from the beginning of the third decade, we may well wonder whether all the conceivable elementary routines have not by now been written in five or ten different forms. Unfortunately, if they have, most of the routines are of no use to anyone, for it would be easier to rewrite them than to retrieve them and even if they were in hand they could not be compiled on an available compiler or run on an available machine, and even if they could, they probably would not link with other routines gathered from other sources. This widespread - not universal but certainly widespread - failure to organize software development in such a way as to build up a coherent system of useful procedures, subroutines, macros, and the like, puts software in the same class as the social sciences in that discomfiting observation, "The natural scientists of each new generation stand on the shoulders of their predecessors, but the social scientists of each new generation step in their predecessors' faces."

Whatever the past, it is feasible now to develop for any specialized problem area a coherent system of routines that will handle the great majority of substantive operations, and to create a problem-oriented language that will put the substantive operations obediently under the command of users not skilled in machine or procedure-oriented programming. That not only can be done, but of course is being done in several fields.

However, most of the problem-oriented systems now being developed are being developed in an uncoordinated way by diverse organizations and they are not likely to mesh together as interlocking subsystems to form a coherent over-all system. Against the criteria of coherent programming, that unlikelihood is a serious and expensive deficiency.

For the purposes of this part of the discussion, let us nevertheless assume that we have, in one large computer, a large, coherent system of compatible, linkable routines. It would be easiest for some of us to visualize such a system, I think, if it were strictly hierarchical, consisting of zero-order routines that do elementary chores all by themselves, calling no subroutines, and first order routines that call only routines of zero order, and second-order routines that call only routines of first order, etc. However, as long as an  $n$ th order routine calls at least one  $(n-1)$ st order routine, as it must by definition, it should be allowed to call routines of any order lower than  $n$ . Moreover, it would be wrong to bar recursive calling just to keep the picture simple. The calling pattern in our hypothetical system of routines, therefore, is perhaps somewhat complex, but it is not excessively so.

The routines are stored, let us assume, in the procedure part of the computer's information base. I use the term "information base" to denote an extension of the concept, data base. A data base contains data. An information base contains any or all kinds of information, particularly including procedure as well as data. Associated with the procedure part of the information base (in which the routines are stored), there may be lists and tables that define the structures of some of the routines. However, the parameters, flags, etc., that specialize a routine for a particular purpose are stored outside the information base in a sector of memory associated with the individual user.

By "command of procedure," in the context we have been developing, I wish, of course, to suggest a rough analogy with a military commander's command of forces. Military command is based in part upon plans (corresponding to prepared routines) carefully thought-out, checked, and filed for use when situations arise to which they are appropriate. At the same time, military command involves strong elements of ad hoc improvisation and real-time problem solving. These elements usually manifest themselves in selection, modification, and rearrangement of existing plans, for the time scale of battle tends to be shorter than the time scale of detailed planning.

Until recently, there was not much opportunity for "command of procedures" in digital



computing. The time scale of computer "battle" was so short, relative to the time scale of selecting or modifying or rearranging programs, that the only feasible mode of operation was to plan everything in advance to the last detail and then to execute the entire plan precisely as written. With the coming of on-line operation of computers, and with the prospect that the gross time scale of processing can be matched economically to the time scale of human thinking, however, the notion of commanding divisions and battalions of computer instructions becomes significant. We have seen the beginning in such work as Culler and Freed's, Weldon Clark's, and Ivan Sutherland's and in military database management or "query" languages. It seems likely that command of procedures will become increasingly important, and possible that, as more and more basic procedures are programmed and made available in procedure-oriented languages, command of prepared procedures will displace preparation of new procedures as the focus of software effort.

Command of procedures is analyzable into several interacting parts. In a direct extension of the schema of conventional programming, one would work out a flow chart that will fulfill the over-all requirement, identify the various parts of the flow chart with procedures that are available in the information base, connect the procedures together in an arrangement corresponding to the flow chart, and then execute the entire arrangement. To command would thus be essentially to prepare the highest echelons of a system of routines and to count on an executive or monitor to retrieve the lower-echelon routines from the information base and to effect the linkages. As suggested, that is a simple schema, but is almost its only virtue. It would be wanton not to take advantage of the fact that being "on line" lets one neglect possibilities that do not arise. And for many problem areas and many people there may be languages that are more natural or more convenient or (even) more powerful than the language of flow charts. It seems desirable, therefore, to explore more complex schemata in the hope that they will prove more effective and appear simpler to the user.

A schema that is attractive to me is based on the use, in the formulation of the over-all procedure, of some workable compromise between natural (human) language and readily machine-interpretable language. The commander would call for "tentative processing," in which mode the initial state and certain intermediate states would be recorded for possible use in display and as fall-back positions, and then he would formulate what he wanted done as far ahead as he could see with reasonable clarity. A "differential com-

piler," partly syntax-directed and partly responsive to semantic aspects of the language, would then try to interpret the sequence of commands. Its objective would be to find an interpretation that was (1) consistent with the input statements, (2) interpretable as a pattern of simple operations (such as branching) and subroutine calls, and (3) capable of providing appropriate arguments for every called subroutine. If the computer found such an interpretation, it would (a) set in motion the machinery for retrieving the subroutines, (b) produce the necessary machine code to implement the highest echelons of the program, (c) link the retrieved subroutine to form the lower echelons, (d) run the program, (e) display intermediate and terminal results, and (f) await further command to revise or continue. If the compiler failed to find a satisfactory interpretation, it would try to indicate the nature of the difficulty: syntactic trouble, undefined term, no subroutine available to implement specified operation, subroutine argument out of bounds, etc. - but with greater diagnostic precision than those terms suggest. If the compiler found a marginally satisfactory interpretation, or more than one satisfactory interpretation, it would work backwards from the interpretation(s) and say, "Do you mean . . . . ?"

By "differential compiler," in the foregoing, I mean a compiler that keeps track of the interdependencies among program statements and permits (1) the addition and testing of increments without recompilation of the base program and (2) the alteration and testing of statements at any point in the program without recompiling statements not affected by the alterations. Perlis has developed such a compiler. It makes use of his "threaded lists" in keeping track of interdependencies. Note that the difference between "compiler" and "interpreter" fades when the compiler operates upon short segments of program under the user's on-line direction.

Retrieval and linking of subroutines are not, of course, the whole problem. The language must permit the user to introduce data (numbers, facts, declaratives) into the data base and to query the data base. The language must be capable of selecting displays and governing their formats and contents. The language must be capable of "modulating" the subroutines, of specializing them for particular questions. (This is mainly a matter of supplying parameter arguments to the subroutines, which we have already mentioned, but it should be handled subtly. For example, adverbs in the user's statements might determine the values of arguments that affect the operation of the subroutine but do not change its basic structure.) Most importantly, perhaps, as the foregoing

discussion of the use of the compiler and mention of communication with the data base suggest, the language must be capable of directing the operation of programs upon routines, and even upon entire programs. Thus some of the contents of the information base would be routines for use as components in constructing programs, some would be programs for managing the data base and displays, some would be programs for operating upon programs, and some would be the basic parts of the language mechanism itself.

In the immediately foregoing discussion, I have, for the sake of simplicity, used "language" in the singular. It seems likely that there will be requirements for dialects within a language and for a multiplicity of languages. If these are to be several languages, then there probably should be a fundamental meta-language or system-control language. To facilitate their learning and retention, the several languages should be as compatible, one with another--as coherent in the sense of fitting together in an over-all design--as is consistent with the requirements placed upon them individually by their applications.



**Section I**

**TIME SHARING**



## JUST NINE PACKAGES BETWEEN YOU AND TIME SHARING?\*

S. Boilen and L. C. Clapp  
Bolt Beranek and Newman, Inc.  
Cambridge, Massachusetts

### ABSTRACT

This paper will describe the design and implementation of a time-sharing system for the PDP-1 computer. The system was designed especially for simultaneous debugging of programs by five independent users. The system has been in daily operation for the past several months. A second generation system with many new features is under development and will be operational shortly. These improvements will be discussed in detail.

\*This paper was not submitted in time for publication.





## STANFORD TIMESHARING SYSTEM

John McCarthy  
Stanford University  
Stanford, California

### ABSTRACT

The Stanford Computation Center is developing A Time-Sharing System involving a PDP-1, a twelve station keyboard and CRT Display System remotely controlled laboratory apparatus, and a direct connection to our IBM 7090. The system will be used for an experimental teaching machine laboratory, for debugging programs on the PDP-1 and 7090 and for other man-machine interaction work. Plans for the system will be described.

### INTRODUCTION

I shall describe the time-sharing system that we are building at Stanford University. The System is like the BBN time-sharing system, but will be considerably extended in both hardware and capability.

The special hardware will be delivered on March 1, 1964, and the remainder in May, so we should have the System in some form of operation by June. Services will be available to the users through a number of input-output devices and, of course, as is typical of a time-sharing system, the users will be able to operate independently of each other.

### THE SYSTEM

The facilities of the system will include twelve keyboard - scope -- light pen consoles, which I will describe below. These consoles will be located in the Computation Center itself, and in addition there will be probably at least five model 33 teletypes which will be located remotely, in different places on the Stanford Campus. Remote experimental apparatus will be connected also. The present two clients for this arrangement are the High Energy Physics Laboratory and the Genetics Department in the Medical School.

The High Energy Physics Laboratory desires to be able to ask the time-sharing system, at any time,

- 1) to read the multi-channel pulse-height analyzer,
- 2) to do some simple computations,
- 3) to plot some graphs which will tell how an experiment is going.

The Laboratory personnel would like to be able to do these things in real time. For these a Calcomp plotter will have to be added to the teletype and the connection to their apparatus.

The Stanford Computation Center would like eventually to make computation for laboratory apparatus a sort of a public service. There would be four wires coming out of the wall in the laboratory, two of which connected to a teletype or similar device. To the other two wires one could connect any apparatus. To accomplish this the time-sharing system will be arranged so that no matter how the apparatus interacts with these wires, it cannot stop the time-sharing system or interfere with other users (even if the user puts 110 volts on it, all he will do is blow a fuse). This is our goal, but so far, the picture of what kind of interaction a general purpose facility ought to provide is not very clear. Therefore, we are providing services for the immediate users who have made their needs known, and later we will try to evolve a general purpose system.

Our present system is shown in Figure 1.

## THE DISPLAY SYSTEM

The display system is of special interest for it can display approximately 200,000 characters per second. It writes on one scope at a time, and the displays are maintained by switching, in sequence, among the twelve scopes. With a total of 200,000 characters per second, each scope has a capacity of 1/12 of that, or 17,000 characters per second. If one displayed forty times a second, 425 characters could be maintained flicker-free on each scope. Our experience with the PDP-1 character generator shows that for programming use flicker-free displays are not needed and therefore, one could legibly display a considerably large number of characters on each scope.

The display system receives 18-bit words from the 16K memory - one word every fifteen microseconds. The system can be in one of three modes:-

1. typewriter mode, in which the 18-bit word is interpreted as three 6-bit characters. The total number of displayable characters is 114. (The system simulates a typewriter closely by correctly interpreting such characters as case shift carriage return and, of course, an escape character which escapes into control mode).
2. vector mode, in which the 18-bit words is interpreted as two 7 vectors, each with sign, using 16 bits. There is a register in the display unit which represents a position on the scope and this vector is added to that position, so that

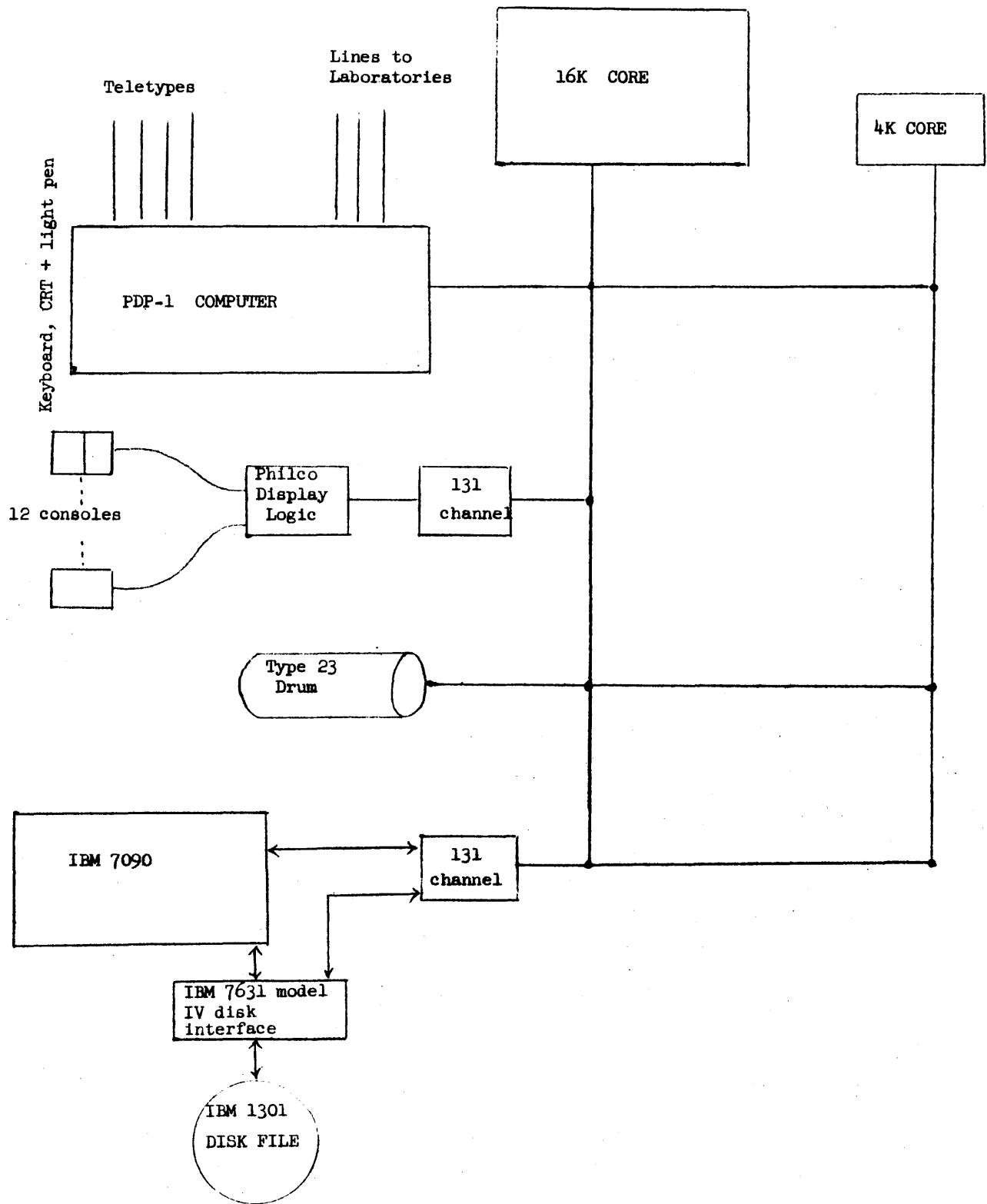


FIGURE 1

vector gives a new position. The remaining two bits are used in the following way: one combination says that the vector is to be displayed. Another combination says that only the endpoint is to be displayed, that is, for making dot pictures. A third combination says that the vector is not to be displayed at all. Thus we are merely moving the position point on the register. Then the fourth combination says that it is not actually a vector, it is positioning. In which case, the leading 6-bits are positioned.

3. control mode. The system can choose to enter one of the other modes; it can change the brightness of the characters; it can change the size of the characters; it can add or stop displaying on the scopes or it can add a scope to those which are displayed.

The PDP-1 uses this system giving a transmission instruction to the 131 display channel. At the end of the transmission instruction, an interrupt occurs and the program gives the next transmission instruction for the next display and so forth.

## OTHER CONNECTIONS

The next item on the system diagram is a connection to the other 131 channel which in turn contains two two-way connections. One of these two-way connections is the Stanford IBM 7090 computer. The other two-way connection is to our IBM 1301 disk file.

The fourth cross point is the 32 field swapping drum.

During time-sharing, the user program can be at most 4K, and inhabits the 4K memory. The 16K memory is used for system programs and for storing the characters which are to be displayed on the various scopes. As in the BBN system, each user gets a quantum of time after which his core image goes onto the drum and the next user comes in. Since this description is not intended to be complete, I shall list only the applications for which the system is currently intended.

## APPLICATIONS

1. The first application is a heuristic programming laboratory which will use the 7090 for studies in artificial intelligence, especially programming in LISP.
2. Similarly, there will be a mathematical laboratory whose object will be to make the commonly used operations on formulas, differentiation, integration, algebraic simplification, etc., available to users directly.

3. The third application will be as a teaching machine laboratory. This is one of the major sources of support for this project and six of these twelve consoles belong to this teaching project. They will also have a number of other gadgets which will be controlled by the PDP-1, such as an audio system, and a computer controlled slide projector.
4. The next application is teaching programming.
5. A final application is picture processing, i.e. the interpretation of spark chamber pictures which we hope will serve as a background activity sopping up what computer time is not used by the time sharing.

The question, whether the PDP-1 can keep all this going at once - can keep all these balls in the air - is somewhat an open question. It is a question the BBN time-sharing system might answer, but because of the fact that the BBN system was never really in production use, we must still guess.

## EDITING TEXT AND FILE HANDLING

### The General Input Routine

A final subject I think I would like to discuss is the question of editing text and file handling. We can look forward to fairly complicated file facilities because we will have files in the drum belonging to currently active users, and files on the 1301 disk file which may be either programs for the PDP-1 or programs for the 7090 which will be edited using the display system.

We are thinking right now about a general purpose editing and input system, and I should like to mention the general input routine. The general input routine is designed to give input to the computer over a complete range of applications. The applications vary in how promptly your own program must pay attention to characters which are typed. As one limiting case assume that you are preparing a file. Here you would like characters as you type them to be simply added to a buffer in core and when this buffer gets full, to be transmitted to a secondary storage. Even then some control characters are needed such as an equivalent of a back space for erasing an immediately detected error, and an escape to a control mode in which transmission operations can be dictated. We would like to be able to enter a cliché, that is, a text that has been predefined.

The other extreme is the use of the keyboard as a control device. Thus, every character hit on the keyboard goes directly to the program which interprets it and takes some action based on it. For example, the keyboard control device may change a display.

There are intermediate situations where in typing some language action may be desired to be taken only at break character times. For example, in DDT, if symbols are used, the system need not take action at every character, but only when a break character is typed.

The Stanford Computation Center hopes to provide a general routine which will give all of the facilities described. There will be a table of all the characters that can be typed on the keyboard. Each character will have a status. One character status is that it should just be added to the buffer when typed. A second is that this character is a delimiter and causes all the characters in the buffer to be transmitted to the users program or to the utility program which is currently working for the user. A third is that this character has an immediate control significance relative to the input itself; for example, the backspace action facility.

# A REPORT ON A LARGE-SCALE TIME-SHARING SYSTEM

Jules I. Schwartz

System Development Corporation  
Santa Monica, California

**ABSTRACT:** The System Development Corporation, under ARPA sponsorship, has developed a time-sharing system on the Q-32 computer. Time-sharing, in this case, implies simultaneous access to the computer by a large number of independent users. The goal of the system is to provide essentially immediate response to queries from all users. Users have at their disposal keyboards (primarily teletypes), displays, and other computers. These devices can be operated from local (within SDC Santa Monica) or remote stations. The system has been operational since June, 1963. It permits program production and debugging, experimentation with human subjects, rapid on-line programming and computation, and other functions which can benefit from computer-human interaction. This paper discusses the system as it appears to the user, the general design of the system, and relates some of the experience had in using the system.

## INTRODUCTION

Since June 1963, a Time-Sharing System has been operational at the System Development Corporation in Santa Monica. This system was produced under the sponsorship of ARPA and has utilized ideas developed at both Massachusetts Institute of Technology and Bolt Beranek and Newman as well as some original techniques. Time-sharing, in this case, means the simultaneous access to a computer by a large number of independent (and/or related) users and programs. The system is also "general purpose," since there is essentially no restriction on the kind of program run under the system. The system has been used for compiling and debugging programs, conducting research, performing calculations, conducting games, and executing on-line programming using both algebraic and list-processing languages.

### Equipment Configuration

The major computer used by the system executive is the AN/FSQ-32 (manufactured by IBM). Also used by the system is the PDP-1 (manufactured by Digital Equipment Corp.),\* which is the major input/output vehicle for the various remote devices.

The remote input/output devices available to users include teletypes, displays, and other computers.\*\* These devices can be run from within SDC, and from the outside, with the exception of displays, which can be operated only a short distance from the computer.

---

\*PDP-1 not operating at this writing, but planned for operation in October.

\*\*The line for other computers is to be available in November.

Computers which are to be used at remote stations include the CDC 160A, the DEC PDP-1, and the IBM 1410.

Figure 1 is a description of the system's remote equipment configuration as it will look in November 1963.

(At present there are 8 teletypes, 6 displays, and no computers in the network.)

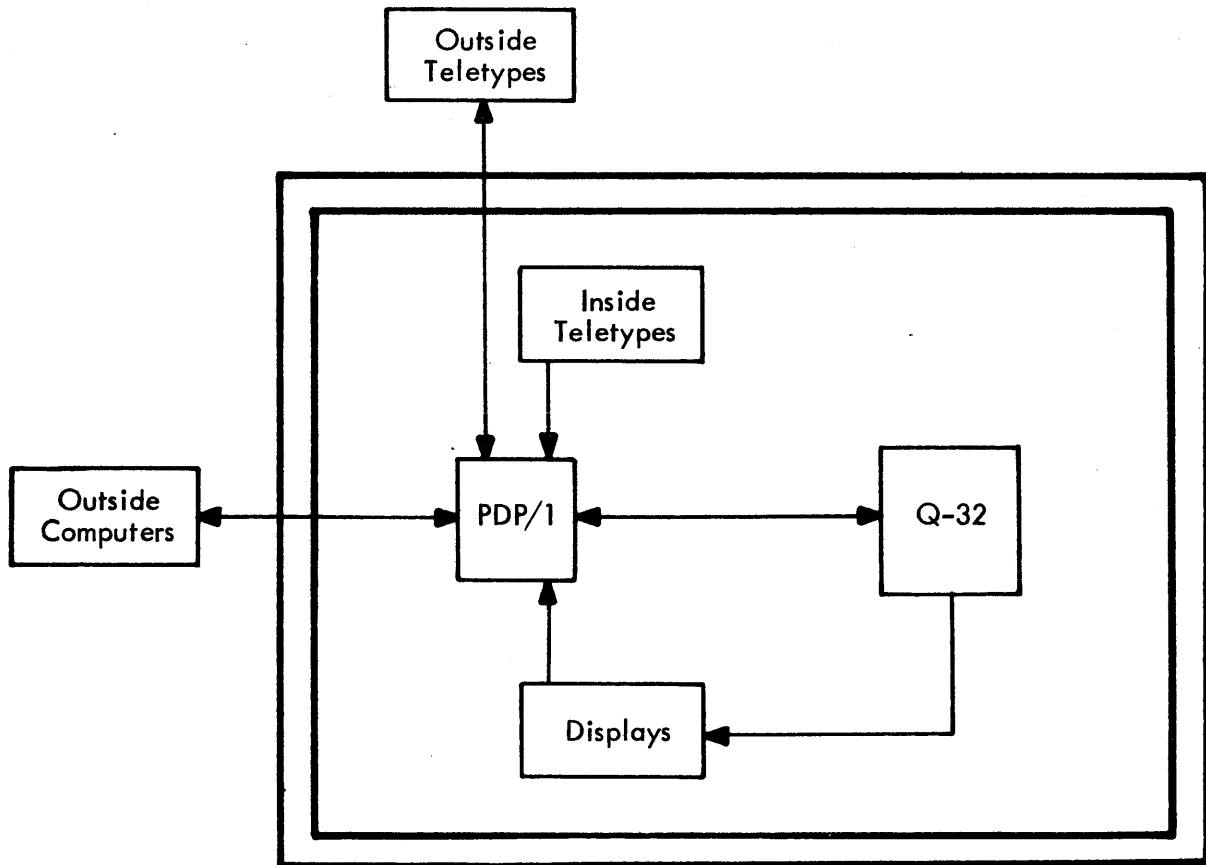


Figure 1

The AN/FSQ-32 computer is a 1's-complement, 48-bit-word computer, with (at present) 65,536 words of high-speed (2.5 microseconds cycle time minus overlap) memory available for programs, and an additional 16,384 words of high-speed memory available for data



and input/output buffering, the latter memory called input memory. The PDP-1 also has access to this input memory. Thus this memory serves as the interface between the two computers. In addition, the Q-32 has an extremely powerful instruction repertoire, including access to parts of words for loading, storing, and arithmetic, and also an extensive interrupt system.

Figure 2 presents a description of the computer and its immediate input/output devices.

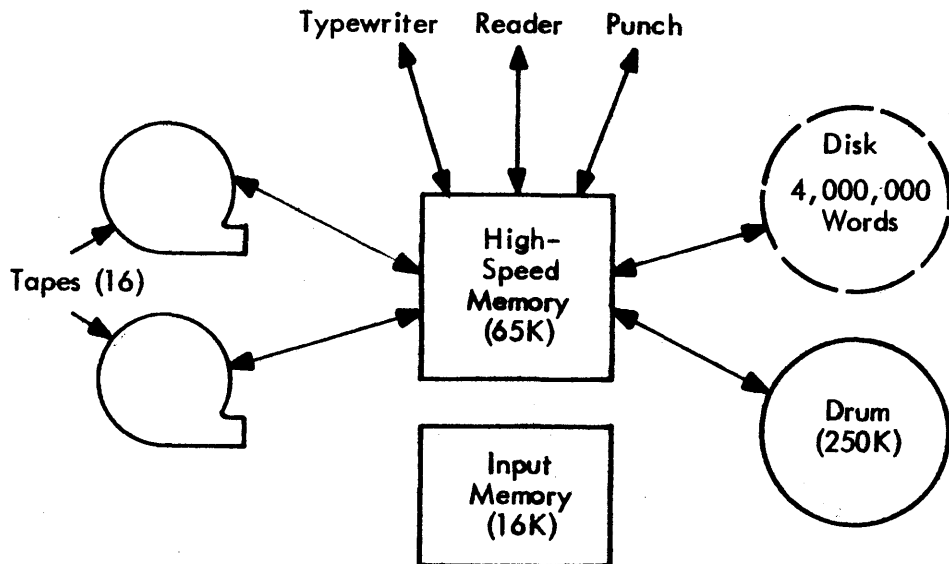


Figure 2

### The Time-Sharing System as it Looks to the User

The time-sharing user today communicates with the Time-Sharing System primarily by means of teletype. He has at his disposal six basic commands to the system:

\*Disk is to be installed in the spring of 1964.

LOGIN

LOAD

GO

STOP

QUIT

DIAL

Briefly these commands mean:

- LOGIN: The user is beginning a run. With this command he gives his identification and a "job-number."
- LOAD: The user requests a program to be loaded (from tape today, eventually, from disk). Once this command is executed, the program is an "object program" in the system.
- GO: The user wishes to start the operation of an object program or restart the operation of an object program that has been stopped. Once the user gives this command, he can send teletype messages to either his object program or the time-sharing system.
- STOP: The user wishes to stop the operation of an object program.
- QUIT: The user has finished a particular job. Upon receipt of the QUIT, the Time-Sharing System punches a card with certain accounting information on it and removes the object program from the system.
- DIAL: The user may communicate with other users or the computer operators with this command.

In addition to these basic commands, the user has available to him a variety of program debugging, or checkout, commands. These commands and the basic symbols used for each one include:

OPEN	s/
MODIFY OPEN REGISTER ADDRESS	±/n
INSERT	V*
MASK	n <sub>1</sub> n <sub>2</sub>
MODE	F or O or I or H
BREAK POINT	s;
DUMP	s n/

s = symbol or address

n<sub>i</sub> = integer

V = floating, octal, integer, or alphanumeric value.

Briefly, the functions of these commands are:

- OPEN: Displays the contents of the given memory or machine register and uses this as a base address for other commands.
- MODIFY OPEN REGISTER ADDRESS: Changes the address of the opened register by the given increment or decrement.
- INSERT: Inserts the given value into the opened register.
- MASK: Inserts values by the given mask.
- MODE: Displays values according to specified mode (floating, decimal, octal, Hollerith).
- BREAK POINT: When this point in the program is reached, notifies the user, and (on options), displays registers, and stops or continues the program. As many as five break points are allowed simultaneously.
- DUMP: Dumps a given set of registers, either on teletype or tape.

## Object Program Input-Output

As stated before, there is no restriction on the type of object program that can run in the system. Therefore, as much input/output equipment as possible is made available to object programs. Thus, today object programs may use tapes, displays, and teletypes for input and output. In November, other computers will also be treated as input/output devices; further, disk storage, when installed, will be made available to object programs. Since, in a system like this, it is impractical to have specific teletypes or tapes referred to by object programs, input/output is done in a general fashion, with all input/output devices given arbitrary names by the object programs and declared to be files used by the object program during its run. Thus only the time-sharing system knows what physical tape drives, teletypes, or areas of drums are being used.

## Additional Tools Available to System Users

The commands and devices mentioned so far have been facilities available to users or users' object programs directly through the Time-Sharing System's Executive. With these tools one could run and debug programs that exist in a binary form. To make the system more useful, however, a number of additional devices (called service routines) are available to users. These are themselves run as object programs, so it is clear that there is no limit to the number of service routines that can eventually be made available. Some of the more interesting routines that are currently operational are:

- FILE - file and update on tape
- JTS - compile from tape
- TINT - interpret on-line JOVIAL programs
- IPL-TS - IPL-V interpreter
- FRDN - sophisticated desk calculator

The functions of these routines are:

- **FILE:** This program enables one to file away and update large volumes of symbolic data onto tape either from card reader or teletype .
- **JTS:** JTS (JOVIAL for Time-Sharing) is a one-pass JOVIAL-and-machine language compiler that accepts input from tape and produces binary programs and listings on tape .
- **TINT:** This program (Time-Sharing Interpreter) permits on-line (teletype) programming in a form of JOVIAL. It executes interpretively the coded program either in parts as it is coded or as a whole . It also permits dynamic input and output of data via teletype .
- **IPL-TS:** An interesting IPL-V interpreter exists that accepts inputs from either tape or teletype . It is strongly oriented to on-line processing and contains a number of unique features for on-line checkout of IPL programs .
- **FRDN:** FRDN is a fancy desk calculator, which permits, in addition to the standard arithmetic operations, exponential and trigonometric functions .
- Other service routines, including a sophisticated text editor, a mathematician's assistant, and a LISP processor are under development at SDC and elsewhere .

### System Operation

The discussion so far has been primarily on the operation of the system from the user's point of view. The following is an attempt to give an over-all description of the system and describes briefly how it operates .

Basically, the system operates as follows: All object programs are stored on drum, put there as a result of the LOAD commands. When a program's time to operate arrives, or preferably ahead of this time, it is brought into high-speed memory. If bringing in a program

to its area in memory causes a storage conflict with another program, the latter must be re-stored to its place on drums. A program's turn will end when it initiates an input or output request, a machine or program error is detected, or its time is up, the time allotted being determined prior to its particular turn. At the completion of its turn, its machine environment (e.g., accumulator, index-registers, etc.) is saved, and it either resides in memory until its next turn or is written on drums. This mechanism is controlled by the time-sharing Executive.

The time-sharing Executive in the Q-32 consists of 8 major components:

INTERRUPT CONTROL

TELETYPE INTERPRETER

DISPATCHER

SCHEDULER

BASIC SEQUENCE

INPUT/OUTPUT

START-UP, CLEAN-UP

DEBUGGER

The major functions of these components are:

- **INTERRUPT CONTROL:** Operates as a result of a computer interrupt and, based on the nature of the interrupt, fires off the next or another Executive module to be operated.
- **TELETYPE INTERPRETER:** Determines if input messages are for the Executive system or for object programs. If the message is for the object program, it simply sets an indication that there is a message waiting for the particular program. If the message is for the Executive system, it either sets an indicator telling what is to be

- done, or, if it is a "one-shot" action such as required by a LOGIN or QUIT command, performs the task itself.
- **DISPATCHER:** (This is the biggest part of the system.) The major functions of the Dispatcher include the allocation of storage, both internal and peripheral, the transfer of programs to and from high-speed storage and the sequencing of all input/output commands.
  - **SCHEDULER:** The Scheduler is a program whose task is simply to determine which object program is to operate. At the completion of every object program, the Scheduler is entered. At its exit it gives three parameters: (1) the program to be run at this time; (2) the amount of time the program is to be allotted at this time; and (3) the program to be run next, this parameter to permit reading-in the next program while the current program is operating. The Scheduler employs several criteria in making these decisions. Basically, it schedules to give all "active" programs (e.g., those not waiting for input or output or not stopped) a maximum response time--currently, this parameter is 2 seconds. In practice, however, most programs receive much better response time than the maximum, primarily because most programs don't use all their allotted time, and the majority of programs are not "active."
  - **BASIC SEQUENCE:** The main function of the Basic Sequence is to fire off object programs, and in so doing, to give them any teletype, light pen (display), or computer inputs that have been made available to them since their last operation.
  - **INPUT/OUTPUT:** This is a set of routines that actually perform input/output for the various devices.
  - **START-UP, CLEAN-UP:** These functions begin and end the system.

- **DEBUGGER:** This is the program that performs the requested on-line debugging commands.

The Time-Sharing System's Q-32 Executive occupies 16,384 words of memory, leaving the remainder of memory for object programs.

The Executive which exists in the PDP-1 is primarily concerned with maintaining the flow of information to and from the remote devices. It does relatively little decision-making. However, it does determine the kind of input/output device concerned, the type of conversion necessary (if any), and the particular channel of the device with which it is communicating.

### Experience with the System

There are some obvious advantages to this kind of system that have been borne out in practice. There is a large class of problems whose compute time is extremely small in relation to the total time the problem is on the computer. This is because a large percentage of time is taken up by human thought and computer input/output. In fact, the use of a computer for this kind of application in a non-time-sharing mode is so inefficient that frequently it would not be worthwhile to run. There are many examples of this kind of problem. The one that most programmers are familiar with is console debugging. This means the checkout of programs with the programmer at the computer--anathema to most computer managers, but desired by a large number of programmers. These kinds of applications have been run with a high degree of success in the SDC Time-Sharing System, with each person involved actually feeling he has the whole computer to himself.

At the other end of the spectrum are those programs that compute for essentially 100% of the time they are on the computer. If these programs compute for long periods of time, say a matter of minutes, they will completely usurp their allotted time and thus tend to make



the on-line user wait for the maximum response period possible. Time-Sharing does not benefit this kind of user, except that this kind of program can be run "in the background" while other on-line interaction programs are idle. In the SDC installation, the percentage of these long-period computer programs has been small, so that no serious system response time delays have been noticed from them.

Questions frequently asked are, "Do people like the system?" "Does it produce better results than other, more standard techniques?" Both the questions are difficult to answer in an absolute sense. However, some reasonable observations can be made that apply to this system and probably to others of this kind.

- (1) Those on-line interaction programs that used to run in a non-time-sharing mode but were converted to time-sharing produce results that are as valid as before but with greater efficiency in computer operation, since a number of different ones are run simultaneously.
- (2) The on-line debugging capability has proved very valuable. This system of debugging gives a feeling of closeness to the computer and control over the program, so that debugging time is reduced considerably while the efficiency of computer utilization stays high. Also, although the tools available so far have been relatively few and unsophisticated, one can see, however, the advantages to be gained by giving everyone immediate access and response from a computer. "Directed" computer runs are the mode of operation. Every step taken is taken only as a result or verification of the previous step. If things do not go as planned, alternative paths can be followed immediately. Without time-sharing, the choices one has had were the "submitting" of a run, followed by an anxious waiting period which frequently is climaxed by a sigh (or worse) and a resubmitting of the same

run, or one man's on-line interaction with the computer, which helps him but causes consternation on the part of others who are waiting for computer runs.

Such a system must be made fool-proof. Because of the nature of the system, one must have a reasonably long time of uninterrupted operation to get satisfactory results. This implies several things:

- (1) The system Executive must be reliable and able to account for any condition that may arise, including object program and machine errors.
- (2) The machine must be reliable. Although the system must provide the ability to analyze each computer error and isolate and stop only the particular object program or programs affected, frequent or solid computer errors can cause the entire system and all object programs to terminate.
- (3) Certain hardware features are essential. These include: Memory protection--the ability to prevent object programs from destroying each other or the Executive system. High-speed large-storage random-access devices--the major bottleneck in a system of this kind is the slow rate at which object programs can be moved in and out of memory. Also, the use of magnetic tapes for such functions as the permanent storage of programs and data files creates operational and timing problems that can be overcome with the use of large drums or disks. Clock interrupt capability--the system requires that no single program run for an excessively long time. Therefore a clock that can be set to interrupt operation at various intervals is necessary for complete control and the assurance of adequate response time.

### Conclusions

When the SDC Time-Sharing System first became operational, it had no memory protection, its Executive was unreliable, and its computer was beset by a much heavier load

than it was used to and reacted accordingly. With these obstacles, the early users were subject to frustrations unlike many found in the twentieth century. The system's life expectancy was no more than ten minutes. The only remarkable thing about the early months was that anything useful was accomplished. Interestingly enough, however, some work was accomplished, primarily through patience on the part of the users. With the passage of time, many of the problems have been alleviated through both equipment improvements and programming improvements, and although no exact figures are available, there is strong feeling that the presence of the time-sharing system has increased the computer's value considerably.

It is interesting to watch a group of people using a computer simultaneously but solving different problems using different tools. At the computer console itself, one can usually see all the available tape drives busy, typewriters busy, drum indicators indicating the drums are busy, the punch punching on occasion, and, the card-reader going at anywhere from quarter to full speed. If one may judge the worth of a computer by the amount of equipment used per second, time-sharing is well worth its investment.

So far, the system at SDC has been developed and used mainly to make computer use more efficient. Since the system has been under development for a relatively short time (it was begun in January 1963), the number of existing service routines is still small. However, one can envision the development of an increasing number of on-line programming aids and techniques of utilizing keyboards, displays, and groups of computers to make a time-sharing network a truly powerful device.

It is certainly conceivable that in the not too distant future, many people will have at their fingertips a device which, at a reasonable cost, enables them to enter an operating network such as this one. While in this network, they will have access to routines, techniques, and computing power unavailable to them by other means. The computing power will include not only the Executive computer but the other computers that are in the network as well. Thus, the possibility of large-scale time-sharing networks seems to be one of the more promising developments in computer technology today.



## M.I.T.'S PROJECT MAC: CURRENT STATUS

Richard G. Mills

Massachusetts Institute of Technology  
Cambridge, Massachusetts

### INTRODUCTION

The Massachusetts Institute of Technology has been participating, since Spring 1963, in a major national program of research on advanced computer systems and their exploitation sponsored by the Department of Defense. The contract of \$2,220,000 to initiate the effort was awarded by the Office of Naval Research on behalf of the advanced Research Projects Agency of the Department of Defense.

The research is being carried out under the project name "MAC," an acronym derived from two titles: machine-aided cognition, expressing the broad project objective, and multiple access computers, describing its major tool. The project director is Dr. Robert M. Fano, Ford Professor of Engineering and Professor of Electrical Communications. The Project MAC program is being carried out adjacent to the M.I.T. campus in the Technology Square office complex. It occupies the eight and ninth floors of the Beta Building, 545 Technology Square, Cambridge, Massachusetts.

### OBJECTIVES

The broad goal of Project MAC is the experimental investigation of new ways in which on-line use of computers can aid people in their creative work, whether research, engineering, design, management, or education. Thus, an essential part of the project is the evolutionary development of a large, time-shared computer system that is easily and independently accessible to a large number of people, and truly responsive to their individual needs. The keynote is ease of access, both physical and intellectual, which must apply to the information stored in the computer system as well as to the computer's information processing capability. The goal is an intimate collaboration between the human user and the computer in a real-time dialogue on the solution of a problem, in which the two parties contribute their best capabilities--for the man, imagination, insight, inspiration and judgment; for the computer, enormous computing power, high-speed data retrieval from a vast store, and the ability to handle the details of very complex logical processes.

Project MAC is capitalizing on a long history of pioneering work on computers and information processing at M.I.T. and M.I.T.'s Lincoln Laboratory, which includes such milestones as the analog computer of Dr. Vannevar Bush prior to World War II, Whirlwind I, the SAGE System, and the TX-2 computer. More recently, the research of the Computation Center on time-sharing systems, and the work of the Computer-Aided Design Project, jointly conducted by the Electronic Systems Laboratory of the Electrical Engineering Department and the Design Section of the Mechanical Engineering Department, have provided the foundation on which the present MAC system is built.

## FACILITIES

The present MAC computer installation includes a specially modified IBM 7094 computer and a Digital Equipment Corporation PDP-1 computer (see Fig. 1). The IBM 7094 computer is the central part of the time-sharing system. The primary terminals of the system are, at present, 40 Model 35 Teletypes and 16 IBM 1050 Selectric typewriter stations. Two of the terminals are located at Lincoln Laboratory in Lexington, the rest on the M.I.T. campus. Each terminal can dial, through the M.I.T. switching central, either the MAC computer installation or the similar installation in the M.I.T. Computation Center. The supervisory program of the two computer installations may, independently, accept or reject the call.

MAC's large-scale computer is an augmented IBM 7094. It has been modified to operate with two banks of 32K core memory, and it has six data channels, as illustrated in Figure 1. Modifications in addition to the two-bank core memory include hardware facilities for relocation and memory protection. These features, together with an interrupt clock and a special operating mode in which input-output operations and certain other instructions result in traps, were necessary to assure successful operation of independent programs coexisting in core.

Two basic motivations for adding the second core bank, which is reserved for the supervisor, are 1) to avoid imposing severe memory restrictions on users because of the large supervisor, and 2) to permit use of existing programs (e.g., FAP) which require all or most of core.

The Programmed Transmission Control (7750) is a stored-program computer which serves as the interface between a 7094 data channel and up to 112 telegraph-rate (100 or so

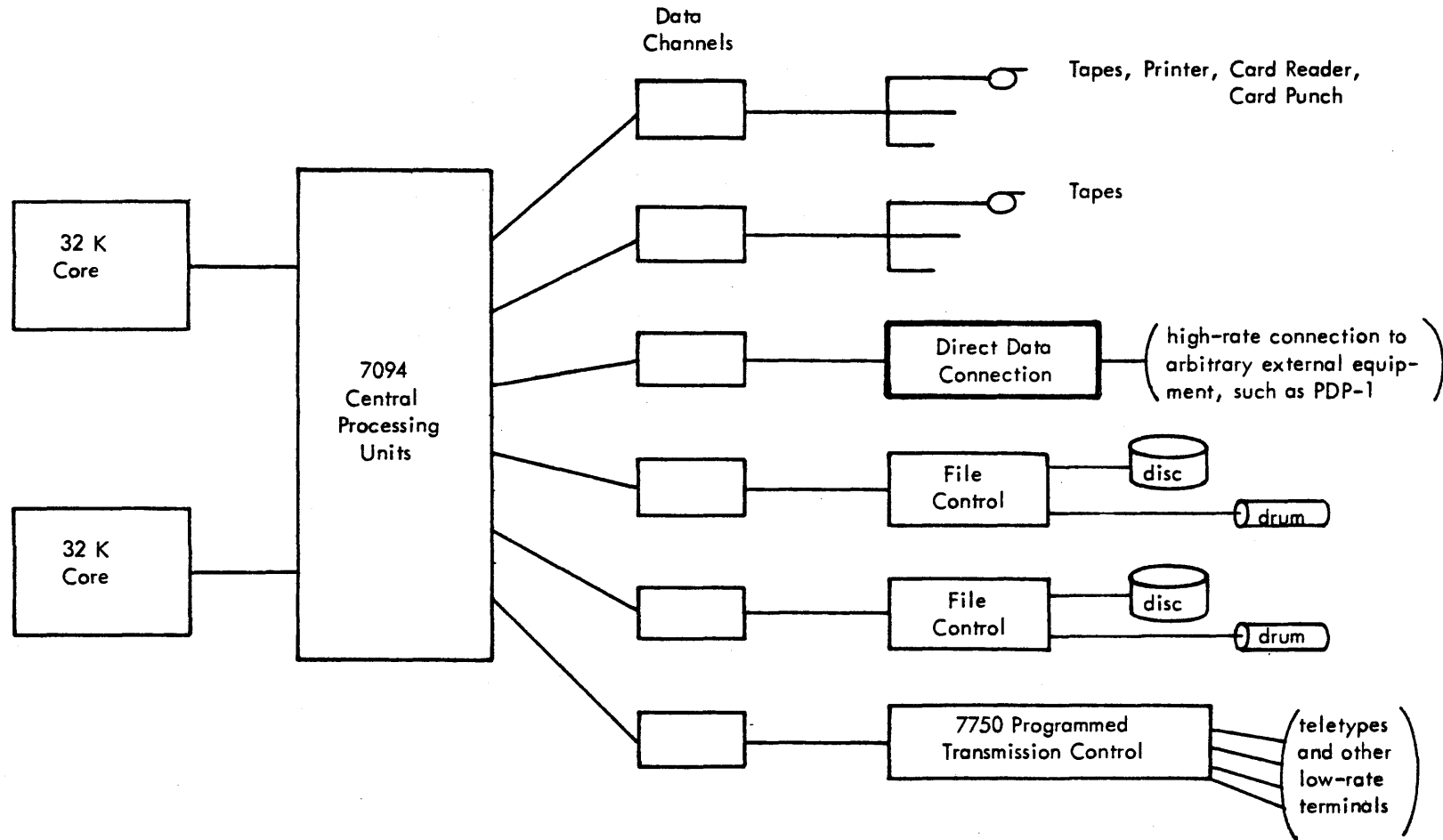


Figure 1 MAC Time-Sharing Computer Configuration

bits/sec) terminal devices. Alternatively, higher-rate terminals (e.g., 1200 bits/sec) may be traded for groups of low-rate lines. The 7750 is compatible with Bell System data sets.

The initial 7750 configuration at Project MAC provides for three 1200-bit terminals, 24 terminals for Model 35 Teletypes, and 28 terminals for IBM 1050 Selectric typewriter stations, all interconnected through a dial network. (The Computation Center's 7750 is identical, except the number of Teletype terminals is 16). An arrangement to allow the computer to initiate calls will be an early addition.

Present plans call for the 1200-bit lines to be used, through data sets, for intercommunication between the MAC and Computation Center 7094's, and also as one means of connecting the 7094 to some of the other computers at M.I.T. Two PDP-1's and a 1620 will be fitted for this connection. As the experimental program develops, other uses will doubtless arise.

MAC now has installed a 16K PDP-1 with high-speed channel and scope display with character generator and light pen. Micro tapes will be added shortly. This machine is one of those mentioned above which will be adapted for 1200-bit-per-second connection to the 7750. For another class of experiments the same machine will be connected at a much higher rate through the PDP-1 high-speed channel to the 7094 direct-data connection. The basic role of the MAC PDP-1 is that of an extremely flexible, high quality (i.e., high data-rate) terminal for man-machine interaction.

The M.I.T. Electrical Engineering Department's PDP-1, which is itself time shared, can also participate, possibly by maintaining several display and typewriter terminals.

In order to provide access from outside M.I.T., the MAC system will be connected in the near future to the TWX and Telex networks operated respectively by the American Telephone and Telegraph Corporation and the Western Union Company. Experiments are being planned in collaboration with a number of universities participating in the national program to provide experience with long-distance operation of time-sharing systems.

MAC's initial supervisory operating system is the M.I.T. Compatible Time-Sharing System (CTSS). An evolving system of programs developed by the M.I.T. Computation Center, CTSS was first publicly demonstrated in 1961. It includes supervisory, scheduling,



debugging, assembler/compiler and input/output facilities. The programming languages now or soon to be available to the MAC users on CTSS are FORTRAN, FAP, MAD, COMIT, LISP, COGO, STRESS, ALGOL, SLIP and SNOBOL. Other languages, and particularly other common utility programs, are planned for future inclusion.

A typical programming session at a time-sharing terminal from the user's point of view might go as follows:

The user first must log in to the system, giving his identification. When he is accepted by the system he may then type in a subroutine, perhaps using the MAD language. He could then call for a printout of his input, edit it to correct errors, and then call for a compilation of this part of his program. The resulting binary program, together with other previously compiled programs, could then be loaded and run and the result or post-mortem data obtained. If necessary, the user may examine the contents of registers in memory, make corrections to the source program, recompile and so on, repeating the process as often as necessary. When the user chooses to terminate his work, he may save the present state of all of his programs with the assurance that they will be ready for him to pick up the threads at his next session at a terminal, perhaps hours or weeks in the future. When he logs out of the system, the user receives from the supervisory program accounting data indicating how much actual computer time he has used.

CTSS allows a conventional batch-processing load to be operated as "background." Any computer capacity which is not demanded by users at remote terminals is absorbed by the background load of the system.

While teletype or other typewriter-like terminals are adequate for many purposes, some applications demand a much more flexible form of graphical communication. An excellent example of graphical communication arose two years ago on the Lincoln Laboratory TX-2 computer in connection with the doctoral thesis of Ivan Sutherland. This work, in the general area of computer-aided design, is directed toward "tightening the loop" in the design process through use of a computer with appropriate terminal equipment. Using such a terminal, and aided by the computer, the designer might sketch in a drawing of a mechanical part using a "light pen" or other tracking device. Working with his visual display the designer could then modify his drawings as he worked and then perhaps subject the partially designed part to simulated testing by indicating at his display the application of loads and having the computer present the reaction of the part such as deformation, failure, etc. When the designer is satisfied, he might then

press a button which causes the computer to produce a tape to control a machine tool to actually produce the part.

The M.I.T. Electronic Systems Laboratory recently completed a visual-display terminal with flexible input-output facilities for use in such applications as those described above. The console which ESL has produced includes an oscilloscope display with character generator and light pen together with some local logical capability to simplify the task of the computer in maintaining the display. For communication from the console to the computer the terminal includes a variety of devices such as knobs, push-buttons, toggle switches and in the near future a typewriter. The meaning of a signal from one of these input devices is entirely determined by the program in the computer. There is no "wired in" local significance. Thus, the terminal is an extremely flexible device which can be used in many fields of application.

While the time-shared computer system is the most visible part of the project at this time, the research activities which use it constitute a major part of the MAC effort and are also well underway. The details of the work are difficult to report at the present stage, but an indication of the level of activity is given by the heavy demands made by the research groups on the MAC computer. To satisfy the project's needs, the computer is operated on a five-day per week around-the-clock basis and will go to a full seven-day week later this month. At present, when the compatible time-sharing system program (CTSS) is on--which is about 80 per cent of the total MAC computer time--as many as 24 persons can simultaneously use the machine from various remote terminals. This number may go as high as 75 when planned improvements to the 7094 and CTSS are completed.

**Section II**

**UTILITY PROGRAMS AND TECHNIQUES**



## RECENT IMPROVEMENTS IN DDT

D. J. Edwards and M. L. Minsky  
Massachusetts Institute of Technology  
Cambridge, Massachusetts

### ABSTRACT

This paper will report new developments and recent improvements to DDT.

"Window DDT" now will remember undefined symbols and define them on a later command. Using sequence breaks, it can change the contents of memory while a program is running, and the contents of memory can be displayed in symbolic form on the scope.

### INTRODUCTION

The distinction between a debugging system and a programming system is not very clear, especially when the two systems work within the same language. In debugging small MACRO programs using DDT, one often adds substantially to the volume of the original program. At least volumetrically, then, he must be programming.

For very small programs it is clearly more efficient, provided access to the computer is freely available, to write the whole program at the keyboard using DDT, because this eliminates a number of preparation phases each of which has a fixed overhead time. With larger programs, on-line programming with DDT becomes more difficult, because there is more to keep track of in what is already written, and more planning to do.

We have added a few features to DDT that make more convenient the construction and debugging of larger programs. Below we describe these new features, and then discuss what it might take to make a really smooth on-line programming system.

#### Use of Undefined Symbols

We have added a second symbol table to DDT. This table lists references to symbols used but not yet defined. When a symbol is defined, the references to it in this table are filled-in and moved over to the regular symbol table. This means that one can refer forward to data or program elements not yet provided for, thus reducing the need for advance planning of storage layouts in full detail. The handling of undefined symbols is completely automatic and does not require any new action on the part of the user. When an undefined symbol is typed-in, the machine acknowledges it by typing an over-strike bar. A special control signal makes the system list the currently undefined symbols; another makes the system define all currently undefined symbols in a consecutive block beginning at the currently open location.

#### The WINDOW Feature

The control signal xxxx causes a listing of the program starting at xxxx to appear on the

scope. This listing has three columns: location, symbolic contents, octal contents. A few special switches on our console then allow one to move this "window" up and down through the program, or "jump" the window to the effective address of an indicated instruction, thus following transfer paths. This feature serves many of the functions of a complete listing and is quite useful in helping to reduce the amount of mental bookkeeping involved in on-line program-construction. It is also very useful in debugging. The window display runs concurrently with DDT, using sequence-break. One operates DDT normally while the window is up, and it indicates changes as they are made in the program.

#### Sequence Break DDT Feature

Using single-channel sequence-break, DDT remains available while the object program is being run, so that parameters can be changed or patches inserted without stopping the program. If the program contains a scope display output, this means that one can quickly adjust parameters and the like without writing special provisions for this into the object program itself. Again, no new actions are required on the operator's part.

The present version does not allow for object program typewriter input-output.

#### Discussion

The three features discussed above seem very useful. They do not make DDT into an ideal machine-language assembly program. We plan to improve the system in a number of respects; the result will be a more-or-less complete on-line macro-assembly system.

a) The programs resulting from DDT are not easily relocatable; this makes it hard to build large programs from separately-written small ones. This is particularly regrettable since almost all the information required for relocation is available to the system at input time. If this relocation information is preserved, it can be used not only to make a relocatable program but also to make the system able to provide good symbolic listings, by suppressing meaningless symbolic interpretations of data quantities.

b) The system is weak because it does not have macros.

We plan to add non-recursive macro definitions to the system, and have it produce a relocatable punch-out. Relocation of program blocks during debugging will be permitted.

The version of DDT with improvements (1), (2) and (3) above will be distributed soon through DECUS. No date is set on the relocatable-macro DDT.

# AN INVISIBLE DEBUGGING PROGRAM FOR A PDP-1 TIME-SHARING SYSTEM

Michael Wolfberg

Massachusetts Institute of Technology  
Cambridge, Massachusetts

## ABSTRACT

An invisible debugging program, fashioned after DDT, is an essential feature of the M.I.T. PDP-1 Time-Sharing System. This paper describes convenient operation for the user and the close connection between the debugging program and the Time-Sharing System Executive Routine.

## INTRODUCTION

The first operating system of the M.I.T. PDP-1 Time-Sharing System incorporates the MACRO assembler, EXPENSIVE TYPEWRITER ( for editing source language tapes), and a debugging routine based on DDT. By Time-Sharing Console and typewriter control, each user can call for any of these utility programs. The on-line debugging program aids in quickly detecting program errors, easily correcting them on-line, and then punching a working object program. Under control of the debugger, the user may save a protected version of his program and then run his entire program or sections of it for debugging purposes.

## TIME SHARING

The Time-Sharing System's design depends on extra hardware appended to the standard PDP-1 (4096-word core memory). Hardware provides a basis for operation of the Time-Sharing Executive Routine, which remains in upper memory in order to control sequencing of active programs and to buffer most in/out data transfers. A magnetic drum is used for temporary storage of up to twenty-two 4096-word fields where utility and users' programs remain except when running in core memory.

A time-sharing interrupt channel causes traps to the Executive Routine whenever a running program executes an in/out instruction, a halt, illegal instruction, or one of a set of special time-sharing instructions. In order to insure users' programs not stopping the system, there is automatic memory protection for references to registers above a memory bound, which would otherwise interfere with the Executive Routine;

such references trap as illegal instructions.

A more detailed description of the structure of the system has been described by N. Kerllenevich.<sup>(2)</sup>

## DDT

DDT (Dec Debugging Tape) is a symbolic debugging program used under normal operation of the PDP-1, which occupies the last 2000<sub>8</sub> words of upper memory. Since DDT coexists in memory with a user's program, it can only be used to debug a program which is not so long that it interferes with DDT. In addition to the 2000<sub>8</sub> registers, DDT builds its symbol table down towards lower memory. Its initial table contains all the initial symbols of the MACRO assembler, i.e., the PDP-1 mnemonic instruction codes. DDT is able to merge a MACRO symbol punch with its table so that the symbols used in the source language of a user's program can be used to make references in the process of debugging. The on-line typewriter serves as a two-way communication channel between the user and DDT. Lower case characters are either symbolic or numeric constituents; typed-in upper case characters serve as action or control characters and cause DDT to perform various operations.

The most commonly used operation with DDT is the examination of a memory register. Typing a symbolic or numeric address followed by a slash ("/") causes DDT to type a tab (tabulator stop) and "open" the register by typing out its symbolic contents followed by another tab. The user can then modify the contents of that register by immediately typing the new contents followed by a carriage return, which "closes" the register.

There are four control characters in DDT which have the same effect as a carriage return in that they "close" a register, but, in addition, they cause another register to be "opened." For example, hitting the "back-space" key on the typewriter causes the currently "opened" register to be "closed" and "opens" the next sequential register.

