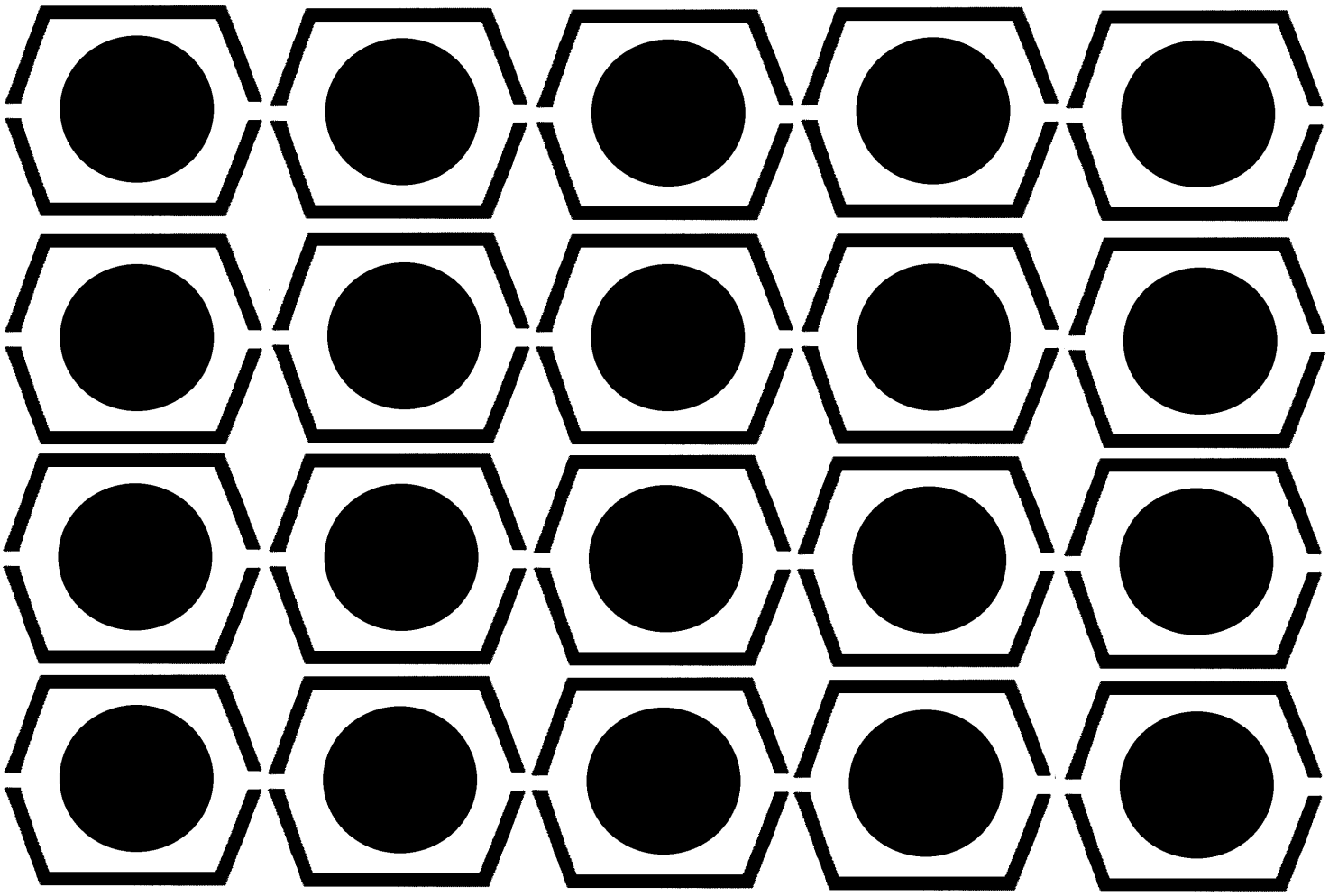


DECUS PROCEEDINGS

FALL 1966

PAPERS AND PRESENTATIONS

of

The Digital Equipment Computer Users' Society

Maynard, Massachusetts

FALL
1966

PAPERS AND PRESENTATIONS

of

The Digital Equipment Computer Users Society

FALL SYMPOSIUM

November 4, 5, 1966

Lawrence Radiation Laboratory

Berkeley, California

CONTENTS

* Abstracts Only

PREFACE

The Fall 1966 Proceedings brings to the reader the papers presented at the Fall Symposium of Digital Equipment Computer Users Society. The meeting was hosted by Lawrence Radiation Laboratory of Berkeley, California, and was held on November 4 and 5, 1966.

This meeting brought to fourteen the number of seminars held by the users group in its five-year history. DECUS meetings are a forum for users to discuss their applications and air problems and ideas informally as well as through formal papers.

Papers published in this volume have been printed as received from authors with no editorial changes. In that way, publication of the Proceedings is hastened considerably. In some cases, papers were not received in time for publication and abstracts of these papers have been substituted. If the omitted papers are at some time submitted to the users group, they will be published in the newsletter, DECUSCOPE. Reprints of papers presented here are available from the DECUS office, Maynard, Massachusetts 01754.

This Proceedings also contains a list of meeting attendees, the program, and an author/speaker index.

Our special thanks to all participants and to LRL for providing excellent meeting accommodations. Special thanks to Anthony Schaeffer and Lee Davenport of LRL for all their help in coordinating arrangements.

Angela J. Cossette (Mrs.)
DECUS Executive Secretary

WELCOME ADDRESS

Winston R. Hindle, Jr.
Digital Equipment Corporation
Maynard, Massachusetts

I am delighted to bring greetings to this DECUS symposium from all of my colleagues at Digital. It has been a real thrill for us to see the growth of this organization during its five-year history. The caliber of the papers to be presented during these two days is testimony enough to the technical level that has been attained.

The relationship between a computer manufacturer and the society of its users is a difficult one, and few users groups have really prospered. There are so many barriers to good communications that both sides are slightly off balance all the time. DECUS, I think, has solved a number of the usual problems by having a status quite independent of the company and by having a group of interested and capable officers and committee chairmen who have been willing to spend time making it work. Within DEC, we have encouraged this through support of a capable Executive Secretary and her staff, through DECUSCOPE and through the program reproduction facilities. But it has been the energy of the users that has really succeeded in building a healthy organization.

Most of the communications during the next two days will be between users, so I thought it would be productive for a few minutes to talk about some internal functions at DEC which affect you, the users, most directly. Some relatively recent organization moves that DEC has made have served to regenerate our responsiveness to outside influences, be these outside inputs ideas for new machines, suggestions for new applications, specific software or hardware complaints or even an occasional word of praise. First, the key organizational tenet that we have followed is to divide the firm's resources on a product-by-product basis. Thus, each product group has a known set of resources and can set its plans accordingly. In return for these resources, each group accepts full responsibility for the product in every regard - from profits to production rates to customer satisfaction. This "product line" type of organization has been in effect for 18 months and has proved invaluable internally. Performance can be closely gauged and problems easily pin-pointed.

An annual plan is developed by each product group. The centralized resources of the company (production, drafting, personnel, etc.) are budgeted through agreements between the managers of each central function and the product managers. The task is then to match performance against the plan and make any adjustments that are needed as the year progresses. Where this process most closely affects those of you here is the programming function-which is now one of the centralized professional groups within DEC. The Programming Department, which had previously been fragmented, was combined as a unified group six months ago. It is directed by Larry Portner, DEC's Programming Manager. The Department performs four major tasks: systems programming, diagnostic programming, software quality control, and program distribution. Each of these four divisions performs its work for all products according to the yearly plan.

The quality control function in software development is relatively new for DEC. The concept here is that before a program is added to the DEC library, the program and programming documentation are checked by someone not involved in their development. This procedure has now been in effect for several months, so the results of this careful screening have not been seen in the field as yet, with the exception of a few recent PDP-6 programs. However, the concept is sound and the outcome will inevitably increase user satisfaction. In fact, the entire reorganization of the programming development group is quite specifically a DEC response to customer inputs on software.

We feel that the product-oriented organization and the new Programming Department are both developments that allow DEC to expand gracefully in the near future. Organizations, however, cannot be fixed and forgotten. So we will undoubtedly continue to make alterations from time to time.

The output of our product-oriented structure can be partly judged by the new DEC products that have been introduced in the last few months - PDP-8/S in July, PDP-9 in August and PDP-10 in November. Another measure is sales and profits - for the fiscal year 1966 sales increased 52% and profits 163%. The results of this first quarter of the current fiscal year show continued increases.

Of course, the ultimate test of DEC or any other company is the reaction of its users over a period of years, and that brings us back full circle to my opening point. We need and want your input, your criticism, and hopefully your commendation and continued support. The vigor of DECUS is good indication of your needs and interests and we will, of course, do our part in its support. Let us make even greater use of this forum and this organization for the exchange of information and helpful advice.

ESI -- CONVERSATIONAL-MODE COMPUTING ON THE PDP-8/S

David J. Waks
Applied Data Research, Inc.
Princeton, N. J.

Abstract

An important problem in the computer business is that of "bringing the computer to the user." This paper discusses the problems involved in the usual solution to this problem -- time-sharing -- and presents, as an alternative solution, a system called ESI, which provides conversational-mode programming on a minimum PDP-8/S.

Introduction

Many people have computational problems which could most economically be solved with digital computers, yet which are still solved by pencil-and-paper or desk-calculator techniques, at a high cost in time and convenience.

Three major barriers exist to putting these problems on computers: (1) The time required to bring them to the computer, wait for them to be run, and bring back the results is intolerable; (2) The problems are of a scale for which the time spent in explaining them to a programmer is large compared to the time required to solve them by hand; and (3) The man with the problem, who should, therefore, write the program himself, is not a programmer and doesn't want to have to become one to solve his problems. This problem is referred to as being that of "bringing the computer to the user," and is one of the most important problems in the computer business today.

Any solution to this problem requires two prongs: Hardware to physically place a computing facility at the user's disposal; and Software to make the computing system approachable by providing an interface with the hardware.

Time-Sharing as a Solution

It is widely believed that the best solution is to share a large computer among many users, in order to reduce the cost per user of the hardware and the software. This requires the construction of an elaborate hardware system, and, typically, an even more elaborate software system largely devoted to solving the problems imposed by the complex hardware. Such systems physically bring the computer to the user by giving him a console as close as he wants it.

The software, on the other hand, leaves him as far from the computer as ever before. This is because most such software systems are "open" in the sense that the entire computer system is available to the user -- in fact, much of the hardware/software design effort is devoted to maintaining this illusion for the user. The trouble is that only an experienced programmer can really use these "open" systems. The last thing the true user, as opposed to programmer, wants or

needs is to have to learn about a complex hardware system and an even more complex software system (in most systems he must read three or four manuals just to be able to write a FORTRAN program and get it running). The inexperienced user desperately needs to be isolated from the hardware and software. He must have an environment expressly created for his use in solving his problems.

The first time-shared system specifically directed toward the totally inexperienced user was the Rand Corporation's JOHNNIAC Open-Shop System (JOSS*). The major goals in the development of JOSS were: (1) it should be easily approachable by somebody completely inexperienced in computer programming; (2) it should be largely self-teaching; i.e., once a user has some basic knowledge, the use of the system should teach him the rest; (3) the user must be completely isolated from the computer; i.e., he neither knows nor cares about what's going on inside; (4) the user must be protected against his own errors, so that, for example, there is no way for him to inadvertently destroy his program or the system; (5) there should be a minimum number of arbitrary conventions which the user must learn. JOSS was implemented on the JOHNNIAC computer, and provided its facilities to up to eight users simultaneously.

An Alternate Solution

A completely different solution to the physical problem, suggested by the availability of computers in the PDP-8 class, is to give the user an entire computer for his own use. This is quite attractive if the computer is priced low enough, since the cost of a console in a time-shared system, including communications and hardware on the large computer, is comparatively high (about $13 an hour or so). The lease price of a small computer, available 24 hours a day, might well be less than the price for two-hours-a-day use of a console in a time-shared system. The problem, however, is whether a user-oriented software system can be built for a small computer.

When the author first learned of DEC's plans for the PDP-8/S, he became interested in the feasibility of implementing a JOSS-like system which would operate on a minimum PDP-8/S. The result of the feasibility study is a program, the Engineering and Scientific Interpreter (ESI), which provides most of the facilities of JOSS to a single user on a minimum PDP-5, PDP-8, or PDP-8/S.

ESI is designed as a self-contained system which greatly simplifies the programming and debugging of modest-scale programs. The system is highly interactive with the user, making the transitions from design to coding to debugging to operation of programs very smooth and continuous. The user can try something out immediately, see if it works, then discard it and try something else if it doesn't, or build on it if it does. The system is very easy to learn to use, taking less than an hour in most cases. The rules for construction of programs are very concise, and are summarized on one page.

A Description of ESI

As in JOSS, ESI statements have the form of English imperative sentences. Every ESI statement contains a main clause which starts with an imperative verb (such as TYPE, DELETE, SET, GO) and ends with a period. Frequently there are parameters between the verb and the period -- such as TYPE X. Any clause which does not involve the

* TM, The Rand Corporation

transfer of control can be preceded by one or more iterative clauses which begin with the word FOR and end with a comma. Any statement, including one which contains one or more FOR clauses, may be preceded by a conditional clause which begins with the word IF and ends with a comma. Thus, the following is a perfectly legal statement:

```
IF X = Y, FOR I = 1(1)N,
FOR J = 1(1)N, TYPE A[I,J].
```

The following pages constitute the User's Manual for ESI. Two pages of "conversations" with ESI, plus explanatory notes on those conversations, constitute an introductory primer; the last page, Permissible Forms in ESI-B, is a concise summary which should be the only manual necessary after starting to use ESI.

In reading the "conversations" with ESI, the reader should look at the notes only if he doesn't understand the statements in the "conversation." Much more can be learned by trying to figure out what's happening than by always referring to the notes.

SAMPLE CONVERSATIONS WITH THE ENGINEERING AND SCIENTIFIC INTERPRETER
DAVID J. WAKS, APPLIED DATA RESEARCH, PRINCETON, N.J.

I. SAMPLES OF DIRECT STATEMENTS:

```
←DELETE ALL.  (1) (2)
←TYPE 2+2.                (3)
      2+2 =      4
←SET X = 5.
←TYPE X.
        X =      5
←TYPE X↑3, X/7, SGN(X).
      X↑3 =    125
      X/7 =      .7142857  (4)
   SGN(X) =      1         (5)
←TYPE X*Y.
ERROR ABOVE: UNDEFINED  (6)
←SET Y = -7
ERROR ABOVE: EH?  (7)
←SET Y = -7.
←TYPE XY.
ERROR ABOVE: EH?  (8)
←TYPE X*Y.
      X*Y =    -35
←SET A = 3E-40.  (9)
←SET B = A↑2.
ERROR ABOVE: EXPONENT  (10)
←SET C = 0.
←TYPE A/C.
ERROR ABOVE: 0 DIVISOR  (11)
←FOR I = 2(3)5, TYPE I, I↑3.  (12)
        I =      2
      I↑3 =      8
        I =      5
      I↑3 =    125
←SET A[1] = 3.
ERROR ABOVE: SUBSCRIPT  (13)
←DELETE A.  (14)
←SET A[1] = 3.
←FOR R=4(2)10, SET A[R/2] = R↑2.  (15)
←TYPE ALL VALUES.  (16)

      A[1] =      3
      A[2] =     16
      A[3] =     36
      A[4] =     64
      A[5] =    100
         C =      0
         I =      8
         R =     12   (17)
         X =      5
         Y =     -7

←IF X = 5, TYPE "YES".
YES
←IF I↑2 < 10*R, TYPE "THAT'S RIGHT".
THAT'S RIGHT
←DELETE ALL.
←
```

II. USE OF INDIRECT STATEMENTS TO COMPUTE THE AREA AND CIRCUMFERENCE
    OF A CIRCLE, GIVEN ITS RADIUS.

```
←DELETE ALL.
←1.1 DEMAND R.    (18)
←DO STEP 1.1.     (19)
R = ←3            (20)
←1.2 SET A = P*R↑2.
←1.3 SET C = 2*P*R.
←1.4 TYPE "IF RADIUS IS"R", THEN AREA IS"A", AND CIRCUMFERENCE IS"C.  (21)
←DO PART 1.    (22)
R = ←2
ERROR IN STEP 1.2: UNDEFINED    (23)
←P=3.1415927    (24)
←DO PART 1.
R = ←2
IF RADIUS IS 2, THEN AREA IS 12.56637, AND CIRCUMFERENCE IS 12.56637
←1.5 TO STEP 1.1.    (25)
←DO PART 1.
R = ←3
IF RADIUS IS 3, THEN AREA IS 28.27434, AND CIRCUMFERENCE IS 18.84956
R = ←4
IF RADIUS IS 4, THEN AREA IS 50.26549, AND CIRCUMFERENCE IS 25.13274
R = ←17
IF RADIUS IS 17, THEN AREA IS 907.9204, AND CIRCUMFERENCE IS 106.8142
INTERRUPTED IN STEP 1.4     (26)
←TYPE R, C.
          R =     17
          C =    106.8142
←GO.  (27)
R = ←1/P  (28)
IF RADIUS IS .3183099, THEN AREA IS .31831, AND CIRCUMFERENCE IS 2
INTERRUPTED IN STEP 1.4
←CANCEL.    (29)
CANCELLED
←DELETE STEP 1.1.    (30)
←2.1 FOR R = 10(5)20, DO PART 1.
←DO PART 2.
IF RADIUS IS 10, THEN AREA IS 314.1593, AND CIRCUMFERENCE IS 62.83186
ERROR IN STEP 1.5: STEP #    (31)
←DELETE STEP 1.5.
←DO PART 2.
IF RADIUS IS 10, THEN AREA IS 314.1593, AND CIRCUMFERENCE IS 62.83186
IF RADIUS IS 15, THEN AREA IS 706.8584, AND CIRCUMFERENCE IS 94.24779
IF RADIUS IS 20, THEN AREA IS 1256.637, AND CIRCUMFERENCE IS 125.6637
←TYPE ALL.    (32)

 1.2 SET A = P*R↑2.
 1.3 SET C = 2*P*R.
 1.4 TYPE "IF RADIUS IS"R", THEN AREA IS"A", AND CIRCUMFERENCE IS"C.

 2.1 FOR R = 10(5)20, DO PART 1.

          A =  1256.637
          C =   125.6637
          P =     3.141593
          R =    25

←DELEKE←←TE ALL.    (33)
←
```

Notes on the Conversations

(1) A back arrow (←) is typed whenever ESI wants the user to "say" something. Thus any line beginning with a back arrow was typed by the user; any without the back arrow, by the computer.

(2) "DELETE ALL" commands ESI to clear user storage of everything associated with the preceding user program.

(3) "TYPE" commands the evaluation and typing out of one or more arithmetic expressions.

(4) All results are stored and presented as decimal numbers with exactly seven decimal digits of precision.

(5) The SGN function is the "sign" or "signum" function of mathematics.

(6) A variable, such as "Y" here, which has not been set to any value is considered to be "undefined" and any use of it in an arithmetic expression is flagged as an error.

(7) Every statement in ESI is an English sentence, and must end in a period.

(8) This command is meaningless, since "XY" is not a valid name for a variable (the only valid names are the single letters A through Z), and the multiplication sign is missing if the intention was to evaluate the product of "X" and "Y".

(9) The single letter "E" means "times ten to the." Thus 3E-40 is ESI's notation for 3×10^{-40}.

(10) All numbers stored by ESI must be in the range of 10^{-63} to 10^{63}. The number "A↑2" is out of this range.

(11) ESI treats any attempt to divide by zero, including Ø/Ø, as an error.

(12) In this iterative statement, "I" will take on values beginning with 2, in increments of 3, until 5; i.e., 2 and 5.

(13) "A" has already been assigned a value as an unsubscripted variable. It cannot simultaneously be subscripted and unsubscripted.

(14) "DELETE" commands ESI to make the variable "undefined." It can now be used with a subscript. Note that no declaration (such as DIMENSION) is required before using a variable with subscripts.

(15) A more complicated example of subscripting; the subscript expression is R/2.

(16) "TYPE ALL VALUES" commands ESI to type out the values of all defined variables.

(17) Note that the values of "I" and "R", after the completion of iterative statements involving them, are not the terminal values specified by the FOR statement.

(18) If a number is typed preceding a statement, the statement is not executed immediately, but is stored away associated with that number, called its "step number."

(19) "DO STEP" commands the execution of a previously stored step.

(20) "DEMAND" (here being executed by step 1.1) requests from the user a value for the indicated variable (in this case "R"). Note that the back arrow (←) again indicates that ESI has given control of the Teletype to the user.

(21) Previous "TYPE" statements illustrated the typing of evaluated arithmetic expressions (TYPE X*Y.) and character strings (TYPE "THAT'S

RIGHT".). This example illustrates how a single TYPE statement may mix these two forms. Character strings in quotes alternate with arithmetic expressions (in this case, note that R, A, and C are very trivial arithmetic expressions).

(22) The digit preceding the decimal point in a step number is called the part number, and must be in the range of 1 to 9. All steps with the same part number are considered to be a part of a program. The "DO PART" command causes each step of the part to be executed in step number order. At the end of execution of the last step (the one with the highest numerical step number), control returns to the step following the "DO PART" if it was executed from an indirect (stored) statement or to the user if it was executed as a direct statement (from the Teletype).

(23) We forgot to give a definition for P. Note that the error was not detected until the statement was executed.

(24) This illustrates an abbreviated form of the "SET" statement which can only be used as a direct statement (executed immediately rather than being stored away). P is, of course, π.

(25) "TO STEP" causes the designated step to be executed next, rather than the next step in step number order. Note that step 1.5 is inserted following step 1.4, causing the entire program to be a five-step loop.

(26) At any time during the execution of a program, the "ALT MODE" key on the Teletype can be typed. At the end of execution of the current step, the execution of the program is interrupted and an appropriate message is typed. When execution is suspended, the only legitimate statements are GO, CANCEL, and TYPE.

(27) "GO" commands that the interrupted program be continued where it left off.

(28) The user may type any arithmetic expression in response to a "DEMAND" but must make sure that only defined variables are used in the expression.

(29) "CANCEL" commands that the execution of the program be cancelled. It cannot now be continued with GO.

(30) At this point, we have decided to evaluate the area and circumference for a given range of values of the radius. We therefore delete the DEMAND statement and insert an iterative statement as part 2.

(31) We forgot to delete step 1.5, which now refers to a non-existent step number.

(32) "TYPE ALL" commands the typing of everything currently stored by ESI -- that is, all steps and all values.

(33) The user can type "RUBOUT" to delete a typing error. Each "RUBOUT" deletes one character and types a back arrow to indicate this. Any number of characters can be deleted this way; if "RUBOUT" is typed with the input line completely empty (all characters rubbed out), the bell is rung.

PERMISSABLE FORMS IN ESI-B:

DIRECT OR INDIRECT:

```
SET C = A*B+C*D.
SET C[I,J] = B[I-1,J↑2]-C[I+1,J/2]
SET Y = IP(X/I).
FOR I = 1(1)N, SET A[I] = B[I]*C[I].

DO PART 3.
FOR R = Ø(Ø.1)1.5, DO PART 2.
DO STEP 3.7.
FOR J = N(-1)1, DO STEP 7.352.

TYPE 2+3+5.
TYPE X.
TYPE X, IP(X), SGN(X), ABS(X).
FOR I = 1(1)N, TYPE A[I].
TYPE "THIS IS A STRING".
TYPE "THE SQUARE OF" X "IS" X↑2.
TYPE STEP 2.3.
TYPE PART 6.
TYPE ALL PARTS.
TYPE ALL VALUES.
TYPE ALL.

DELETE X.
DELETE A[1,3], B[I,J], C, D.
DELETE ALL VALUES.
FOR I = 1(1)N, DELETE A[I].

LINE.
```

DIRECT ONLY:

```
DELETE STEP 1.1.
DELETE PART 2.
DELETE ALL PARTS.
DELETE ALL.

X = 3*ABS(A-B)
Y=Z+R
Z = 14
A[1]=1.3E-6
A[-43] = 1.414E+32
A[2] = 3175*I
```

INDIRECT ONLY:

```
1.1 TO STEP 1.7.
1.7 TO PART 4.

2.3 END.

4.1 STOP.

6.1 DEMAND X.
7.35 DEMAND A[I,J].
8.1 DEMAND A[45].
```

CONDITIONAL CLAUSES:

```
IF A = B,
IF ABS((N-O)/N) < 1E-6,
IF IP(X) GE IP(Y),
IF SGN(X) NE 1,
IF (A-B)/C LE D-X↑2,
```

NUMBERS:

```
2
3.141593
.ØØ3
Ø.Ø1
-3.7E5
4.36E-7
-3.273E+17
```

OPERATIONS:

```
+  -  *  /  ()  ↑
```

RELATIONS:

```
< > = GE LE NE
```

FUNCTIONS:

```
IP(X)     INTEGER PART
FP(X)     FRACTION PART
SGN(X)    SIGN PART
ABS(X)    ABSOLUTE VALUE
```

INTERRUPTED OR STOPPED:

```
ANY "TYPE" STATEMENT.
GO.
CANCEL.
```

← IS TYPED AND THE BELL RINGS WHENEVER A USER TYPE-IN IS REQUESTED.
[AND] ARE USED TO DENOTE SUBSCRIPTS.
? TYPED AT THE END OF ANY LINE CAUSES IT TO BE DISREGARDED.
"RUBOUT" DELETES THE PRECEDING CHARACTER AND TYPES ← TO SO INDICATE.
STEP NUMBERS ARE IN THE RANGE 1 TO 9.999999.
VARIABLES ARE THE SINGLE LETTERS A THROUGH Z.
"ALT MODE" INTERRUPTS EXECUTION OF A PROGRAM AT THE COMPLETION OF THE CURRENT STEP; ON A "DEMAND", "ALT MODE" CANCELS EXECUTION.

