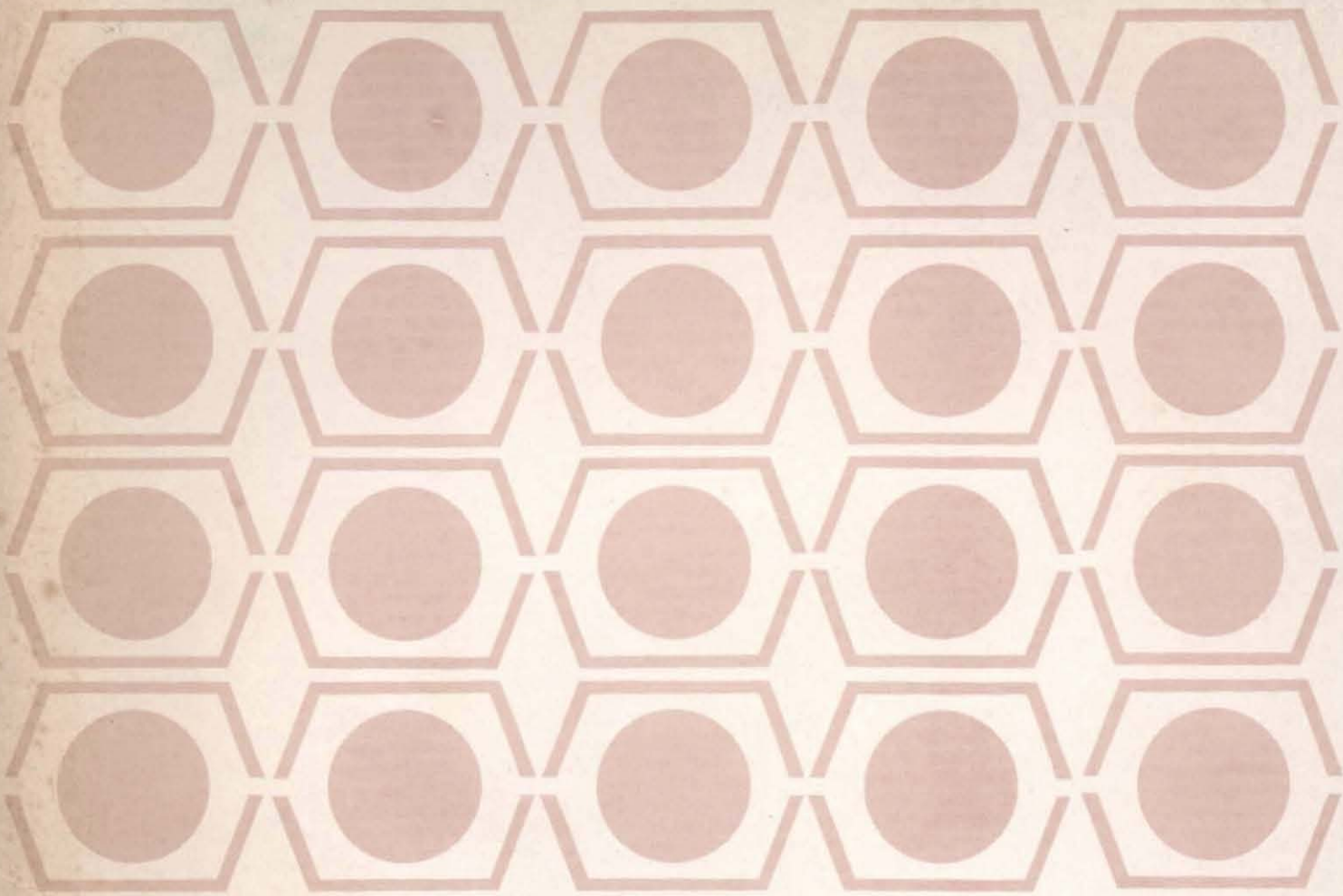


DECUS PROCEEDINGS

1962

PAPERS AND PRESENTATIONS

of

The Digital Equipment Computer Users Society

Maynard, Massachusetts

1962

PAPERS AND PRESENTATIONS

of

The Digital Equipment Computer Users Society

Maynard, Massachusetts

ACKNOWLEDGEMENT

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

President C. M. Walter

DECUS OFFICERS

March 1961-October 1962

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

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

PREFACE

This is the first Proceedings of meetings of the Digital Equipment Computer Users Society. Formed in March 1961, for the purpose of fostering the interchange of information, ideas, and the advancement of the art of programmed data processing - particularly with application to the Digital PDP-1, the Society (DECUS) has grown in numbers and in scope. DECUS now maintains a programming library facility for its members and issues DECUSCOPE, a technical newsbulletin, every month.

The papers presented at two Meetings which took place in 1962 are the subject of these Proceedings. A one-day Symposium was held May 17, 1962 at ITEK Corporation in Lexington on the subject: "Image Processing and Displays." A two-day Annual Meeting, in October 1962, was hosted by the Computation and Mathematical Sciences Laboratory, AFCRL, Hanscom Field, Bedford. The papers presented covered a wide range of subjects and the meeting was highlighted by a lively Panel Discussion called: MACRO, DECAL, and the PDP-1. Some of the papers given then are still in the germinal state but the authors were prevailed upon to contribute them. During 1962, users of a second Programmed Data Processor, the PDP-4, were welcomed to DECUS. More will be reported in the 1963 meetings about this data processor.

The rapid growth of DECUS and its diverse interests are evidenced by the presentations themselves. What may not be clearly visible is the remarkable spirit of cooperation in the interchange of such diverse information. The 1962 Proceedings are a testimonial of this cooperative spirit and a tribute to the authors. I regret that there was not space for the sparkling good humor and even wit, which enlivened the discussion between papers and during the questioning periods. Every user member was represented and participated fully.

DECUS is deeply grateful to all who have contributed to the substance and embellishment of this first endeavor.

Elsa Newman
DECUS Secretary

TABLE OF CONTENTS

FOREWORD

These Proceedings comprise a broad spectrum of papers whose color, in a figurative sense, ranges from the deep blues of special utility programs and debugging aids, through the lush greens of problem oriented techniques, to the rosey hues of new hardware aids designed to enhance the on-line use of computers. In organizing the papers we have attempted to portray the typical cycle of events centered about the utilization of a new class of computers.

Much initial energy has to be expended on the creation and improvement of utility programs and systems before anything very useful can be accomplished with our systems. To those of us who are strictly problem oriented, this is an extremely frustrating time, made bearable by the naive hope that it might be brief and end with some powerful general problem solving language in our possession. Unfortunately, this dream is inevitably dispelled as we proceed to call for a diversity of modes of control, and of action, which strain the existing hardware and programming systems to their technological limits, in our quest for useful results.

From the insight thus gained, however, is created the structure of new programming systems, and of processor configurations better fitted to provide each particular user with assistance in solving the problems of interest to him. The onset of the second stage of activity is already clearly discernible from the orientation of a majority of the papers in these Proceedings. The theme is closer man-machine interaction. This theme in present, both in the increased emphasis on on-line programming, debugging and problem solving aids utilizing scope and light-pencil communication, and in the requisite improvements in flicker-free scopes, time sharing hardware, and optical I-O devices.

It has been a singular pleasure, during the past two years, to have been in a position to observe the evolution of a Society which spans such a diversity of on-line processor configurations and uses. In this brief interval of time the small scale processor has evolved from a meager and inadequate substitute for a large central computer, into a formidable device whose flexibility and increasingly lower cost makes it the logical candidate for a multitude of real-time information processing operations.

The ease with which hardware can be tailored to particular applications has already out-stripped the software development problem. However, as the engineering technology rapidly improves, and the ultimate user becomes more intimately tied to the operating system, we may look forward to an era in which better control can be achieved and maintained over the growing software domain.

C. M. Walter,
DECUS President

Section I

UTILITY PROGRAMS AND TECHNIQUES

LAWRENCE RADIATION LABORATORY'S PDP-1 A PERIPHERAL PROCESSOR FOR LARGE COMPUTERS*

Dorothy Monk

Abstract

LRL's PDP-1, installed last summer, is now in full time operation as a peripheral processor for Livermore's Larc, Stretch, and 7090s, which are occupied full time with scientific computing.

We will discuss briefly 1) the motivation (economic and otherwise) which led to the choice of the PDP for this job; 2) general considerations which affect the specifications of systems and production programs used on a computer for this purpose; 3) some of the specific tasks with which our computer is occupied; 4) advantages and disadvantages observed so far of the PDP in this environment; and 5) some implications for the design of a peripheral processor of PDP class.

*The Editor regrets that the full paper was not received in time to be included in these proceedings.

TRANSLATION PROBLEMS OF A PERIPHERAL COMPUTER IN A MULTI-LINGUAL HOUSE

R. P. Abbott and L. E. Mish

Introduction

Datamation - the oracle of our burgeoning industry - has mentioned, on more than one occasion, that Lawrence Radiation Laboratory is a hodge-podge of incompatible machines. This paper might be considered, somewhat, as an "Insider's Confession." Our present computer configuration consists of three 7090's, two 1401's, one LARC, one STRETCH, one 650, and, of course, one PDP. We are in the process of converting the three 7090's to two 7094's. In the foreseeable future, the computer complex will be expanded to include a CDC 3600, as well as a CDC 6600. These machines speak such different languages as Decimal, Binary, Concise, BCD-eeze, XS3-eeze, and, of course, thet old standby, Hollerith. Frequently, the output from one of these machines is needed as input to a code which is on some other machine. Usually, the two machines do not speak the same language or the data must be rearranged or both.

In addition to the computer complex, there are many data gathering devices located at various testing stations, both at LRL and at other agencies. These devices generally speak one of the aforementioned languages, but the dialects include 35 mm photographic negatives, 5 or 8 channel paper tape, punched cards, and 7 or 8 channel magnetic tape. Most of these may be odd parity, even parity, or both. Thus, we are called upon to make a translation, conversion rearrangement, or both. The PDP, to which can be attached enough IO gear to be able to accept and generate all of the languages and dialects, was programmed to make these translations, conversion, etc.

Programming for the PDP

Two ground rules were established prior to initiating the programming.