Another class of operations which DDT performs is the examination of a block of registers. A search may be made for all words equal to an expression typed before the control character "W," and all occurrences of the expression are typed out with their locations. Typing "E" after an expression causes DDT to search memory for



all words having an effective address equal to the expression. This search follows all indirect addressing chains to a maximum depth of  $100_8$ .

Another block operation DDT performs is the zeroing of registers. It is common practice to zero all of memory not occupied by DDT before reading in a program to be debugged.

DDT performs three operations which are associated with reading of paper tape: reading a binary tape (program) into memory, merging a MACRO symbol table into DDT's table, and verifying or comparing the contents of a binary tape with the contents of memory.

There is the provision to punch out binary tapes with DDT in either of two binary tape formats. Rather than running through another MACRO assembly, it is often more efficient to modify an incorrectly assembled program in memory by DDT and then punch out a corrected program on binary tape.

In order to run the user's program, typing an address followed by "G" causes DDT to transfer control to that register. The user's program is then in control of the computer. If the user wishes to return control to DDT, he can use the switches on the console to "STOP" the computer and then "START" it running again at the entry point of DDT.

One of DDT's most useful features is the implementation of a breakpoint. By specifying an address followed by "B," DDT is conditioned to insert a breakpoint at that location when DDT passes control to the user's program, as is done by the "G" operation and two others which will be mentioned below. The breakpoint is an instruction (viz., jda) which transfers control back into DDT, whereupon DDT preserves all necessary registers and indicators, so that upon future return to the user's program, they can be restored. When a break occurs, DDT types out the location where the break occurred and the contents of the accumulator at the time of the break. The preserved contents of the accumulator and in-out register are in "A" and "I," consecutive registers internal to DDT. These registers are equivalent to program registers for purposes of examination and modification.

Before inserting the special breakpoint instruction in the user's program, DDT saves

the instruction at the break location and replaces it whenever control returns to DDT, as if there had been no breakpoint assigned there.

After a break occurs and the user desires the sequencing of instruction in his program to continue as if there had not been a break, he can "proceed" by typing a "P." After necessary registers and indicators have been restored, this operation causes DDT to execute the instruction which was at the break location and then transfer control back into the user's program at the appropriate location. Before proceeding, the user may remove the breakpoint assignment by typing "B" alone, or its location may be altered by typing a new address followed by "B." In any case, the proceed operation returns to the instruction which caused the last break. When an expression is typed preceding the "P," the breakpoint will not cause a trap for that number of times. This operation is termed "multiple proceeds."

The third action which may cause control to pass to the user's program is the "execute" command. When an expression followed by an "X" is typed, DDT restores necessary registers and indicators and sets up the breakpoint, and then the expression is executed as an instruction. If the executed instruction is not some type of jump operation, then DDT is re-entered, and the necessary registers and indicators are again saved. (1)

## A DEBUGGER FOR TIME SHARING

Under the time-sharing system, regular DDT would not operate because the executive routine occupies upper memory, however, a version occupying registers 4000<sub>g</sub> through 6000<sub>g</sub> could run as a user's program. Such a DDT, with symbol table, would reduce available memory for the user's program to approximately 3400<sub>g</sub> registers. This is one strong motivation for the incorporation of a debugger which would occupy essentially no space in the user's allotted memory more than the few registers necessary in the Executive Routine area, and still have the full capabilities of the DDT debugger.

Such an invisible debugger, called "ID," has been written in conjunction with the Executive Routine, and it has the advantage over DDT in that it cannot be harmed by a user's program running wild. All features of DDT have been incorporated into ID; however, ID also adds three major new features to the debugging process: "save" and "unsave" operations, "type-in" mode, and multiple breakpoints.

The "save" operation allows a user to copy an image of his program from the drum field on which it is stored by the system onto another field, so that it may be "un-saved" or restored at a later time. Utilizing these operations can guarantee a user's not having to read in a program more than once.

"Type-in" mode allows a user to suppress all timeouts in a series of register modifications until he types a carriage return. This feature is helpful when using the debugger to insert a small program into memory (actually the user's drum field).

Breakpoints will be discussed below in greater detail.

Each user of the system is assigned his own ID, which occupies its own drum field when not in core. The user's program and debugger are never concurrently active. When ID is active and operating, it makes references to the user's program on the drum field where that program is kept by the Executive Routine. When the user commands ID to run his program (by "G," "P," or "X"), ID is made inactive, and the user's program becomes active. When the user's program executes an illegal instruction or a breakpoint instruction, the program is dismissed and made inactive, while ID is assigned to be active.

## BREAKPOINTS

In the Time-Sharing System, a special instruction, bpt, has been included, which is used as the breakpoint instruction (corresponding to the special jda of DDT) since it causes a special trap to the executive routine when encountered. With the absence of some of the indicator lights of the main console under time-shared operation, extensive use of breakpoints becomes one of the most efficient ways to debug a program. With this in mind, ID has been written to allow for four breakpoints, and can easily be extended to allow for any number of them.

The format used in ID defines "B" as an internal register which contains the first breakpoint assignment. Subsequent internal registers through "B+3" allow the user to specify a total of four breakpoints. A redundant assignment causes ID to remove the original assignment, so the user may not have two breakpoint assignments for the same location. In an examination of one of the registers "B" through "B+3" which does not contain a breakpoint assignment, ID types out an overbar. The user inserts an overbar to remove a single breakpoint assignment. Also, the special

combination of "B" followed by an overbar removes all breakpoint assignments. To allow for rapid stepping of breakpoints through a program and to retain compatible format with DDT conventions, typing an address followed by "B" causes that address to be inserted into register "B."

The multiple proceeds counter of ID is associated only with the breakpoint assigned in register "B."

In examining the problems of implementation of multiple breakpoints, a further restriction to breakpoint placement in DDT was found to be necessary. To avoid errors, breakpoints cannot be placed at jsp or jda instructions. The following example demonstrates such an error:

1. A breakpoint is assigned to a jsp instruction which calls a subroutine.
2. The program is run, and the breakpoint is encountered.
3. Before proceeding back to the program, the assignment is changed to somewhere in the subroutine called by the jsp command.
4. After proceeding, the breakpoint is encountered, but now the exit of the subroutine contains the breakpoint-return address somewhere in DDT. Since another break occurred in the meantime at a location other than the jsp instruction, incorrect operation results.

The above error cannot occur in ID since the method of breakpoint service has been totally changed. There are no longer any necessary restrictions to breakpoint placement at jsp or jda instructions. A more thorough discussion of breakpoints in ID can be found in the author's B.S. Thesis. <sup>(3)</sup>

#### RETURNING CONTROL TO ID

In the initial system, the CALL Button on the user's Time-Sharing Console always returns control to ID. If the button push interrupts an operation of ID, then that operation ceases and a carriage return is typed; whereas, if the user's program is running, a typeout identical to the breakpoint format is performed. The user may then type a "P" to resume running the program as if it had not been interrupted.

Whenever a user's program causes an illegal instruction trap, ID is brought back into control, indicating the location of the instruction. Therefore, if the user remembers the instruction replaced, any number of illegal instructions can be used as breakpoints.

## REFERENCES

- (1) "DDT," Memorandum PDP-4-1 and Supplement, PDP-1 Computer, M.I.T.
- (2) Kerllenevich, Natalio, "Hardware Provisions for Efficient Time-Sharing Operation of a PDP-1," Paper presented at DECUS May Symposium, M.I.T. Published in DECUS PROCEEDINGS 1963.
- (3) Wolfberg, Michael, "An Invisible Debugger for a PDP-1 Time-Sharing System," B.S. Thesis, Department of Electrical Engineering, M.I.T., June 1963.



## MODIFICATION OF A PROGRAM SYMBOLIC AT COMPILE TIME

John B. Goodenough  
Air Force Systems Command  
Bedford, Massachusetts

### ABSTRACT

This paper defines two new DECAL compiler operations and shows how they may be used to provide options for altering a program's symbolic at compile time. These alterations may be planned so that DECAL, in effect, compiles only an optimum program for a particular usage or equipment configuration.

The new operations (called action operators in DECAL language) are "omit" and "dd?". "omit", when followed by an octal number N or a symbol whose value is an octal number N, causes the N lines following the action operator to be omitted. "dd?" tests whether the symbol preceding "dd?" has been previously defined or not. If so, the rest of the line is omitted. Using these two action operators and other DECAL features, it is possible to compile certain lines of a symbolic only when a particular option has been chosen.

Usage of the action operators will be illustrated for a display package and the DEC Input-Output Subroutine Package modified to include the following options: 1) 16 channel, standard, or no sequence break system; 2) automatic or standard multiply; and 3) multicore or single core use.

A method is described here which will facilitate the sharing of programs among programmers whose applications and even whose machines differ. While the method is aimed primarily at making general purpose programs even more general, it will work with any program in the program library.

Basically the method involves the use of two new action operators in DECAL. Action operators are small programs executed at compile time whenever the action operator name is seen. For example, one action operator is "stp" which causes DECAL to halt. Another is "dec" which converts the decimal number following "dec" to octal. The action operators defined to allow editing at compile time have been called "omit" and "typeget." These action operators, which are used in writing the symbolic of a program, allow the user at compile time to tell the properties of his machine or application, e.g., whether it has 16 channel sequence break or not, and if so, what channels are connected to the various in-out devices, or, in the case of in-out buffers, what size buffer is optimum for the desired use of the program. DECAL then compiles the program so that it will be compatible with his machine or use.

This is accomplished as follows: the omit action operator is followed by an octal number, *n*, or a symbol whose value is the octal number, *n*, or a symbol whose value is the octal number *n*. It causes the following *n* "lines" of the symbolic to be ignored at compile time, where a "line" is terminated by a carriage return. Since DECAL statement terminators are carriage returns and semicolons, several DECAL statements can be placed on one "line." For example if the symbolic reads:

```
omit 1
lac a; dac b
lac c; dac d
```

only the "lac c; dac d" will be compiled because carriage return 1 signals DECAL to execute the omit action operator and carriage return 2 signals the omit program that one line has been omitted. This coding could also have been written:

```
omit 1; lac a; dac b
lac c; dac d
```

Semicolon 3 signals DECAL to execute the omit action operator program. The omit program finds carriage return 4 and considers one line omitted.

Probably the bulk of editing desired when implementing options is substituting several lines of coding in place of an alternate group depending on the option chosen.

One way to do this is by defining a symbol such as "16chnsb" to be 1 if compilation for a 16 channel sequence break is desired, and 0 (zero) otherwise.

Then, when there is a point in the program where one group of lines is to be compiled if the program is used with the 16 channel system, but a different group is to be compiled if no sequence break is used, the coding could read as shown in

Figure 1.

Figure 1

```
omit 16chnsb; omit 17
.
.
. 178 line of
. coding only to be
. Included when sequence break is to be used by
. the program
.
omit 16chnsb; omit 1
omit 10
.
. 108 lines of coding
. to be included only when sequence
. break system is not to be used by the program.
```



If 16chnsb = 1 then the 10 line group will be omitted but the 17 line group will be compiled. Reversing the value of 16chnsb will cause the 17 lines to be omitted and the 10 lines to be compiled.

Selection of groups of lines to be compiled depends only on defining the value of one symbol for each option, no matter how often the situation represented by Figure 1 occurs in the program.

Choosing between two groups of lines is not always the most efficient way to edit for an option. It is sometimes easier to modify a symbolic by writing in terms of a dummy symbol which is defined at compile time in one way when one option is chosen and a different way when an alternate option is chosen. For example, suppose that a program is written which uses multiplication in several places, and it is desired to provide a choice between the use of automatic or nonautomatic multiply. Every time multiplication by the number in the register "operand" is desired the programmer should write the dummy statement "mpy operand." Then in the case where automatic multiply is desired, "mpy" is defined as "mul" and the dummy statement compiles as though "mul operand" had been written. For nonautomatic multiply, it is necessary to call the "mpy" subroutine.

In this case, "mpy" is defined so that the dummy statement compiles as the calling sequence for the "mpy" subroutine, namely "jda mpy; lac operand."

The method for determining the definition of "mpy" at compile time is illustrated in Figure 2, where the symbol "mulstep" equals zero if automatic multiply is desired, and 1 for nonautomatic multiply.

Figure 2 - Selecting Dummy Symbol Definitions

```
omit mulstep; mpy ewd mul      . . .this defines mpy as the
                               . . .instruction mul

omit mulstep; omit 3
mpy dig* beg 1v0 opl nlc
jda ths
lac 1 1stend
```

If the multiply option had not been implemented by selecting the definition of a dummy symbol, these more cumbersome lines would have had to be written every

\*See last page of this paper for footnote.

time multiplication was desired:

```
omit mulstep; mul operand; omit 1  
jda mpy; lac operand
```

options involving the use of omit are chosen by defining the values of the parameters used with omit, i.e., setting mulstep and l6chnsb equal to one or zero, in the examples above. To obtain these values the action operator "typeget" has been defined. When "typeget" is followed by a symbol, this symbol is typed out on-line at compile time and the compiler then waits for the operator to type in the value of the symbol. For example, writing "typeget l6chnsb" would cause "l6chnsb" to be typed on-line; the user would type a 1 if he wanted to compile for a 16 channel sequence break system; otherwise a zero. A more self explanatory message such as "Do you want to compile for a 16 channel sequence break system: Type 1 if yes, 0 if no." could also be typed, at the expense of some space in the symbol table.

The typeget action operator can also be used to achieve a different kind of program modification, namely determining buffer sizes and sequence break channel assignments. Normally space for a buffer is reserved by use of the "lve" action operator, which when followed by an octal number n or a symbol whose value is the number n, reserves n registers in the program. Hence "lve 100" is equivalent to "lve buffersize" if buffersize has the value 100<sub>8</sub>. But since the value of symbols such as "buffersize" can easily be obtained on-line at compile time through the use of "typeget," one can make variable buffer sizes an option in any program as easily as making them a fixed size.

The typeget action operator is used similarly to obtain sequence break channel assignments. For example, if "punchchannel" is given the value 10 by means of typeget, and if all sequence break instructions are phrased in terms of punchchannel, i.e., asc punchchannel, isb punchchannel, lio bio\*<sup>\*</sup>punchchannel, etc., the program will compile these instructions with reference to channel 10, i.e., as though the programmer had written asc 10, isb 10 and lio bio 10.

In summary, this paper has described how the action operators "omit" and "typeget" can be used to provide modification of a symbolic at compile time. Modifications made by this method are limited in that the original programmer must have anticipated the desired changes. On the other hand, the person who sets the pattern of omit action operators which are necessary to implement a particular option

\* See last page of this paper for footnote.

is the person best qualified to do so, namely, the original programmer. In many cases, the options which can be provided with this method are easily incorporated into a program symbolic without effort on the programmer's part. From a library distribution standpoint one tape can efficiently serve all users despite incompatible equipment configurations. From a users's standpoint, the reprogramming of routines made necessary by equipment changes can be significantly reduced if the possibility of change was recognized by providing options at the time the programs are written. For my own part, I would especially like to see programs dealing with the 16 channel sequence break system at least grant options for channel assignments.

#### Footnotes

- \* (dig means define instruction generator. The code following the "dig" tells DECAL what instructions to generate each time mpy is seen. In this example mpy is being defined so that when mpy is followed by a symbol such as "operand," the object code produced will be "jda mpy; lac operand").
- \* (bio is a DECAL action operator which calculates the address where the in-out register is preserved for a particular channel, i.e., lio bio 0 is equivalent to lio 2.

---

This paper is identified as DSL-ALPR-63-10, Decision Sciences Laboratory, Hq. Electronic Systems Division, Bedford, Massachusetts.

## APPENDIX

The definitions of the omit and typeget action operators follow. These definitions are written for DECAL-BBN of 26 Sept. 1963, and have been checked out. I want to thank Richard McQuillan of BBN for his help and time given to write these action operators.

```
omit      dao beg
          cal 1521      ...cal prs...process statement
          spa; jmp 155  ...jmp rml...return if negative number
          cma
          dac n         ...store number of lines to be omitted
          cal 2466      ...cal rsy...gulp over ; or cr
          nop          ...if symbol is number
          cal 3163      ...cal com...skip over comment
          isp n; jmp → -4
          jmp 155      ...jmp rml
n:
end
xsy n
```

```
num      ewd 100
typeget dao beg
          cal 2466; hit ...cal rsy
          cal 2120      ...cal sch
          cal 2253      ...cal enter
          lac 753       ...lac mwd
          dip'345       ...dip'nsd
          cal 3405      ...cal tcr
          lac 345; cal 3331 ...cal tts
          cal 3407      ...cal ttb
          dzm num
b:        clf 1
          szf' 1; jmp → -1 ...get octal number
          tyi; cla
          rcr 6; s'za'; jmp a; rcl 3
          lio num; rcl 3; dio num
          jmp b
a:        lac 351       ...lac cad
          dap'345
          lac num; dac'351 ...store value as word
          idx 351       ...idx cad
          jmp 155      ...jmp rml
end
xsy num a b
```

These action operators may be added to DECAL using the procedures described in the manual.

# A VERSATILE PROGRAMMING SYSTEM FOR LARGE PDP INSTALLATIONS

Theodore R. Stollo

Air Force Cambridge Research Laboratories  
Bedford, Massachusetts

## ABSTRACT

A programming system has been developed for the Dynamic Experimental Processor at AFCRL which provides users with two new and versatile programming languages - Amp and Aidex. Amp is an assembler featuring:

- 1) A powerful Fap-type macro compiler
- 2) An optional assembly listing
- 3) Exceptionally detailed error diagnostics

Aidex is a compiler featuring:

- 1) Automatic manipulation of floating point variables
- 2) Fortran - type statements
- 3) An optional compilation listing

These languages produce fully relocatable output with symbolic linkage capabilities. The languages are used in conjunction with a monitoring system. The monitor automatically selects the appropriate language for translating user's program from the magnetic system tape. This system tape also contains a number of utility routines as well as a separate file for relocatable, linkable library programs.

## INTRODUCTION

In March of 1963 work was begun at AFCRL on a new programming system for the DX-1 and other large PDP installations. The system, which would take advantage of the auxiliary storage and asynchronous input-output capabilities of the DX-1, was proposed with several objectives in mind:

1. A programming system should be rapid and easy to use to minimize the time and effort required to get a program from the flow chart to a binary coding and to encourage symbolic corrections to programs instead of binary patches.
2. Good error diagnostic communication between the language and the user must exist to completely describe the type of error involved and to accurately point out the position in the source program where the error exists.
3. The basic statement format should be kept simple with the more difficult format specifications attached to the infrequently used options.
4. A large, easily maintained, readily accessible library of useful subroutines must be available to take advantage of previous programming efforts.
5. The more significant features of the Decal, Fap, Fortran, Frap, and Macro programming systems should be available in the proposed system.

The result of the work applied to meet these specifications is a programming system called ASYS (pronounced <sup>\*\*</sup>ā sīs). In order to make ASYS easy to use, a control system,

the ASYS MONITOR, was developed. The MONITOR is the first record of the ASYS (magnetic) SYSTEM TAPE.

Simple control statements enable the MONITOR:

1. to call the ASYS languages from the ASYS SYSTEM TAPE
2. to load and link the relocatable binary magnetic tape output of the ASYS languages.
3. to search and selectively load subroutines from the ASYS LIBRARY (which is a file on the ASYS SYSTEM TAPE).
4. to call the ASYS debugging system (which has been named ADT because of its similarity to DDT).
5. to call a number of utility routines (such as Expensive Typewriter, Expensive Desk Calculator, etc.) which have been written on the ASYS SYSTEM TAPE.
6. and to select the ASYS EDITOR which performs many useful magnetic tape control functions (such as write, read, space, search, and copy) and which maintains the ASYS SYSTEM TAPE itself.

The first language written for ASYS is called AMP.

AMP is a two-pass assembler with pass 1 from the paper tape reader and pass 2 from the intermediate, magnetic tape output of pass 1. AMP pass 2 produces relocatable, linkable object programs on magnetic tape. The object programs are loaded into core by the ASYS LINKING LOADER under the control of the ASYS MONITOR. Thus, the symbolic to binary translation process with AMP involves 2 assembly passes and 1 loader pass, but only 1 pass is from the paper tape reader. Therefore, AMP requires less handling of paper tape than any present PDP-1 language.

A subroutine, when it is fully debugged, may be reclaimed from the magnetic tape in punched paper tape relocatable binary form for loading with subsequent programs or for appending it to the ASYS LIBRARY. This eliminates the necessity of assembling a subroutine each time it is needed by a program.

AMP features a very powerful macro-compiler with capabilities very similar to BELL Macro-Fap and Midas. Macros may be nested and are capable of the macro-call argument list. Definite as well as indefinite repeats are permitted in macro-expansions.

AMP allows Decal-type indefinite length symbols for program tags as well as macro names and macro definition dummy argument names. AMP makes available a very large symbol table to permit the assembly of very long programs.

A significant feature of AMP is the optional assembly listing and/or symbol table map. The assembly listing presents a side by side octal representation of the assembler's output with the associated symbolic statement in a format similar to Modern Frap and Fap listings. The listing is extremely useful both for debugging programs and for presenting a complete final listing of the programmer's efforts. AMP offers the assembly listing using the sequence break asynchronous input-output method with several media:

1. the on-line typewriter
2. the on-line punch for off-line flexoriter listings
3. a magnetic tape for off-line listings with a 1401 or similar tape to printer process.

A second language has been proposed for ASYS and would be a compiler called AIDEX. AIDEX would be similar to Fortran-Mad type compilers producing relocatable binary output capable of linkage with AMP assembly programs and ASYS LIBRARY programs. Until AIDEX or another true compiler is available for ASYS, many useful permanent macros have been defined for AMP to enable dimensioning of arrays, subscripted variables, and counted iteration loops with compiler-type programming format.

The ASYS debugging routine (ADT) is useful for symbolically examining and changing selected core registers. ADT permits expressions, using the SYSTEM SYMBOLS from the ASYS LOADER'S symbol table, to be used in selecting registers and making changes. Several break points have been provided to interrupt programs at desired points, interrogate the contents of certain core or machine registers in symbolic, octal, decimal, or floating-point, then proceed from the break point.

ASYS has grown from an idea to a functioning, useful experiment at AFCRL. It has shown us that a small computer system can present all of the programming ease and capabilities of large computer systems while maintaining an informal, individual user, man-machine atmosphere.

\*AFCRL - Air Force Cambridge Research Center

\*\* The ASYS System consists of:

1. A control monitor and relocating linking loader
2. An assembler "AMP"
3. A compiler "AIDEX"

These are available from DECUS and may be requested as DECUS Library No. 53. The System also has a debugging program, DECUS Library No. 54.





FLINT 36 A3D  
DESCRIPTION AND OPERATING PROCEDURES

Jacob M. Baker and David J. Isenberg  
Charles W. Adams Associates, Inc.  
Bedford, Massachusetts

ABSTRACT

FLINT is an interpretive routine that permits the Digital Equipment Corporation PDP-1 to perform double-precision floating-point arithmetic, input, output, and elementary function evaluation. Originally written in FRAP for use in lens design work (though nonetheless a general-purpose program), FLINT has been translated into DECAL to be compatible with other programs in this language. Arithmetic and function evaluation are performed interpretively, input and output are handled by closed subroutines addressed directly by the user's programs, and overall format control is left to the user's routines.

INTRODUCTION

Since FLINT, written in FRAP, was released about a year ago by Itek Corporation, through The Digital Equipment Computer Users Society, there has been considerable demand for improved documentation and a revised listing. As a service to DECUS, Adams Associates offered to undertake the conversion and redocumentation of FLINT, and has done so with the permission and assistance of Itek. The results of its work are reported in the paper. New FRAP and MACRO listings will be made available by Adams Associates. Other modifications are being considered. Among these are the production of a totally relocateable version of FLINT, the removal of exponent bias, and the addition of other floating-point instructions such as a floating index.

INSTRUCTION REPERTOIRE

The instructions currently available for the interpreter are listed below:

Floating Operations		Operation Code
<u>Function</u>	<u>Mnemonic</u>	
Deposit floating accumulator	fda	00
Floating add	fad	02
Floating subtract	fsu	04
Load Floating accumulator	flo	06
Floating square root	fsr	24
Floating sine	fsi	26
Floating cosine	fco	30
Floating skip	fsk	32
Floating multiply	fmu	54
Floating divide	fdi	56
Floating operate	fopr	76

## Entering Interpreter

<u>Function</u>	<u>Mnemonic</u>	<u>Octal Code</u>
Enter interpretive mode	cal . .	160000
Enter interpretive mode and load floating accumulator	cal y	16yyyy

### FORMATS

Floating-point quantities are expressed in the form  $y \cdot 2^x$  where the magnitude of  $y$  is less than one. Arithmetic is done using a floating-point accumulator (FLAC) which consists of four storage registers. The absolute value of  $y$  is stored to double-precision accuracy in the first two registers, the sign of  $y$  in the third, and  $x + 11$  in the fourth. With a bias of +11, the exponent ranges from -42 to +20. This range was selected by Itek as being most useful for their work.

Operands for floating-point instructions are assumed by the interpreter to be stored in either two or three consecutive storage registers, depending on whether Program Flag 5 is off or on. In the two-register format (Program Flag 5 off), bit 0 (bits being numbered 0 to 17 from left to right) of the first register contains the sign of  $y$ . As shown in the diagram below, the first 17 bits of the absolute value of  $y$  are stored in bits 1-17 of the first register, and the remaining 12 in bits 6-17 of the second register. Bits 0-5 of the second register contain the signed quantity equal to  $x$  plus the exponent bias.

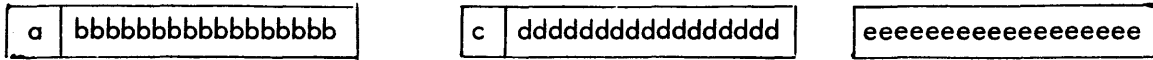


- a sign of  $y$
- b first 17 bits of  $y$
- c  $x$  plus exponent bias
- d final 12 bits of  $y$

### TWO-WORD FORMAT

In the three-register format (Program Flag 5 on), as illustrated below, bit 0 of the first register contains the sign of  $y$  and bits 1-17 are the first 17 bits of the absolute value of  $y$ . Bit 0 of the second register is always zero and bits 1-17 contain the remaining bits of the absolute value of  $y$ . The third register contains the value of the exponent incremented by the exponent bias. This three-word format is especially useful for

saving and restoring FLAC and is often used only for that purpose.



- a sign of y
- b first 17 bits of y
- c zero always
- d final 17 bits of y
- e x plus exponent bias

### THREE-WORD FORMAT

Instructions to be processed interpretively are written in the same format as normal PDP-1 instructions and are assembled with a five-bit operation code, an indirect address bit, and a twelve-bit address. This address refers to two or three consecutive locations, depending on the position of Program Flag 5. Thus, in the description below of the interpreted operations, the symbol C(Y) refers to the contents of locations Y, Y+1, and optionally Y+2, where Y is the address part (after indirect addressing, if any, has been performed) of the instruction being interpreted. If Y is zero, the instruction is interpreted as referring to FLAC itself.

There are eleven floating-point interpretive instructions which, with their overflow and underflow conditions, are described in detail later.

When floating-point operations are to be performed, it is necessary to enter the interpretive portion of FLINT. This is accomplished by the PDP-1 instruction cal, which transfers control to location 101a with the location of the next instruction to be interpreted in the accumulator. Since it may often be necessary to enter and leave the interpretive mode, the cal instruction is interpreted as a floating load (flo) as well as an entry instruction whenever the address of the cal is other than zero. Indirect addressing may not be used with the cal instruction since this is assembled as a jda instruction; therefore, if indirect addressing is desired, the correct sequence of instructions would be cal . . ; flo 'Y;

The interpreter is so arranged that once the cal instruction is encountered, it will regard each succeeding instruction as a floating-point instruction until it encounters an exit instruction. Any instruction with an operation code number of 10 through 23, 34 through 47, or 60 through 75 will be regarded as an exit instruction with the exception of 16, the cal instruction.

Instructions with these operation code numbers will be simultaneously executed and used as exit instructions when encountered in the interpretive mode. All succeeding instructions will be considered normal machine instructions until another cal is encountered. Thus, such instructions as xor - operation code 06, and - operation code 02, or dio - operation code 32, may not be used in their normal sense while in the interpretive mode. The instructions whose operation codes have thus been pre-empted by floating instructions were selected because they are unlikely to be used while in floating mode. It is important to note that, once in the interpretive mode, instructions not having the operation codes cited in the preceding paragraph will be interpreted as floating instructions whether or not they are so intended.

### UNFLOATING ROUTINE

The instruction jda unflo enters a subroutine which converts the floating-point number stored in FLAC to a fixed-point integer. This integer is equal to the value of the contents of FLAC divided by the quantity two raised to the power of the contents of location fixexp. The integer resulting from this conversion is stored in the accumulator and the contents of FLAC are destroyed. (The unflo subroutine truncates rather than rounds the quotient obtained by dividing two to the appropriate power into C(FLAC). Thus if FLAC contains  $1.4_8$  and fixexp contains 0, jda unflo will put 1 into the accumulator; if FLAC contains  $1.4_8$  and fixexp contains 1, jda unflo will put 0 into the accumulator; if FLAC contains  $1.4_8$  and fixexp contains -1, jda unflo will put 3 into the accumulator.)

### INPUT ROUTINES

There are three input subroutines which, like the output subroutines, are addressed directly from the main program. The first, entered by the instruction jda readc, reads translates single characters. The second, entered by the instruction jda readg, handles groups of characters. Each of these two routines reads from punched tape or from the console typewriter, depending on whether the input control word (icword) contains taper (for tape) or typer (for typewriter). FLINT is arranged so that icword contains taper unless this is altered by the user's routine. Such alteration is accomplished by writing: lac taper; dac icword; etc.

After a character is read, it is compared with the entries in a table containing the standard FIO-DEC Code for each character as well as a control code that may have one of eight different values. Code 0 marks characters to be ignored, such as illegal configurations which do not correspond to typewriter or Flexowriter symbols. Code 1 marks

characters such as space or tab, which serve as delimiters indicating the end of an alphanumeric word. Code 2 marks the decimal digits 0-9 and Code 3 marks the symbols used in floating-point numbers, such as a minus sign or a period (used as a decimal point). Codes 4-7 are assigned to the alphabetic characters; only one bit is tested and all characters having any of these four codes are treated identically.

The readc routine reads a single character, looks it up in the table to find the control code, and returns to the main program with the concise code (with 20 and 0 reversed) in bits 12-17 of the accumulator, which elsewhere is filled with zeros and the iotble entry in IO. If the control code is 0, another character is read and processed in the same manner before returning to the main program.

The readg routine reads numerical or alphabetic groups and determines which group is being read by noting the control code of the first character. If the code is 4 through 7, the group is alphabetic; if 2 and 3, it is numeric; if 0 to 1, the character is ignored and the next character treated as the first.

When reading from paper tape, location buff4 must be set to zero before a call to readg the first time that this instruction is called, and if successive calls to readg are interspersed with calls to any of the other read routines which are also reading from paper tape.

If the group is alphabetic, the characters are translated and their concise codes are saved until either a delimiter (control code 1) is encountered or four characters with control codes 2 through 7 have been read. Characters with control code 0 are always ignored.

The concise codes of the one, two or three characters preceding either the delimiter or the fourth character are then assembled in the accumulator, each occupying six bits with the first one to the left and the whole group right-justified, with zeros on the left if necessary. The control and the concise codes of the delimiter or fourth character are put in IO bits 0-2 and 12-17, respectively. Program Flag 4 is on if four characters were read, and off if a delimiter was encountered. Control is then returned to the main program.

If the group is numeric, characters are read until a delimiter or a character with control code 4 through 7 is encountered. A plus or minus sign may, but need not, appear anywhere in the number, and there may be a maximum of ten decimal digits. (In FLINT,

a plus sign is indicated by "(", a left parenthesis, rather than by "+," the conventional plus symbol. If there are two or more minus signs, all but the last are ignored.)

If a decimal point appears, the resulting number is considered to be a floating-point integer and is formed in FLAC, Program Flag 4 is turned off, and overflow or underflow is signalled as in floating add. If two or more decimal points appear, all but the last are ignored. If no decimal point occurs, the result is considered to be a fixed-point integer, Program Flag 4 is turned on and, if it exceeds 131, 071 in magnitude, Program Flag 6 is also turned on. The fixed-point integer appears in the accumulator when control is returned to the main program. Whether the integer is floating-point or fixed-point, the control and the concise codes of the character which served as a delimiter appear in IO bits 0-2 and 12-17, respectively, and the previous contents of FLAC are destroyed.

The third subroutine, entered by the instruction jsp buff, brings characters from paper tape to the IO register. Before the jsp, the instruction dzm buff4 should be given. The first succeeding jsp buff instruction will then read enough characters from paper tape ( $45_8$  as the buffer length is now set) to fill the buffer and put the Flexowriter code of the first character into IO bits 10-17. The next jsp buff instruction places the second character read from the buffer into IO bits 10-17, and each such succeeding instruction brings another character from the buffer into the IO register until all the characters have been brought in. The next jsp buff instruction reads another buffer full of characters from tape, and the entire process is repeated.

## OUTPUT ROUTINES

There are three output subroutines, all of which write information on punched tape, the console typewriter, or both, depending on whether the output control word, location ocword, contains tapew (tape only), typew (typewriter only), or bothw (tape and typewriter). There is also the write-IO routine (entered by the instruction jda writio) which writes on paper tape the eight-bit character contained in bits 10-17 as many times as specified by the number in IO bits 0-7. If IO bits 0-7 are zeros, the eight-bit character is written once. No look-up or conversion is performed and the character is written on tape regardless of the contents of the output control word.

The write-character routine, (entered by the instruction jda writc) writes the six-bit concise code character contained in IO bits 12-17 as many times as specified by the

contents of IO bits 0-7, using the same convention as the write-IO routine.

The write-integer routine (entered by jda write) writes the integer in the accumulator converted to decimal form, followed by the character in IO bits 12-17. The final character may be written repeatedly according to IO bits 0-7 in the same manner as the write-IO routine. Insofar as the sign and initial spacing or zero suppression is concerned, the format is controlled by the value of the format control word, format.

The write-floating routine (entered by jda writf) writes the contents of FLAC converted to decimal form, followed by the character in IO bits 12-17 exactly as in the write-integer routine. The contents of FLAC are destroyed after calls to either the write or the writf routine.

Format control is specified by the contents of location format as follows:

- |            |   |   |
|------------|---|---|
| Bits 0-5   | - | The number of digits to the left of the decimal point. If zero or less than the number of significant digits, all significant digits will be printed; otherwise spaces or zeros will appear on the left to fill out the required number of spaces to right-justify the column; this must be $12_8$ or less for fixed-point numbers. |
| Bits 6-11  | - | The number of digits to the right of the decimal point. This must be zero for fixed-point integers; if zero for floating-point numbers, no decimal point will be printed.   |
| Bits 12-14 | - | Sign control. If zero, no sign will be printed; if 1, 2 or 3, a minus sign will be printed for negative numbers and nothing, space or plus sign, respectively, for positive numbers.  |
| Bits 15-17 | - | Zero control. If zero, spaces are used in place of initial zeros; if one, initial zeros are printed, this being useful for handling long integers and fixed-point numbers other than integers.  |

The contents of format may be altered by the following sequence of instructions: lac nf; dac format; etc., where nf contains the desired contents of format.

Listed below are system symbols declared by FLINT; therefore, they should not be used by a program which uses FLINT and is assembled with it:

iotble	writio	ocword	taper
fixexp	bothw	writc	icword
unflo	tapew	readg	readc
writf	typew	buff	buff4
write		typer	format

## Description of Instructions

- flo - floating load: Unpack C(Y) from its two- or three-word format into the four-word format and place in FLAC.
- fad - floating add: Place the arithmetic sum of C(Y) and C(FLAC) in FLAC. If the sum is greater than  $2^{131061}$ , the result is incorrect and Program Flag 6 is turned on. If the result is less than  $2^{-131084}$ , or if the mantissa of the sum is zero, the mantissa of FLAC will be positive zero and the exponent of FLAC will be -42 upon completion of the operation. Such astronomical exponents can be obtained only because an entire 18-bit word is allocated to the exponent in FLAC.
- fsu - floating subtract: C(Y) is subtracted from C(FLAC) and the difference is put in FLAC. Overflow and underflow are handled as in floating add.
- fmu - floating multiply: The product of C(Y) and C(FLAC) is placed in FLAC. Overflow and underflow are handled as in floating add.
- fdi - floating divide: C(FLAC) is divided by C(Y) and the quotient is put in FLAC. Overflow and underflow are handled as in floating add.
- fsr - floating square root: The square root of C(Y) is put in FLAC if C(Y) is positive. Overflow conditions are not possible. If C(Y) is negative, the contents of FLAC are left undisturbed and Program Flag 4 is turned on.
- fsi - floating sine: C(Y) is treated as an angle in radians. The sine of this angle is put into FLAC. Error conditions are not possible.
- fco - floating cosine: Cos C(Y) replaces C(FLAC) as in floating sine.
- fda - floating deposit accumulator: C(FLAC) is packed into the two- or three-word format depending on the position of Program Flag 5, and deposited into locations Y, Y+1, and optionally Y+2. With Program Flag 5 off, if the magnitude is as large as  $2^{20}$ , Program Flag 6 is turned on. If less than  $2^{-43}$ , the quantity deposited has a mantissa of zero and an exponent of -43. If Program Flag 5 is on (three-word format), no such check is performed.
- fsk - floating skip: The interpreter clears the IO register and sets the sign of the accumulator to the sign of C(FLAC), then loads the most



significant bits of the mantissa in bits 1-17. It then skips or executes the next sequential instruction, depending on whether the condition tested for is true or false.

fopr - floating operate: This instruction places the sign of FLAC in the accumulator, executes the instruction specified by the address part of the fopr (e.g., fopr 200 - clear accumulator and therefore sign register) and returns the result to FLAC.

It is possible that the fopr specified may not change the accumulator (e.g., fopr 15 - set Program Flag 5). In this case the operation will leave the sign of FLAC unchanged.

In preparing a DECAL symbolic tape which will make use of the floating skip and floating operate instructions, the required format is fsk or fopr followed first by the indirect bit if required, and then by the address of the appropriate skip or operate instruction. Thus a floating skip on non-zero accumulator would be written as fsk ' 100 and a floating complement accumulator as fopr 1000.

#### Possible Modifications by Users\*

Partially relocatable version:

All but the first 100<sub>8</sub> instructions for FLINT may be relocated. To do so, the following changes should be made in the symbolic tape:

1. The instruction immediately before the comment "divide here" should be followed by "blk" and "fin"; this is the end of the fixed part.
2. The instruction immediately after the comment "divide here" should be preceded by "blk"; this is the beginning of the relocatable part.
3. The following should be declared as system symbols at the beginning of the fixed part:

norm4	fadr
flor	fsur
a5	fsrr
a3	fsir
a4	fcor
5y	fskr
brkpt	fmur
fdar	fdir
	foprr

\*(A Print-out of the Symbolic Tape was omitted in the publication but may be obtained from DECUS.)

These symbols must be located in the relocatable part and their delimiters changed to " ' " (apostrophe).

4. The following should be declared as system symbols at the beginning of the relocatable part:

q  
a2a  
a1  
pc

These symbols must be located in the fixed part and their delimiters changed to " ' " (apostrophe).

5. The two parts should be assembled and two loader tapes obtained. The fixed part must be loaded into locations starting at  $100_8$ . The relocatable part may be loaded into any  $2051_8$  consecutive locations.

Expansion of input buffer:

The size of the "read group" buffer area may be altered by changing; first, the number currently set at buff42 to the desired value; secondly, the number currently set at buff1+1 to the new value in buff42-1; and, thirdly, the number currently set at buff2a+4 to the new value in buff42.

#### ACKNOWLEDGMENT

Adams Associates wishes to acknowledge with thanks the substantial contribution made by Edward J. Radkowski of Itek Corporation to the revision of FLINT. Readers of this paper are invited not only to request additional copies of it from Adams Associates but also to forward to the company any suggestions or criticisms. These should be marked to the attention of David J. Isenberg or Jacob M. Baker.

## THE MIDAS ASSEMBLY PROGRAM AND THE PDP-1 \*

Robert A. Saunders  
Information International Inc.  
Maynard, Massachusetts

### ABSTRACT

An advanced assembly program for the PDP-1 is described. The assembler processes up to 6 character symbols and has a highly sophisticated macro-instruction processor similar to that in Bell System FAP for the IBM 7090. The source language has been kept compatible with that of MACRO except for some necessary changes in the format of macro-instruction definitions. Product and logical syllable combination operators permit greatly increased flexibility in the source language. Various versions of the assembler are planned for various machine configuration.

\*The Midas Assembly Program will be available on request. The editor regrets that there was not sufficient time to include the 38 pages in this publication.



## Section III

### PROBLEM ORIENTED TECHNIQUES



# THE PDP-1 AS A VERSATILE RESEARCH TOOL

W. Fahle and D. Brand

Systems Research Laboratories, Inc.  
Dayton, Ohio

## ABSTRACT

This paper describes work supported by and in conjunction with the Biodynamics and Bionics Division, Biophysics Laboratory, Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Ohio, Contract AF 33 (616)-8280, and presents the utilization of the PDP-1 as a laboratory instrument providing considerable research capability. The paper briefly shows how communication barriers among disciplines were circumvented.

Analog to digital conversion and data analysis techniques demonstrate the use of the PDP-1 as both a valuable analysis tool and a satellite to a large scale computer facility. Output formats and result displays are described to further emphasize the applications of the computer system.

## INTRODUCTION

Work supported by and performed in conjunction with the Biodynamics and Bionics Division, Biophysics Laboratory, Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Ohio, (Contract AF 33(616)-8280) is described. The paper explains the utilization of the PDP-1 as a laboratory instrument which provides considerable research capability to a mixture of disciplines. The discussion is in the form of a progress report by Systems Research Laboratories, Inc., Dayton, Ohio. All but the final phase of a development program designed to integrate a digital computer into the research efforts conducted by the Biodynamics and Bionics Division is covered.

The computer installation is used both on-line and off-line by physiologists, neurologists, mathematicians, physicists, and engineers for work in the areas of on-line experiment monitoring, data conversion and analysis, and IBM satellite applications.

## SYSTEM COMPONENTS

### Standard and Optional DEC Equipment

The basic PDP-1 has a 4K memory, paper tape, a typewriter, and a single channel Sequence Break System. It is capable of automatic multiply and divide. Additional equipment includes the Type 23 high speed data channel, the Type 30 precision cathode-ray tube display, the Type 32 light pen, and the Type 50-51 magnetic tape transport and control. Figure 1 is a block diagram of the system.

### Other Components

Complementing the basic equipment are a Raytheon AD-50A converter with a type DM 120 multiplexer. The inputs are Magnacord and Ampex tape recorders, an SRL PPM ramp decoder, and an analog cochlea. The SRL Model 408 Remote Control Console has two slave 5-in. cathode-ray tube displays paralleling the digital scope and an amplitude sampler.

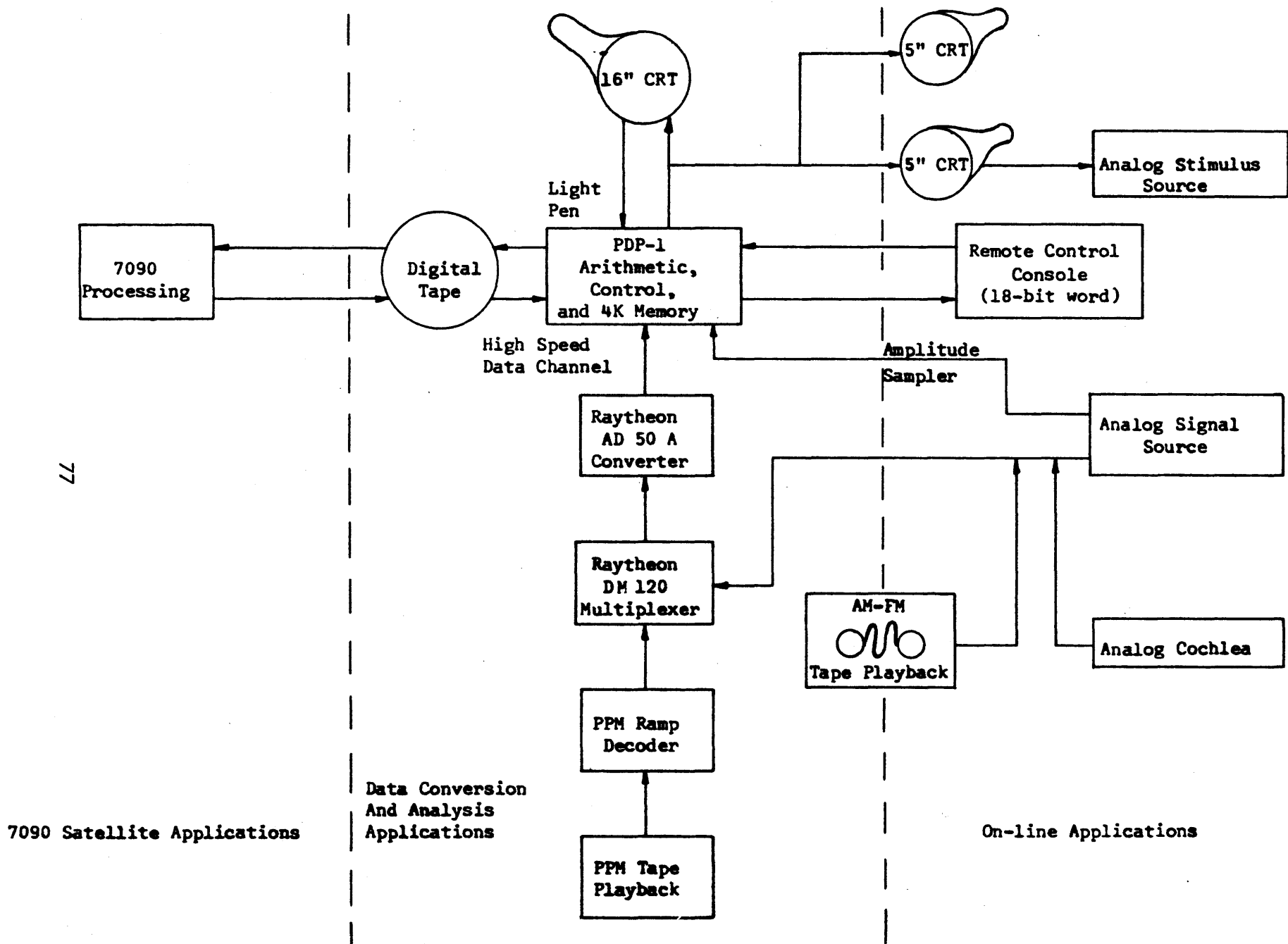
## COMPUTER APPLICATIONS

### On-Line Applications

The Remote Control Console designed by SRL is a device which may be operated at several different sites distant from the computer. An effective communication channel between the remote control console operator and the PDP-1 is maintained using the sequence break system and special computer input-output options. When the operator wishes to either check or change machine status, he activates an "ENTER" switch on the remote console which triggers the sequence break system. Program control is transferred (provided the PDP-1 is in the Sequence Break Mode) to storage location 3 where the monitor begins. The operator has 4 possible operate modes (RUN, CHANGE PROGRAM, DISPLAY PARAMETER, and CHANGE PARAMETER), 20 program numbers, 20 parameter numbers, and a 3 decimal digit number to use in defining and setting up a program. Each selection is coded into an 18 bit remote control word and a flag setting as follows:

<u>flag 6</u>	<u>remote word bits</u>			<u>meaning</u>
	0	1 - 5	6 - 17	
off	0	not used	not used	RUN in last program selected
off	1	parameter no.	not used	DISPLAY selected PARAMETER in the last program selected
on	0	parameter no.	parameter value	CHANGE value of selected PARAMETER in last program selected to given value
on	1	program no.	not used	CHANGE selected PROGRAM no. to given value





77

7090 Satellite Applications

Data Conversion  
And Analysis  
Applications

On-line Applications

Figure 1. PDP-1 Data Processing System

At the completion of each decoding and monitor function the remote monitor enters a wait loop for next remote control command. A permanent status log is maintained by the console typewriter.

The amplitude sampler is a data collection device controlled at the remote console. Given a signal and a preset threshold voltage level, the amplitude sampler may be interrogated to determine whether the signal has exceeded the threshold since the last interrogation. If it has (i.e. a pulse has occurred), bit zero of the IO is set to one upon execution of the special amplitude sampler command. No pulse status is indicated by a zero bit zero.

The two slave scopes are used to monitor results and digital-analog conversion provided by the on-line 16" CRT. A programmed function generator is designed to derive modulation waveforms on the digital display, which are then used remotely as controlled stimulus sources administered to animal subjects.

The basic software for the system consists of a remote console interpreter program, a program monitor, and a basic set of subroutines containing the alpha display, binary-to-bcd conversion subroutines, etc. This basic package remains in core at all times. When a program change is requested, the program monitor searches magnetic tape for the desired program and reads it into the common program store. All the programs may reference the basic subroutine set, and all are under control of the remote console interpreter. In this way many different programs may be used at the same experimental site, or different experiments at different sites may use the computer without the necessity of operation at the site of the computer.

The physiological data processing applications can be described in three categories: muscle, neuron, and cochlea responses. Located physically apart from the central processor, the remote control console serves the purpose of assisting (via the PDP-1) the researcher in conducting an experiment. Manipulation of parameters and programs gives the physiologist enough flexibility to effectively monitor and/or magnify results being obtained in real time, thus minimizing erroneous data recorded permanently on analog tape.

The cat and rabbit muscle response program is used to investigate response characteristics under varying stress stimuli. The program parameters include date, run number, experiment number, subinterval width, number of points, and various scale and input-output options. The analog signal for one stimulus-response relation is immediately

displayed then edited while computations are made to reflect points of interest. The results are then typed on-line, the response data is written onto tape, and the edited signal is displayed to the researcher along with experiment identification for photographing. The tape containing digitized response data is then processed on the 7090 and log-log results of Fourier transforms are returned to the PDP-1 for display and photography purposes.

The neuron response program set is used to examine neural transmission characteristics evident under repetitive stimuli. Input parameters to these programs include date, run number, experiment number, delay time, number of subintervals, subinterval width, display block, and various scale and input-output options. Several modes of analysis are provided to the researcher in order to establish fundamental properties of pulse rate information. The histogram is the basic graphical tool used to exaggerate pulse response frequency. Variable window widths to identify pulse or no pulse and a sliding time base to "fit" an optimum analysis time to the response are controlled remotely by the researcher. Raw neuron responses are represented as a string of zeros and ones (developed by the amplitude sampler command) and are written onto digital tape for reprocessing.

Problems in speech recognition are also accommodated on-line utilizing an electronic instrument which simulates the mechanics of the basilar membrane. The analog cochlea, as it is named, responds to spoken sounds in the same manner as the human cochlea. The resultant data from the converter is a sequence of samples which outline the profile of amplitude over time. The variable intensity feature of the digital display effectively presents the fading out of a particular pattern as the sound stimulus changes. In this way, various sound stimulus-response relations are compared.

#### Data Conversion and Analysis Applications

A vitally important role of the PDP-1 in providing research flexibility is that of data conversion. High speed analog-to-digital conversion is required in practically every on-line application of the computer.

The conversion system is composed of a Raytheon Model DM 120 Multiplexer and a Raytheon Model AD 50 A Converter. The multiplexer can accept up to 14 channels of analog information from a variety of inputs which includes analog tape transports, an analog cochlea, and several remote signal sources. The converter can sample at many different rates which may be manually selected to use the high speed channel or may

be under program control. An upper bound of 200 KS (kilosamples) is governed by the 5  $\mu$ s cycle time of the computer although the converter can effect a 5 MS (megasample) rate. The 50 ns aperture time of the converter provides excellent resolution with a bandwidth of 0-20 kc, and baseline drift is negligible. Word packing the 8-bit samples is optional. Continuous sampling and storing over extended time periods is limited to 10 KS by the data transfer rate between the computer and the Type 50-51 tape transport and control. Thus far this restriction has not hampered the analysis stage of waveform reduction since the bandwidth of the raw data, in most cases, does not exceed 5 kc.

Back to back with the conversion system and the associated software is the program set which is used to further reduce and edit digital data once it is either in memory or on digital tape. Pattern recognition programs are necessary to isolate characteristics of waveforms for the purpose of synchronization, development of recursion rates, filtering, etc. Data reduction, in some instances, occurs simultaneously to compress sample sizes into a sequence of representative quantities relating points of interest, zero crossings, integrals, periods, etc. One such example is a neuron response represented by a string of zeros and ones which indicates pulse or no pulse from the amplitude sampler. Editing, in most cases, is done with the light pen since decision processes involved in determining data validity are usually subjective. The light pen technique has been extremely useful in smoothing out artifacts in physiological data and, thus, in preserving the time basis required for correlation studies.

The library developed for data analysis was designed to accommodate the needs of the many research disciplines and includes programs to derive auto-correlation functions, power spectra, and Fourier transforms. Most of these analysis programs are used to accentuate primary and secondary wave components of complex waveforms. An experiment to define frequencies at which the brain is "driven" uses the power spectra resulting from a Fourier transform of the EEG auto-correlation function. Data is played back from analog tape, sampled at a rate chosen by the researcher, processed by the PDP-1, and results are displayed.

Preliminary statistical computations are made on discrete sample spaces using a generalized statistical program. This program accepts input from paper tape, magnetic tape, and the analog-digital converter. Typed output consists of sample size, mean, standard deviation, variance, and frequency distribution while the latter is displayed in  $\sigma/8$  intervals from the mean to  $\pm 3\sigma$ . Other statistical evaluations such as regression

curves and correlation coefficients, are made using variations of the basic program.

A numerical analysis package of commonly used subroutines is in development and will be available soon. Simpson's rule integration and Lagrangian interpolation and differentiation presently comprise the completed part of this effort.

### IBM 7090 Satellite Applications

The PDP-1 has been used successfully in satellite applications for the IBM 7090 of Aeronautical Systems Division at WPAFB. Previous to the installation of the PDP-1, data were converted manually to cards for input to auto-correlation and power spectra programs on the IBM 7090. Consequently the requirement for the use of the PDP-1 as an off-line converter for the larger computer facility was imminent.

PDP-1 programs to convert, edit, and write data onto digital tape in the standard IBM format at low density (200 lines/in.) were prepared. However, both because of the special binary input tape format required by existing FORTRAN programs and because of the problem inherent in matching a foreign read-tape program to a FORTRAN program, an interpreter program written in IOCS (in-out control system) is used on the 7090 as an intermediate stage of processing to prepare a FORTRAN tape from the complemented, word-packed PDP-1 tape format. This process runs at tape speed and can precede the execution of the FORTRAN program set.

The PDP-1 is also used effectively in displaying results computed on the 7090 for an instant evaluation and record. Such work has been done in the examination of Fourier transforms of cat muscle responses.

### CONCLUSIONS

To summarize, the PDP-1 has been found to be effective in circumventing the communication barriers inherent in mixed-discipline research programs by utilizing many input-output devices (e.g., digital tape, typewriter, and display). In the aforementioned areas of application, on-line, data conversion and analysis, and IBM 7090 satellite, the PDP-1 computer installation has served physiologists, mathematicians, physicists and engineers in a wide variety of ways. To the physiologist, the computer is a monitor that assures homogeneity in experimental procedure and describes response phenomena, both graphically and analytically. To the applied scientist, the computer is an analyst which can develop meaningful trends and transforms from sampled complex

waveforms. It is also a statistician which can accept discrete data points and derive moments and distribution functions, regression, equations, and tests of hypotheses. To the engineer, the computer affords a capability of establishing reliability criteria for instrumentation, filter design, recording-reproduction, etc. Auto-correlation and power spectra computations magnify frequency response characteristics and suppress noise levels that occur in raw measurements made under varying conditions.

Naturally, as the familiarity of laboratory personnel with the equipment increases, the significance of the computer in the role of research support also increases. At the same time there arises a need for additional system components to accommodate the increased work load. The addition of another tape unit with a control that will provide more programming freedom as a result of the addition of automatic block read-write hardware is anticipated. An expansion of the program library to include more analysis capability and more on-line programs is necessary and inevitable. Hardware to effect computer control of a 6-degree-of-motion experimental device is currently planned in the digital-to-analog applications.

The 24-hr/day computer utilization schedule anticipated within several months certainly speaks for itself and demonstrates conclusively that the PDP-1 is a truly versatile research tool.

## TIME SHARING IN THE PROCESSING OF NUCLEAR RESEARCH DATA

A. J. Ferguson, B. Miles, J. Leng

Atomic Energy of Canada, Limited  
Ontario, Canada

### ABSTRACT

Several independent laboratories carrying out nuclear research at Chalk River will share a PDP-1 computer for data storage and processing. It is planned to use a 4-core module system in such a way that: (a) Data will be received by two modules continuously and independently of the computer control circuits. (b) Transfer of such data to magnetic tape storage will be effected by high priority sequence breaks. (c) Console operation can proceed at the lowest priority level.