THE BBN ON-LINE PROGRAMMING SYSTEM

Nancy E. Lambert
Bolt Beranek and Newman, Inc.
Cambridge, Massachusetts

Since BBN has had running time sharing systems in daily operation since mid 1962, I thought it worth while passing on some conclusions of the practical aspect of that experience. In particular I want to discuss our experiences with putting programming aids into the time sharing system: the Midas assembler, a symbolic Editor, and DDT. But first let me give you some background about our project.

In the fall of 1962 BBN had a three-user time-sharing system running on a PDP-1-b. BBN then received funding from the National Institute of Health and the American Hospital Association to develop a time-sharing system for medical applications, working together with Massachusetts General Hospital. A time-sharing system we called "Little Hospital", built to determine the feasibility of such a system, was running on a PDP-1-b by mid 1963. It was not possible to do any further research on that system because of the lack of bulk storage and core. Since that PDP-1-b was also needed for other work at BBN the Hospital Computer Group at BBN worked together with DEC to develop the PDP-1-d, making additions to facilitate time-sharing. The additions included allowing each 4k bank of memory to operate independently from the others, a set of character handling instructions, and the ability to trap all IOT and illegal instructions. The corner stone of the system is the DEC swapping drum which allows complete swapping of 4k of core in 33 milliseconds. The PDP-1-d has 24k of core, a Univac Fastrand drum with a capacity of 50 million characters and two Univac magnetic tape drives. The system currently allows 64 Teletypes, handled by a 630 scanner.

The initial aims of the project were to develop a time-sharing system for hospital applications which would handle the data in the hospital connected with the patient care and the patient record. We did not and do not intend to handle diagnosis or real time monitoring or directly connect the patient to the computer in any way.

Programs running under the time-sharing system perform such jobs as receiving all the data of an admission and keeping track of where all the patients are, their transfer and discharges. Laboratory tests can be entered and reports generated. The system keeps track of medications as they are ordered and types out a list of drugs to be administered at the proper care unit one half hour before they are due. Each Teletype has access to programs that operate on the data base and doctors and other authorized persons may run programs which allow them to ask questions about the data base and modify it. There are twelve terminals for experimental use in the care units in the hospital as well as twenty terminals in other areas such as the admissions office, laboratory and central record office. Members of the Hospital's Laboratory of Computer Science use these terminals regularly and work with BBN in testing programs and methods.

The original goals of the project were focused on a system to handle hospital data. There was no provision for program preparation on-line nor for background or batch processing. It was of course always possible to debug programs

under the Hospital time-sharing Executive but program preparation such as assembly and editing were done off line at regularly scheduled times using standard PDP-1 software on the non-time-shared machine. We put the Editor and Assembler into the system when we felt we could no longer justify running the machine without time sharing it. The factor in putting them on-line was more necessity than desirability.

Our Executive system consists of a Scheduler, I/O routines, and many system subroutines. The Executive occupies a 16k bank of memory. In addition there are two 4k user banks that operate independently.

There are two DDT's on our system. One is an Executive DDT which may examine and change the Executive system at any time as well as access any of the running user programs. Its language is very much like that of a standard DDT. It is operated only from a Teletype in the computer room by authorized persons and is used to examine the state of the system and correct failures.

The second DDT is used to debug any program that will run under the system and is called Invisible DDT. It is called Invisible because it does not take up any space in the core of the program being debugged.

When DDT is being used to debug a user program it performs many of the functions that the Executive might perform in other time-sharing systems. It loads programs and allows the user to save and get images of partially debugged programs, handles illegal instruction traps, provides additional core on request, and processes input from the Teletype character by character. DDT allows the user to have 7 independent images of core storage. He may address each 4k image at will. In reality only one core is available for Invisible DDT and these other seven programs, because a user program is allowed only 4k of storage. Each core that has been loaded into DDT is in actuality written on the drum. When the user examines one of his programs DDT reads the pertinent part of the program into its core and responds to the user just as standard DDT would. However, when the user wishes to run one of his programs DDT changes places with the program and allows it to run. Should the program encounter a breakpoint, execute an illegal instruction, or be interrupted from the keyboard by the break key the Executive program will return control to its DDT. Notice that because DDT is not actually resident in the same core as the user program the user program cannot destroy it.

Invisible DDT is in fact a 2nd Executive program running under the time-sharing Executive. The programmer uses DDT to control and run all programs that are not debugged and in addition, to run all other programs of the Programming system to facilitate communication among them. For instance a programmer might have a Midas program in one of his cores, an Editor in another and the program he is currently debugging in a third. The time to switch from one of these cores to another is well under one second.

The paper tape page editing program of the non-time shared machine, Expensive Typewriter has been replaced by a drum file editing program. Symbolic programs are typed into the Editor and are stored on the drum until expunged by specific command. The Editor was designed with the control language of DDT in mind. Lines may be examined by typing the symbolic address followed by a /; the contents may be modified by typing a new string on the same line. There are commands to insert, delete, and change lines. The Editor can handle files of very large size and all its data is stored on the drum.

The Midas Assembly program is a derivative of the non-time sharing Midas Assembly program, an assembler with extended macro capability that was originally designed to work in 16k of memory. Compressing Midas into the 4k allowed user programs required using drum storage for the symbol table and segmenting it onto the drum. It can assemble the same files that the non-time-shared Midas could with the exception that the source file must be on the drum. Midas is controlled in the same manner in a time-sharing environment except that the controls are typed instead of entered in the toggle switches. There are commands to initiate a pass, continue a pass, put a jump block on the binary, pseudo-punch the binary and symbol table on the drum etc. The binary and symbol table output files are stored permanently on the drum until they are expunged by explicit command.

It was inefficient to put extended file handling capabilities into all the programs of the programming system so a new programming aid called Handle was added to the programming system. It creates the system of indexes necessary for a set of programmers files, it allows the user to delete files, store them on magnetic tape, punch them on paper tape, and copy them from other programmers files.

We learned as predicted that a time-sharing system was not the ideal environment for a programming system. Some of the disadvantages of a time-shared system could have been alleviated on a regular non-time-shared machine with a large backing drum store if the expense of such a drum could have been justified for the non-time shared machine. Let me review each of the first three programs I mentioned again and talk about what happened to them under time-sharing.

DDT is obviously the ideal sort of program to be placed in a time-sharing environment. It was the impetus for time-sharing at BBN as developed by McCarthy and for small computer time sharing at M.I.T. under Jack Dennis. In fact as everyone had predicted DDT is the sort of program that only benefits from a time-sharing environment to the extent that a lot of users can debug simultaneously without being aware that they are time-sharing. When transferred to a time-sharing environment DDT retained all its good features: its response time seemed to be as good as that of a standard DDT except in the extreme case. In addition it allowed the user 7 cores of quickly accessible memory storage and almost inexhaustible drum storage whose access time was in the order of a couple of seconds, and it was always available immediately.

Editor also adapted well to the time-sharing environment. It gained the added advantages of always being available and having readily addressable drum storage. It, however, put more of a load on the system and suffered more from a busy system than DDT because of the structure of its data storage. To get a file ready for editing or to store it afterwards required a lengthy drum to drum transfer for files of any size. This transfer took a few minutes for a good-sized file but the main work of Editor - accepting corrections - always proceeded at top speed.

It seems that Midas was not a program that adapted well to being time shared. To process a symbolic of any size required very little Teletype I/O but a good deal of processor time and in the order of thousands of drum reads because of its drum symbol table and segments. To assemble a symbolic of 10 pages or more or one that contained macros seemed to take many times

what it would take with standard Midas even with only a few users on the system. In addition if four or more people were assembling at the same time the rest of the programs on the system slowed intolerably. We have found some ways to ease the situation. Our computer operators perform assemblies for us at low use times. We have been considering a program which will perform a series of assemblies that will run by itself. But these things do not help us time share the assembler; they help us avoid to a certain extent the fact that we have to time-share it. But the assembler does gain the same advantages as Editor from time-sharing, it handles permanent drum files not paper tape. Programs may be assembled at any time. In the case of smaller programs without macros the assembly completes quickly without much effect on the response time for other users.

The results can be applied to determining the success of time-sharing other programs. A time-sharing system has many different resources it can offer. When any program utilizes one of these resources too heavily trouble develops. Not only is the system slowed but the utilization of the other resources is neglected, and the time-sharing system loses efficiency quickly.

JOSS:* INTRODUCTION TO THE SYSTEM IMPLEMENTATION

G. E. Bryan
The RAND Corporation
Santa Monica, California

Abstract

JOSS is a time-shared computer system that provides for the solution of numerical problems via an easily learned language at remote typewriter consoles. The PDP-6 hardware used to implement JOSS consists of 32,000 words of 1.75μsec core memory, a 1-million-word 4μsec drum, a 6-million-word discfile, and various peripheral devices. A special data relocation mode for memory references has been added to facilitate interpretation of JOSS programs. The JOSS consoles, built around a Selectric I/O typewriter, were specially manufactured to RAND specifications. Features include full duplex signaling, line parity checking, a page eject mechanism, and several buttons and lights to control and report console status. The stand-alone JOSS software consists of the JOSS language interpreter and its arithmetic subroutines, a monitor for user scheduling and resource allocation, and I/O routines for the disc, drum, consoles, and other peripheral devices. JOSS service is currently available to nearly 500 users through 34 consoles, six of which are remote to RAND operating over both private and dataphone lines.

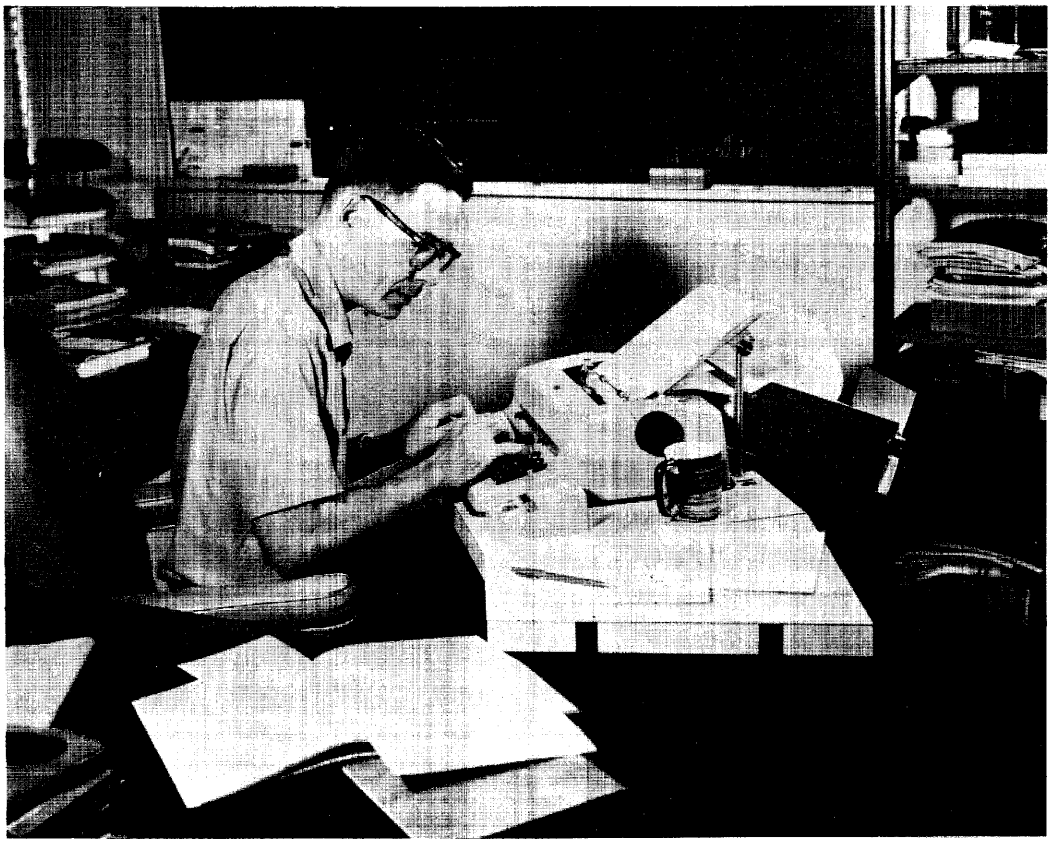
The JOSS System

JOSS is a computer system that allows a user** direct interaction with a powerful computer through a familiar device (typewriter) and in a familiar language (arithmetic or algebra). The machine that houses JOSS is dedicated exclusively to that task 24 hours a day, 7 days a week. No background tasks are performed. In contrast with project MAC, SDC, and other "general purpose" time-sharing systems, JOSS has been designed for the casual user and applications programmer rather than for the systems programmer.

In order to make such a service available to many people at an economic price, the system is time shared; that is, simultaneous and noninterfering service is supplied to a number of users at their individual typewriter consoles. The primary advantage of JOSS is its ability to provide fast solutions to reasonably complex problems with a minimum of administrative delay. The user must specify all data relevant to describing his problem and the algorithm for its solution, but need only provide a minimum of detail regarding how his problem is to be solved on the available hardware. The JOSS user has at his command a machine of about the power of a

*JOSS is the trademark and service mark of The RAND Corporation for its computer program and services using that program.

**Usually scientists and engineers, but also secretaries and kids.



4K 704, with the additional bonus of a language interpreter.

JOSS is a problem-solving tool that the user can apply to small- and medium-size problems with a minimum investment on his part in learning its use.

History

Work was first started on JOSS in 1960. The system was implemented on the JOHNNIAC computer (now retired) by J. C. Shaw, to whom goes the bulk of the credit for both design and construction. The system was partially operational in early 1963 and fully operational with eight consoles in January 1964--no small accomplishment considering the 4000 word memory and doddering years of JOHNNIAC.

However, JOSS was impressive enough on the few days JOHNNIAC felt well to substantiate the acquisition of a new computer and the creation of a well-supported project to build a second JOSS.* As one user quipped, "It's better than beer--we're hooked."

Introduction of the new JOSS in formal operation took place in mid-February 1966, although selected users had been contributing to system debugging since its first coherent words in early November 1965. Implementation on a large modern computer gave the new JOSS about an order-of-magnitude more capability than its predecessor--30 times as fast, 5 times the storage per user, 4 times as many

*The name JOSS still stands for JOHNNIAC Open Shop System in spite of the fact that JOHNNIAC now resides in the Los Angeles County Museum. It has been suggested that JOSS should now be interpreted as JOSS Open Shop System.

consoles, 50 percent faster consoles, room for several powerful new language features, and, in addition, spare capacity. We believe that well over 100 consoles can be handled within the present configuration without service degradation.

Scope and Intent

JOSS is commonly characterized as a tool for the solution of small numerical problems--and so it is. But the word "small" would be better rendered as "not large." To say that JOSS is a good desk calculator is a substantial understatement, although it is often used effectively for that purpose. A list of the limitations of JOSS is perhaps more instructive than one of its capabilities. As a data retrieval system, it is poor; no provision exists for handling large files of information; it can't tackle very large problems (by today's standards); the 40-page FORTRAN code is unfeasible; and the compact but long-running program, say, 2 hours on a 7094, although possible, would be extremely tedious--perhaps as much as 60 hours.

Together with other so-called time-sharing systems, JOSS enjoys the substantial advantages of the interactive environment. The user is able to approach his console with perhaps only a partially formed idea of his problem and to come away in a few minutes or hours with the answer. This method is estimated to be about ten times faster than the usual problem-inception-to-problem-solution approach to a computer. It is successful enough that many problems that weren't worth the effort before are now being solved.

JOSS differs from the general-purpose interactive time-shared systems in that its operation is simple and its goals are limited. What little information the casual user does have to remember about the system's operation can usually be brought back to mind by experimenting at the console without recourse to a manual of operation or to the help of a system "expert." For this ease of use, JOSS gives up many general-purpose features, but retains a large complement of casual users. As Willis Ware has said, "For a certain class of problems, at least, the programmer as the middleman between the problem and the machine is no longer needed."

Hardware

The essential hardware components of the PDP-6 computer system used for JOSS are outlined in Fig. 1.

The arithmetic processor, a word-organized multiaccumulator-index register machine, is provided with 32,000 words of high-speed core memory in two independently accessible 16,000-word boxes. JOSS uses one of the boxes, the low-addressed one, for the JOSS software, and the other to hold the user programs and data.

The processor contains a relocation register whose contents are added to memory references if certain conditions are met. The RAND PDP-6 has been modified so that the appearance of bit 20 in the address satisfies the requirement. The contents of the relocation register are set to the base address of the locations in memory that contain the user's program. All user references, as signaled by bit 20, are modified by the hardware to refer to the correct current user location. The contents of the relocation register represent, therefore, the context that determines the user of the moment. Since bit 20 corresponds to a real address of 32,768, both the size of the JOSS system code and the size of individual users are limited to a maximum of 32,767 locations. In effect, the RAND relocation mode provides a second level of indexing.

Because the capacity of core memory is not sufficient to hold data for all possible users simultaneously, the magnetic drum is used to store data for some of the users during those times when interpretation of their data is not required. Drum transfers are controlled by the input/output processor through independent ports to the memories. Thus, transfers of user

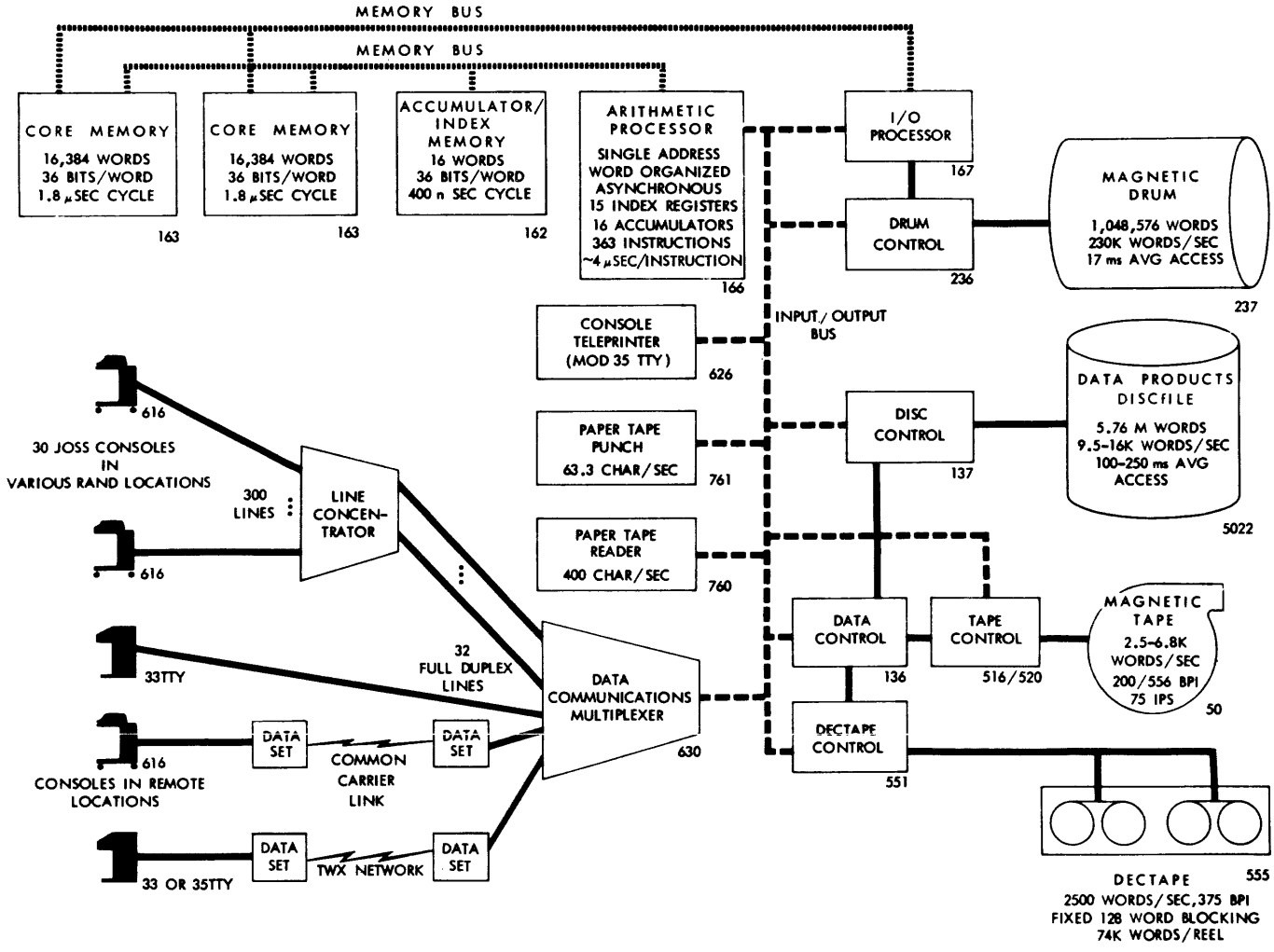


Fig. 1—JOSS PDP-6 system

data between the drum and core are accomplished independently of the arithmetic processor. The attention of the arithmetic processor is needed only to initialize the I/O processor for the transfer and to take action after its completion. Memory cycles are taken by the I/O processor as needed to service the drum. These cycles interleave with those taken by the arithmetic processor in the interpretation of JOSS users' programs.

Logging of information descriptive of the gross system operation is done on the console teleprinter. The information is printed each minute and includes the number of present users, the number of users computing, the amount of computing accomplished, the total lines transmitted to and received from users, and various errors.

The data communications multiplexer scans lines connected to the JOSS consoles and reports via a machine interrupt to the arithmetic processor when a character has been received or the transmission of an output character has been completed. All communication with the consoles passes through the multiplexer and down the I/O bus to the arithmetic processor.

In addition to the local JOSS console lines, the multiplexer has timing and other special gear necessary to interface with dataphones connected to remote JOSS terminals as well as with local TTY's and TTY's on the TWX network.

The data control handles the transfer of information to both the discfile and the magnetic tape. Because it cannot transfer data to more than one device at a time, usage must be shared.

The discfile provides long-term storage for users' programs and data. Its capacity of 200 million bits is sufficient for many thousands of user programs. Access time to individual records on the disc is generally about 200 ms, but long programs, queued use of the disc by several users, and computing commitments may extend an individual file or recall action to several minutes. Normally, however, a disc action takes about a second.

The IBM-compatible tape unit is used to collect accounting records and statistical information, which are processed in the general RAND accounting system on another computer. The discfile is periodially dumped onto tape for backup purposes, using an off-line program not contained in the regular JOSS software.

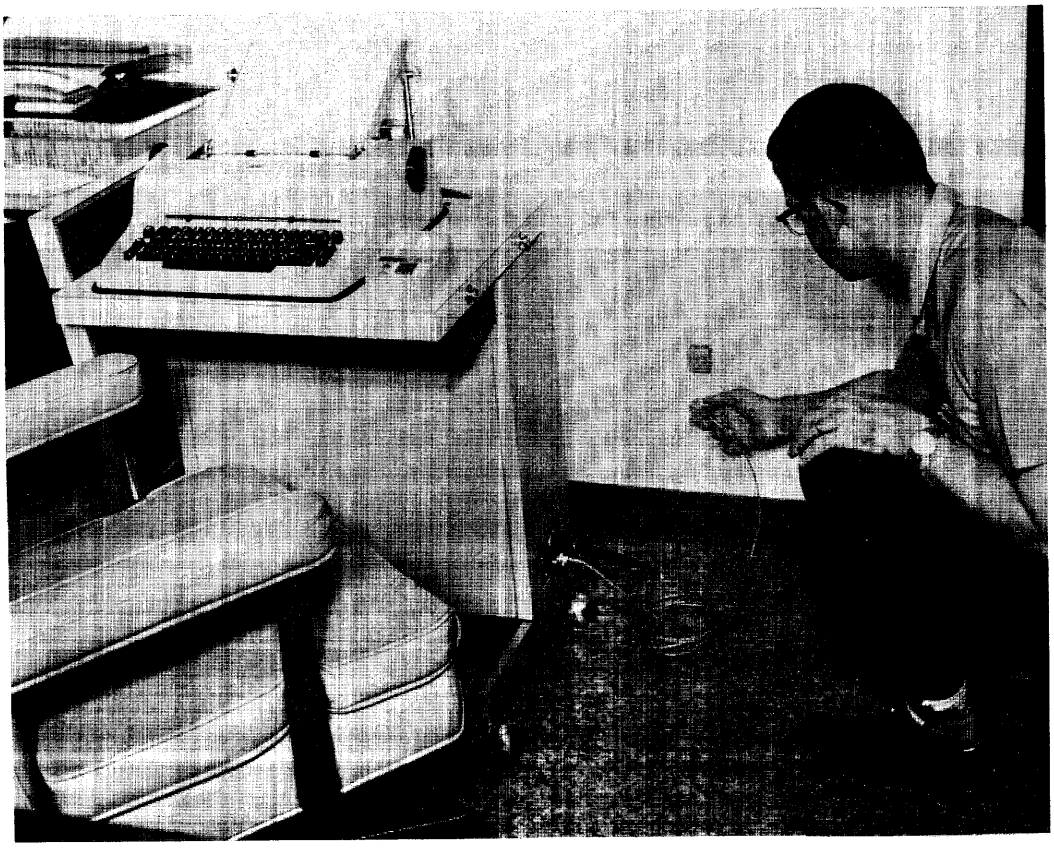
The JOSS system includes four Dectape drives that are used in system support. Operating binaries for the DEC-supplied time-sharing system and its subprocessors (assembler, editor, file manipulator) used in JOSS development are contained on Dectapes, as well as symbolic, relocatable, and absolute copies of the JOSS software. The IBM-compatible tape is used for assembly listings during DEC time-sharing system operation.