1) Because there are at least two IO devices involved in each translation code, the processing speed of each code should be equal to the maximum rate of the slowest of the involved devices.

2) Whenever possible, all Input and Output data must be checked for validity and parity.

IBM cards - intermixed Hollerith and binary - to IBM magnetic tape

The translation from intermixed Hollerith and binary has been done on our 1401's, but to take advantage of the faster card reader and for backup reasons, it was decided to make this available on the PDP also.

Ground rule 1) calls for an examination of rates, so-oo. The card reader can run at 2100 cards/min. The maximum rate for putting card images on 15KC tape is about 3750 card images/min. The processing rate, by rule 1), shall be 2100 cards/min. Rule 2) says that we shall validity check the Hollerith cards. The binary cards will be checked by the large computers at read-in time. Each record on magnetic tape shall be checked and "standard tape techniques" shall be used. "Standard tape technique" means that if a bad record is encountered, that record shall be backspaced, a space equal to one record gap shall be erased, and the record shall be rewritten and rechecked. If ten erasure attempts fail to yield a good record, the problem shall be restarted and the tape replaced.

The conversion from Hollerith to BCD is not wired into the PDP so it must be programmed. The first approach took 103 decimal cells and, on the average, took 47.6 ms to empty the card reader buffer and convert. This was a weighted table search where a maximum of 16 searches were made for each character. The only trouble with this method was that the 47.6 ms is much too large for an individual card cycle of 28.4 ms when running at 2100 cards/min.

The method finally used is a direct table-look up using the Hollerith punches as the table address. For each character, the punches in rows 1 - 9 provide the table address, and the punches in rows 12, 11 and 0 provide a correction factor. This approach empties the buffer and converts in only 10 ms, which is well under the required 28.4 ms. Space-wise, the fast conversion is a dog. It takes 463 decimal cells, of which only 87 contain data or instructions

and the rest contain zeroes. The zeroes are important to the conversion, however, in that an invalid Hollerith punch is converted to one of these zeroes and when it is written on the even parity tape, a character skip will result and the standard tape technique will stop the problem.

An interesting sidelight is to be found in the time versus space analysis in the previous example. Speed was increased by a factor of 4.7. On the other hand, space was increased by 4.6. This rule was found to be true in other examples: That is, an increase in speed by a factor of N causes an increase in space by the same factor N.

Now that the code has been debugged and timed, it is running at 1600 to 1900 cards/min, instead of 2100. This discrepancy seems to be due to the summation of plus and minus fudge factors on the time quotations for the card reader and tape drive. For instance, the card reader time is not a hard fast 28.4 ms, but more like 28.4 ±2 ms, under ideal conditions.

Tape start time, and tape stop time, are quoted at ±1 ms. The fact that the PDP is a 5 microsecond machine doesn't begin to dent mechanical delays on the order of 5 ms. If a highly integrated timing study is made, it might be possible to strike 2100 cards/min, but is it really worth the effort?

IBM magnetic tape to LARC magnetic tape

Another translation problem of interest is the translation of IBM magnetic tape to LARC magnetic tape, or vice versa. Our PDP has an IBM compatible tape control and a LARC compatible tape control. Each control has 2 tape drives. To make this a general purpose routine, it was decided that the code should stand ready to accept variable length records. Rule 2) (validity & parity checks) will be satisfied by standard tape techniques. A look at rule 1) (timing) provides the brilliant idea of simultaneity. That is -- let's start one tape reading and, at the same time, start the other tape writing and perform the translation while they are in motion! It is sort of a "he takes the high road and she takes the low road and I'll be in Scotland before them." Only, it doesn't work. The command structure of the PDP (specifically lack of index register) is such that the bookkeeping necessary to convert three separate characters from a PDP word requires more than 200 microseconds per word converted.

PDP Saves Processing Steps

The procedure which was finally used is to start the read tape, convert the first N words; start the write tape, and finish the conversion of the remaining words. The processing speed attained with this method is about 6 ms longer than the read time. This particular translation code is a good illustration of the processing steps which can be saved with the PDP. This code takes output from, say, a 7090 and, in one step, produces LARC input. The old method consisted of an additional processing pass on the 7090 to prepare a Hollerith image tape, which was punched off line on a 1401. The cards were then converted to a LARC tape by a Remington Rand Card-to-tape Converter.

Other routines

Other conversion routines on the PDP include:

Paper tape to magnetic tape, with options to check odd or even parity on the paper tape, write odd or even parity IBM or Remington Rand tapes.

Print LARC or IBM tapes on the PDP Printer.

Magnetic tape to visual CRT, and precision CRT with 35 mm camera. This routine handles both characters and graphical data.

35 mm negatives with graphical data to magnetic tape.

IBM tape to IBM tape.

LARC tape to LARC tape.

Magnetic tape to printer and magnetic tape to precision CRT - simultaneously.

Our assembly routine deserves mention in that it reads the instructions from Hollerith cards, uses IBM tape as temporary storage, prints the listing on the Anelex printer at 1000 lines/min (the printer uses the XS3 language), and the object code is punched in binary form on IBM cards.

Conclusion

The coding techniques which were finally used in our conversion routines are not complex, but they do serve to illustrate that one cannot afford to approach the trivial problem of data conversions with careless contempt.

A SYSTEMS TAPE FOR THE PDP-1

Fraser Bonnell

Abstract

Routines which are frequently used are stored on magnetic tape. In order to use these routines, it is sufficient to start the PDP with a short paper tape (under 100 words long) and then to type four characters which identify the desired routine. A loader on the systems tape then finds the routine on the tape, loads it into memory, and transfers control to it. A routine has been written for altering the systems tape.

The Systems Tape

The systems tape is started by means of a short paper tape. The paper tape contains a program which finds a loader on the systems tape, loads the loader into memory and transfers control to the loader. Once the loader is in memory, the systems tape may be used simply by starting the computer at a specified location. The systems tape loader does the following:

1. Types - TYPE ID.

2. Reads an ID from the typewriter. (Each routine on the systems tape is identified by a three-character ID.) The loader will accept the last three characters typed before a slash (/) is typed.

3. Searches a table for the ID. (If the ID is not found in the table, the machine will type ---ILLEGAL ID---and start again at (1).) Gets from the table the number of the record in which the desired routine is to be found.

4. Moves the systems tape to the proper record.

5. Searches the record for the desired routine and reads it into memory. (Several short programs may occur in the same record. This eliminates unnecessary record gaps.)

6. Checks for tape reading errors. (If there are errors, several attempts are made to read the tape. If none are successful, the machine will type---TAPE ERROR---and then proceed as if there were no errors.)

7. Gets a tape disposition instruction from the systems tape. Generally, this instruction will be to start the systems tape moving to the head of the nearest loader, so that it will be in position for the next use. However, certain routines called from the systems tape will need to use immediately the tape control which controls the systems tape. For this reason, there is the option of leaving the systems tape where it has stopped.

8. Transfers control to be called routine. The systems tape may still be moving to the nearest loader. The loader occurs at intervals on the systems tape so that no matter what position the tape is in, a loader is not far away. Optimal placement of loaders has not been determined.

Diagram for Systems Tape Records

Figure 1 shows a diagram which describes the format of a "standard" systems tape record. Provision is made for nonstandard records. For example, a routine on the systems tape may be divided into two parts. The first part may be called in the normal manner, while the second part is stored in the next systems tape record to be read by the first part at a later time. Or, the second part may be read into the part of memory which is occupied by the systems tape loader.

One should have a routine for altering a systems tape. One should be able to delete and insert both programs and whole records. The routine must make the necessary changes in the loader's table of ID's and their corresponding record numbers. It should be able to read a program from paper tape on other media and arrange it in the proper format for storage on the system tape. It should see to the placement of loaders at specified intervals. (Because certain records on a systems tape should not be separated by a loader, it is a good idea to use one word in each record to indicate whether a loader may immediately follow that record on the systems tape.)

FORMATS OF SYSTEMS TAPE RECORDS

I) Loader:

Record Number	000000 (Indicates that this is a loader)	The address of the first word of the loader's table	ID_1	$RN_1^{(*)}$	ID_2	$RN_2^{(*)}$		ID_m	$RN_m^{(*)}$	Loader Program

Loader's Table

(*) RN_i is the number of the record in which is found the routine whose ID is ID_i.

II) Other Records:

A) Standard

Record Number	A positive number means advance to nearest loader. A negative number means backspace to nearest loader.	S-1, where S is the number of records which must be skipped to get to the nearest loader. (S < 0, if next record is loader)	$I_1^{(*)}$	$F_{1,1}$	$L_{1,1}$	$(L_{1,1} - F_{1,1})$ words (To be put in locations $F_{1,1}$ through $L_{1,1} - 1$)	I_1	$F_{1,2}$	$L_{1,2}$	$(L_{1,2} - F_{1,2})$ words	- - -	I_1	F_{1,k_1}	L_{1,k_1}	$(L_{1,k_1} - F_{1,k_1})$ words	343536 (Indicates end of program; will not be an ID in concise code.)

Systems tape disposition. (Stop or else move to nearest loader)	"Jump Address"	$I_2^{(*)}$	$F_{2,1}$	$L_{2,1}$	- - -	$I_p^{(*)}$	$F_{p,kp}$	$L_{p,kp}$	$(L_{p,kp} - F_{p,kp})$ words	343536 (End of program)	Systems tape disposition	"Jump Address"	343536 (Indicates end of record)	000000 (†)	000000 (†)

(*) $I_1, \ldots, I_p$ are the ID's of the P programs contained in the record.

(†) The last two words of the record may be used to give information to a routine which alters the systems tape.

B) Nonstandard

The first three words of the record must be the same as on a standard record.

TED: A TAPE EDITOR

Charles R. Brown and Donald W. Connolly

Abstract

The principle and operation of a utility program for the PDP-1 computer are described. The program is an aid in the editing or modification of alphanumeric text in that the operator may communicate with the computer in the very alphanumerics of the text itself. It is a computer time-saver in that the modifications and the control instructions for their accomplishments may be prepared at an inexpensive, off-line machine.