### INTRODUCTION

This paper will discuss the problem of using the Chalk River PDP-1 system when the standard console operation is to be interrupted every ten minutes or so for a few days at a time by an on-line experiment requiring data transfer and storage. During standard console operation the computer is often stopped, and is rarely in the sequence break mode. It is hoped that certain modifications will allow this time sharing to be feasible, and not too restrictive to the operator.

### APPLICATIONS AT CHALK RIVER

I will briefly review the present system at Chalk River which is mostly used to process data read directly from kicksorters (or pulse height analysers) connected to the tandem accelerator experiments. There are now two memory modules, and the standard input-output is by typewriter and paper tape. In the near future two more memory modules, and magnetic tape storage will be added. Time sharing has already been tried, where analogue to digital converters can break into the standard data handling program. This break is programmed to both transfer the ADC information and then reset it for the next event. This is a convenient arrangement as the standard program is often typing out or displaying on the 'scope, two functions that can be interrupted without penalty. To facilitate inserting these sequence breaks into a program using the standard input-output routines, the one-channel sequence break has been modified. The five status bits normally connected to this system have been removed, and replaced by the two coincidence outputs used in this particular experiment. This would not be necessary with the type 120 sequence break system, where any break channel could be activated

or deactivated under programmed control. Eventually all input-output operations should be in a sequence break system, to give full control of one piece of hardware over another. Practically all programs would have to be rewritten to be sequence break oriented.

These applications are reasonably conventional and will not be discussed further.

## FUTURE PLANS

Eventually two extra memory modules will be used in a somewhat unusual manner. These memories will be addressable by the computer in the normal way, but will also be available to other equipment completely independently of the central processor. It will even be possible for separate pieces of equipment to use different memories simultaneously. Extra hardware is needed to avoid referencing a memory register from more than one source at any one time. A switching network has been proposed to route the operations in separate memories independently. In this scheme it would rarely be necessary to use a sequence break when external equipment used a computer memory. The system would handle data transfers in the most efficient manner possible; this would be true time sharing, not time sequence interruption. Several of the free input-output commands may have to be used to start and stop these operations, but it has been found easy to wire in commands as required. This computer memory can provide a PHA facility having a versatility limited only by the programming effort involved.

## MORE IMMEDIATE PROBLEMS FOR THE CHALK RIVER SYSTEM

To use the sequence break system the computer must be in the "Run" state. Obviously any program with a halt command in it, or one requiring a "Start" cannot, in general, service any sequence break requests from outside equipment. More particularly if an operator were debugging a program it may be convenient to look at a memory register by hand, and perhaps to change it. It is not always possible to use debugging programs such as DDT; the memory requirements could clash, or the debugging program may not work in the mode the operator requires. To avoid these restrictions the computer time can be scheduled into two periods, a period for time sharing operations using known programs developed with the sequence break system in mind, and other periods to debug and use the computer outside the time sharing mode.

At Chalk River there is, however, a time sharing requirement that cannot be met as easily as this. A laboratory will require the computer to process a small amount of data



(say 100 words) at regular intervals of time (about 10 minutes) for a few days on end. Now a programmer requires frequent short periods on a computer to develop programs efficiently, progress drops sharply if this is not the case. So it is desirable to be able to debug programs and also run other programs not normally run during the time sharing phase. Theoretically it may be possible to rewrite every program ever to be used so that it is compatible with time sharing operations. These programs would always be in "Run" and the Start, Examine and Deposit functions would be handled automatically, perhaps as in DDT. But this requires major changes in most programs, many of which were not written at Chalk River. These would have to be understood in detail before attempting any changes. This is not a reasonable solution in the circumstances. If this break, occurring every 10 minutes, is to be serviced, then the following restrictions must be overcome.

- 1) The "break program," or the program entered to service the sequence break, must be in a part of memory that does not clash with the operator's program. This also applies to the "break registers" or the memory used by the automatic break operation.

- 2) If the computer is stopped for any reason, or the operator is using Start, Examine, Deposit, or Read In, the sequence Break Request signal must be held. At present, operating these four switches will clear any sequence break request so it is impossible to use the sequence break through these operations.

- 3) The normal input-output operations in the operator's program must be preserved. For paper tape, except for the Read-In Mode, operations are done one at a time so a break is innocuous. No breaks can be handled during the RIM operation. Display on the 'scope, use of the light pen and typing out on the typewriter can all be interrupted by breaks without trouble. But if a series of type-in's is interrupted by a long break, it is possible to lose some input if a type-in overwrites a previous type-in waiting to be serviced, but held up by the break program. A system of interrupt priorities could make the typewriter break on a higher priority, but this is not the way the typewriter is used at present. Magnetic tape operation is another operation that cannot be interrupted at random without harming the flow of information.

## DISCUSSION OF POSSIBLE SOLUTIONS

I now propose three specific modifications to enable these restrictions to be lifted, at least in part. Proposing these modifications does not necessarily mean they will be easy to engineer. The aim is to service the breaks required and still run any program we like. Some operating restrictions are introduced, this is inevitable. It is an advantage here to have only few programmers using the system, for an establishment where many departments develop their own programming, any operator restrictions at all might be

intolerable. Normal operation is not completely excluded, if the operator can wait until the on-line experiment finishes a run. This is not desirable as a general course to follow.

1) A change that will give considerable help to the programmer is to move the "sequence break memory" from module zero to a higher memory dictated by the size of the installation. This high memory could have both the "break registers" (4 for the standard system, 100 octal for the type 120 system) and all the break programs. This memory space need never be touched by other programs. As many programs require to work in memory zero (in the Chalk River Machine) and in general it is easy to get into the habit of loading programs into memory zero, normal operating procedure would be retained. Also program changes can be held to a minimum.

2) This modification would hold any Sequence Break Requests through the SC (or Start Clear) operation. SC is the operation given by "Start," "Examine," "Deposit," and "Read-In" and it at present clears any sequence break requests that are being held. With this modification, if the computer were not in "Run," or the sequence break mode was off and a sequence break request entered, it would be held but not granted. This unserved request would have to be indicated visually (or audibly) to the operator. The action taken by the operator on noting this request could be quite simple, a "Start in Sequence Break Mode" to practically anywhere in the memory. This operation is only reasonable if breaks can be delayed for several seconds before being granted, and these breaks do not occur very often to impede the operator's work. This is essentially the case in this problem.

The inevitable drawback to this scheme is that the break system must be zeroed by program at the beginning of the experiment. The extra programming required is negligible, and perhaps it is better to integrate this operation into a program than to forget it occurs automatically on "Start."

3) This change is of more general interest than in the time sharing context only. The typewriter keyboard should be locked after every type-in, and unlocked only when the type-in is serviced by the computer on the tyi command. Otherwise, with most time-sharing operations, the type-in operation must be given a very high priority break to service every input. Having the lock feature there may be no need to have the type-in on the break system at all. This would be the case if break programs were short compared with typing frequency. But the most valuable result of this modification lies outside the time-sharing context. During standard computer operation information can now be lost during type-in due to bad programs, misuse of programs or the

very occasional computer malfunctions. This should not be, other computer systems have a typewriter locking feature. The typewriter is very useful to control program branching, and all type-in's should be relevant, or at least digested by the program. If comments, useful only as a printed record, are needed, this can easily be programmed.

A possible scheme is to lock the keyboard when the tyi status bit is "on." This bit is cleared when the tyi command services this type-in. With no unserviced type-in's the type-out is not affected. Another useful modification is to clear the tyi status bit on a type-out. It is now possible to type-out and still leave the tyi status bit "on," But a type-out clears any previous type-in. So the tyi status bit remaining "on" is only of use if one wants to know, after a type-out, whether or not an unserviced type-in occurred before this type-out. Surely the time to check this is before the type-out, not after. This proposal would also ease the situation where someone left the computer with an unserviced type-in and the next operator is then faced with a locked keyboard. Generally a program requiring a type-in will give an initial type-out, which would thus unlock the keyboard. If this is not deemed adequate, a manual unlock facility could be provided.



SCANNING AND MEASURING OF NUCLEAR PARTICLE  
TRACK PHOTOGRAPHS WITH THE PDP-1\*

Professor Martin Deutsch  
Massachusetts Institute of Technology  
Cambridge, Massachusetts

ABSTRACT

The Laboratory of Nuclear Science is engaged in an extensive program of PDP-1 scanning and measuring of Nuclear Particle Track Photographs with the PDP-1. One project, SPASS, is designed for spark chamber photographs and has been producing data (over 500,000 pictures) since November, 1962. The other, PEPR, aims at the more difficult bubble chamber problem and is approaching the stage of operation. A brief discussion of the general methods in these and similar systems will be given and a more detailed discussion of the experience with SPASS will be presented.

\*This paper was not submitted in time for publication.



# A PHOTO-INTERPRETIVE PROGRAM FOR THE ANALYSIS OF SPARK-CHAMBER DATA\*

Harry Rudloe

Massachusetts Institute of Technology  
Cambridge, Massachusetts

## ABSTRACT

An operating computer program that processes photographically recorded data is described. The input to the program consists of spark-chamber photographs on which tracks of high-energy particles are recorded. The program automatically scans, measures and performs the preliminary interpretation of these photographs. In continuous operation a processing rate of 5,000 photographic frames per hour is achieved.

## INTRODUCTION

It is often natural and convenient to record data photographically. Before such data can be processed further, however, a data-takeoff stage must be introduced. This stage customarily involves visual scanning and measuring of the photographic data by trained personnel. If the body of data is at all large, this process may be extremely time-consuming, expensive and tedious, so much so, in certain cases, as to render the entire procedure impractical.

The present paper describes an operating computer program (PIP) which, for a case of particular interest, allows this process to be performed automatically at high speed.

PIP is designed to perform the analysis of photographed tracks which are generated by the passage of high-energy particles through spark chambers. In processing the 270,000 photographic frames obtained in a particular experiment, PIP achieved an average hourly rate of approximately 3000 frames per hour, including film changes and operator intervention. Working on a small sample of these photographs, a group of three human scanners processed 6000 frames in one week, i.e. about 50 frames per man-hour. During uninterrupted operation PIP achieves a rate of about 5000 frames per hour.

\*The work described in this paper was supported in full under AEC Contract AT(30-1)-2098 (Laboratory for Nuclear Science, Massachusetts Institute of Technology).

Subsequent to the presentation of this material before the May DECUS Meeting a paper appeared in the "Communications of the Association for Computing Machinery", Vol. 6, No. 6, June 1963 by H. Rudloe, M. Deutsch and T. Marill.

## EQUIPMENT

The equipment consists of a PDP-1 computer<sup>1</sup> augmented by a film-reading mechanism.<sup>2</sup> The computer's CRT Output Display<sup>3</sup> is used as an integral part of the film-reading system. This scope has the capability of momentarily displaying a point, under program control, at selected  $(x, y)$ -coordinates. Each of the two coordinates of the point is given as a ten-bit quantity, and arbitrarily located points may be displayed at a rate of 20,000 per second.

The film reader uses the computer scope as a flying-spot scanner. The process is as follows. A point, that is, a momentary flash of light, is displayed at scope coordinates  $(x, y)$ . By suitable optics, an image of this point is projected onto the film at corresponding film coordinates  $(x', y')$ . A photomultiplier tube located beyond the film responds to the momentary flash if and only if the film is transparent at coordinates  $(x', y')$ . The response of the photomultiplier sets a flip-flop ("program flag") in the computer; the state of this flip-flop may be sensed and altered by the program. By the nature of the equipment, only one bit of information (black or white; opaque or transparent) may be obtained per interrogation of the film. Hence, only high-contrast photographs may be read by the present technique.

Two separate film-readers are provided to permit the quick successive scanning of two correlated films and to allow the system to scan one film while the other is being advanced (see Fig. 1). Both readers operate off the same computer scope but set different flip-flops in the computer. To scan one film, the program interrogates the state of one of these flip-flops after having produced a point of light on the scope; to scan the other film the program behaves identically except for interrogating the state of the other flip-flop.

## PHYSICAL EXPERIMENT AND DATA

The data were obtained with the experimental arrangement shown in Figure 2. The devices marked "chamber" are spark chambers consisting of parallel plates to which a high voltage pulse is applied when the counters indicate that an event has occurred. When an ionizing

---

<sup>1</sup>Manufactured by Digital Equipment Corporation, Maynard, Mass.

<sup>2</sup>Developed at Laboratory for Nuclear Science, M.I.T.

<sup>3</sup>DEC Type 30.



particle, such as a proton, traverses a gap between the plates, a spark strikes at (or near) the point of passage. The sequence of such sparks forms a track. A gamma ray does not cause a track, since it does not ionize. It may, however, produce an ionizing electron in one of the plates. This electron produces further secondary electrons which form tracks spreading forward from the point of first interaction in a shower. In a favorable case the tracks in the shower have the appearance of a whisk broom with a short handle. The beginning of the handle lies on the line of the gamma ray.

Each chamber is photographed (through mirrors) in two mutually perpendicular views. Three frames are required to record each event. One frame contains the two views of the range chamber; a second frame, the two views of each of the two tracking chambers; a third, the two views of the shower chamber. The (binary-coded) running frame numbers are recorded with the tracking-chambers and with the shower chamber. Certain additional digital information is also recorded with the shower chamber. The tracking chambers and the range chamber are on one strip of film, the shower chamber on another.

The physical information is contained in the average coordinates of the tracks in the tracking chambers, the length (or range) of the track in the range chamber, and the coordinates of the starting point of the shower in the shower chamber. PIP is required to extract this information from the photographs, to read the binary-coded data and to perform a variety of checks to ascertain that the data pertain to a valid event. In the tracking chambers, a single point, the midpoint of the track, is needed in each view. This point is determined with an accuracy of  $\pm 0.1$  percent of full scale. The range is determined (in each view of the range chamber) by the number of the gap in which the last spark of the track occurs. In each view of the shower chamber, the coordinates of the initial spark of the shower are determined.

A sample of the raw data is shown in Figure 3. An explanatory guide to Figure 3 is given in Figure 4.

#### GENERAL APPROACH

A possible approach to the problem of reading film is to scan the entire photograph point for point and to form in memory a "core image" of the photographic image. With the equipment at hand, we can select  $2^{20}$  points, which provides the required accuracy; to scan all  $2^{20}$  points, however, takes 50 seconds. Since it is desirable to scan one event (three frames) in two seconds, a different approach evolved as follows:

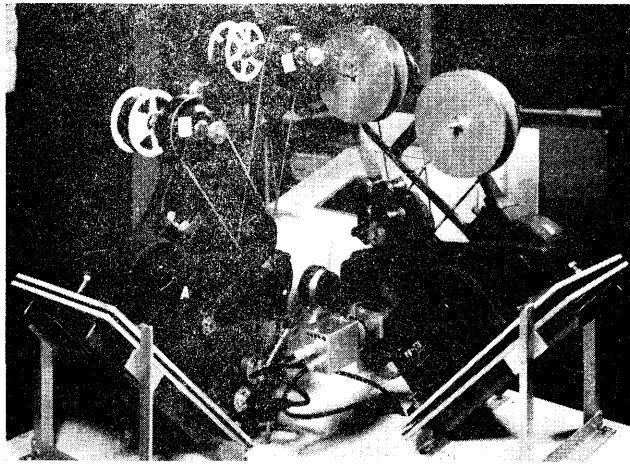


FIG. 1. Film-reading mechanism

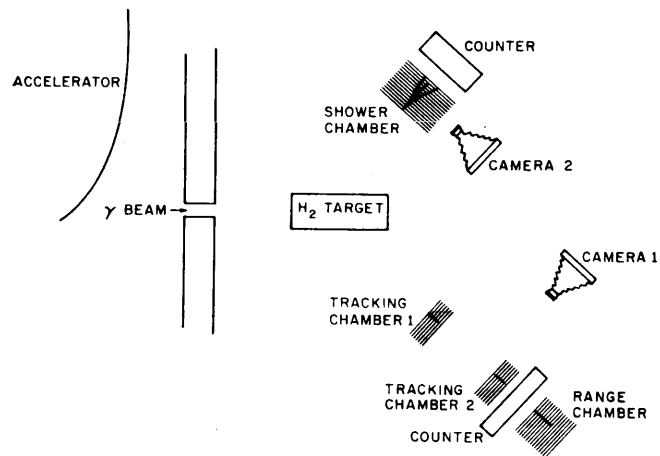
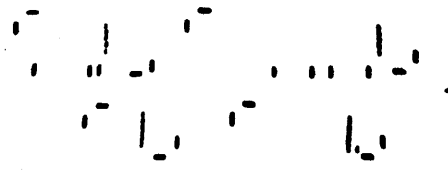
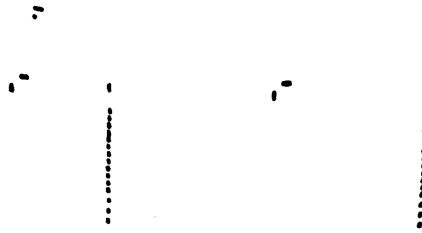


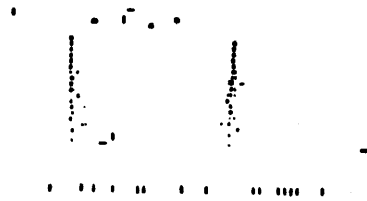
FIG. 2. Schematic of experimental arrangement



Tracking Chamber

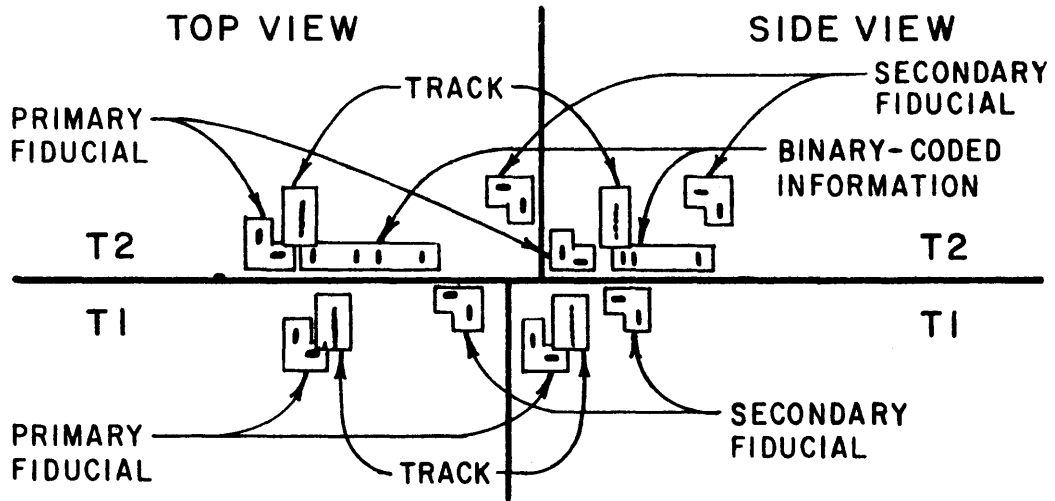


Range Chamber

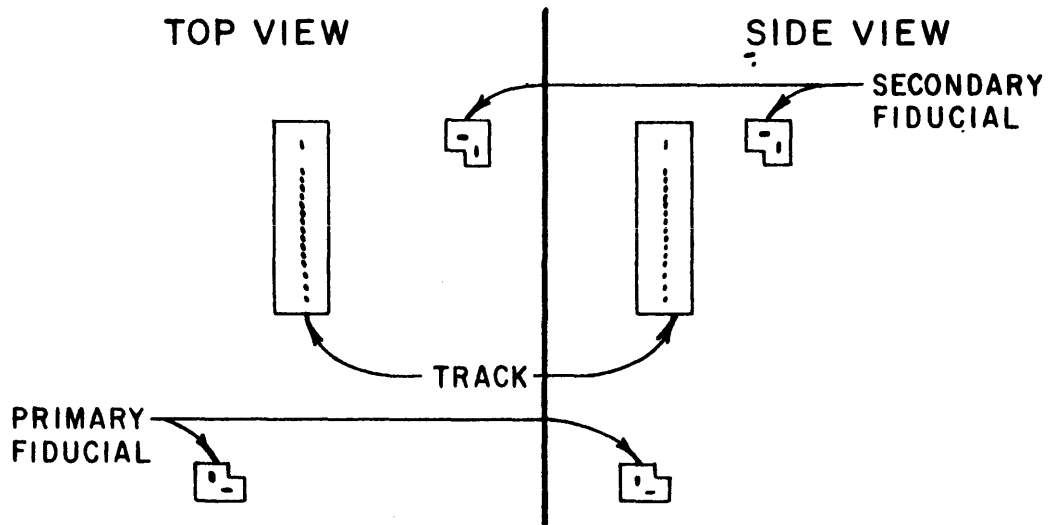


Shower Chamber

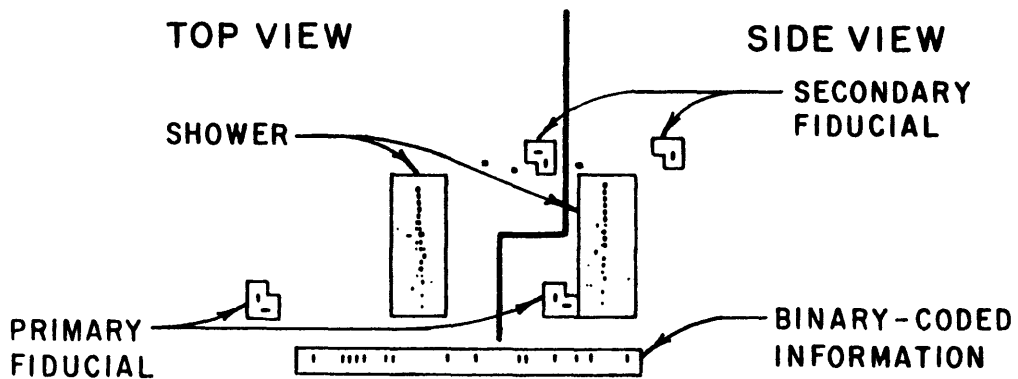
Fig. 3  
Sample raw data read by PIP



**TRACKING CHAMBERS**



**RANGE CHAMBER**



**SHOWER CHAMBER**

Fig. 4

- a) The format of the photograph is arranged so that the objects to be scanned occur in nearly fixed locations on the photograph. For example, the track sparks in a given chamber view will occur only in the spark chamber gaps for that view and nowhere else. Likewise the position of the number lights occur in a fixed position with respect to the frame. Thus the program will start by searching only those regions of the film where it expects to find something. This information is provided in calibration tables which are constructed from special calibration photographs prior to scanning.
- b) Any data obtained in scanning is reduced as soon as it is acquired. Thus if a search stroke encounters an object, the objects dimensions are measured immediately and its center stored if it is found to be acceptable. Likewise, tracks acquired by the tracking routine are immediately tested for acceptability and discarded if not of interest. Tracks accepted are represented by 2 or 3 numbers which express these essentially physical characteristics and stored in a table.
- c) Whenever possible, the reduced data is used to establish even tighter predictions of the location of further data on the photograph. In track-following, the program will use the position of previous sparks acquired to predict the location of the next spark in line. When the whole track is completed, it may be used to predict the position of its continuation in a connecting chamber.

## CALIBRATION

All calibrations for a given version of PIP are kept in a large table which is kept distant from PIP and generally loaded into another core. This is called the Event Description Table (E.D.T.) and contains all measurements, calibrations and constants relevant to a particular experiment. In its overall organization it is divided into sections for view, chamber, frame and experiment variables. Each of these is in turn divided into sets of data blocks relevant to a given view, chamber or frame. The most important of these are as follows:

- a) Fiducials

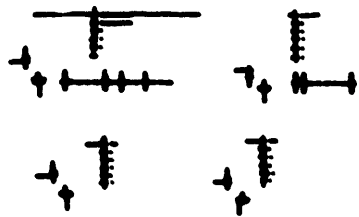
each view has a data block which describes the fiducial for that view. The description includes its estimated location and the length, number and orientation of the search strokes used to find it. The fiducials are used to define the coordinate system of the view. Thus the program must find the fiducials before it can find anything else.

- b) Gaps

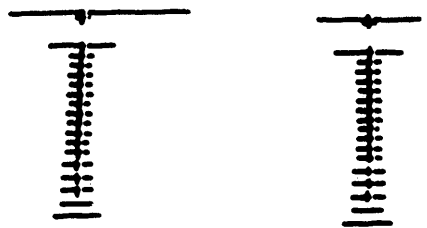
each view has a pair of data blocks which describe its gaps. One gives the principal Y-value for each gap. The other gives corrections which approximate the location of each gap by a set of straight horizontal lines. This is necessary because straight lines on the photograph do not necessarily map onto straight horizontal lines on the scope.

- c) Rulers

The rulers are a set of marker lights which appear on calibration photographs.



### TRACKING CHAMBERS



### RANGE CHAMBER



### SHOWER CHAMBER

Scan used by PIP in processing data of Fig. 3 - This picture was obtained by photographing a small repeater scope employed for debugging and calibration; the horizontal and vertical scales are arbitrary.

Fig. 5

The locations of these rulers are recorded and used to convert scope coordinate measurements into the coordinate system in which the experiment took place.

d) Binary Coded Information

These are considered frame variables and a data block is provided for each set of number lights on the frame (see Fig. 4). These give the locations of the number lights relative to a given fiducial and the stroke size used in searching for them.

## THE CALIBRATE PROGRAM

The sections of the EDT described above are loaded by means of a calibration program which also performs other hardware checking and diagnostic functions. It is controlled by a repeater scope (paralleling the computer output scope which is hidden in the film reader), various test word and sense switches, and an on-line typewriter.

On command from the typewriter the display scope scans a predetermined area of the film and displays the information. A dark point on the film appears as a bright point on the scope on redisplay. The density of the scan roster is controlled by sense switch options. The area scanned is indicated on the scope screen by corner brackets which can be moved by typewriter and their current position typed out on command.

A pointer in the form of a cross is displayed at the same time as the brackets and scan redisplay. The pointer can be moved on the screen by test word switches. A sense switch option permits accelerated motion. The pointer can also be set to a predetermined position by typing-in the desired coordinates. On command from the typewriter the current location of the pointer is entered in a pointer table, either in the lowest unoccupied location or overwriting a selected previously filled location. On command the entire pointer table or any selected entry is typed out. The positions of all pointers entered in the table are continuously displayed. Up to 47 (octal) pointers may be entered.

Upon typewriter command the program scans the film to the right of any selected pointer entered in the table or of all pointers in sequence. When a dark object is encountered before the edge of the scan area is reached, the center of the object is found by a repeated criss-cross scan and its coordinates are entered in an object table. When a single line scan is called for, the corresponding object coordinates are typed out immediately. When the entire table is called a separate command calls the type-out. When no object is encountered by a scan a - is typed. Repeated object scans and type-outs may be performed without changing the pointer table. After type-out of the object table, upon

typewriter command the program punches, in read-in format, the lists of pointer coordinates, object coordinates and bracket positions together with pointers to the beginning and end of the tables. This tape may be used to reset the tables at a later time. When a set of pointer coordinates is deemed satisfactory it can then be punched out in a format suitable to direct loading into the EDT.

#### THE STRUCTURE OF PIP - The Photo-Interpretive Program

The subroutines of PIP are divided into five major series A, B, C, D and E. The overall idea in establishing these subroutines was to provide a set of modules with well defined inputs and outputs which could serve in a variety of spark chamber experiments. Thus the contents of particular subroutines can be revised easily to suit the requirements of a particular experiment while the overall structure remains unchanged. In addition, wherever possible, the functions of acquiring a given type of data, and evaluating it are split into two separate subroutines. Thus one will see one routine for finding objects and another for measuring them, one routine for following tracks and another for reducing them, etc.

The five major series briefly are described below. Then a detailed outline of each series follows:

##### A Series

The first half of this series is composed of the basic object search and measurement. The second half has the basic data handling and information retrieval subroutines.

##### B Series

The first half is concerned with the overall supervising functions of PIP. The second half deals with acquiring track origins and the details of track following.

##### C Series

The C series is composed of all the major tables and temporary storage of PIP. The C series writeups give details of table formats and how to modify parameters.

##### D Series

The D series worries about finding and evaluating fiducials, rulers and number lights. It also contains the track evaluation routines.

##### E Series

These routines handle matters relating to interchamber predictions, stereo recognition and final data reduction and output.



## A Series

- A1. xsch xlsch (x search)  
Searches the film with horizontal line segments.
- A2. ysch ylsch (y search)  
Searches with vertical line segments.
- A3. smap (spark map)  
Determines center and dimensions of an object found by A1 or A2.
- A4. xedgehunt  
Called by smap to find horizontal edges.
- A5. yedgehunt  
Called by smap to find vertical edges.
- A6. dpsetup (display set up)  
Sets spot intensity, program flag connected to photomultiplier, timing of display loops and various flag sensing instructions in PIP A1 - A6.
- A7. fetchpt, insrtpt (fetch point, insert point)  
The basic data tables of PIP are composed of entries with a fixed number of words per entry which depends on the particular table. fetchpt will fetch a desired entry from the table specified in its calling sequence. insrtpt will replace a given entry with a new one.
- A8. storept (store point)  
Adds an entry at the end of a given table.
- A9. fetch, store, read, write  
A set of subroutines which manage communication between PIP and the event description table. read brings a whole data block into PIP, fetch brings a particular entry from such a data block. write returns a whole data block to the E.D.T. from PIP. store is the inverse of fetch.
- A10. clear, zero  
Subroutines to clear or zero a table of the sort described in PIP A-7.
- A11. This subroutine was deleted.
- A12. bit manipulation package  
Perform a variety of functions on a designated block of registers. These include inserting or extracting a given 3 bit segment from a block, inserting or testing single bits, clearing a block or counting the one's in it.

## B Series

- B1. super (super-visor)  
The supervisory routine for the whole system.  
Super cycles through the frames and controls film advances, output of data, error checks and starting and stopping of the program.
- B2. advfilm, cksfilm, err, bughlt  
advfilm states the film advance.  
cksfilm checks for completion of a film advance.  
err assembles an error code, when the error is caused by some malfunction of the film or equipment.  
bughlt a halt location to which many programs escape when one makes impossible demands of them.
- B3. frame  
Processes a single frame with reading of the number lights and scanning of the individual views. Contains sense switch options for single step operation of the program.
- B4. view  
processes a single view with reading of the fiducials and ruler marks. Just before quitting it calls the predict subroutine which predicts into other view on the basis of the view or views just scanned.
- B5. scnrcv, scntcv (scan\_range\_chamber\_view, scan\_tracking\_chamber\_view)  
Subroutines to scan range and tracking chambers views.
- B6. scnscv (scan\_show\_chamber\_view)  
A subroutine to scan shower chambers views.
- B7. scnpzt (scan\_prediction\_zone\_table)  
A subroutine to scan the area specified by the prediction zone table. If the predict routine has ascertained that tracks are likely to appear in certain places in the current view, than a prediction zone table will have been constructed for the current view. scnpzt takes each entry in such a table and follows all tracks that occur in the region it specified.
- B8. scncpz (scan\_complement\_of\_prediction\_zones)  
If scnpzt didn't find any good tracks, one may wish to call this routine which looks every place that scnpzt missed.

- B9. gapsw, gapsw1, gapsw2, gapsw3 (gap sweep, etc.)  
gapsw examines a specified segment of a spark chamber gap region and fills tortable (track origin table) with objects likely to be track origins.  
gapsw1 is similar to gapsw but processes an entire gap.  
gapsw2 calls gapsw1 and then follows out all tracks originating in the examined gap.  
gapsw3 is similar to gapsw2, but calls gapsw to establish track origins.
- B10. trker1 (tracker 1)  
A routine which calls trker (B11) to follow out a track. If the track is long enough it enters it in the current track summary table. It tries to ignore tracks which have already been followed by checking the gaps immediately above the tracks origin.
- B11. trker (tracker)  
The basic track following routine. Given a track origin it tries to follow the track. Various parameters tell it which way to go and when to quit.
- B12. cone  
A subroutine which trker uses to construct prediction cones from one gap to the next in the operation of track following. cone calls the sweep routine to search the base of each cone it constructs and then enters in trktable (the track element table) whatever spark is closest to the predicted center of the cone base.
- B13. line  
When cone has just finished entering a spark in trktable it calls line to construct a straight line between the first and last sparks in trktable. This straight line determines the prediction for the next time that cone is called. In cone, the call to line may be replaced by a call to line1 (D8) if a true least squares approximation is desired.
- B14. sweep  
sweep will search a specified region of a specified gap and enter its findings in cstable (center of spark table).

## C Series

- C1. edt (event description table)  
This is an enormous table which sits in core 1 and contains all parameters, options and calibrations for the experiment. The major categories included in the E.D.T. are view, chamber, frame and event variables. Under the category of view variables there will be a number of data blocks corresponding to each spark chamber view. These may either apply to the system as a whole, or some specialized function, such as containing locations of gap zones or ruler marks. The same applies to chamber and frame variables. The E.D.T. has a tree structure wherein a given data block is located by hunting through a chain of pointer blocks. Each pointer of a pointer block either points to the head of a data block or to another pointer block. Data is inserted or retrieved from the E.D.T. by presenting the routines of PIP A-9 with a string of digits that uniquely locate a desired data block. The virtue of this scheme is that one may readily rearrange or modify the E.D.T. without a minimum of effect upon PIP proper.
- C2. Data block addresses  
The locations of all the data blocks in the E.D.T. are provided on this tape. Each data block address is a string of instructions which load the AC with the integer. The string is terminated by a zero. To request a given data block, the address of the first register in it's data block address is given, by the calling sequence to the desired routine of PIP-A9.
- C3. parameters  
This tape defines the location of the current active view, chamber and frame parameters. Its contents are renewed at the start of each new view, chamber or frame by reading in the appropriate data blocks from the E.D.T. It is also used to set up portions of the E.D.T. by locating the parameter locations with DDT and then writing them in the desired section of the E.D.T. using the write routine (PIP -A9).
- C4. system registers  
Defines the location of most of the important system registers, i.e., registers that handle communications between the major subroutines.
- C5. Internal tables  
The basic data handling tables for PIP-2. They are as follows:  
cstable      spark centers and sizes as found by sweep.

<u>tortable</u>	potential track origins, set by <u>gapsw</u> and <u>gapswl</u> .
<u>trktable</u>	track elements (of a single track) set by <u>trker</u> .
<u>rultab</u>	ruler marks of current view.
<u>bcitab</u>	each entry is the number light readings on a single frame.
<u>tst<sub>0</sub>-tst<sub>i</sub></u>	one track summary table per view. Each entry contains track angle, first and last gap, ruler coordinate and stereo recognition pattern words and tags. <i>i</i> = the number of views in the experiment.
<u>pzt<sub>0</sub>-pzt<sub>i</sub></u>	one prediction zone table per view. Each entry contains the predicted location of a track in a view, including search radius, and inclination.

### D Series

The D Series is in charge of fiducials, rulers, number lights and track evaluation.

- D1. findfd (find fiducial)  
The supervisory routine for fiducial findings. Its functions are to initialize the fiducial search parameters, call the fiducial search routine, perform alignment checks if desired and decide what to do about missing fiducials.
- D2. ffd  
Carries out the gubby details of actually finding a fiducial.
- D3. rnli (read number lights)  
Reads a single set of number lights.
- D4. rdhci (read binary coded information)  
Reads and evaluates and stores all binary coded information on a given frame.
- D5. rrlm (read ruler marks)  
Transfers the estimated ruler locations in the E.D.T. to rultab, with the option of reading the ruler marks directly from the film if necessary.
- D6. convrt and dcnvrt (convert, deconvert)  
convrt is presented with a track origin and inclination and projects the track onto the ruler. Its output is the ruler coordinate of the track. dcnvrt is given a track's ruler coordinate and inclination and will provide the track's scope coordinates for any desired gap.

- D7. evaltt, trkout (evaluate track table, track output)  
evaltt extracts the vital information from a track and subjects it to a number of checks for void conditions. It then constructs a track summary which trkout releases to the track summary table.
- D8. linfit line1  
linfit is called by D7 to construct a line which is a least squares fit approximation to the track in trktable. A call to line1 can replace a call to line in PIP-B12 if a slower but more accurate prediction is required in track following.

### E Series

- E1. predict  
The supervisory routine for all interchamber prediction functions, i.e., how to cope with a future chamber or view on the basis of what occurred in past ones.
- E2. prediction arithmetic  
Subroutines which handle the computational details of predicting where a given track will enter another chamber.
- E3. makptw (make pattern word)  
A routine which will take tactable and construct a pattern word for stereo recognition purposes, based on spark size information included in the table.
- E4. datout (data output)  
A routine which takes the track summary tables reduces them and releases them to the world through some in-out device, usually magnetic or paper tape.

### ACKNOWLEDGMENTS

The author is indebted to William E. Fletcher, who helped in the preliminary planning and contributed two important fundamental routines; to R. A. Bolt, who contributed several routines; to D. M. R. Park, who provided much-needed assistance during the check-out phase; and to Professor M. Deutsch of M.I.T. who gave an introduction to this paper at the DECUS Conference.

## THE DIGIGRAPHIC DISPLAY PROGRAM FOR THE DX-1 COMPUTER SYSTEM

John T. Gilmore, Jr.

Charles W. Adams Associates, Inc.  
Bedford, Massachusetts

### ABSTRACT

The Digigraphic display program for the DX-1 computer system utilizes a buffered display scope, light pen and push-button panel to provide the console user with the basic ability to draw charts, diagrams, curves, etc. on the face of the display scope. The drawings may contain graphical and alphanumeric information, which is reduced to a condensed digital format called an Entity Table. The Table can be operated on by special-purpose software operators either during the drawing action or after the drawing is completed.

Certain man-machine programming techniques have been designed to accelerate the manual task of drawing (or drafting). The console user may draw points, lines, circles, arcs, and freehand or third-degree curves. Distances and angles may but need not be specified. Sub drawings or their copies may be moved about, rotated or reflected; angles and distances may be queried, and drawn dimensions can be generated on the drawing itself.

### PROGRAM AND SYSTEM

#### Function of Program

The Digigraphic Display Program (DDP) was designed to enable a console user to introduce to a computer graphic and alphanumeric information which can be displayed without flicker and stored in memory in condensed digital form. The program itself facilitates the preparation and digitation of all types of charts and drawings. However, the digital description of the graphical data can be used by other software routines either during or subsequent to the actual drawing process. DDP is an improved version of a program written by Charles W. Adams Associates as part of a joint project with the Itek Corporation.

#### System Components

The DX-1 system consists of a standard 4K PDP-1 computer, a display scope and four magnetic-tape units with type 52 control (all developed and manufactured by Digital Equipment Corporation); a Bryant magnetic drum; a flicker-free display processor and a fiber optics cable light

pen (both developed by Itek Corporation and now being produced by Control Data Corporation); and a push-button control panel with two foot pedals (suggested by Adams Associates and Itek Corporation and produced by the DX-1 engineering staff).

The computer has an information exchange buffer which permits communication with another 4K PDP-1, the latter having a color display scope and light pen. An additional 4K core bank as well as the tape units and controls can be attached to either computer. While only the first computer can presently use the magnetic drum and flicker-free display processor, equipment now on order will provide the second computer with access to the drum and a flicker-free display processor for the color display scope. DDP currently restricts itself to the first computer (see Figure 1 on page 109) but the program will be modified for use by the second computer and its color scope.

As shown in the figure, the 15 spring-loaded buttons on the control panel, the push button mounted on the light pen, and the two foot pedals are connected in parallel to the 18-toggle-switch test word of the first computer.

The flicker-free display processor was described in a paper by Earle W. Pughe, Jr. at the 1962 DECUS meeting. However, to those not present at that meeting the following abstract of Mr. Pughe's paper may be of interest:

The Itek Flicker Free Display displays line drawings on a 10" x 10" scope 30 times a second with a maximum total line length of 600 inches. The Display is controlled from a Telex disc through logic which controls the beam. There are about 20 instructions used to control the display. When the display is to be changed, new instructions are put on the disc by a block transfer from the PDP, otherwise the computer is not needed for the display.

It should be noted that the Telex disc is a Bryant drum in the DX-1 system, and that the four-bit byte instructions referred to by Mr. Pughe are now six-bit bytes stored on the 20,000-byte display track. The change from four-to six-bit byte logic and the longer drum display track have increased the drawing capacity to approximately 2,000 linear inches. However, to accommodate extremely crowded drawings, a new six-bit byte instruction has been added to permit the use of two display buffer tracks rather than one whenever necessary. While the reduction of the repetitive cycle in this case to 15 times a second produces a flickering effect, it was done to avoid losing any display information.



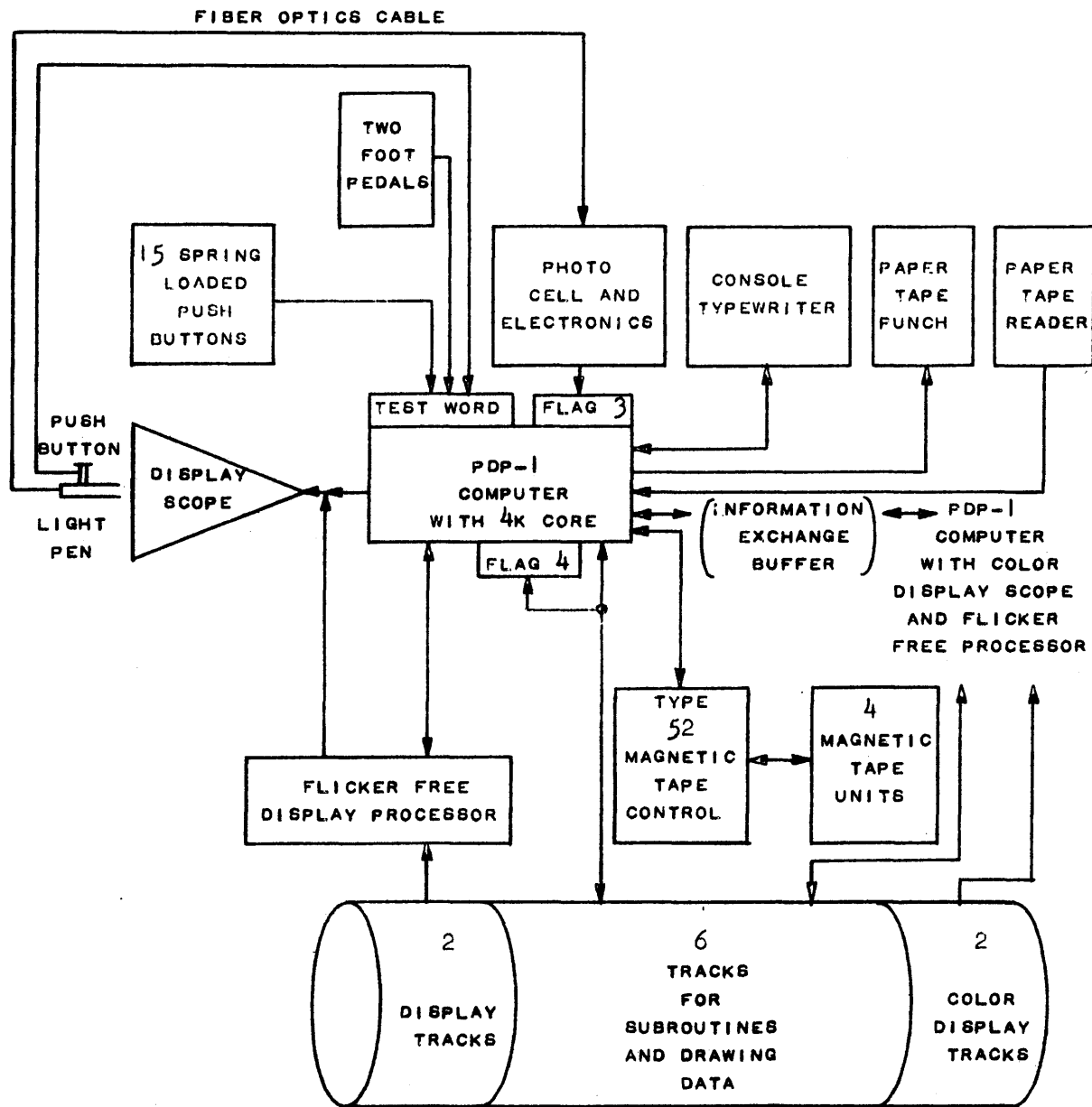


Figure 1

DX-1 EQUIPMENT USED BY  
THE DIGIGRAPHIC DISPLAY PROGRAM

The drum consists of ten tracks, each containing twenty sectors with 1,000 bytes in every sector. (A byte is composed of six bits of data and a drum parity bit.) While reading or writing is begun at the start of a sector, the length of the block of information transmitted is variable.

The display scope has a useable area of 10" x 10", the bottom 2" of which are used to provide numerical display registers and a series of displayed points utilized as control buttons (called light buttons). A plastic template is used to label the points and registers. The original preference for light buttons over an additional set of push buttons was influenced by the versatility of the former in changing the number and configuration of control buttons. The light buttons will be replaced by hardware, however, as soon as DX-1 users have had sufficient experience to finalize the design.

### Geometric Digital Description of Drawing

Drawings are made of different figures combined to effect the desired representation. DDP can produce standard figures and manipulate them in various ways. To do so, each item of the drawing, referred to as an entity, is stored internally in digital form and oriented on a  $2^{18}$  grid, the smallest increment of which represents  $2^{-5} \times 10^{-2}$  or .0003125 paper inches, and the maximum spread of which is  $2^{13} \times 10^{-2}$  or 81.92 paper inches. Since actions are performed in terms of entities, the functions will be described in these terms.

The types of entities that can be produced are:

Point	Used for reference, centers of circles, etc.
Line	Used as part of the drawing, guide lines, etc. (Note: DDP distinguishes between horizontal, vertical and slanting lines, but the user need not concern himself with this capability.)
Circle	Used in various ways.
Curve	Third-degree curve or curves defined by a series of points. (Note: a freehand polystring of lines can be replaced by a series of third-degree curves to produce a smoother representation.)
Arc	Parts of circles

Polygon	Regular polygon of three to fifteen sides. (Note: A polygon entity need not close in the DX-1 system. In fact, this entity is used in freehand drawing to represent a series of points connected by straight lines, commonly called a freehand polystring of lines.)
Dimensions	Lines, arrow heads and numbers used to show dimensions.
Remarks	Text attached to a drawing as a note or explanation, composed of letters, numbers, punctuation and special characters.
Grid	A matrix of points whose location, density, number, individual values and maximum time are specified.

As a drawing is prepared, each entity is assigned an identifying subdrawing number, called a group number. This number can apply to one entity or to a number of entities which are to be treated in a similar manner. While each entity may have only one group number assigned to it, the entity may be reassigned if required or desired. The numbers 0 to 63 may be used for group numbers.

## MAN-MACHINE CONTROL

### Push Button Keyboard

There are basically two modes of light-pen operation: using the pen as a pointer, and using it as a pencil. If the user wants to point at an existing figure on the display scope, there is no need for tracking the pen. Rather, the light-pen flip-flop is monitored and, when a light response occurs, the position of the drum display track is queried. DDP is capable of determining from the track position which graphic entity has been touched by the pen.

If the user wishes to point at a position on the scope where there is no data, he must guide DDP to the position by using a light-pen tracking cross (developed, as far as the author knows, by Roland Silver). The tracking cross is also used for continuous pointing, i.e., using the pen as a writing pencil, and also for showing the movement of a subdrawing.

One of the two foot pedals is used to indicate whether the pen is to be used as a passive pointer or tracked for writing or moving. The two routines that control these actions are called Pick and Sketch, respectively.

The spring-loaded push buttons are used to further describe the light-pen action. Since there are only 15 buttons, an upper- and a lower-case function assignment was adopted. The Sketch functions are labeled on the top of the buttons and the Pick functions on the bottom. Each button has a red and a blue light behind it which indicate the function in use. A diagram of the push-button keyboard is shown in Figure 2 below.

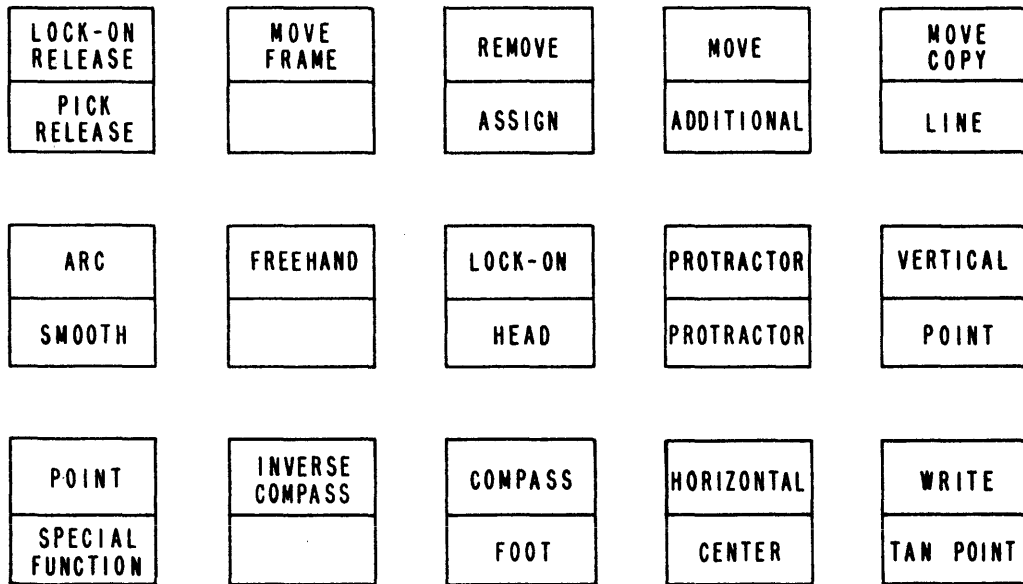


Figure 2

### PUSH BUTTON KEYBOARD

#### Pick Push Buttons

The following pointing capabilities are currently provided by the Pick routine:

- Pick a point
- Pick an additional point
- Pick a center point
- Pick an additional center point
- Pick a tan point
- Pick an additional tan point
- Pick a line
- Pick an additional line
- Pick a center line
- Pick an additional center line
- Pick the head point of a protractor straight edge angle (This can be the point on any graphic entity.)

Pick the foot point of a protractor straight edge angle (When the head and foot have been selected, the protractor straight edge angle will be set according to the angle

from the foot point to the head point.)

Pick the protractor straight edge angle of a line (This angle will be set according to the angle of the line and its direction when originally drawn.)

Pick an entity and assign to it the logical group number displayed in the Group Number Register.

The PICK RELEASE push button is used to cancel or change a picking operation. The SMOOTH and SPECIAL FUNCTION push buttons are not associated with Pick.

Before proceeding, it should be made clear that the push button on the light pen sets a bit in the test word of the computer which, in effect, provides a logical shutter to the pen. The opening and closing of the shutter is achieved by depressing and releasing the light-pen push button.

In all the Pick functions listed above, the Pick program initializes a picking operation if any push button or combination of buttons is depressed at the moment the pen shutter is opened. The monitoring of the light pen and the entity being touched continues as long as the button is depressed. To assure the user that he is pointing at the desired entity, Pick intensifies the entity by displaying it directly from the computer. If a point is being picked, the entity being pointed at will be intensified and Pick will choose the point on that entity which is closest to the center of the aperture of the pen. (The one exception is that if the point is within a fixed position from the end point of a line, the end point will be chosen instead.) If the pen shutter is closed, Pick will intensify only the current point. Reopening the shutter allows the user to choose a different point.

Once a selection has been made of the desired point or line (the latter in this case being a locus of points and either straight or curved), the push button (or buttons) is released and that point or line will be remembered by the program. In the case of HEAD, FOOT, PROTRACTOR, and ASSIGN, the actions indicated will be carried out.

To remind the user that a given point or line is currently being remembered by the program, a temporary symbol is displayed next to the point or line. Each kind of Pick action has its own symbol. These remembered points and lines are used by a set of Construct routines which are described later.

## Sketch Push Buttons

Except for two situations, lock-on and remove, Sketch assumes that the light pen is to be tracked if its shutter is open. Whenever the shutter is opened, Sketch will display the last center point of the aperture of the pen. If the pen is no longer over the point, Sketch will monitor the shutter and continue to display the point brightly until the user positions the pen over the spot or closes the shutter.

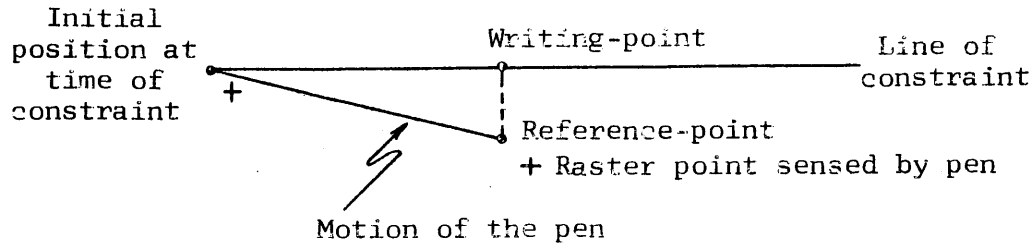
Once the pen has been positioned over the previous center point, a Roland Silver tracking cross is used to continually determine the new center position of the aperture. A pen position prediction routine was not used because, there being only one buffered scope, there is presently no need to use the computer for anything except tracking. The tracking cross, moreover, is displayed directly from the computer and cuts out the buffered display data; consequently a random interval between tracking crosses is required to allow all of the buffered display to appear on the scope (at the reduced rate of 25 times per second). Pen tracking currently uses about 15 percent of the computer time, the remainder being idle.

All Sketch tracking operations use a writing-point and a reference-point which are initially superimposed and displaced 1/4" above and to the left of the center of the tracking cross. The coordinates of the writing-point are those used to determine the end points and parameters of an entity. The reference-point is used as a guide under certain circumstances.

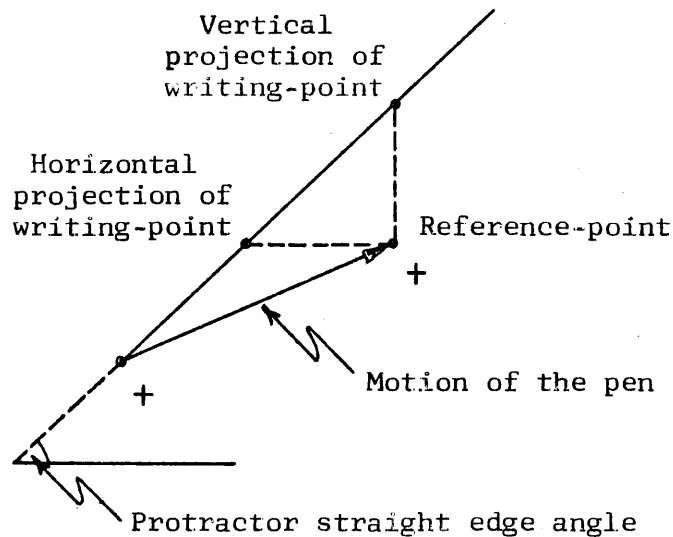
Since it is desirable to simulate a straight edge, there are three push buttons that provide constrained tracking horizontally, vertically, and according to a protractor straight edge setting. This means that the writing-point will be constrained whenever one of these three constraint buttons is depressed. Thus the reference-point and the writing-point separate from each other, the reference-point remaining in the same relative position from the tracking cross while the writing-point is constrained to some angle of motion.

During horizontal tracking, as shown in the diagram below, the writing-point will continue to have the same y coordinate that it had at the time the HORIZONTAL push button was depressed. Its x coordinate will be the same as the x coordinate of the reference-point. The reverse is the case in vertical tracking.

In protractor-constrained tracking, an illustration of which appears at the top of page 11, the writing-point will be constrained to move along a path whose slope is the same angle



as the protractor straight edge angle and which passes through the original coordinates of the writing-point at the time the PROTRACTOR push button was depressed. The writing-point may be positioned on the protractor-constrained path either by a horizontal or vertical projection of the reference-point onto the path line. Since both situations are desired, the HORIZONTAL and VERTICAL push buttons have an auxiliary function of allowing the user to switch from one to the other projection while keeping the PROTRACTOR push button depressed.



The reference-point in the two illustrations above greatly reduces the need for guide lines. Furthermore, the ability to interrupt tracking long enough to use the pen as a pointer for the reference-point provides an effective drawing ability. This is done by depressing the LOCK-ON push button, which causes the Pick routine to be activated in choosing an existing point on the drawing.

As soon as the LOCK-ON push button is released, Sketch is reactivated, the reference-point is given the new coordinates of the picked point, and the writing-point is repositioned according, first, to the constraints (if any) imposed on it and, second, the coordinates of the reference point.

One final note on constrained tracking: when a constraint push button is released, the reference-point is repositioned to the coordinates of the writing-point, and the tracking cross is repositioned according to the new position of the reference-point. At times this snaps the tracking cross out from under the pen and forces the user to reposition his pen. This is done so that he will begin tracking again from the last mathematical point of the previous constraint.

Briefly, the functions of the remaining push buttons for Sketch are:

WRITE	To draw all straight lines
POINT	To draw points
COMPASS	To draw circles (The initial point is the center and the last point on the periphery.)
INVERSE COMPASS	To draw circles (The initial point is on the periphery and the last point on the center.)
ARC	To draw arcs (Two straight lines originating at the center are used, like the hands of a clock, to specify an arc.)
FREEHAND	To draw freehand curves (A string of connected points, the density of which is proportional to the tracking speed.) Note: Immediately after the curve has been drawn, it can be smoothed by releasing the Sketch foot pedal and then depressing the SMOOTH push button. The curve will then be replaced by a series of third-degree curves, thus producing a much smoother curve. The amount of smoothing required will depend upon the density of points defining the curve.
MOVE	To move entities having a common
MOVE COPY	group number or a copy of them
REMOVE	To remove an entity, the pen being used as a pointer
MOVE FRAME	To select a new view of the drawing in the frame, which can be moved about regardless of size (When this push button is released, the new view in the frame is recalculated and displayed.)