Paper tape reader and paper tape punch are used for loader and maintenance program input and for paper tape copying.

The line concentrator is built around Strowger line-finder stepping switches in much the same way as an ordinary PBX or CDO. It serves to concentrate 300 lines from user offices to 40 multiplexer inputs. The presence of a JOSS console at the plug in the user's office is detected by the concentrator and it establishes a connection to the computer. It is through this device that "plug in" computer power is provided in the individual user's office.

In addition to the JOSS console described below, up to eight input lines may be devoted to teletype operation. Currently, two model 33 TTY's are in use at RAND and two lines are available on the TWX's network.

The JOSS console was specially built to RAND specifications by the Digital Equipment Corporation. The IBM Selectric I/O writer includes a pin-feed platen, a two-color computer controllable ribbon shift, a special character set adapted to algebraic languages, and a specially built form-feed mechanism for moving the paper to page top.



The cabinetry that holds the typewriter and, in its base, all the associated electronics is mounted on casters for mobility and includes a four-way detachable side table that provides a convenient work surface on the left or right side of the console at the user's option.

A console control box at the right side of the typewriter contains two control keys and three white status lights. The power ON/OFF key in addition to applying or removing power at the console sends a unique signal to the computer reporting logical ON and OFF for the console. An interrupt button provides a signal to request return of control of the console to the user during those times when the computer has control.

The status lights report that (1) the JOSS system is on, (2) the console has power on, (3) the typewriter is ready to print, (4) the computer controls the typewriter (red light), and (5) the user controls the typewriter (green light). During times when the computer is in control (red light on), the keyboard is locked and the ribbon color is shifted to black. When control of the typewriter is returned to the user, a beep tone sounds, the green light turns on, the keyboard unlocks, and the ribbon color shifts to green.

The console electronics contained in the base of the cabinetry control the console operation and send and receive signals to the computer over a full duplex line in an 8-bit, 11-unit start-stop code. The 67-ms signaling time per character is designed to keep the typewriter operating at its full capacity of 15 characters per second. Six bits are sent for each character (including up and down shift characters), one bit is used to indicate control information (request console status or console status report), and one bit is used for a parity check.

The JOSS Language

The language provided for JOSS users is simple and easy to learn with relatively few rules governing correct use. In many ways it is similar to other algebraic languages in wide use on every variety of computing machine. The language has been specially adapted to convenient, direct use by an active user at a typewriter console.

Most familiar statement types exist: Replacement (assignment), transfer of control, input, output, and formatting are executed by the verbs: Set, To, Demand, Type, and Form. The conditional if clause may be appended to any JOSS statement. The JOSS Do statement acts more like a subroutine call than the similarly named FORTRAN statement.

Significantly, some statement types do not appear. Declarations such as DIMENSION are unnecessary, because of the linked-list memory assignment in the user's block, and modes (e.g., REAL, INTEGER) are handled implicitly.

Whether a statement is to be interpreted immediately or stored for future execution is indicated implicitly by prefixing statements to be stored with a "step label," which gives the proper location of the new statement relative to others already stored. Thus, a step labeled 1.25 will be inserted after step 1.2 and before 1.28; it will replace any previous step labeled 1.25. Statements without step labels are interpreted immediately.

A collection of steps with step numbers having the same integer part is called a "part." Thus, all steps labeled one-point-something constitute part 1. A Do statement causes interpretation of a part as if it were a subroutine. Example 1 illustrates the use of six common JOSS verbs, the conditional if clause, direct and indirect program statements, and the ordering of statements by step number.

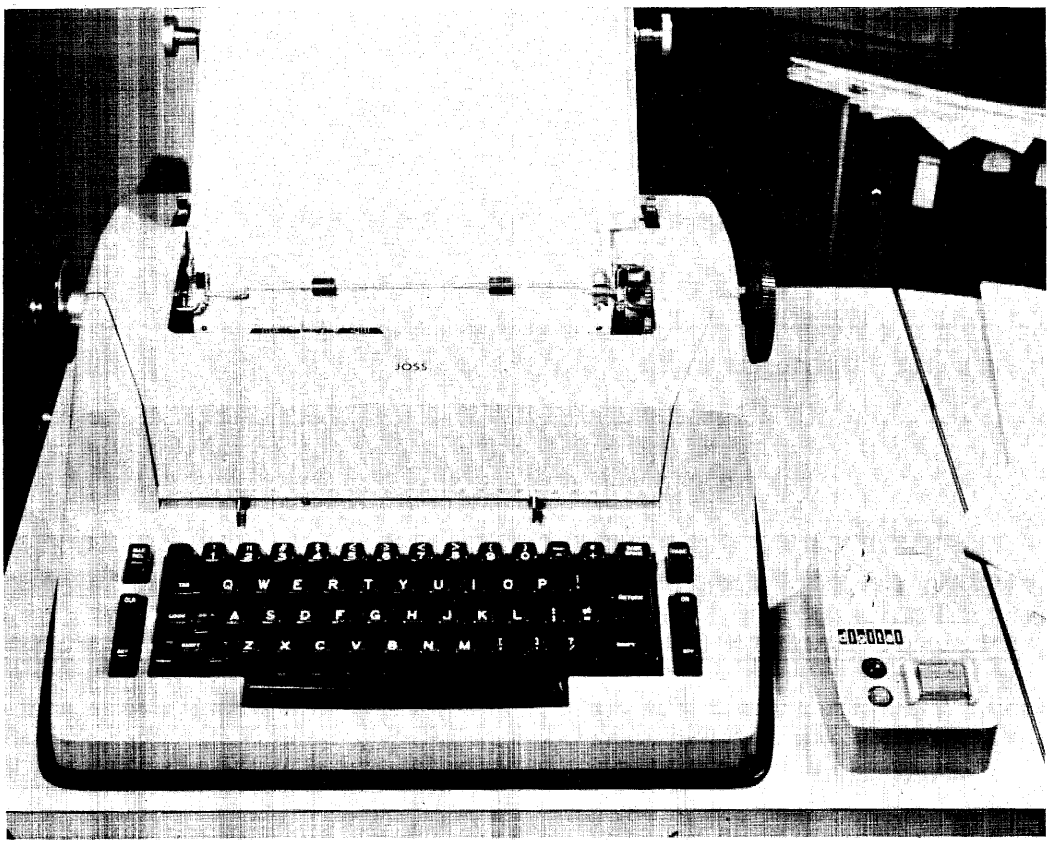
Certain of the JOSS language facilities deserve special mention because they are less frequently found in the common algebraic languages. The verb Let defines a formula or rule for computation. It may have up to ten parameters. The functions sum and prod allow direct expression of the mathematical operations for summation and product over a specified range of values. Use of these and similar functions (max and min) eliminates many program loops and aids in the compact expression of the desired computation.

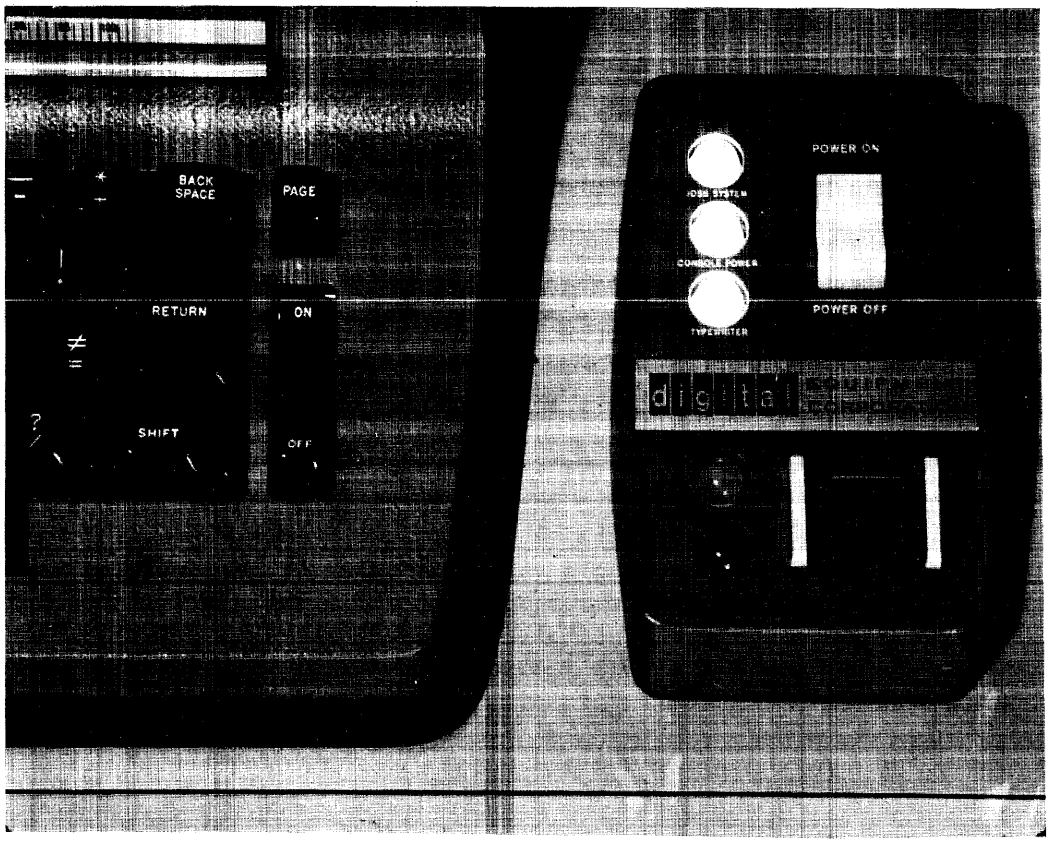
Example 1--Sample JOSS Program

```
1.1 Demand x(i).
1.2 Set s=s+x(i).
1.3 Type i,x(i),s in form 1.
1.35 Set i=i+1.
1.4 To step 1.1 if i<4.

Form 1:
i = _  x(i) = __.__  sum = __.__

Set s=0.
Set i=1.
Do part 1.
       x(1) = 97.45
i = 1  x(i) =  97.45  sum =  97.45
       x(2) = -67.98
i = 2  x(i) = -67.98  sum =  29.47
       x(3) = 33.47
i = 3  x(i) =  33.47  sum =  62.94
```


JOSS


BACK SPACE
PAGE
RETURN
ON
SHIFT
OFF
POWER ON
POWER OFF
TYPEWRITER

Conditional expressions, which may be used wherever expressions are valid and which use colons and semicolons to denote the if...then... if...then...if...then...otherwise... notation, again contribute to compact notation of complex choices and discontinuous functions. For example, see P(x) in Example 3 below.

The first function allows the user to find the first value of an index that satisfied a given proposition.

The compact expression achieved by these features is shown in Examples 2-4, which give formulas for polynomial root finding, prime number determination, and two-point Gaussian integration.

As shown in Example 2, roots of the polynomial p(x) are found by Newton's method expressed in formula i(x), where q(x) is the approximate derivative of p(x). The formula r(x) recursively improves the root until it is sufficiently close to zero. The program at step 1 prints the three roots as found by setting approximate starting values through the for clause of the controlling Do statement.

The formula P(x) given in Example 3 has value true or false depending on whether x is prime or not. The first part, P(x), filters out certain special cases (e.g., negative numbers). The second formula tests for primeness by finding the first exact divisor (fractional part = 0) of x and reporting

Example 2--Root Finding

```
1 Type r(x),p(r(x)) in form 1.

Form 1:
x = __._____     p(x) =  _._____

        i(x):  x-p(x)/q(x)
        p(x):  x*3-10•x*2-6•x+10
        q(x):  [p(x+d)-p(x)]/d
        r(x):  [ |p(x)|<10*(-6): x; r(i(x))]

           d =        1•10*(-4)
Do step 1 for x=-5,1,10.
x = -1.24670     p(x) =   .00000
x =   .76528     p(x) =   .00000
x = 10.48142     p(x) =   .00000
```

Example 3--Prime Number Determination

```
1 Type x if P(x).

        P(x):  [x≤0:false; x≤3:true; p(x)]
        p(x):  first[ i=2,1(2)ip[sqrt(x)],x: fp(x/i)=0 and i≠1] = x
Do step 1 for x=0(1)12,100001(2)100101.
           x =        1
           x =        2
           x =        3
           x =        5
           x =        7
           x =       11
           x =   100003
           x =   100019
           x =   100043
           x =   100049
           x =   100057
           x =   100069
```

"true" if that divisor is x itself. Only divisors up to the integer part of the square root of x are tested. The typing program at step 1 and the controlling Do statement test the formulas and give some output.

The function I in Example 4 integrates the function f over the range a,b in n intervals.

Example 4--Gaussian Integration

```
        I(f):  h/2•sum(i=1(1)n: sum[j=1(1)m: f(x(i,j))])
          h:   (b-a)/n
      x(i,j):  a+h/2•[t(j)+2•i-1]

          m =          2
          n =         30
a=0
b=1
t(1)=-1/sqrt(3).
t(2)=-t(1)
Type I(exp),exp(1)-1.
      I(exp) =          1.71828184
    exp(1)-1 =          1.71828183
Let R(x)=(1+x*2)/(1+x*4).

         I(R) =          1.11072073
sqrt(2)•arg(1,1) =       1.11072073
```

JOSS arithmetic is carried out by an interpretive package of routines that operates on numbers carried in scientific notation--an integer magnitude and a decimal exponent. Primary advantages of this notation are exact I/O number conversion and the restriction of repeating fractions to those familiar in the decimal system.

Users may save programs, data, forms, and formulas on the discfile and retrieve them from the file using the verbs File, Discard, and Recall. Items stored on the disc are in symbolic form. The file operation behaves as if the user were typing on the disc and the recall operation acts as if the disc were typing on the user's program space. This means that the user's current core contents are only changed as implied by the contents of the disc. Statements replace current statements of the same number, and new values are assigned if variables defined on the disc were previously defined in core. The user may reference the files with his program to accomplish a limited form of chaining.

There are a number of features normally included in computing systems that JOSS does not have: (1) There is no way for a user to handle high-volume I/O, which precludes the use of large tape files. (2) The interpretive mode of JOSS operation (even down to the arithmetic) limits the speed of operation. Thus, very long, detailed calculations are impractical. (3) The maximum amount of core available to individual use is limited to 4K, making very large programs or programs with large data bases infeasible. Finally, (4) JOSS operates only on numbers, which rules out generalized symbol manipulation programs. All of these limitations were imposed because their inclusion was considered incompatible with a high-speed, highly interactive computing service for a large number of casual users.

JOSS Software

The JOSS operating software is divided into five principal parts: the interpreter, its arithmetic and

function subroutines, the monitor (which also contains the drum, tape, and TTY console I/O routines), the distributor (JOSS console and TTY I/O), and the disc routines. All of this code is permanently resident in lower memory. Its total size is about 16,000 memory locations, and includes all I/O buffers for the consoles and several thousand words devoted to the gathering of performance statistics.

The interpreter, along with the arithmetic subroutines, is the part of JOSS that examines users' commands and computes answers in response to them. Users' programs are analyzed character by character by the interpreter, as the name implies, to produce the indicated results. No compilation of user programs is done. User commands and data are carried in linked lists within variable-size user blocks. The entire user block must be in core during interpretation, and the RAND relocation hardware requires that this block must always occupy a contiguous area of memory. User blocks have a minimum size of 1000 words and increase in increments of 1000 to a system-imposed limit of 4000 words.

The distributor and disc routines are trap time I/O packages that control transmission of data to and from the user consoles and the discfile. Both communicate with the monitor about the I/O activity that they control through signals, which are software analogs of machine interrupts. These signals are referred to as logical traps or logical interrupts.

The monitor acts as a scheduling, resource-allocating, and synchronizing device, deciding when, and ensuring that, all data and hardware necessary for a particular action are simultaneously available. To carry out this process, the monitor maintains a series of queues of users in various activity states and of data for hardware devices. Signals from the other software components and a time-interrupt signal control both the changing of users from state to state and through these states the monitor's scheduling of the tasks of the system.

Also included in the monitor code are trap time routines to handle tape, drum, TTY console I/O, real-time clock interrupts; routines to gather and display performance statistics about overall JOSS operation; routines to provide accounting information to be input to the RAND computer time-charging mechanism; and routines that act as processors on the level of the interpreter to supply the log-on procedures of receiving initials, project number, department name, and the final log-off process.

Figure 2 shows the interrelations of the JOSS software, including the data buffers that interface certain of the components. Control and data paths are shown, although it is sometimes difficult to classify a particular path as control or data.

The disc routines and the distributor, as well as the portions of the monitor that deal with the drum, tape, and TTY console I/O, are trap time routines; that is, they operate only in response to hardware interrupts from the I/O devices that they control. Brief exceptions to this rule occur when the routines are entered to initialize an I/O action. These trap time routines generally signal completion of activity through logical traps that are interpreted by the monitor in its main processing loop.

Top-level control of the machine is shared between the monitor and the interpreter, with the bulk of time spent in the interpreter when users are requesting computation. The monitor regains control at least every 200 ms through the trap time setting of the signal COMEBACK, the logical interrupt signal for the interpreter. During its control periods, the monitor adjusts user states according to the current logical signals, initializes I/O actions as may be appropriate, and determines, from the user states, which user should receive the attention of the interpreter next.

The logical control signals that pass between the various software components of JOSS are summarized in

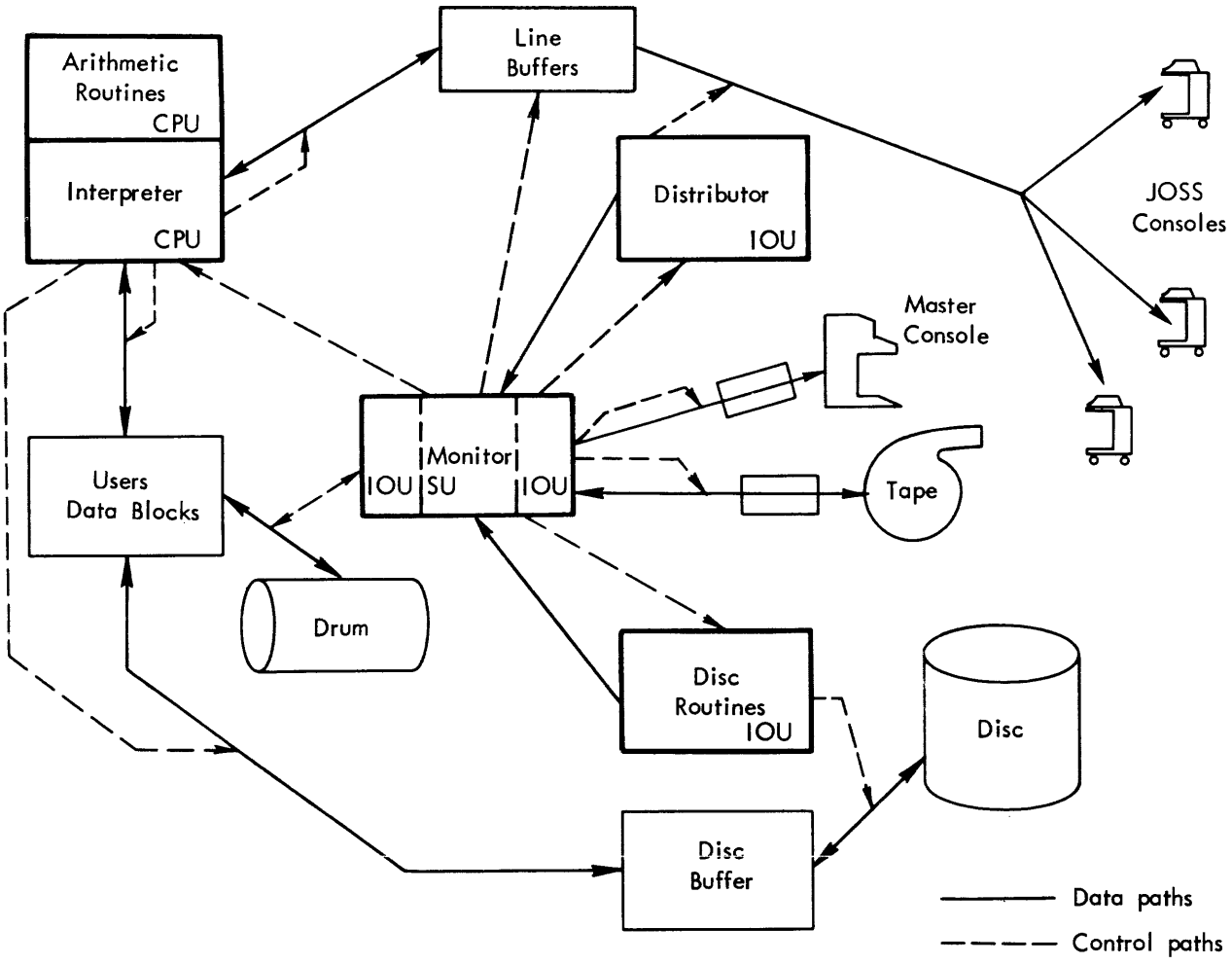


Figure 2 Interrelations of JOSS software

Fig. 3. Most of the signals are indicated by the setting of certain communication cells. Some signals, however, particularly the ones that start I/O action, are routine calls.

User State Transitions

During JOSS operation, users are sequenced through the monitor states in response to signals from the consoles, signals from the interpreter (in response to the executing program), and a time signal generated by the hardware.

A diagram of the transitions between states is presented in Fig. 4. Although the figure gives an excellent impression of the overall operation of the monitor, it is by no means complete. Reference to the actual code must be made if the exact details of state transitions are required.

Several subcycles are identifiable in the figure. At the top left of the diagram are the states that relate to console turn on and turn off; at the top center, the states applying to program input and short computations; at the lower left, the typewriter output limited sequence; at the lower center, the compute limited sequence; at the lower right, the disc activity sequence; and at the upper right, the drum transfer sequence, used in certain cases when more core blocks are needed by a user.

The states are broadly divided into two priority groups: a high-priority group, shown with heavy shading, and a low-priority group. Users whose states are in the high group require interpreter service, while those in the low group do not. Signals from the consoles or from the user's program cause changes from one state to another.

The ON and OFF sequence provides mechanisms for calling the separate software processors that provide the user's initial core block and that handle the initial salutation, including receipt and checking of the user's initials, project number, and department, and the writing of the accounting record at turn-off time. At turn on, the user is placed in QM state if the system limit on the number of users has been reached. Under most operating conditions, this limit is set higher than the number of consoles so that the limit is never reached. Before the installation of the drum, the limit was lowered to a value that would keep all active users in high-speed core. When the user turns his console off, his state is changed to TOF, except when a disc action is in progress. In the latter case, the OFF signal is flagged in the user's status word, and his state remains unchanged through completion of the disc action. At that time, the change to TOF state, production of final accounting records, and the reenabling of the console occurs.

During periods when a user is typing program steps and doing short computations via direct commands, he cycles through the states in the Input and Short Computation Sequence. During typing the console is green and the user's state is GR. As soon as the carrier return is pressed, his state changes to RC and subsequently to CU for interpretation of the line just typed. If the line is accepted without comment, the state returns directly to GR while error messages or response lines return to GR via DSU during the time that the typewriter is printing the output line. Attachment of a buffer to each green user is required for receiving his input. In the event that none are free, the user's state is changed to ABG until a buffer becomes available.

The Output Limited Sequence controls those users whose programs produce output faster than can be printed at the console. One of the system parameters is called the choke number. When the number of lines of output ready for printing at the console equals this number, the user's state is changed to CK and further computation ceases. When the number of output lines is reduced to the value of the unchoke number, the user's state is changed to UC and computation is resumed. If, during this sequence, the user presses the interrupt button, his state is changed to RIB and computation is resumed for response to the interrupt request.