Introduction

TED is the OAL program for preparing and editing symbolic tapes with the aid of the PDP-1 computer. With TED all typing involved in preparing and editing tapes may be done at an off-line typewriter punch. Thus, TED permits off-line editing in the sense that all typing may be done off-line and the only on-line time required is that necessary for the reading of text and control tapes.

Editing Features

With TED tapes are edited by killing a line or block of lines, inserting a line or block of lines, and substituting one line for another line. Memory may be inspected by requesting one or more lines to be typed or displayed on the scope. A line may be referred to by its symbolic address. This is true for the typing, displaying, reading, punching, killing, inserting, and substituting of lines.

TED has a text mode and a control mode; a mode change occurs when an overstrike is typed in or read in. When TED is in text mode, characters are stored in the temporary text buffer until a carriage return occurs. The carriage return resets the temporary text buffer and adds the line to the permanent buffer.

In control mode characters are stored in a temporary control buffer. A carriage return resets the control buffer and causes the control statement to be executed. Control statements are not stored in a permanent buffer.

The backspace operates in both modes to kill the last character. In control mode it operates on the temporary control buffer; in text mode it operates on the temporary text buffer. When in control mode, the typewriter types in black. Thus, the typewriter is in red only when characters are being stored in a text buffer. TED starts in control mode, black and lower case.

Control Characters

Control statements in TED begin with a control character that denotes the operation to be performed, and operates upon one or more lines of the permanent text buffer. The control characters and their general meaning are listed below.

Character	Meaning
t	type
. (period)	"
, (comma)	"
d	display
p	punch
r	read
k	kill
· (middle dot)	"
/ (slash)	"
i	insert (after)
s	substitute
l	leader

Listed below are the eight control statements which consist only of a control character. Carriage return is indicated by ↓.

Single Character Statements

Statement	Meaning
t ↓	types all lines
. ↓ (period)	types last line
, ↓	types first two lines
p ↓	punches all lines
r ↓	reads all lines
/ ↓	kills all lines
· ↓ (middle dot)	kills last line
l ↓	punches leader

Control Statements With Symbolic Addresses

All control characters which are letters (except l) may be used with symbolic addresses; the others may not. In TED a symbolic address is the string of characters from the carriage return fo the tab. For DECAL symbolics, block symbols, program symbols, and system symbols would normally be symbolic addresses for TED. In the following examples, the symbolic addresses are block symbols in the DECAL system. For TED a symbolic address may not contain spaces or begin with a period. A control statement with one symbolic address is simply the control character followed by the symbolic address. An example is the "type" statement:

tstart: ↓

which will result in the line with the symbolic address start: being typed. This example is appropriate for the control characters p, d, and k. The control characters r, i, and s require some additional explanation. The control statement

rstart: ↓

will result in the tape being read up to and including the line with the symbolic address start: . The control statement

istart: ↓

will result in the last line of permanent text buffer being inserted after the line with the symbolic address start: . The control statement

sstart: ↓

will result in the last line of permanent text buffer being substituted for the line with the symbolic address start: . This statement is equivalent to

istart: ↓
kstart: ↓

A line which does not have a symbolic address may be referenced by indicating the number of lines it is below a line with a symbolic address. Any line with a symbolic address may be chosen. The control statement

tstart:/5 ↓

will result in the fifth line after start: being typed. The count is in octal and should not exceed octal 77. The slash is used here rather than plus to avoid case shifts.

The control characters t, d, p, k, and i may be used with two symbolic addresses. The format is the control character, the first symbolic address, a comma, the second symbolic address. The "type" example

tstart:,finish: ↓

is appropriate for d, p, and k. The first symbolic address indicates the first line of a block; the second symbolic address indicates the last line of a block. Thus, the example would result in the block of lines beginning with start: and ending with finish: being typed.

The insert statement has a different meaning. The control statement

istart:,finish: ↓

would result in the block of lines beginning with finish: and ending with the last line of permanent text buffer being inserted after start: .

Special Features

Whenever TED reads from tape a control statement it cannot execute, it does one of two things. (1) If the statement begins with a legitimate control character the reader stops, the unexecutable statement is typed, control is turned over to the typewriter, and TED goes to control mode. Thus, the correct control statement may be typed. (2) If the control statement does not begin with a legitimate control character it is ignored. Unexecutable control statements (beginning with a legitimate control character) which are typed in result in "error" being typed back. Unexecutable statements generally occur because a symbolic address was incorrectly typed.

If TED's permanent text buffer is filled, "buffer full" will be typed out in red. If this occurs while TED is trying to satisfy a read statement, the reader will stop, control will be turned over to the typewriter, and TED will go to control mode.

If sense switch 5 is up when the buffer is filled, TED will punch and kill the first seven lines and continue reading, repeating this until the whole tape has been read. Thus, TED will read a tape of any length. However, control statements should not refer to lines which have been read more than two or three pages earlier since this much ordinary symbolic program material is about the capacity of the permanent text buffer.

TED assumes that a tape begins in text mode. Thus, if a tape is to begin with a control statement, the tape should begin with an overstrike. When TED finishes reading a tape, it always turns control over to the typewriter and goes to control mode.

A display resulting from a statement which is typed will be terminated by the striking of any key except space bar. Striking the space bar will cause the block being displayed to be indexed. Thus, if lines three through five are displayed before the space bar is struck, lines four through six will be displayed after the space bar is struck. In this manner, the display may be stepped through an entire program.

If a display statement is read from tape, the display will be modified by the space bar as described above. A carriage return will cause the display to terminate and TED to continue reading tape. Typing a left paren will turn control over to the typewriter and TED will be in control mode. All other keys are ignored.

TED displays both lowercase and uppercase characters. Thus, a block of lines which is displayed will look just as it would if they were typed. The only qualification is that the display feature has only two fixed tabs.

Suggestions and Cautions

In the following, ⇑ indicates uppercase, ⇓ indicates lowercase, and ⇒ indicates tab. So that the first line of a symbolic tape will always have a label, it is suggested that the first line be:

⇑⇓ ⇒

If this is done, there will never be a line in a symbolic tape that cannot be referenced by a label, since control statements of the form

t⇑⇓/n

are permissable. This procedure will not affect the complication with DECAL.

In order that material typed off-line will look as if it had been typed on-line, change ribbon color after the overstrike is struck. Tapes should start in red.

If several lines have identical symbolic addresses, TED assumes that a control statement with such a label refers to the first one.

If tapes are to be compiled with DECAL, one may tab after dss or fin., making these symbolic addresses for TED.

Before returning the carriage or tabbing, the typewriter should always be in lower case. If this is not done, TED may not recognize symbolic addresses and displayed lines may be in the wrong case.

TED will read leader and trailer consisting of feed holes or "7" holes. Errors made by the operator of the off-line typewriter-punch should be corrected by backspaces and never by "delete" punches.

APPENDIX

Examples of Editing with Ted

The following subroutine, in symbolic form, with hand-noted changes could be edited in a variety of specific ways. Two examples are given.

```
dss              hold mdck     ipt ⟵ dss check
dss cr
dss prmd black wait
dss temp cntrl os
rpt'---------law 0
                 eli
                 rpa'-
                 rcr-6
                 rir-1
                 spi-
                 jmp a
                 sft 4
t:               dac hold
                 jmp mdck
a:               szf' 4
                 jmp rpt
                 clf 4
                 dzm ipt
                 lac prmd
                 sad black
                 jmp wait
                 lac cr
                 dac temp
                 law' 2
                 dac cntrl
                 lac os
                 jmp t
fin.
```

rpt' → rpa' jsp check

The original symbolic tape is first read into TED by striking r ⇓. The control tape below is then read the same way.

Off-line control tape:		Function:
rpt'*	rpa'*	Text to be substituted
srpt'		Control instruction to execute same
krpt'/1,rpt'/5		Removes lines "cli" through "spi"
	jsp check*	Line to be added after "rpt"
_* irpt'		Inserts above line after "rpt"
dss check*		New dss line to be added
idss		Inserts above after first dss, which has been made a "symbolic address" here
ddss,a:/15		Will cause display of entire edited text At this point striking a carriage return would cause the program to continue reading the control tape. Striking the (would return control to the operator
1		Will make 500 lines of feed holes for leader
p		Will punch off entire edited text
1		Will make trailer, as above
/		Will kill entire text, preparatory, for next editing job

Example 2:

Off-line control tape:		Function:
dss check*		Text to be inserted
rpt'*	rpa'*	
_*	jsp check*	
irpt'/5,fin.'1		Will insert all three of the lines in red (these having been added to the permanent text following fin. which here has been made a symbolic address by a tab after the period).
krpt'/rpt'/5		Will kill the six lines at the top beginning with rpt'
1		Leader, punchoff and trailer as before
p		
1		

Note: The extra carriage returns in the above examples are not needed in the actual control tape.

*These symbols print out in red.

SCOPETRACE

John R. Hayes

Abstract

A computer program for the Digital Equipment Corporation PDP-1 computer, but applicable in principle to other computers, is described. The program is designed to facilitate the debugging of object programs by providing a geometrical representation of the operation of the object program.

Introduction

Scopetrace is designed to aid in the process of debugging programs. It provides a pictorial mapping on the scope of the operation of the object program and simultaneously displays the following four quantities.

a. the contents of the accumulator,

b. the contents of the in-out register,

c. the address of the instruction being executed,

d. the effective instruction, i.e., that instruction which the computer performs after indirect addressing and execute commands have been taken into account.

The Scopetrace Display

A typical scopetrace display is shown in Figure 1. The segment of program traced in Figure 1 is listed in Figure 2. In the top row, from left to right, are shown the contents of the ac, the contents of the io, the address of the instruction just executed, and the effective instruction. The addresses being traced are shown in the first or left-hand column.

The first address and all following addresses which end in zero are displayed in full. All other addresses are indicated by a short horizontal bar.

The left-hand side of the second column contains arrows which indicate the course of the program. The arrows are assigned by the following four rules:

a. When control proceeds from register n-1 to n and then to n+1, the arrow " ↓ " is placed next to the address n, i.e., the program moves through consecutive instructions.

b. When control proceeds from n-1 to n to some register other than n+1, the arrow " → " is assigned, i.e., the program is shown to "jump out."

c. When control proceeds from some register other than n-1 to n and then to n+1, the arrow " ← " is used, i.e., the program is shown to "jump in."

d. When control proceeds from some register other than n-1 to n to some register other than n+1, the arrow " ↻ " is assigned, i.e., the program is shown to "jump in" and then "out" again.