## SKETCH DIMENSIONS

To permit the user, through the use of the second foot pedal, to see the length and angle of the lines being generated in the Sketch mode (If desirable to restrict the length of unhooked lines to a specific tolerance, say, 1/8", the tolerance is keyed in and the user will not draw an unhooked line whose fraction is other than a multiple of an eighth.)

### Light Button Display Panel

If all push buttons and foot pedals are in the release position and the light-pen shutter is open, the light-button display panel is turned on and displayed directly from the computer. The basic philosophy in light-pen operation in this case is that the last light-button point seen by the pen prior to the closing of the pen's shutter will be the button to be activated.

The functions of the registers and light buttons illustrated in Figure 3 on page 118 are described briefly below.

### Left Half of Panel

MESSAGE Register

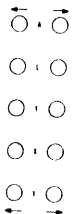
For passing comments to the user; for example, the word MORE is displayed if parameter data is insufficient.

PROTRACTOR STRAIGHT  
EDGE INDICATOR

Displays a small vector line in the circular cut-out of the template to indicate the angle of the protractor straight edge.

ANGLE Register  
LENGTH Register  
TOLERANCE Register  
X-COMPONENT Register  
Y-COMPONENT Register

These are display registers used to show numerical values. The light buttons to their left are used to prevent the contents of these registers from being cleared after an operation.



These five light buttons, to the right of the numerical display registers, are used to transfer the contents of the registers to the selected accumulator, or the contents of the latter to the display registers.

AC-1 Register

These two light buttons, to the

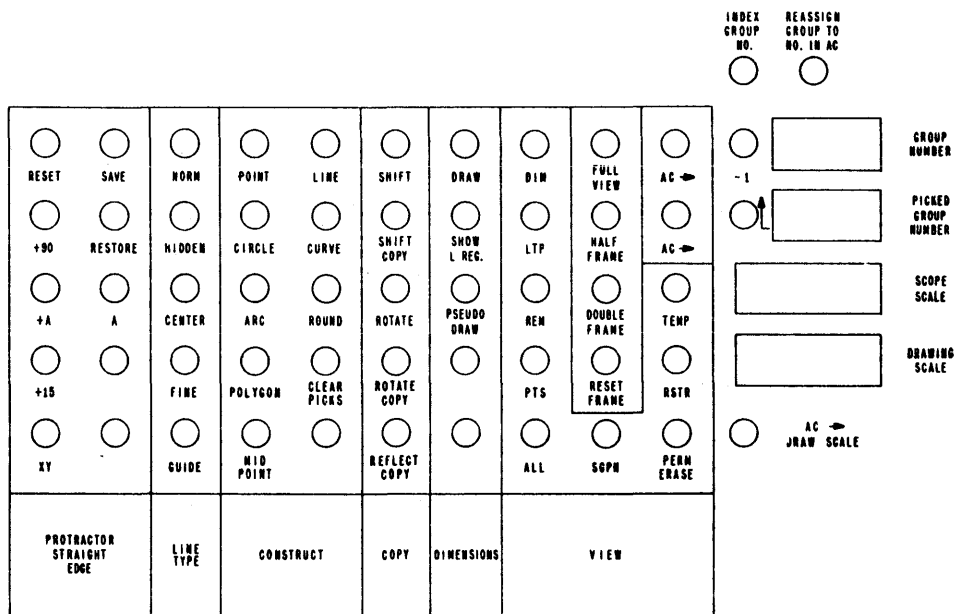
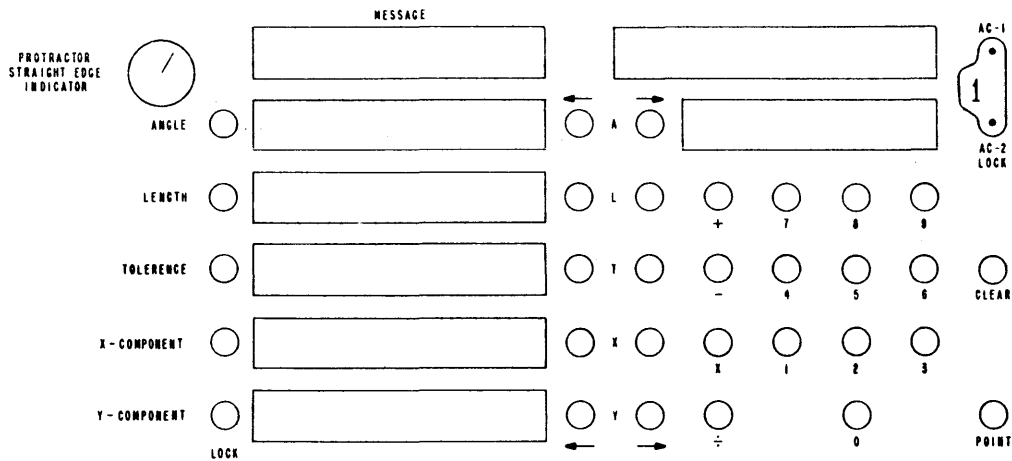


Figure 3  
LIGHT BUTTON DISPLAY PANEL

AC-2 Register

right of the two accumulators, are used to prevent the contents of the accumulators from being cleared after an operation. Between the buttons a small numeral 1 or 2 is displayed to show which accumulator is active. If one numeral is touched by the pen, the other accumulator is activated and its numeral will appear.

0 through 9

These ten light buttons are used as a regular numerical keyboard, the values being keyed into whichever accumulator is active.

+  
-  
x  
÷

These four light buttons are used to add, subtract, multiply and divide between the two accumulators.

CLEAR

This light button clears the active accumulator.

POINT

This light button is used to indicate that any additional numerals keyed in will be placed in the fraction part of the active accumulator.

Right half of panel

PROTRACTOR STRAIGHT EDGE:

RESET

Resets the protractor straight edge angle to 0.

SAVE

Saves the current setting of the protractor straight edge angle.

RESTORE

Restores the previous setting of the protractor straight edge angle.

+90

Adds 90° to the protractor straight edge angle.






+A

Adds the angle in the Angle Register to the protractor straight edge angle.

+15	Adds 15° to the protractor straight edge angle
XY	Sets the protractor straight edge angle according to the values in the X- and Y-Component registers.
A	Reads the protractor straight edge angle into the Angle Register.

#### LINE TYPE:

All entities (except Remarks and Dimensions) may be sketched or constructed with any of the following lines:

Normal	
Fine	
Hidden	
Center	
Guide	

#### CONSTRUCT:

The general procedure in the Construct mode is to specify information required for the construction of an entity. This is done by introducing values into the numerical display registers and/or by using the pen as a pointer to indicate where the construction is to occur; and then by touching the appropriate Construct light button. If the data furnished is insufficient, the word MORE will (as previously stated) appear in the Message Register; if too much data has been specified, only that necessary will be used. Certain constructions produce ambiguous situations; for example, where three lines define a circle, one of the two possible figures is displayed. Touching the Construct CIRCLE light button a second time will produce the other figure

The following constructions are currently available in the DX-1 system:

POINT	The Digigraphic Display Program presently permits the expression of a point and a line in six ways, a circle in seven, an arc in two, and a
LINE	
CIRCLE	
ARC	
POLYGON	

polygon in three. However, since there are numerous ways of constructing geometric figures, it would be only a matter of developing specific subroutines to expand the program's construction capabilities.

#### ROUND

This light button is used to replace the intersection of two lines by an arc joining them. The arc will be drawn between the two lines in the quadrant specified by a point. The point and the ends of the line are removed.

#### CURVE

This light button is used to construct a continuous curve composed of a series of third-degree curves defined by two points. The curve is drawn by picking a series of points, tangent points, or a combination of both, which define the various segments of the curve.

#### CLEAR PICKS

This light button clears all picks (since entities, once picked, cannot be cleared individually) as well as the AC-1, AC-2, A, L, T, X and Y registers unless they have been locked. This button must also be used following ambiguous constructions because in such cases the picks are not automatically cleared.

#### MID POINT

This light button is used to construct a point in the middle of a picked line or midway between two picked points.

#### COPY:

The five light buttons described below are available for accurate manipulation of groups of entities. They provide for moving, rotating and reflecting entities and are similar to the corresponding functions in the Sketch mode.

SHIFT	Same as MOVE except angle and distance are specified by horizontal and vertical components or by two picked points.
SHIFT COPY	Same as MOVE COPY except as stipulated above.
ROTATE	Rotates a group, specified by the Picked Group Number Register, about a picked center point. The rotation is counter-clockwise and specified by the value in the A Register. The picks are not cleared and the function can be used repetitively.
ROTATE COPY	Operates in the same manner as ROTATE, but leaves the original group undisturbed. This function is used when drawing gears or other figures which are repetitive in a rotational sense. The copies are given the group number appearing in the Group Number Register. When the numbers in the Group Number and Picked Group Number registers are the same, the angle of rotation is doubled after each rotate operation. If the two registers differ, the value of the angle is accumulated after each rotation.
REFLECT COPY	Creates a mirror image of the groups specified by the Picked Group Number Register. The copy is generated by reflecting the selected group about a picked center line. The copy is given the group number specified by the Group Number Register and the picks are cleared.

#### DIMENSIONS:

The three light buttons described below are used for showing dimensions on drawings. Actual or arbitrary dimensions may be drawn or dimension value may be shown in the Length Register.

DRAW	Used to draw a line dimension alongside the picked line or between two picked points.
SHOW L REG	Used to display, in the Length Register, the numerical value of the picked line or the distance between two picked points.
PSEUDO DRAW	Used to draw an arbitrary dimension alongside the picked line or between two picked points. The arbitrary length is entered in the Length Register prior to this operation.

#### VIEW:

DDP provides facilities for erasing, permanently or temporarily, specified portions of a drawing. When entities are temporarily erased they may be restored later; but when permanently erased they are not recoverable. To effect any of these operations, entities are specified by certain characteristics. In addition, it is possible to indicate that either all entities or only those within a specified group number may be affected. At least two specifications must be made: classification and type of action, the light buttons involved in each being:

#### Classification:

DIM	Only dimension entities will be affected.
LTP	Only entities with the line type currently selected will be affected.
REM	Only remark entities will be affected.
PTS	Only point entities will be affected.
ALL	All entities will be affected. This light button can be used alone or in conjunction with LTP; for example, all entities with guide lines.

Group Number:

SGPN

This light button provides further classification of entities in that only those belonging to the group whose number appears in the Picked Group Number Register are affected. Otherwise, entities in all groups will be affected by the action.

Action:

TEMP

The entities specified will be temporarily removed from the display but retained within the system.

RSTR

The entities of the type specified, which have been temporarily erased, will be restored to the display register. All other entities will be unaffected.

PERM ERASE

The entities specified will be permanently erased from the system. Extreme caution should be used in regard to this light button.

### Scale

While some types of drawings (such as mechanical or architectural) have a direct dimensional relationship to the items they represent, many others (such as block and schematic diagrams) do not. To facilitate the preparation of dimensional drawings, DDP permits the use of a drawing scale which, as already mentioned, represents the ratio of the drawing to reality. The light button and register involved in scaling are described below:

AC→DRAW SCALE

To set the drawing scale, the integer specifying the object/drawing is entered into either AC Register and the AC→DRAW SCALE light button is touched. The corresponding scale will be transferred to the Drawing Scale Register. Normally the drawing scale would be set before a

DRAWING SCALE  
Register



drawing is begun and not changed. However, the scale may be changed at any time and the new value will affect the drawing only from that point on.

The scope scale is the ratio of the scope dimensions of the displayed information to the paper dimensions of the drawing (not reality). A scope scale of 1:4, for example, signifies that 1" on the 10" x 8" scope display area corresponds to 4" on a 40" x 32" drawing.

### Magnification and Frame Control

The 10" x 8" drawing area of the scope is bounded by four displayed lines, called a frame. The buttons and register concerned with magnification and frame manipulation are as follows:

HALF FRAME

Any part of the drawing can be magnified by touching the HALF FRAME light button and positioning the frame, now reduced to half-size, over the desired area of the drawing by depressing the MOVE FRAME push button in the Sketch mode. Releasing this button will cause the frame to return to full scope size and correspondingly magnify everything within it.

MOVE FRAME

The currently available scope scales are:

<u>10" x 8" Display Area</u>	=	<u>Paper Drawing Area</u>
1:8	=	80" x 64"
1:4	=	40" x 32"
1:2	=	20" x 16"
1:1	=	10" x 8"
2:1	=	5" x 4"
4:1	=	2-1/2" x 2"
8:1	=	1-1/4" x 1"
16:1	=	5/8" x 1/2"

As this table indicates, the scope-scale-operation can be used for preparing drawings smaller than 80" x 64" and magnifying portions of a drawing for examination or detail work. For the former purpose, all work is done at a magnified scale; for instance, a scope scale of 1:1 is used for 10" x 8" drawings. For the latter purpose, the frame is reduced by halving it as many times as desired, thus expanding the frame and its contents to full scope size. When the detail work is completed, the scope is set to the next desired scale.

FULL VIEW	To view the entire drawing and show the relative location and size of the frame on it, the FULL VIEW light button is touched. To reset the frame to full scope size, the RESET FRAME light button is touched. The latter does not affect the scope scale but restores the frame to 10" x 8"
RESET FRAME	
DOUBLE FRAME	Whenever the frame has been reduced in size by halving, it may be increased by touching the DOUBLE FRAME light button, which will double the size of the frame. This permits the user to move back up the scale, step by step, as well as zoom down it. The Scope Scale Register will always display the scope scale of the current view. The frame may be doubled beyond the 10" x 8" scope area if the scope scale in use is less than 1:8, thereby allowing the user to see more of the paper drawing than he is actually viewing.
SCOPE SCALE Register	

### Group Number Control

When an entity is generated, it is assigned the group number appearing in the Group Number Register. The following registers and light buttons are available for assigning and changing group numbers:

GROUP NUMBER  
Register

This register contains the group number currently being assigned to entities. It is also used for other functions.

PICKED GROUP  
NUMBER Register

This register is used in the Construct mode to identify the entities to be used.

AC → GPN

This light button is used to transfer the integer in the selected accumulator to the Group Number Register.

INDEX GROUP NO.

This light button is used to increase the value of the Group Number Register by one.

-1

This light button is used to decrease the number in the Group Number Register by one.

AC → PGPN

This light button is used to transfer the integer in the selected accumulator to the Picked Group Number Register.

REASSIGN GROUP  
TO NO. IN AC

This light button permits the assignment of a new group number to an entire group of entities. The group to be changed is specified by the Picked Group Number Register and the new group number to be assigned is in the Group Number Register. The new group number will appear in the Group Number Register at the completion of the operation.

## SPECIAL FUNCTIONS

DDP has a special-function selection feature which enables the user to extend his control to auxiliary functions. An auxiliary function is called by entering the function number as an integer into AC-1 and the desired mode number as an integer into AC-2. Then the SPECIAL FUNCTION push button is depressed and released.

## Alphanumeric Input

Remarks may be added to the drawing by means of the SPECIAL FUNCTION push button. Since they are entities, remarks have group numbers, may be moved about, and possess all the other properties associated with entities. The basic size of the characters used is .12 x .08 inches at a scope scale of 1:1. The size of the characters will vary with scope scale and can also be changed by the user.

The procedure for entering remarks is to position the writing-point at the location of the first desired character, enter a 1 into AC-1, clear AC-2, then depress and release the SPECIAL FUNCTION push button. The typewriter will execute a carriage return to indicate a ready state for accepting alphanumeric information. The remark is entered through the keyboard and terminated by momentarily depressing the push button on the light pen. As each character is typed, it is displayed in its proper position on the scope.

The allowable characters are capital letters, digits 0 through 9, and the following symbols: quotation marks (single and double), brackets (right and left), comma, period, oblique, question mark and right arrow; and plus, minus, equal, square root and summation signs. Twenty-four spaces are reserved for future use. The typewriter keyboard symbol  $\uparrow$  is used to specify a change in the size of the characters. Magnification is given relative to the base size of .12 x .08 inches. The number entered immediately following the  $\uparrow$  symbol specifies the power of magnification; for example  $\uparrow 3$  means that all future characters will be .36 x .24 inches at a scope scale of 1:1.

The symbol  $v$  specifies a subscript mode and all the characters that follow will be entered as subscripts relative to the last character before the  $v$ . Subscripts can be displayed at different levels by the successive use of  $v$ . The character  $\wedge$  specifies superscript mode relative to the last character before  $\wedge$ . It is therefore possible to have subscripts and superscripts at different levels, to have subscript superscripts, and vice versa. To return to normal mode, the character  $\sim$  is entered.

Characters may be deleted by typing  $<$ . Each time this is done, the preceding character is erased from the display. The carriage return, space bar, back space and tab are used normally. Tabs are set by SPECIAL FUNCTION light-button operation.

### Concluding Remarks

The Digigraphic Display Program is a basic graphic drawing program. Except for a few control routines, it consists of a string of independent routines which can be supplemented by those of a special-purpose nature that are peculiar to the needs of individual users.

The Entity Table format has been designed for minimum storage space since detailed graphic drawings can be several thousand words long. Address pointers for common points, connected groups, etc., have not been included since their use is directed toward saving computer time for searching. With a completely buffered single display scope, however, there is abundant time for internal searches.

There are and will continue to be valid reasons advanced for expanding the entity format, and the DDP is so designed as to permit this to be done with very little program modification.



DX-1 CONSOLE CONFIGURATION  
FOR DIGIGRAPHIC DISPLAY PROGRAM

# SIGNAL REPRESENTATION AND MEASUREMENT DATA MANIPULATION IN N-SPACE USING AN ON-LINE PDP SYSTEM

Charlton M. Walter

Air Force Cambridge Research Laboratories  
Bedford, Massachusetts

## ABSTRACT

The representation of functions and time varying signals as points in an N-dimensional vector space has proved to be a highly useful mnemonic device for abstractly visualizing the behavior of complex sensor data processing and pattern recognition schemes. In order to articulate this capability an on-line programming system is being evolved for the Experimental Dynamic Processor DX-1. The present version of this system permits the user to rapidly specify various two-dimensional sections through N-space and to project both stored and real-time signal data onto the specified section as points of light on a CRT, color coded to exhibit a priori information about the source of the sensor data.

A variety of options permit signal data to be input and displayed in real-time, projected on any plane through N-space, while also being stored for subsequent examination from different points of view. Stored data, to which labels have been attached, can be called up and projected on the given plane, along with the data being input in real-time, for data comparison and monitoring purposes. Data derived from various sources can be both labeled and color coded. This type of coding is highly effective as a means for monitoring the pattern discrimination capabilities of different types of information filtering and pattern recognition systems.

Alternative points-of-view can be specified through such options as the use of the light pencil to construct new coordinate axes in N-space, or through the use of stored sample vectors as axes. Means are also provided for analytically generating new coordinates using eigenvector attribute extraction and other statistical filtering and smoothing techniques.

Both central processing units (PDP-1's) of the DX-1 system are used in implementing the above system. One CPU is used for real-time data inputting through an A-D converter and for carrying out the data projection and transformation operation, while the other CPU is primarily used for display bookkeeping and updating.

## INTRODUCTION

The representation of functions and of time-varying signals as points in an N-dimensional vector space has become an important means for abstractly visualizing the behavior of various complex sensor data processing and pattern recognition schemes. The dynamic application of the approach to actual data has heretofore been impeded by the lack of computational capability coupled in real-time to an effective display medium and backed by simple means for permitting the investigator to rapidly specify new points-of view from which he may wish to examine the data.

The system to be described involves the use of the Experimental Dynamic Processor, DX-1<sup>1</sup>,

consisting of two interconnected PDP-1 processors, coupled to a colored oscilloscope for graphic output, and to A-D conversion equipment for new data inputting. Both on-line keyboard and light pencil were provided for logical and functional communication with the processing system.

## SIGNAL REPRESENTATION IN N-SPACE

In the discussion we shall consider a signal vector,  $x$ , as an ordered N-tuple of "component measurements,"  $(x_1, \dots, x_n)$ . The component measurements may be samples from some continuous process  $x(t)$ , at discrete intervals of time, e.g.,  $x_i = x(t+i\Delta t)$ ,  $i=1, \dots, N$ . or, in other situations, the components of the vector may represent the outcome of a collection of difference measurements carried out according to a given set of operational procedures, e.g., the results of a set of psychological and physiological tests carried out on a specified individual.

The numerical value of the outcome of each measurement may be considered as the distance out from a fixed origin along a coordinate axis associated with that particular measurement procedure. Since the separate measurements may be interrelated in various ways, both statistically and dynamically, it is convenient to let the angle between pairs of coordinate axes be specified by some suitable measure of this interrelation. Generally this is done in such a way that minimum absolute value of the interrelationship, in the appropriate statistical or dynamical sense, corresponds to orthogonal axes, and maximum value to parallel axes.

Since it is difficult to visualize more than three coordinates at once, and since, for purposes of graphic presentation, it is difficult to handle more than two non-orthogonal coordinates at a time, we shall confine our attention to two-dimensional sections through the N-dimensional space. This is not particularly unfortunate since the concept of an individual measurement operation can be viewed as an orthogonal projection of the signal vector on the coordinate axis associated with the given measurement procedure. From considerations of consistency and completeness of the measurement process, it is desirable to work only with vectors of "unit length," where length  $\|x\|$  of vector  $x$ , and angle  $\theta$  between vectors  $x$  and  $y$  are defined, in terms of an inner product operation  $(x|y)$  as  $\|x\| = \sqrt{(x|x)}$  as  $\cos \theta_{xy} = \frac{(x|y)}{\|x\| \|y\|}$ . For orthogonal components  $(x_1, \dots, x_n)$  and

$(y_1, \dots, y_n)$ , one may write  $(x|y) = \sum_{i=1}^N x_i y_i$ .



In particular, any normalized vectors  $u, v$ , etc., in the given  $N$ -space may be considered as measurement - or data filtering - procedures, and we may inquire how the other data vectors, or basis measurement procedures, are interrelated, when viewed from the procedures  $u$  and  $v$ .

The problem now is reduced to determining the projection of an arbitrary normalized vector,  $x$ , on the plane determined by the vectors  $u$  and  $v$ . Analytically, this can be characterized by computing the coordinates  $u_x$  and  $v_x$ , where

$$u_x = (x|u),$$

$$v_x = (x|v),$$

and observing that, for the  $u$ -axis arbitrarily taken as horizontal and for a  $v$ -axis making an angle  $\theta$  with  $u$ , the vertical "scope" coordinate  $y_x$  (cf. Fig. 1) of the projected vector  $x$  will be

$$\theta y_x = \frac{(x|v) - (x|u)(u|v)}{\sqrt{1 - (u|v)^2}} .$$

## SIGNAL VECTOR MANIPULATION PROCEDURES

Typical of the kinds of control which are needed in attempting to maneuver in  $N$ -space is the requirement for rough specification of a new measurement or data filtering procedure based upon intuition derived from the examination of certain examples of prior signals, specified in terms of their component representation on a previous basis of measurement procedures.

This requirement is handled by providing the investigator with capability for selecting, through suitable alpha-numeric descriptors, previously stored vectors and displaying them on the color oscilloscope in coordinate form, using different colors to identify the different vectors. The light pencil can then be used to sketch in the structure of new filters, using intuition based upon the displayed data and a "feel" for the process under investigation, as illustrated in Figure 2. All of the relevant signal data can then be called out, through alpha-numeric category descriptors, and displayed as points projected on the plane specified by the new filters, using different colors for the different categories into which the data has been grouped. This is illustrated in Figure 3, where, in lieu of color, we have utilized different symbols to distinguish the different types of data.

The hand sketch approach to filter investigation is particularly useful for obtaining a "feel" for which factors are most influential, and roughly to what extent, based upon various heuristic criteria of utility. It is also highly useful in obtaining a feel for the effects of small perturbations about a filter structure which has been designed using some analytical procedure.

Means for calling up various statistical filter design procedures have also been incorporated into the program control. This is at present limited to the designation of pre-programmed subroutines, which are called out and executed on demand, and does not yet have incorporated a capability for the on-line synthesis of arbitrary analytical design procedures using an on-line algebraic translator.

A rather elaborate eigenvector filter design procedure has been implemented, however. This system [2] has the capability for forming an interaction matrix from designated samples of stored signal vectors, using any of a variety of different definitions of "interaction." The system then proceeds to find the orthogonal transformation which diagonalizes the matrix. Options are available for displaying various intermediate iterations of the first  $m$  eigenvectors and eigenvalues, in order to provide information on the rapidity of convergence of the process. Figure 4 illustrates the behavior of the system when operating on a typical signal data cross-correlation matrix.

Control options are also provided in the main signal representation system for the real-time inputting, display, and storage of signal data. Limitations on high speed storage and the absence of a drum buffer, for both data storage and display updating, on the Phase I, DX-1 system, places a serious limitation on the amount of data, and signal data sample rate, which can currently be handled in the real-time input mode.

Incoming signal data can be both displayed in coordinate form (on the top half of the scope) and simultaneously projected and accumulated on any specified pair of filter axes (on the lower half of the scope) as illustrated in Figure 5.

#### References

1. Walter, C. M., "The Experimental Dynamic Processor DX-1", 1962 IRE International Convention Record, Part 9, March 1962.
2. Gagan, R. P., Wisowaty, M.; Walter, C. M., "An on-line Sub-Program for the Determination of Eigenvalues and Eigenvectors with Graphic Convergence Monitoring Features", Scientific Report No. 2, AF19(628)-1614, Wolf R & D Corp., AFCRL-63-365, September 1963.

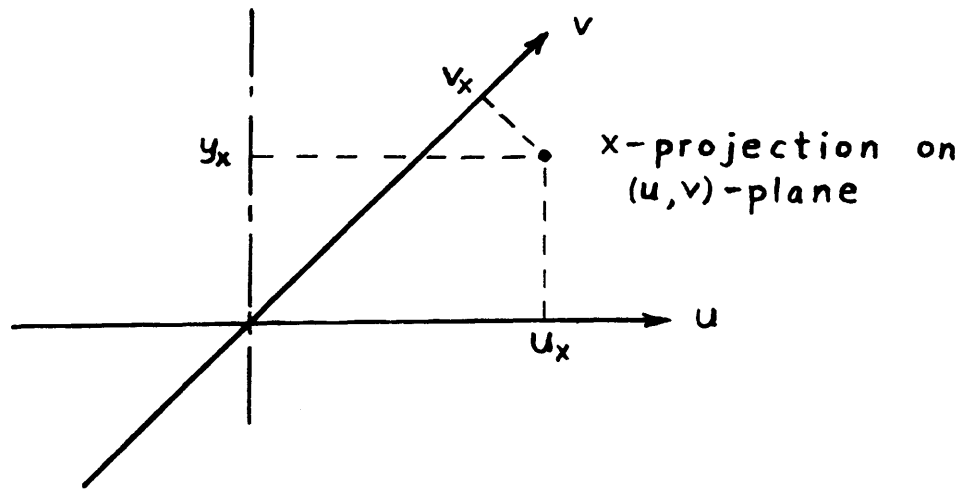


Fig. 1

Geometry of projection of vector  $x$  on plane determined by vectors  $u$  and  $v$ .

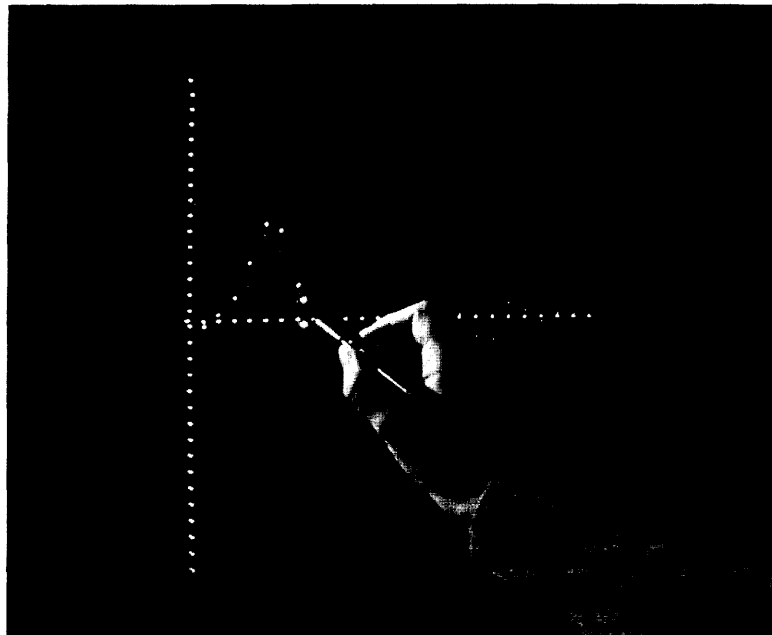


Fig. 2

Figure 2. Use of light pencil in drawing filter structure. Background reference data is displayed in different colors.

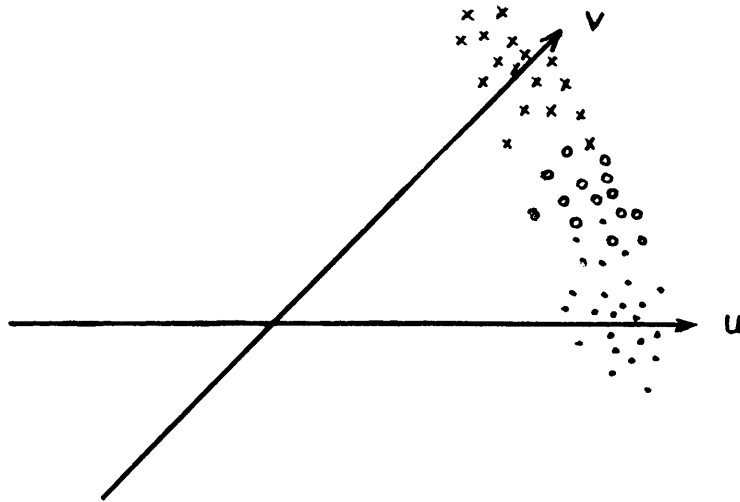


Fig. 3

Figure 3. Sketch made from typical colored scope display showing partially clustered data from different signal sources projected on  $(u,v)$  - plane.

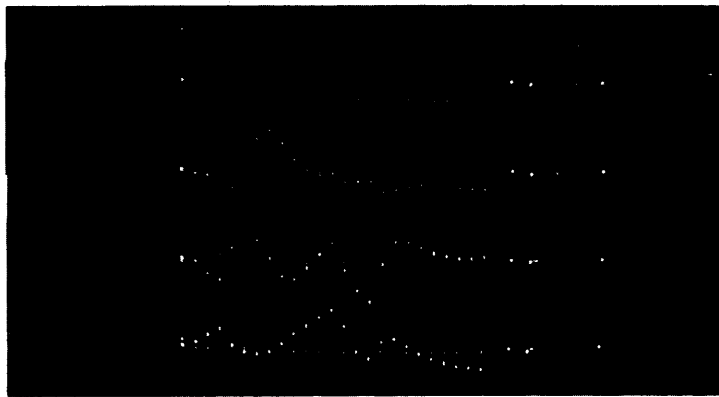


Fig. 4

Overlay of fifty iterations of first five eigenvectors and associated eigenvalues (shown ordered in row presentation on the right and unordered in column presentation following fifth eigenvector) of typical cross-correlation matrix based on pressure cardiograph data.



Fig. 5A

Cardiograph signal input (top) and sets of signal data from three sources projected (bottom) on an arbitrary set of filters.

5.

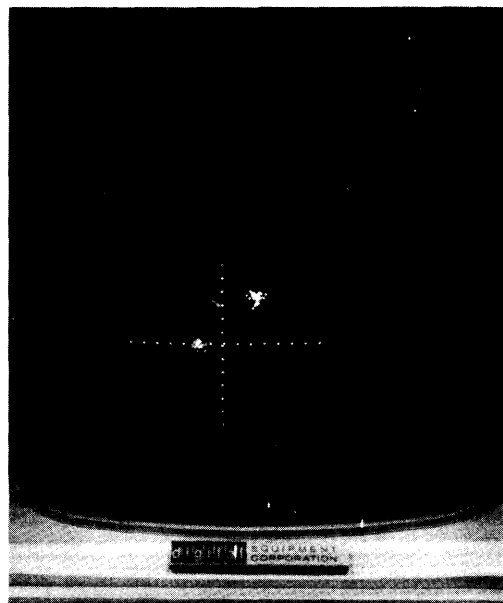


Fig. 5B

Similar data to that of figure 5a projected on a set of eigenfilters.



STEPS TOWARD COMPUTER SIMULATION OF  
SMALL GROUP BEHAVIOR

Dr. Thornton J. Roby<sup>1</sup> and Raymond S. Nickerson<sup>2</sup>

ABSTRACT

This paper will describe the initial phases of an attempt to simulate on the PDP-1 computer several aspects of small group behavior.

The elements of the model include a stochastic, segmented environment, a set of action agents (both sensors and effectors) which operate on it, and a decision agent which assigns action agents to environmental regions in accordance with a variety of rules.

The environment is stochastic in the sense that it is in a state of continual flux. It is segmented in that different regions may be changing at different rates.

The locus of interest is the steady state condition of the environment as a function of interaction between such parameters as the following:

- (1) The change characteristic of the environment,
- (2) The reliability of the sensing agents,
- (3) The precision and activity level of the effector agents,
- (4) Information delay within sensor-effector communication link,
- (5) Decision rules with respect to the assignment of sensors and effectors to environmental regions.

The paper will discuss the conceptual framework within which the model is developed, outline several substantive problems under investigation, present results obtained to date, consider some programming problems encountered in the inception and extension of the model and suggest possible future refinements and elaborations.

Although in its present form the model is cast in the context of the small group, an attempt has been made to keep it sufficiently general to be applicable to other information processing systems.

1. Dr. Thornton J. Roby, Tufts University, Medford, Massachusetts
2. Raymond S. Nickerson, Decision Sciences Laboratory, ESD, AFSC, Hanscom Air Force Base, Bedford, Massachusetts

## INTRODUCTION

The general objective of the work to be described is to develop a simulation model to be used for investigating certain aspects of the performance of man-machine systems. The model will incorporate behavior and effectiveness of the human component, ordinarily a small group or team, and other parameters describing important properties of the task environment. Assuming definite functional relations between these sets of parameters and specific numerical values, it will attempt to predict the criterial performance of the total system.

The particular substantive framework used as a point of departure for these investigations has been discussed by one of the writers in a number of earlier reports (Roby and Forgays 1953, Roby and Lanzetta 1956, 1958, Roby 1960). This framework treats a small work group or team as a special kind of information transducer or input - output mechanism. It postulates distinct component processes that may take a variety of superficially different forms in various groups but that are functionally equivalent. Examples of these component processes are: vigilance, information storage, information transmission, and action coordination.

In regard to each of these component processes, it is stressed that their importance for criterial performance in a task environment can only be assessed when the specific task demands of that environment are known.

\* This is a draft of Technical Documentary Report No. ESD-TDR-63-629. This work was supported in part by Contract Number AF19(628)-2450 with Tufts University. Further reproduction is authorized to satisfy the needs of the U.S. Government. The authors wish to acknowledge the assistance of Mr. Alan Meskil in preparing the figures.



Storage, for example, is an extraneous activity if complete, fresh information on the environmental state is always accessible.

In addition to its emphasis on the critical demands as imposed by the task, a second feature of this framework is important for the present simulation effort. This is the fact that the framework suggests description and analysis of the basic processes at various levels of detail. Thus, for some purposes it is essential to treat vigilance, storage, transmission and perhaps coding information each as distinct processes. For other purposes, however, it is sufficient to represent the net effects of this whole set of component processes by a single aggregative index -- in this case an index of overall information "orientation", where orientation is defined as the correspondence between the group's information state and the true state of the world. This representation of the net effects of several component terms in a single summary index is similar in spirit to the use of "lumped circuit constants" in network design.

In practice, then, the projected series of simulation studies will begin by investigating parametric relations between rather gross indices of the group or team and the task environment. As these relations are elaborated, component processes will be shredded out and examined in greater detail. For example, initial study might concern the effects of overall group orientation on performance in a specified task environment. Follow-up studies would then examine the results of producing a fixed amount of orientation by component processes as described above. The plan is to proceed step-by-step from the rather abstract, aggregative parameters to a detailed simulation model that reproduces, almost literally, the task behavior of live groups, in the laboratory or field organizations.

The present report will indicate the tactics to be followed in this series of investigations and will follow through several illustrative problems. It will also introduce some of the methodological questions that beset this approach and describe our present tentative response to these questions.

#### System characteristics

The investigation is concerned primarily with a special class of task environment that can be described in terms of attributes. That is, it assumes that all relevant conditions can be expressed in terms of the presence or absence of certain elements. Real life approximations to such environments are represented by machine maintenance organizations, machines being functional or non-functional, and inventory systems in which supply items are considered on hand or exhausted.

Attribute task environments are particularly amenable to treatment in terms of digital models and hence to computer simulation. The rules describing functional interaction among components can be stated very simply and unequivocally for such models. Finally, although the restriction to "yes-no" values may not be fully descriptive for most real world systems, the latter can, in principle, be represented to any desired degree of fidelity by sufficiently increasing the number of binary variables.

The particular physical system to be investigated is not based directly on any real world prototypes. It is instead an entirely hypothetical task situation that is simple enough so that the operations involved can be visualized readily, yet rich enough so that rather complex processes can ultimately be incorporated. This choice of make-believe task situation

detracts from immediate applicability but it also guards against the intrusion of prejudices about how the system ought to be organized.

The hypothetical task environment consists of a large warehouse containing a number of lights (attributes), each controlled by a single switch. The ideal condition of this environment is one in which all lights are on. This state is maintained as well as possible by an agent G who represents the action of a small work group or team. The ideal state is, however, undermined by a second agent (or set of agents) D representing a sort of disruptive demon.

This formulation of the problem makes it possible to express all relevant conditions in terms of three sets of indices. G parameters refer to the operator characteristics of the "good" agent; D parameters are the operator effects of the disruptive agent; and W parameters describe the state of the world as determined by these agents.

It should be noted that D characteristics, though related to a super-imposed demon, may actually be considered properties of the environment. That is, they describe the tendency for the physical system to deteriorate over time -- the refractoriness of W. In a real life problem, D effects might be determined by the reliability of machine components, and so forth. Again, although G parameters primarily concern performance of constructive human agents they may also be affected in the real world by physical conditions -- e. g. , communication facilities. Finally, W (the dependent variable) might include, in a real problem, certain conditions concerning the social state of the group.

#### Representation

As noted above, the dependent variable of interest in this situation is simply the number or proportion of lights "on" at a specified time. It is

most convenient to represent this variable by a vector of the form  $(P \{1\}, P \{0\})$  in which  $P \{1\}$  is the number or proportion of lights "on" and  $P \{0\}$  is the complementary number or proportion of lights "off".

The G agent can then be represented as a matrix of the form

$$\begin{pmatrix} g(11) & g(10) \\ g(01) & g(00) \end{pmatrix}$$

In this matrix,  $g(11)$  is the probability that the G agent, upon encountering a light that is on, will leave it on. The cell entry  $g(10)$  is the probability that G will, incorrectly or inadvertently, turn an initially "on" element "off". The sum of  $g(11)$  and  $g(10)$  must always be unit. The two entries in the second row,  $g(01)$  and  $g(00)$  are the corresponding probabilities that G will turn an initially "off" element "on" or leave it "off".

The direct effect of G on the environment is fully described by this matrix. To illustrate, suppose that a G described by

$$\begin{pmatrix} 1.00 & .00 \\ .75 & .25 \end{pmatrix}$$

acts on an initial environmental state (W) described by (600, 400), i. e., 600 lights on and 400 lights off. The top row of the matrix shows that he will leave all of the 600 initially "on" lights "on", and the second row says that he will turn 75% of the initially "off" elements "on". Thus, after G has operated once on the warehouse, the new W is described as (900, 100).

It will be seen that this new vector is obtained by the conventional matrix postmultiplication of the initial vector by the G matrix. As will be demonstrated, much of the important functional behavior of this system can be represented by suitable vector-matrix operations.

The operational effect of the D agent is represented by a matrix that is precisely parallel to G. Thus in the matrix

$$\begin{pmatrix} d(11) & d(10) \\ d(01) & d(00) \end{pmatrix}$$

the top row describes D's effect on initially "on" elements , and the bottom row describes his effect on initially "off" elements. The vector multiplication procedure is identical with that for the G matrix.

In practice it is convenient to consider D and G as taking distinct turns in operating on the warehouse environment. Beginning with a specified initial vector, D operates first, and typically produces a somewhat degraded W (unless it is already at an unfavorable point). Then G operates on the resulting vector, typically increasing the number of lights "on". This cycle is repeated for the duration of the hypothetical performance period.

The resulting process can be studied by the matrix techniques introduced above. If  $W_1$  is the initial vector, it becomes  $W_1D$  after operation by the D agent, and this in turn becomes  $(W_1D)G$  after operation by the G agent. A crucial point here is that this process is associative in the mathematical sense: that is, the result of applying G to the term  $(W_1D)$  is identical with the result of applying the matrix product  $(DG)$  to the initial vector,  $W_1$ . This leads to a great simplification of the symbolic operation.

As is well known, matrix products are not generally commutative. That is, the matrix product taken in one order differs from the product taken in the opposite order. Thus, if

$$D = \begin{pmatrix} 1/2 & 1/2 \\ 0 & 1 \end{pmatrix} \text{ and } G = \begin{pmatrix} 3/4 & 1/4 \\ 1/2 & 1/2 \end{pmatrix}$$

the product

$$DG = \begin{pmatrix} 5/8 & 3/8 \\ 4/8 & 4/8 \end{pmatrix} \text{ whereas } GD \text{ is } \begin{pmatrix} 3/8 & 5/8 \\ 2/8 & 6/8 \end{pmatrix}$$

This means that some care is necessary in studying composite products. In the present case, however, the fact that D and G always operate in that order, and that the dependent measure W is taken after a full cycle, makes it possible to treat the product DG as a composite matrix. This matrix describes the behavior of the total system during a given epoch, and it is quite permissible to take powers of this matrix, e. g. ,  $(DG)^2$ ,  $(DG)^3$ , etc. , to describe successive operations for several epochs-- $DG, DG, DG, DG, DG$ , etc.

Finally, among a number of aspects of the performance of such systems that might be investigated, the simplest and perhaps most important is the point of equilibrium, or "steady-state" that it assumes. For the present type of system, such steady-state vectors are always unique and independent of the initial condition of the environment. In simplest terms it is precisely the vector that is unchanged upon multiplication by the composite operator DG.

Although the determination of the "characteristic vectors" for general matrices may be a difficult and tedious problem, the characteristic vectors corresponding to the steady-states for the 2 x 2 matrices, such as used here, can be determined by inspection. They simply require that the ratio of terms in the steady-state vector be equal to the ratio of the terms in the lower right-hand and upper left-hand cells of the composite matrix.

For the illustrative DG matrix shown above, these terms are, respectively 4/8 and 3/8. The corresponding steady-state vector would be (4/7, 3/7). It is readily confirmed that,

$$\begin{aligned} (4/7, 3/7) \begin{pmatrix} 5/8 & 3/8 \\ 4/8 & 4/8 \end{pmatrix} &= (32/56, 24/56) \\ &= (4/7, 3/7) \end{aligned}$$

It will be observed that this criterion is "favorable" to G -- that is, it represents the state of the environment at the point in the cycle when the most lights are on. For some real world systems this might be a realistic criterial measure, but for other systems, the lowest point in the cycle or the average value would be a more valid criterion.

An estimate of the least number of lights on at steady-state can be obtained very directly by multiplying the above steady-state vector by D alone. In this case, the result is the vector  $(2/7, 5/7)$  which, it will be noted, is just the steady-state vector for the composite operator GD. The average of the two extreme steady-state vectors -- in this case  $(3/7, 4/7)$  -- would give an indication of the mean proportion of lights on during the entire cycle.

The following sections will build on this general conceptual framework to investigate three substantive problems. The first two problems treat the G agent as a unitary organism and are concerned with the consequences of a total activity aggregate. The third problem considers explicitly the division of labor question under special conditions. In each case a mathematical (analytical) treatment of the problem is possible and will precede the discussion of the computer study.

The purpose of the analytical discussion in this report is largely to indicate the nature of the functional relations, and to show why certain effects are anticipated in the computer study. Actually, it will be evident that the analytical treatment could be carried considerably farther in each case, It may be questioned whether this matrix treatment could not be extended to cover the entire range of problems projected for simulation study. This question is reconsidered in the final discussion section but, for now, it may be

remarked that the duplicate approach used here has proved very useful at this stage of investigation. The analytical and computer programming operation seem to suggest different lines of attack, and of course the availability of reciprocally confirmatory results is reassuring.

Problem 1: Precision and Activity Potential

The first problem investigated is one of pervasive relevance, not only to group performance, but to the performance of individual organisms. It concerns the extent to which directed, precise, apposite, action can substitute for sheer brute force activity, and vice versa. The objective of the investigation is to explore the conditions under which such substitution is possible and to determine the exact equivalences. The present formulation of the problem permits a succinct quantitative expression of the intuitively meaningful but rather vague terms "precision" and "activity potential".

The precision ratio of group adjustment will here be defined by the quotient:

$$\frac{g(01)}{g(01) + g(10)}$$

As an illustration, the precision ratio for the G agent instanced above is

$\frac{.50}{.50 + .25} = .67$ . In general, of course, a higher precision ratio is associated with superior performance.

It is immediately clear, however, that a given precision ratio may characterize a very wide range of operator matrices. For example, the precision ratio .67 describes all matrices in which  $g(01) = 2g(10)$ . Thus it describes the matrix

$$\begin{pmatrix} .50 & .50 \\ 1.00 & .00 \end{pmatrix}$$



at one extreme, and the matrix

$$\begin{pmatrix} .99 & .01 \\ .02 & .98 \end{pmatrix}$$

toward the other extreme. The independent parameter that differentiates between the two matrices is obviously an index of "activity". In some sense, the first G agent is 50 times as active as the second one. Activity potential will be defined as the mean of the two probabilities associated with change:

$$\frac{g(01) + g(10)}{2}$$

Although precision and activity are meaningful characteristics of the group, it is clear that they should not be construed as absolute properties. Rather they are both potentials for performance, whose effect can be determined only after it is known what kind of environment is faced by G. That is, the actual number of off-to-on and on-to-off changes depends on the input vector which G encounters on each turn.

By the same token, the 'effectiveness' of an operator is partially determined by the nature of the environment (in this case the D agent) with which the operator must deal. In gross terms, activity is relatively more important for control of a rapidly disintegrating state, while precision is relatively more important for final adjustment of a more quiescent environment.

To take extreme examples, consider the two G agents

$$G_1 = \begin{pmatrix} .9 & .1 \\ .2 & .8 \end{pmatrix} \quad \text{and} \quad G_2 = \begin{pmatrix} .4 & .6 \\ .8 & .2 \end{pmatrix}$$

Here the precision ratio for  $G_1$  is .67 and that for  $G_2$  is only .57. On the other hand,  $G_2$  is at least four times as active as  $G_1$ .

Suppose first that D is completely inert, i. e. ,

$$D = \begin{pmatrix} 1.0 & 0 \\ 0 & 1.0 \end{pmatrix}$$

Then the steady-state for either G agent will be its own characteristic vector, (667, 333) and (571, 429) respectively. On the other hand, if the D operator is

$$\begin{pmatrix} .5 & .5 \\ .5 & .5 \end{pmatrix}$$

the resulting composite matrices are respectively

$$\begin{pmatrix} .55 & .45 \\ .55 & .45 \end{pmatrix} \quad \begin{pmatrix} .60 & .40 \\ .60 & .40 \end{pmatrix}$$

The corresponding steady-states are (550, 450) and (600, 400); that is, the more active agent is now superior.

The interesting problem in this case is to find the D agent for which these two very different types of G operator are exactly equivalent. Alternatively, we can begin with given D operators and determine families of G agents with varying precision and activity that are exactly equivalent.

The attempt to actually generate useful data from the model forces a consideration of sampling economics since an exhaustive treatment of D and G parameter combinations is clearly prohibitive. Fortunately, the substantive assumptions of the model impose some restrictions on the types of G and D matrices which should be studied. In the first place, it is assumed that G's objective is to keep the lights on: thus, the range of interesting G operator matrices is limited to those for which the probability of a "positive" change,  $g(01)$ , is greater than the probability of a "negative" change,  $g(10)$ . Secondly, although D is not conceived as necessarily purposive, it is assumed that his influence is primarily disruptive. A friendly D is ruled out by the restriction that  $d(10)$  must be at least as great as  $d(01)$ . For purposes of this report the sampling space is further reduced by restricting attention

to two "pure" types of D agent: a neutral D, who turns lights "off" and "on" with equal probability ( $d(01) = d(10)$ ), and a malign D who only turns lights "off" ( $d(01) = 0$ ). Actually, since these are the extreme cases of all possible D's of interest (excluding friendly D's), it can be assumed that intermediate cases will be adequately represented by mixtures of these operators. The results to be reported were all generated within these restrictions.

It should be pointed out that the results to be reported here are included merely for illustrative purposes and represent rather arbitrary specific aspects of the general problem areas. As previously mentioned the independent variable of interest is the steady-state of W as measured following G's activity. Each data point denoting steady-state on the following graphs represents the mean per cent of lights on in W during the 50 final epochs of activity, the per cent being determined following G's activity at each epoch. The time (no. of epochs) required for W to reach steady-state depends on the activity potential of the agents as well as on the initial state of W. With most of the activity parameters used to date, W reaches steady-state in relatively few epochs. Unless otherwise specified, all runs for a given set of data were terminated after the same number of epochs (either 150 or 250).

Figure 1 represents steady-state as a joint function of  $g(01)$  and  $g(10)$  when D is neutral and has an activity potential of .25. A set of such surfaces has been generated for different activity potentials of D. By imposing on such a surface lines representing different G precision ratios ( $PR_G$ ), others representing different activity potentials ( $AP_G$ ), and still others representing different steadystates ( $SS_G$ ), the basis is provided for identifying families of G's with varying precision and activity but exactly equivalent with respect to the resultant steady-state.

Examples of such families are included in Figure 2. Each curve identifies all those combinations of  $PR_G$  and  $AP_G$  which will produce a given  $SS_G$  when working in opposition to the specified D. As would be expected, losses on either variable may be compensated for by gains on the other, but not in a one-to-one fashion. In general, if  $AP_G$  is initially high, a large decrease can be offset by a relatively small increase in  $PR_G$ ; however, if  $AP_G$  is initially low, further decreases may be offset only with proportionately greater gain in  $PR_G$ . The exact trade-offs of course, vary also with  $SS_G$  and the characteristics of D.

Figure 3 shows  $SS_G$  as a function of the activity of D for a set of Gs, with equal PR but variable AP. That the malign D has a generally greater effect on  $SS_G$  than the neutral D is shown by the greater dispersion of points in Figure 3b. It will be noted that irrespective of the type of D, when  $AP_G > .5$ ,  $SS_G$  varies inversely with  $AP_D$ , when  $AP_G = .5$ ,  $SS_G = 667$  and  $AP_D$  has no effect, and when  $AP_G < .5$ ,  $SS_G$  varies directly with  $AP_D$ . The last point is somewhat surprising since it appears that the state of affairs improves as a function of the amount of activity of an essentially blind D.

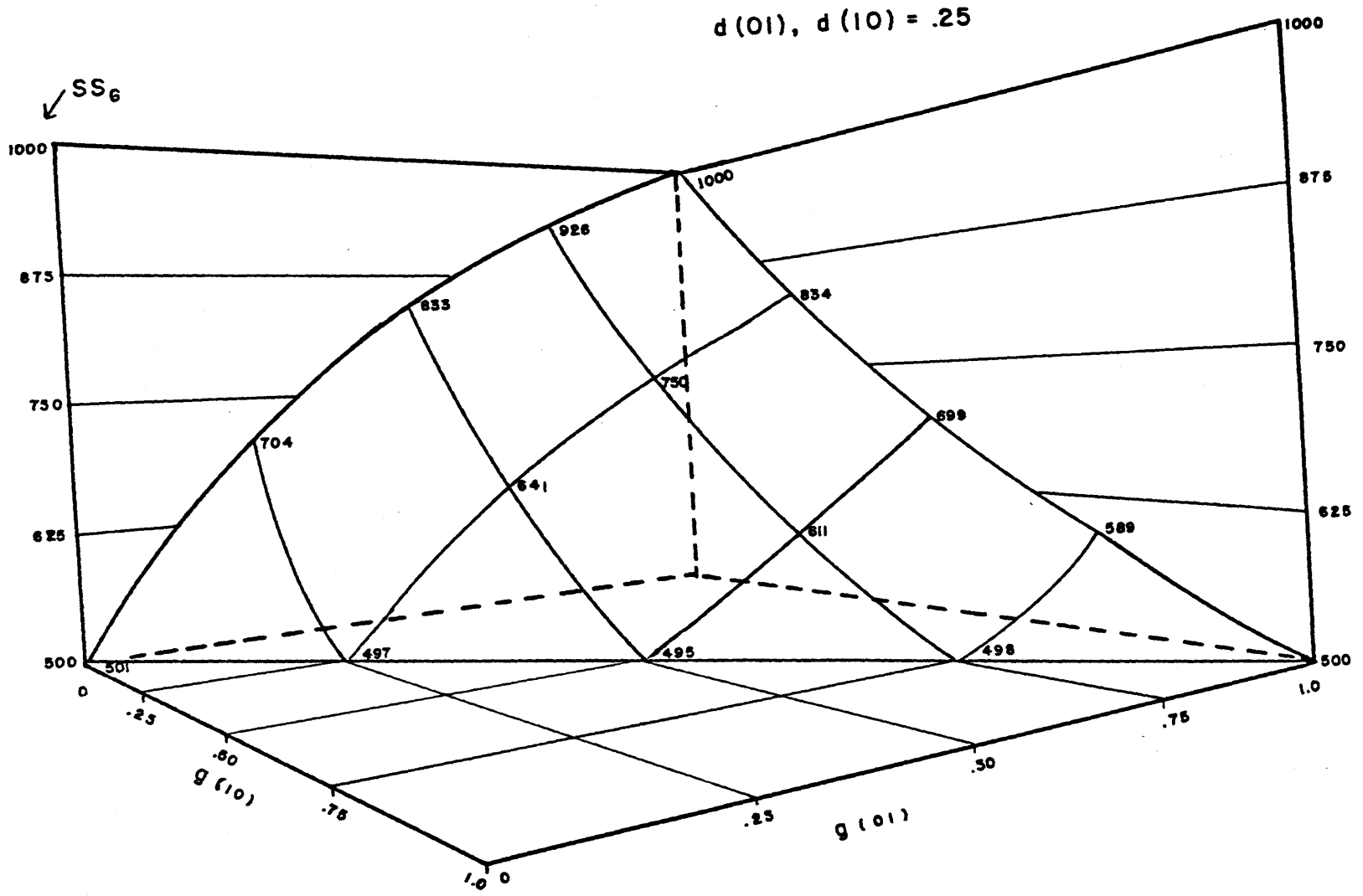


FIG 1 Steady State of W as a function of  $g(01)$  and  $g(10)$  with constant D.