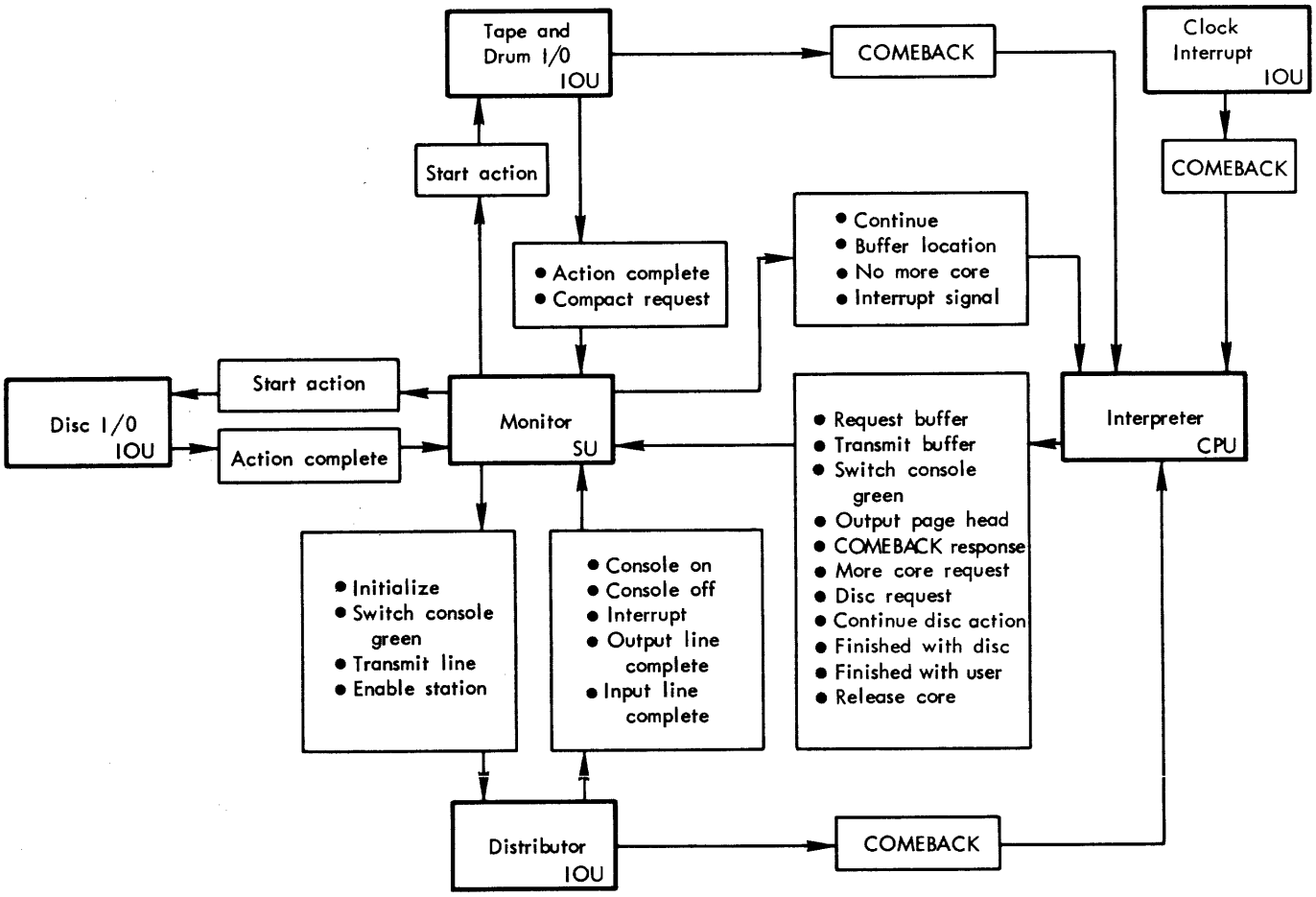


Figure 3 Summary of logical control signals

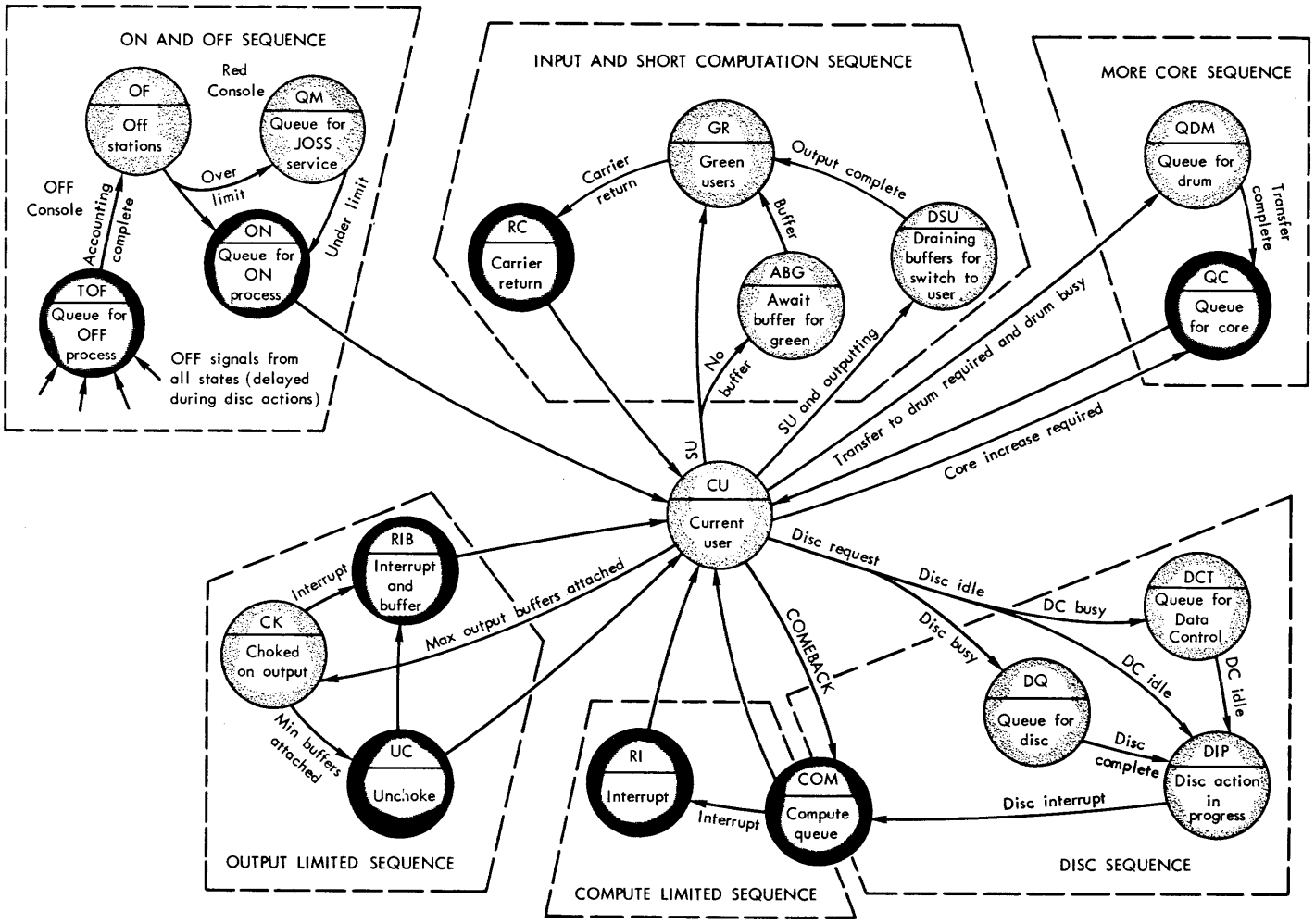


Figure 4 JOSS user state transitions

In the Compute Limited Sequence, those users in the COM queue share the computer in 200-ms time slices or "quanta," as controlled by the setting of the signal COMEBACK through the hardware clock interrupt. During these sequences, the user's state is changed to RI if he presses the interrupt button.

The purpose of the Disc Sequence is to queue users for access to the disc, allowing only one user on at a time, and to synchronize the use of the data control device, which is also used for transfer of information to tape. The major cycle is CU-DIP-COM with actual disc activity occurring only during DIP. During COM the disking user waits his turn for the attention of the interpreter, and in CU the disc buffer is either filled or emptied depending on the command in process. Other users requesting disc service during this time wait in DQ, while the disc routines wait in DCT if the tape unit is using the data control.

The More Core Sequence is employed when a user requests an additional block of core storage (implicitly through the interpreter) and no idle blocks are available in core. In this case, the user is transferred to the drum and his state is changed to QC, a high-priority state that forces the swap algorithm to make core space available and restore the user to the main core. If the drum is busy when the request is made, the user's state is changed to QDM where he waits for transfer to the drum.

Main Processing Loop

The JOSS monitor main processing loop provides for action on a number of possible asynchronous events. These events or signals are either such that no machine interrupt is available to flag the event or are such that processing at the time of occurrence would be impossible, improper, or inconvenient. The deferral of these actions to the main processing loop ensures that certain routines need not be trap-protected or coded as pure procedures.

Figure 5 outlines the flow of control for the major functions acted on in the main processing loop. There are two main paths, JOSS active (processing a user's request) and JOSS idle. The idle path has a minor branch not shown on the figure that distinguishes idle passes when the drum is busy. This indicates that a user currently on the drum has requested service and no other activity could be performed in overlap with the drum transfer. Usually this happens when there are many users but a very light computing load.

The interpreter releases control to the monitor whenever any of the monitor-provided facilities are desired and also when the signal COMEBACK is set. COMEBACK is set by the occurrence of monitor-distributor signals and every 200 ms by a real-time clock interrupt. Thus, when the interpreter is processing long computations, the monitor regains control in order to service signals from other consoles and to time-share the use of the machine among compute-bound programs.

Console signals are produced for the monitor by the distributor. These signals are translated into state changes, which will later control the selection of system tasks.

The disc interrupt signal is set by the on-line disc routines when a disc buffer is either filled during an input action or emptied during the filing of user information on the disc. The state of the user with disc action in progress is changed to compute (COM) so that the interpreter can proceed with the filling or draining of the next buffer load of information. This ensures that the user's block will be in-core during the necessary points in the disc transfer without requiring it during the entire transfer.

A final mode of processing is provided by a monitor flag that indicates the monthly production of disc accounting records. This disc-to-tape operation is synchronized through logical traps from the disc and tape routines.

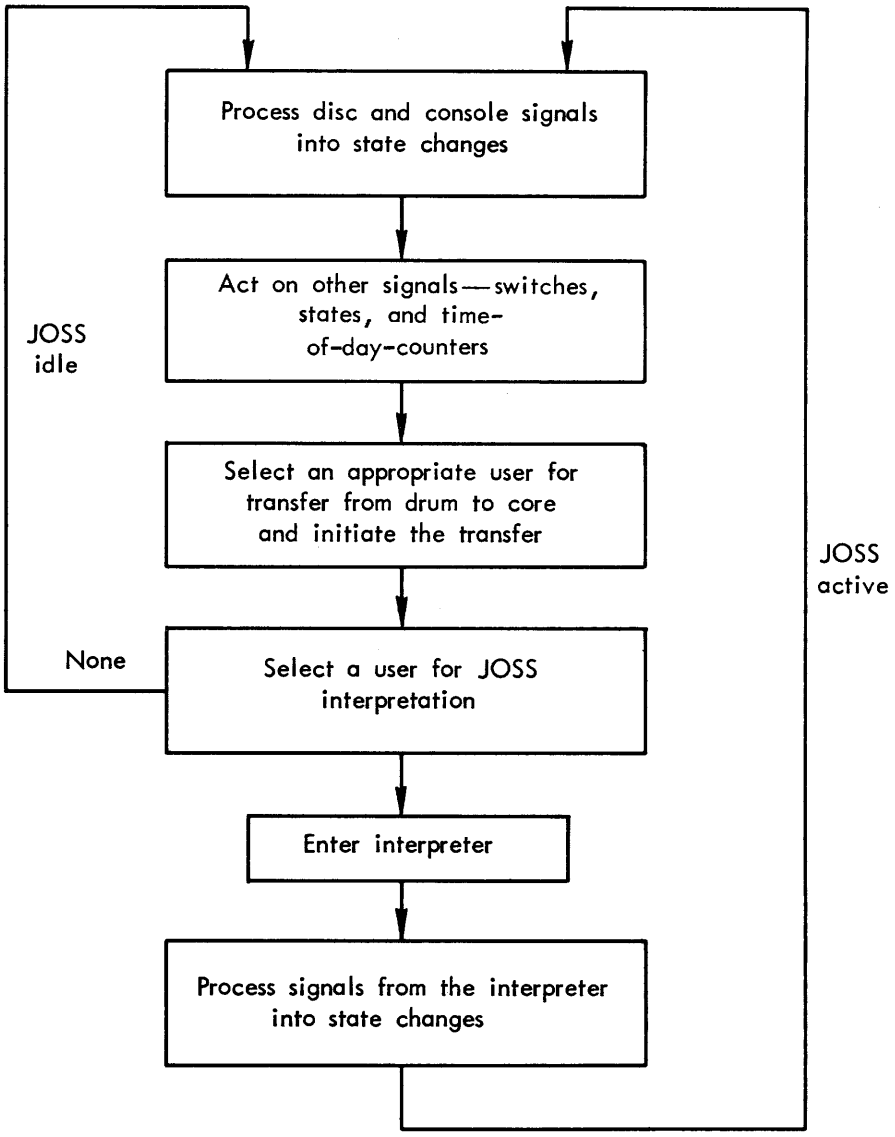


Figure 5 JOSS monitor main processing loop

The miscellaneous functions performed in the main processing loop can be divided into four broad categories: switch examination, dormant queue service, core compaction, and periodic functions.

Two special switches are monitored: one to enable the system shutdown procedure and the other to forcefully shut down the system.

Certain user states are entered to await the occurrence of particular events. The main processing loop examines the queues corresponding to these states and, if a user is waiting, checks further to determine if the appropriate condition has been obtained. If it has, action is taken. Queues serviced in this way and the resulting actions are as follows:

DCT Disc user waiting for use of the data control. When the data control becomes idle, the user state is changed to DIP and the disc routines are entered.

ABG Users awaiting a buffer for switch to green.

QDM Users waiting for the drum to idle in order to be transferred there.

QM Users in the JOSS service queue. Users in this queue are given service whenever the number of active users falls below the allowable number.

During some kinds of drum swaps, more than one small-size user may be transferred to the drum in order to make room for one large user. The users to be transferred are selected on the basis of their priority. Thus, there is no guarantee that the core space freed is contiguous, which it must be to read in the large user. Compaction of core is performed at the main processing loop level to bring all available core blocks adjacent to one another at a time when they are not in use by the interpreter. The waiting transfer from the drum to core is then initiated.

Major time interval incrementing--the counting of minutes, hours, days, months, and years--is accomplished from the main processing loop. Functions performed in this part of the MPL include initiating log reports each minute, accumulating statistics on the minute and hour, and initiating accounting for the disc each month.

Next in line in the MPL is the selection of an appropriate user to be transferred from the drum to core. This action is only performed if a transfer is not already in progress. Of course, only the high-priority users are candidates for transfer into core.

Finally, an in-core user is selected and the interpreter entered. The main processing loop is completed when the interpreter returns to the monitor.

JOSS Usage

The statistics given below are intended to describe briefly the extent to which JOSS is typically used, as of the fall of 1966. It should be noted that usage patterns are still changing fairly rapidly. Improvements that have had a substantial impact on usage include that from JOHNNIAC to the PDP-6, which increased computational speed by a factor of thirty, and the later additions of the drum and discfile, which gave users five times more core space and the ability to file programs and data. Intermediate system changes of smaller magnitude, as well as the natural growth of user interest, will, no doubt, keep the usage patterns changing on the positive side, as is usual throughout the computing industry.

With the exception of six hours per week of scheduled maintenance and occasional (or possibly frequent) unscheduled down time, JOSS operates 24 hours per day, 7 days per week.

Usage

- About 500 users, some casual, some addicted.

- About 300 different users each month.
- Computation is divided into 150 sessions each day or about 3000 each month.
- The users are served by 34 consoles, of which 2 are on the TWX's network and 5 are in remote locations (3 on the East Coast). The remainder are located in RAND.
- Usage is highest during mid-morning and mid-afternoon, normally peaking at 25 concurrent users. The average number of simultaneous users for a 24-hour period is 6; average in prime shift is about 13.

Computing

- Actual computing--as contrasted to idling when waiting for work--is 130 hours per month, which is 18 percent of total time or 74 percent of prime shift.
- Approximately 3,000,000 JOSS statements are processed every 24 hours in an average time of 5.5 ms per statement.

Sessions

- A typical console session lasts 45 minutes; 10 percent are less than 2 minutes; and 10 percent greater than 2 hours.
- Computing time during a session averages 2 minutes or about 5 percent of session time, but 60 percent use less than 10 seconds and 1 percent compute more than 1 hour.
- On the average during each session two items are retrieved from the files, one is discarded, and one placed in the file.
- Users input 2 lines per minute on the average and output 6, which totals 90 input lines and 270 output lines during the typical session.
- Mean interaction (that is, time for an individual user between carrier returns) is 30 seconds and is distributed approximately exponentially with 40 percent less than 6 seconds and 1 percent greater than 5 minutes.
- Average character rates per user are 1 per second input and 3 per second output.
- User programs average 320 JOSS cells or 1000 machine words. Twenty percent use less than 10 JOSS cells and 7 percent more than 1000.
- The session time distribution of size is as follows:

<1K words	58%
1K-2K	30%
2K-3K	5%
3K-4K	7%

The early usage of JOSS as implemented on the PDP-6 confirmed it to be an effective computational tool. The average computation per session of 20,000 statements indicates that complex problems are being solved, but the hardware and implementation are such that considerable computational capacity remains. This additional capacity appears sufficient to support between 100 and 200 consoles at current usage rates.

PDP-10 SYSTEM PHILOSOPHY
NEW PRODUCT ANNOUNCEMENT

David Plumer
Digital Equipment Corporation
Maynard, Massachusetts

Abstract

An analysis of the system design philosophy behind Digital's newly announced PDP-10 Computer series. A discussion of the salient features of hardware and software which combine to make the PDP-10 a true family of totally modular machines. Topics to be considered include: 1) device and memory independent software; 2) batch processing within conversational time sharing; and 3) true upward compatibility within four PDP-10 software systems.

Introduction

The PDP-10, a new, expandable computer system available in five configurations and offering optimum power and versatility in the medium price range, was introduced by Digital Equipment Corporation at the opening session of the 1966 Fall Joint Computer Conference in San Francisco, and at the DECUS Fall Symposium, Lawrence Radiation Laboratory, Berkeley, California.

The PDP-10 is designed for on-line and real-time scientific, engineering and process control applications with prices starting at $110,000. The PDP-10 features a 1 microsecond cycle time, a 2.1 microsecond add time, I/O bus transfer rates up to 7,200,000 bits per second and a modular, proven software package that expands to take full advantage of all hardware configurations. Memory can be expanded in 8,192 word increments to the maximum directly addressable 262,144 words.

The Five Configurations

The basic configuration, PDP-10/10, includes an extremely powerful programmed processor with 15 index registers, 16 accumulators and 8,192 words of 36-bit core memory, a 300-character-per-second paper tape reader, a 50-character-per-second paper tape punch, a console teleprinter, and a two-level priority interrupt subsystem. Source and object programs reside on paper tape.

PDP-10/20 adds two DECtapes, Digital's unique fixed-address magnetic tape system which allows compact file structured storage. A single tape may contain multiple files of various types; any single file may be deleted, changed, or expanded without effecting other files on the tape. Source and object programs reside on DECtape providing maximum flexibility and programmer convenience.

Another configuration, the PDP-10/30, offers the user still more versatility. It includes, along with the standard processor features, 16,384 words of memory, and additional I/O devices. This system is designed to facilitate stand-alone computing capabilities incorporating specially designed customer hardware and software.

Time Sharing

The fourth configuration, PDP-10/40, incorporates 16,383 words of memory, extended order code and the memory protection and relocation feature. The PDP-10/40 multiprogramming/time-sharing software has been in successful operation for the last two years. It includes facilities for real-time and batch processing.

The final configuration, the PDP-10/50, permits swapping between 32,768 words or more of core memory and fast access disk file via the multiplexer/selector channel. This system includes complete multiprogramming time-sharing software.

Software

PDP-10 utilizes five distinct levels of software keyed to a variety of processor and/or core memory options. All software systems have been designed to expand PDP-10 performance easily to include any standard option without requiring costly special reprogramming.

All software systems assure upward compatibility from the standard 8,192 words of memory through the multiprogramming and swapping systems at both the symbolic and relocatable binary level. (For example, a FORTRAN IV program compiled on the basic PDP-10 system can be link-loaded and executed under time sharing without the need to recompile.)

The software package includes real-time FORTRAN IV, a control monitor, a MACRO assembler, a context editor, a symbolic debugging program, an I/O controller, a peripheral interchange program, a desk calculator, and library programs. All attainable via the DEC conversational mode.

In addition to software compatibility throughout all configurations, the system's programs are device independent. They obtain their I/O services through calls to the same I/O package used by the FORTRAN object programs and accessible to any user's program. Also, designed into each of the software systems is the ability to incorporate service routines for real-time applications and/or non-standard I/O devices.

The PDP-10 system is expected to find wide acceptance in applications such as physics and biomedical research, as a departmental computation facility, in simulation and aerospace, process control, chemical instrumentation, display processing, and as a science teaching aid.

PDP-6 SYSTEM AT LRL, LIVERMORE

John G. Fletcher
Lawrence Radiation Laboratory
Livermore, California

Abstract

An information-handling system, built around a pair of PDP-6 processors and their 256K memory, is being implemented at the Lawrence Radiation Laboratory, Livermore. These processors are being modified to use segmented and paged addressing. This system will control communication between teletypewriters, display and printing devices, and six computers as large or larger than an IBM 7094; it will maintain files using disc, data cell, and photo-digital storage devices, the latter having a capacity of 10^{12} bits; and it will engage in time-shared program execution, supplementing an existing system operating on CDC 6600's.

At the Lawrence Radiation Laboratory in Livermore, we are currently attempting to involve most of our computer complex in a single system, centered around the pair of PDP-6 central processors and their 256K-word memory. The functions of this system may be divided into three areas: communication, file maintenance, and time-shared processing. Each of these will now be considered in turn.

The communication system is called Octopus. The PDP-6's will interface with a large number of computing and communication devices available at the Laboratory. These devices include six large computers: two CDC 6600's, an IBM 7030 (or Stretch), a CDC 3600, and two IBM 7094's. These computers are used chiefly for production computation of large numerical problems. There are also two small computers: an IBM 1401 and a CDC 160-A. These are used chiefly for small I/O-related computations. And finally there is a PDP-8, which will be used to control a large on-line data collecting network. In addition to the computation devices, there are input-output devices, such as teletypewriters (perhaps as many as 1,023), a high speed printer capable of printing 500 lines per second, various plotting and display devices such as a DD-80 and Cal-Comp plotters, tape drives, and card reading and punching devices.

The PDP-6's will behave as a giant switch or dispatcher routing messages between these various devices. For example, teletypewriter messages may be used to interact with the time-sharing mode of operation available on the 6600's, or teletypewriter messages may be used to insert jobs into the batch monitor inputs of the other computers, or the output from various computer operations may be routed to the high-speed printer. The communications system itself will not have to depend upon the proper functioning of any of these various devices other than the PDP-6's. The computer complex maintains a modular, and therefore flexible, arrangement. Any failure in one device does not interfere with the communication activities of the other devices, unless of course both PDP-6's fail.

The control language that will be used to direct Octopus operations from the teletypewriters will be a compilable language. Subroutines written in this language may be called by running computations. Thus it will be very easy for computations themselves to simulate teletypewriter inputs and thereby run one another in a very complicated way. It might be, in fact, possible for the user to reduce all the things that he normally does at the teletypewriter to a few routines which he will define himself in terms of existing routines. When he sits down at the teletypewriter, he must only type a few lines, and everything else will automatically occur for him.

The file maintenance system is called Elephant. The file storage devices which will be available in the system include, in addition to the core of the PDP-6's, a high speed disc, capable of transferring at the rate of 20 megabits per second and having a capacity of .8 billion bits. This disc does not have moving heads, but only electronic switching between them. Therefore the only delays of significant magnitude in accessing the disc have to do with the rotational de-

lay. Average random access time is 35 milliseconds, but overlap scheduling will permit essentially continuous operation. There is also an IBM Data Cell, perhaps in the future several such Data Cells. Each Data Cell unit has a capacity of about 3.2 billion bits. Random access time, however, is of the order of half a second and overlapping is extremely difficult. Finally, there is an IBM Photo-Digital Mass Storage Device. This device has a capacity of over 1 trillion bits! This device stores information on film chips, which are kept in small boxes that move pneumatically from their storage locations to reading heads. It is a write-once device, in that after a film chip is written, it must be developed and thereafter can only be read. Random access time is very high, approximately 5 seconds on the average. However, extensive overlapping is possible and the average reading rate is approximately 1.5 megabits per second. So as long as there is provision for keeping the computers busy with other jobs during the 5 second delay in accessing a single file, this random access time causes no difficulty whatsoever.

The various computer outputs of the computers in the Octopus network, or various messages originated by users at teletype-writers, can be stored in this vast filing system. It is hoped that extensive use of this facility will drastically reduce the consumption of magnetic tape, printer paper, and punched cards at the Laboratory. An interesting statistic in this regard is that the printer paper consumption at the Laboratory can be measured in the hundreds of miles per month.

The manipulations within the filing system will be largely user-invisible. Users will know their file by name; the exact location of their file in storage at any moment will not be of any interest to them, but they can if they wish find out where it is. Since with a large computer complex such as ours, there is considerable liklihood of duplication in file names, in order to implement this name-oriented filing system, we have decided to introduce a directory system somewhat similar to what is proposed for the MULTICS project. Each user will have a file associated with him called his root directory. This directory will contain a list of the names of various files to which he has access. Some of these files may in turn themselves be directories, which in turn may list further directories and so on, there being a tree-like structure leading from the root directory to the various files that are to be actually used in computations. A user has no way to access a file not listed in this "tree". To reference a file, the user gives what is called a chain name. That is, he gives the name of a directory as it is listed in the root directory, then the name of a directory as listed in that directory, and so on until he gives the name of the file itself. Of course there will be ample provision in the control language for changing from the root directory to one of the sub-directories, so as not to have to continually repeat the beginning portion of a chain name. Users can share sets of files by sharing a directory listing them, and names need only be unique within each directory.

An entry in a directory contains a number called the file pointer. This number will not, however, necessarily directly describe the location of the file in the filing system, since the file may move from device to device. A file that is little used may remain on the mass storage device, but when it is referenced it will be moved up to the disc, so as to be available for use. With each filing device will be associated an index, a list of the files currently resident in that device together with their locations. When a file is requested, the proper directory entry will be used to obtain the desired pointer. This will then be checked, first of all against the disc index, then against the data cell index, and then against the index of the mass photo-storage. If it is found in none of these, it is presumed that the file has had to be moved to the shelf, it being possible to actually remove the film chips from the photo-storage. In this case, the file pointer does actually describe the location of the file on the shelf.

Each file will have a lifetime, which may be declared by the user. Typical lifetimes might be, say, four hours, two weeks, or essentially forever. Whenever a file is not referenced for longer than its lifetime, it is destroyed. By having this feature of automatic destruction, we prevent the wasteful accumulation of unneeded files. For example, such things as diagnostic files from Fortran runs, which users often forget about, unnecessarily clog the present storage system of the 6600 time-sharing system. Since the user may never make reference to such files, he will then of course never have any occasion to decare them to have a lifetime longer than the minimum (e.g., four hours). Therefore, they will automatically be destroyed, it being a reasonable assumption that if he doesn't refer to a Fortran diagnostic within 4 hours of production, he probably has no use for it. If he does have use, of course, he can specifically request that the file be kept longer.