The right-hand side of the second column shows the number of consecutive times that the arrow in the left of the column was assigned at the location. Whenever an arrow is assigned to a location which is different from the last arrow assigned to that location, the pictorial mapping moves over a column. Thus, when the sample program jumped out from 4021 and jumped back in at 4010, the mapping moved from column 2 to column 3. Again, when the program jumped out at 4020, the mapping moved from column 3 to column 4.

Control of Scopetrace

The steps outlined below should be followed to obtain pictorial mapping of a program:

1. Read in the Scopetrace tape and start the program (Loscopetrace starts at 1, Hiscopetrace starts at 5700.

2. Type in the following four numbers, each followed by a space:

a. the first address to be displayed.

b. the last address to be displayed. (Note: the difference between the first and last address should not be greater than octal 75 for Loscopetrace or octal 64 for Hiscopetrace.)

c. the address at which the object program starts.

d. the memory option, either 10 or 30. With the 10 option, 6 display columns are available for pictorial mapping and 10 octal words of storage are required per instruction mapped. With the 30 option 22 display columns are available for mapping and 30 octal words of storage are required per instruction mapped. The 30 option should not be used in Hiscopetrace when more than 24 octal instructions are being mapped. To do so would cause the program to exceed the available storage. When the final space has been typed, the address column will appear on the scope.

3. Select a "breakpoint," that is, select the first address at which you want program mapping to pause and put this address in the address portion of the test word. If you want the mapping to pause on the nth time it comes to the breakpoint rather than the first time, then, place n in the instruction part of the test word.

4. Type the letter "b," Scopetrace will halt at this point to allow the operator to reset the test word. Thus, use of the test word to set the breakpoint does not prevent mapping of programs which use the test word.

5. Press continue. The full Scopetrace display will now appear.

The breakpoint may be reset as often as desired. Each resetting will allow mapping to continue to the new breakpoint.

Single stepping through the program may be accomplished, after the first breakpoint has been set, simply by striking the spacebar.

The use of breakpoints and single stepping will suffice for almost all cases. Some cases which cause trouble should be mentioned, however. The one normal process which requires special treatment is typing in. To accomplish a type-in, the following steps are necessary:

a. Set the appropriate breakpoint address into the test word.

b. Strike "carriage return."

c. Strike the character that it is desired to type-in. Scopetrace halts at this point to allow resetting of the test word.

d. Press continue. This completes the type-in.

Scopetrace will not execute an instruction which would modify the scopetrace program. Instead tracing stops before executing such an instruction and the mapping of the program up to that point is displayed on the scope.

An illegal instruction in the object program will halt Scopetrace. Loscopetrace will halt at 502 and Hiscopetrace at 6401.

Storage Required

The program plus fixed storage occupies 1077 octal registers of memory. Loscopetrace occupies the region from 1 to 1100 and Hiscopetrace, from 5700 to 6777. The extra storage required depends on the number of instructions mapped and on the memory option. (See Scopetrace control 2d.). To calculate the octal number of extra storage registers required, simply multiply the number of instructions mapped by the memory option.

Comment

While Scopetrace was designed primarily as a debegging tool, it is also useful as an aid in teaching programming. Since the accumulator, the input - output register, and the effective address are displayed, the effects of the various instructions may easily be demonstrated.

Figure 1

A TYPICAL SCOPETRACE DISPLAY

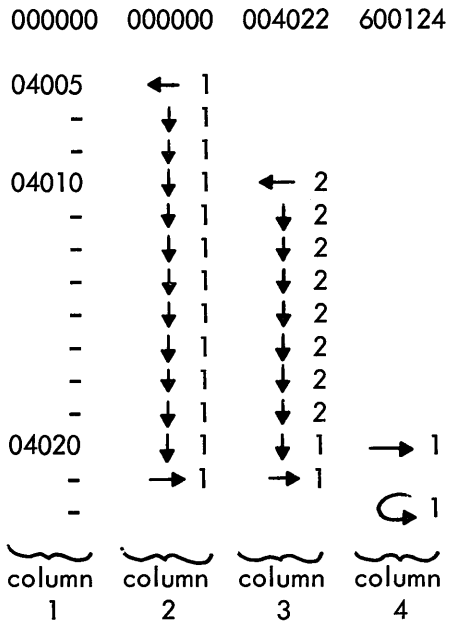


Figure 2

A TYPE-OUT ROUTINE

```
typeword'       dac save
                law' 3
                dac count
loop:           lac save
                cli
                rcl 6
                dac save
                dio temp
                lac temp
                sas oct76
                tyo'
                isp count
                jmp loop
                jmp rml
save:           ..
count:          ..
temp:           ..
oct76:          oct76
                fin.
```

MATRIX PACKAGE FOR THE DX-1 EXPERIMENTAL DYNAMIC PROCESSOR AT AFCRL

Carmine J. Caso, James M. Sexton, Janet N. Seltzer

Abstract

The Matrix Package is a system of computer programs to perform general Matrix Operations. It is centered around an executive routine designed to allow effective usage by people with minimal programming experience.

The Matrix Package breaks down into two parts consisting of an Interpreter and Control Portion, and an Operation Portion. This sub-division is necessitated by the form of the input operation request data which is similar to certain Fortran statements and other problem oriented languages. The Interpreter and Control Portion of the system accepts the input operation request data, whether it be from punched paper tape or from the on-line typewriter, and interprets each statement as a specific request. In doing so, an operation command table is generated and subsequently used by the Control Portion in guiding the operation of the stated requests. The Operation Portion of the system performs the operations as requested by the Control Portion. The Operation Portion contains all the necessary routines to perform the Matrix Operations as indicated by the input request data.

The Matrix Package operates from magnetic tape with the Interpreter and Control Portion being the first block. The subsequent tape blocks contain the programs of the basic Matrix operations, add, subtract, multiply, eigenvalue, eigenvector, inverse, and transpose, and also the necessary input-output routines.

Input-output media may be punched paper tape, the on-line typewriter and magnetic tape.

There are three types of statements for the operations request data. The first such statement is the Dimension statement. This statement will be used to allocate storage locations to the Matrices involved in the subsequent input request data statements. All Matrices used must be mentioned here; otherwise, an error printout will occur when the Interpreter Portion discovers a Matrix name which has not been assigned storage area. The Dimension statement must be the first statement of the input request data. Its format consists of: 1) the word SIZE which is a notation to offset this type of statement from the others, and to indicate that its contents are the Matrices' dimensions: 2) the Matrix name, symbol or identification, which must be at least one character and no more than three characters; and 3) the actual Matrix dimensions enclosed by parenthesis and separated by a comma. As many Matrices' names as desired may be used until storage allocation is exceeded.

E.G., SIZE A(5,5), BB(20,20), CDE(1,10)

The CALL Statement

The second type of operation request data statement is the CALL statement. This type of statement is subdivided into two parts, Input/Output, and Functional. The input/output statements contain certain components; a call code, an I/O source, and I/O address, a data format, and in the case of Magnetic tapes, a block number. The call codes are READ, PUNCH, PRINT, WRITE, and CALL. The I/O sources are PAPER for paper tape, TYPE for the typewriter, and MTAPE for Magnetic Tape. The I/O address may be any Matrix name, symbol, or identification. The data format specifies the type of conversion to be performed between the internal machine language and external notation. The notations used are I for integer, F for fixed point and E for floating point. The number of positions of the field which appear to the right and left of the decimal point are required when using the fixed and floating point data format. All Input/Output statements are limited in that certain components are legal only with specific components. For example, it is correct to READ PAPER, READ TYPE, and PUNCH PAPER, but is is illegal to PUNCH TYPE, which is also somewhat of a difficult task. Punching output on the on-line typewriter would be quite a job to perform. Input statements do not require a data format whereas Output statements do need a format. Magnetic Tape call statements require a block number for its data format.

OPERATION	FORM
READ PAPER TAPE	READ PAPER, A,
READ ON-LINE TYPEWRITER	READ TYPE, BB,
PUNCH PAPER TAPE	PUNCH PAPER, CCE, F3.2
PRINT ON-LINE TYPEWRITER	PRINT TYPE, BB, 14
WRITE MAGNETIC TAPE (Block 2)	WRITE MTAPE, A, B2
READ MAGNETIC TAPE (Block 6)	CALL MTAPE, CDE, B6

Functional Statements

The functional statements are concerned with the calling of a particular Matrix function. These statements contain three components; a call code, an input address, and an output address. The call codes are EVALUE for the Eigenvalue function, EVECTOR for the Eigenvector function, INVERSE for the Inverse function, and TRANSPOSE for the Transpose function. The input and output address is the Matrix name, symbol, or identification. An example of a functional statement is TRANSPOSE ABC, XYZ, which will be interpreted to mean - find the Transpose of Matrix ABC and store the results in the Matrix area XYZ.

The Arithmetic Statement

The third and last type of operation request data statement is the Arithmetic statement. This type of statement is concerned with three Matrix operations; addition, subtraction, and multiplication. These statements are made up of the Matrix name, symbol, or identification, the storing element, and the arithmetic operations. For example, A = BC + DEF where A, BC, and DEF are Matrix names, the literals define the storing element and the plus sign is the arithmetic operator. Other operators are the minus sign (-) and the multiplication sign (x). The Matrix area where the result of the operation is to be stored is always to the left of the literals. To the right are the Matrix names used in the operation along with the desired operator.

Input Statement

The media of inputting the source data used by the Functional and Arithmetic Statements is determined by the Input statements of the operation request data. An example would be READ TYPE, ABC, which would be interpreted to mean read from the on-line typewriter, data input to be stored in the Matrix area assigned ABC. Another example, READ PAPER, Z1, would mean read input data from paper tape into the matrix area Z1.