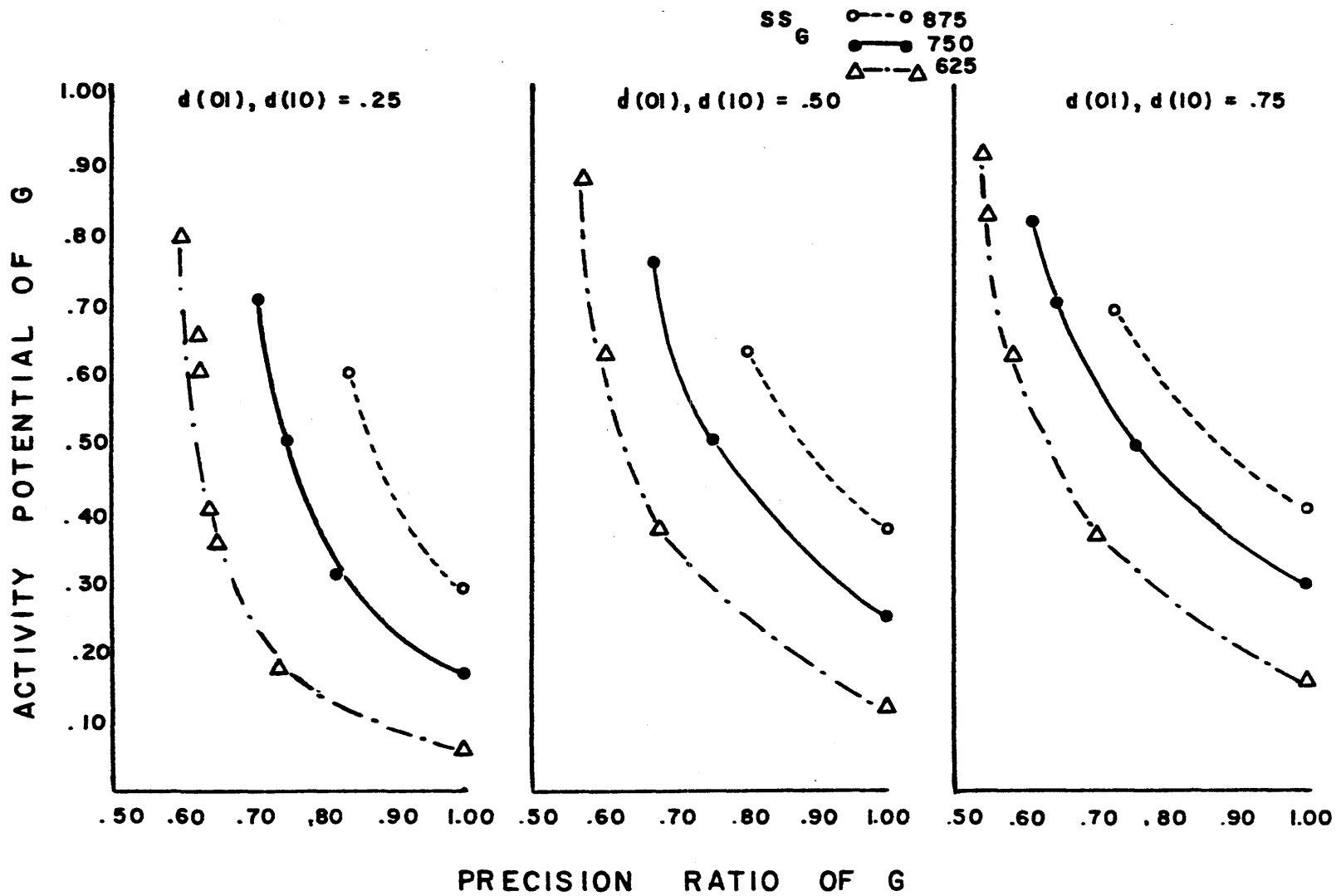


FIG 2 Combinations of  $AP_G$  and  $PR_G$  which will produce equivalent steady states when working against the specified  $D_s$ . Data points were read from graphs such as Figure 1.

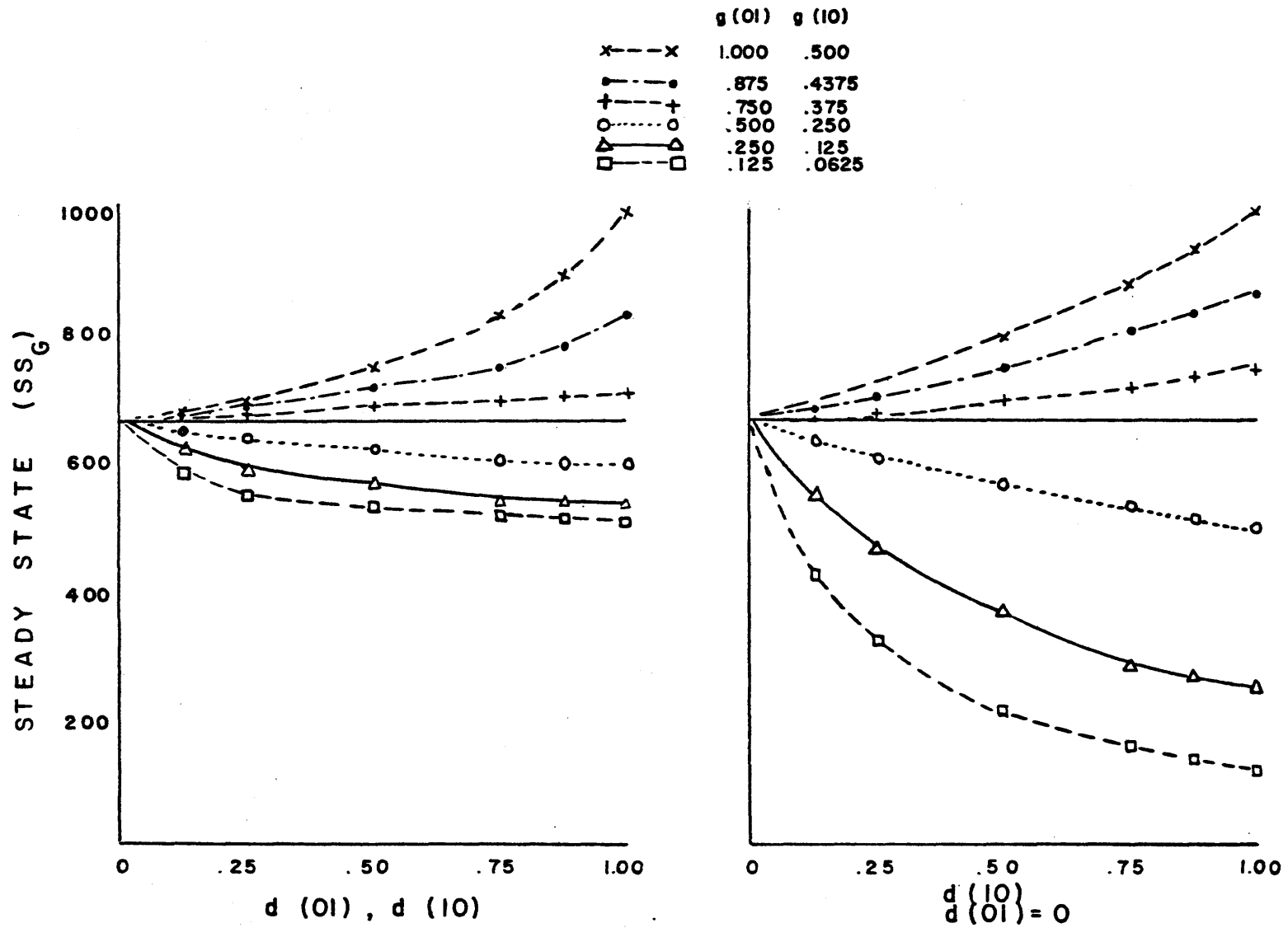


FIG 3 Steady state of W as effected by symmetrical (neutral) vs one-sided (malign) Ds.





## Problem 2: Information Lag

The preceding section was concerned with the general importance of precision of adjustment. In this section, a special problem of information loss or degradation will be examined. The particular form studied is the loss due to the condition that information cannot be applied immediately. Thus a picture of the environment that is veridical at the time it is obtained will have "staled" by the time it is used. This is not a difficulty unique to group performance processes, but it is of increased severity in that situation. The fact that group members have to relay and perhaps collate information before it is used imposes an inescapable lag on the application process.

To indicate the theoretical treatment of this topic it is sufficient to consider a one-interval lag. This means that at epoch  $t + 1$  the G agent is using a representation of the W vector that was correct at epoch  $t$ . This in turn implies that G's intended action will still be appropriate only for those elements that have remained the same. The effect of G's action on elements which are in a different state than that in which he "believes" them to be depends on the mechanics of the operator. For the moment we assume that his effect will be exactly opposite of that intended. (See the program notes in Appendix A for a description of alternative operator mechanics.)

The symbolic treatment is most conveniently shown by a numerical example:

$$\begin{aligned} \text{Suppose } D &= \begin{pmatrix} 3/4 & 1/4 \\ 1/4 & 3/4 \end{pmatrix} \text{ and } G = \begin{pmatrix} 3/4 & 1/4 \\ 1/2 & 1/2 \end{pmatrix} \\ \text{so that } DG &= \begin{pmatrix} 11/16 & 5/16 \\ 9/16 & 7/16 \end{pmatrix} \text{ and } GD = \begin{pmatrix} 10/16 & 6/16 \\ 8/16 & 8/16 \end{pmatrix} \end{aligned}$$

and the steady-state (with no lag) is (643, 357). The latter vector then, describes the state of the environment in gross terms after D and G have

each completed a large number of successive turns, say at epoch  $t$ .

Suppose that, at this time, a lag is introduced into  $G^S$  orientation system. This means that  $G$  records, at epoch  $\underline{t}$ , the exact identity of 0 and 1 elements immediately after  $D^S$  move, but this record is not available for use until  $G^S$  move at epoch  $\underline{t+1}$ .

At epoch  $\underline{t}$ ,  $G$  must use the state description that in fact characterized conditions immediately after  $D^S$  move at epoch  $\underline{t-1}$ . This is a vector of the form (571,429) describing the steady-state for the operator  $GD$ . However a number of the specific elements will have been changed by the interviewing  $GD$  cycle.

It is convenient to consider this informational vector in two parts: one consisting of elements that have remained the same and another consisting of elements that have been changed to the opposite value. The former component is estimated by multiplying the vector (571,429) by the main diagonal elements of the  $GD$  operator. Thus

$$(571,429) \begin{pmatrix} 10/16 & - \\ - & 8/16 \end{pmatrix} = (356,214)$$

and this vector indicates the elements on which  $G$  will operate appropriately in spite of the lag. The other component is obtained by multiplying (571,429) by the counter-diagonal elements of  $GD$ .

That is

$$(571,429) \begin{pmatrix} - & 6/16 \\ 8/16 & - \end{pmatrix} = (215,215)$$

$G$  will operate on these elements with a matrix in which the rows and columns are exactly reversed from the normal matrix.

This means that  $G^S$  output vector, at epoch  $\underline{t+1}$  is given by

$$\begin{aligned} & (356,214) \begin{pmatrix} 3/4 & 1/4 \\ 1/2 & 1/2 \end{pmatrix} + (215,215) \begin{pmatrix} 1/2 & 1/2 \\ 1/4 & 3/4 \end{pmatrix} \\ & = (374,196) + (161,269) = (535,465) \end{aligned}$$

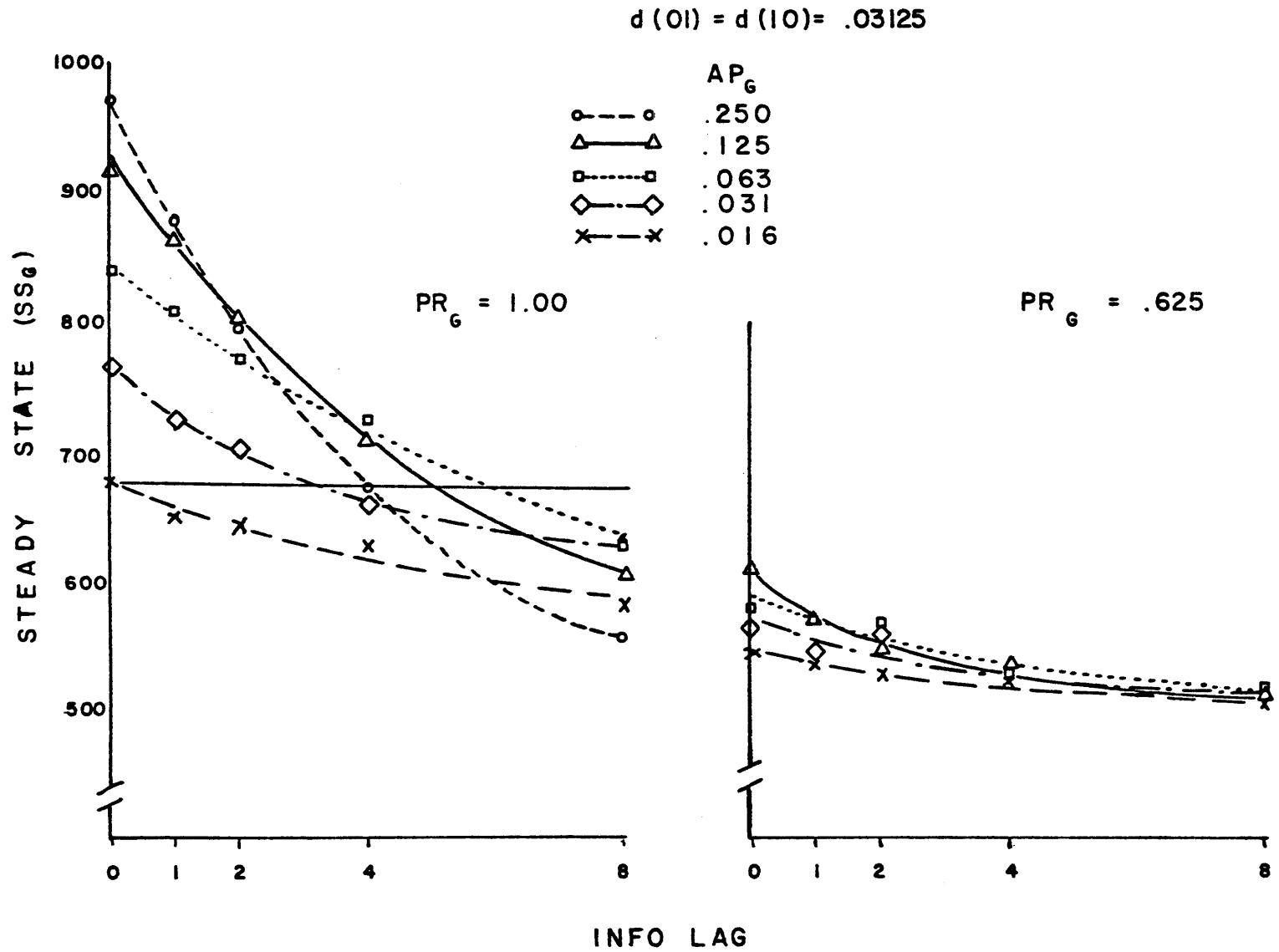
Notice that this is much below the no-lag steady-state of (643,357) after G operation.

In order to compute the new steady-state it is necessary to take into account the changed effect of the operator GD due to  $G^S$  information lag. This will not be carried out here as it is rather intricate, and does not add much to an intuitive grasp of the problem.

Perhaps the most interesting problem in connection with this topic concerns the trade-off between information lag and activity and precision of a G operator when opposed by a specific D. Figure 4 shows, for example that with  $d(01) = d(10) = .031$ , and  $PR_G = 1.00$ , a G with AP of .063 and lag of 6 is about as efficient as a G with AP of .125 and lag of 5, or AP of .250 and lag of 4, or AP of .031 and lag of 3, or AP of .016 and lag of 0. From a set of graphs such as Figure 4, one may construct graphs representing trade-offs between information lag and precision or activity of G when operating against specific Ds.

As can be seen from Figure 4, the magnitude of the effect of information lag is a function of both the precision and activity of G. Low precision Gs are relatively ineffective even with up to date information, and high activity potential (4b). However, given a high precision level, more active Gs are more effective when using up-to-date information, but may become less effective than relatively inactive ones, when operating with old information (4a).

The intuitively compelling notion that stale information will be more useful in a relatively quiescent system than in an active one is also borne out by Figure 5. Here steady-state is shown as a function of information lag with D and G operators matched with respect to activity potential. It is apparent from the figure that the more active system (circles) are more adversely affected by information lag than are the less active ones (triangles).



**FIG 4** Effectiveness of a set of  $G_s$  operating with varying information lags.

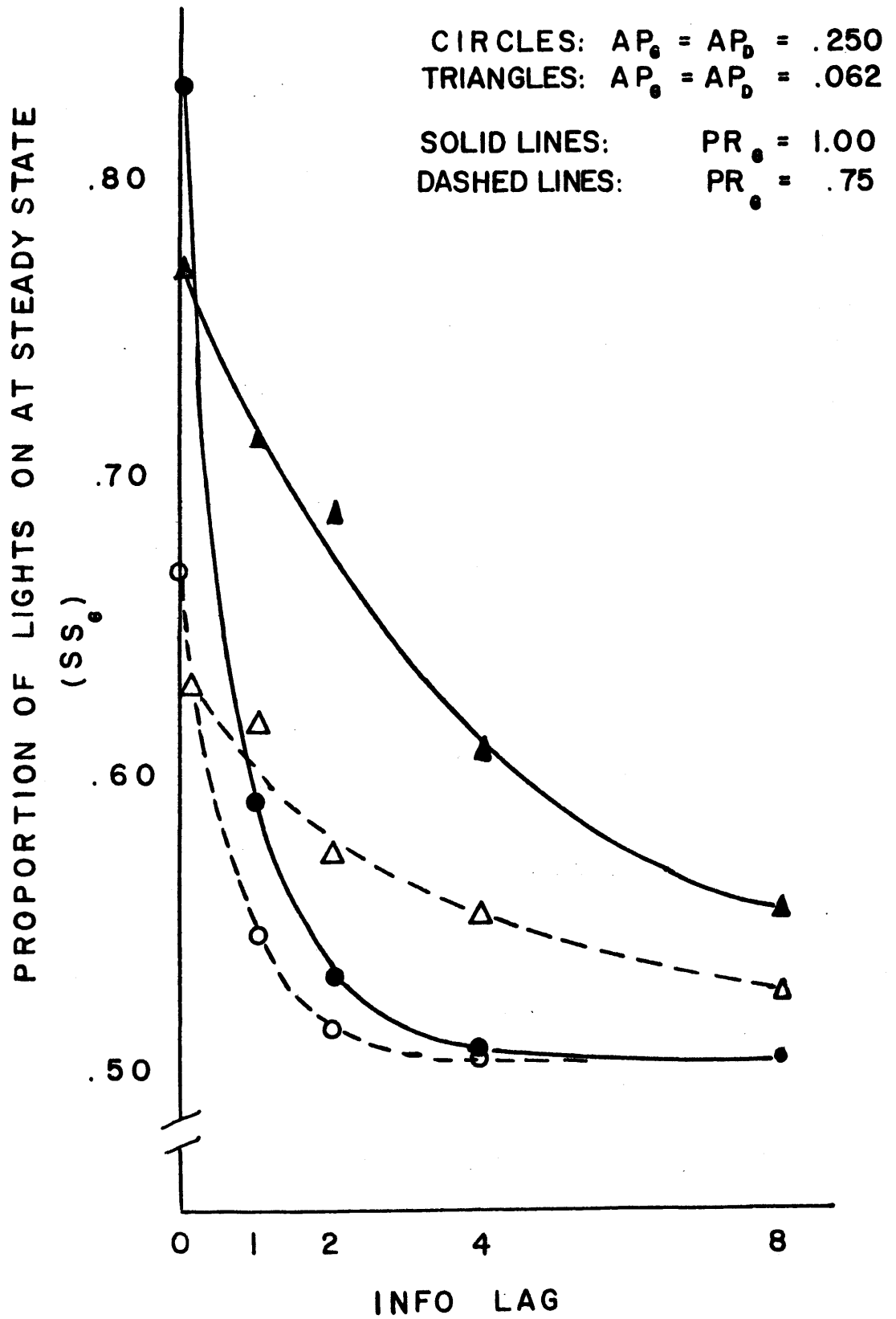


FIG 5 Effect of information lag with G and D matched with respect to activity potential.

It is also obvious that the positive relationship between steady-state and the activity potential of G observed with zero lag becomes negative with non-zero lag. Differences due to precision ratios of G are maximum with 0 lags and decrease steadily with increasing lags; however information several epochs old may be quite useful to a high precision operator in a relatively inactive environment.

Figure 6 shows the relative effects of symmetrical, i. e. , "neutral" and one-sided, i. e. , "malign" Ds with varying lags and a given G. The information lag, of course, still applies to the G operator. D always works with 0 lag. As would be expected in this situation, a one-sided D is more disruptive than a corresponding symmetrical D. In addition, it can be seen that, whereas with symmetrical Ds, all DG combinations converge to the same steady-state with increasing lags, with one-sided Ds the system settles to different levels.

### Problem 3: Segmented Operators

Preceding sections have treated the group as a unitary organism. Although the problems considered are particularly severe in group operations, this might easily pertain to the performance of an individual agent. This section will examine a problem that applies uniquely to group performance -- that of differential assignment of personnel.

The physical situation visualized here differs from the one formerly considered in that the warehouse is now divided into a number of compartments, each infested by a distinct 'sub-demon',  $D_i$ . The various  $D_i$  may have different operator characteristics. Similarly, G is divided into a set of group members,  $G_i$ , with differing operator characteristics. The  $G_i$  may be assigned to compartments according to various policies as described below.

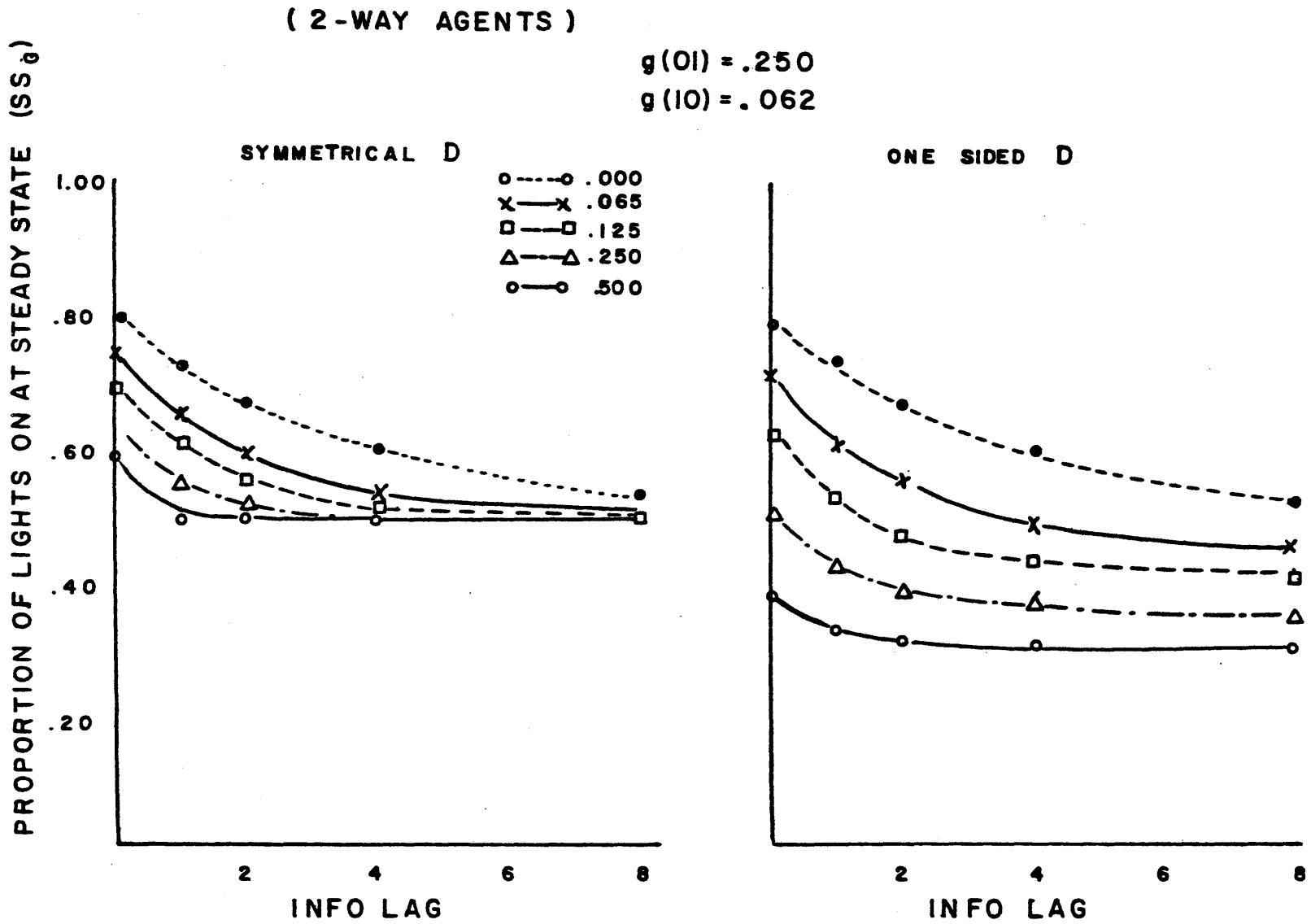


FIG 6 Effect of information lag with symmetrical vs one-sided D and constant G.

The general principle to be investigated is that, with a given set of  $G_1$  and  $D_1$ , the more closely the  $G_1$  are matched to the  $D_1$  the better will be the overall performance of the group. The basic notion is that, under poor matching, ineffective\*  $G_1$  will not be able to cope with highly disruptive  $D_1$ , while more effective  $G_1$  will be wasted upon a comparatively easy assignment. A very simple example will illustrate the general effects of matching.

Let

and

$$D_1 = \begin{pmatrix} .80 & .20 \\ .00 & 1.00 \end{pmatrix} \quad D_2 = \begin{pmatrix} .90 & .10 \\ .00 & 1.00 \end{pmatrix}$$

Here  $D_1$  is twice as disruptive as  $D_2$ . Similarly, let

$$G_1 = \begin{pmatrix} 1.00 & .00 \\ .40 & .60 \end{pmatrix} \quad \text{and} \quad G_2 = \begin{pmatrix} 1.00 & .00 \\ .20 & .80 \end{pmatrix}$$

Both these operators have precision ratios of 1.0, but  $G_1$  is twice as active.

Suppose, first, that  $G_1$  is matched against  $D_1$  and  $G_2$  is matched against  $D_2$ . This leads to the two compound operators,

$$D_1G_1 = \begin{pmatrix} .88 & .12 \\ .40 & .60 \end{pmatrix} \quad \text{and} \quad D_2G_2 = \begin{pmatrix} .92 & .08 \\ .20 & .80 \end{pmatrix}$$

The steady-state criterion measure with this assignment can be computed by averaging the steady state of the two separate operators; with 1,000 lights it is (750,250).

Now, suppose that  $G_1$  is matched with  $D_2$  and  $G_2$  with  $D_1$ . This yields the two operators

$$D_1G_2 = \begin{pmatrix} .84 & .16 \\ .20 & .80 \end{pmatrix} \quad \text{and} \quad D_2G_1 = \begin{pmatrix} .94 & .06 \\ .40 & .60 \end{pmatrix}$$

\*Effectiveness" is used here as a rather loose synonym for "more precise and/or more active". As already noted, precision and activity do not permit a simple, unequivocal ordering of  $G$  agents, as one condition may be relatively more or less important than the other, depending on  $D$ . To avoid ambiguity in this section,  $G_i$  agents are varied along one of these axes only.



Here the steady-state component produced by  $D_2G_1$  is very high, but that produced by  $D_1G_2$  is quite low. The average is (732,268) appreciably poorer than for the matched assignment.

A number of assignment procedures are theoretically possible, and may occur in real world situations. At one extreme there is random matching in which a  $G_1$  agent may be assigned to any  $D_1$ , either permanently or epoch-by-epoch. An intermediate assignment procedure is one in which the  $G_1$  are permanently assigned to compartments on the basis of the long-run average conditions in these compartments. An even more exact assignment would take into account the conditions at each epoch and reassign  $G_1$  on the basis of the momentary state of each compartment. The feasibility of these assignments procedures in practice depends on the information available to a central executive agency, and the difficulty of transporting operators.

At the present time data on the segmented operators problem is just beginning to be collected, and hence is not included in this report.

## DISCUSSION

The foregoing results are intended to give a picture of the problems investigated to date and the approach that has been taken to those problems. A few examples may serve to indicate the general directions in which these studies will be extended.

The "information lag" section assumed only that there was some fixed delay between the initial recording of a state description of the warehouse and the ultimate application of that description. The next step is to translate this effect into terms that are more descriptive of actual group behavior.

Specifically, it will be assumed that there are distinct "sensor" and "effector" agents in the group, and that some transmission delay occurs between these agents because of limited communication. Thus, it may be the case that each sensor reports to a central agent who in turn relays the information to the effectors. Alternatively, sensors may contact the effectors directly in a certain priority order and transmit the current information for that epoch. The suggested investigation will examine a number of such procedures, probably with an eye to determining the optimal transmission procedure for a given set of communication constraints.

A second line of further investigation would carry the "division of labor" problem several steps further. The present study examined the effects of compartmentalizing operators, but still treated the attributes as homogeneous -- for example, a given  $G_i$  operator is capable of turning any light on or off. A more complex (and also more realistic) condition is one in which the several distinct  $G$  matrices are differentially effective vis a vis various environmental attributes. A still more complex situation is one in which several  $G_i$  operators must act jointly in order to turn a light on. Studies incorporating such conditions would build up from the compartmentalization results now being collected.

In discussing the parallel analytic-simulation approach employed in this report, the question was raised whether the simulation approach is necessary. That is, would it not be possible to program a computer to obtain results more directly (and without sampling error) by numerical substitution in the relevant matrix formulae? Several considerations suggest that the Monte-Carlo treatment will be the more fruitful one in the long run.

First, it appears that the simulation programs can be built up piecemeal, either adding sub-routines or making minor substitutions. As opposed to this, the more analytical treatment seems to require a rather radical overhauling as each new stage of complexity is encountered. The danger of inadvertently building in unjustified assumptions is considerably greater in the latter case than for the more gradual and continuous development.

A perhaps more fundamental advantage lies in the highly operationalistic nature of the Monte-Carlo simulation. Processes are depicted by a very literal representation of possible system behavior and the criterial performance measure is available in terms of actual state conditions at any time rather than as abstract numerical indices. Thus, it should be easier to interpret surprising results in the simulation studies, and to relate the functional relations to real world systems of interest. It should be added that the two approaches are in no way mutually incompatible and the intention is to retain both of them if possible.

With regard to the simulation effort, there are two major problems of "metaprogramming" that need to be considered: first, a decision policy for accepting an estimate and for terminating the run; and second, a strategy for exploring the various parameters in a functional relationship. Considering first the stopping policy, the present program is based on a fixed number of runs for each condition. A periodic print-out of the means and standard deviations insures that the N is sufficient and that the estimates are reasonably reliable. In addition, comparatively complete coverage of the values builds in enough redundancy so that no gross discrepancies are likely. This procedure is adequate for exploratory studies but rather uneconomical for a more comprehensive investigation.

An ideal scheme would employ a cut-off decision function based on some sequential testing principles. A fixed standard error of estimate or a fiducial tolerance would be specified and the computer would stop or move to a new problem when the estimates were within the specified tolerance limits. In the present case there are possible hazards in such a procedure. For example, it appears that although very active systems tend to reach an asymptotic value more rapidly than less active systems, the variance about the asymptotic value will tend to be greater for the active systems. This suggests that parametric statistics may not be entirely appropriate here -- a variance that would be reasonable for a low activity system may be too stringent for a high activity system.

The general procedure that looks most attractive at present is some form of sign test. This would be based on the assumption that if the  $W$  vector is at a steady-state, there will tend to be an equal number of data points above and below the mean. Therefore, a program that takes the difference between successive data points and stops when the increases and decreases are equal should produce a reasonably good estimate of the true steady-state. Even this approach may run into trouble at extreme values of the steady-state vector if the distributions tend to be skewed at extreme values. It is not clear at this point how serious this difficulty may be.

A second general problem is one of exploring the range of values in some optimal way. The problem is acute because of the fantastic number of possible combinations that are generated by even a few system parameters.

There are two aspects of this problem. One is the necessity of guaranteeing that the exploration is thorough enough so that the important functional relationships have been studied. In the interest of economy this implies a scheme in which the variance attaching to changes in various parameters is

maximized over the choice of parameters. Opposed to this, however, is the necessity for presenting the results in an orderly way and making interpretative sense out of them. One can imagine a "steepest descent" program, or something of the sort, that explores directions at maximum variation, but hops around so erratically that the results, however comprehensive, are incomprehensible.

Both the problems of setting acceptance levels on estimates, and of exploring parameters, are now under investigation. Presumably, we should be able to draw on the simulation procedures in other areas as well as on theoretical considerations and on our cumulative experience in this study. At the same time, it appears that the problems mentioned here should be solved in some satisfactory way before it is feasible to proceed to more complex substantive studies.

#### REFERENCES

1. Roby, T.B. & Forgays, D.G. A problem solving model for analysis of communications in B-29 crews. San Antonio, Texas: Human Resources Research Center, Lackland AFB, Texas, August 1953, Research Bulletin 53-30.
2. \_\_\_\_\_ & Lanzetta, J. T. Work group structure, communication and group performance. *Sociometry* 1956, 2, 105-113.
3. \_\_\_\_\_ & Lanzetta, J. T. Considerations in the analysis of group tasks. *Psychol. Bull.*, 1958, 55, 88-101.
4. \_\_\_\_\_ Contributions to a theory of group performance, Institute for Psychological Research, Report No. NR 170-408, Contract 494(15) Tufts University, Medford, Mass., 1960.

## APPENDIX A

### Program Notes -

The work reported here represents the initial phase of what will hopefully be a continuing effort to simulate various aspects of group behavior. A series of revisions of the model is anticipated, each expanding on, and incorporating the better features of, its predecessor. The primary purpose of each new version will be to increase both the generality of the model and the degree of correspondence between it and the phenomena it is intended to describe. Initial versions will necessarily lack generality and will be more illustrative of an approach than descriptive of phenomena.

Currently the first two versions of the model are programmed. They will be referred to here simply as W1 and W2. Both are programmed in DECAL for the PDP-1 single core, no-special-devices, machine.

W1 was designed specifically to treat the first two substantive problems described in this report. The major components of relevance to the model are a 1000 element environment, W, (1000-light warehouse) and the two action agents D and G, each characterized by variable precision ratios and activity potentials (as discussed in more detail elsewhere in the report). An epoch is defined as that period of time during which both agents operate on the warehouse in the order D, G. Program mechanisms of W1 will not be described in any detail; however, excluding bookkeeping and statistical computations the major events of a single epoch are as follows: each agent operates on each element of W in the sense that it first determines the current state (off-on) of an element, then effectively tosses a coin with a bias corresponding to the particular probabilities associated with its action alternatives and takes the action indicated by the outcome of the toss. The individual

treatment of each element is to be distinguished from the quite different (though at this point functionally equivalent) procedure of treating *W* in the aggregate, determining how many elements are in each state and reversing the state of the appropriate proportion of elements.

A principle feature of *W1* is the facility for introducing an information lag into the sensor-effector communication link. In effect, an agent senses *W* at each epoch, but the information obtained at epoch *t* may be applied as many as 8 epochs later. The obvious consequence of the application of old information in this context is that the agent will at times be operating on an off element which it thinks is on and vice versa. What happens in this situation depends on the mechanisms of the agent. Two possibilities suggest themselves as representative of "real" situations.

With a two-way operator the decision to take action will result in a change of state for the element. So, for example, if the operator thinks an element is off but it is actually on, the decision to turn it on will actually turn it off. This is the effect that one would expect when the change from off to on and from on to off is accomplished by the same maneuver, as say, pulling a light cord.

With a one-way operator the decision to take action will result in a change of state for the element only if the believed state and the actual state correspond. This is analogous to the situation in which the state of the element is controlled by a bipolar switch. If the operator believes the switch is down and it is really up, the attempts to push it up will have no effect.

With both types of operators the decision to leave an element as it is does just that, irrespective of whether the believed and actual states correspond. The program mechanisms of the operator is shown in Figure **A1** and **A2**.

The data included in this report were generated with two-way operators. The relative merits of the two types of operators is left for future consideration.

W2 elaborates W1 in several ways. The environment, W, is partitioned into 8 regions, each occupied by a different D. G is now a group in the sense that it includes several agents. Specifically, 8 Ss (sensors) 8 Es (effectors) and one M (decision maker).

Each individual agent operates as an individual and may or may not have the same characteristics as other agents of the same type. The interaction of Ss and Es is as follows: S observes a region of W and reports on the state of each element. The E which is assigned to the same region takes action on the basis of S's report. Ss vary with respect to the reliability of their reports, and unreliable reports may be biased in either direction. E's vary with respect to precision and activity potential. (With a completely reliable S, E is equivalent to G of W1 except for its restricted area of responsibility). M's function is the assignment of Ss and Es to regions of W on the basis of a variety of decision rules.

The program structure of W2 is provided at a fairly gross level in a set of flow charts below. The structure is intended to facilitate development and addition of a variety of executive routines implementing different decision rules for M.



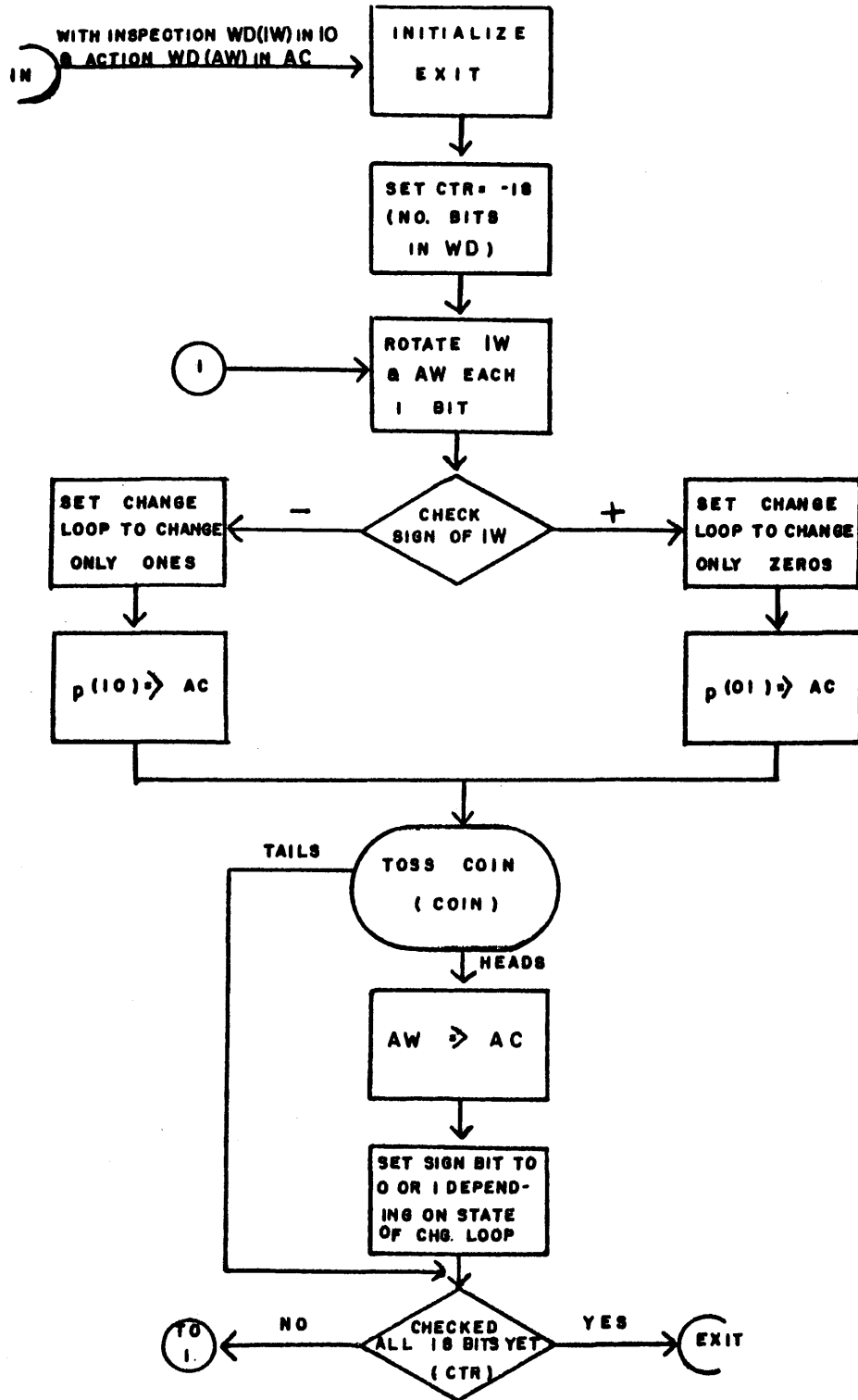
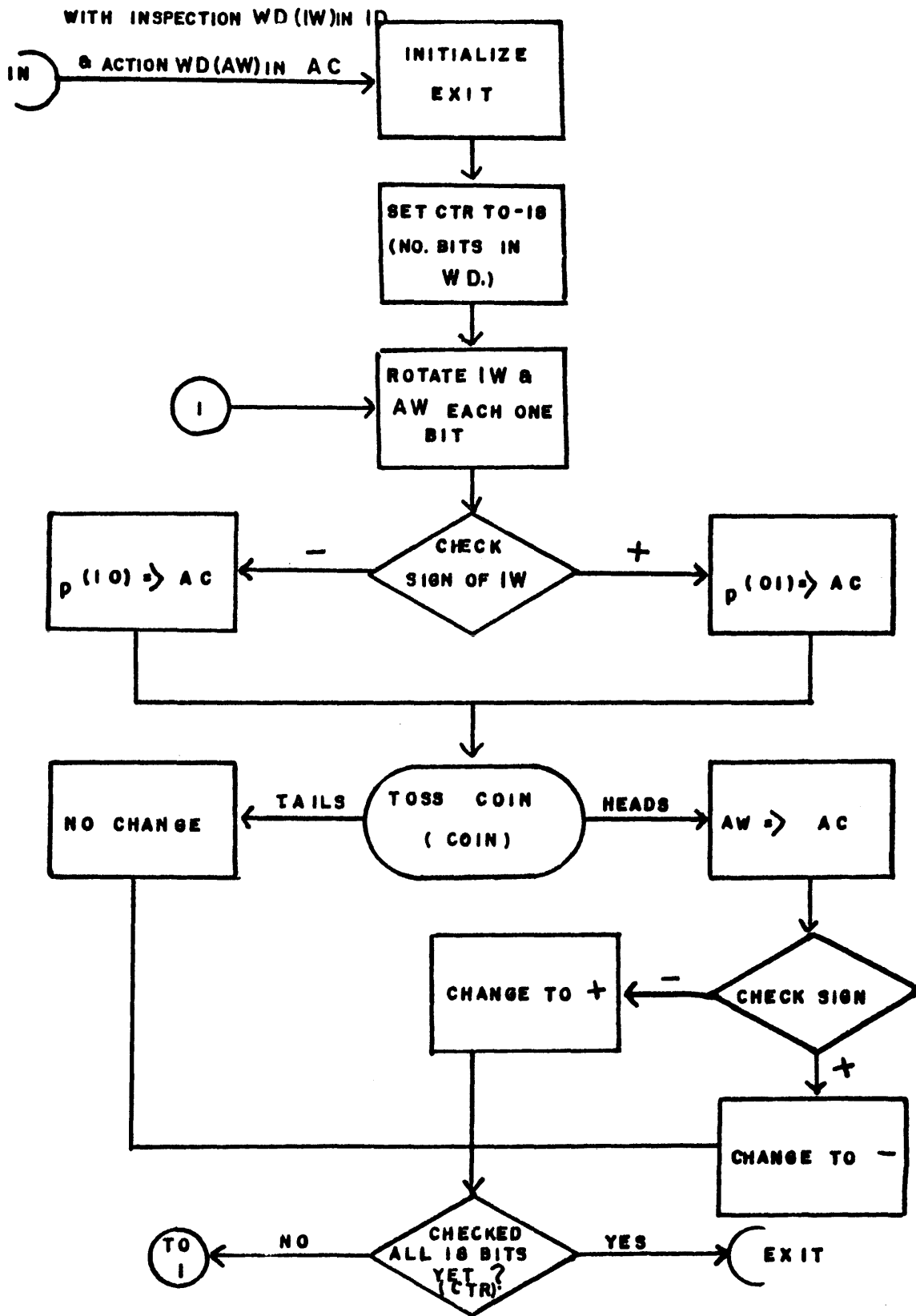


FIG ONE-WAY OPERATOR



OPERATION OR  
SET OF OPERAT-  
IONS PERFORMED  
WITHIN PROGRAM

OPERATION  
OR SET OF OPER-  
ATIONS PERFORMED  
BY SUBROUTINE

CHOICE  
POINT

INPUT,  
OUTPUT

**FLOW DIAGRAM LEGEND**

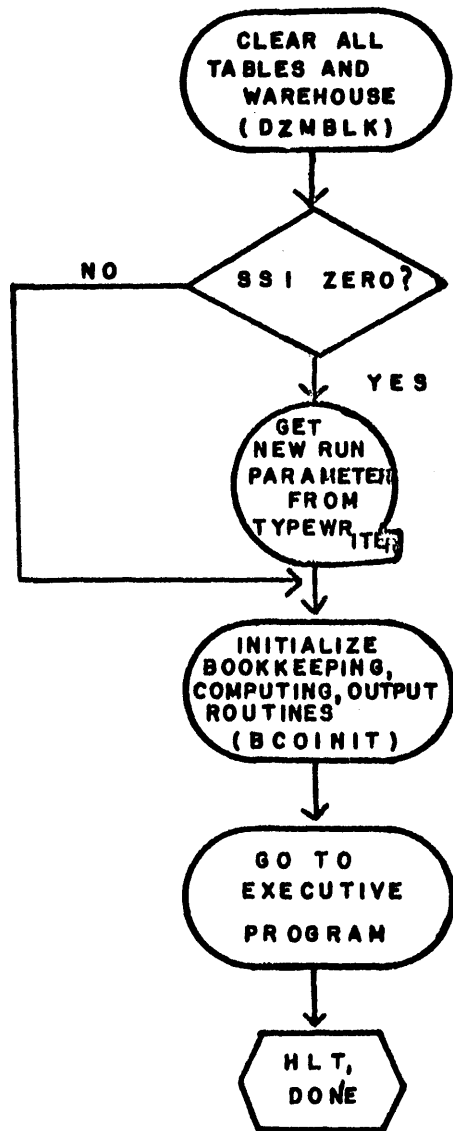


FIG. W2, MASTER  
A4

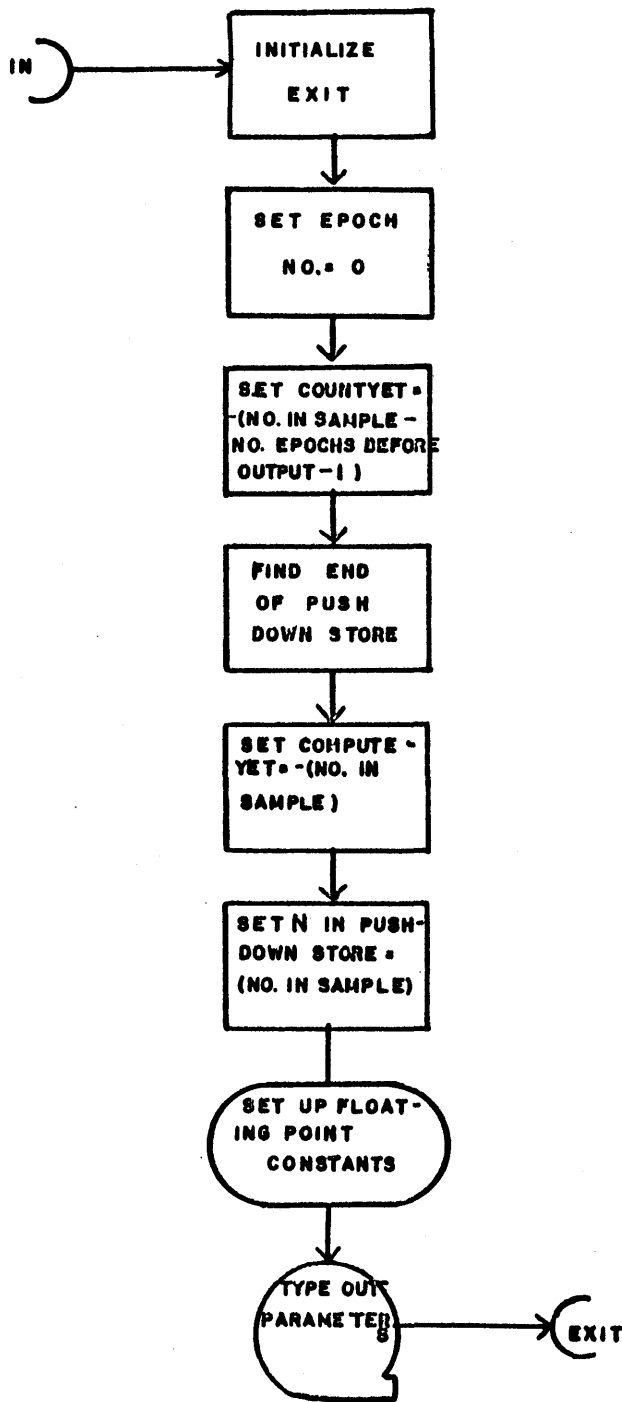


FIG PROGRAM FOR INITIALIZING BOOKKEEPING, COMPUTING AND OUTPUT ROUTINES  
A5 (BCOINIT)

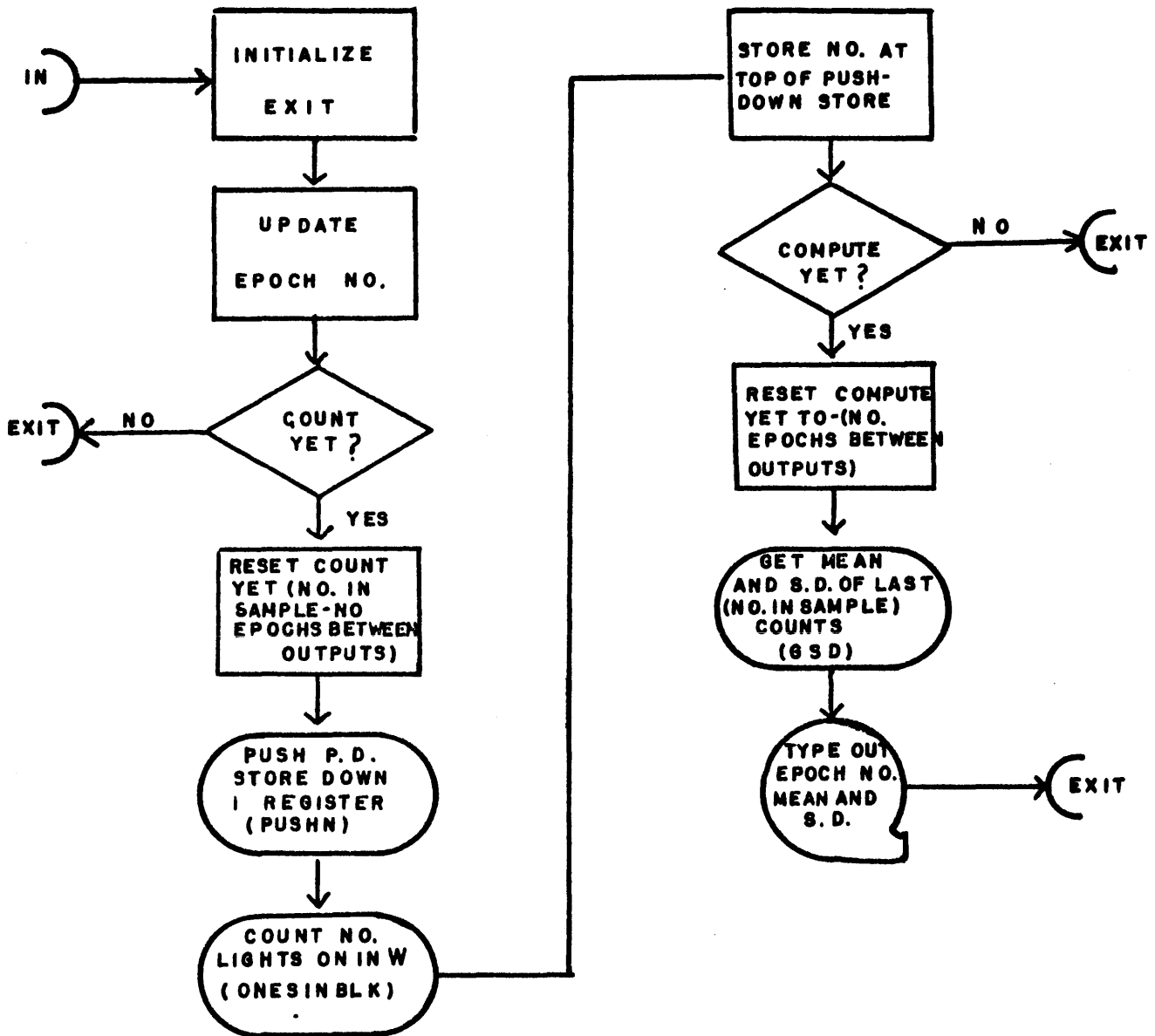


FIG BOOKKEEPING, COMPUTING, OUTPUT (BCO)

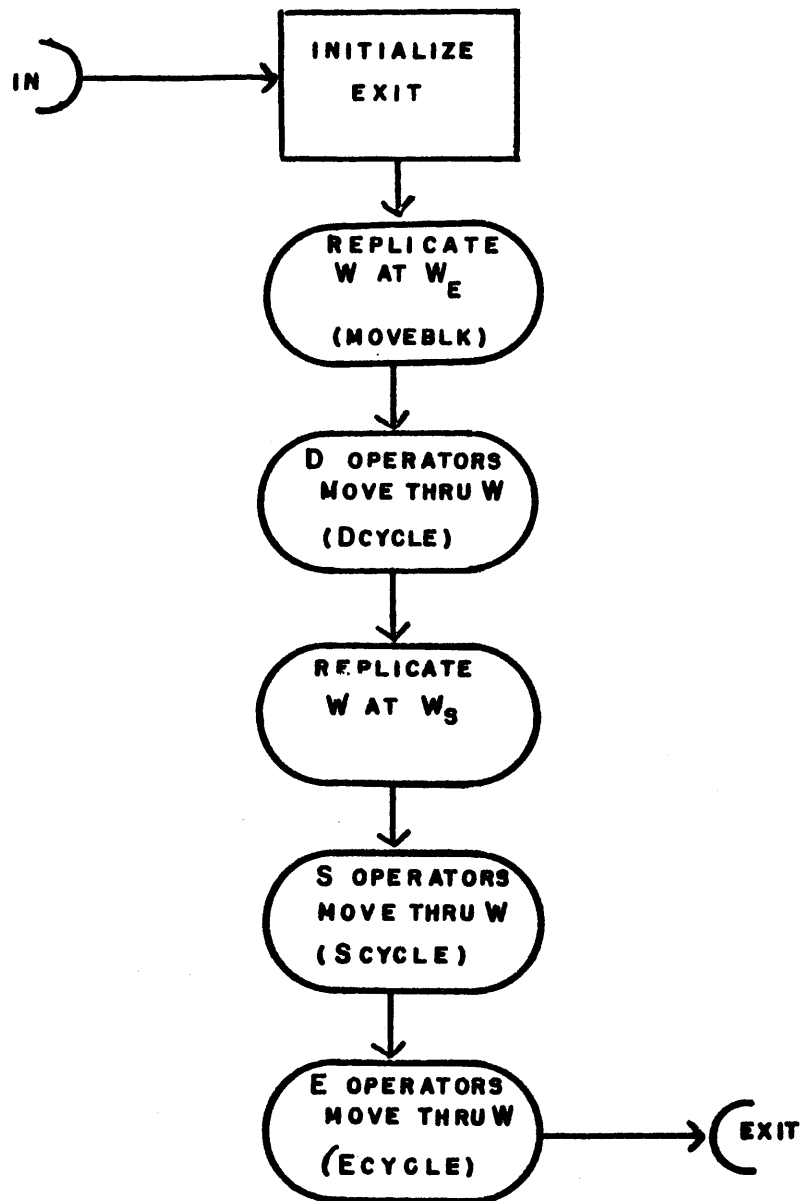


FIG PROGRAM FOR OPERATOR SCHEDULING  
A7

(PASS)

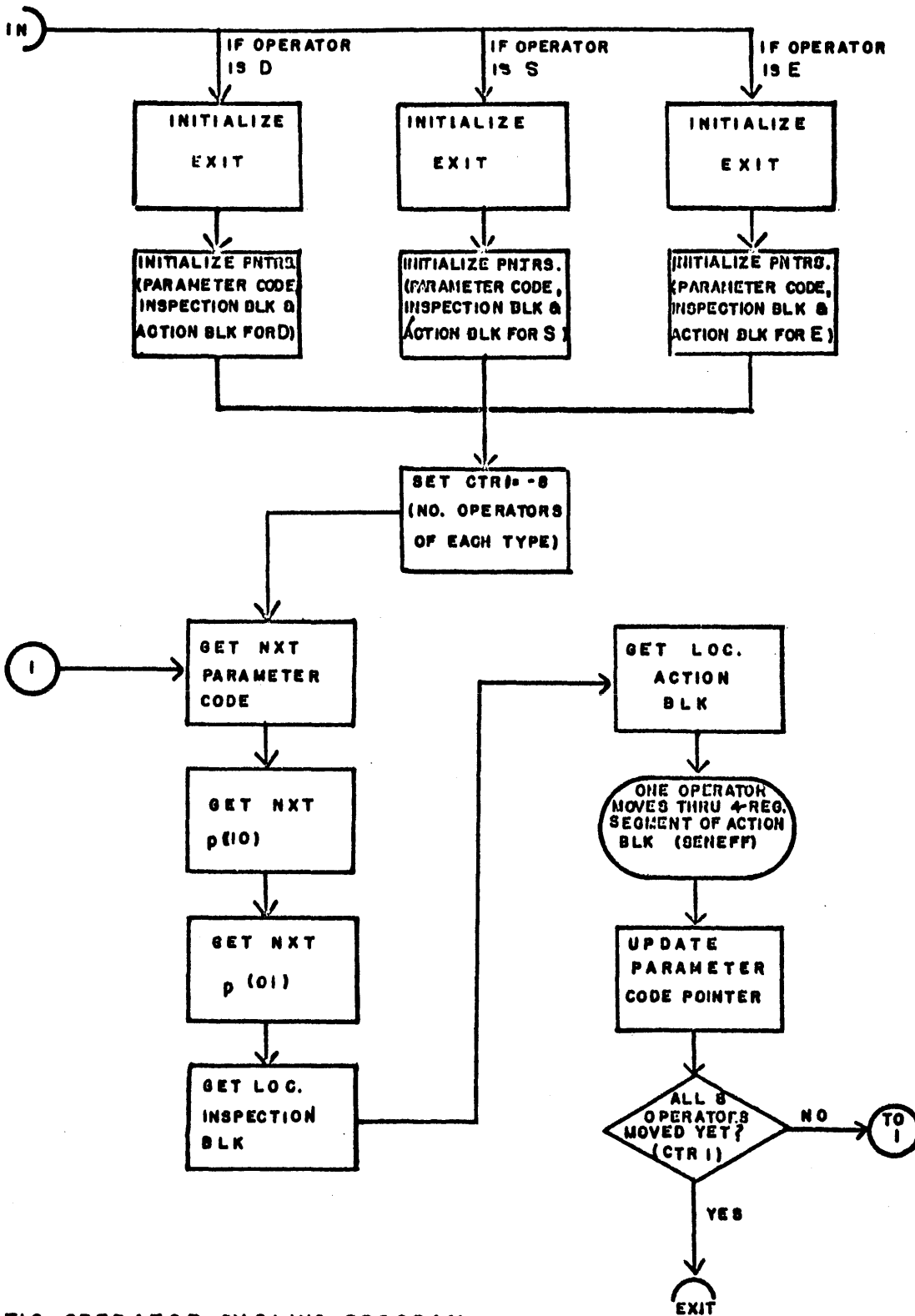


FIG OPERATOR CYCLING PROGRAM  
A8

(DCYCLE, SCYCLE, EDCYCLE)



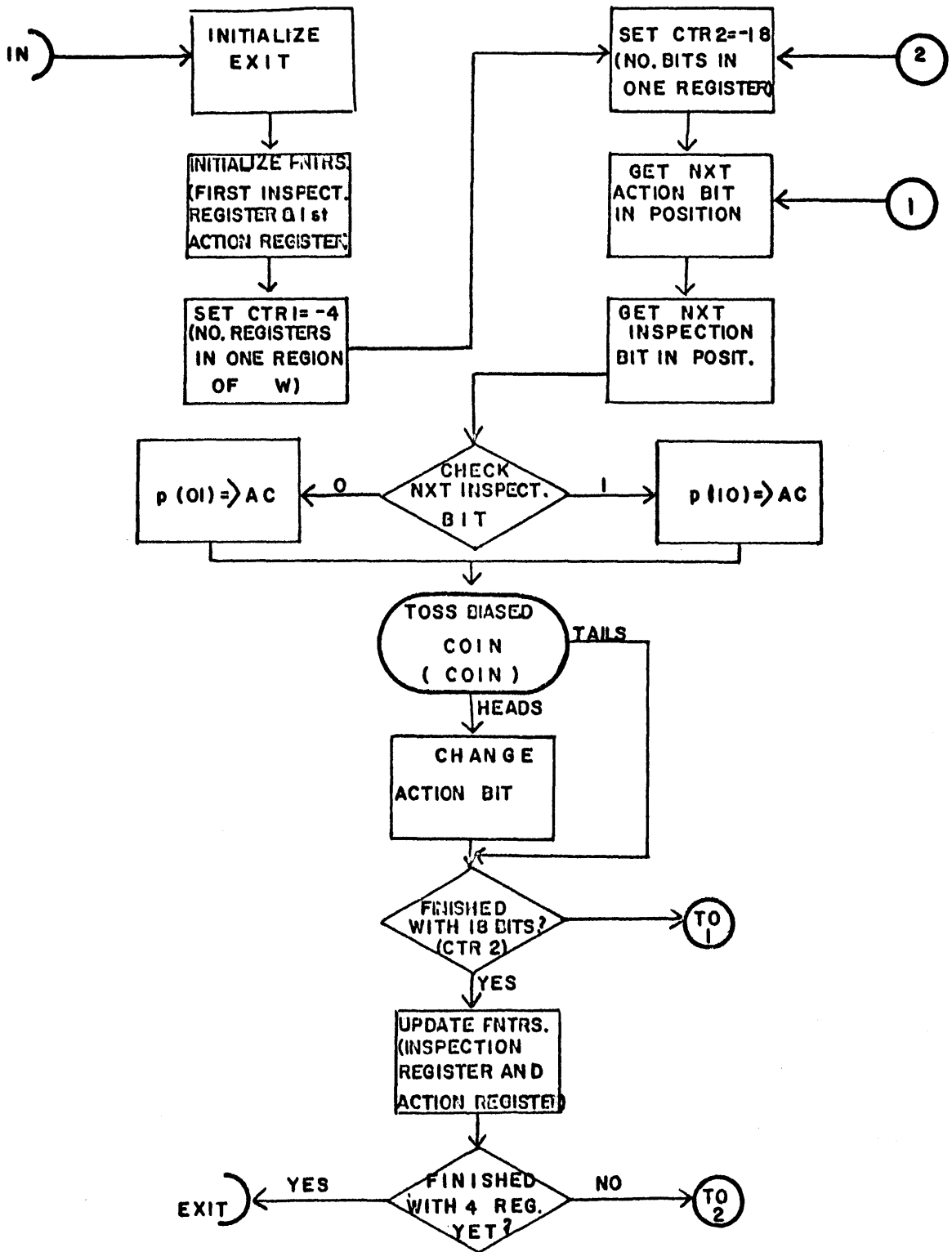


FIG. OPERATOR (SENEFF)  
A9



# A HYBRID PDP-1 SYSTEM FOR SPEECH RESEARCH

Douglas L. Hogan and Robert J. Scott

Department of Defense  
Washington, D. C.