We envision that several users will want to make access to the same file. Obvious examples are the public files, containing the standard arithmetic routines, file manipulating routines, and what-have-you. It is very undesirable to have users capable of

randomly modifying such files. Therefore all files will be distinguished by having what are called access bits. A file can be marked so that it may not be written, but only read. In fact, we have even provided for files which may not even be read, but merely have the coding in them executed. Such a feature would, perhaps, be very desirable in a commercial environment, where persons may wish to sell programs that they have written but are somewhat reluctant to allow other people to see how they are written. Of course these access bits depend upon the user. The system programmer who originally writes a particular utility routine should retain the ability to rewrite it in the event that a "bug" is discovered or an improvement is devised.

The time sharing system of the PDP-6's is called Alligator. This time sharing system, at least initially, will be independent of, and supplemental to, the time sharing system already running on the 6600's. There are several reasons why we would like to remove some of the time sharing load from the 6600's, one of which is that time-sharing seems to be a poor use of the great speed of these computers. Time-sharing is to some extent interfering with the execution of the very long and large numerical calculations which must be necessarily done on these large machines. We intend that such things as file manipulation and, hopefully, short arithmetic or desk-calculator type computations will be done on the PDP-6's.

Our time-sharing system uses many new and modern features which are supposed to be desirable in a time-sharing system and which we are going to try in order to find out how desirable they really are. In order to implement some of these modern features, we have had to make modifications in the PDP-6 in regard to its relocation mechanism in user mode. It might, therefore, be wondered why we chose a PDP-6 as the central computer of our system. One answer is that the computers with the various relocation features which are to be described were not available when we were making our decisions. But even beyond that we find the PDP-6 has a variety of very useful instructions for a communication-type computer. Such things as the byte manipulation, half-word, and test instructions are very valuable indeed. Also in the multi-processing environment we have with two processors operating in the same memory, we have found that instructions like add-one-and-skip (AOS) and subtract-one-and skip (SOS), which read a word, modify it, and test it all at once, are almost essential to efficient programming. Another desirable feature of the PDP-6 is its general interrupt facility and its ability to interface with a large variety of asynchronously operating I/O devices. The interrupt facility relieves the PDP-6's from the necessity of having to perform continuous or periodic interrogations of various data channels.

Various features of our time-sharing system, which will in many cases serve the communication and file maintenance systems previously described, are as follows: First of all, we have a paged core; that is, the core is divided into blocks, and hardware modifications permit a single computation to be located in non-contiguous blocks without the programmer having to be aware of it and without any actual change in the words stored. The hardware performs automatic relocations, selecting the proper area of memory to which an address refers. What is hoped to be gained by pagination is that it will be possible to more efficiently utilize the core of the PDP-6's. Present experience in the 6600, which offers a feature of automatic relocation almost identical to the standard relocation features of the PDP-6, is that at any one time there may be sufficient core for one, or perhaps several, jobs in addition to the ones that are currently running in the time-shared mode. However, this core may not consist of one contiguous block of core, but of several pieces of unused core separated by the core being used by the jobs that are already running. The only alternatives that one has under these circumstances are, either, that the new jobs are denied the use of the computer until one or more of the other jobs has left the core, or else, that activity must stop for awhile and the core be repacked so as to bring the vacant areas into one contiguous block.

Paging not only makes it possible for a job which appears to the user to consist of a contiguous piece of core to, in fact, be spread around the memory of the computer, but it also makes it possible that the entire job not be in core at the time at which the job is running. If during the course of the execution, the running program makes reference to a portion of the code which is not in fact in core, an automatic interrupt occurs. The executive system is called; it retrieves the missing page, presumably from the disc, and then returns control to the program that required the page.

In addition to being paged, our machine provides for segments or two-dimensional addressing. The address of a word in a computation consists of a segment number and then a word number within the segment. This word number, of course, is actually broken into a page number and a line number by the paging mechanism, but, as previously remarked, this is user-invisible. Segmentation offers further capabilities in the area of better core

utilization. An interesting example of the situation now occurs, say, with the Fortran compiler. It may be that several of the jobs running in the 6600 at one time are, in fact, compiling Fortran. For each one of these jobs, a distinct copy of the Fortran compiler must be brought into core. Moreover, a job of higher priority may arrive, necessitating removal of one of the jobs currently occupying core. If the job which is removed, and the job for which it is removed, are both intending to compile Fortran, the situation obtains that the Fortran compiler is removed from core in order to make room for the Fortran compiler. By having two-dimensional addressing, we make it very easy for users to share coding. Two users may use the same block of coding in core but refer to it by different segment numbers. Therefore, the same coding is useful to both programmers even though they do not think of it as having the same address. Such programs as Fortran compilers, assemblers, and various utility routines, such as ones which manipulate files, can be shared by all users who desire them at one time, and considerable savings in core should result. Of course, the segment number plus the word number must exceed the 18 bits available in the address space of the PDP-6. Therefore we have made some modifications in the PDP-6 addressing scheme, but discussion of this detail does not seem appropriate here.

Another feature we have is dynamic linking. The segments of a computation performed by a user are not all initially loaded; only a few segments are loaded. Then, whenever a reference is made to a segment which is not currently loaded, a trap occurs and the segment is retrieved from the files. This is a further step in attempting to reduce burdens on the core and on the filing system. That is, a segment that is never used need never even be retrieved from the files. An example might be error routines, which need only be loaded if the error actually occurs. There is of course a further hardware modification that makes this automatic trap for the dynamic linking possible.

A side effect of having segments is the ability to have expandable arrays. An array can begin in a certain segment and then may grow to the full possible length of the segment (which is 32K words, incidentally), without having to worry about initially setting aside the entire space. In fact, a growing array is initially a segment having no length whatsoever. As entries are made into it, pages are created for them, and the segment gradually grows a page at a time to the maximum possible size.

Mention has already been made of the fact that we will have parallel processing available in our system. We really have more than one central processor working in the same memory. We envision that they will actually run simultaneously. In fact it will be possible for one user to start two parallel processes of his computation and perhaps, if the traffic load is low, to get them actually running in parallel, one beside the other, rather than, as is the case in most time-sharing systems, alternating with each other in a single processor. We, of course, will have the latter capability also.

The fact that users will be able to share segments means of course that considerable attention must be paid as to how procedures are written. A procedure should not modify itself as it runs, because then if two users are simultaneously computing with it, it will get a bit confusing as to which one of the users is actually permitted to make the changes in the procedure. In fact, what we are really talking about here is pure or reentrant procedures, procedures which do not modify themselves as they execute. We are also providing for procedures being recursive, capable of calling themselves either directly or indirectly in ways that are probably familiar to list-processing programmers, but perhaps are not so familiar to persons accustomed to Fortran.

Of course, it is one thing to say that a procedure is pure and therefore will not write into itself, but it is another thing to say that a user will not accidentally or perhaps maliciously write into the procedure by treating it as a data segment. It is at this point that the access bits discussed earlier in connection with files have further importance. The hardware mechanism permits each page to have associated with it three bits which indicate respectively whether one may write into the addresses in the page, whether one may read from them as data, or whether one may read from them for instruction fetch. These bits are called respectively, the write, read, and execute bits. Clearly, pure procedures will be marked as execute, or perhaps execute and read, but not as write; thus it will be impossible for a person sharing this pure procedure to modify it and thereby disturb other users. It is through the use of these access bits and the various protections offered by the hardware mechanism which manipulates segments that we prevent users from interfering with one another even while they are actually sharing coding with one another.

We propose to solve our accounting problem by instituting the very American notion of capitalism. Each user is given a bank account of some arbitrary credit units which he may spend as he sees fit for the various system resources, each one of which has a

price. That is, he will be charged so much for use of storage space, for use of processor time, for use of I/O devices. The prices of these devices will be set automatically by the system on the basis of use and will be revised periodically, say weekly or monthly. As the device proves more and more popular, its price will gradually rise; this will discourage people from using it and encourage them to use less used devices, the price of which may gradually lower. A user who desires extra service, that is, who wishes to be served more rapidly than the system would normally serve him, may voluntarily raise the price he pays for a device; that is, he may spend his capital more rapidly than normal in exchange for this increased service. By having one medium of exchange, the number of the decisions burdening the supervisory staff is cut down considerably. They do not have to decide how much of this, or how much of that, each user is entitled to; they merely decide what his overall worth is, and he then allocates his resources as best suits his needs. The supervisory staff also has the problem of deciding, when the price of a particular device has risen extremely high, whether it should be raised further or whether perhaps an additional device should be bought. Conversely, when the price of a device drops to zero, one might consider whether or not it would be reasonable to dispose of it.

The system described here is in process of being implemented now and should begin operating in a fairly complete fashion about a year from now. Much of this delay is in waiting for the delivery of some of the hardware devices, particularly the mass storage device. Needless to say, there are a great number of people, both hardware and software experts, involved in the design and implementation of this system. Not all of these people work at the Laboratory, since we have taken the liberty of using what we consider to be the best ideas available in the published literature.

A COMMON MEMORY SYSTEM FOR A TIME-SHARED PDP-6

E. Brazeal, S. Sharpe
United Aircraft Research Laboratories
East Hartford, Connecticut

Abstract

A multiprocessor capability has been added to United Aircraft Research Laboratory's time-shared PDP-6 computer. A flexible interface was designed to realize this capability. The hardware, "MEMTIE," controlled from the I/O Bus of the PDP-6, allows the existing PDP-1 computer to share a type 164 memory module with the PDP-6. This common memory system provides the communication potential for various multiprocessing applications.

This paper describes the combined computer facility, the design requirements for the interface, some details of the interface, and certain planned applications for the system.

Introduction

The past several years have seen an expansion of digital computation capabilities within the Simulation Section of United Aircraft Corporation's Research Laboratories. The first digital machine acquired by the Simulation Section was a PDP-1. This machine soon saw heavy use in hybrid computation and scope display work utilizing its interfaces to a Beckman 2133 analog computer and a DEC type 340 scope.

The aquisition of a PDP-6 time-sharing system has expanded the Laboratory's capacity to do hybrid computation (through a hybrid linkage interface to four analog computers) and includes multiuser teletype service offering a multitude of system programs (editors, compilers, "desk" calculators, debugging programs, etc.), and a multiprocessing interface to the PDP-1 through a common memory system.

This paper will discuss the justification, design, and application of this multiprocessor interface.

It was considered desirable to include a scope display I/O device, such as the existing type 340, in the PDP-6 system. However, the nature of the problems encountered implied a great demand in processor time required to service the display. If the scope were connected directly to the 166 processor, a prohibitive load would be put on the time-sharing programming system (especially considering the real time response demanded by hybrid simulation). For this reason and because much usable display software already existed for the PDP-1, another scheme utilizing the PDP-1 as a display processor was decided upon as the best approach.

MEMTIE was designed as a hardware interface to realize the goal of developing a high-speed, highly efficient scope display for the PDP-6 system. As illustrated in Figure 1, MEMTIE appears as a second processor to the PDP-6 system with connection to the 166 processor, the I/O Bus, the PDP-1 and 164 memory module III.

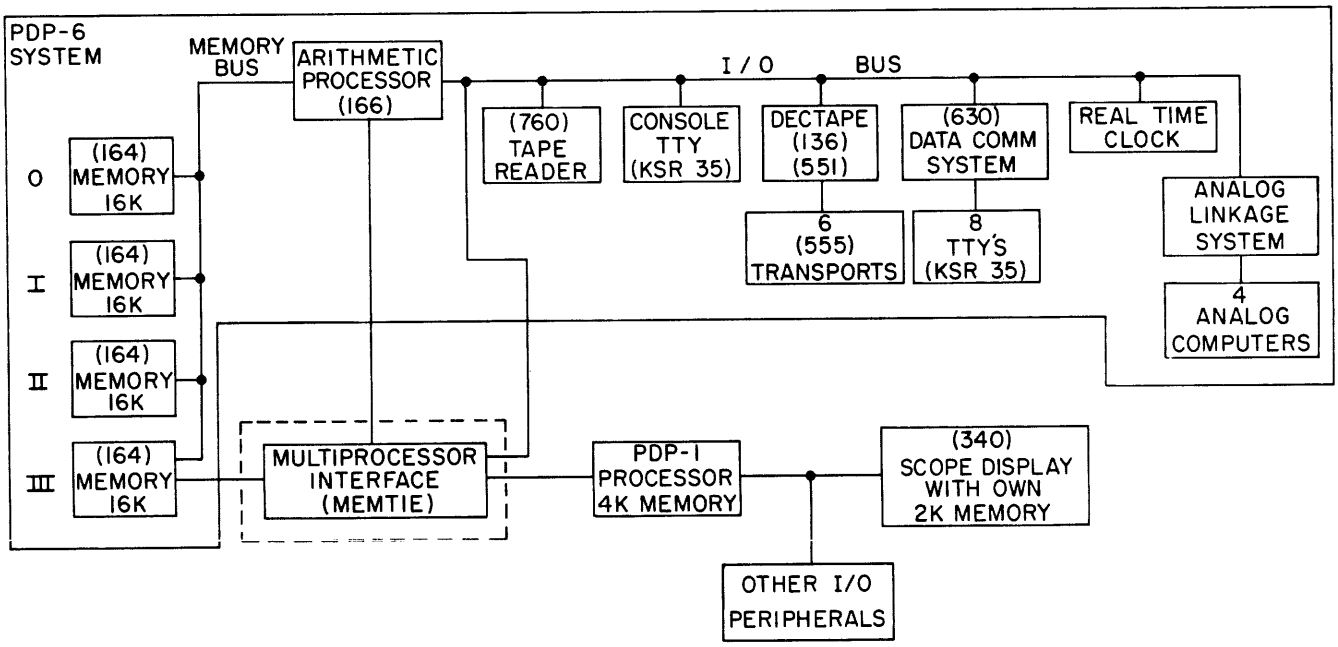


Figure 1 System Block Diagram

Although the major goal of MEMTIE was to implement a scope display system, many other applications have been envisioned. Some of these include hybrid computation with calculation loops running concurrently in both machines (to enhance real time simulations), utilization of the I/O facilities (other than the scope) of the PDP-1 from the PDP-6 (or vice versa), and extended memory operation of the PDP-1 alone.

PDP-6 System Considerations

A multiprocessor type of operation within the PDP-6 time-sharing system can generate some difficult problems in the areas of protection, relocation, and general control of the common memory area. It was decided that the PDP-6 should have control over MEMTIE. Design of this control included some hardware modifications to the 166 processor and some software modifications to the Time-Sharing Monitor[1] program.

It was determined that MEMTIE should be as flexible as possible. This requirement was derived from the design of previous devices that were interfaced to the PDP-6 (such as the Analog Linkage System[2]), and from the variety of potential multiprocessor applications. To achieve this flexibility, several design requirements were established for MEMTIE:

1. It must be a program controlled device, accessible by a PDP-6 system user through interpretive command instructions executed by the Time-Sharing Monitor.

2. The amount of "shared" memory between the two processors must be variable and controlled by the Monitor. This "shared" memory must be protected by hardware and software from transfers initiated by non-multiprocessing PDP-6 users.

3. Use of the normal and extended modes of operation on the PDP-1 should simulate that of the DEC type 15 Memory Extension Control.

4. Error detection circuitry should be included.

The PDP-6 time-sharing system has been set up to fully utilize the relocation and protection hardware available within the machine. This logic is under the control of the Monitor.

As each new user enters the system, his core request is processed in accordance with what is available within the system. If his request is granted, the Monitor will reserve an integer number of consecutive 1K blocks for the user. Also, a "job data area" will be established in the first 140_8 locations of the user's core for system bookkeeping. The user's n-block core assignment is relocatable; that is, the relative addresses used by the user's programs do not necessarily equal the absolute addresses within the machine. Relocation of a user's addresses is accomplished by a "relocation register." The state of this register is under control of the Monitor. In time-sharing, each user's program is run for some relatively small amount of time after which the Monitor switches control to another user's program. Before switching control, the Monitor changes the relocation register to the proper quantity needed for the next user's program. If one of these users (the "multiprocessor user") is to have access to the core common to the PDP-1, significant problems arise in connection with addressing. One obvious solution is to furnish the relocation constant to the interface so that the PDP-1 will absolutely address the same core area that has been assigned by the Monitor to the multiprocessor user. Unfortunately, this scheme requires a relatively large amount of hardware and Monitor modifications to implement.

A simpler scheme was devised in which a fixed area of core was chosen for multiprocessor use. This core area was defined physically by the highest possible addresses in the PDP-6 memory system, i.e., the "top of core." Memory "shuffling" of jobs by the Monitor is such that unused core is always at the top. Since the addresses of this special area are always the same, the addressing logic for the PDP-1 is much simpler.

Connection between the multiprocessor

user's job and the MEMTIE core must be maintained even though his job may be absolutely located elsewhere in core. This is accomplished through the use of logic circuitry that enables conditional suspension of relocation and protection. The scheme is implemented by defining the top of core with two sets of addresses. The first set contains the regular sequential addresses in the system, but the contents of the second set correspond to the highest possible locations the PDP-6 can address. When the suspension circuitry is enabled, the multiprocessor user's program can access this special core area using the high address set without violating protection or relocation bounds.

The PDP-1, an 18-bit machine, can directly address 4K of memory. The Extended Memory option allows indirect addressing of 32K. In a 32K PDP-1, the memory is divided into 8 modules, numbered 0 to 7. Module 0 is defined to be the 4K memory within the PDP-1. Taking module 0 into account, it can be seen that the maximum amount of core that can be addressed by the PDP-1 (in terms of 36-bit PDP-6 words) is 14K.

In MEMTIE, a 3-bit register, known as the "mode" register, is used to control core assignment and address bounds of the common memory area. A total of four "modes" exist and are summarized in Figure 2. Note that as the mode is increased, the core gained by the PDP-1 is twice that used within the PDP-6 system. This is due to the fact that two 18-bit PDP-1 words are packed into one 36-bit memory location.

Of major concern were the addressing requirements of the PDP-1 under this flexible core arrangement. The PDP-1 user would like to see additional memory modules added to his present system as the mode is changed to a higher value. Accordingly, addressing to the 164 memory from the PDP-1 is coded by MEMTIE in the following way:

1. PDP-1 MA Extended bits, 3-5, are exclusively or'd with the one's complement of the mode register to become MA bits 22-24 to the 164 memory.

2. PDP-1 MA bits 6-16 become MA bits 25-35 to the 164.

3. PDP-1 MA bit 17 becomes the "left-right" bit, that is, the half of the 36-bit word in the 164 that contains the 18-bit PDP-1 data is selected by bit 17.

4. For system consistency, MA bits 0-3 to the 164 are permanently set to logical one.

To illustrate these ideas, Figure 3 has been prepared. The core maps show the arrangement of PDP-1 core as a function of the mode. The numbers to the left of each map indicate the two possible octal addresses that define the core locations. Note that in all modes, the lowest location in PDP-1 core is just above the highest location in the PDP-6 system.

Control of MEMTIE by the PDP-6 is accomplished with two executive mode I/O instructions; that is, instructions that are executed only by the Monitor. The first of these instructions can load the "mode" register and enable or disable the suspension of relocation circuitry. The second instruction can read the status of MEMTIE into the Monitor, i.e., what mode the system is presently in, etc.

The Monitor has been modified to execute the above instructions at the beginning of each job. For the multiprocessor user job, the Monitor enables the suspension of relocation circuitry, but for all other jobs, it disables this circuitry. During the initialization of the multiprocessor user's job, the Monitor loads the mode register in MEMTIE in accordance with the user's request and "sets aside" this special core area rendering it unavailable to the rest of the system.

PDP-1 Modifications

The PDP-1 is controlled by a timing chain of sequentially connected delay units rather than a fixed rate clock. The chain is composed of eleven timing pulses designated TP0 to TP10 (actually twelve pulses

MODE	MODE REGISTER STATES	CORE USED 164	CORE GAINED PDP-1
Ø	Ø, 2,4,5,6	Ø	Ø
1	1	2K	4K
2	3	6K	12K
3	7	14K	28K

Figure 2 Mode-Core Relationship in MEMTIE

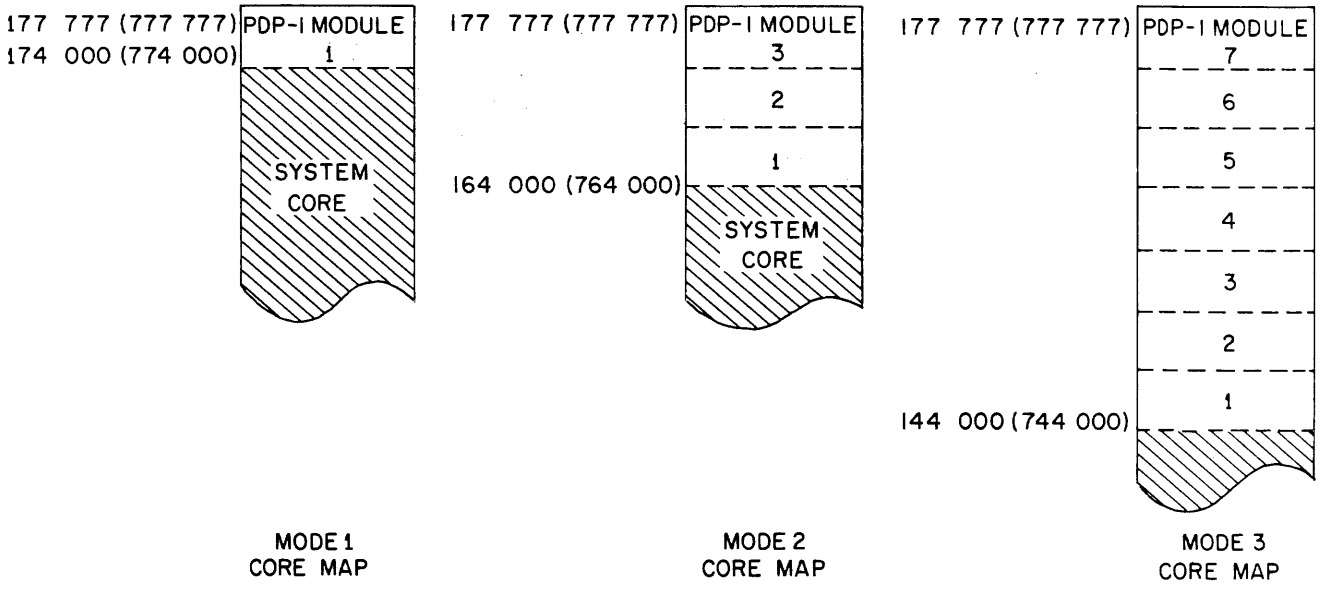


Figure 3 MEMTIE Core Maps

if TP9a is included). Requests for memory cycles are generated by TP2. The PDP-1 memory responds to this request with a "strobe" pulse that generates TP4, TP5, and TP6. TP2 generates TP7 (through TP3) after TP6 has occurred, thus insuring continuity of the chain. Due to the connection of TP3 to TP7, the timing of TP6 is constrained. If the memory should be "late" with its "strobe" pulse, TP6 may occur after TP7. This possibility is highly probable in a memory that is shared by two or more processors. The PDP-1 timing chain was modified to prevent this possible error from occurring. The existing connection between TP3 and TP7 was broken and a new connection between TP6 and TP7 was made. The delay between TP6 and TP7 was set a 0.4 microseconds to satisfy the 5 microsecond requirement on PDP-1 cycle time. Figure 4 illustrates the timing chain modifications.

Logic to detect error conditions in the interface to MEMTIE was incorporated into the PDP-1. Error conditions can occur during a request for a memory cycle. If an error condition does occur, the memory request is not made. Instead, the following sequence of events occurs:

1. An error flip-flop is set.

2. The Run flip-flop is cleared.

3. TP4 is generated to finish the cycle.

There are two possible error conditions:

1. Illegal Cycle - MEMTIE has a mode assignment other than 1, 2, or 3 when a memory request to other than module 0 is being made. Or, a memory request to a memory module that is not defined by the existing mode is being made.

2. Memory Fail - lack of a "read restart" pulse from memory within a time lapse of 300 microseconds will produce this error condition.

As we have seen, the hardware design for implementing the multiprocessor capability on the PDP-6 system included modifications to the PDP-6 type 166 processor and its time-sharing programming system, modifications to the PDP-1 control logic, and the actual device (MEMTIE) that ties the two processors together.

Scope Display Application

It is now appropriate to discuss an application of the multiprocessor system. Our goals for a scope display system were defined as follows:

1. Manipulation of display data should have a minimum effect on the PDP-6 processor, which is doing the calculation, since many of our problems require real time response from the computation.

2. Efficient display maintenance is required. A good system would display the maximum number of raster points without flicker. Also, when new information is available for display, the picture would be refreshed and restored quickly.

3. Storage of display data should be efficient, requiring as few peripheral words as possible to specify and identify the scope formatted data. Easy manipulation of data in terms of entities being displayed is required as well as convenient identification of entities sensed by the light pen. A minimum amount of processing should be needed to output data to the scope as the picture is being displayed.

4. The display should communicate with a user's program on the highest possible level. That is, to use the display, a program would specify only the name and in general terms the characteristics of each entity to be displayed. The display would do the detailed work of setting up the data in proper scope format and seeing that the specified entities are displayed. If then, for example, the operator should touch an entity with the light pen, the display system would note this fact and provide the name of the specified entity to the user's program.