On-Line Output

On-line output may be Monitor Information, Error Diagnostics and Data Listings. When the Matrix Package operation is initiated, a self-explanatory message will be printed on-line. This message, determined by the setting of sense switch 1, will inform the operator as to the media by which the input operation requests are to be entered. Sense switch 1 down signifies that the input operation requests will come from paper tape and if it is up, that the input operation requests will come from the on-line typewriter.

As the sequence of commands indicated in the input operation requests proceeds, a running log of operations is printed on-line. The messages may be information as to the present operation or they may be certain instructions for the operator to follow. Monitor information may be omitted by setting sense switch #2.

Error diagnostics will be printed on-line whenever a grammatical or machine error is detected. The printout will be a three digit code in red, which will represent a specific explanation of the error.

On-line data listings as requested in the operation requests will be listed, headed with the Matrix name symbol, or identification. The listing will have a maximum of eight columns across the page. If the row has more than eight columns, the remaining columns will be listed directly below the preceding row. There will be a blank line between each row.

Single Precision Floating Point

The Matrix Package in its present form as designed for the DX-1 computer uses 8K Core Memory, the majority of which is used for data storage, and also uses three magnetic tapes (type 52 control), one of which is required; this being the System tape. Another tape is used only if magnetic tape input/output statements are used. The third tape is used when the Eigenvector function is called on. The arithmetic used in this version of the Matrix Package is single precision floating point.

DECAL-BBN
(BBN Symbolic Version of DECAL)

R. J. McQuillin

Abstract

The following talk reports progress on the new version of DECAL presently being completed at Bolt, Beranek and Newman. This version will have many new features. At this time a report will be given on the improvements to the existing system as well as the addition of ALGOL-like features to the algebraic compiler.

Introduction

For some time work has been going on for the purpose of making a good symbolic version of DECAL. The starting point was to produce a disassembly of the old DECAL F17C, which existed only as a binary tape. The disassembled listing was reorganized and written in DECAL symbolic form. After this first symbolic version of DECAL F17C was produced, work was begun on extensive rewriting of the system. DECAL F17C had been patched and modified so often there were many artifacts that were awkward and clumsy that were used so that things would compile properly. These artifacts have been removed in the rewrite. In fact, the rewrite has been so extensive that the name of the system has been changed to DECAL-BBN to reflect the differences between old and new systems. Particularly affected in the new version is the algebraic compiler.

In addition to DECAL itself, the linking-loader feature was also rewritten, partly to make it more elegant, and partly in order that it would be able to handle the increased capabilities of DECAL-BBN.

The New Features of DECAL-BBN

DECAL-BBN has been extensively debugged. (One of the drawbacks of older versions of DECAL were the errors in the system.) In addition, to correcting of eliminating errors the system has been made much more efficient, both in speed of compilation and in the economy of the compiler itself. The new version of the algebraic compiler is very efficient, in fact, temporary storage allocation is kept to a minimum at all times.

The following completely new features have been added:

1) Improved Typeout Format. In DECAL-BBN only error typeouts are done while a program is compiling. All other typing is done at the end of the compilation and only after all punching is completed. In addition, all typeouts may be suppressed through sense switch option. An action operator is written which will allow all the normal typing to be punched out at the end of compilation. Then the tape can be run off-line to get the listings.

2) Conditional Statements. The conditional statements are handled exactly as described in the ALGOL manual. In ALGOL language:

```
    if B then U else S
or  if B then U
```

where B is a Boolean expression; U is an unconditional statement; and S is a statement. These items are the same as described in the ALGOL manual. As example, we may write conditional statements in the following ways:

```
if A > b then goto c
if (a > b) ∧ (c ≠ d) then d else goto e
if a then goto b else if a > c then goto e
if a > b then c else d then goto e else goto f
```

3) For Statements. The for statements are also as described in the ALGOL manual. They are of the form:

```
for k <= F do S
```

where k is a variable, F is a for list and S is a statement. These items are also described in the ALGOL manual. We may have for statements of the form:

```
for i <= 0 step 1 until n do S
for i <= m,n step 1 until p do S
for i <= 0 step 1 while a < b do S
```

4) Procedures. Procedures, as described by the ALGOL manual, are like subroutines in machine language. The differences are that they compile in a standardized way, and that they allow a variable

number of arguments. When a procedure designation is encountered in statement, a certain standard coding is produced. For example, suppose we have an expression like:

a + J (k,r) => b

Here J may be a procedure for finding the Bessel function arguments k and r. (Actually according to ALGOL, procedures that have a single value are called functions, so we may call J (k,r) a functional designation.) The coding that is produced here is:

```
lac  K
jda  J
jac  r
add  a
dac  b
```

We may declare a precedure by using the expression:

procedure J(k,x); B

Where B is the body of the procedure. We may also specify a definite result to be communicated back upon exit from the procedure. Thus, suppose we wish to find the Hankel function and return with the address of the first element of the two-element array which contains the results - the first element contains the Bessel function, and the second element the Neuman function. Then we would write:

procedure H (k,r,H); result adr H; B

5) Subscripted Variables. DECAL-BBN will have the capabilities of handling subscripted variables. It will actually produce an intermediate coding which will be handled by a subscript interpreter during run time. Thus we may write:

A [i,j] + B [i,j] => C [i,j]
A [ixj,k] + B [i/j,k] => C[i+j,j+k,k+m]
lac A [i,j]
if A [i,j] > 0 then goto b

We may also have subscripts on subscripts. Thus:

A [i[j]] => b

will be handled properly by DECAL-BBN.

6) Hybrid Operators. DECAL-BBN has a new kind of operator called the hybrid operator. It is a combination of an instruction generator and an action operator. More precisely, it is an action operator with precedence.

7) Address Arithmetic on System Symbols and Undefined Symbols. Additions have been made to both the DECAL-BBN and to Linking Loader-BBN to make possible address arithmetic on system symbols and undefined symbols. Thus we may write:

```
        dss  a
        jmp  a + 1
or      lac  a + 1
    a:  . .
```

The form for writing address arithmetic on Systems S or undefined symbols is not general but sufficient for most purposes. It must be of the form; the location followed by a (+) or (-), followed by an octal number.

Other Features

The basic structure or skeletal form of DECAL-BBN is such that it requires no further recompilation of the system. Supplementary, however, to this basic structure are separate tapes of action operators, instruction generators and words. The programmer may select only those of the above that he needs and feed them into the basic system. Any modification of the basic DECAL-BBN is made through action operators, and not through recompilation.

Recommendations for Increased Communications

It is hoped that DECAL-BBN can be adopted as the standard compiler of the DECUS organization. The DECAL-BBN symbolic form makes possible the use of action operators. These action operators can then be communicated in their symbolic form to DECUS members. There is every indication that a standard programming language for PDP-1 users would insure a larger medium for communication. In addition it is hoped, that the algorithms as published in the Communications of the ACM will be easily and closely translatable into PDP-1 code, thus opening a vast communication link to the whole computing community.

Conclusions

In this paper I have discussed only some of the new features contained in DECAL-BBN which will make it a more elegant system than its predecessor. The hybrid operator and the incorporation of ALGOL-like features described earlier make the new version a really useable tool for communication among PDP-1 users and between the DECUS members and other computer users.

Section II

PROBLEM ORIENTED TECHNIQUES

MINIMAX DETECTION STATION PLACEMENT

Richard D. Smallwood, Lt.

Abstract

One problem in the detection of enemy operations is the location of a given number of detection stations within a limited area. A practical requirement is that one assumes complete knowledge of the enemy of the effectiveness and location of the station net. This in turn necessitates a minimax solution of the problem, that is, the stations must be placed so that the minimum probability of detecting an enemy event within the area is a maximum.

The solution of the simpler one-dimensional problem, i.e., the placement of n stations on the unit line, has been programmed on the DX-1 computer. The program uses an iterative technique with typeouts and a display available after each iteration.

One problem that arises in implementing a network for the detection of specific operations over an area of interest, is the optimum placement of a given number of detection stations. It is of particular interest, to know where n stations should be placed in the area so that the probability of detecting an enemy operation is maximized.

One assumption that is made in the analysis is that the enemy will know the effectiveness, as well as the position, of each of the stations. With this knowledge the enemy will certainly cause any operations to occur where our probability of detection is a minimum. Thus, the problem is to place the stations so that this minimum probability of detection is maximized.

In solving this problem, we assume that we are given a function p(x) which is the probability of detection for one station if an event occurs a distance x from the station. Furthermore, it is assumed that each of the stations operates independently from the others so that the overall probability of detection, P, for an event is one minus the probability that none of the stations detect the event:

$$P = 1 - \prod_{i=1}^{n} \left[1 - p(x_i)\right]$$

where x_i is the distance from the event to the i^{th} station.

This problem has not been solved for an arbitrary area; however, a set of necessary conditions for the optimum placement of n stations on the unit line has been derived (1). If we write the overall probability of detection for an event occuring at x ($0 \leq x \leq 1$) as P(x), then these conditions require that the value of P at the end points, and the minimum value of P between each pair of stations all be equal.

The solution of this one dimensional problem for any number of stations and an arbitrary function p(x) has been programmed on the PDP-1. The optimum placement of the n stations is found by an iterative technique that attempts to minimize a function that is a measure of the deviation from the optimum placement. If a_o is the value of P(x) at x=0, a_n is the value of P(x) at x=1, and a_i is the minimum value of P(x) between the i^{th} and $(i+1)^{th}$ stations, then the technique attempts to minimize:

$$V = \sum_{i=0}^{n} \left| a_i - \frac{1}{n} \sum_{j=0}^{n} a_j \right|$$

or the absolute deviation of each of the critical values of P from the average.

In addition, the program displays a measure of P(x) on the color scope at the end of each station so that the convergence of the solution may be observed visually.

Several interesting results have been obtained from this program. It is hoped that this work will lead eventually to the solution of the two-dimensional problem and in addition, to the solution of other more realistic extensions of the model.

(1) Smallwood, R.D. "Detection Station Optimization - I"

Project CAMBRIDGE Memorandum No. 5, Detection Physics Laboratory, AFCRL, 9 July 1962

USE OF THE PDP-1 IN OPTICAL DESIGN

M. V. Morello, E. J. Radkowski, M. P. Rimmer and R. R. Shannon

Abstract

The design of an optical system consists, in a basic manner, of solving n different non-linear equations in m unknowns. Often n and m may be of the order of twenty or more. The problem is further complicated by engineering and physical constraints as well as the fact that no definitive statement can be made as to when the best solution has been found.