## ABSTRACT

A hybrid computer composed of both analog and digital devices is discussed in this paper. We have combined in one system the advantages inherent in an analog system with those inherent in a digital system.

Speech, being analog in nature and essentially band-limited, is well-suited to analysis by a hybrid processor. Input to the processor is usually a voltage analog of the acoustic signal. An analysis of the power spectrum is probably the most used investigative procedure. Filtering techniques are used in spectrum analysis of the input signals by analog computation. Digital computation involves discrete time-sampling of the signal. The original signal can conveniently be stored in digital form in the system with very small quantizing error by using a high sampling rate and high precision digital coding.

The system is a basic PDP-1 with a 16 channel sequence break and three high speed data channels and controls. The analog computer is a set of PACE TR-10's with an interface to the PDP-1.

The system under PDP-1 programmed control provides a high degree of communication between the analog and digital subsystems.

The interface has 32 multiplexed analog to digital and digital to analog channels. States of 18 analog comparators may be examined by the PDP-1.

A specific application of the system to a speech research problem is considered. Vowels are synthesized by approximating the steady state vowel spectrum with a three pole transfer function. Vowel-like sounds are obtained by passing a pitch signal through the device specified by the transfer functions. Seven parameters specify the three pole locations, their bandwidths, and the fundamental pitch or excitation frequency. The vowel parameters are stored in the PDP-1 digital memory and are supplied to the TR-10 analog computers which then generate vowel transfer function. An audible output at the console allows the operator to alter the parameters to improve the speech-like quality of the synthesizer.

## INTRODUCTION

During the past few years a number of groups have demonstrated the utility of computer simulation in various aspects of speech communication research including speech analysis and synthesis and psychacoustic testing. Some groups, notably, Bell Telephone Laboratories, have resorted wholly to digital computer simulation using sampled data signal and system representations. Others have used specialized analogs of the speech generative process and controlled these generally by analog function generator techniques. Both of these techniques have limitations. Fully digital simulations run in many times real time, while analog function generators can either produce only short time control functions or require curve reading devices.

It is convenient in studying speech signals, to consider both frequency domain and time

domain representations of these signals. Frequency domain operations, such as filtering, are much more conveniently carried out in an analog computer. On the other hand, time domain operations can be handled readily using time samples in a digital computer. Control and logic decisions are also more conveniently handled with a digital computer.

### The Hybrid System

For these reasons it was felt that a combined analog and digital computing system would be a useful tool for research in speech analysis and synthesis. Since the analog computer is being used for signal processing as opposed to solving differential equations per se, and since speech signals have a dynamic range of about 50 db, an accuracy of about 0.1% or 0.2% seemed to be adequate throughout the system. The final system is shown in the block diagram of Figure 1. The analog computer is an expanded TR-10 analog computing system made by Electronic Associates, Inc. Connected to this there is an analog-digital interface and a PDP-1 computer, both made by the Digital Equipment Corporation. The PDP-1 has 16,384 words of high-speed memory a 16 channel sequence break and three high speed channels. One of these is used with a Type 510 control to operate four IBM 729-VI magnetic tape drives. Another channel controls input from the A-D section of the interface and output to a display. The remaining channel controls output to the D-A section of the interface and input from a spectrum analyzer. Thus A-D and D-A operations may take place concurrently. The display is a Data Display Type dd26A. It has both point plotting capability and automatic character generation for the ten digits and five symbols. No light pen is provided, however, there is a "track-ball" which is a ball riding on rollers mounted at right angles. These rollers drive potentiometers, thus providing X-Y coordinates under manual control. The coordinates are read into the computer through the A-D portion of the interface. Other peripheral equipment includes a line printer, card reader, and microtape.

The A-D-A interface was designed in a somewhat different manner than any of which we are aware. All control and set up is from the digital computer stored program, however, the analog computer may cause a sequence break, thus giving the analog computer a measure of control.

A total of 32 data channels are provided in each direction. Going in the analog-digital direction a multiplexer is used followed by a 10-bit A-D converter which makes the conversion in 6 microseconds. Going in the digital to analog direction 32 separate D-A converters are used with each conversion made in 2.5 microseconds and with selection still being under control of the multiplexers.

The multiplexer operates in two modes, a fixed cycle or addressed random access. Data

flow may be A-D only, D-A only, or interleaved operation. The conversion rate is set from the computer in integral multiples of the basic 5 microsecond memory cycle. A choice of 10-bit or 9-bit digital representations is available in both directions. When a 9-bit digital representation is used, the samples are stored two to a word as they go into the core memory and may be retrieved from core in the same manner.

An additional interchange of control signals is provided by two 18-bit registers. One accepts "comparator outputs" from the analog computer. Each bit represents the state of a particular control signal which the analog computer may make available. A change in this register may call for a sequence break. The second register provides control signals which can set comparators, or initiate analog process under control of the PDP-1. All 64 data channels and the two 18-bit registers are terminated at a plugboard on the console of the analog computer. This console contains trunks to the individual TR-10 computers.

To recapitulate, the entire interface is under stored program control while both computers have the capability of interrupting each other. This flexibility of control combined with a high data rate has made a useful tool for a variety of problems in speech communication research.

### Speech Producing Mechanism

The speech producing mechanism as shown in Figure (2) consists of the lungs producing the air pressure, the vocal cords modulating the flow of passing air and the oral and nasal cavities acting as resonators.

The vocal cords when vibrating produce a sequence of irregularly shaped pulses which occur as seen in figure (3) at a frequency called the fundamental pitch frequency. The shape and size of the resonators as well as the pitch frequency are controlled by muscles.

### Vocal Characteristics

The vocal tract is acting as a set of filters reinforcing certain harmonics of the fundamental pitch and suppressing others. The three main points of resonance are called the first three formants. Steady state vowel sounds are very nearly periodic. The positions of the formants are nearly invariant with respect to pitch changes for a given vowel.

The vowel chart in Figure (4) shows the relative position of the tongue for the production of the various vowel sounds. The hump of the dorsum of the tongue divides the oral cavity into two regions. The position of the hump with respect to its position back from the front of the mouth defines the abscissa while the tongue height defines the ordinate. We may now superimpose two frequency axes which relate the various vowels to their approximate formant positions. There is seen a definite relationship between the formant positions and

the sizes and shapes of the two oral cavities separated by the tongue hump. The third formant seems to have little to do with the identification of vowel sounds and is not included for that reason.

### Vowel Synthesis

If we can simulate the transfer function acting on the vocal excitation function by the vocal tract, we should be able to produce a sound similar to the acoustic signal emitted from the lips. Figure (5) illustrates the effect of this transfer function on a periodic pulse input. Because of its periodicity,  $S_i(t)$  can be represented by a Fourier Series. In the same manner  $S_o(t)$  can also be represented by a Fourier Series with the same fundamental frequency. The passive transfer function  $T(\omega)$  affects the frequency spectrum or Fourier representation of  $S_i(t)$  in some non-linear manner as shown in Figure (6) to produce  $S_o(t)$ .

As previously mentioned we can get reasonable vowel synthesis using just two poles or filter locations.

### Parameter Description

The parameters required for the synthesis of the first two formants are shown in Figure (6). For each formant we need amplitude, bandwidth, and formant center frequency. The analog computer is used to simulate two band-pass filters which together act as the transfer impedance of the vocal tract. Each of the six parameters is controlled by a voltage supplied from the digital computer through the digital to analog converter. Another parameter is the fundamental pitch frequency. A control voltage from the digital computer can vary an analog derived pitch signal or a pitch signal can be synthesized digitally.

### Program Description

The program has two parts (1) generating the parameter values and (2) vowel playback. The track-ball on the data display is used to plot the desired frequency location of each formant with respect to time. With the coordinates displayed as in Figure (7) on the scope, the track-ball is positioned to the initial first formant center frequency at  $t_0$ . Typing an "s" begins in the tracking program. The tracking program cycles through three subprograms (a) A to D which inputs track-ball positions (b) computation and storing parameter values (c) computer to data display giving coordinates and track-ball path information. The bandwidth and amplitude parameter values are kept constant for a particular vowel synthesis but can be varied if desired. The pitch is tracked in the same manner as the formants.

The track-ball is moved through the desired formant trajectory until reaching the termination position at which time as "e" is typed to stop the process. The trajectories are described in this manner for both formants and the pitch.

The D to A converter holds whatever level was given last so that only changes in parameter

values need be converted in the program. To reduce the amount of stored parameter data, we store only the voltage amplitudes when a significant change has occurred and the time duration since the last significant change. The voltages from the digital computer control two active filters synthesized on the TR-10 analog computer. The filter output is then available to the programmer as audio output through a loudspeaker.

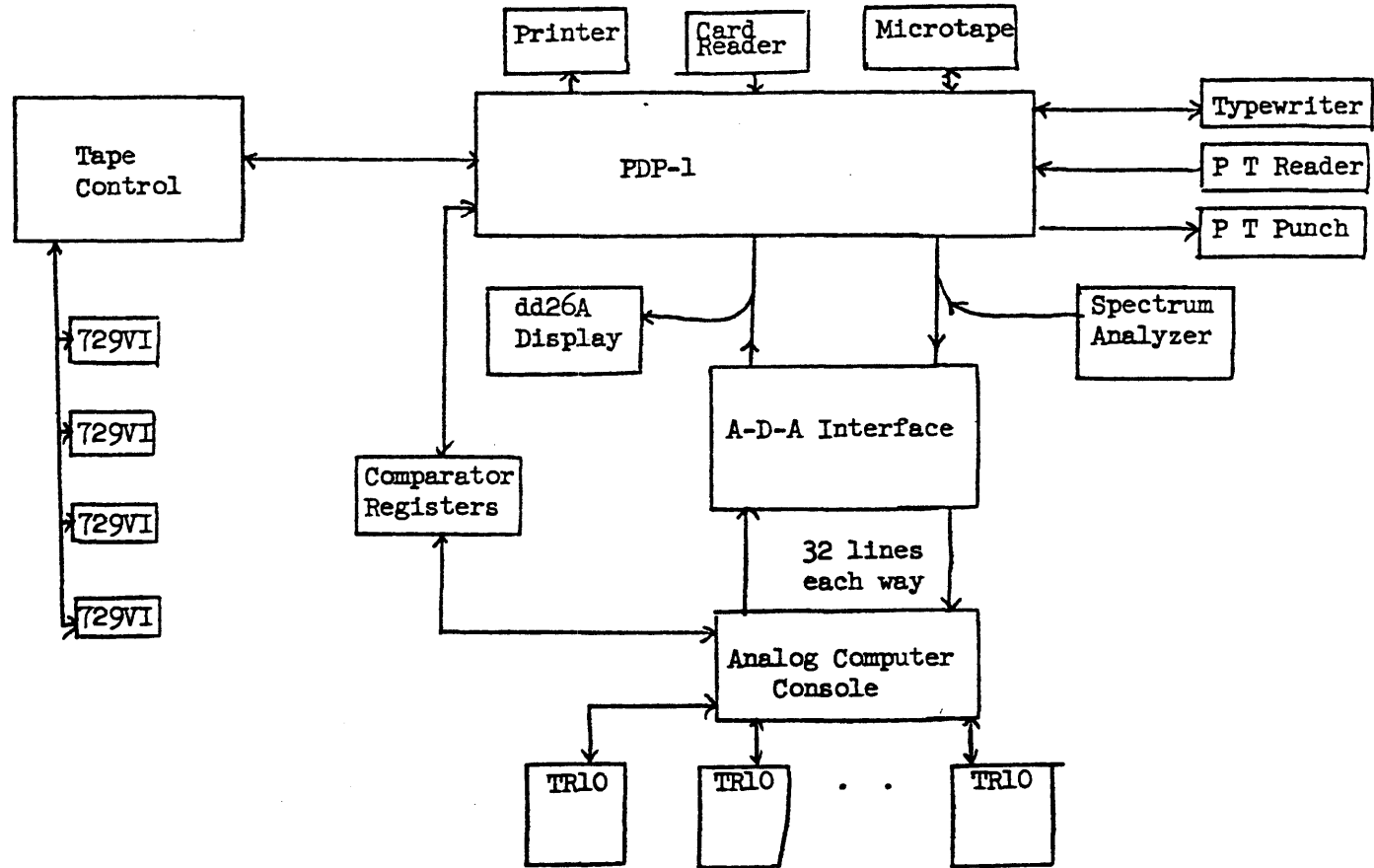


Figure 1  
Block Diagram of Final System



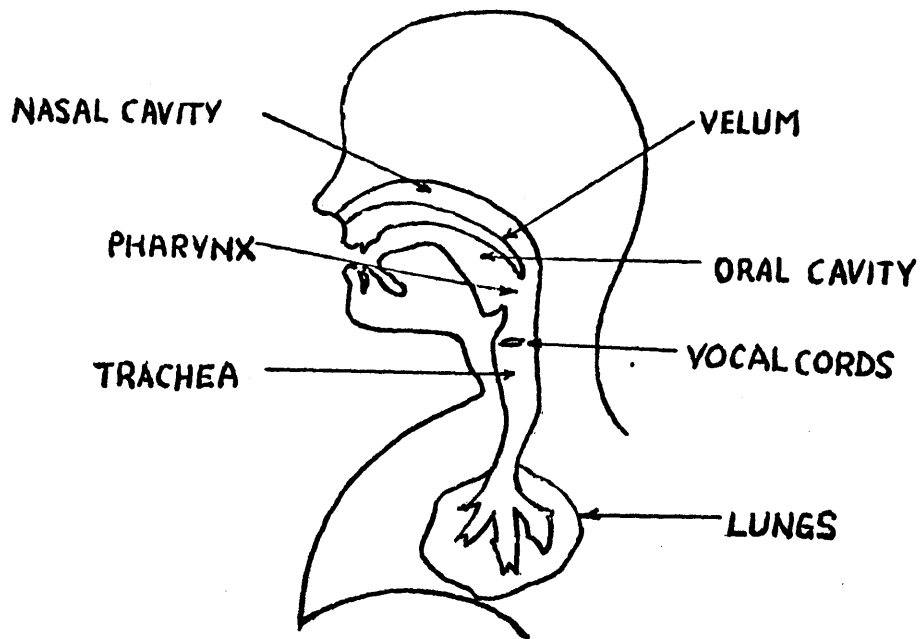


Figure 2  
Speech Producing Mechanism

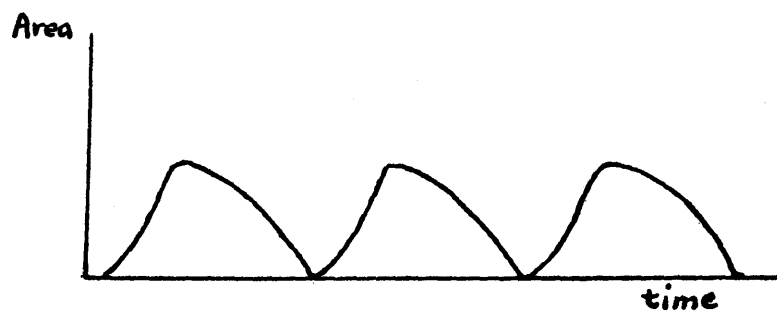


Figure 3  
Speech Pulse of Vibrating Vocal Cords

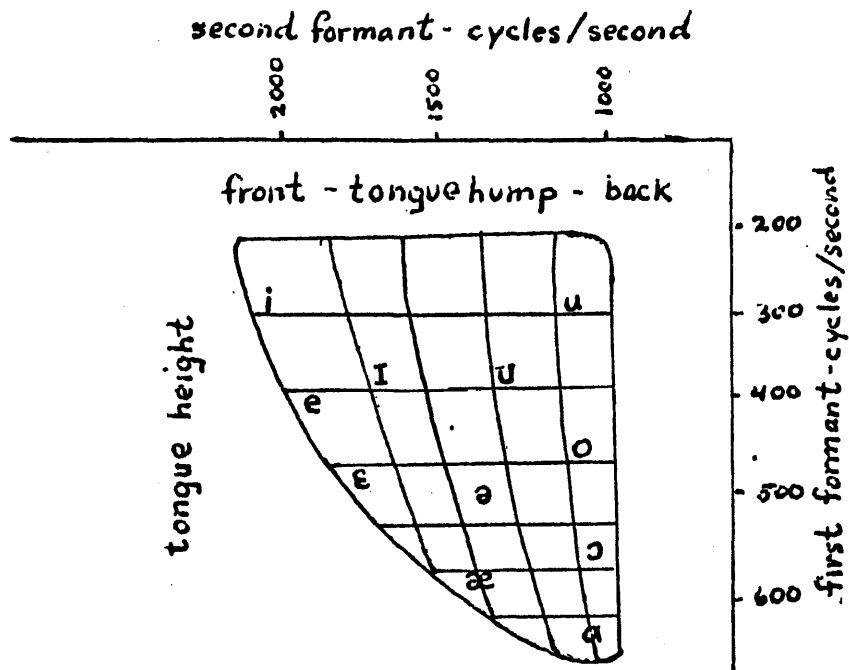


Figure 4  
Vowel Chart of Tongue Position

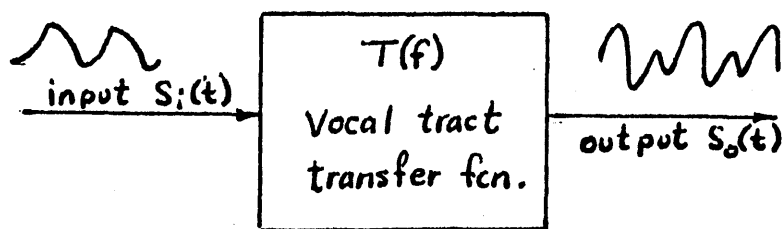


Figure 5  
Fourier Series -  $S_1(t)$

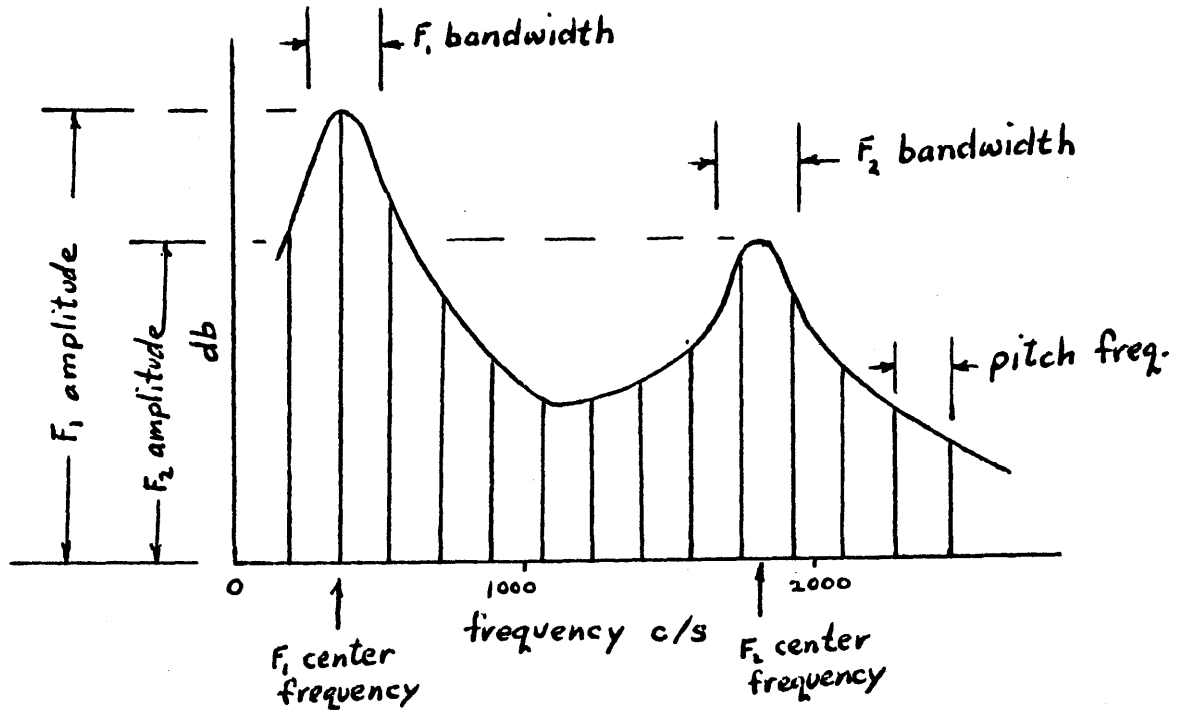


Figure 6

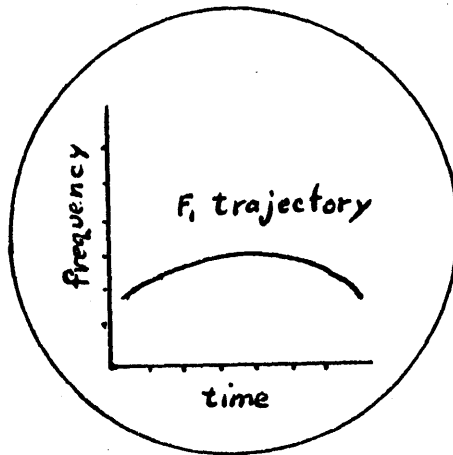


Figure 7



# COMPUTER AIDS TO NUMBER THEORY

Malcolm Pivar

Information International, Inc.  
Maynard, Massachusetts

## ABSTRACT

The on-line use of computers for general purpose number theory calculations has been greatly facilitated by the development of communication devices such as the PDP-1 typewriter, scope and light pen. The program under development displays desired numbers on the scope and allows the user to derive arithmetic and other functions useful in number theory by use of a light pen. In addition, the program will incorporate a system for retrieving theorems relevant to number theory problems being studied.

## INTRODUCTION

For some years the assistance which computers can give to research mathematicians has been on the upgrade. The research of Wang in theorem proving, Slagle in the automation of indefinite integration, the work of Chowla in number theory (described in the August 1963 issue of *Computers and Automation*), and others have increased the use of computers in mathematics. But the development of aids which, unlike the above programs, are of an on-line nature, enabling the user to get answers to a variety of different questions quickly has been much less abundant, and has awaited the development of communication devices such as the PDP-1 typewriter, scope and light pen. The low cost of PDP-1 time also has helped. It is noted that economically feasible on-line systems, however, still await the development of a truly large and efficient time-sharing system. This paper describes work now in progress at Information International Inc. towards the development of a computer program designed for on-line use to help mathematicians make new discoveries. The programs in development are, in fact, being done under a contract whose principle objective is to bring closer the goal of a practical time-sharing system. The area of mathematics selected for the project was number theory.

Number Theory, for those not familiar with the subject, deals with the divisibility properties of integers, with questions about the sequence of prime numbers (numbers having no divisors other than themselves and unity such as 2, 3, 5, 7, 11, etc.) and with the problems of finding solutions to equations like  $x^2 + y^2 = z^2$  where  $x, y,$

and  $z$  are required to be integers.

The system under development is designed to be used by someone already familiar with number theory. We don't propose to supplant the classroom or teach beginners. However, we expect the program to be useful depending on the sophistication reached after taking a one semester undergraduate course in number theory, on through to the level of someone doing frontier research. Of course we already have some experimental evidence from trial programs, that our system is a powerful aid to research, but our current concern is to create a program which can give this thesis a fair examination.

The system will have two main programs

1. Using the computer to perform calculations, including ordinary arithmetic and other computations especially useful in number theory.
2. A method of selecting mathematical theorems relevant to the problems being studied. It is basically the descriptor method used in information retrieval systems, only tailored to this particular application.

We will describe now the program which exploits the computer's ability to perform simple arithmetic more quickly and accurately than any mathematician like Gauss and Euler, both of whom made extensive use of calculation and inductive study of the properties of numbers to help them discover some very remarkable relationships among integers.

As a preparatory step, we wrote a program which used just the typewriter, sense switches, and the get-decimal-number routine. The size of our numbers was limited to the range of a PDP-1 register which has an upper bound of 131,071. We were able to compute: remainders upon dividing one integer by another, the greatest common divisor of two numbers, whether an integer was prime or not, prime factorization of an integer, the phi function of an integer (or how many integers less than a given integer there are which have no common factor with the given integer), how many primes there are less than a given integer, the sequence of twin primes (or pairs of primes which differ by two such as 11 and 13, 29 and 31); and a function which resulted from a bug, producing, for a given integer, the sum of the exponents in the prime factorization of a number (thus for 135 the function would compute that  $135=5^1 \times 3^2$ , and that the sum of the exponents 1 and 3 was 4 and would therefore print "4").

While using this program someone fairly versed in number lore observed for the first

time how dense primes actually are in the sequence of integers. Even with numbers in the hundred thousands, most sets of ten consecutive integers, (such as 120, 470 to 120, 479\*) contain at least one prime. We were also surprised by the fact that if you pick an odd number between 1 and 50 there is about an even chance that it will turn out to be prime. Then we started looking for sets of three consecutive integers all having the same number of prime factors, such as  $33=3 \times 11$ ,  $34=2 \times 17$ , and  $35=5 \times 7$ . Would you expect such triplets to become rarer or commoner as one proceeds up the scale of integers? After checking all integers below 300, and then looking up around 1000, we discovered, contrary to expectations, that these triplets occur quite frequently, sometimes close together, almost always with less than 100 between successive triplets, and that only once did the last number of one of the sets have 3 as a factor. Another question that struck our fancy was "What decimal numbers made from just the digits 1 and 0 are prime." In the course of examining all numbers of this type up to the limit of a PDP-1 register, we found that all primes up to and including 53 were factors of one or more of these numbers, except for 29. As this puzzled us we developed a kind of inverted long division which enabled us to discover a decimal number of about twenty digits, all ones and zeros, having 29 as a factor. Then continuing to work with the machine, in a few hours we collected empirical information on questions about numbers ranging in difficulty from trivialities, to doctoral thesis problems, to questions of the type number theory is famous for, questions that can go unsolved for a hundred years or even a thousand years (e.g., squaring the circle).

Multi-precision arithmetic: The program we are presently working on will use a routine capable of doing multi-precision arithmetic with numbers of up to twenty-two decimal digits. Numbers upon which mathematical operations are to be performed will come in from the typewriter, and immediately be placed on the scope. Nearly all further control will be accomplished with the light pen. The usual arithmetical operations, and the functions described in the earlier program, as well as calculations of special interest to the number theorist such as solution of linear congruences and linear diophantine equations, will all have their appropriate symbols on the scope. By touching with the light pen the desired numbers and operational symbols, computation will be effected and the results displayed. Control options are to include facility for obtaining the results of having the same operation applied to a series of consecutive integers without the user's having to call for each computation separately. The output from a series of computations may be

\*sets of the form  $10n$  to  $10n + 9$  inclusive

punched or printed by touching the appropriate symbols. This covers what we plan to do in the calculation part of the system. Before proceeding further, however, we must consider the question of what use this program is to a really serious investigator for in its present state, mathematics (even in number theory) deals primarily with abstractions of such a high order that concrete numbers bear little direct relation to the articles being published in the leading journals. In reply we ask two further questions:

First, what kind of scrawling do we find when we rummage through the wastebasket of today's mathematician? If we are to believe two of the leading mathematician's in the United States today, Polya and Ulam, the answer is that, there is a pretty good chance that you'll see numbers aplenty. Second, will a mathematician even bother to ask questions which are difficult to answer by reasoning and which at the same time involve conjectures impossible to judge the validity of except with involved calculation? Or, more precisely, will he bother to ask as many questions of this type (of which mathematics is full) that arouse his curiosity as he would if he had an easy-to-communicate-with computer at his disposal? Mathematicians as a rule do not like to concern themselves with laborious calculations and the time alone required for such work would turn him away from pursuing research in certain directions which would otherwise be quite valid. So we may conclude that such a device, apart from the aid it may give to current research, will permit and encourage the asking of questions which, might never be asked.

Selective Routine: The second program of this system seeks to mechanize a technique for retrieving and selecting theorems from number theory which are likely to be of service in solving problems in the field.

Let us suppose you have a definite conjecture in mind, that the numerical examples examined using the first part of the system have confirmed your hunch, and that you now wish to find a rigorous proof of its validity. What you will then do in order to use the program is to single out the main concepts which occur in the wording of the problem, or which you otherwise believe to be relevant to it. As soon as you type the names of these concepts in at the console, the computer will look through a large body of theorems in core or on magnetic tape, and for each theorem it looks at it will make a count of how many of the concepts you typed in are contained in that theorem. The theorems receiving the highest count from this process may then be displayed, typed out, or referenced to a nearby book where the proofs and corollaries may be found, depending on the requirements of the problem. The



supposition we are making, of course, is that the computer will select the theorems which are, in fact, the ones most likely to help you solve the problem, and we are also assuming that it does enough of the right kind of counting to enable it to give approximately the same results for different users who will have different opinions as to what concepts are relevant to a given problem. We must point out here, before you begin to imagine that the program is doing something very complicated, that we have kept matters simple by making our comparison not with the theorems as actually worded, but with a list, for each theorem, of what are considered to be its most important concepts. Also, we have avoided the problem of storing a large number of synonyms by making available, to the user, a list of those names for concepts which the computer will recognize.

For example let us suppose you wish to prove:

$$1. \quad \phi(m) = m(1 - 1/p)(1 - 1/q)(1 - 1/r)$$

where  $m$  is any integer equal to the product of three distinct primes,  $p, q,$  and  $r.$

Looking down your list of available concepts you note and type in; "phi," "equal," "product," and "prime." As soon as you signal that you are finished entering concepts, say, by typing "go" and a carriage return, comparing and counting will proceed as previously described. The scope will then display:

$$2. \quad \phi(p^e) = p^e - p^{e-1}, \text{ when } p \text{ is a prime}$$

The main concepts of 1 above were considered to be; "phi," "prime," "equal," and "exponent;" so that the theorem 2 would have been given a count of three, since that is the number of concepts it had in common with the ones typed in. The scope will also display the following theorem;

. . . if several numbers  $p, q, . . . s$  are relatively prime, then phi of their product equals  $\phi(p) \times \phi(q) \times . . . \times \phi(s)$  which was assigned the concepts "phi," "equal," "relatively prime," (which does not match with "prime") and "product," and, therefore, also had a count of three concepts in common with those we originally assigned to our problem.

Perhaps you are still not sure how to attack the problem using these theorems and so look up the reference to 2 and see that a corollary is

$$4. \quad \text{If } p \text{ is prime, then } \phi(p) = p - 1$$

you then realize that the left hand side of 1 can be rewritten as

$$5. \quad \phi(p) \times \phi(q) \times \phi(r), \text{ because any two distinct primes are relatively}$$

prime and we can apply 3; and now by applying the corollary 4, you can further rewrite 5 as

$$6. (p - 1) (q - 1) (r - 1)$$

Now looking back at the right hand side of 1 your intuition tells you to replace the "m" by its prime factors. Our equation is then:

$$7. (p - 1) (q - 1) (r - 1) = pqr(1 - 1/p) (1 - 1/q) (1 - 1/r)$$

And now the proof is easily completed by multiplying through on the right hand side of 7, the p times  $(1 - 1/p)$  giving  $(p - 1)$ , and doing the same for q and r making both sides of 7 identical, Q.E.D.

Several things are immediately evident about the program. The important thing is selection of the one or two theorems to try first among several dozen that appear more or less equally relevant. Second, the computer takes a much less vital role in this part of the system than in the calculation part previously described, the user being depended upon to make the final selection of which principles to try; for doing all of the reasoning required; and also for the original assignment of concepts to the problem. It is furthermore desirable to have the user decide what he considers to be the main concepts of theorems stored in computer.

The dependence on people for doing most of the work in using this system was carried all the way in some experiments in which people, making no use of the computer, did their own selection of theorems, of main concepts to be associated with the theorems, and did their own counting, using suitable notation to reduce the paperwork.

Some may have gathered, by this time, that we lack solid experimental evidence for the effectiveness of this procedure, yet the few people (half dozen) who have tried it have been amazed by how much their ability to solve problems at the end of textbook chapters has been enhanced. The results were equally striking when the same technique was applied to other branches of mathematics, including topology, vector spaces, and other branches of abstract algebra.

The pilot study for this program which is currently in development was a simple affair which identified concepts and theorems by numbers which the user interpreted by means of a list telling what number was assigned to what concept and what

theorem. The results obtained by people using the computer were about the same as by those who followed the procedure by hand, except that when the amount of counting to be done became large, the computer was favored.

Probably one of the reasons the method was found to be useful is that the user is forced to start making one kind of attack or another upon a problem and, therefore, will not flounder around as much as he otherwise might. But that it is useful apart from the well known fact in psychology, that any novelty or change in working conditions may cause improved performance, is indicated by the fact that when the textbook has given hints with problems, "try such and such a theorem," then the theorems suggested have usually been the same ones that were given the highest relevance count using the program. We are, therefore, of the opinion that though innumerable factors affect people's problem solving performance, and it is risky to make strong statements in favor of one method as against another, the computer, in this program, is making a pretty educated guess as to what would be a good theorem with which to attack the problem. Even when the program works badly, that is, lists as relevant things which aren't, and does not give the important theorems the highest count, it still seems to give better results than not using it at all.

Experience has shown that it is best not to have too many concepts which may be assigned to problems or associated with theorems in the computer. The optimal results seem to be obtained when just those concepts are used which the author of a text in the field of study has seen fit to define for the reader, sometimes adding a few that the programmer thinks he ought to define if he hasn't.

There are two devices we shall be using to increase the program's ability to discriminate against theorems which could receive a high count without being the most useful. One of these devices follows from the well known fact that in most branches of mathematics a few important theorems carry most of the weight in deriving the bulk of results in that field. These theorems will be starred or underlined in the display so that when they occur near the top but not at the top in relevance the user will know that they deserve precedence. The other device arises from the facts, or we should say, the tautologies, of logic. If what you are trying to prove is of the form

A implies B

then you generally have more use for a theorem of the type

A implies X

which tells you something you can derive from A to help you to "get" to B, than a theorem of the type

X implies A

telling you one way to derive A which is probably of little use. For similar reasons a theorem of the form

X implies B

which provides a possible step or route to the desired result, is considered more useful than one of the form

B implies X

The exploitation of these facts should now be apparent. Whenever possible we separate the concepts associated with hypothesis of the theorem from those which are associated with its conclusion. If the user will then oblige by making the same distinctions, separating the concepts most relevant to hypothesis of his conjecture from those most relevant to its conclusion, then our counting system will take advantage of the distinction and thereby improve its powers of discrimination.

In addition to the above theoretical points the system will include the usual advantages: facility to enter or delete theorems from the system, facility to alter the concepts associated with any particular theorem, etc.

# REQUIREMENTS OF A TIME-SHARED COMPUTER SYSTEM FOR PUBLISHING APPLICATIONS \*

Lawrence Buckland  
Inforonics Inc.  
Maynard, Massachusetts

## ABSTRACT

A program of research will be described to develop publishing and computer processing techniques required for recording useful textual data in machine form at the time of primary journal publication.

The approach to this objective consists of three steps:

1. To develop a system for recording journal articles in a machine interpretable form, such that the separate requirements of typographical composition, selective data extraction, and data retrieval are simultaneously satisfied by one keying.
2. To develop transformation procedures to convert the recorded data to a form useful for information retrieval and secondary publication purposes.
3. To explore the application of existing machines and techniques such as time-shared computer operation, to the publication system developed and to discuss any new machine characteristics required.

\*This paper was not submitted in time for publication.



# THE PDP-1 COMPUTER AS A TEACHING AID IN PROBLEM-SOLVING\*

Wallace Feurzeig

Bolt Beranek and Newman, Inc.  
Cambridge, Massachusetts

## ABSTRACT

A novel computer-aided teaching system is described and some applications with it are shown. The computer program states a problem to a student and engages him in "conversation" while he attempts to solve the problem. The student is free to choose his own line of questioning, using any of the items from a prescribed list of a possibly extensive vocabulary. The computer responses may depend not only on what has just been said but on everything that went before. Thus the system can discourse with the student in the manner of a personal tutor.

This is a report on our experience with a novel computer-aided teaching system that has been programmed for the PDP-1 recently. The initial objectives of our work in this area date from 1959 but the invention of an adequate program structure was first achieved in 1962. In the present paper some early applications of the system will be shown.

The computer program, called the Socratic System, states a problem to a student and engages him in "conversation" while he attempts to solve the problem. Instruction takes place at the computer console. The "conversation" is accomplished through the use of an electric typewriter connected to and controlled by the computer. The student types a question or an assertion and the computer types back a response--an answer, a comment or, possibly, another question.

Before the study session begins the student is given a list specifying the vocabulary for the problem. During the session the student's questions and declarations must be chosen from the terms on this list. The vocabulary can be extensive. The student is allowed considerable freedom in his approach to solving the problem--he can specify the information he wants when he wants it, and he can make assertions as to the solution whenever he wishes.

---

\*Work partially supported by Behavioral Sciences Laboratory, 6570th Aerospace Medical Research Laboratories, Wright-Patterson Air Force Base, Dayton, Ohio and by Office of Naval Research, Personnel and Training Branch, Washington 25, D. C.

The system records the information given to the student. It answers each question or declaration of the student by typing one or more responses from a pre-specified set of responses. The particular responses at any time are determined by the student's knowledge at that point. Each computer response may depend not only on what has just been said but on everything that went before.

The possibility of designing a sophisticated teaching machine was raised in an internal BBN memorandum (quoted below) by John A. Swets entitled "Some Possible Uses of a Small Computer as a Teaching Machine," dated 14 August 1959.

"Let's say we want to make good diagnosticians out of our blossoming M.D.'s. So we have lots of cases in a computer. A student comes into the computer room, selects a card out of a file, and learns that John Doe has a medical history of thus and so, that some intern has "worked him up" on his recent admittance thus and so. What's John's problem? The student sits down at an available typewriter, and decides what else he wants to know. He wants to know if John has urea in his urine, so he asks the computer and the computer tells him the answer is "yes." "Aha, then how many white corpuscles does he have?" Answer: "150." "Well," he tells the computer, "this is clearly a case of mononucleosis."

The computer replies: "Don't you think you ought to know whether John shows a Babinski reflex before deciding such?" "Yea," says the student, "I guess so, does he?" Answer: "Yes." "OK, now I'm sure it's mononucleosis." "But" says the computer, "you are forgetting that John's pulse is normal, which you well know, is inconsistent with your diagnosis. Etc. Etc."

It seems that such a facility would speed the learning process considerably. Maybe it also turns out to be a good idea to program case histories of research. "Jonas Salk found x to be true. What did he do next? No, he considered that possibility, but instead performed experiment y. Why do you suppose he did that? Yes, that's part of it, but you see, he also figured that if the result was z, then a and b must necessarily follow. Etc. Or maybe a budding pathologist comes in to find a certain slide projected on the wall. He grabs a typewriter and tells the computer that he sees a flim-flam at A6 and wants to know if that means carcinoma of the epiglottis. The computer tells him that he's a darn fool, that this is a section of the renal gland, and that wasn't a flim-flam anyhow. The computer also tells him that if he doesn't look a lot smarter on the next question, that he has no business tackling Slide 366, and he had better go back and get checked out at the 200 level.



As a matter of fact, students become addicted to this game; they sell their golf clubs, they drop out of the bowling league, they stop vacationing in Las Vegas. Even when they become interns and residents, they dash over for a try whenever they have fifteen minutes free. The word spreads that a Dr. Lahey at Johns Hopkins is the only man alive to solve case 174 and he did it in just ten exchanges with the computer--worldwide competition sets in. Medics all over take on a new hobby--they take a crack at programming each interesting case that comes along. Staff promotions at a junior level come to depend on how many cases one solves, and at a senior level on how many one sells."

### THE SOCRATIC SYSTEM

The author became involved in the area of computer-aided teaching toward the end of 1962. After considering the quoted memorandum and surveying the work being done at BBN and elsewhere, he decided with Swets that a more powerful computer-aided teaching system could and should be designed. The programming problems were solved early in 1963. The first version of the Socratic System was built around April 1963. It was debugged on a naive, though very useful problem--a guessing game, aptly designed for this purpose by Judith Harris of BBN.

In this game, called the Alphabet Guessing Game, the student (or player) tries to guess a letter of the alphabet by asking questions of the computer. The questions, for the most part, name simple geometric characteristics of a printed letter, such as the number of horizontal, vertical, oblique or curved line segments in the standard drawing of the letter, and so on. The student is provided with a picture of the letters in the standard drawing.

### ILLUSTRATIONS OF THE METHOD

The illustrative computer run that follows shows that the responses of the computer can be of some interest, even in this rather simple problem situation. In the photocopy shown here, the student's inputs are indented and capitalized to distinguish them from the computer's responses, which begin from the margin and have normal capitalization. (This convention is observed later in another illustrative run.)

Start when ready.

CURVES?

one.

LOOSE ENDS?

two.

OBLIQUES?

zero.

C.

You don't have enough information yet to get the right answer. How do you know it isn't J, for example?

J.

You don't have enough information yet. How do you know it isn't C, for example?

HORIZONTALS?

zero.

That was a wasted question. It did not eliminate any possibilities.

VERTICALS?

zero.

Good question.

G.

Wrong. Your guess is inconsistent with the information you've been given.

You must have forgotten that the right letter hasn't any HORIZONTAL LINES.

The letter you guessed can't be right--it has too many LOOSE ENDS.

CURVES?

The answer is still: one.

X.

Wrong. Your guess is inconsistent with the information you've been given.

I think you only made that guess to see what I would say.

C.

Yes, you've done it. Good work.

If you want to play again, start right in.

Since the letter game was applied, the system has been applied to an electronic trouble shooting problem and to a problem in medical diagnosis. The medical case was designed by Dr. Preston K. Munter of the Harvard Health Center. The author was ably assisted by Myra Breen of BBN in programming that case for the system.

A short, illustrative computer run with the medical problem is shown next. It is preceded by the statement of the problem (the case to be diagnosed) and the vocabulary, listing the terms available to the student for communicating with the computer. These are given to him just prior to the study session with the computer.

## PREAMBLE - MEDICAL CASE I

It is mid-December. Late this afternoon, a patient was admitted to the hospital. He is a well developed, well nourished, 34-year-old Negro coal miner. He was brought in by his wife who reports that he has had "the grippe" for about four days. During the past 3 days she has given him fluids and aspirin every six hours with no sign of improvement. Today he suddenly became much worse. For the past two days he had complained of feeling very warm and of pain in his belly. This morning he had a severe shaking chill that was so bad his teeth rattled.

You, the admitting physician, are requested to diagnose the case. To obtain the information you need to make this diagnosis, you are required to perform a methodical physical examination of the patient and to order all pertinent lab tests. This is done by typing questions to the computer. The set of questions you may ask is listed in the vocabulary you have been given. The computer will respond to each question with an answer or comment.

When you have finished the examination and lab tests, type: proceed. At this point the day "ends." Before you "go home" you may take a second series of lab tests during the night so the results will be ready for you "tomorrow." It is assumed that the second series consists of just those tests you think necessary.

Tomorrow morning, results of some of the first lab tests will be returned to you; others won't be available until late in the day. On the basis of information you have been able to obtain, you will be asked for the diagnosis. If you feel that no conclusive diagnosis can reasonably be made, you may then perform another full physical exam. Also, reports from the second series of lab tests may be requested.

You may make a diagnosis at any time you choose by typing the name of one of the diseases in the vocabulary. At various times the computer will discuss certain diagnostic possibilities with you. If questions are raised you must answer them by typing a diagnosis or an appropriate declaration from the final section of the vocabulary.

If you make a typing error, press the carriage return key and begin again. Please remember that questions must end with a question mark; everything else must end with a period. Begin the physical examination as soon as you are ready.

## VOCABULARY - MEDICAL CASE I

### Questions

physical exam?  
abdominal exam?  
chest exam?  
    auscultation?  
    localized tenderness?  
    percussion?  
    respiratory excursion?  
eent exam?  
    distension of neck veins?  
    stiff neck?  
general appearance?  
    backache?  
    contact with birds?  
    cough?  
heart exam?  
musculo-skeletal system?  
    extremities?  
neurological exam?  
rectal exam?  
vital signs?  
    blood pressure?  
    pulse?  
    respiratory rate?  
    temperature?  
laboratory reports?  
differential?  
esr?  
hemoglobin?  
rbc?  
wbc?  
urine exam?  
x-ray?  
    evidence of fluid?  
    lateral?  
blood culture?  
nose and throat smear and culture?  
sputum culture?  
sputum smear?  
urine culture?

### Declarations

acute pyelonephritis.  
appendicitis.  
friedlander pneumonia.  
influenza.  
mild upper respiratory infection.  
pleural effusion.  
pneumococcal pneumonia.  
primary atypical pneumonia.  
psittacosis.  
pulmonary infarction.  
staph pneumonia.  
strep pneumonia.  
the grippe.  
viral pneumonia.

yes.  
no.  
proceed.

evidence from blood culture.  
evidence from abdominal exam.  
evidence from blood test analyses.  
evidence from chest exam.  
evidence from eent exam.  
evidence from heart exam.  
evidence from ms system.  
evidence from neurological exam.  
evidence from nose and throat smear  
    and culture.  
evidence from rectal exam.  
evidence from sputum smear and culture.  
evidence from urine analysis.  
evidence from urine culture.  
evidence from x-ray findings.  
positive evidence of another disease.  
positive evidence though inconclusive.  
complications.

Please type your name, the date and a slash.

DR. SOCRATES 1963 /

**GENERAL APPEARANCE?**

Acutely prostrated, lying in bed on his right side. Looks acutely ill, sweaty, apprehensive, but alert. Paroxysmal cough, bringing up a little rusty streaked sputum. Breathing with difficulty.

**PULSE?**

100 a minute. Occasional extra systole.

**TEMPERATURE?**

103.

**BLOOD PRESSURE?**

150/84.

**RESPIRATORY RATE?**

20 a minute and shallow.

**EENT EXAM?**

Essentially negative.

**CONTACT WITH BIRDS?**

None, except patient has pet canary.

**RESPIRATORY EXCURSION?**

Apparently equal. Questionable pain on right side on inspiration.

**PERCUSSION?**

Normal.

**AUSCULTATION?**

Normal.

**LOCALIZED TENDERNESS?**

Questionable, on right.

**HEART EXAM?**

Not enlarged to percussion but low grade systolic murmur at apex. Otherwise normal.

**ABDOMINAL EXAM?**

Diffusely tender, some splinting noticed in lower right quadrant and occasionally radiating to right flank area.

#### MUSCULO-SKELETAL SYSTEM?

Essentially negative, except for generalized weakness and aching.

#### RBC?

A red count will be done.

Results will be sent to you as soon as possible.

#### DIFFERENTIAL?

Request noted.

#### WBC?

The WBC will be taken.

#### URINE EXAM?

The analysis will be done.

#### X-RAY?

X-rays will be taken.

#### NOSE AND THROAT SMEAR AND CULTURE?

They'll be ordered now.

When you've requested all the lab reports you want, let me know that you're ready to proceed.

#### PROCEED.

You may want repeats on some of the lab work. Would you like to alert the night nurse to set up the lab tests you'll want to see tomorrow?

#### YES.

Good. That will speed things up.

However, there's nothing further that can be done tonight to help the patient. Let's call it quits for now. Perhaps you'll be in a better position to diagnose the case tomorrow after the first lab reports come in and you have reexamined the patient.

Have a good night's sleep and when you're back on duty tell me to "proceed."

#### PROCEED.

It's morning and you're making your rounds. You've come to our patient and are given the night nurse's report:

"Temperature remained high but patient spent a reasonably comfortable night. Cough diminished but persists. Occasional traces of blood in sputum. Patient still dyspneic and still has pain in abdomen. The second series of lab tests was initiated."

Some of the first series of lab reports have just come in.

Urine exam: insufficient fluid for analysis.

RBC: 5 million

WBC: 15500.

Differential: Shift to left.

When the other reports are in you'll be notified.

Before proceeding with the reexamination it might be instructive to pause and consider the diagnostic implications of the information at hand. Perhaps you think it too early to arrive at any definitive conclusion. On the other hand, you may be prepared to make the diagnosis now. Are you?

NO.

Fine. Nevertheless, we've got a fair amount of information - at least enough to raise some, perhaps several, diagnostic possibilities. Have you any in mind?

YES.

Name one.

PNEUMOCOCCAL PNEUMONIA.

Good. The general signs of a pneumonic process are present. Of course, you haven't isolated the bug. Do you think the belly pain and flank pain reinforce the case for pneumococcal pneumonia?

NO.

Right. Though the white count is consistent, isn't it?

YES.

I think so too, but a later reading is indicated to confirm this finding.

We've considered one possibility. Do you have another diagnosis in mind?

NO.

In that case I'd like to talk about pulmonary infarction. What I like about that diagnosis is the presence of high fever, rusty sputum and shaking chills; I don't like the dyspnea, pleural pain and signs of consolidation - do you agree?

NO.

You're quite correct. I was just trying to see whether or not you were alert - and you were.

Some more of yesterday's lab reports have come in.  
Nose and throat smear and culture: no growth after 12 hours.

There was some trouble at the x-ray lab. Please excuse the delay.  
The x-rays are normal.

Now let's see how the patient is today. Begin the physical when you're ready.

**GENERAL APPEARANCE?**

Essentially unchanged since last night.

**TEMPERATURE?**

103.

**PULSE?**

105/minute. Still occasional extra systole.

**RESPIRATORY RATE?**

24/minute and still shallow.

**BLOOD PRESSURE?**

130/84.

**PNEUMOCOCCAL PNEUMONIA.**

Yes, I remember you liked this diagnosis before.

Are you absolutely sure?

NO.

What would help you nail down the diagnosis?

**EVIDENCE FROM SPUTUM SMEAR AND CULTURE.**

Good. Anything else?

**EVIDENCE FROM X-RAY FINDINGS.**

Good. X-ray shows area of consolidation at right lower lobe, consistent with a pneumonic process. Lab reports gram positive organisms in the sputum smear and the presence of pneumococci in the culture.

Had you thought to rule out things like appendicitis and acute pyelonephritis?  
Apparently you weren't thrown off by the referred abdominal pain. In any case you've made the correct diagnosis.



## CONCLUSIONS

It may be of interest to compare this run with the imagined dialogues in the Swets memorandum reference in the Introduction, keeping in mind that the system is still quite new and undeveloped, and that this is our first experience in medical case design. Medical educators have responded favorably to the results obtained thus far and have encouraged further development of the system for medical applications.

Recently, some success also has been achieved in other applications, notably in the area of management problem solving.

In the future the Socratic System will be programmed for a large time-shared PDP computer at BBN.



## Section IV

# HARDWARE AND INPUT-OUTPUT TECHNIQUES



# HARDWARE PROVISIONS FOR EFFICIENT TIME SHARING OF A PDP-1 COMPUTER

Natalio Kerllenevich

Massachusetts Institute of Technology  
Cambridge, Massachusetts

## ABSTRACT

This talk describes the hardware elements used to allow the M.I.T. time-sharing system to handle up to seven typewriters to be used as communications elements between users and their programs. The drum, used as a temporary storage, the use and assignment of I/O equipment and the typewriter interface connections are the main features discussed. The memory protection and special instructions implemented for the system, are discussed also.

This paper describes the hardware implementation of a proposed "Time-Sharing System for a PDP-1 Computer."<sup>1</sup> The system was implemented under the supervision of Prof. Jack B. Dennis on the computer donated by Digital Equipment Corporation to the Department of Electrical Engineering of Massachusetts Institute of Technology.

The main capabilities of the time-sharing system are the assignment and reassignment of external equipment, and the ability to trap on illegal instructions which are not allowed to the user or would stop the machine.