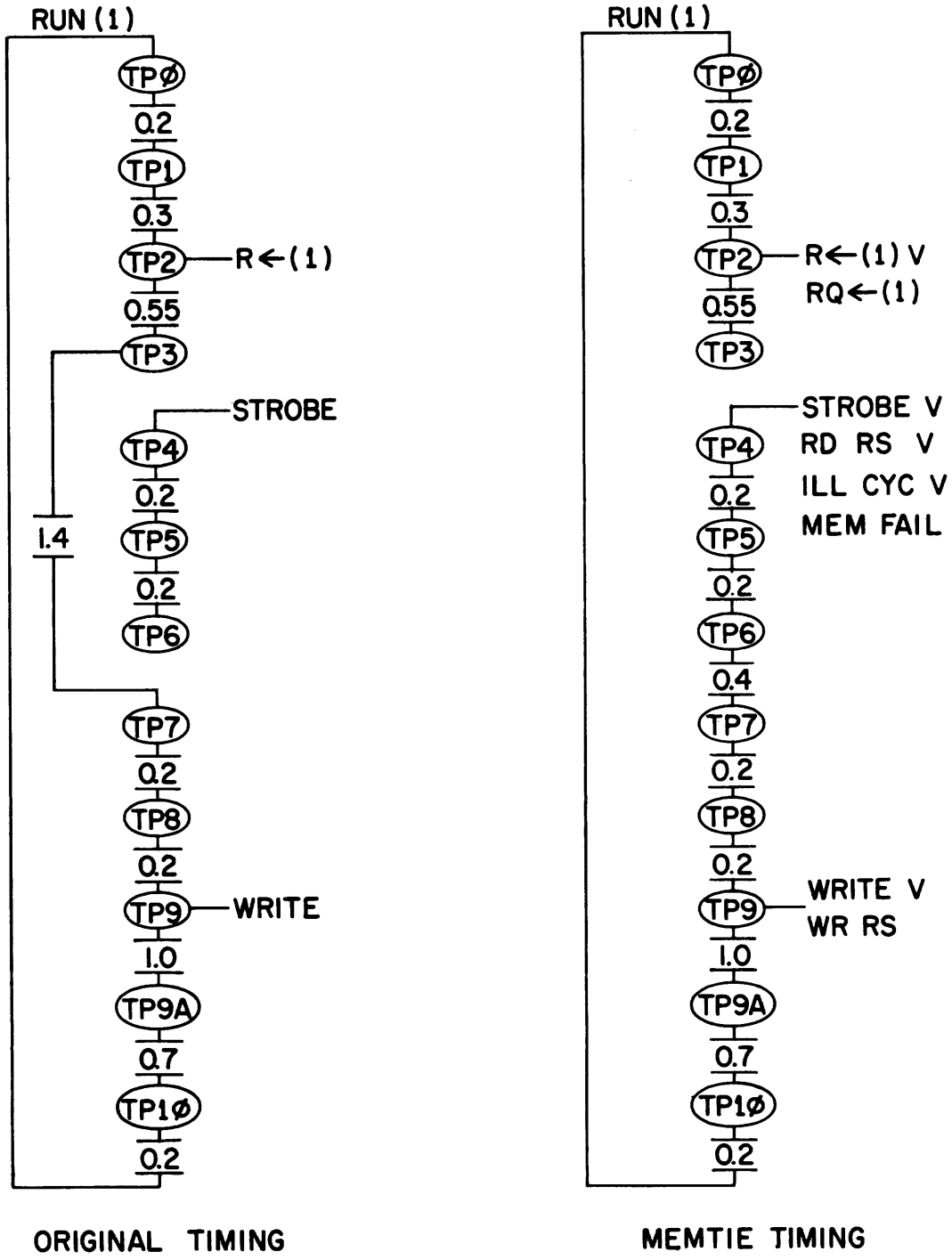


Figure 4 PDP-1 Timing Modifications

The PDP-1 processor, connected to the PDP-6 system through MEMTIE, provides an excellent method of doing the display generation and maintenance activities. As such, the PDP-1 processes inputs (things to be displayed, operations on things that are being displayed, etc.) and outputs (information on light pen identification, pentracking, and other PDP-1 interrupts) for a calculation running in the PDP-6. The PDP-1 also organizes and updates all data in scope format and controls the transfer of data to the scope as required. This operation of the display would be quite independent of the calculation, which need only organize the items for display and decide what operations it wishes to perform.

In order to enhance the rate at which data can be transferred to the scope, a high-speed data channel (HSDC) has been added to the PDP-1. Control of this device is accomplished in such a way as to provide good display rates and still account for efficient storage of display data in terms of entities being displayed on the scope.

Figure 5 illustrates this data layout and display control. The scope data is organized in terms of entities or "things" which are to be displayed. Each thing requires three words in addition to those needed for raw scope data. These are:

1. An entry in a list of thing addresses which contains a pointer to the data in scope format. This table of pointers becomes a list of all the things which are currently being displayed.

2. A control word which signals the hardware to go on and display the next thing in the list of thing addresses.

3. A name which provides identification of a thing in case of a light pen interrupt.

With this information two hardware registers control the output of data to the scope. As data is requested by the scope, scope data is transferred from the contents of the location specified by the HSDC address register to the scope. Then the register is incremented by one in traditional HSDC fashion. First, however, the current scope data is tested. If it is the special control word, then the contents of the address specified by the HSDC "thing" counter are transferred to the HSDC address register before the HSDC can honor any more requests for scope data. Then the HSDC thing counter is incremented by one. In this way the hardware can step through a list of things to be displayed while requiring no PDP-1 processor time. Then only a few instructions are required to restart the HSDC for another cycle through the display. Since the HSDC thing counter can be read using a PDP-1 I/O instruction, the identity of the thing currently being displayed is quickly established in the event of a light pen interrupt.

Thus, data is transferred to the scope rapidly, scope data is stored efficiently in terms of displayed entities, and things are easily identified upon light pen contact. A minimum amount of PDP-1 processor time is required to refresh the display, leaving the computer free to process new information flowing to and from the calculation in the PDP-6 and to perform other tasks such as interrupt handling, pentracking, and garbage collection.

The software for the scope display application consists mainly of a PDP-1 display monitor program called Scope-6. This monitor provides a high-level software interface between a PDP-6 program and the PDP-1 scope display system. The interface is "operational" in nature; that is, Scope-6 accepts output from the PDP-6 in terms of operations. The operations include taking data (in forms used by the PDP-6 calculation) representing things to be displayed, processing the data to form identifiable scope formatted data for the things, and maintaining the display of the things which exist. Different kinds of things may be used. A list of points which are to be connected by straight lines, or a string of alphanumeric characters are two examples. Once things have been formulated, further operations are possible. For example, scale or intensity may be changed, a thing may be moved on the screen by changing its initial

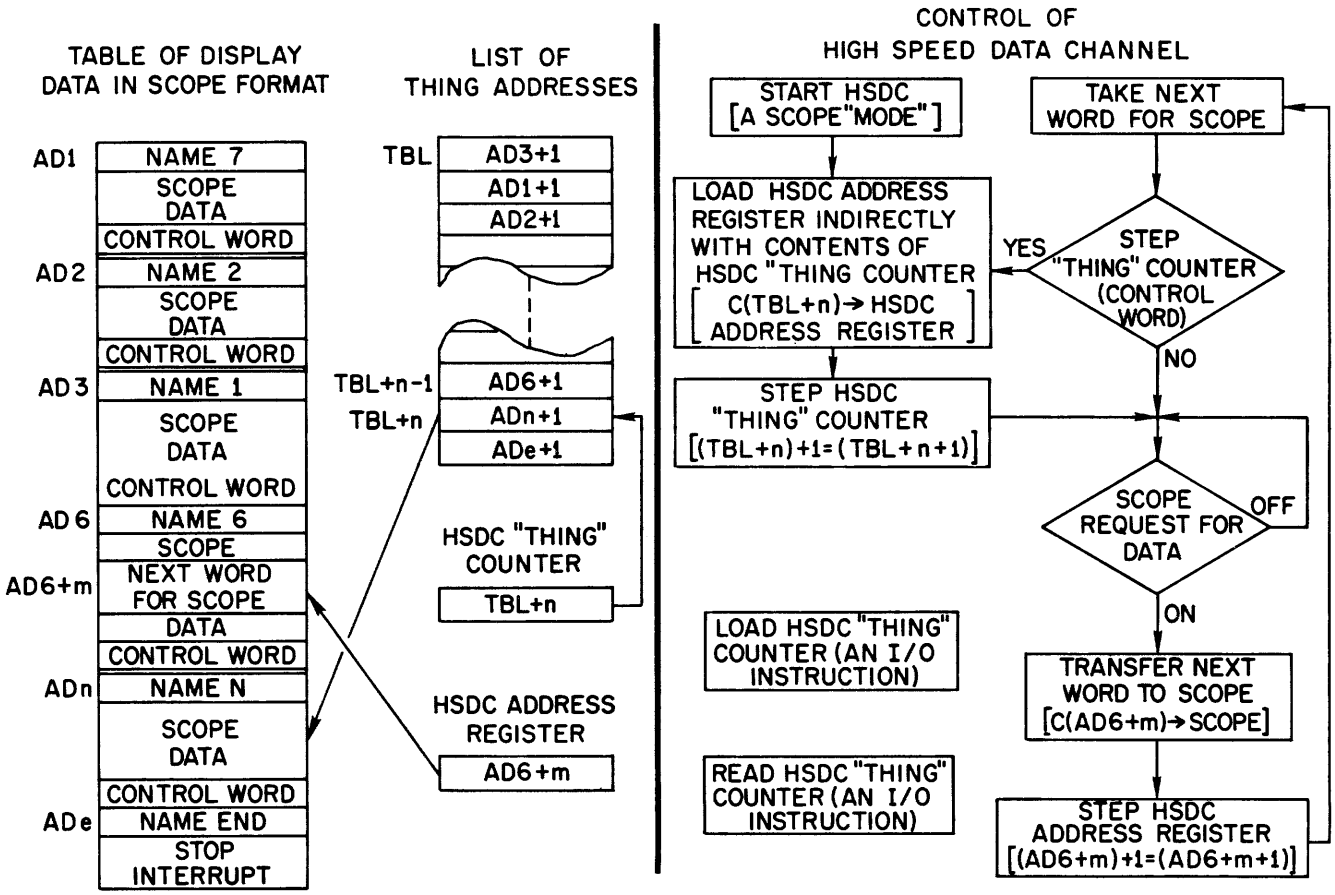


Figure 5 Scope Data Layout and Display Control

position, it may be deleted from the display but not from the data structure, or it may be deleted entirely. These operations provide the opportunity for the PDP-6 to set up and manipulate things on the display.

Besides operations on things, the PDP-1 handles certain other tasks. Interrupts having to do with the display console are processed. When an operator touches a thing with the light pen, certain data is recorded in the common memory area. This includes the name of the thing touched, the raster point where it was touched, and the status of a software flag. Scope-6 also provides for setting and checking the status of a push-button control panel, located near the scope, allowing convenient selection of various options in the user's program. Pen-tracking is available with full knowledge of the location of the light pen within the PDP-6 program.

Thus, including a multiprocessing capability within the time-sharing system has allowed development of a convenient means trough which a PDP-6 program can collect and manipulate graphical information on the scope display, and provide for real time operator interaction with the display while remaining capable of highly efficient display maintenance.

Acknowledgements

The authors wish to acknowledge the work of Ronald Gocht under whose direction the MEMTIE project was carried out, Gabriel Rosica who supervised the detailed engineering of the hardware, and Stephen Jackson who has made the necessary Monitor modifications.

References

1. PDP-6 Multiprogramming System Manual, Digital Equipment Corporation, Maynard, Massachusetts, 1965.

2. R. Belluardo, R. Gocht, and G. Paquette, "A Time-Shared Hybrid Simulation Facility," Proceedings, 1966 Spring Joint Computer Conference.

3. Programmed Data Processor-1 Maintenance Manual, Technical Bulletin F-17, Digital Equipment Corporation, Maynard, Massachusetts, 1962.

ON-LINE REAL-TIME TIME-SHARING OPERATION OF A PDP-7

Lloyd Robinson
John Meng
Lawrence Radiation Laboratory
Berkeley, California

Abstract

Both hardware and software developments which permit the simultaneous use of a single PDP-7 for the taking and analyzing of data by up to eight experimenters are described. Hardware includes a memory protect system which guarantees the sanctity of switch-selected areas of memory and a unique set of remote consoles. Some programs to be used with the system are still under development, but presently operating routines include ones for storing data on IBM-compatable tape and DEC tape, routines for displaying data and operating the remote consoles, and routines for approximating curves with third-order polynomials. Routines are also in use which produce Cal-Comp point plots or histograms.

Introduction

The object of the system described here is the replacement of up to eight pulse-height analysers with a single PDP-7. A body of commonly-used programs is available to all users on a time-sharing basis. They are called using specially-designed remote consoles via an executive monitor routine, and include simple arithmetic operations, timing operations, display manipulations and plotting operations. Also available are data storage on DEC tape and/or IBM-compatible tape. The data-taking areas of memory are protected from other data-areas by a hardware-implemented memory-protection scheme. Protection of data areas from programs is provided by the executive program which monitors trapped commands. Trapped commands are those which have addresses in illegal areas of memory, and input-output commands. Two identical systems have been constructed for use in two different experimental areas.

Hardware

The PDP-7 system which was purchased for this purpose included 8K of core storage, extended arithmetic, a dual DEC tape unit, a real-time clock and display logic. Additional hardware which has been built at LRL includes a calcomp plotter interface, an IBM-compatible tape drive interface, a data-break multiplexer and memory-protection circuit. The data-break multiplexer accepts inputs from the IBM-compatable tape unit as well as from a separate multiplexer for up to eight ADC's. The remote-console units are interfaced to the accumulator.

Memory Protection

The function of the ADC is to produce an address (12 bits) corresponding to the voltage of an input pulse. The multiplexer for the ADC's then generates a signal requesting a data break from the data-break multiplexer. This then results in the incrementing of the contents of some memory location in the computer that corresponds to the code generated by the ADC. There is no program intervention in this chain of events, and consequently protection of one data area from another has to be under hardware control. Protection of data areas from programs is a different matter, however, and is placed under the control of an executive program. This is achieved by hardware monitoring of instruction addresses, and forcing the computer into the trap mode if an illegal instruction address appears. Trapped instructions switch control to the executive program.

Illegal addresses are determined in the same fashion in both cases, and in

fact the hardware is itself common to both. This consists of a box with ten groups of five switches on it. The switches determine the boundries of areas to be protected, and represent the five most significant bits of the addresses of such areas. The hardware allows the 8K memory to be divided into as many as 9 independent fields, in 256 word segments. Protection of data storage from other data storage is accomplished by adding the 12-bit ADC output code to the lower boundry of the data region. The result is compared with the upper boundry, and the "increment memory" is inhibited if the result is too large.

The comparator used is interesting in that is uses analog techniques. The digital address signals are changed to currents and the results thereby compared. The hardware required, consisting basically of nothing more than precision resistors and an amplifier, is considerably less than what would be required if digital circuitry were used throughout.

Remote Consoles

The remote console is another very interesting piece of hardware. One individual experimenter may be located just across the hall from the computer, but another one might be located three floors up and down at the end of the hall. In the latter case, the remote console becomes very important. The experimenter must be able to converse with the computer easily and reliably while adjusting his other equipment. The hardware provided is a CRT display plus a panel with numerous switches on it. The panel also contains lamps which are controlled by the computer. The lamps can easily be used to indicate the status of devices under program control.

The consoles require very little hardware, being entirely under program control. The console to be interrogated is enabled by a level provided by a corresponding bit (Bit #1-8) of the accumulator, and the console status is read into bits 9-17 of the accumulator, using only a single information collector interface for all 8 consoles. Bit 0 of the accumulator selects one of two switch banks in the console, so that an 18-bit code can be read. The console itself consists only of switches and a diode coding matrix. Development of a complete and efficient set of programs for an on-line use of the computer is a sizeable job. Efficiency in use of memory is of the utmost importance since each ADC is capable of generating 4096 channels of data, and there are only 8192 channels of storage in the machine. Of course, not every experiment will utilize the full 4096-channel capability of the ADC, but even so the system will undoubtedly be memory-limited as use increases. Additionally, this memory limitation forces the use of machine language programs in the operating system. The use of Fortran will have to be restricted to times when no experiments are being run on the machine. With these things in mind, let us look at the operating-system programs which exist and those which are planned.

Programs

Existing Routines

The routines presently operating may be classified as executive, utility and user programs. The executive monitors traps and the remote consoles, performs the necessary initializations at the very beginning of operating, and controls the execution of utility and user programs. More specifically, it reads the boundries of the data-storage areas at startup and prints them out, setting program pointers at the same time. It allows the display routine to operate during idle periods and reads the remote console switches periodically making the necessary alterations to display parameters. If the experimenter is demanding some peripheral operation, the executive services the demand (or returns a busy signal if the peripheral is in use.) A preset time delay is also available to the experimenter via the executive program.

Utility routines include the display routine, calcomp plot routines, a DEC tape storage routine, and IBM-compatable tape dump routine, a program for fitting a 3rd order polynomial to four points and subtracting the fitted curve from a marked data peak, and a typewriter routine for typing out data channels and values.

Probably the best method of ex-

plaining the operation of the above routines is by an example. Assume an experimenter has set up an experiment at one of the remote terminals where he wishes to take data for preset time intervals, compute the background curve under a peak and type out the sum under the peak and the difference between the sum under the background and the sum under the peak. Then he may wish to store each spectrum for future analysis. His operating sequence is as follows:

At the computer console, (assuming the program is already operating) he types the number of tenths of a second he wishes for his time interval followed by a T. (If he wanted to count events for eighty seconds, he would type 800 T.) Having done this, he can return to his remote station for the remainder of the experiment. At the remote station, the READY lamp is winking periodically indicating that all is well back at the computer. He clears his data-area by turning the appropriate rotary switch to ERASE and pressing the COMMAND push button. The display in front of him obediently becomes a straight horizontal line at the bottom of the screen. He turns the rotary switch to START COUNT and presses the COMMAND push button. The ADC-ON light begins glowing and the display screen starts to fill with a spectrum. After precisely 80 seconds, the ADC-ON light goes off. The experimenter then positions his display so that the peak he is interested in is centered on the screen. To do this, he merely turns the display-origin switch to the correct group of 128 channels, and expands the display until there are only 64 points on the screen. Each tenth point is intensified so he does not lose his place in the spectrum. Next, he turns the NORMAL-PUSHBUTTON switch to PUSHBUTTON. The appropriate pushbuttons allow fine adjustments to be made in display origin and allows a constant vertical bias to be applied to the display. At any time, he may restore his display by simply returning the NORMAL-PUSHBUTTON switch to NORMAL.

A Y-axis full-scale adjustment is also available to the experimenter. He may select a power of ten (full scale) via one rotary switch and a multiplier (1, 2, or 5) via another one.

Having positioned his display, he must now mark the channels he wants to use for his background and peak calculations. This is accomplished with four pushbuttons. The first one to be pressed is the REMOVE pushbutton. This removes all markers except one; positioned at the left end of the display screen. This marker may be moved, either rapidly or slowly, across the screen, (and beyond if desired). A CREATE pushbutton is the next to be used, and generates another marker at the left end of the screen. This last-generated marker is the one which can be moved now. This process is repeated for six markers. Unfortunately at this point the program currently in use (October 1966) deserts the remote console. A trip to the teletype is required to complete the desired operation. Q must be hit in order to order the markers, and a number must be hit to indicate the number of channels to be averaged around each marker.

Returning to the remote station, the rotary switch is turned to DISPLAY and the COMMAND button pressed. The computer computes a 3rd order approximation to the four outer marked data points on the display and returns displaying both the approximation and the data. The two inner markers mark the limits of the peak.

A carriage return on the typewriter now sends the computer off to do summing under both curves between the peak markers, and types out the two sums and the difference. Hitting other keys on the typewriter allows data storage on both DEC tape and IBM-compatable tape, complete with one line of comments and identification. Also, point plots and histograms may be made on the calcomp plotter.

At this point, the direction of future program planning should be obvious. It is desired to eliminate the experimenter's dependence on the typewriter and his trips back to the computer.

Future Programs

To fully realize the ability of the system to operate entirely from a remote console it is necessary simply

to have the stop-count signal initiate a user-routine. The user-routine will consist of selected utility subroutines (including those already developed) plus any special routines the experimenter may want to use. These will be called from DEC tape into a working area of memory. The busy light on the remote console will signal when the area is being used by someone else. A very simple programming language will be needed to allow a relatively inexperienced experimenter to tie together the basic utility routines to produce a working user-routine.

This is what is still required on a basic level to really put time-sharing into operation the PDP-7.

Core Usage

At the present time, about 2000 words of core (out of 8196 available) are used for the programs described above. However, all of the programs used are permanently in core while operating. By following through with our user-routine scheme above, it should be practical to substantially reduce the amount of core required for programs. Plans are also under way to increase available storage with either a disc or another core stack, allowing use of the machine to closely approach its maximum capability.

PDP-8 AS A DATA COLLECTOR IN A TIME-SHARED SYSTEM*

Robert Abbott
Institute of Medical Sciences
San Francisco, California

Abstract

The Institute of Medical Sciences at Presbyterian Medical Center is designing an automated retrieval system for utilization review and hospital admissions. The system will be used to schedule maximum utilization of the basic hospital facilities, (i.e. beds, operating theater, etc.)

The use of computers in this restricted area of hospital utilization and scheduling is not new. The intent of the funding agency, U. S. Public Health Service, is to provide a minimum service at reasonable costs to hospitals which do not, at present, have any automated features.

In the pilot study the methods to be employed will involve remote teletypes throughout a hospital (or hospitals); a PDP-8 serving as a data collector; and telephone lines to a large time-shared computer service. This paper will describe the role of the PDP-8 in this system.

*This paper was not received for publication

A PDP-5 PROGRAMME FOR USE IN NUCLEAR COUNTING

by
D.R. Thompson, E.E. Wuschke, A. Petkau
Whiteshell Nuclear Research Establishment
PINAWA, Manitoba, CANADA

Abstract

A programme designed to process data from three independent nuclear counting systems is discussed. Programme operations performed on data from two 512 channel gamma spectrometers are print, normalize and subtract background, erase, spectrum strip, integrate, and spectrum data tape read in. A third system, an auto-recycling alpha-beta ionization detector, has routines to initialize for the number of samples and runs, read and store data, calculate counts per minute, and print out data.

A discussion of the programming objectives, ease of use, brevity, and revision flexibility, is followed by a description of all programmed operations and their use.

Introduction

The Medical Biophysics Section at the Whiteshell Nuclear Research Establishment has three independent nuclear counting systems interfaced with a PDP-5. The three systems are shown as a block diagram in Figure 1 and consist of:

(1) A bioassay gamma spectrometer consisting of a lead castle, a 2" x 2" NaI (T1) crystal with a single photomultiplier tube, a 512 channel pulse height encoder, an oscilloscope display, a control panel, and a standard ASR 33 teletype.

(2) A Whole Body Counter consisting of a large steel room, a 11½" x 4" NaI(T1) crystal with seven photomultiplier tubes, a 512 channel pulse height encoder, an oscilloscope display, a control panel, and a standard ASR 33 teletype.

(3) A two channel, auto-recycling, alpha-beta ionization detector with pre-set count and time of count. The detector uses the same teletype as the bioassay gamma spectrometer.

A general discussion of the Analyzer Programme will be followed by a description of the programmed operations and their use.

General Discussion

To interface the three nuclear counting systems with the PDP-5 digital computer, some additional hardware was added to the system. Figure 2 is a block diagram of this hardware.

The two encoders and the oscilloscope display system gain access to the computer memory via the DATA BREAK MULTIPLEXER. The DATA BREAK MULTIPLEXER has been modified so that when an encoder break request occurs, the memory location specified by the encoder address is automatically incremented by one. Thus the memory locations simply count the number of pulses of each size.

To provide sufficient capacity two memory words are used for each encoder channel. The 9 bits from the encoders control bits 2 to 10 of the memory address. Bit 11 is normally set to 1 by the CONTROL LOGIC. If an overflow is detected when a memory increment occurs bit 11 is set to 0, and the memory is again incremented. In this way a double word is used for each encoder channel. Bits 0 and 1 of the address are permanently wired to select separate sections of memory for each encoder.

The display system also uses the DATA BREAK facility. The two memory locations corresponding to an encoder channel are read into an 18 bit register connected to a digital to analogue converter. The CONTROL LOGIC automatically selects encoder channels sequentially until all locations have been displayed. The two oscilloscopes time share the digital to analogue converter. Intensity modulation is used to display only the channels associated with each oscilloscope.

All other functions use the PROGRAMME INTERRUPT. With this feature the operator-to-computer communications via the teletypes

were minimized. This was an important consideration since a number of people, not necessarily familiar with the system, might wish to use it periodically.

Switches were installed which, with an augmented instruction list, are used to call up the various routines that perform the programmed operations. When a switch is pressed, a flag is set and a programme interrupt occurs. Computer control is then transferred to a search routine which checks all flags and calls up that routine whose flag has been set. Only one routine can be called up at a time but it does not interfere with the operation of the DATA BREAK facility. When not servicing a programme interrupt, the computer "idles" in a wait loop of NOP's.

The routines that can be called by switches are: (1) Binary Loader, (2) Initialize α-β Detector, (3) Erase, (4) Background Subtract, (5) Spectrum Strip, (6) Integrate, and (7) Print.

The flag for the SCALER READY routine is set by any one of three circuits associated with the alpha-beta detector, namely Scalar A, Scalar B, and the Timer. The routines are discussed more fully later.

An additional ASR-33 teletype with separate IOT commands was also installed. This permits printing of data from the two encoders on different teletypes. The two teletypes are on different floors in the building.