These problems will be discussed, and it will be shown how the PDP can serve as an ideal lens design computer. A demonstration of current programs will be given.

Introduction

The PDP-1 offers some unique and interesting possibilities for applications to optical design work. Perhaps the best way to emphasize this is to outline what is the basic advantage. The PDP permits an optical designer to observe in "real time" the details of the light distribution in an image. In essence, the machine communicates directly with the designer through the display. The designer can then react and make changes in the system via the typewriter, display or console. The aim here is not to use the machine as a gigantic sketch pad, but to usefully present a large amount of data in a usable manner.

The lens design problem may be described as that of solving n non-linear equations in m variables. It is not uncommon for n and m to be each of the order of 20 or 30. Furthermore, there are non-linear boundary conditions on each variable and no well-defined way of knowing when a solution has been reached. Of course all of the parameters involved have physical significance.

Now, how is the problem approached? A lens is set up as a starting point by deciding upon the focal length, aperture, field angle, spectral region, intended use and desired resolution. From widely respected basic rules a starting configuration is picked. Traditionally, this starting point has been empirical, based upon the experience of the designer. With a computer, this is still largely true. However, the intelligent use of a computer permits one to investigate a large number of possible solutions, and thus to develop a broad level of quasi-empirical experience within a short time. Thus the first major step in design is more or less mechanized. It has been repeatedly shown that there is no more sense in starting the design of a lens by randomly dropping glass blanks in a computer than there would be in designing a gas turbine by starting a program out on a pile of metal plates.

The Design Process

To proceed with the actual design process requires a short digression as to the model of an optical system which is used.

The basic geometrical model of a lens depends upon tracing a geometrical pencil of light, that is a ray, through the lens using Snell's law. The product of index of refraction and angle with respect to the normal of a surface is conserved at the surface. That is, $N \sin I = N' \sin I'$. Between surfaces, the rules of analytic geometry are used to trace ray paths and to find the angle with the normal. To trace rays is a repetitive process and therefore ideally suited to a computer. Each ray, however, provides only a limited amount of data concerning the lens. Many rays originating from several object points are required to fully assess the state of correction of the lens. The measure of this state of correction or of the aberration residual of a lens is obtained from the size of the patch over which the rays spread in the image plane. An exact picture of the geometrical characteristics of the lens may be obtained.

To satisfactorily and completely analyze a lens may require tracing 100 rays in each of three colors at three field points, or about 900 rays. For a sample lens of six surfaces with one aspheric some interesting numbers are generated for the time required for doing this. See Table 1 below.

Time Required to Plot and Analyze a Sample lens of six surfaces with one aspheric surface.

Calculation of Ray	By Human	By LGP	By PDP
Surface Spheric	3 to 10 min	2-13 sec	10-50 msec
Aspheric	1/2 to 1 hr		
One Ray	1 hr-avg.	23 sec	1/10 sec
900 Ray	900 hours or 100 days	5 hrs	90 sec
Plot and Analysis	2 days	2 days	1 sec

Table 1

Obviously such an approach becomes really practical only for a PDP type machine.

The time required for hand work is so absurd that an entire approach using approximate techniques has developed over the years. This consists of expanding the sine of the angle of incidence to successively greater orders of accuracy. First order, or Gaussian optics, leads to formulae for magnification and image position. Third order aberration theory gives the first aperture and field error terms which are separated into such aberrations as spherical, coma, astigmatism, distortion and curvature of field. These permit a rapid but approximate picture of the state of correction over the field to be obtained. Fifth and seventh order theory extends the accuracy somewhat.

Times again, are:

Calculation of One Surface	By Human	By LGP	By PDP
3^{o}	2 min	4-20 sec	1/10-1/2 sec
3+5	15 min		
Sample Problem	12-90 min	24 sec 2 min	.6-3 sec
Output + Analyze	5 min (Coeffs.)	30 sec (Coeffs.)	1 sec (Graphs)

Table 2

Obviously, an improvement. Using third order perhaps twenty to a hundred iterations may be required for a solution depending upon the complexity of the problem. Using third plus fifth, ten to fifty iterations. Needless to say, the final stages must be carried out using exact ray trace data. A minimum ray fan for each final step would be as follows, for 51 rays,

Human	2 hours
LGP	3 min
PDP	5 secs

Now, I believe you can see how the PDP can be a design partner. Economically, it is feasible to use the machine as such.

Where do we stand now? We have working and are now using, third order, ray trace, and fifth order programs. We are working on refinements of these programs with respect to improving display communication. At present we display the curves of greatest use to a designer. We will add new displays showing spot diagrams and the lens itself as time proceeds. We also plan to add a certain amount of semi-automatic solving as an aid and adjunct to the display through the light pen.

So far I have mentioned only geometrical optics. When the wave length of light is included, diffraction occurs and must be taken into account. We have one general system analysis or study program which computes the spatial frequency response. This gives a picture of the lens response complete enough to be of use in accurately computing resolution and amount of information passed by the lens. In this regard the PDP serves as a research as well as an analytic and engineering tool.

Our optics programs are extremely complex. The design package so far completed requires, with FLINT and lens data storage, about 6000 words. Obviously the disc display and low speed expanded memory are essential to complete integration of these programs. We require about 3000 to 3500 active core locations to work with. An exchange with the disc is obviously profitable. Our final package for design and analysis will probably use about 10,000 to 12,000 instructions by the end of this year.*

*APPENDIX (added March 1963)

The Optical Design Package has been integrated under a Master Executive routine which calls prestored programs from auxiliary disc storage.

Ray Trace Intercept Curves

Ray fans from on-axis, seven tenths and full field relative object heights are traced in the sagittal and tangential directions. The ray intercept curves in the image plane are plotted directly on the CRT and recursively displayed by the Itek display system. Figure 1 illustrates these curves plotted from different wavelengths.

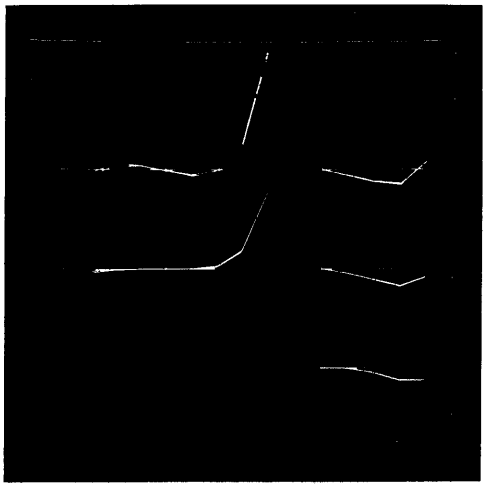


Figure 1

Lens Drawing

A scaled representation of the lens system is drawn on the scope from the lens parameters. Figure 2 is an illustration of a lens drawing.

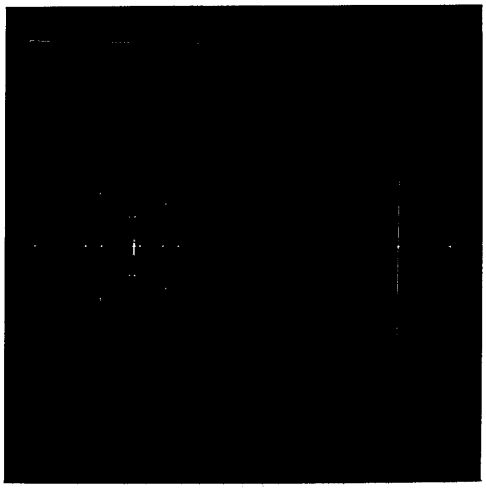


Figure 2

Spot Diagram

By tracing a grid of rays arranged in a hexagonal close packed pattern and displaying the intercept points in the image plane on the CRT, it is possible to obtain an excellent approximation to the shape and structure of the image.

The spot diagram shown in Figure 3 was calculated and displayed on the CRT for 511 rays in 150 seconds. A graph-plotter program reproduces the CRT display on an on-line graph-plotter so that the lens designer can take a hard copy image of the scope display with him.

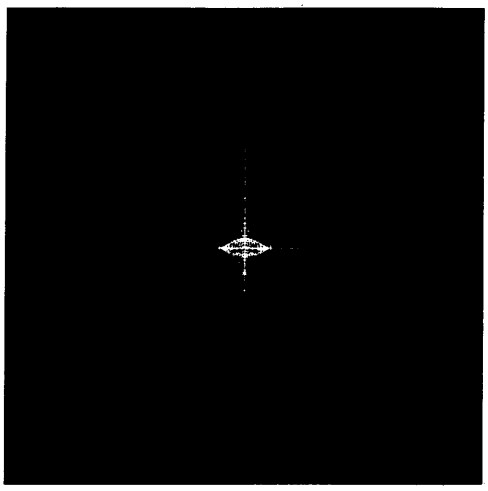


Figure 3

Lens Analysis Program

Response characteristics for various spatial frequencies are also calculated and displayed on the CRT. Figure 4 illustrates an example from this program.

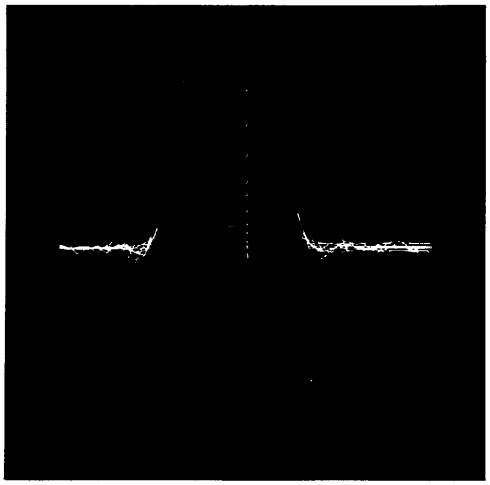


Figure 4

COMPUTER AIDED ANALYSIS OF MULTIVARIABLE SYSTEMS USING COLOR SCOPE

C. M. Walter

In the investigation of many complex multivariable statistical and dynamical systems it is often sufficient to obtain a feel for the general behavior of the system. In order to achieve this "feel," it is necessary to construct and to evaluate a variety of mathematical models. This is usually a laborious task for very complicated systems and the usual sequence of operations runs somewhat as follows: 1. A specific mathematical model is constructed from physical insight into the system. 2. Computational algorithms are prepared. 3. Computer programs written, often not in close concert with the orginator of the problem. These programs are almost always debugged in a messy fashion in a long series of bouts with an off-line processor. 4. Finally, a run is made and a large amount of tabulated data is placed in the hands of the problem originator. 5. Graphs must then be prepared in order to see what is going on. This whole operation is then carried out repeatedly, with the outcome at any given stage usually either unavailable, or only partially transcribed from relatively useless tables into intelligible graphical form, before a decision on the next sequence of off-line computer runs can be made.