The ability of the machine to interrupt programs is accomplished by means of a modified Sequence-Break System. There are two channels in this SBS. The higher priority channel breaks to register 7000, as explained below. This channel also defines two modes of operation; (1) Executive Mode (while servicing a break), and (2) Non-Executive Mode.

In the executive mode, all the executive and IN-OUT instructions are allowed and executed, on the devices selected by the machine. User programs operate in non-executive mode where these instructions are traps.

The lower priority channel is just a normal SBS for the user to work with.

Breaks can occur for two reasons:

1. Any completion pulse from external devices, call button from consoles, or a counter of drum revolutions that keeps track of running time for each user.

This counter is loaded by the executive routine individually for every user. These breaks are asynchronous with the machine and are called "Interrupts." The number zero is placed in the AC.

2. Any attempt of execution of an instruction pertaining to a set of so called executive instructions (used by the executive routine to service users, like setting assignment register or loading the counter for the amount of time that his program runs), any IOT instruction, or any illegal instruction (like hlt or ill op). These also provide a number in the AC that determines the cause of the break, so that the executive routine can take proper action. These breaks are called Traps.

During the execution of a user's program, the occurrence of a trap or interrupt originates a break, as occurs in normal SBS. The AC, PC, and IO are stored in registers 7000, 7001 and 7002 respectively, and the instruction in register 7003 is executed. The machine now is in executive mode. Dismiss from this mode is accomplished by executing a jmp i 7001 instruction. Buffer full and Buffer empty flip-flops for each console determine which IO device has characters to transfer to or from the program. Execution Routines on each interrupt services the consoles. Also the time counter is checked, and in the case of a time out, ER dismisses the running program, and checks for the next program with a task to perform. If there is one that is not the one already in core, a swap from the drum is executed, storing the dismissed program and loading into core the next, which begins its running time. If the dismissed program has more tasks to perform and no others are waiting, then the dismissed program is restarted. Finally, if no programs are waiting, ER waits in program active loop. The call button can constitute another interrupt. If it is pressed for a given console, the debugger (DDT) is made active for that console.

In the case of traps, the trap number determines the routine to be selected to handle the instruction that caused the trap. On illegal instruction, a comment is typed and control returned to the debugger (DDT). On IOT instruction the character to be transferred is stored in the buffer, if it is not full, and control is returned to the user. If it is full on a character transfer out or empty on a transfer in, the user is dismissed and control given to another program. This dismissed user will be made active again when the cause of his dismissal disappears.

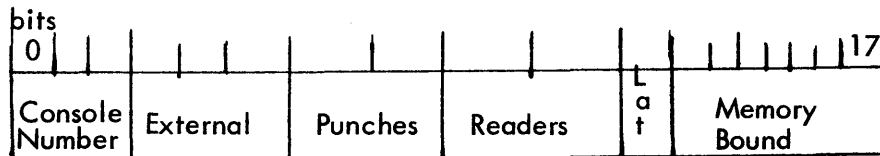
As a special case the user is allowed to execute three instructions to communicate with the executive routine: arq (assignment request), dsm (dismiss) and bpt (break-point). These three instructions trap, and their special trap number defines different operations to be executed.

arq request the assignment of an IO device, which depends on the number in the AC at the time of the execution of the arq. If the assignment is successful, the PC will skip the next instruction; otherwise, it will not skip.

dsm has been designed so that a user can begin a program by means of a console, and afterwards disconnect that program from the console. The console can be used for other purposes, and eventually the disconnected program can be addressed by a console by request to the administrative routine. This program will run until completion or trap, and then remain stored in the drum.

bpt is an instruction that allows the debugger to execute a breakpoint in a user's program.

Assignment is handled by means of a register, called the Assignment Register, that holds for each user information concerning the external equipment and memory bound assigned to him, which is loaded every time that his program runs. The register is implemented as follows:



These decoded levels define which of the external devices are to be activated on IOT instructions. The memory bound, which is variable, defines what is the highest register that the user can address without trapping as an illegal instruction. The contents of the Assignment register is changed by the administrative routine according to the needs and requests of the users.

Each console has its own sense switches which are selected with the corresponding console and are installed in a small board with the console switches.

These console switches serve the following purpose:

ON-OFF--This switch is used to connect the typewriter of the console to the system; if it is OFF, the system will not listen to the typewriter.

CALL--This is a momentary contact button, that calls the debugger when needed, interrupting the user's program (like in infinite loop).

DISPLAY LEVER--This is also a momentary contact, which when depressed, a longer quantum of time is allowed to the user of the corresponding console. His display instructions will intensify points in the screen of the C.R.T. (otherwise, dpy's are treated as nop's). The longer quantum allows the user to make observations on the Crt.

STOP PRINT--This switch stops the typewriter from typing out, permitting pagination, without interrupting or destroying the normal sequence of the program.

Levers in the console, and single step or single instruction switch have been disabled. Hlt and ill op code will not stop the machine, so that the machine will never stop in time sharing operation.

Provision has also been made for external connections to users' equipment. When a user requests external assignment, a number in the assignment register is decoded into a level that is used to get his in-out external instructions. Several users can use the same instructions without interfering with each other.

This system has been in operation since June 1963, with two consoles and two other programs that do not use typewriters. One is spacewar, and the other computes coordinate transformations in real time to track celestial bodies with a radio astronomy antenna. Its computation needs require about seven ms. per minute.

On debugging or editing operations, no difference is noticeable compared with non-time sharing operation, except for the fact that a smaller core memory is available to the user. (Each one has five drum fields to which he has access.)

---

<sup>1</sup> J. E. Yates, "A Time Sharing System for the PDP-1 Computer", DECUS PROCEEDINGS, 1962



## MICROTAPE: ITS FEATURES AND APPLICATIONS

Leonard M. Hantman

Digital Equipment Corporation  
Maynard, Massachusetts

### ABSTRACT

DEC has recently introduced the Type 550 Microtape Control and the Type 555 Dual Microtape Transport, which have the flexibility, speed and storage capabilities of magnetic tape, but yet maintain the convenience of paper tape. This paper will describe the distinguishing features of the Microtape system, indicate what programs are available for their use, some applications that are possible, and describe some ideas for future systems.

### INTRODUCTION

Most people, during their computer experience, have used and have become familiar with various types of magnetic tape storage. The Microtape system, however, has some features which make it flexible from a programming standpoint, at the same time producing an extremely reliable method of recording information, and a convenient system to use physically. Some of the features which will be described include Manchester type "polarity sensing," pre-recorded mark and timing tracks, individually addressable blocks and, in a sense, individually addressable words, bi-directional reading and writing, non start-stop operation, and the ease of loading, unloading and storing tape. In addition the technical characteristics of the Type 550 Control and some future controls will be discussed so that Microtape as a system can be more fully understood.

### TYPE 555 DUAL MICROTAPE TRANSPORT

The Type 555 Transport consists of two logically independent tape drives capable of handling 260 foot reels of 3/4 inch, 1.0 mil Mylar tape. The bits are recorded at a density of 375 ( $\pm 60$ ) bits per track inch and since the tape moves

at a speed of 80 inches per second, an effective information transfer rate of 90,000 bits per second is achieved. Individual 18-bit words, which are assembled by the tape control unit, arrive at the computer approximately every 200 microseconds, and therefore a block of  $256_{10}$  words will be transferred in 53 milliseconds\*. Traverse time for a reel of tape is approximately 40 seconds.

The 3-1/2 inch reels are loaded simply by pressing onto the hub, bringing the loose end of the tape across the tape head, attaching it to the take up reel and spinning a few times. Individual controls on the transport enable the user to manipulate the tape in either direction manually.

There is no capstan or pinch-roller arrangement on the transport, and movement of the tape is accomplished by increasing the voltage (and thereby the torque) on one motor, while decreasing it on the other. Braking is accomplished by applying a torque pulse to the trailing motor. The stopped condition is maintained by applying a small, equal, but opposite torque to both motors. As there is a small amount of roll on stopping or turning around, the units are not used on a start-stop basis, but rather to transfer fairly large amounts of data. In the present system the roll amounts to approximately one and one-half blocks. Start and stop time average 0.15 - 0.2 seconds and turn around time takes approximately 0.3 seconds. The reels themselves have been so designed that the ratio of outside tape diameter to inside tape diameter is a relatively low 1.3 to 1, thus keeping the torque requirements, and therefore tape tension, almost constant in either direction throughout the length of the tape. The small size of the reel makes storage or movement of many reels fairly convenient.

---

\*For the purposes of this paper, the term "block" will refer to a record on the tape consisting of  $256_{10}$  eighteen bit words plus associated control words. Note, however, that there is nothing inherent in the present system, or necessarily desirable, about blocks with the indicated format.

The units can be "dialed" into a particular selection address by means of a switch on the front of the transport. Up to four Dual Transports, i.e., eight drives, can be connected to one control unit.

All of the read and write circuitry, as well as the block format detection logic, is contained in the Type 550 Control unit. The transports consist of essentially nothing more than the motor drives, tape heads, and relays necessary for selection, motion control and transfer of information.

Physical dimensions of the unit are given in Appendix C.

### RECORDING TECHNIQUE

The Microtape system uses the Manchester type polarity sensed (or phase modulated) recording technique. This differs from other standard types of tape recording, where, for example, a flux reversal might be placed on the tape every time a "1" is desired. In the polarity sensed scheme a flux reversal of a particular direction indicates a "0" while a flux reversal in the opposite direction indicates a one. By using a timing track, recorded separately in quadrature phase, to strobe the data tracks, the polarity of the signal at strobe time indicates the presence of a zero or one. Using the timing track on the tape as the strobe also negates the problems caused by variations in the speed of the tape.

One disadvantage to the system is that the control must read double the number of flux reversals of other systems. However, with this type of recording one need not worry about the amplitude of the signal but only its polarity, thus removing some of the signal to noise problems, and allowing the use of read amplifiers with high uncontrolled gain. It also allows the changing of individual bits on the tape without changing the adjacent bits.

Reliability is further increased when we see that all five of the information tracks on the tape are recorded redundantly. This is accomplished by simply wiring the two heads for each information track in series, and on reading, the analog sum of the two heads are used to detect the correct value of the bit.

Therefore, a bit cannot be misread until the noise on the tape is sufficient to change the polarity of the sum of the signals being read. Noise which reduces the amplitude would simply have no effect. During testing, the tape has actually been read correctly with a piece of paper covering one half of the tape head. Tapes have also been read without loss of information in cases where the tapes were stopped by hand, and then released.

One other item affecting reliability should be mentioned, especially in a system which allows bi-directional transfer of information. That is the problem of tape skew as it passes over the head. Some tape systems will strobe on the first bit of a slot that it sees, then impose some arbitrary delay after which all signals present are then read. This produces problems in that there may be differences in the two directions. Variations in tape speed between write time and read time would result in non-compensated changes in the necessary delay. In the Microtape system the redundant heads are placed in a relationship to each other which, first of all, eliminates most of the cross-talk between the most important tracks, and second, places the timing tracks at the edges of the tape so that strobing on the analog sum of the timing track signals will guarantee that the data tracks are read when they are in the most favorable position. The data tracks are placed in the middle of the tape where the effect of skew is at a minimum in any case. The actual head arrangement is as follows:

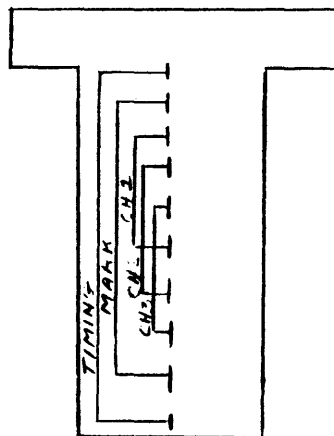


Figure 1

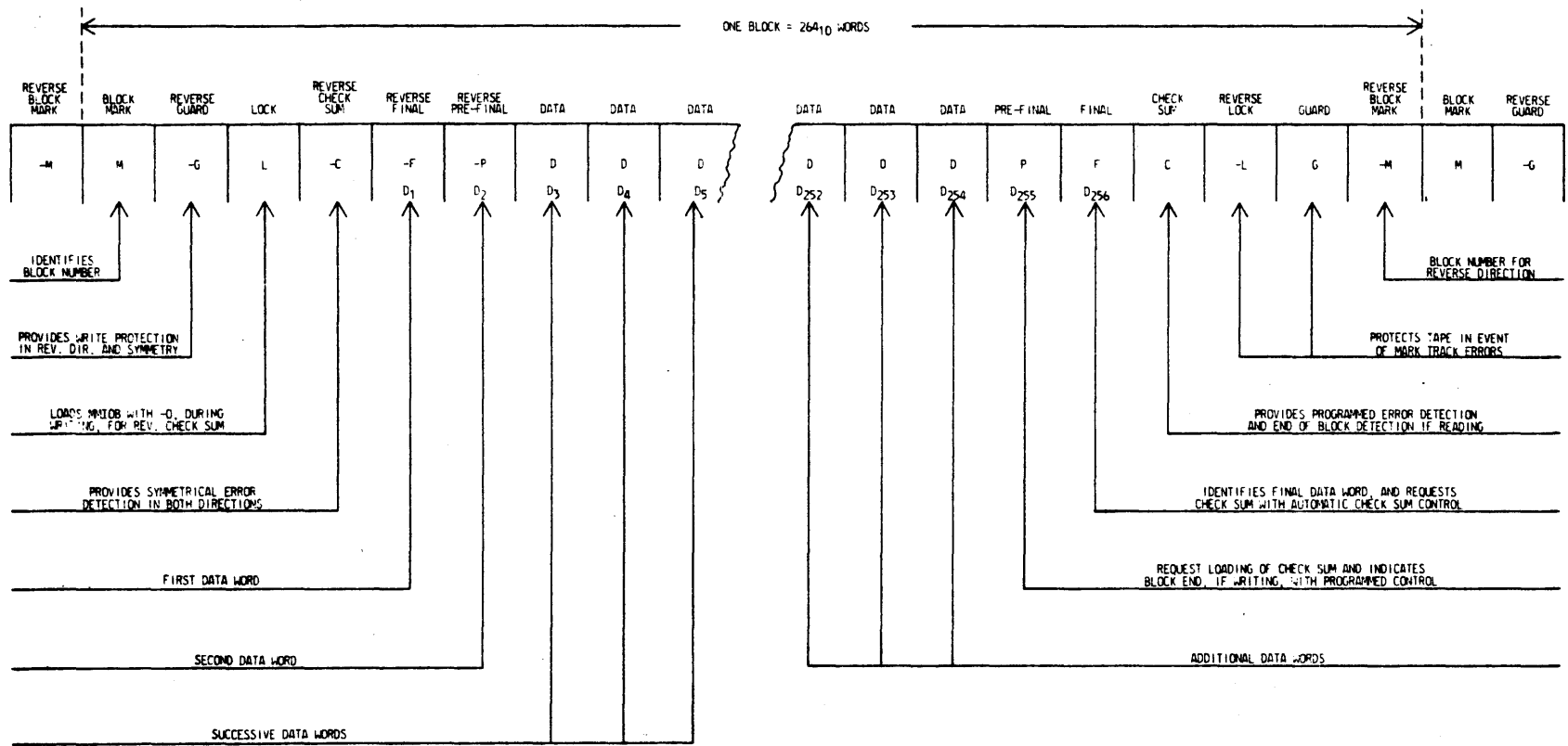
Placement of Microtape Tracks

The five tracks actually written consist of the timing track (used to strobe the other tracks), the mark track (used to raise flags in the program, create sequence breaks, detect block mark numbers, and protect control portions of the tape), and three data tracks. An eighteen bit word therefore uses six slots of three bits each on the tape.

# MICROTAPE MARK TRACK FORMAT

(ASSUMES 256<sub>10</sub> DATA WORDS PER BLOCK)

← MOTION OF TAPE



NOTE: END MARKS WHICH IDENTIFY THE PHYSICAL ENDS OF THE TAPE, ARE THE ONLY MARKS NOT SHOWN.

FIGURE 2

# FLAG RAISING, SEARCH MODE

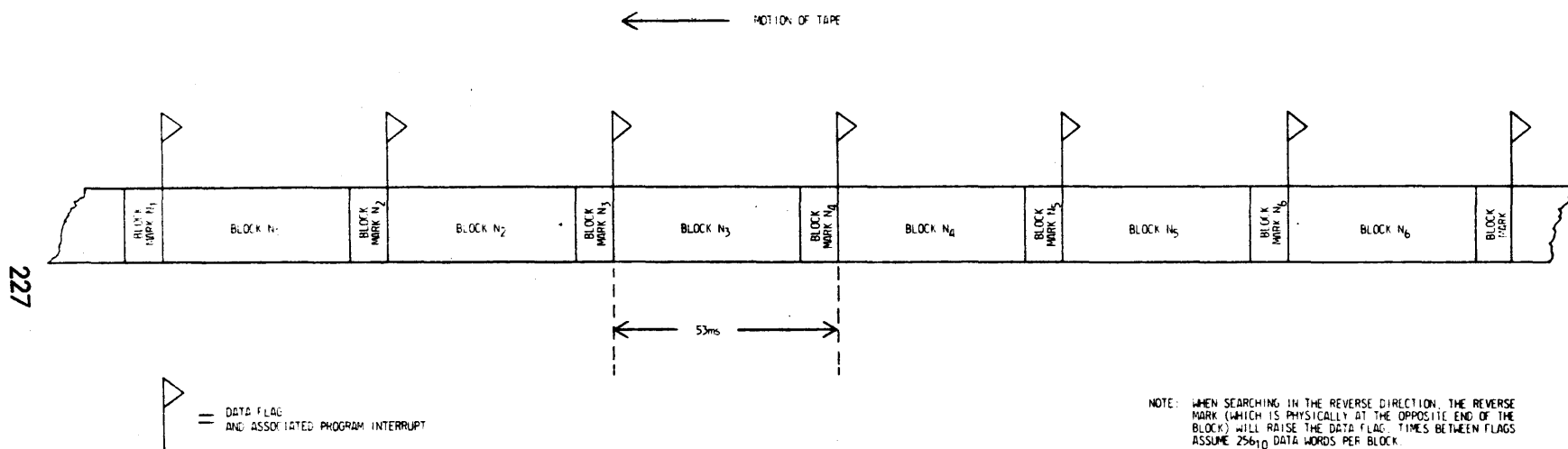
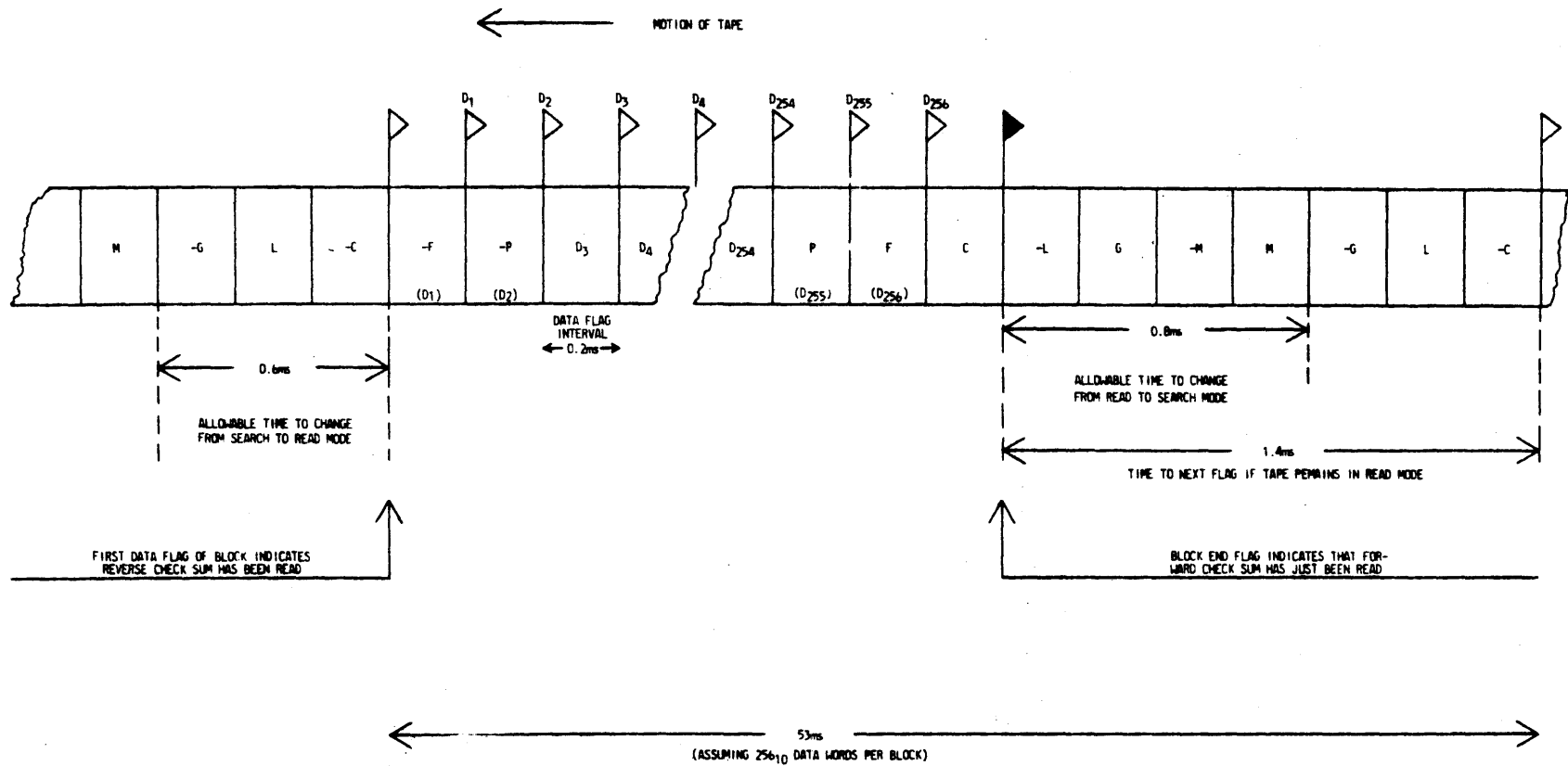


FIGURE 3

# FLAG RAISING—READ MODE



▲ = DATA FLAG AND ASSOCIATED PROGRAM INTERRUPT (NUMBERS OVER FLAGS INDICATE NUMBER OF DATA WORD JUST READ INTO MPROB)

▼ = BLOCK END FLAG AND ASSOCIATED PROGRAM INTERRUPT

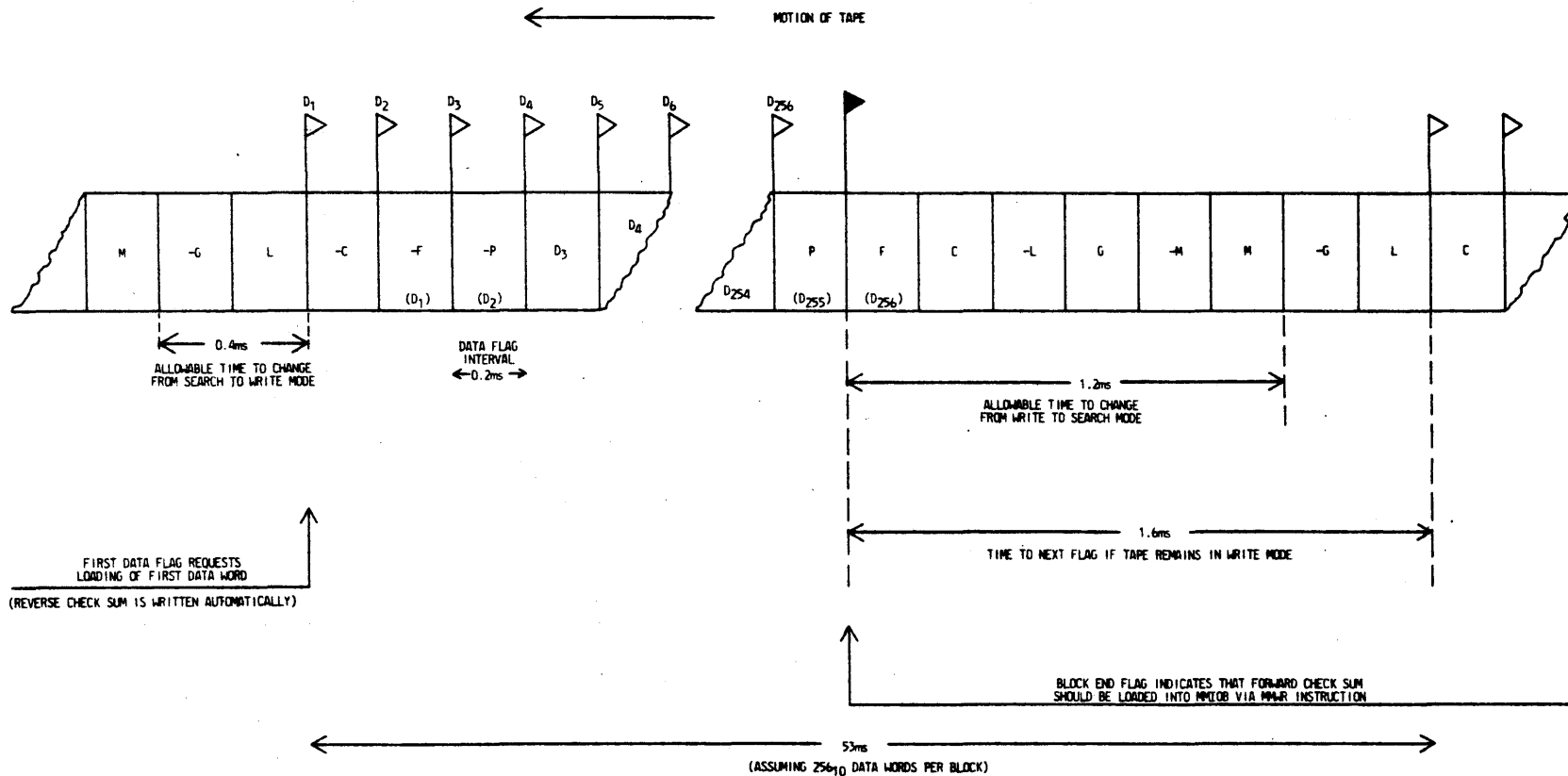
NOTE: FLAGS ARE SYMMETRICAL IF READING IN THE REVERSE DIRECTION.

FIGURE 4

228



# FLAG RAISING-WRITE MODE



229

= DATA FLAG AND ASSOCIATED PROGRAM INTERRUPT (NUMBERS OVER FLAG INDICATE NUMBER OF DATA WORD WHICH MUST BE LOADED INTO MMIOB)

= BLOCK END FLAG AND ASSOCIATED PROGRAM INTERRUPT

NOTE: FLAGS ARE SYMMETRICAL IF WRITING IN THE REVERSE DIRECTION.

FIGURE 5

# MARK AND INFORMATION TRACK BIT FORMAT

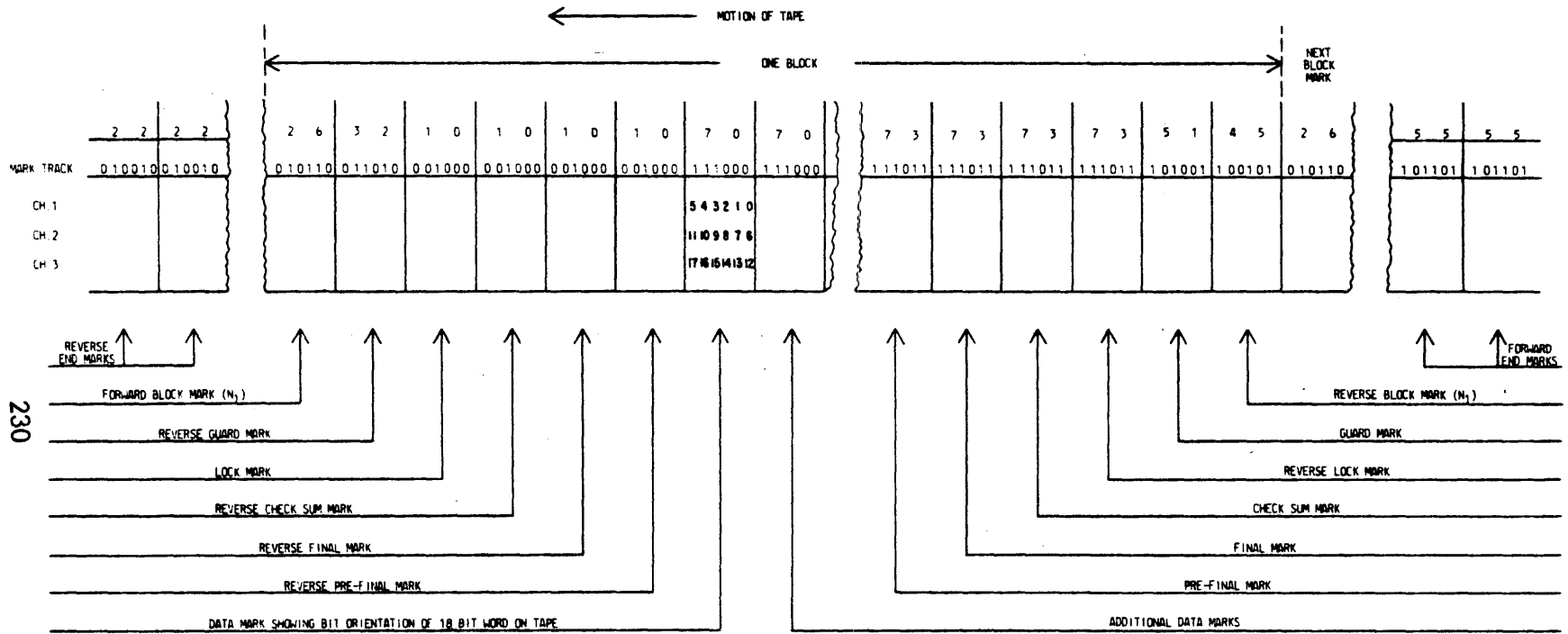


FIGURE 6

## TAPE FORMAT

The heart of the Microtape system is the pre-recorded timing and mark track. In the previous section the importance and use of the timing track was explained. This section will include a complete discussion of the mark track.

First, however, the reader should completely understand the meaning of the words "pre-recorded." At present, one of the programs provided with the Microtape system is one which will write the timing track and block format desired for the individual user. The Microtape system includes a programmed mode of operation called "Write Timing and Mark Track" and a manual switch which both permits writing on the timing and mark tracks and also activates a clock which produces the timing track and flags for program control. Unless both the mode and the switch are used simultaneously, it is physically impossible to write on the mark or timing tracks. A red indicator light will be lighted on all transports connected to the appropriate control when the manual switch is in the "on" position. In this mode only, information channel "one" (high order bits 0-5) is also connected to the mark track channel. Therefore, in one pass of the tape, the timing track, mark track, block format, and block mark numbers are created. Since part of the data word must be reserved to produce the mark track, it is impossible to write intelligent data in the information channels at the same time. For this reason also, only twelve of the eighteen bits are used for block mark identification, and bits 0-5 must be "anded" out when checking block mark numbers. (See figure 6 for format of bits on the tape). It is possible that the tape manufacturer may, in the future, supply tapes which will have the correct format actually pre-recorded on the tape. Once the format has been recorded the user is able to use the Microtape system for actual data storage.

The actual mark track which is written on the tape (see figure 2) was selected after careful consideration and provides many functions not readily discernible at a casual glance. Some of these are listed below and will be discussed fully.

- a) Program synchronization
- b) Block end detection
- c) Error checking and prevention
- d) Protection of control information
- e) Block and word addressability
- f) Automatic bi-directional compatibility
- g) End of tape detection
- h) Variable block format
- i) Inclusion of marks to allow expansion for more automatic systems of the future

For complete understanding of the questions of program synchronization and block end detection, figures 2 through 6 should be studied closely, using the explanation which follows to clarify certain main points.

There are three main programmed modes of operation which require that the user either provide information to the Microtape system, or accept information from the Microtape system. These are the "Search," "Read," and "Write" modes. (A fourth mode, "MOVE," simply moves the tape without supplying or requesting information.) In order to indicate to the programmer that the system is ready to transfer information, certain flags are raised\*. When these occur, the programmer must either load new information to be written, or unload information just read, and must do so within a specified time to prevent loss of information and error indications.

In order to produce these flags, the mark track is read by passing the bits through an 8-bit "moving window" which shifts bit by bit as the tape moves. A decoder associated with the window interprets the pattern present, and raises the

---

\*If the program interrupt mode is being used, assume that the raising of any of the flags mentioned also causes a sequence break in which the individual flags must be interrogated.

appropriate flags, if necessary. An 8-bit window is used, even though each mark is six bits long, to provide greater reliability, since a mark will not be recognized as legitimate unless the last two bits of the previous mark were legitimate. This is one of the reasons requiring ordering of the marks on the tape, the other will be mentioned later. Note that whether the program is reading or writing, the mark and timing tracks are always being read.

In Search mode the Data Flag will be raised when, and only when, a Block Mark mark is read (see figure 3). The program must unload the buffer within 53 ms, and bits 6-17 will contain the block mark number. Bits 0-5 will contain the mark code.

In Write mode, the Microtape system automatically writes the reverse check sum and raises Data Flags when it requires information to be written on the tape (see figure 5). The first data flag requests the first data word of the block, and the last data flag requests the last data word of the block; therefore there are a total of 256 data flags for a 256 word block. Note that the program loads each data word as the Microtape system is writing the previous one; thus a flag is raised requesting a data word when it has just passed the place on the tape two words ahead of where the word is to be written. Compare this with Read mode discussed below. Time between Data Flags is approximately 200 microseconds. When the pre-final mark is detected, a Block End Flag is raised which accomplishes two things. First it is a request for the program to load the calculated check sum (normally the complement of the 18 bit ring sum of the reverse check sum and the data words), and second, it allows the program to detect the fact that a block has been completed, without the use of any programmed counters. After the check sum is written the writers are turned off, to avoid any possible way of destroying the control portion of the block. Approximately, 1.2 ms is available to switch to Search mode if a check of the next block mark number is desired. If the control remains in Write mode the Microtape system will write the next reverse check sum and raise the next Data Flag after approximately 1.6 ms.

In Read mode, the first Data Flag is raised when the reverse check sum has been read (see figure 4). The reason for this becomes fairly obvious when one

remembers that we may read a block in either direction independent of the direction in which it may have been written. The first word read, therefore, is used to set the register which the program will use to accumulate the check sum. Each successive Data Flag indicates that a data word has been read, and should be unloaded from the buffer, stored in memory and accumulated in the check sum. When the check sum mark is detected, a Block End Flag is raised indicating both the end of the block and the fact that the check sum is in the buffer. This word would normally be unloaded and added to the accumulated check sum producing a total of zero. Any other result indicates that the tape has been read incorrectly, and the programmer has the option of continuing in any manner desired. Note that when reading there are 257 Data Flags for a 256 word block, and that each flag states that the associated data word is in the buffer. Note also that in the present system, validity checks on the data portion of the tape, are done by program control only. As a matter of fact, if for some reason a check sum is not desired, the check sum word can be used as simply another data word.

Checking of the mark and timing tracks is accomplished through the hardware and the physical characteristics of the mark track itself. One check, i.e., that of checking the last two bits of the previous mark, has already been mentioned, however another interesting fact emerges if we examine figure 6 and see what happens as the tape passes by the mark detecting window, bit by bit. Close examination will show that unless the window is actually looking at a legitimate mark on the tape (except an End mark) the bits in the window will differ by at least two bits from any possible legitimate mark.\* This guarantees that a one bit error any place on the mark or timing tracks can not cause an erroneous mark to be detected. It also allows checking for asynchronous marks. For example, once the window is in synchronization (normally by passing over a block mark) a Mark Track Error will

---

\*In the rare instance where they are only 1 bit different, the window has been cleared for other control purposes, so that one bit can make no difference at all.

be indicated (and the Error Flag raised), if a legitimate mark is found in less than six shifts of the window, or if a legitimate mark is not found after each six shifts of the window. These combinations of checks makes it virtually impossible to misinterpret the mark track and thereby destroy information.

Nothing in the system prohibits the changing of modes at any time during the movement of the tape. Thus it can be seen that, with some limitations\*, one might find a particular block in Search mode, count passed n words in Read mode, write one word or the rest of the block in Write mode, then switch back to Search mode to find the next block. Within those limits almost any combination of modes can be used, and because of the polarity sensed recoding technique, even individual words can be replaced.

One other unique feature of the mark track is that the six control marks before the data marks are, what we have chosen to call "complement obverses", of the six control marks after the data marks.\*\* The data mark is the complement obverse of itself. Thus, since when reading in the reverse direction, the flux reversals on the tape are opposite to those when reading forward, and the bits are read in the reverse order, the mark track window sees exactly the same thing in both directions. With one exception, no special logic is required to distinguish the format of the tape in either direction. The one exception involves the shifting of information into the Microtape buffer. Since the assembling of the 18-bit word is done by the hardware, it is necessary to shift the buffer in opposite directions for opposite movement of the tape in order to present words to the

---

\*Some of the things to be careful of include the difference in counting words when switching from read to write or from write to read, the recovery of the read amplifiers after writing (about 2 word times) and the fact that writing in various locations in the block will invalidate the check sum at end of the block.

\*\* The complement obverse of a word is defined as the complement of a word with the bits read in the reverse direction, i.e.:

010110 (26) and 100101 (45)
001000 (10) and 111011 (73) etc

computer as they were originally written. This means that if a record is read opposite to the way in which it was written, each 18-bit word will appear in the buffer exactly as it originally appeared in memory; however, the last word written would be the first one read, etc.

The End marks on either end of the tape illustrate this bi-directional ability even better. As the End marks are complement obverses of each other, only that end of tape will be recognized, at which the tape will physically come off the reel if further movement continues. Thus, here again, no special hardware is needed for opposite ends of the tape and there is no harm in coasting into or turning around in the end zones. Errors will be indicated only if attempting to go further into the end zone. The particular bit structure of the end marks is a repetitive one so that any shift of three bits in the window will appear as another end mark. This makes it virtually impossible to pull the tape off the reel in any of the normal modes. Sensing of the appropriate End mark will stop the tape and raise the Error Flag, if the tape is in any of the normal modes.\*

It can be seen therefore, that although the blocks are structurally alike in terms of the types of marks on the mark track, they need not contain the same number of data words. Indeed every block on the tape can be of different length, if such a format was created originally. The system will operate in the manner outlined no matter what the length of the block. One other feature exists which

---

\*There are only two "abnormal" modes. One is the Write Timing and Mark Track mode mentioned previously in which no marks can be detected since they are being written. The other is the case where a tape has been left moving but not connected to the control (deselected). In this case, only the marks on the actually selected tape will be recognized. In only these two circumstances can the tape be pulled off the reel.



may prove useful, especially in future designs. If for any reason the distance between blocks must be lengthened it can be done simply by adding "01" codes between the Reverse Block Mark of block N and the Forward Mark of block N+1 (see figure 6). Since the pattern "01010101" already appears at the junction of the two marks, it may be continued indefinitely without harm.

Additional flexibility has been retained for future expansion. For example, in the future the contents of the Lock Mark might be used to determine if the block is "file protected", i.e., cannot be written on. The Final Mark could be used to request the check sum from the hardware, in a system having automatic sum checking, etc.

#### AVAILABLE PROGRAMMED SUBROUTINES

Three main groups of programmed subroutines are provided with the Micro-tape system for both the PDP-1 and the PDP-4. The first is a basic set of subroutines for searching, reading and writing; the second is a set of maintenance and diagnostic programs which can accomplish combinations of Microtape functions using the toggle switches on the console (MICROTOG); and the third is a simple routine to save programs or data on Microtape, and allow quick retrieval via the toggle switches (MICROTRIEVE). Both MICROTOG and MICROTRIEVE use the basic read, write and search subroutines as provided for the programmer, and are basically the same for both the PDP-1 and PDP-4. There are however, some differences in the basic subroutines which will be described below.

For the PDP-1, the basic subroutines are designed to read or write one block of information, in either direction, depending on the current position of the tape and the direction in which the tape must be searched. If the tape is used in the reverse direction, data will be transferred starting with the end of the block in core storage; otherwise data will be transferred normally starting with the beginning of the block in memory. This allows the direction of reading to be independent of the direction of writing without destroying the normal order of the words in

memory. The search subroutine needs only the appropriate unit, block number, and an error return as parameters. The read and write subroutines, which require a unit, block number, starting address and an error return as parameters, automatically enter the search subroutine to find the block requested. All three subroutines leave the tape running when completed, to allow additional tape functions if desired. Programs have been written in MACRO for both the single channel and sixteen-channel sequence break systems. Multi-programming will occur only during searching however, as the total machine time is preempted during the actual transfer of data. Errors are detected, saved in status bits, and indicated by a special return, at which point the programmer has the option of continuing in any manner desired. Approximately 200<sub>8</sub> words of core storage are used.

The basic PDP-4 subroutines allow the user to specify the total number of words to be transferred irrespective of the block format on the tape. Searching will occur in either direction; however, reading and writing will be done in the forward direction only. If the number of words specified during writing is not a multiple of the block length, the final block is completed with words of plus zero (+0). On reading, only the correct number of words will be stored in memory; however, reading will continue until the end of the last block so that the final check sum can be calculated and checked. The program assumes the use of the program interrupt. One auto-index register must be defined by the main program, and "DISMIS" must be defined as a JMP to the instructions which dismiss the interrupt. Instructions to check the appropriate flags must also be included in the programmer's interrupt sequence.

The search subroutine, which requires a unit, block number and error return as parameters, will search for the specified block and either stop, remain running in the forward direction, or remain running in the reverse direction according to the subroutine entrance used. As soon as searching is started, a return is

made to the main program to allow simultaneous multi-programming .

The read and write subroutines which require a unit, block number, starting and ending core addresses, and an error return as parameters, automatically enter the search subroutine to position the tape . During data transfers no multi-programming is permitted, and when the transfer is completed the tape is stopped. Errors are detected, coded numerically, saved in status bits and indicated by a special return. The programmer can decode the type of error and continue in any manner desired. Approximately 350<sub>8</sub> words of core storage are used.

MICROTOG for both the PDP-1 and PDP-4 is a collection of fairly short programs which allow the user to perform various Microtape functions using, as input to the program, only the toggle switches on the console . The programs available include those which will allow: creation of the mark track and desired individual block format, reading or writing specified portions of the tape, writing a "virgin" tape (tape with known block content for test purposes) in either direction, sum checking specified portions of the tape in either direction, "rocking" the tape in both directions in specified modes for indicated times or distances, generation of specified types of data blocks, and exercising the tape by writing, reading and sum checking in both directions . Errors are completely analyzed and typed out together with the number of the block causing the error, and the exact status of the Microtape at the time of the error. Detailed descriptions of the use of the various sub-programs are currently available .

MICROTRIEVE allows the user to specify via toggle switches the information necessary for the storing and retrieving of data or programs on a Microtape library tape . When storing data the program will search for the block indicated, and write the indicated area of memory on the tape together with an identification and two control words . A message is typed upon completion which includes the starting and ending block numbers used for storage and a number indicating the total check sum of the entire area written . When retrieving the information, only the unit and block number need be specified, as the control words on the tape will indicate the starting address and length of the information in memory . A check

is made to guarantee that the block specified is actually the start of a storage area. Upon completion a message is typed which shows the starting and ending block numbers and the total check sum. This can be checked against the data typed during writing to insure that the correct information was read. Errors are fully analyzed and typed as in MICROTOG.

### FUTURE TRENDS

As has been described, the Microtape system for the PDP-1 and PDP-4 is basically an 18-bit control operating in the program interrupt mode. Since the introduction of the PDP-5 and PDP-6 which are 12 and 36 bit computers respectively, much thought has been given to the desirability of making the tapes produced with the various systems compatible. Therefore it is expected that a control will be introduced in the future which will allow the assembling of variable length words which are a multiple of three bits. To achieve this flexibility certain changes will have to be made. For instance, the calculating and checking of the check sum will be done automatically, so that no difficulty arises in checking tapes produced on a machine with different word length. In addition the data will be placed on the tape in a slightly different manner so that the length of the word does not affect the order in which the bits are assembled.

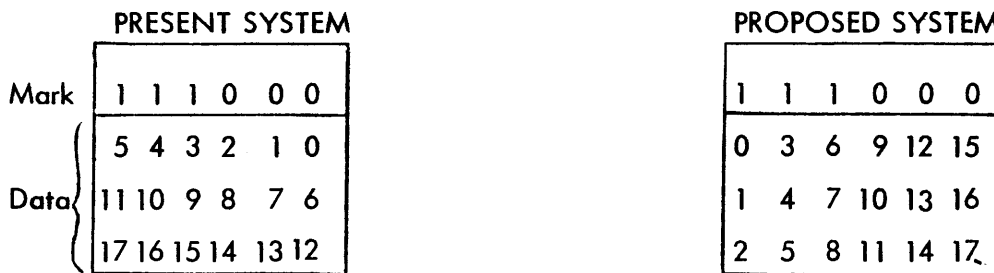


Figure 7  
Placement of Bits on Microtape

An Additional mode, "Write all Information Bits" will permit writing in the information channels of the control words of the block, allowing a simpler creation of the block format, and giving additional flexibility to the programmer.

Transfer of information will be in terms of full blocks; however, Data Flags will still be provided at each word so that in desired cases, the user can still address individual words of the block.

### APPLICATIONS

It is of course, impossible for the writer to list all of the possible applications of the Microtape system. There are certain characteristics of the system, however, which make it particularly suitable for certain applications, and these will be covered briefly. As Microtape is used in the field, additional applications will most certainly suggest themselves and will be reported on.

The first application is simply as a storage device for programs and data. Since the tape handling is extremely simple, it is easy, and in fact, desirable to store the programs one needs on Microtape, and simply carry it to the computer for use when needed. To carry the same amount of data on either cards or paper tape would be unwieldy to say the least. Different library tapes can be changed easily if necessary, and retrieval of any portion of the tape is relatively fast. If modifications to the programs are necessary, the tape need not be either re-written entirely to preserve the order, or added to at the end. The program can be read in, modified, and re-written in the same location on the tape, providing its block length is not changed. This indicates also that many programs can be written utilizing a minimum number of drives.

On an on-line system, use of individual Microtapes to store information keyed in by individual users provides a fairly cheap and efficient way of handling data. The ability to multi-program during searching (which requires by far, the greatest amount of time) means that more than one individual can have access to the computer without appreciably affecting internal processing, and without causing an inordinate

amount of waiting time for the user. An extension of this is discussed in the next application.

Since the Microtape reel is fairly small, and the system can read or write in both directions, random access to any point on the reel is relatively fast. A fairly large amount of data can be stored however and for example, one tape can hold more than 22 complete 4K memories. In a real-time, multiple-user, random access system, many tapes can be moving simultaneously even though data can be transferred on any one tape at a time. For example, let us imagine a system with several remote teletypes, each requiring random access to information stored on several Microtapes. When the first request occurs, the program can place the appropriate tape in Search mode and begin searching for the block. If another request occurs, the program can note the approximate position of the first tape in relation to the block requested, select the new tape (leaving the old tape moving) and start searching for the new block. A programmed clocking device or timing loop can be used to determine when to re-select the first tape, check for the correct block, and transfer the data. As new inquiries enter the system, a queue can be formed with the request for the nearest information and the time needed to reach it, at the top of the queue. As information is found, the clock is reset to the time necessary to reach the next request and so on. In this way, multiple requests for information on a single tape can be fairly easily handled, if both records can be found by searching in the same direction. There will be times, of course, when data will be reached on more than one tape simultaneously. In this case the tape searching for the later request can either be stopped before the record is reached, or can be turned around if the record has been bypassed. In terms of overall time to the user, very little difference will be noticed. Of course if two separate Microtape controls are used, data can actually be transferred on more than one tape simultaneously, providing the program is fast enough to react to the various flags.

A third type of application involves the continuous movement of the tape. There are many instances when it is desirable to store sampled data on a tape for future analysis by other programs. Memory fills up rapidly however, and

during the time information is transferred onto the tape, sampling is usually stopped to avoid synchronization problems. Thus the information stored usually consists of data relating to many relatively short samples. With Microtape, one whole tape can be written with one command and therefore, an extremely long sample of fairly rapid data can be achieved. If desired the entire tape can be considered as one long block of information. Storing of information from an analog-digital converter, would be a logical use of such a system.

Some thought has been given to the question of sorting and merging using Microtapes. Though all of the problems have not yet been worked out, it appears that the continuous motion of the tape, and the ability to read and write in both directions, may make certain types of sorting very efficient. For example, in an unbalanced polyphase sort, one could continuously store information on every third or fourth block on the tape in both directions. Depending on the final position of the tape, some rewinding would have to be done to re-read the information for the next pass. However, once the information has been read, new data can be stored beginning with the current position of the tape. The tape then theoretically becomes an endless loop, and rewind time is reduced appreciably. Depending on the speed of the program, if the tape can remain moving continuously except for turn-around time, there need not be any start-stop time delays from the tape unit. It is obvious however that the relatively small size of the reels would limit the amount of data which could be sorted, and the programming necessary to compensate for the normal roll of the tape, may also become prohibitive.

APPENDIX A

MICROTAPE INSTRUCTION LIST

<u>PDP-1 Mnemonic</u>	<u>PDP-1 Binary</u>	<u>PDP-4 Mnemonic</u>	<u>PDP-4 Binary</u>	<u>Function</u>
MRD	720501	MMRD	707512	READ. Clears IO or AC and transfers one word from MMIOB to bits 0-17 of IO (PDP-1) or AC (PDP-4)
MWR	720601	MMWR	707504	WRITE. Transfers one word from bits 0-17 of IO (PDP-1) or AC (PDP-4) to MMIOB.
MSE	720301	MMSE	707644	SELECT. Connects the unit designated in bits 2-5 of the IO (PDP-1) or AC (PDP-4) to the Microtape Control
MLC	720401	MMLC	707604	LOAD CONTROL. Sets the Microtape Control to the proper mode and direction from bits 12-17 of the IO (PDP-1) or AC (PDP-4), as follows:  Bit 12 = Connect (Go) Bit 13 = Reverse Bit 14 = Spare Bits 15-17 = Mode: <ul style="list-style-type: none"> <li>0 = Move</li> <li>1 = Search</li> <li>2 = Read</li> <li>3 = Write</li> <li>4 = Spare</li> <li>* 5 = Read through block ends</li> <li>* 6 = Write through block ends</li> <li>7 = Write timing and mark track</li> </ul> <u>i.e.</u> 42 = Read Forward 62 = Read Reverse 43 = Write Forward 41 = Search Forward 61 = Search Reverse *Not presently connected



MICROTAPE INSTRUCTION LIST (CONTINUED)

<u>PDP-1 Mnemonic</u>	<u>PDP-1 Binary</u>	<u>PDP-4 Mnemonic</u>	<u>PDP-4 Binary</u>	<u>Function</u>
MRS	720701	MMRS	707612	<p>READ STATUS. Clears the IO or AC and transfers the Microtape status conditions into bits 0-8 of the IO (PDP-1) or AC (PDP-4) as follows:</p> <p>Bit 0 = Data Flag            Bit 1 = Block End Flag            Bit 2 = Error Flag            Bit 3 = End of Tape            Bit 4 = Timing Error            Bit 5 = Reverse            Bit 6 = Go            Bit 7 = Mark Track Error            Bit 8 = Tape Unable</p>
		MMDF	707501	<p>Skip on Microtape Data Flag            In Search Mode: Block mark number should be unloaded via (M) MRD instruction            In Read Mode: Data or Reverse Check Sum should be unloaded via (M) MRD instruction            In Write Mode: Data should be loaded via (M) MWR instruction</p>
		MMBF	707601	<p>Skip on Microtape Block End Flag            In Read Mode: Unload forward Check Sum via (M) MRD instruction            In Write Mode: Load calculated forward Check Sum via (M) MWR instruction</p>
		MMEF	707541	<p>Skip on Microtape Error Flag            Timing Error, Mark Track Error, End of Tape, or Tape Unable Condition has occurred. Use (M) MRS instruction to detect specific error.</p>

NOTE: MMSE and MMLC clear the Error Flag and MMSE, MMLC, MMRD, and MMWR clear the Data and Block End Flags.

APPENDIX B

MICROTAPE OPERATION CHART (PDP-4)

<u>FLAG</u>	<u>MOVE MODE</u>	<u>SEARCH MODE</u>
Data Flag cleared on  mmrd mmwr mmlc mmse  This flag causes interrupt	No Data Flags raised. Tape motion is continuous until End marks are sensed at far end of tape.	Data Flag means that the MMIOB contains a Block Number. Write mode may be specified within 400 microseconds to transfer the block. Read mode may be specified within 600 microseconds.* Any other mode (including Stop), may be commanded at any time. Transfer of Block Number must be completed in 53 milliseconds to avoid a MISS.**
Block Flag cleared on  mmrd mmwr mmlc mmse  This flag causes interrupt	Should not occur	Should not occur
Error Flag cleared on  mmse mmlc (also clears MISS, END, MTE)  This flag causes interrupt.	Error Flag means that an error has occurred. An mmrs command will load AC bits 0-8 with status information. (END is only possible error.) END Status bit is set when tape reaches far end. Error Flag is raised. Tape stops.	Error Flag means that an error has occurred. An mmrs command will load AC bits 0-8 with status information. (END, and MISS are only possible errors.) End status bit is set when tape reaches far end. Error Flag is raised. Tape Stops. MISS Status bit is set when a Data or Block Flag has not been cleared from previous use.

\* All times are nominal for forward direction. In reverse direction add  $\pm 20\%$ .

\*\* MISS indicates a programmed timing error; i.e., information will be lost (missed) because the routine is taking too long to transfer data to or from the buffer.

APPENDIX B

MICROTAPE OPERATION CHART (PDP-4)

<u>FLAG</u>	<u>READ MODE</u>	<u>WRITE MODE</u>
<p>Data Flag cleared on</p> <p>mmerd mmwr mmic mmse</p> <p>This flag causes interrupt.</p>	<p>Data Flag means that MMIOB contains a data word. An mmerd must be given within 200 microseconds for data transfer.</p> <p>First Data Flag in block indicates Reverse Check Sum.</p> <p>Change to other modes possible within 200 microseconds. If Write mode is desired, a one word delay occurs after mmwr is given.</p>	<p>Data Flag means that MMIOB is ready for Data word. An mmwr must be given within 200 microseconds for data transfer. Initial (-0) Check Sum is written automatically. First flag in block is a request for first Data word.</p> <p>Change of mode possible within 200 microseconds. Since tape system is bidirectional the initial Check Sum written may be placed at either Forward or Reverse Check Sum location in block, depending only on direction commanded.</p>
<p>Block Flag cleared on</p> <p>mmerd mmwr mmic mmse</p> <p>This flag causes Interrupt.</p>	<p>Block Flag means that Check Sum is in MMIOB. First Data Flag of next block will automatically occur in 1.4 milliseconds.</p> <p>Change to Search mode must be made in next 800 microseconds in order to catch next mark. Change to Write mode must be made within next 1.2 milliseconds in order to start new block (not recommended - Block Number should be checked by Search Mode).</p>	<p>Block Flag means that Check Sum should be loaded into MMIOB with an mmwr.</p> <p>First Data Flag of next block will occur in 1.6 milliseconds. Change of mode commanded at last Data word (<math>D_{256}</math>) is delayed while Check Sum is written.</p> <p>Change to Search mode must be made within 1.2 milliseconds to read next Block Number. Preferred method of stopping is to change to Search mode, then check succeeding Block Number for correctness before stopping.</p>
<p>Error Flag cleared on</p> <p>mmse mmic (also clears MISS, END, MTE)</p> <p>This Flag causes interrupt.</p>	<p>Error Flag means that an error has occurred. An mmrs command will load AC bits 0-8 with status information. (END, MISS, Mark Track Error (MTE) are only possible errors.)</p> <p>END status bit is set when tape reaches far end. Error Flag is raised. Tape stops. MISS status bit is set when a Data or Block Flag has not been cleared from previous use.</p> <p>Mark Track Error (MTE) Status bit is set upon discovery of certain Mark Track and timing track Errors.</p>	

## APPENDIX C

### PRELIMINARY SPECIFICATIONS

TAPE AND REEL	260 feet of 3/4 inch tape on a 3 1/2 inch reel. Tape is 1.0 mil Mylar.
WORD TRANSFER RATE	One 18-bit word each 200 ( $\pm 10$ ) microseconds. Bit rate is constant when moving forward. Although velocity varies slightly, bit density changes serve to maintain a constant bit rate due to the constant rate timing track. In reverse direction the variation in time between words becomes $\pm 20\%$ depending on location along the tape.
SPEED	Varies according to reel diameter from 70-80 ips.
DENSITY	375 ( $\pm 60$ ), 3-bit characters per inch. 3 million bits per reel.
START TIME	Less than 0.2 seconds.
STOP TIME	Less than 0.15 seconds.
TURN AROUND TIME	Less than 0.3 seconds.
START AND STOP DISTANCE	Less than 8 inches.
ACCELERATION	700 ( $\pm 150$ ) inches per second per second
COMMAND SIGNALS	Contact closures: Select, Go, Reverse, and a 10-wire Select Buss. One Two-wire write interlock loop. Two connector plugs wired in parallel for easy bussing.
INFORMATION SIGNALS	5 shielded triplets. 5 millivolt p-p normal read signal over 30 feet of cable. 120 ma nominal write current. Phase or Manchester recording used with reference to timing track zero crossing for read and write timing.
POWER REQUIREMENTS	110 to 120 volts, 60 cps, 400 watts maximum.
WEIGHT OF TRANSPORT	65 pounds.
DIMENSIONS OF DUAL TRANSPORT	19 inches wide, 16 inches deep, 11 inches high. 1 3/4 inch switch panel. Mounts in 19 inch standard rack.