In writing the programme routines two somewhat conflicting factors were considered important. They were brevity and revision flexibility. Since over one half of the memory is used for data storage, it was essential that the programme routines be as short as possible. This can be accomplished by interlocking the various routines so that they are dependent of one another. At the same time, it was important that the programmed operation routines could be altered without forcing major changes in others. A delicate balance of inter-dependence was required and aimed for.

This has been achieved by indirectly addressing all routines through fixed page zero locations. All arithmetic operations and assembling is done on page zero so that the results are immediately available. If the status of the (AC) and (L), and various locations on page zero remain unchanged upon entry and exit, most routines can be altered without changing any other.

Part I - The Alpha-Beta Ionization Detector

Samples to be counted are placed in metal trays and stacked on the right of the detector. The samples move from the bottom of this stack into the ionization chamber where they are counted, and then are collected in a stack on the left. When all samples have been counted a "run" is completed. If the sample changer is started after a run is finished, the samples will be restacked on the right and counted again.

The count in the two channels and the time of count are each registered in 18 bit counters displayed on the control panel (see Figure 2). These three counters are external and are not part of the PDP-5 memory (see Figure 3). Three switches below these counters enable the operator to preset the count and time of count in powers of 2 from 7 to 17. Counting is terminated when any counter exceeds the value set on its switch.

Since each counter is 18 bits, two 12 bit words are needed to store them in memory. Therefore, 6 words of memory are required to store the data from each sample. With a maximum of 50 samples, 300_{10} or 454_8 words of memory are used.

Communication between the alpha-beta ionization detector system and the computer is accomplished through the programme interrupt facility.

When a count is terminated by the preset count or timer, a "Scalar Ready" flag is set and a programme interrupt occurs. Control is transferred to a search routine which, as mentioned before, checks all flags and calls up the appropriate routine. In this case, the routine called sequentially stores into memory the data from the two counting registers and the timer. Note that the data is loaded into memory under programme control and not through the data break facility.

The routine then checks to see if all samples have been counted. If not, it starts the counting of the next sample and exits. When all samples have been counted, the routine retrieves the data, calculates the counts per minute to one decimal place for each count register, and prints out the sample number, the counts per minute for each channel and the time of counting for each sample. If all runs have been completed, the routine is exited without starting

the sample changer. Otherwise, the count is re-initialized and the changer started.

The count is initialized by pressing the "Initial" switch on the control panel. This sets up the starting address of the data storage and types out the message, "No of Samples". As many as 50 samples may be counted. After the number of samples has been typed in, the message, "No of runs" is printed out. Up to 4095 runs may be made. In both cases, the number typed in will be an unsigned decimal integer which is delimited by a period (.). Errors can be corrected by typing a question mark (?), followed by the corrected number. See Figure 4 for a typical initialization and data print out.

Part II - The Gamma Spectrometers

The bioassay gamma spectrometer and the Whole Body Counter are essentially the same except for the different detectors. Therefore, from a programming point of view only slightly different initialization procedures were required.

The 512 channels of the encoder are broken into four groups of 128 channels. The "Channel Select" encoder switch enables the operator to select these groups individually, or the first and the second together, or the third and the fourth together, or all taken together. Any grouping that can be selected on the encoder switch is referred to as a "data field". Each channel is allocated two words in memory. Thus, 2048 words are required as data storage for the two gamma spectrometers. This is one-half of the total memory.

Before discussing the programmed operations for the gamma spectrometers, a description of the counting procedure will be given.

Counting

To begin counting, the "channel select" encoder switch is set to the desired data field, that area in memory is erased, and the "count" switch on the control panel is pressed. The data is loaded directly into memory through the data break facility. There is a "Data Break" switch on the control panel to enable or disable this facility. This switch must be "On" in order to count.

The first three channels of the selected field are used to count:

(1) The overflow, (OF). Any gamma ray detected with an energy outside of the calibrated limits is counted in this channel.

(2) The live time, (LT). This channel counts in 10ths of a second the time that the encoder is free to accept inputs - that is, the total time less the time used in the digitizing process.

(3) The dead time, (DT). This channel counts in 10ths of a second the time that the encoder is busy processing signals.

The live time plus the dead time is the total time. Since the gamma events occur at random intervals, the dead time divided by the total time indicates the fraction of events lost while the encoder was busy. A meter above the oscilloscope indicates the percent loss while the sample is being counted. The counting is stopped by pressing the "Stop" switch on the control panel.

The oscilloscope display shows the counts (y-axis) versus the channel number (x-axis). An x - y plotter will be added shortly to make permanent graphs of these spectra.

Some radionuclides emit gamma rays of characteristic energies. Peaks in the spectrum will occur in those channels that correspond to the energies of the gamma rays being detected. Thus, the operator, by determining the energy at any peak, can readily identify the nuclide. Figure 5 shows a typical energy spectrum. In this example, with a calibration of 7.5 KeV increment per channel, the peaks at channel numbers 156 and 177 indicate that two gamma rays of 1.17 MeV and 1.33 MeV, respectively, are being detected. This identifies the sample as Co^{60}.

Programmed Operations

(A) Erase

This routine will clear the area in memory occupied by the data field selected on the encoder switch. This routine is short and sweet; less than one second short, 17_8 lines sweet.

(B) Background Subtract

There is always background radiation present and it must be subtracted from the gross sample spectrum in order to get the net spectrum, or the spectrum due to the sample alone. Background counts are taken

regularly and vary from one hour to two days.

This routine will normalize and subtract a background (spectrum) from a (sample) spectrum. The spectrum must be placed in the lower half of the selected field and the background in the upper half.

The normalizing factor, (NF), is calculated to three decimal places by dividing the spectrum live time, (LT), by the background (LT). This is not a straight forward operation since the divide routine is in fixed point. However, a degree of floating point can be achieved by using the multiply routine since multiplying by a fraction code will give an answer in fraction code. It is up to the programmer to keep track of the decimal point, to decide which part of the answer is in integer code and which part is in fraction code.

Consider the following example:

$$\begin{array}{r} 0013_8 \\ \underline{\times \quad 4000_8} \\ 00054000_8 \end{array}$$

This can be interpreted in many ways. If one assumes

0013_8 is $1.3_8 = 1.375_{10}$

and 4000_8 is $.4_8 = .5_{10}$

then 00054000_8 is $.54_8 = .6875_{10}$

Or, if one assumes

0013_8 is $13._8 = 11._{10}$

4000_8 is $.4_8 = .5_{10}$

00054000 is $5.4_8 = 5.5_{10}$

Thus the (NF) is calculated in the following manner. First the spectrum (LT) is multiplied by $10{,}000_{10}$ and then divided by the background (LT). Four assumed decimal places are in the answer but are in integer code. The fraction code is obtained by multiplying by $.0001_{10}$ to give a (NF) accurate to three decimal places.

The background is then multiplied by the (NF) and subtracted from the spectrum. The result is left in the lower half of the selected field. The first three channels, (OF), (LT), and (DT), are not changed.

By storing a background spectrum in the upper half and counting samples in the lower half of a selected field, the operator can easily get the net spectrum as soon as the count is finished. For a selected field of 512 channels this routine takes less than 6 seconds.

(C) Print

This routine will print out in signed decimal the data in the field selected on the encoder switch (see Figure 6 for example format). Twelve inches of tape with leader/trailer, (200), code are typed out at the beginning and end of the print out.

The code "G" or "W" is printed at the start to indicate the bioassay gamma spectrometer or the whole body counter, respectively. The first three channels, (OF), (LT), and (DT), are printed out individually. The spectrum data follows in six columns. The first column gives the number of the first channel on each line. The next five columns are the channel counts.

The routine is exited automatically once all channels in the selected field have been printed. However, the routine also checks the "Print" flag before printing each line of data. If the flag is set, the line will be printed. If the flag is not set, the routine will be exited. Hence, the print out can be stopped manually after the first three channels or any line by clearing the "Print" flag with the "Stop" switch on the control panel. This routine takes less than 6 minutes to print out 256 channels.

(D) Spectrum Strip

This routine is used when dealing with composite spectra. In this case it is desirable to subtract a reference spectrum of one element from the composite spectra, thus eliminating one component from the composite spectra.

Since different counting times and sample strengths are involved, it is necessary to multiply the reference spectrum by a normalizing factor (NF) which may be greater or less than one. The results of the subtraction are observed on the display, and the process is repeated with different (NF) until the component of interest is completely removed. Since it is possible to subtract too large an amount the routine also permits addition of the reference spectrum. At the completion of these operations the net (NF) is printed on the teletype.

In a composite spectrum with many components more than one reference spectrum is used. A read in routine is provided to read the reference spectra from paper tape into memory.

The routine is entered by pressing the "Spectrum Strip" switch. It then waits for one of four codes to be entered on the teletype. Typing an (R) will cause the routine to read a spectrum data tape into the upper half of the selected data field. The routine then exits.

The stripping operation is performed by pressing the switch and entering (+) or (-) and the (NF). The normalizing factor consists of an integer less than 4096 and a fraction with not more than three digits. The integer is delimited by a period; the fraction by a space. Errors in either part can be corrected by entering a question mark (?) and retyping the correct value.

The routine then performs the arithmetic operations and waits for another code. The operator may enter further (+) or (-) codes and (NF)'s until the results are satisfactory. A (P) code is then entered and the routine calculates and prints the net (NF) and exits, (see Figure 7 for format).

(E) Integrate

This routine will integrate between limits which are typed in by the operator. The limits will be unsigned, decimal integers not greater than 4095, delimited by a period (.). Errors can be corrected by typing a question mark (?), followed by the correction (see Figure 8).

Since the channels are discrete the routine only has to add up the counts in the channels specified. Both limits are inclusive. Negative counts are ignored. If the total count exceeds the double word length capacity of the conversion routine, an error message, "Overflow", is printed out. This error can be overcome by splitting up the integration into smaller parts.

Conclusion

The Analyzer Programme occupies 12_8 pages, leaving $3\frac{1}{2}$ pages available for other uses. The Analyzer is self-contained, no additional routines are needed. Because of the revision flexibility feature, programmes can be easily altered or added to fit future needs.

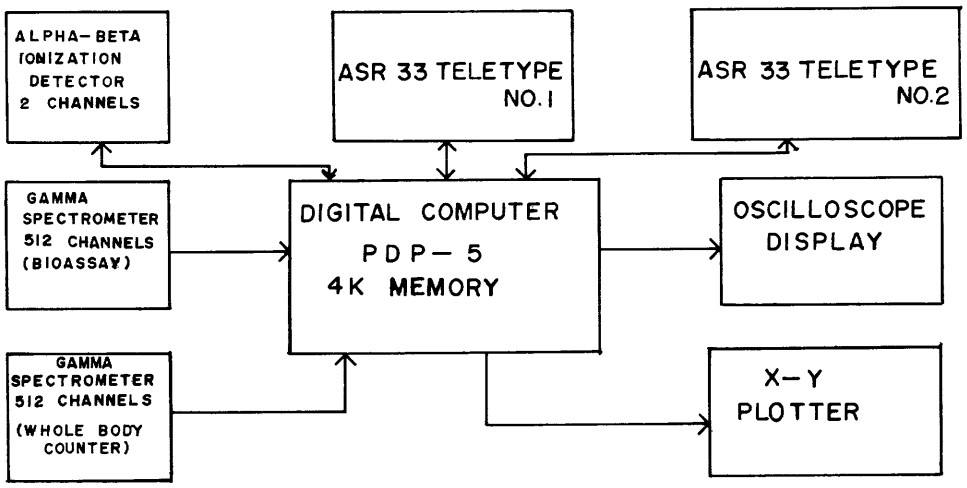


Figure 1 Block Diagram of Integrated Counting System Using a Small On-line Digital Computer

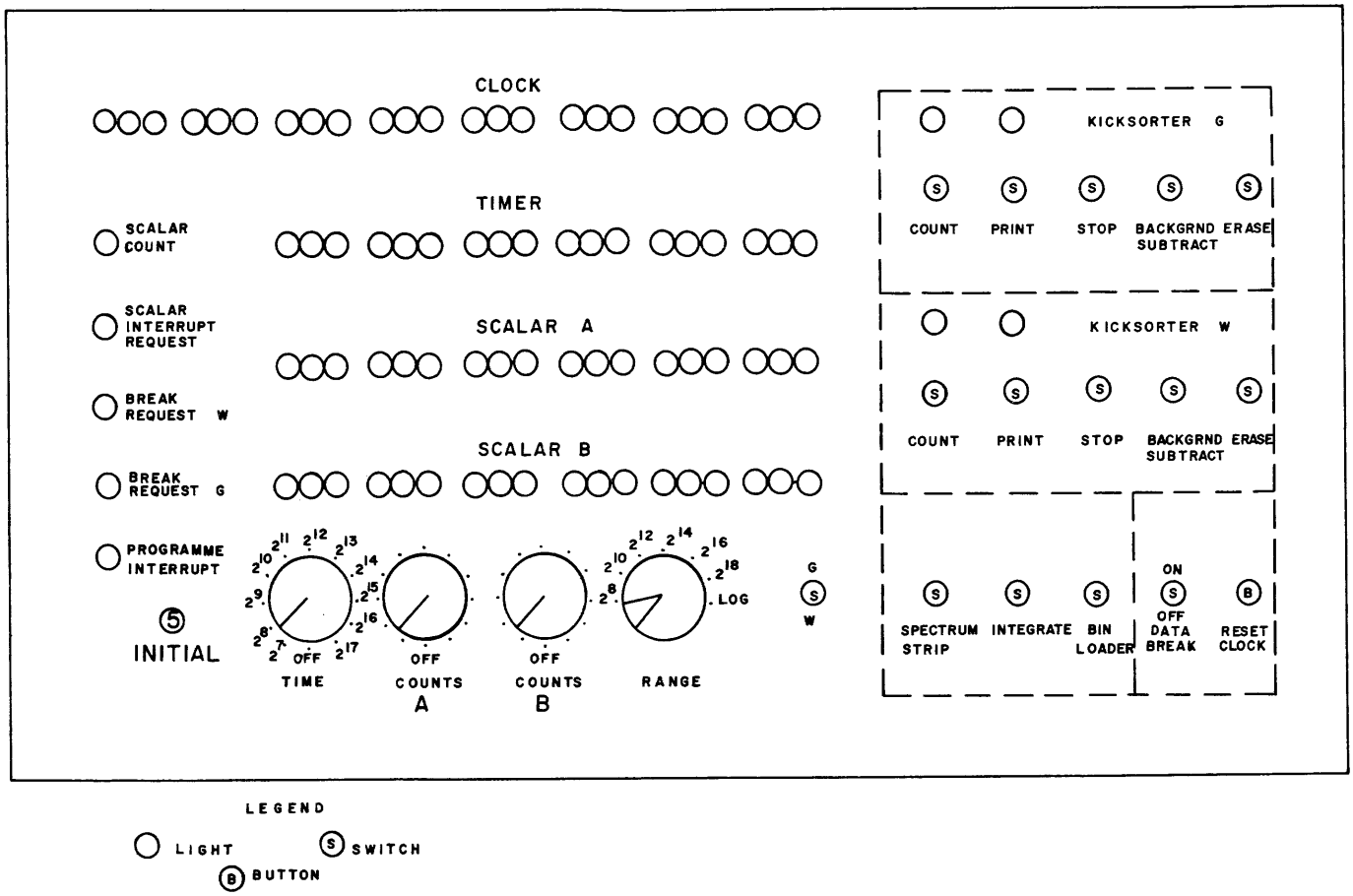


Figure 2 Control Panel for Additional Circuits Used to Interface an Alpha-beta Detector and Two Encoders with a PDP-5

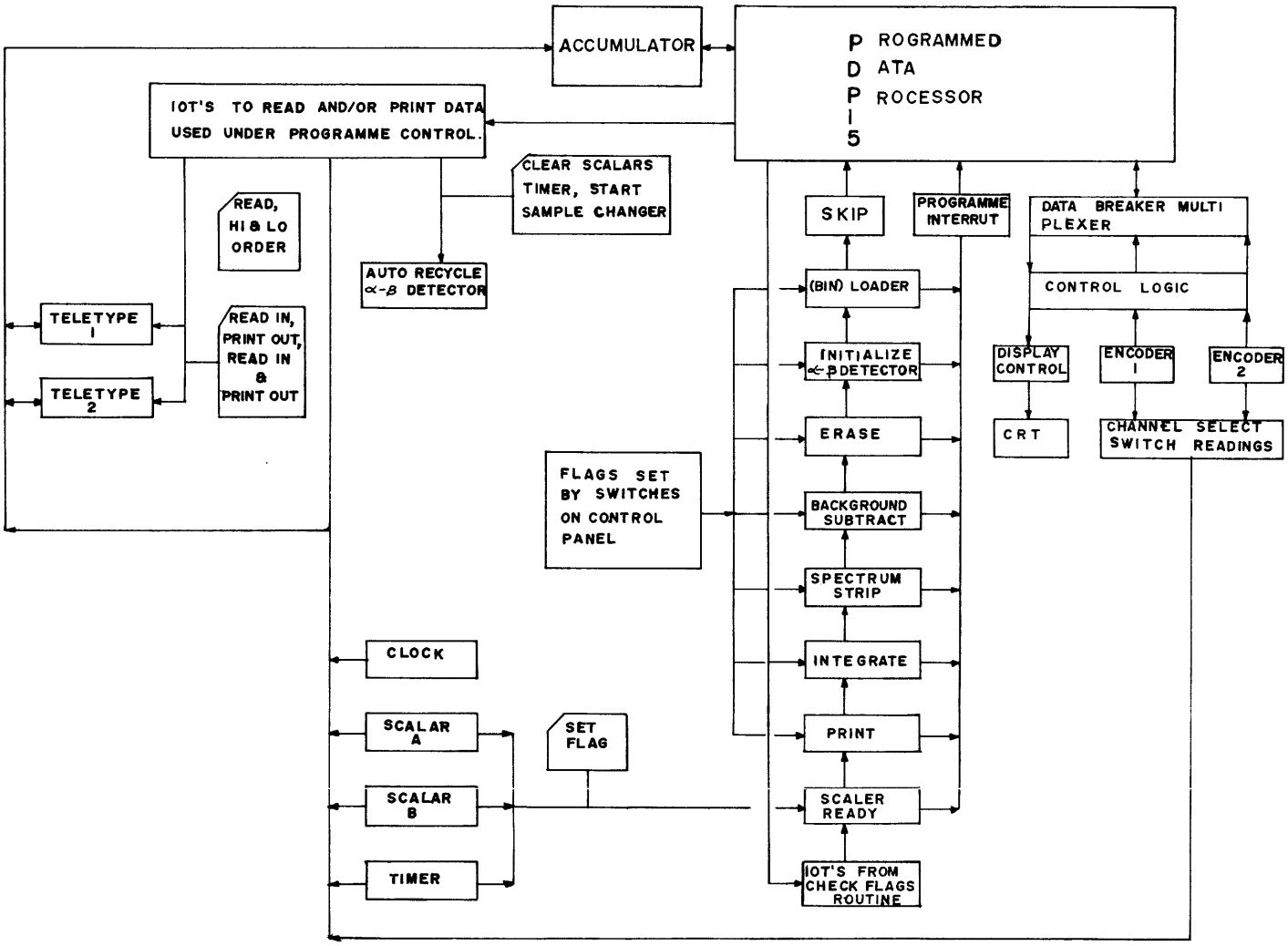


Figure 3 Block Diagram of Additional Circuits Used to Interface an Alpha-beta Detector and Two Encoders with a PDP-5

NO OF SAMPLES 4?5.
NO OF RUNS 3.

} INITIALIZATION

1	149	4	257
2	22446	22279	219
3	128	2	257
4	34405	34191	143
5	25643	25446	192
1	172	21	257
2	21660	21496	227
3	168	11	257
4	35400	35145	139
5	25715	25550	192
1	161	18	257
2	21985	21841	224
3	165	7	257
4	34745	34478	142
5	25307	25126	195

} DATA PRINT

Figure 4 Format of Data Print Out for Alpha-beta Detector

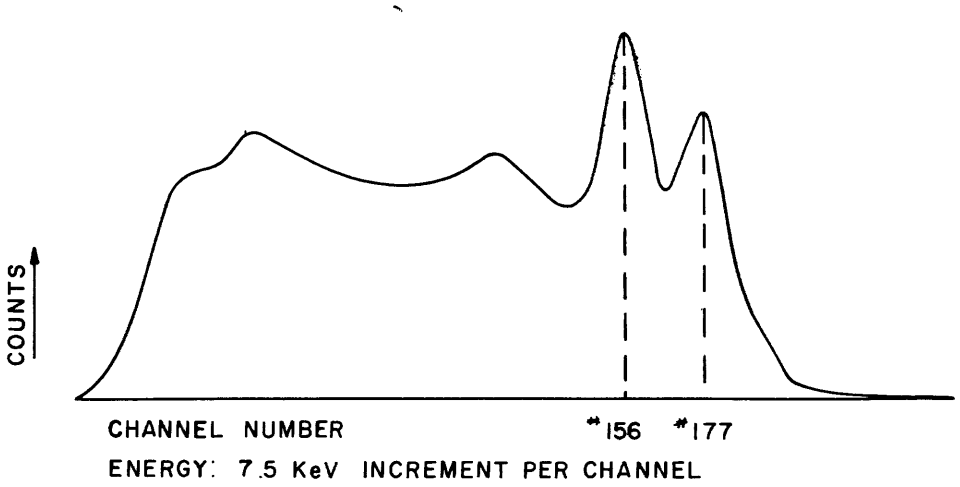


Figure 5 Co^{60} Spectrum

```
G OF      52
  LT      56
  DT      63

   1        0        4       31      147      244
   6      403      531      661      870      975
  11     1158     1122     1152     1233     1148
  16     1147     1216     1152     1180     1186
  21     1186     1197     1310     1350     1409
  26     1392     1456     1474     1446     1393
  31     1374     1387     1348     1389     1331
  36     1291     1227     1288     1230     1232
  41     1202     1128     1128      981      880
  46      816      673      661      586      551
  51      491      489      445      455      446
  56      505      531      638      757     1007
  61     1383     1960     2660     3396     4362
  66     4919     5199     5047     4709     4025
  71     3201     2365     1727     1225      843
  76      576      413      385      350      316
  81      318      282      254      298      291
  86      303      279      259      300      264
  91      268      275      270      252      271
  96      284      278      299      237      271
 101      231      253      252      277      258
 106      269      276      244      279      273
 111      290      272      270      253      248
 116      250      279      272      278      273
 121      263      247      263      290      301
```

Figure 6 Format of Data Print Out for Gamma Spectrometer

```
+1.134
-1.358
P-        0.224
```

Figure 7 Format Used to Obtain the Net Normalizing Factor (NF)

```
HIGH LIMIT 58/?58.
LOW LIMIT 1.
INTEGRATED COUNTS      54896


HIGH LIMIT 0.
LOW LIMIT 1.
INTEGRATED COUNTS     OVERFLOW
```

Figure 8 Format for Integrate Routine

FLYING-SPOT SCANNER*

Ray Kenyon
Lawrence Radiation Laboratory
Berkeley, California

Abstract

A description of the Flying-Spot Scanner used with the PDP-6, and associated hardware, at the MB Virus Laboratory, University of California.

*This paper was not received for publication

THE PDP-6'S ROLE IN ANALYZING PHOTOGRAPHS OF BACTERIAL COLONIES*

Fraser Bonnell
University of California, MB Virus Laboratory
Berkeley, California

Abstract

A PDP-6 and a Flying-Spot Scanner will be used to analyze photographs of bacterial colonies. This analysis involves finding, sizing, and counting colonies; as well as using "optical density profiles" and colony morphology to identify various organisms.

This project is being carried out in the Molecular Biology Department at the University of California.

*This paper was not received for publication

APPLICATION OF THE PDP-5 TO DATA HANDLING FOR MESON-PRODUCED X RAYS*

Robert W. Lafore

Lawrence Radiation Laboratory
University of California
Berkeley, California

Abstract

This paper describes a series of programs written to process, on a PDP-5 computer, data from an experiment measuring pi-mesic-produced x-rays at the Lawrence Radiation Laboratory's 184-Inch Cyclotron. The programs display data from an A/D converter on an oscilloscope, record it on magnetic tape, and provide feedback to the gain and bias settings of the data amplifier for stabilization of the system transfer function. One program accepts 2048 incoming channels, the other accepts 4096. A third program permits analysis of data previously stored on tape, the area and centroid of a given peak being calculated using a light pen to set the peak boundaries and background line.

I. 2048 Channel Program

Memory Locations

Figure 1 shows the physical arrangement of the system. The 2048-channel program uses only half the full capacity of the Nuclear Data analogue-to-digital converter. Half of the 4096-word computer memory is reserved for the program, leaving the second half free to pulse-height sort the incoming data, each channel corresponding to one word. The maximum count of 4096 counts per channel available with the 12-bit word was found to provide adequate vertical resolution.

Oscilloscope Display

In operation, the fairly low count rate of 500 per second requires a time of at the least several minutes to produce a satisfactory spectrum. During the growth of the spectrum the horizontal scale and origin as well as the vertical scale may be varied, using commands inputted to the teletype. Eight horizontal scales and three vertical scales are provided so that the overall spectrum may first be viewed, then a particular peak selected for closer examination. The display continues while data is being stored. In order to minimize the deadtime between incoming data, the display is executed one point at a time. After each point, the program returns to check if a new data word is ready to be processed; only if not does it return to display another point.

Tape Transfer

Once the spectrum growth has reached the size required for statistically accurate data, it may be transferred to magnetic tape, using a teletype option. Alternatively, the program may be asked to write out the memory on tape every time the highest peak grows off scale. An identification section is included in each data record placed on tape; this can also be altered using the teletype.