One of the principal motivations behind the Experimental Dynamic Processor, DX-1, at AFCRL was to provide a research vehicle for studying the utility of closer man-machine communication in providing a better feel for the behavior of various surveillance and other sensor information processing systems.

Since most of the sensor attribute extraction techniques, and other signal filtering machines, involve an extensive matrix manipulation capability, considerable effort has been put into the development of a matrix handling package for the PDP-1 type machine. The preliminary version of this package has been described in a previous article by Carmine Caso of the Wolf Research and Development Corporation. A primary objective of this utility system is to facilitate on-line matrix manipulation. Also, in order to make effective use of the color scope as a graphical output medium, and as a mechaism for manual intervention for the purpose of rapid parameter and function alteration it has been necessary to underwriting the development of an elaborate system of display utility programs. This work will be reported on at a later date. Sufficient routines exist at the present time, however, to give the following cursory illustration of the potential of this sort of approach.

One of the most important processes underlying a wide variety of problems in many areas of dynamics, statistics and of signal analysis, is the eigenfunction, or stationary mode description process. Consequently we have begun to evolve a series of control procedures to simplify the study of this complex process.

The determination of all of the eigenvectors and eigenvalues associated with a large matrix can be prohibitively time-consuming, and is usually unnecessary, particularly when the given matrix is obtained from a sampling process, or is only an approximation to some complex interaction process.

A key problem, therefore, is to know when to terminate the iterative procedure which is attempting to determine the set of eigenvectors, $\{\vec{u}\}$, and eigenvalues, $\{\lambda\}$, of the matrix $\underline{A}$. i. e., the solutions of the implicit equation $\underline{A}\vec{u} = \lambda\vec{u}$. Another related problem is to obtain a feel for the degree of stability of the eigenvectors under small perturbations about their estimated positions. These are questions which are almost impossible to answer analytically, except in trivial situations, or in terms of very crude bounds.

Through the use of the color scope and light pencil on the DX-1 it is possible, (see figure 1) to exhibit a particular estimate for some eigenvector, $\vec{u}$, as a set of points, $(u_1, \ldots, u_n)$, in one color (blue), with the abscissa corresponding to the coordinate number and the ordinate indicating the coordinate value of the n-dimensional vector. Any variant of this vector, say v, can then be entered, via the light pencil, (see figure 2) in a different color, (green) and the resulting transformed vector W ($= \underline{A}\vec{v}$) is exhibited, by the computer after suitable normalization, in still another color, (red) to see how near it is to u. (See figure 3.)

Another capability of importance in studying the overall convergence problem, is to exhibit the successive iterations tending toward a given eigenvector in different intensities of one color, with the most intense hue corresponding to the most recent iteration for that vector. Tendencies of the system to oscillate or to exhibit other anomolies then stand out clearly in terms of the visible past history showing both the rate and nature of convergence.

These techniques provide a very effective and elegant means for studying the stability and convergence properties of selected eigenmodes of the system under investigation, both as a function of purturbations in the eigenvector component values and as a function of the particular analytical representation of the system. E.g., whether it is characterized in terms of a differential system of equations or in some equivalent integral equation formulation. Often the algorithmic description needed for purposes of computation will introduce spurious instabilities which are extermely difficult to evaluate analytically, and which have no "physical" counterpart in the actual system under investigation.

In resolving problems of the above sort, the capability for having the originator of the problem interact directly with the problem as it is running on the processor leads to very great savings in the overall time needed to get the requisite understanding of the behavior of the model under investigation.

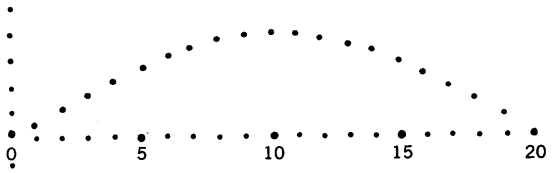


Figure 1. First Eigenmode of Vibrating String $\vec{u}$

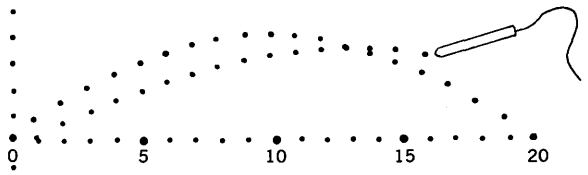


Figure 2. Variation on First Eigenmode $\vec{v}$

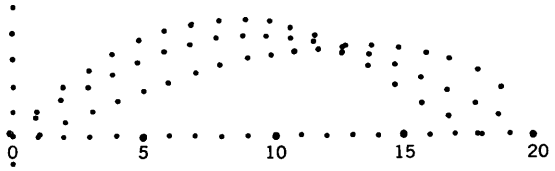


Figure 3. Response to Variation $\vec{w}$

A WORLD OCEANOGRAPHIC DATA DISPLAY SYSTEM

Edward Fredkin

Abstract

The purpose of the world oceanographic data display system is to enable a researcher to conduct an analysis of oceanographic data using a series of visual displays generated by a digital computer. The eventual goal of the program will be the design of a computer system capable of storing the entire body of world oceanographic data and making it available for visual display and analysis.

The researcher will be able to select specific data for display, in the form of contour lines against a map of a selected oceanic area. He will be able to vary the parameters of the data selected, and observe the resulting variations in the contour lines. It is anticipated that the world oceanographic data display system will open up the presently available body of oceanographic data to a rapid and meaningful analysis by oceanographic researchers.

Introduction

Oceanographic data has been collected over the course of many years. This data may, typically, take the form of readings (at a specific location) of temperature, degree of salinity, percentage of oxygen content, and density at varying depths. It is generally collected in the form of water samples taken during the course of oceanographic expeditions. Thus, at the present time, a vast amount of oceanographic data has been collected; the task of analyzing this data manually presents formidable difficulties.

Description of System

The World Oceanographic Data Display System will consist of a PDP-1 computer with a magnetic tape unit, a cathode ray tube visual display unit, a light pen, an input-output typewriter, and control switches. Oceanographic data will be stored on magnetic tape and will be processed and displayed by appropriate programs.

Method of Operation

In operation, a world map will appear on the CRT visual display unit of the computer. The researcher will then designate a specific geographical area (e.g., the South Atlantic) for enlargement on the visual display scope by pointing a light pen at the desired geographical area. He may then select a sub-area within the South Atlantic area for enlargement and display in the same manner. He may continue this process until the desired degree of enlargement has been obtained.

At this point the researcher, by typing an appropriate symbol on the input-output typewriter, will instruct the computer to transfer oceanographic data for the selected region from magnetic tape storage into the computer memory unit. It is anticipated that data to be used in the World Oceanographic Data Display System will include the latitude, longitude and identification number for each oceanographic station,[1] and for each recorded depth at the station, the following information: (See figure A) (1) temperature, (2) salinity, (3) oxygen percentage and (4) density, or specific volume anomaly. After data read-in has been accomplished, two additional features will appear on the visual display scope. First, super-imposed on the area map selected for visual display will appear a number of illuminated dots. Each dot will represent the location of a specific oceanographic data station. All oceanographic data stations in the selected area will be represented by these light dots. Second, five thermometer-type data indicator scales will appear in the lower righthand corner of the visual display scope. These will represent (1) depth, (2) temperature, (3) salinity, (4) oxygen percentage and (5) density. They may be used by the researcher to select modes of data display in following manner.

The researcher may indicate a specific depth on the depth indicator scale with his light pen.[2] He may also indicate a specific value for one other parameter on the appropriate indicator scale in the same way. For example, he may designate a depth of five hundred meters and a temperature of five degrees centigrade.

This will cause all stations reporting temperatures of five degrees or higher at a depth of five hundred meters to appear as bright dots on the visual display scope. Stations reporting temperatures lower than five degrees at a depth of five hundred meters will appear as dim dots on the visual display scope. In this way a "temperature gradient" will be outlined on the visual display scope in the form of a brightly lit (or, conversely, a dimly lit) area.

Combinations of Data for Analysis

The World Oceanographic Data Display System will make possible the analysis of data in the following combinations:

(1) depth vs. temperature

(2) depth vs. salinity

(3) depth vs. oxygen percentage

(4) density vs. depth

(5) density vs. temperature

(6) density vs. salinity

(7) density vs. oxygen percentage

In this way the researcher will have available a rapid and flexible method of determining the relationships which exist (a) within a given set of two parameters and/or (b) between several individual parameters.

Other Capabilities of the System

The researcher, in the conduct of his analysis, may note an anomalous station displayed on the scope. By pointing his light pen at this station, he will cause to be displayed on the scope the name of the cruise and other pertinent information. He can, if so desired, cause this information to be typed out by the computer for purposes of maintaining a permanent record. In this connection, the computer system will be able to accept and record comments (together with the name of the user) regarding the validity of recorded data; such information will be retained by the computer and made available to subsequent users upon request.