# ON-LINE INPUT OF GRAPHICAL DATA

William E. Fletcher

Bolt Beranek and Newman Inc.  
Los Angeles, California

## ABSTRACT

Immediate input to a computer of data which exists on paper in graphical form is often desirable but rarely possible using available digitizing devices. This paper will discuss a device developed by Bolt, Beranek and Newman Inc. to facilitate rapid input of graphical data. The tip of a pen held by the operator is traced over a curve of interest or pointed at discrete spots. A light-weight arm connects the pen to the device which then makes the digitized position of the point of the pen available to the computer. Suitable programming allows the construction of a complete digital representation of graphs, charts, maps, or freehand sketches in the time that it takes to trace or sketch them by hand. Accuracy, cost, versatility, ease of use, and applications will be discussed. This device has been in daily use in conjunction with a PDP-1 Computer at the Los Angeles office of Bolt Beranek and Newman Inc. for more than six months.

## INTRODUCTION

In general the digitizing of data existing in graphical form is accomplished by manually positioning x and y cursors over the desired points on the data or its projected image by rotating knobs. This process ties up both hands of the operator and except in the case of operators with rare skill, must be done sequentially, i.e., position the x cursor, then the y cursor. In addition, the medium on which the data to be digitized is recorded is rather restricted. It is usually best if it is on film. Though it is possible to project data from opaque surfaces, such as paper, the form of the paper is subject to restrictions. A page in a book or report, or a large architectural drawing will not do.

About a year ago Bolt Beranek and Newman Inc. faced the problem of obtaining a digital representation of a large body of data which existed in graphical form. In particular, this data consisted of 1/3 octave band (OB) and spectrum level acoustic and vibration curves obtained during a missile research and development program. These data were processed by three different agents and existed in a multitude of bound and unbound reports and were printed on various grades and sizes of paper. It was this situation which led us to develop the device and techniques to be described in this paper.

## A GRAPHICAL INPUT DEVICE

To the obvious requirement that the device must be able to accomplish the job we faced as outlined above, we added three more:

1. The device should be easy to use, i.e., human engineered.
2. The device should utilize existing special purpose equipment in our computer laboratory - in particular, a computer controlled A/D converter and a computer controlled bank of relays.
3. The device should be usable on-line to facilitate error checking.

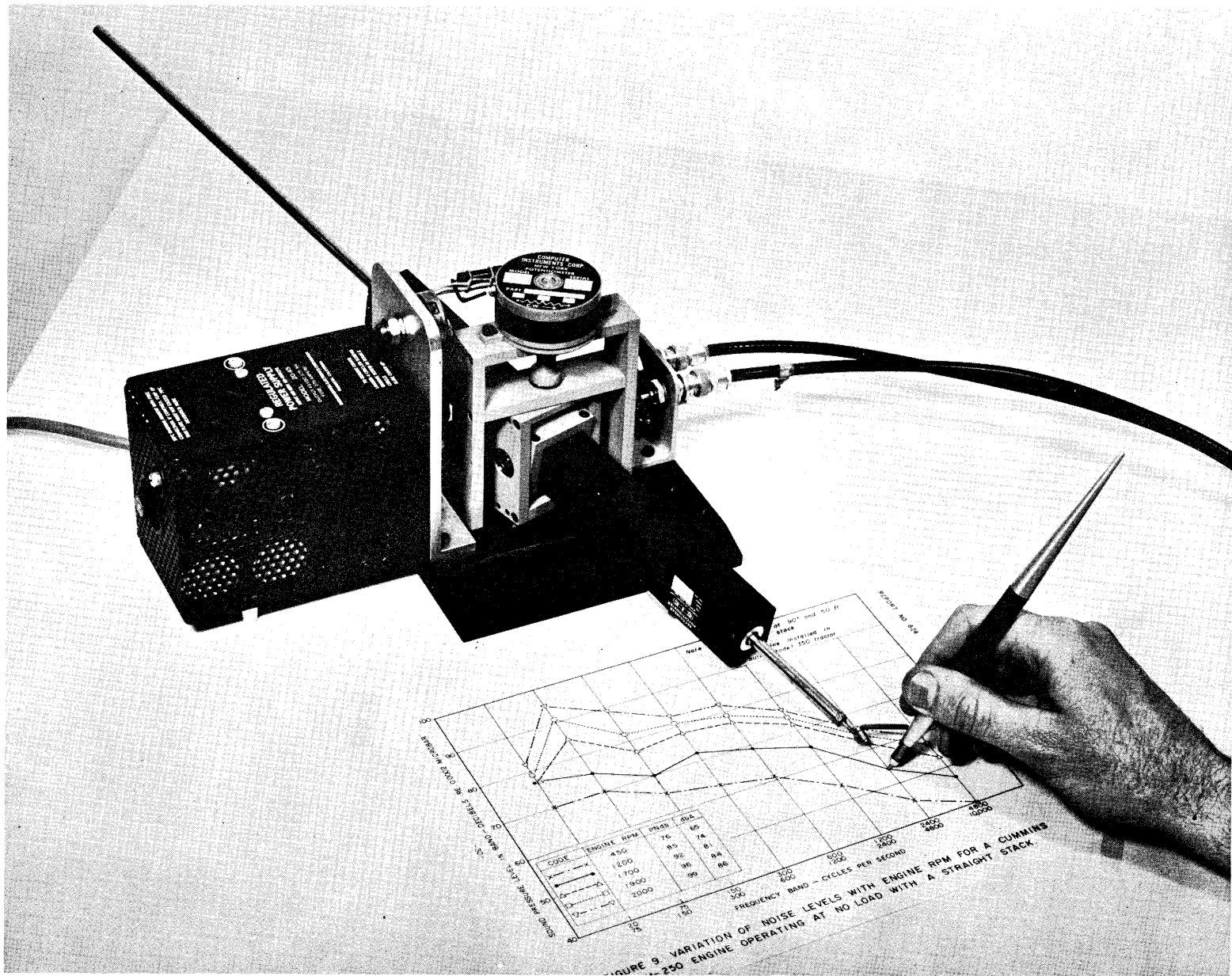
With these ends in mind the device shown in Figure 1 was constructed. A precision power supply applies a known voltage across a rectilinear and an angular potentiometer. A pen and anti-parallax assembly is attached to the end of the shaft on the rectilinear potentiometer. The rectilinear potentiometer itself is held in a gimbal mechanism which allows it to rotate left and right and move up and down. The angular potentiometer is coupled to that part of the gimbal which moves left and right. The up and down movement is not detected. It can be seen that the two voltages existing on the wiper arms of the potentiometers at any particular time will uniquely define the position of the point of the pen if it is in the same plane as the base of the gimbal mechanism. The errors caused by the point of the pen moving up and down as much as 1/2 inch are not significant when compared to other errors in the system. A block diagram of the device and its connections to a PDP-1 computer is shown in Figure 2.

The first of our additional requirements - human engineering - is well satisfied by the device shown. After the initial calibration procedure, which will be described later, the operator need only place the pen at points of interest on the sheet of paper, page of a book or magazine or report, or large drawing in a manner and with an instrument which he has been using since age three. Lines can be traced or drawn over with the same ease. The device shown meets the rest of our requirements by omission. It is simple; therefore the program can be as complicated and versatile as desired.

## PROGRAMMING FOR THE DEVICE

The programs we eventually developed worked for the particular type of data with which we had to deal. However, the problem of handling graphs of arbitrary sizes was approached in a very general fashion and the program to do this will be described next. Let us consider the input of points from x-y graphs which will fit on a single sheet of 8-1/2" x 11" paper. The device may be located with respect to the paper at any place

FIGURE 1. PHOTOGRAPH OF THE GRAPHICAL INPUT DEVICE.



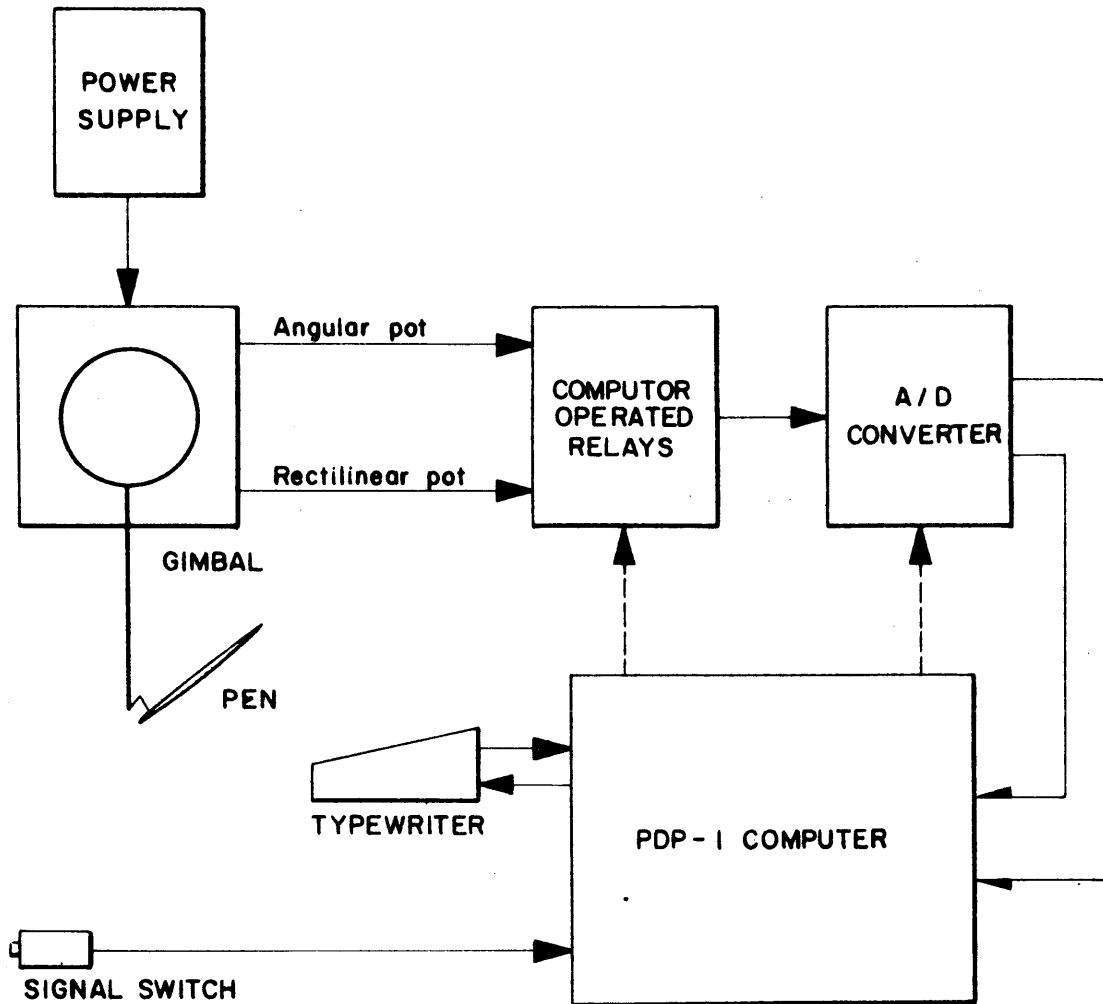
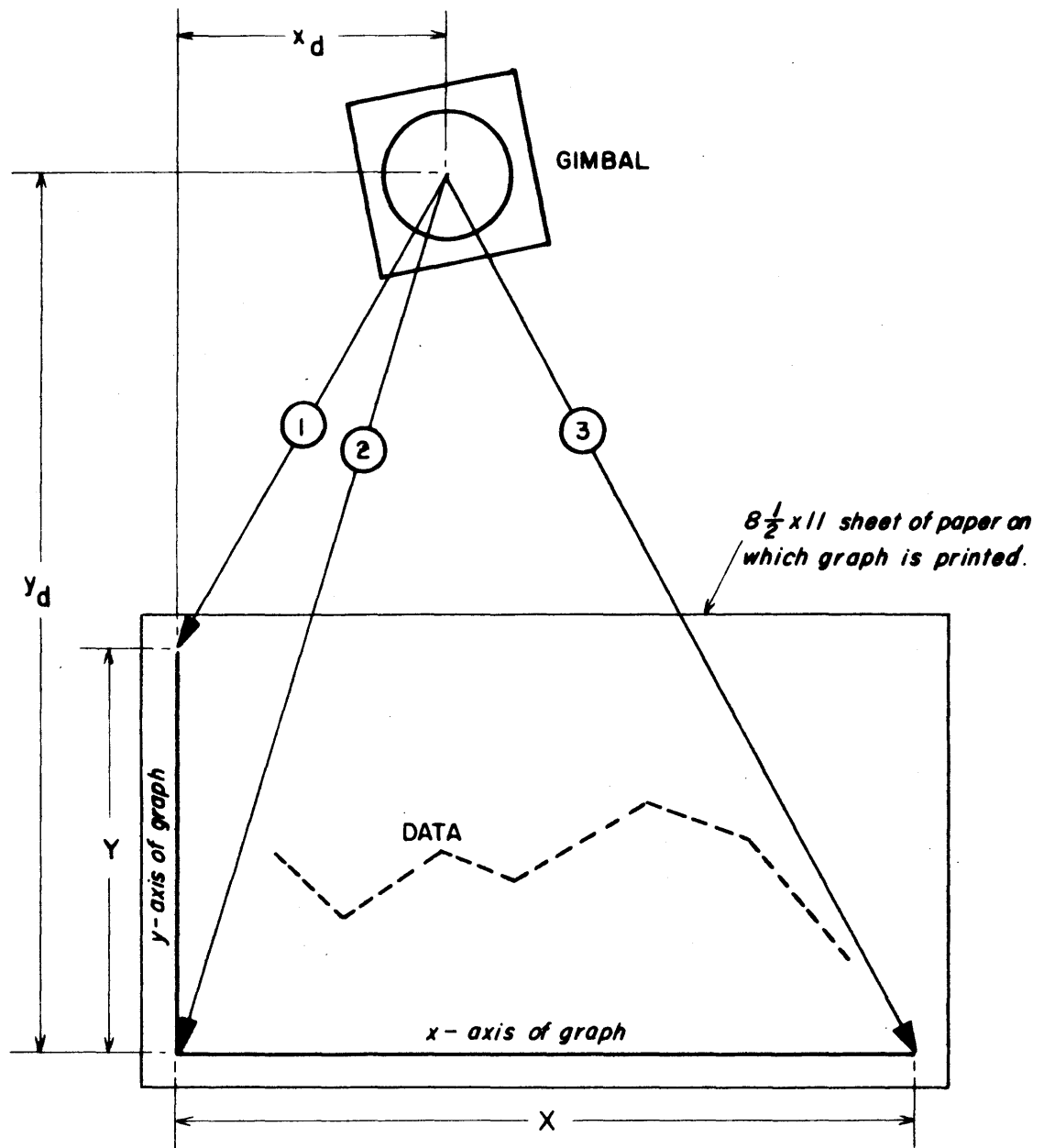


FIGURE 2. LAYOUT OF GRAPHICAL INPUT DEVICE CONNECTIONS.





Note: ①, ②, ③ show successive positions of pen point during calibration.

FIGURE 3. SCHEMATIC SHOWING THE RELATION BETWEEN THE DEVICE AND A GRAPH DURING CALIBRATION.

where the pen can reach all pertinent points on the paper. The device will remain in its position because it is relatively heavy but the paper must be clipped or taped down to remain motionless. Next, the point of the pen is placed at the following three positions in order and the computer notified of each position by use of a signal switch.

1. x minimum, y maximum (upper left corner)
2. x minimum, y minimum (lower left corner)
3. x maximum, y minimum (lower right corner)

At this point the program has six voltages to work with. They are:

1.  $A_1, B_1$  r,  $\theta$  voltages corresponding to x minimum, y maximum.
2.  $A_2, B_2$  r,  $\theta$  voltages corresponding to x minimum, y minimum.
3.  $A_3, B_3$  r,  $\theta$  voltages corresponding to x maximum, y minimum.

In addition the program has the following constants which were determined when the device was built:

1.  $K_1$  Radians/volt on the angular pot.
2.  $K_2$  Volts from pivot to electrical zero on the rectilinear pot.

Figure 3 shows the situation schematically. With the information in hand it is possible to calculate the position of the pivot point of the device in the coordinate system of the graph ( $x_d, y_d$ ) and the length of the x and y axes of the graph (X and Y). With these values calculated it is simple to convert each subsequent r,  $\theta$  voltage digitized into x and y re the graph origin. Even more useful is  $x/X$  and  $y/Y$  giving a percentage value re the axis as pointed out during the calibration.

The basic program, called xyin, performs the functions outlined above. It accepts the first three points to define the graph and then converts subsequent points to the corresponding  $x/X, y/Y$  values. Higher level programs can then form digitized representations of particular types of graphs without worrying about the absolute size of the graph, its aspect ratio, its position relative to the device, or its angle relative to the device.

In the particular data processing problem we had, the higher level programs accepted typed input describing the data, the maximum and minimum values of x and y on the graph being digitized, and scale type (log, linear, dB, etc.) and then produced a standard digital data element for subsequent processing.

## ACCURACY

When the device was initially designed the accuracy desired was  $\pm .02''$ . Since the device operates over an area of  $8\text{-}1/2'' \times 11''$  this amounts to .2% if the graph occupies nearly the whole area. The calculated error was contributed to by non-linearity in the potentiometers (.1%), drift in the power supply (.05%), mechanical play in the mounting system (mostly in the anti-parallax assembly) (.1%), and A/D conversion error (.05%). Careful measurement with high quality graph paper indicated that the calculated maximum error was realized in actuality but only if the operator was extremely precise when pointing with the pen. After about 700 graphs were digitized using the device, we gave our system an end-to-end accuracy check in the following manner. Nine copies of a particular 1/3 octave band curve were produced and three of these were digitized by each of three individuals. The nine data elements thus produced were analyzed to determine the mean, 1st percentile, and 99th percentile of the set. The results are plotted in Figure 4. It can be seen that we can expect 98% of the inputs to be within  $\pm 1/4$  dB or  $\pm .36\%$ . A Gaussian distribution of the errors about the mean was assumed.

## COST

The cost of acquiring and then running a device of this sort must be considered before discussing other possible uses. We estimate the cost considering design and production of the device, programming, and special computer equipment (A/D converter, relays, signal device) at about \$15,000. The computer time to input a typical 1/3 octave band graph (36 discrete points) was two minutes, which cost \$2.00 at our current charge-out rate. Considering that this represented the entire cost for translation from points on a piece of paper to a digital data element ready for processing, it is not too great. However, we have since realized that the immediate error checking capability which we had by being on-line is not in all cases worth the cost of being on-line. We have undertaken to design an off-line unit which will record the position information on punched paper tape. This system, outlined briefly in Figure 5, will consist of the gimbal and pen mechanism, a special purpose keyboard with foot or hand switch, a single panel of electronics for rack or table mounting and a paper tape punch. The Data Equipment Company in Santa Ana, California, is presently proceeding with production of this off-line system. The commercial version is expected to cost about \$8,500 and will be available by 1 April 1964. The program xyin will be converted to accept its input from paper tape and versions of the program prepared for computers other than the PDP-1.

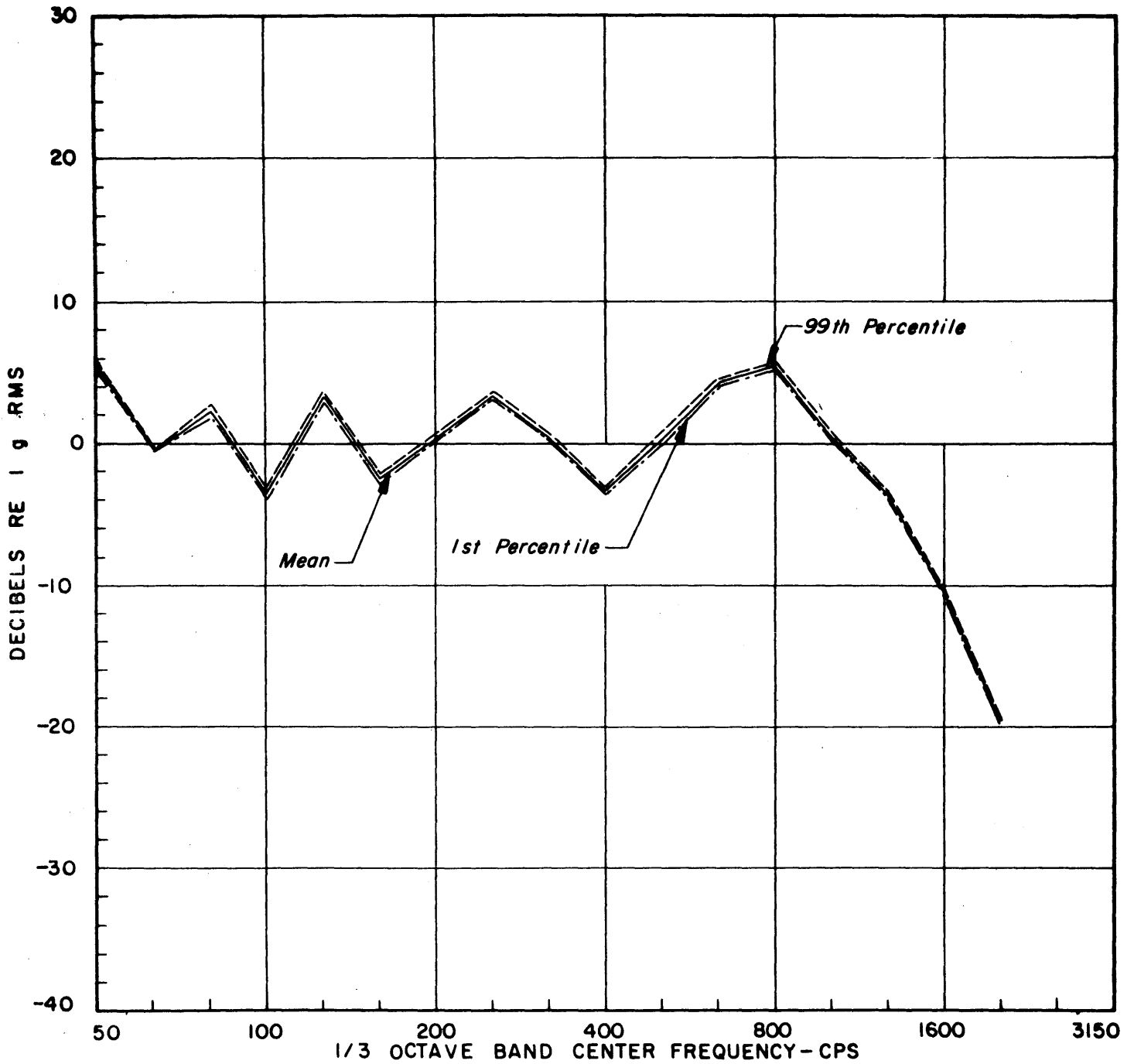


FIGURE 4. EXAMPLE OF REPEATABILITY OF GRAPHICAL DIGITIZING DEVICE FOR 1/3 OCTAVE BAND DATA.

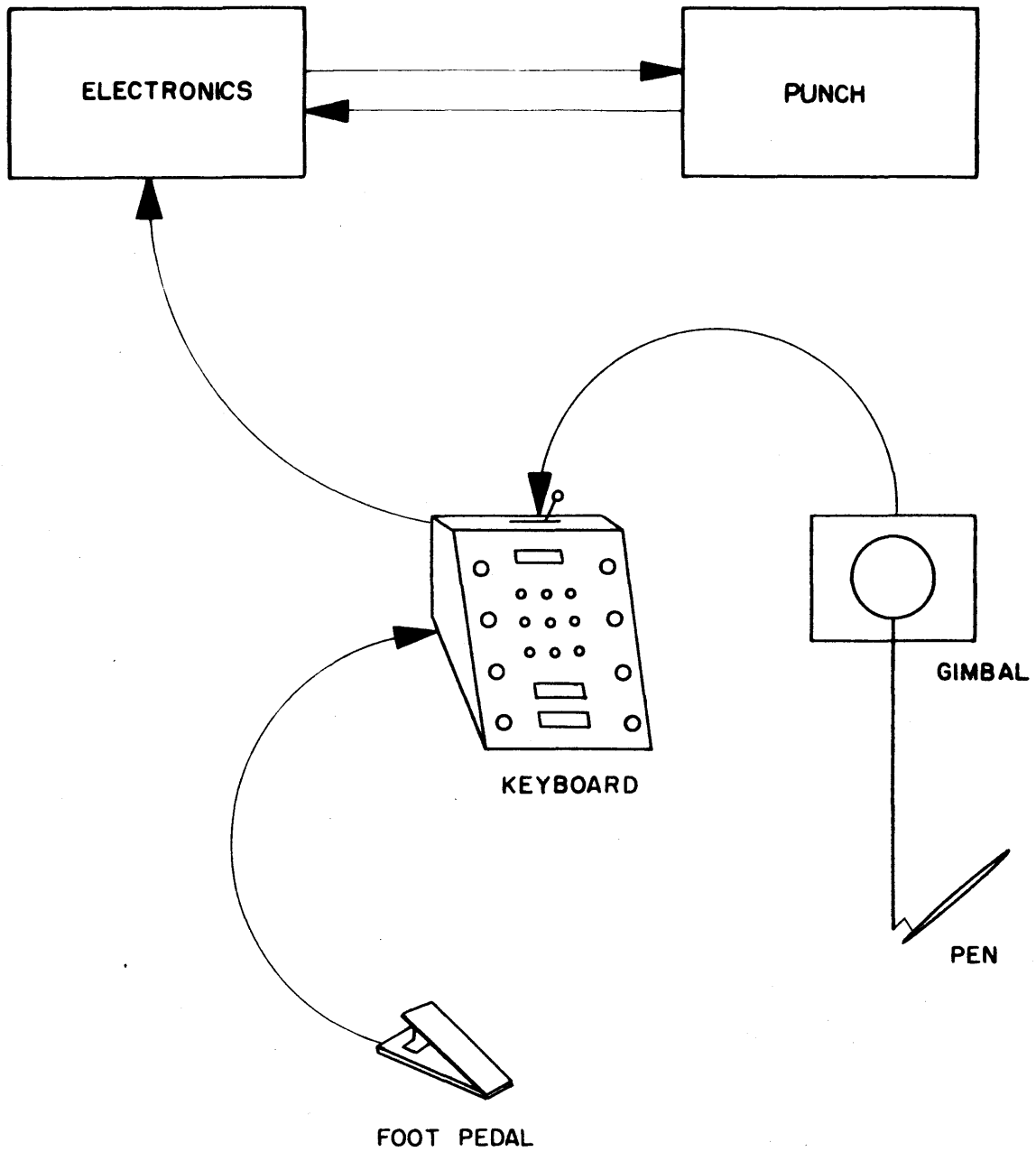


FIGURE 5. SCHEMATIC OF PARTS FOR OFF-LINE DEVICE.

## SPEED

The system we used could only digitize a maximum of 25 points per second. This restriction was caused solely by the slow operation of the relays which were used for commutating our A/D converter. The off-line system presently being constructed will be limited by the punch to about the same maximum rate. The new system will be capable of a rate of 1000 points per second if it is connected to some device which will accept data at that rate. The slow rate (25 pps) is sufficient for slowly tracing by hand a continuous curve but is not sufficient for recording a freehand sketch. The high rate (1000 pps) is sufficient for sketches and signatures. It will allow at least .02" accuracy for movement rates up to 20"/sec.

## APPLICATIONS

There are many areas where the accuracy limitations of the device described here would preclude its use and it would be necessary to fall back on a system with cursors. We have, however, used it in a situation where the accuracy was quite adequate. In the use described it is unlikely that the data we digitized was plotted in the first place with as great an accuracy as we achieved while digitizing it and even more unlikely that the measurements were anywhere near that accurate.

Before outlining possible applications it is necessary to point out another mode that is easily possible but has not yet been mentioned. The xyin program as described assumes that all of the data of interest on a particular graph exists within the 8-1/2" x 11" area over which the device can move. With relatively minor program changes the device could be used to digitize strip chart data by starting as with a single graph and then pulling the paper to the left through guides until the point which was previously x maximum, y minimum is at the extreme left. At this time two points are digitized. One is the previous x maximum, y minimum which is now at the left, and the other is the new x maximum, y minimum. This procedure could be continued until a chart many feet long was completely digitized. Another case to consider is that of a large drawing or map. Since the device is small, relatively lightweight, and designed to work on a desktop without being anchored down, there is nothing to prevent the physical movement of the device over a large drawing and then treating each small area of interest as a separate graph.

The following is a list of possible applications that have occurred to us. It is by no means claimed to be complete nor are we sure all the suggestions are really feasible.

- |   |   |
|---|---|
| 1. Maps                                       | Extraction of contour information for cut and fill calculations.  |
| 2. Strip Charts                               | Oscillograph records.   |
| 3. Graphs                                     | Stock market trends, integration or differentiation of functions.   |
| 4. Photographs                                | Nuclear tracks (bubble, spark, cloud chamber) point or circle areas of interest for subsequent more accurate investigation by other means.                |
| 5. Blueprints                                 | Encoding information from mechanical drawings for programming of automatic machine tools or production of system drawings from sets of complete drawings. |
| 6. Pantograph                                 | Scaling drawings or graphs for publication.   |
| 7. Long Distance Communication of Signatures  | Bank record checking, signing of checks or documents.   |
| 8. "Spur-of-the-Moment" Sketches and Drawings | Battlefield maps, input of curves generated on the basis of judgement or intuition to be used in a data processing problem.                               |

## CONCLUSION

The device and techniques described here have worked out quite well for our original problem. We are presently beginning to use it to input airport layouts, flight paths, and flight profiles to a program intended to evaluate projected noise levels on the ground around various airports. We have found it convenient to use and reliable and expect that it will be useful to others for similar problems of inputting graphical data.

The information contained in this paper is the private property of Bolt Beranek and Newman Inc. and is supplied as restricted information for the DECUS group. The paper and descriptions of the equipment contained therein is not to be circulated or reproduced in any form without prior approval of Bolt Beranek and Newman Inc.





THE HYBRID COMPUTATION FACILITY AT  
UNITED AIRCRAFT CORPORATION RESEARCH LABORATORIES

R. Belluardo, R. Gocht, G. Paquette

United Aircraft Corporation Research Laboratories  
East Hartford, Connecticut

**ABSTRACT:** This paper presents a description of the hybrid computation facility at United Aircraft Corporation Research Laboratories. Reasons for adopting a hybrid system are discussed. These include a reduction in time and effort required for large problem set up, and improved reliability and repeatability. The possibilities associated with having a large random access memory, (large by analog standards), and the logical capability of a general purpose digital computer as part of a large analog facility are also discussed.

Hardware contained in this facility includes a general purpose digital computer having 4,096 words of 18 bit core storage (expandable to 64,000 words). The operational speed of this computer is roughly one half that of an IBM 7090. The analog computer presently being used in this facility contains 100 operational amplifiers, 20 time division electronic multipliers, 10 quarter square multipliers, 20 diode function generators, 150 potentiometers and a Digital Output-Input Translator (DO-IT). Conversion equipment consists of an analog to digital converter, a 20 channel multiplexer with sample and hold and 10 channels of digital to analog conversion. The digital computer and conversion gear may be used in conjunction with the analog either open or closed loop. The performance of this combination on projects listed below is discussed.

Among the systems presently being studied with this facility are complete VTOL aircraft and advanced aircraft and engine systems. An aircraft simulation is frequently done in real time since certain studies involve use of a dual cockpit from which realistic inputs can be introduced by a human operator. Techniques for using the digital computer to perform static and initial condition checks on analog circuitry contained in the problem are also discussed. Check voltage values are calculated in the digital computer by means of a special program (UAC-8). These voltage values are automatically compared with those that exist in the analog computer through a special interface between the digital computer and the DO-IT system on the analog computer.



THE HYBRID COMPUTATION FACILITY  
AT UNITED AIRCRAFT CORPORATION RESEARCH LABORATORIES

R. Belluardo, R. Gocht, and G. Paquette  
United Aircraft Corporation

I INTRODUCTION

The hybrid computation facility at United Aircraft Corporation Research Laboratories was installed during the latter part of 1962 as an addition to the analog facility which then consisted of three large analog consoles and associated equipment.

As part of the analog facility, the hybrid system is presently being used to simulate complete VTOL aircraft, advanced aircraft and engines systems.<sup>1</sup> Aircraft simulation is frequently done in real time since certain studies involve the use of a dual cockpit from which realistic inputs can be introduced by a human operator. All of the systems involve the simultaneous solution of non-linear algebraic and differential equations.

The hybrid system was selected to help relieve some of the problems associated with analog solution of very large systems of equations; namely, time and effort required for large problem machine set-up, reliability and repeatability. Performance of the hybrid facility as compared to all analog simulation of a large problem is given in Section III A of this paper.

Techniques for using the digital computer to perform static and initial condition checks on analog circuitry contained in the problem are discussed in Section III B.

## II EQUIPMENT

### A) Analog Computer

The analog portion of the system is a standard moderately sized analog computer. With the expectation that the digital computer would absorb much of the algebraic calculations, less emphasis than normal was placed on non-linear equipment. Accordingly, this console contains 100 operational amplifiers, 20 time division electronic multipliers, 10 quarter square multipliers, 20 diode function generators and 150 potentiometers.

Automatic read-out and set-up features are needed. A Beckman Ease 2133 Analog Computer and DO/IT (Digital Output/Input Translator) was selected for this application.

### B) Digital Computer

The digital computer is of a rather simple design and also moderate in size. It is a single address, single instruction, stored program machine and has a word length of 18 bits which is more than consistent with analog computer accuracy and dynamic range. In a typical simulation, the digital computer executes the same sequence of instructions periodically. The digital computer samples analog variables, performs calculations using these variables, and returns answers to the analog computer. The operating speed of the digital computer is of utmost importance since all calculations must be performed in some limited time (say 1/50 of a second). A 5  $\mu$ Sec per cycle machine allows 2,000 to 4,000 instructions to be performed in 1/50 second which would indicate that a 4,000 word memory is of adequate capacity for both instructions, and stored data. Since high-speed arithmetic operations are the computer's prime concern, high speed multiply and divide capabilities are mandatory.

The input/output features of this digital computer are versatile enough to allow the tie-in of analog-to-digital and digital-to-analog converters and any control logic required. In addition to this special input/output equipment, a paper tape reader and punch and a typewriter have to date proven sufficient. The Digital Equipment Corporation PDP-1 was selected as best meeting the above requirements.

### C) The Linkage System

The linkage system can be divided into two distinct parts: the computing linkage and the slower control and set-up linkage.

Digital-to-analog and analog-to-digital converters make up the computing linkage. The emphasis here was placed on ease of utility and simplicity of design. The converters are placed under complete program control.

The digital-to-analog converters (of which there are 10) each have a 14 binary bit flip-flop buffer storage which can be loaded from the PDP-1's in/out register. A computer instruction is associated with each digital-to-analog converter. This instruction will take the contents of the in/out register and load the appropriate flip-flop buffer. The final output voltage appears at the analog patchboard as output of a decoupling amplifier. A full range scale of plus to minus 128 volts was selected.

Analog-to-digital conversion equipment consists of a 20-channel Packard Bell multiplexer, sample and hold amplifier and a multiverter. A "select channel" instruction for each of the 20 analog-to-digital channels is provided. Once the program specified channel is selected the converter is set in operation with a "convert" instruction. The PDP-1 allows several options with respect to in-out timing. In the "convert" case one option stops the program sequence until conversion is complete.

Another option allows the computer to proceed with instructions that follow. In any case when it is certain that the proper time has elapsed a "read converter buffer" instruction loads the computer's in-out register with the converted voltage. Again full scale was selected to be  $\pm 128.0$  volts.

The control linkage system allows the PDP-1 to control analog computer modes. A single instruction corresponds to each of the analog computer modes.

The Beckman Ease analog computer has the Digital Output/Input Translator. This unit allows typewriter or paper tape control over the analog computer. For example, through its use the pot settings of an analog computer program can be recorded on paper tape. This paper tape can then be used to set the potentiometers after which digital voltmeter values are read and fed to the digital computer for checking purposes. Any analog component can be selected and its output voltages read. Hardware has been provided such that any DO/IT input-output feature can now be handled by the PDP-1. Actually the system has been designed to provide control over three separate analog computers.

### III SOME EXAMPLES OF HYBRID COMPUTER USES

#### A) Hybrid Simulation of a Single Rotor Helicopter

An example of hybrid computing application is found in UAC's Single Rotor Helicopter Simulation. The simulation consists of three basic portions.

1) Total force, 6-degree of freedom fuselage dynamics with virtually unrestricted motion.

2) Rigid, hinged blade main rotor dynamics including effects of mach number and stall.

3) A fixed base, flight simulator including flight instruments and a Norden Conalog three dimensional, television display.

All dynamics and simulator intercommunications are contained in the analog portion with a minimum of non-linearities.<sup>1</sup>

The digital computer provides most of the algebraic and trigonometric operations and function interpolations. Basically, the digital program includes the following:

- 9 Bivariant functions
  - 2 Univariant functions
  - 7 Trigonometric functions
  - 2 Square root
  - 2 Two-dimensional coordinate transformations.
- Numerous term calculations.

Function interpolation<sup>2</sup> is completely digital using linear interpolation, a reasonable replacement for analog diode function generation.

The digital program is time shared between rotor and body data. Outputs occur at 175 points per second for the rotor and 35 points per second for the body.

The algebraic operations provided by the digital computer have been compared to an all analog simulation. The digital program corresponds to 150 amplifiers, 250 potentiometers, 110 variable products and 64 function curves of analog equipment. In the latter, the digital curves include 22 63-segment and 42 31-segment curves actually requiring 258 11-segment diode function generators for exact substitution. Set-up time required for this portion of the hybrid program is about two (2) minutes.

In addition to the on-line hybrid system, a digital program is being prepared to provide rapid parameter modification in both computers. Pot changes on the analog will be automated using the DO/IT linkage directly from the PDP-1.

## B) Analog Computer Symbolic Set and Debug

A group of three PDP-1 programs provide problem checks for analog computer users. The program group includes an equation loader, an interpretive mathematical and logical equation solver, and an output printer or puncher.

The digital program communicates directly with the analog computer to set potentiometers and read check values by means of the DO/IT linkage. Checks on the analog and on-line converter values are made by the digital program within user specified tolerances and error indications are printed.

The user communicates with the analog check program by paper tape or typewriter using an equation language which defines the components of his computer and their relations. These equations may be defined directly from the circuit diagram by technical aide personnel.

Analog computer components are identified by a letter and a four digit address identifying the component type, console number (up to four consoles may be included), and component address. Component types include amplifiers, function generators, servo multiplier-resolvers, electronic multipliers, trunks, and potentiometers. In addition, dummy codes for switches, relays, test voltage, etc., not read by DO/IT are available. All the above components except potentiometers may be defined by equations relating to other components. Potentiometers can be defined by numeric value only. In addition to assigning component or test values, numeric values are used to state gains, references, and function coordinates.

Mathematical or logical operations permitted include the following:

- Addition
- Subtraction
- Multiplication



Division  
Square root  
Function interpolation or extrapolation  
Trigonometric operations sine, cosine, and arctangent  
Dead zone, symmetrical or not  
Limit, symmetrical or not  
Logical term control by greater or less than test as with  
operational relays

In addition, operators are used to signal function coordinate data and non-standard comparison tolerances.

A loading program is used to enter the equations and data from typewriter or paper tape. In addition, the loader allows the user to change existing equations or add to the current check system.

An interpretive equation solving program processes all mathematically defined components and checks results with analog voltages as solutions are obtained. Equations to be solved must represent an open loop system, i.e. equations are solved sequentially, not simultaneously. The order of equations presented is irrelevant as the program automatically determines the order in which solutions may be obtained.

An output program provides typed solution values with their component codes. In addition, the output program provides the user with an optional punched or printed copy of his updated equation language.

#### REFERENCES

1. Krasnyl, The Functional Design of a Special - Purpose Digital Computer for Real - Time Flight Simulation Electronic Systems Laboratory, Final Report ESL-R-118, M.I.T., August 1961.
2. Paquette, UAC-10 Fast Bivariate Generator, Digital Equipment Computer Users Society, Decus No. 34, January 9, 1963.



## USES FOR THE PDP-1 AT LIVERMORE\*

Norman Hardy  
Lawrence Radiation Laboratory  
Livermore, California

### ABSTRACT

The PDP-1 was bought for serving as a peripheral computer to the LARC. The machine has magnetic tape, paper tape, and cards. It has a printer, typewriter and Type 30 and 31 scopes. A large majority of programs which are run on the machine are for the purpose of transforming information in one medium to another medium. The ultra precision scope in conjunction with a photo multiplier is used as a programmed scanner for transparencies. A number of small simulation programs of various types have been written for the machine. The main reason for this is the accessibility of a fast inexpensive computer with a display.

\*This paper was not submitted in time for publication.



## THE PDP-1 AS A DISPLAY MAINTAINING CONSOLE\*

Alan Kotok

Massachusetts Institute of Technology\*\*  
Cambridge, Massachusetts

### ABSTRACT

The M.I.T. (E.E.) PDP-1 is being connected to the M.I.T. Computation Center's IBM 7090 via Dataphone. Information specifying the nature of the display is passed to the PDP-1, and return information regarding the light pen is sent to the 7090. Since the Dataphone is a low capacity channel it is necessary to encode the display format. The PDP-1 then continuously displays the required points and can update the display lists by request from the 7090.

\*This paper was not submitted in time for publication.

\*\*Mr. Kotok is presently employed at Digital Equipment Corporation, Maynard, Massachusetts.

**Section V**

**APPENDIX**

## SPRING MEETING

Place: Little Theatre, Kresge Auditorium  
Massachusetts Institute of Technology  
Cambridge, Massachusetts

Date: May 3, 1963

### PROGRAM

- 0900 Registration
- 0920 Greetings - Edward Fredkin, President
- 0930 The M.I.T. Time-Sharing System - Professor J. B. Dennis, Chairman
- Hardware Provisions for Efficient Time-Sharing Operation of a PDP-1-  
Natalio Kerllenevich, M.I.T.
- An Invisible Debugging Program for a PDP-1 Timesharing System - Michael  
Wolfberg, M.I.T.
- The PDP-1 as a Display Maintaining Console - Alan Kotok, M.I.T.
- 1130 Just Nine Packages Between You and Time Sharing? - S. Boilen, L. Clapp, BBN
- 1200 Announcements
- 1230 Lunch at M.I.T. Faculty Club
- 1345 PDP-1 Scanning and Measuring of Nuclear Particle Track Photographs -  
Dr. Martin Deutsch, M.I.T.
- 1400 A Photo-Interpretive Program for the Analysis of Spark-Chamber Data -  
Harry Rudloe, BBN
- 1430 Time Sharing in the Processing of Nuclear Research Data - A. J. Ferguson,  
B. Miles, J. Leng, Atomic Energy of Canada
- 1500 Midas Assembly Program and the PDP-1 - R. Saunders, Information International Inc.
- 1530 The PDP-1 as a Versatile Research Tool - William Fahle, David Brand, Systems  
Research Laboratories, Inc.
- 1600 Requirements of a Time-Shared Computer System for Publishing Applications -  
Lawrence Buckland, Inforonics, Inc.
- 1630 PDP-1 as a Teaching Aid for Problem Solving - W. Feurzeig, BBN
- 1700 Dinner
- 1900 Time-Sharing Demonstrations at M.I.T. and Bolt Beranek and Newman, Inc.





# ANNUAL MEETING

Place: Computation Center  
Lawrence Radiation Laboratory  
Livermore, California

Date: November 18 - 19, 1963

## PROGRAM

### November 18 - Monday

- 0830 Registration and Collation
- 0930 Welcome - Joseph E. Wirsching, Lawrence Radiation Laboratory
- 0940 Opening of DECUS - Edward Fredkin, President, Information International Inc,
- 1000 Address - J. C. R. Licklider, Advanced Research Projects Agency
- 1030 Stanford Time-Sharing System - John McCarthy, Stanford University
- 1100 Report on a Large-Scale Time-Sharing System - Jules Schwartz, System Development Corporation
- 1130 Lunch at Lawrence Radiation Laboratory
- 1300 The Digigraphic Display Program for the DX-1 Computer System, John T. Gilmore, Jr., Charles W. Adams Associates, Inc.
- 1330 On-Line Input of Graphical Data - William Fletcher, Bolt Beranek and Newman, Inc.
- 1400 Recent Improvements in DDT - Marvin Minsky, M.I.T.
- 1500 Coffee
- 1520 Computer Aids to Number Theory - Malcolm Pivar, Information International, Inc.
- 1600 Signal Representation and Measurement Data Manipulation in N-Space Using an On-Line PDP System - Charlton M. Walter, AFCRL
- 1645 Announcements
- 1700 Tour of Livermore Computer Area

### November 19- Tuesday

- 0830 Registration and Collation
- 0930 The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories - R. Belluardo, R. Gocht, and G. Paquette, UAC

- 1000 A Hybrid PDP-1 for Speech Research - Douglas L. Hogan and Robert J. Scott, Department of Defense
- 1030 Uses for the PDP-1 at Livermore - Norman Hardy, Lawrence Radiation Laboratory
- 1100 M.I.T.'s Project MAC: Current Status - Richard Mills, M.I.T.
- 1130 Lunch
- 1300 Steps Toward Computer Simulation of Small Group Behavior - Dr. Thornton Roby, Tufts University and Raymond Nickerson, AFSC
- 1330 Modification of a Program Symbolic at Compile Time - John B. Goodenough, AFSC
- 1400 A Versatile Programming System for Large PDP-1 Installations - Theodore Strollo, AFCRL
- 1445 Coffee
- 1500 Microtape: Its Features and Applications - Leonard Hantman, Digital Equipment Corporation
- 1530 Flint 36 A3D - Jacob M. Baker and David J. Isenberg, Charles W. Adams Associates, Inc.
- 1600 DECUS Secretary's Report - Elsa Newman
- Introduction of New Officers
- Lewis Clapp, President (BBN)  
Elsa Newman, Secretary (DEC)
- Committee Chairmen
- Richard McQuillin, Programming (BBN)  
Joseph Lundy, Meetings (Inforonics)  
William Fletcher, Equipment (BBN)  
Elsa Newman (Mrs.), Publications (DEC)

ATTENDANCE  
SPRING MEETING

May 3, 1963

Massachusetts Institute of Technology

CHARLES W. ADAMS ASSOCIATES  
Bedford, Massachusetts

Charles Gauman  
David Isenberg  
Allen Rousseau

DEPARTMENT OF DEFENSE  
Washington, D. C.

John R. Alexander, Jr.  
Edward Benz - D  
Robert J. Scott

AIR FORCE CAMBRIDGE RESEARCH LABS.  
Bedford, Massachusetts

F. L. Barber  
Roger Bove  
Eunice Cronin  
John Mott-Smith  
E. Prange  
Theodore R. Stollo  
Charlton Walter - D  
Weiant Wathen-Dunn - D

DIGITAL EQUIPMENT CORP.  
Maynard, Massachusetts

Harlan Anderson  
Robert Beckman  
Derrick Chin  
John Koudela, Jr.  
Nick Mazzaresse  
Robert F. Maxy  
Stefan Mikulski  
Elsa Newman  
Kenneth Olsen  
George Rice

AIR FORCE SYSTEMS COMMAND  
(Electronic System Division)  
Bedford, Massachusetts

Charles R. Brown - D  
Ira Goldstein  
John B. Goodenough  
John R. Hayes  
Raymond S. Nickerson  
Robert H. Simmons  
Paul Weene  
Robert Westfield

THE FOXBORO COMPANY  
Natick, Massachusetts

Saul B. Dinman  
Gerald E. Mahoney  
David F. McAvinn

ATOMIC ENERGY OF CANADA, LIMITED  
Chalk River Canada

Brian Miles - P

HARVARD UNIVERSITY  
Cambridge, Massachusetts

A. S. Bregman  
George Miller  
Donald A. Norman

BOLT BERANEK & NEWMAN, INC.  
Cambridge, Massachusetts  
Los Angeles, California

Sheldon Boilen - P  
Lewis C. Clapp - P, D  
Wallace Feurzeig - P  
Richard McQuillin

INFORMATION INTERNATIONAL INC.  
Maynard, Massachusetts

Edward Fredkin - D  
Malcolm Pivar  
Robert Saunders - P  
James D. Wood

INFORONICS, INC.  
Maynard, Massachusetts

Lawrence Buckland - P  
Joseph T. Lundy  
William R. Nugent

ITEK CORPORATION  
Lexington, Massachusetts

W. Bivona  
Charles R. Burgess - D  
T. R. Cullen  
Richard Glantz  
Richard Hagan  
C. Hurlburt  
George Mac Ilroy  
H. P. Peterson  
Robert Rizzo

MASSACHUSETTS INSTITUTE OF TECHNOLOGY  
Cambridge, Massachusetts

R. E. Butler  
Elizabeth Campbell  
J. T. Connolly  
John B. Dennis (Professor) - D  
Martin Deutsch (Professor) - D, P  
Tom Eggers  
Richard Fredman  
Dorothy Gerety  
Natalio Kerllenevich - P  
Alan Kotok - P  
Charles Laywine  
Sanford Libman  
J. P. McKenzie  
Lars Monrad-Krohn  
Martha Pennell  
H. J. Rudloe - P  
Michael S. Wolfberg - P  
J. S. Wright

MITRE CORPORATION  
Bedford, Massachusetts  
Joseph A. O'Brien

PRINCETON-PENN ACCELERATOR  
Princeton, New Jersey

Leon Goldberg  
Grahme Salmon  
A. Richard Zacher

SYSTEMS RESEARCH LABORATORIES  
Dayton, Ohio

William A. Fahle - P

UNITED AIRCRAFT RESEARCH LABS.  
East Hartford, Connecticut

Julian D. Miller  
Gerard A. Paquette -D

WOLF RESEARCH & DEVELOPMENT CORP.  
West Concord, Massachusetts

Richard P. Gagan  
Robert D. Keim  
Sherman P. Shriber

Notes: D - DECUS Delegate  
P - Paper or Speaker

ATTENDANCE

ANNUAL MEETING

November 18-19, 1963

Lawrence Radiation Laboratory

CHARLES W. ADAMS ASSOCIATES  
Bedford, Massachusetts

John T. Gilmore, Jr. - P, D  
David Isenberg - P

ADVANCED RESEARCH PROJECTS AGENCY  
Washington, D.C.

J.C.R. Licklider (Dr.) -P

AIR FORCE CAMBRIDGE RESEARCH LABS.  
Bedford, Massachusetts

Theodore Strollo - P  
Charlton M. Walter - P, D

AIR FORCE SYSTEMS COMMAND  
(Electronic System Division)  
Bedford, Massachusetts

John B. Goodenough - P  
Raymond S. Nickerson - P

AIR FORCE TECHNICAL APPLICATIONS CENTER  
Washington, D. C.

John Davidson

BECKMAN INSTRUMENTS  
Fullerton, California

Frank Ingram - D

BOLT BERANEK AND NEWMAN, INC.  
Cambridge, Massachusetts  
Los Angeles, California

Lewis C. Clapp - D  
William E. Fletcher - P, D  
Alice K. Hartley

DEPARTMENT OF DEFENSE  
Washington, D. C.

Robert J. Scott - P

DIGITAL EQUIPMENT CORPORATION  
Maynard, Massachusetts

Robert Beckman  
Leonard Hantman - P  
Kenneth Larsen  
Nick Mazzaresse  
Elsa Newman  
Stanley Olsen

DOUGLAS AIRCRAFT CORPORATION  
Santa Monica, California

James A. Fetherlin  
James S. Morison

EDGERTON, GERMESHAUSEN & GRIER  
Las Vegas, Nevada  
Santa Barbara, California

David A. Haas  
Brian Glusovich  
Thomas Wilson  
George Woodmansee

INFORMATION INTERNATIONAL INC.  
Maynard, Massachusetts

Edward Fredkin  
Malcolm Pivar - P  
Robert Saunders

INTERNATIONAL TELEPHONE &  
TELEGRAPH (Information Systems  
Division)  
Paramus, New Jersey

Albert M. Loshin  
Jack Tauber

ITEK CORPORATION  
Lexington, Massachusetts  
Charles Burgess - D

LAWRENCE RADIATION LABORATORY  
Livermore, California

R. P. Abbott  
Fraser Bonnell - D  
James E. Braley  
Donald Cooper  
Raymond DeSaussure  
David C. Evans  
Dr. S. Fernbach  
J. V. Franck  
Norman Hardy - P  
G. D. Hornbuckle  
Michael G. Hurley  
Robert M. Lee  
W. Wayne Lichtenburger  
Rudolph S. Langer  
Loyd Mish  
Melvin W. Pirtle  
Arthur Rosenberg  
Stephen R. Russell  
Joseph C. Sharp  
Alex Tschekaloff  
Donald Waterman  
Willard H. Wattenburg  
Joseph E. Wirsching

MASSACHUSETTS INSTITUTE OF  
TECHNOLOGY  
Cambridge, Massachusetts  
Richard Mills -P, D  
Marvin Minsky (Professor) - P  
John E. Ward

STANFORD UNIVERSITY  
Stanford, California  
John McCarthy (Dr.) - P

SYSTEM DEVELOPMENT CORP.  
Santa Monica, California  
Clayton E. Fox  
Arthur M. Rosenberg  
Jules I. Schwartz - P

UNITED AIRCRAFT CORPORATION  
East Hartford, Connecticut  
Ralph Belluardo - P

WOLF RESEARCH & DEVELOPMENT CORP.  
West Concord, Massachusetts  
Richard P. Gagan

Note: D - DECUS Delegate  
P - Paper or Speaker

## AUTHOR AND SPEAKER INDEX

		<u>Page</u>
Baker, J. M.	"Flint 36 A3D" .....	61
Belluardo, R.	"The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories" .....	261
Boilen, S.	"Just Nine Packages Between You and Time Sharing?" .....	11
Brand, D.	"The PDP-1 as a Versatile Research Tool" .....	75
Buckland, L.	"Requirements of a Time-Shared Computer System for Publishing Applications" .....	201
Clapp, L. C.	"Just Nine Packages Between You and Time Sharing?" .....	11
Deutsch, M.	"PDP-1 Scanning and Measuring of Nuclear Particle Track Photographs" .....	89
Edwards, D. J.	"Recent Improvements in DDT" .....	41
Fahle, W.	"The PDP-1 as a Versatile Research Tool" .....	75
Ferguson, A. J.	"Time Sharing in the Processing of Nuclear Research Data" .....	83
Feurzeig, W.	"PDP-1 Computer as a Teaching Aid for Problem Solving" .....	203
Fletcher, W. E.	"On-Line Input of Graphical Data" .....	249
Gilmore, J. T., Jr.	"The Digigraphic Display Program for the DX-1 Computer System" .....	107
Gocht, R.	"The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories" .....	261
Goodenough, J. B.	"Modification of a Program Symbolic at Compile Time" .....	51
Hantman, L.	"Microtape: Its Features and Applications" .....	221
Hardy, N.	"Uses for the PDP-1 at Livermore" .....	271
Hogan, D. L.	"A Hybrid PDP-1 for Speech Research" .....	183
Isenberg, D. J.	"Flint 36 A3D" .....	61
Kerllenevich, N.	"Hardware Provisions for Efficient Time Sharing of a PDP-1 Computer" .....	217

		<u>Page</u>
Kotok, A.	"The PDP-1 as a Display Maintaining Console".....	273
Leng, J.	"Time Sharing in the Processing of Nuclear Research Data" .....	83
Licklider, J. C. R.	"Command of Procedures" .....	i
McCarthy, J.	"Stanford Time-Sharing System" .....	13
Miles, B.	"Time Sharing in the Processing of Nuclear Research Data" .....	83
Mills, R.	"M.I.T.'s Project MAC: Current Status" .....	33
Minsky, M.	"Recent Improvements in DDT" .....	41
Nickerson, R. S.	"Steps Toward Computer Simulation of Small Group Behavior" .....	139
Paquette, G.	"The Hybrid Computation Facility at United Aircraft Corporation Research Laboratories".....	261
Pivar, M.	"Computer Aids to Number Theory".....	193
Roby, T.	"Steps Toward Computer Simulation of Small Group Behavior" .....	139
Rudloe, H.	"PIP: A Photo-Interpretive Program for the Analysis of Spark-Chamber Data" .....	91
Saunders, R.	"The Midas Assembly Program and the PDP-1" .....	71
Schwartz, J.	"Report on a Large-Scale Time-Sharing System" .....	19
Scott, R. J.	"A Hybrid PDP-1 for Speech Research" .....	183
Stollo, T.	"A Versatile Programming System for Large PDP-1 Installations".....	57
Walter, C. M.	"Signal Representation and Measurement Data Manipulation in N-Space Using an On-Line PDP System" .....	131
Wolfberg, M.	"An Invisible Debugging Program for a PDP-1 Time-Sharing System" .....	43