Stabilization

During data processing and display, the program is also stabilizing the transfer function of the system (see Fig. 2). This is accomplished by having in effect two separate input spectra. The first spectra consists of the actual mesic-produced x rays that the experimenter wishes to measure; the second is produced by two radioactive sources. The peaks produced by the sources have an absolutely constant energy, and may thus be used as a reference. If an incoming datum is flagged by the scintillator circuitry associated with the sources (as opposed to a mesic-produced x-ray), the program compares the data with a value previously inserted in the program by the operator. If the incoming

*This work was done under the auspices of the U. S. Atomic Energy Commission.

value is too high, the computer outputs a pulse to a 512-count scaler (see Fig. 3). The reading of this scaler is in turn transformed into an analogue signal which controls the gain (or bias) setting of the data amplifier. The calibration values of the peaks and window width (a range about each peak within which data must fall to be used for calibration) are typed in by the operator.

II. 4096-Channel Program

Storage

Since the memory of the PDP-5 cannot contain a separate word for each of 4096 incoming channels, the simple pulse-height sorting of the previous example cannot be used. Instead, incoming data is simply stored sequentially, one datum per word, in a 1536-word buffer. When full, this buffer is automatically dumped on tape.

The obvious disadvantage of this system is the much increased amount of tape necessary and the longer time to process it. The CDC-6600 is used to transform tape records of this type into ones similar to those obtained with the 2048-channel program above.

This disadvantage is partly compensated for by the fact that it is possible in effect to store two complete spectra at the same time; in this case the incoming spectrum composed of the stabilization peaks can be separated from the spectrum of the mesic-produced x rays. This is done by marking each datum from the stabilization spectrum with a preceding 7777. Thus, whenever the analysis program encounters a 7777 datum, it ignores it and assumes the following datum is part of the stabilization spectrum.

Oscilloscope Display

Another disadvantage is the impossibility of making an oscilloscope display out of the sequentially stored buffer. This problem was solved by using a separate 512-word buffer for the oscilloscope, and pulse-height sorting incoming data into it at the same time the data were being stored in the sequential buffer. This presented another difficulty, however, since 1-channel resolution of the spectrum on the oscilloscope was necessary, which was clearly impossible if the entire spectrum were viewed at once(since only 512 of the total 4096 channels would be displayed). Three separate storage modes for the oscilloscope memory were therefore used, the first stored every eighth channel and showed the entire spectrum, the second stored each channel, but only covered one-eighth of the spectrum, while the third showed each channel of one sixty-fourth of the spectrum, or sixty-four channels total (see Fig. 4).

In operation, it is necessary therefore to erase the oscilloscope buffer when changing from one scale to another, and let the new picture "grow" again. The data rate was, however, sufficiently fast that this was not too inconvenient. The teletype could also be used to decrease the vertical scale by a factor of 2, so that during a long run the oscilloscope display would not grow off-scale.

Programming Comparisons

The penalty paid for having separate memories for the tape buffer and oscilloscope buffer is quite severe in terms of program space and complexity. Two routines are required for storage, clearing, updating, etc., where only one was required in the 2048-channel version. Many nonessential but convenient routines for which there was space in the 2048-channel program can not be accommodated in the 4096-channel program.

III. Analysis Program

Originally it was intended that the tapes generated would be analyzed on the CDC-6600. However, it was found that the PDP-5 itself, by permitting on-line examination of the spectra from the tapes, could give satisfactory results. Previously written tape records can be read into memory (in the 2048-channel format), and any part of the resulting spectrum examined in detail.

The centroid and area of specific spectrum peaks are the desired results, but the subtraction of a fuzzy background from the peak must be done before the calculations are made. To simplify the definition of the background, a light pen is used to draw in vertical lines bounding the peak, and a line connecting them for the background (see Fig. 5). Arithmetic routines then perform the area and centroid calculations, printing out results in decimal notation.

IV. Summary

The programs have provided a great measure of flexibility for the experiment and facilitated making the changes necessary to meet the experiment's evolving requirements. The use of the PDP-5 for 4096-channel analysis shows that it is not necessary (although certainly it is preferable) to be able to reserve one memory word for each incoming channel.

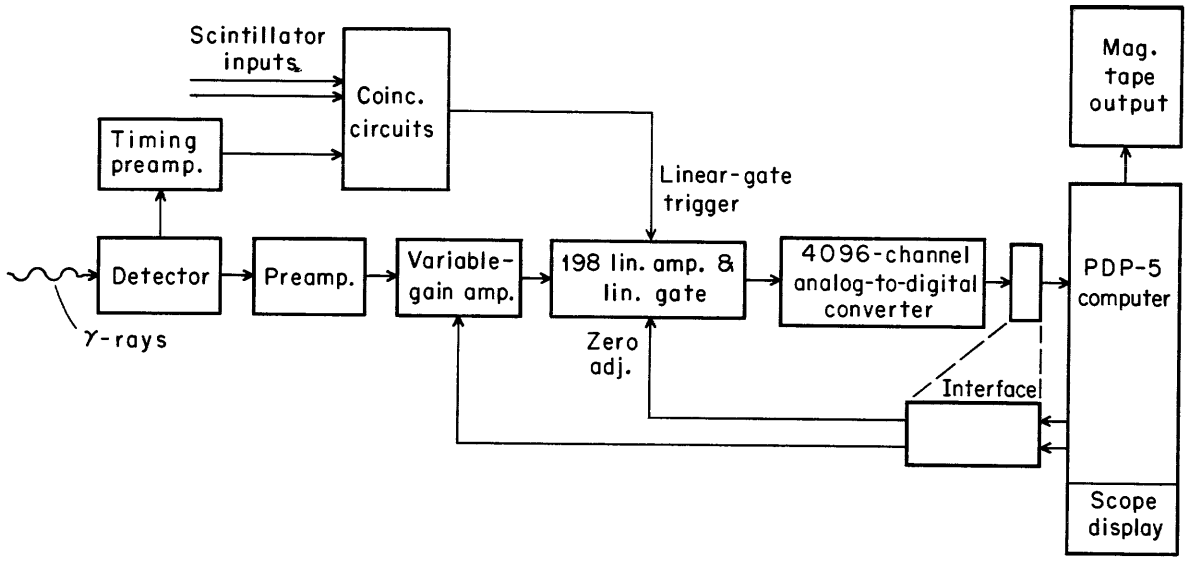
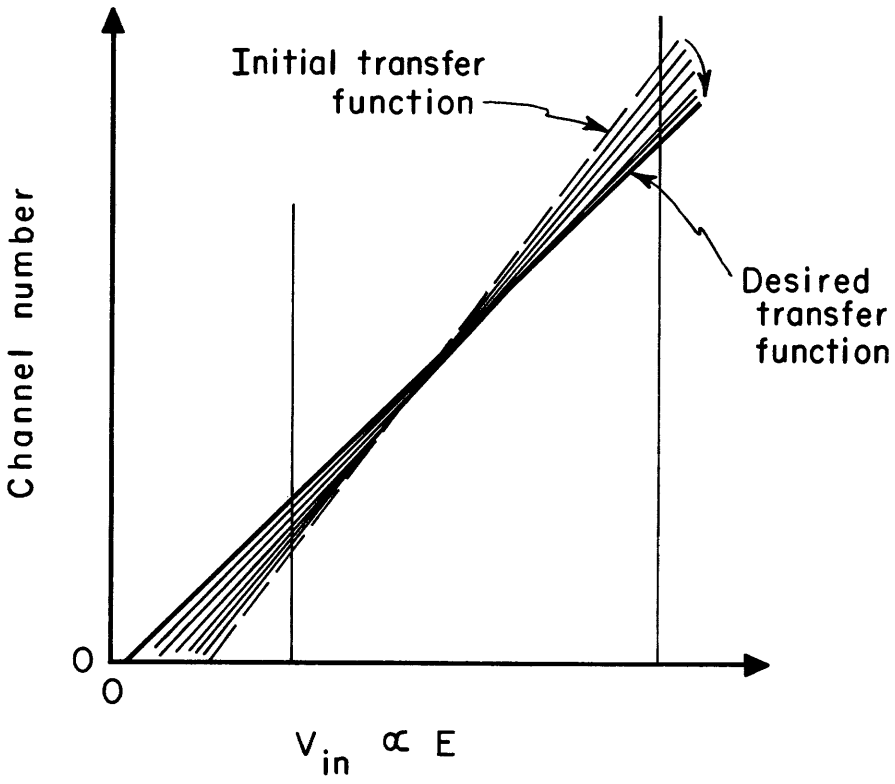


Fig. 1. System block diagram.

MUB-13721



MUB-13722

Fig. 2. Stabilization process.

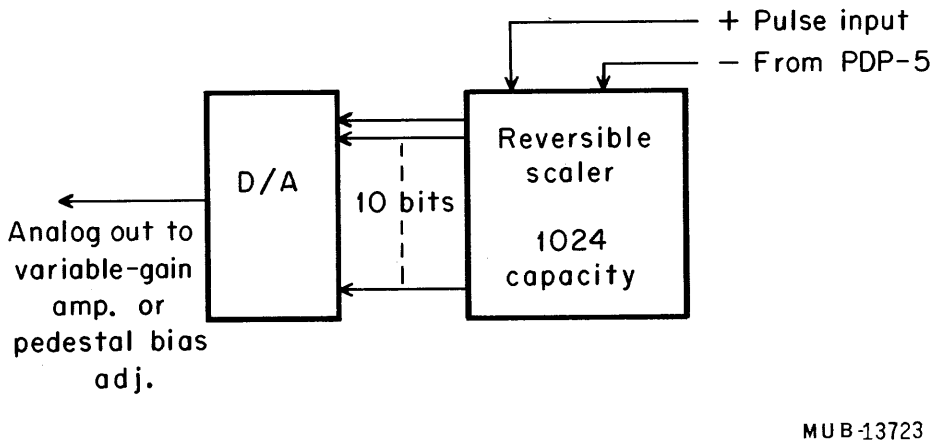


Fig. 3. Feedback block diagram.

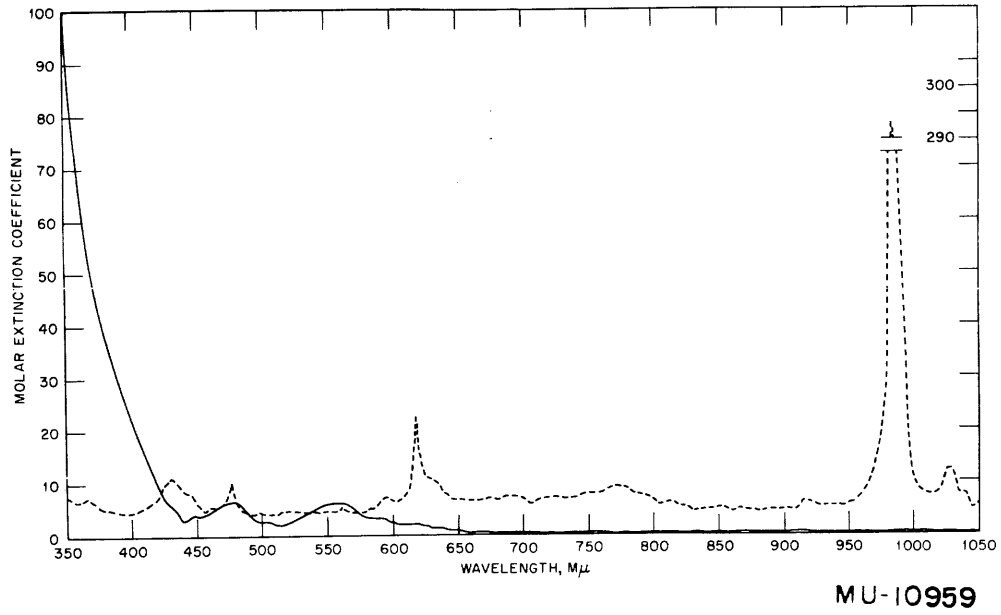


Fig. 4. Oscilloscope scales, 4096-channel program.

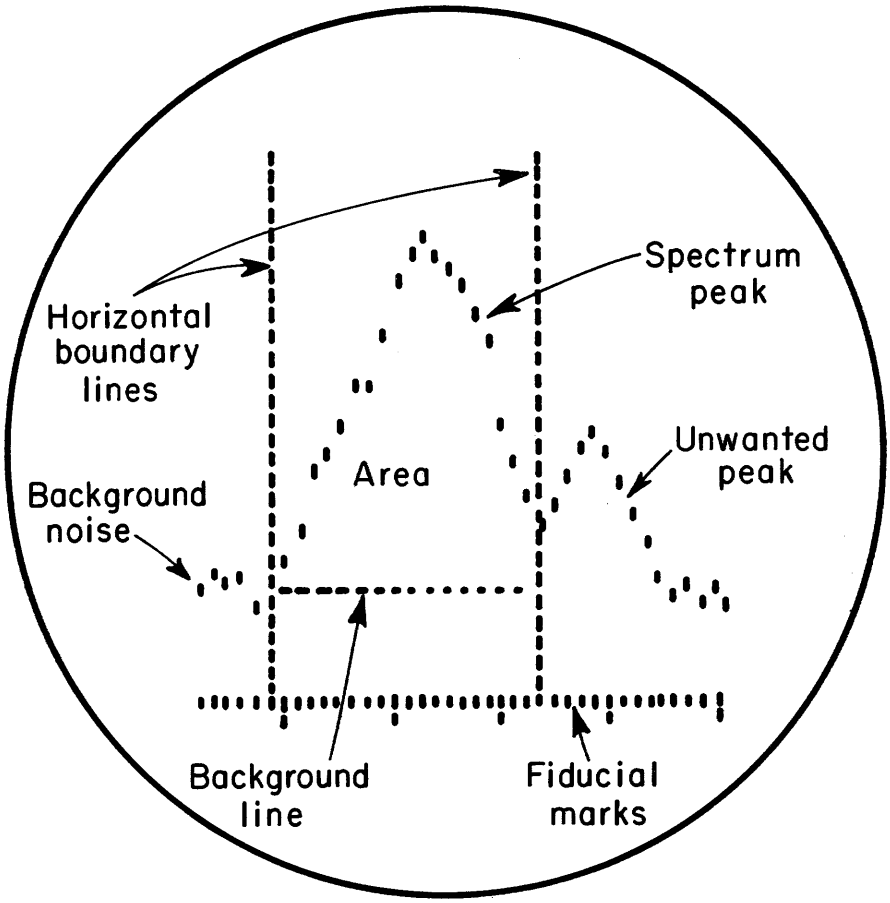


Fig. 5. Oscilloscope display of spectrum peak. analysis program.

APPENDIX 1

DECUS FALL SYMPOSIUM PROGRAM

Building 50 Auditorium
Lawrence Radiation Laboratory
University of California
Berkeley, California

November 4 and 5, 1966

Friday, November 4, 1966

8:30-9:55 Registration

10:00 Opening
Professor Donald A. Molony, Chairman

10:15 Welcome Address
Winston R. Hindle,
Digital Equipment Corporation

10:30 ESI-Conversational Mode Computing for the PDP-8/S-David Waks,
Applied Data Research, Inc.

11:00 The BBN On-Line Programming System
Nancy E. Lambert,
Bolt Beranek and Newman, Inc.

11:30 JOSS: Introduction to the System Implementation-G. Edward Bryan,
The Rand Corporation

12:00 Lunch

1:30 PDP-10 System Philosophy New Product Announcement-David Plumer,
Digital Equipment Corporation

2:15 PDP-6 System at LRL, Livermore
John G. Fletcher, Lawrence Radiation Laboratory

2:45 A Common Memory System for a Time-Shared PDP-6-Earl Brazeal and Stuart Sharpe, United Aircraft Research Laboratories

3:15 Coffee

3:30 An Operating System for the PDP-8
Jacob L. Meiry, Massachusetts Institute of Technology*

4:00 PDP-6 Discussion Session

General Discussions (Meeting Room)

6:00-7:30 Cocktails
Hotel Durant - Regents Room

7:30 Banquet
Hotel Durant - Main Dining Room
Guest Speaker,
Dr. Willard H. Wattenburg,
Berkeley Scientific Laboratory

* This paper was not presented.

Saturday, November 5, 1966

9:00-9:30	Registration
9:30	Opening of Second Day's Session
9:45	On-Line Real-Time Time-Sharing Operation of a PDP-7-Lloyd Robinson and John Meng, Lawrence Radiation Laboratory
10:15	Control of High Energy Particle Accelerators-Robert Belske, Lawrence Radiation Laboratory
10:45	Coffee
11:00	PDP-8 as a Data Collector in a Time-Shared System-Robert Abbott, Institute of Medical Sciences
11:30	A PDP-5 Program for use in Nuclear Counting-D. R. Thompson, E. E. Wuschke, A. Petkau, Whiteshell Nuclear Research Establishment
12:00	Lunch
1:30	Business Meeting
2:00	Flying-Spot Scanner-Ray Kenyon, Lawrence Radiation Laboratory
2:15	The PDP-6's Role in Analyzing Photographs of Bacterial Colonies-Fraser Bonnell, University of California, MB Virus Laboratory
2:45	Application of the PDP-5 to Data Handling for Meson Produced X-Rays-Robert Lafore, University of California Lawrence Radiation Laboratory
3:15	Coffee
3:30	Workshops

APPENDIX 2

AUTHOR AND SPEAKER INDEX

APPENDIX 3

ATTENDANCE

Abbott, Robert P.
The Institute of Medical Sciences
San Francisco, California

Anderson, Albert
Stanford University
Physics Department
Stanford, California

Anderson, Roger E.
Lawrence Radiation Laboratory
Livermore, California

Bahnsen, Robert M.
University of Oregon
Eugene, Oregon

Bailey, Carolyn
Lawrence Radiation Laboratory
Livermore, California

Barker, D.
Digital Equipment Corporation
Palo Alto, California

Beal, Al
Digital Equipment Corporation
Palo Alto, California

Belshe, Robert A.
Lawrence Radiation Laboratory
Berkeley, California

Benham, Monte
Battelle Northwest
Richland, Washington

Bevington, Philip R.
Stanford University
Physics Department
Stanford, California

Blackmore, Ewart W.
University of British Columbia
Physics Department
Vancouver 8, B. C.

Borgerson, Barry R.
Lawrence Radiation Laboratory
Berkeley, California

Brandon, Gordon
Lawrence Radiation Laboratory
Berkeley, California

Brazeal, Earl
United Aircraft Research Labs.
East Hartford, Connecticut

Brown, J. Royce
M.I.T. Lincoln Laboratory
Lexington, Massachusetts

Bryan, G. E.
The Rand Corporation
Computer Sciences Department
Santa Monica, California

Cichon, Clarence
Digital Equipment Corporation
Palo Alto, California

Collom, Penny
Lawrence Radiation Laboratory
Berkeley, California

Cossette, Angela J.
Digital Equipment Corporation
Maynard, Massachusetts

Crossman, E. R. F. D.
Human Factors in Technology
Dept. of Electrical Engineering
University of California
Berkeley, California

Cunningham, Warren
Lawrence Radiation Laboratory
Livermore, California

Curtis, Kent
Lawrence Radiation Laboratory
Livermore, California

Delorey, James
San Francisco Bay Naval Shipyards
Vallejo, California

Donahoe, C. G
San Francisco Bay Naval Shipyards
Vallejo, California

Duncan, Richard E.
Lawrence Radiation Laboratory
Livermore, California

Ekonomi, Engin
Lawrence Radiation Laboratory
Berkeley, California

Elliott, G. R.
Sandia Corporation
Sandia Base
New Mexico

Fanshier, Donald R.
Lawrence Radiation Laboratory
Berkeley, California

Fensch, Frederick C.
University of Michigan
Ann Arbor, Michigan

Fisk, Irving
Investment Statistics
San Francisco, California

Fleck, P.
M.I.T. Lincoln Laboratory
Lexington, Massachusetts

Fletcher, John G.
Lawrence Radiation Laboratory
Livermore, California

Fox, Tom
Sandia Corporation
Albuquerque, New Mexico

Frazer, Donald L.
Lawrence Radiation Laboratory
Livermore, California

Frazer, Jack W.
Lawrence Radiation Laboratory
Livermore, California

Frensch, Theo
Hewlett Packard Company
Palo Alto, California

Friesen, Dave
M.I.T. Laboratory of Nuclear Science
Cambridge, Massachusetts

Goldstein, Robert
Lawrence Radiation Laboratory
Berkeley, California

Gray, Patrick H.
Lawrence Radiation Laboratory
Livermore, California

Gruen, Richard
Digital Equipment Corporation
Palo Alto, California

Hantman, Leonard M.
C. W. Adams Associates, Inc.
Cambridge, Massachusetts

Harvill, James R.
Lawrence Radiation Laboratory
Berkeley, California

Henry, Robert
University of California
MB-Virus Laboratory
Berkeley, California

Hindle, Winston R., Jr.
Digital Equipment Corporation
Maynard, Massachusetts

Horovitz, Mark
Lawrence Radiation Laboratory
Berkeley, California

Jewell, W. S.
University of California
Berkeley, California

Jones, Richard C.
Applied Data Research Inc.
Princeton, New Jersey

Kent, Douglas A.
Lawrence Radiation Laboratory
Livermore, California

Kerns, Cordon
Lawrence Radiation Laboratory
Berkeley, California

Kirsten, Frederick A.
Lawrence Radiation Laboratory
Berkeley, California

Klezmer, Stanley
Lawrence Radiation Laboratory
Berkeley, California

Lafore, Robert
Lawrence Radiation Laboratory
Berkeley, California

Lambert, Nancy
Bolt Beranek and Newman Inc.
Cambridge, Massachusetts

Landrum, Jerry H.
Lawrence Radiation Laboratory
Livermore, California

Larsen, Ken
Digital Equipment Corporation
Palo Alto, California

Lindsay, Jane M.
University of Michigan
Mental Health Research Institute
Ann Arbor, Michigan

Lund, Paul
Lawrence Radiation Laboratory
Livermore, California

MacGinitie, Gordon
Granger Associates
Palo Alto, California

Meng, John D.
Lawrence Radiation Laboratory
Berkeley, California

Miedaner, Terrell L.
University of Wisconsin
Space Astronomy Laboratory
Madison, Wisconsin

Moloney, Donald A.
Rutgers University
New Brunswick, New Jersey

Marston, Glen P.
University of British Columbia
Dept. of Electrical Engineering
Vancouver 8, B.D., Canada

McClure, Eldon Ray
Lawrence Radiation Laboratory
Livermore, California

McClure, John W.
Lawrence Radiation Laboratory
Livermore, California

McInturff, Robert
Digital Equipment Corporation
Palo Alto, California

McQuillin, Richard J.
Infronics Inc.
Cambridge, Massachusetts

Metcalfe, Robert M.
M.I.T. Laboratory of Nuclear Science
Cambridge, Massachusetts

Morgon, Gary B.
Lawrence Radiation Laboratory
Livermore, California

Mortensen, Keith W.
Lawrence Radiation Laboratory
Livermore, California

Mougel, Jean F.
French Atomic Energy Commission
Paris, France

Myers, Myron
Lawrence Radiation Laboratory
Berkeley, California

Plumer, David M.
Digital Equipment Corporation
Maynard, Massachusetts

Ragsdale, Ronald
University of Toronto
Toronto 5, Ontario, Canada

Robinson, Lloyd
Lawrence Radiation Laboratory
Berkeley, California

Rosenberger, T.
University of California
Human Factors in Technology
Dept. of Industrial Energy
Berkeley, California

Ross, W. A.
Hewlett Packard Co.
Palo Alto, California

Rudden, Robert J.
Lawrence Radiation Laboratory
Berkeley, California

Russell, Stephen
Stanford University
Stanford, California

Sarbin, T. R.
University of California
Human Factors Laboratory
Berkeley, California

Schaeffer, Tony
Lawrence Radiation Laboratory
Berkeley, California

de Saussure, Raymond
Lawrence Radiation Laboratory
Livermore, California

Shelor, Ted R.
GD/Convair
San Diego, California

Sifnas, Martha
Digital Equipment Corporation
Maynard, Massachusetts

Stedman, Jon D.
Lawrence Radiation Laboratory
Berkeley, California

Stevenson, Charles W.
Information Systems Design, Inc.
Oakland, California

Storch, David
Lawrence Radiation Laboratory
Livermore, California

Strollo, Theodore R.
Bolt Beranek & Newman Inc.
Cambridge, Massachusetts

Strople, Joseph E.
Lawrence Radiation Laboratory
Berkeley, California

Suffert, Martin
Stanford University
Stanford, California

Sullivan, Mervyn
Northern Electric Company
Ottawa, Ontario, Canada

Sweeney, John
University of California
Donner Laboratory
Berkeley, California

Thomas, R. A.
Lawrence Radiation Laboratory
Livermore, California

Thompson, David R.
University of Manitoba
Winnipeg, 15, Manitoba, Canada

Thompson, Lee
University of Wisconsin
Space Astronomy Laboratory
Madison, Wisconsin

Upham, Frank T.
Lawrence Radiation Laboratory
Berkeley, California

Vangerov, Martin
Associated Aerosciences Laboratories
Pasadena, California

Waks, David J.
Applied Data Research, Inc.
Princeton, New Jersey

Walter, Edward
Information Control Systems
Ann Arbor, Michigan

Wells, Donald O.
University of Oregon
Eugene, Oregon

Williger, Phil
Digital Equipment Corporation
Los Angeles, California

Windsor, Alfred A.
Lawrence Radiation Laboratory
Berkeley, California

Wing, P.
University of California
Human Factors in Technology
Dept. of Industrial Engineering
Berkeley, California

Wolverton, Michael J.
Lawrence Radiation Laboratory
Berkeley, California

Wylie, William R.
University of Oregon
Eugene, Oregon

Zeligman, Mark M.
Lawrence Radiation Laboratory
Livermore, California

Zurlinden, Don
Lawrence Radiation Laboratory
Berkeley, California

PRINTED IN U.S.A.

15-1/67