Summary

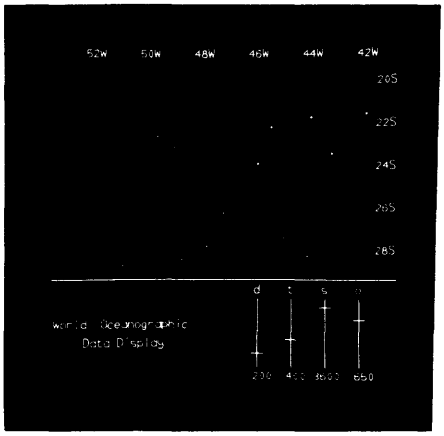
Access to the vast body of oceanographic data has been at best slow, difficult, and in many cases impractical under manual methods of data analysis. The World Oceanographic Data Display System would immediately open up the large body of data now available in computer-readable form to rapid and meaningful analysis by oceanographic researchers. In this sense it would serve as a technological stimulus which might lead, in turn, to rapid and extensive advances in the state of the art of world oceanography.

---

[1] Each location where oceanographic data has been collected (e.g., by use of an oceanographic current meter) is referred to as an oceanographic "station."

---

[2] Density may be similarly plotted against other parameters.



THE VORTEX OCEAN MODEL

Edward Fredkin

Abstract

The Vortex program is a computer program written for the PDP-1 computer. This program simulates a Vortex Ocean Model and displays the results of the simulation on the computer scope. In addition, the computer collects certain statistics on the model, and allows a researcher to control the configuration of the simulation by appropriate entries made on the input-output typewriter. At any time during the computation, the value of any of the variables in the computation may be examined or changed by using the typewriter. In addition, certain of the variables may be given a random assignment of values by pressing a single key on the typewriter.

In general this program allows a researcher to work with the Vortex Ocean Model so as to gain familiarity and understanding of the behavior of the model.

In the Vortex Ocean Model, a set of vortices interact in a circular or in an unbounded ocean. The number of vortices may be varied from one to thirty-one. In addition, there is a special test point which is used to collect statistics on the effects of the vortices. Each vortex rotates with a strength, where the i^{th} vortex rotates with strength S_1. When the direction of the rotation is counterclockwise, it is positive; clockwise rotation is negative. The rotation of each vortex affects the position of all other vortices. This effect is in direct proportion to the strength, and in inverse proportion to the distance.

This program performs all computations in fixed point, which is essential in order to gain the desired speed of computation; however, most input and output is handled by a floating point system. The program, not counting tables, is approximately 3600 (decimal) registers long. It operates with sufficient speed to show several vortices in operation, flicker-free.

An unusual feature is that the program automatically tests to see whether the computer is standard, has the optional high-speed multiply and divide, or has the Itek type of divide. In each case it modifies itself appropriately in order to take complete advantage of the capabilities of the computer.

The Vortex Control System

The Vortex program is controlled from the typewriter. At any time, during the operation of the program, various parameters may be examined or changed. In addition, the computation or various parts of it may be halted, continued, or restarted at will. In general the Vortex program is controlled by typing certain letters on the typewriter. These letters are called "control characters." When a control character is typed, the appropriate action is taken, and the program continues. Certain control characters, in particular those that are directly related to the value of some parameter, usually behave in the following fashion: when they are typed, the value of the corresponding parameter is printed. In order to change the value of a given parameter, the character slash [/]* is used. Slash always changes the value of the parameter last referred to. A parameter may be changed as ofter as desired without retyping its control character, just by reusing the slash[/].

Some variables are put into the program and printed out by the program in floating decimal, i.e., as a decimal number with an exponent. For example, the value of pi would be printed [+3.1416+0] meaning 3.1416×10^0. The number of feet in a mile would be typed out as follows [+5.2800+3] which is interpreted as 5.2800×10^3 or 5280. The number of seconds in a memory cycle, 5 microseconds, would be printed out as 5.0000-6. When numbers are being typed in, no such rigid rules need to be followed. The actual number typed may have a decimal point or the decimal point may be omitted as you like. If the decimal point is omitted the number is assumed to be an integer. This number

*Square brackets will be used to indicate the exact string of characters that are typed out or typed in.

is then multiplied by -10 raised to the power of the number immediately following the first number. The input of a floating decimal number is terminated by the first non-numeric, non-sign or decimal point encountered. Thus, pi may be put into the computer in any of the following formats. [3.14159], [31416-4a], [+3.14159+0], [+.00031416+4z].

A decimal integer is typed in as a string of digits preceded by a sign, where the plus sign is optional, and terminated by any non digit.

Following is a table of the control characters and their functions.

[t] The value of Δt, the time increment, is printed. The value of Δt may be changed by use of the slash. The value of Δt is typed in floating point format, $-2 < \Delta t < 2$.

[k] The value of the parameter k is printed. The value of k may be changed by use of the slash. The value of k is typed in floating point format, $-256 < k < 256$.

[c] The value of the parameter c is printed. The value of c may be changed by use of the slash. The value of c is typed in floating point format, $-256 < c < 256$.

[n] The value of the parameter n is printed. n is the number of vortices in the system, not counting the test point. The value of n may be changed by typing the slash. The value of n is typed as an integer, $1 <= n <= 31$.

[i] The value of the parameter i is printed. i is the current value of the index for subscripted quantities. Some of the parameters, namely x, y, and s, have a value for each of the n points and also for the test point. In order to examine these values for a given point, the parameter i is set to a small integer, $i < n$. Once the value of i is set, it remains at that value until it is changed. In order to examine or change the parameters of the test point, i is set to the value zero. The value of the index, i, may be changed by use of the slash. The value of i is typed in as an integer, $0 \leq i \leq 31$.

[x] The value of x_i is printed. The value of x_i may be changed by use of the slash. The value of i must have been set previously. The value of x_i is typed in floating point format, $-1 < x_i < 1$.

[y] The value of y_i is printed. In all respects the same as for x.

[s] The value of s_i is printed. In all respects the same as for x.

[p] The value of p is printed. p is a parameter that controls whether the vortices are affected by the boundary. p is a decimal integer that must be set to one of three different values, 0, -0, 1. If the value of p is 1, then the vortices will be affected by the boundary. This is done by computing image vortices and computing their effects on the normal vortices. If the value of p is 0, the computation will proceed as before, however, the vortices will be uneffected by the image vortices. If the value of p is -0, most of the computation for computing image vortices will be skipped. The only difference between the value 0 and -0 is one of timing; the total difference in computation time between the value 0 and 1 is negligible, while with value set to -0 the speed of the program is measurably increased.

[r] The control character r is a toggle switch. Every time [r] is typed, the program changes from one of two states to the other. The function of r is to freeze the computation, and to display additional data on the scope. If [r] is typed again the computation will continue, if r is then pressed again it will freeze, etc. When the computation is frozen by means of r, the values of the variables remain constant, e.g., there is no change in any of the x's, y's or strengths (s's), and the statistics remain constant. Various parameters may be examined and changed while the computation is frozen. In case of certain types of errors, after an error print indicating the type of error and where in the program the error occurred, the computation will be automatically frozen.

[a] The value of a, the radius of the circle that is used as the boundary in the image computation, is printed. The value of a is typed in floating point format, $-4 < a < 4$.

[o] In order to display the circular boundary, the letter o may be typed. o is a toggle switch as is r, the boundary appearing and disappearing with each subsequent type of the letter o. Displaying the boundary will noticeably slow down the computation and display.

[b] The control character b stands for beginning. Beginning sets a new set of random numbers in x_i, y_i and s_i. For all values of i other than i=o. Thus,

all points are assigned random positions and strengths except for the test point. The random numbers used range uniformly $-1/4 <$ random variable $< +1/4$. Thus the points are more or less grouped in the center with fairly small strengths. Beginning also resets to zero the statistics and cycle count.

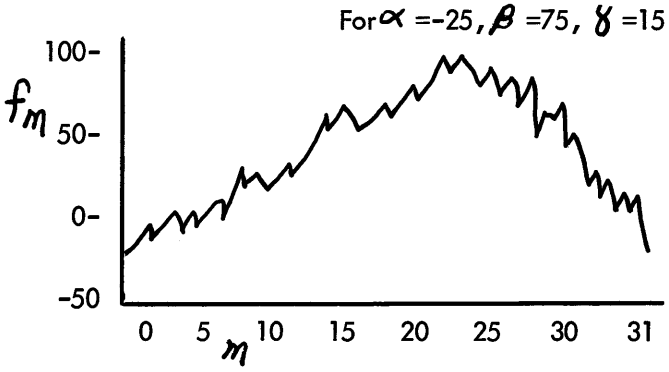
[v] The control character v restarts, the same as does the control character b, however, v will always restart with the same table of random numbers as did the previous b. Thus, once an interesting display results from typing the letter b, it may be repeated as often as desired by typing [v]. When restarting, the statistics and cycle are reset also.

[h] Typing [h] will cause the vortex program to type all the statistics. The statistics typed out will be (1) the total time, (2) the average value of dx for the test point, (3) the average value of dy for the test point, (4) the variance of dx for the test point $\overline{(dx-\overline{dx})^2}$, (5) the variance of dy $\overline{(dy-\overline{dy})^2}$, and (6) $\overline{(dx-\overline{dx})} \times \overline{(dy-\overline{dy})}$.

[d] The display routine. In order to allow the insertion of a title, which will be displayed on the scope whenever the computation is frozen, type [d] followed by a title. When typing the title, the choice of characters is unlimited, except that tabs and carriage returns should not be used, and the title should be no more than 50 spaces long. Although the message must not exceed 50 spaces in length it may actually contain more characters because of the upper and lower case characters, and non-space characters. When you have finished typing in the title, then move sense switch one up and down to indicate so to the program.

[m] Restart statistics from zero. If it is desired to start the statistics at zero this may be accomplished by typing the letter m. This will set the number of cycles, the time, and all statistics to initial values.

[z] The control character z is used to set the f function table. This allows a fairly automatic setting of all the values of the f function table. After typing [z] the program will type out the word [base], at this point a floating decimal number should be typed in. Next the program will type out the word [top] which should be followed by typing in another floating decimal number then the program will type out [+ or -/] which should again be followed by a floating decimal number. All these numbers affect the value of the entries in the f table. Let us assume that three numbers typed in are represented by α, β, and γ respectively, then the allowable range of these numbers would be determined as follows, $-256 < \alpha - \gamma, \beta + \gamma < 256$. The f table consists of 32 different values the entire table is filled after the typing of the three numbers. γ, which is typed after the type out [+ or -/], is the range of a uniformly distributed random variable that is added to each value put into the table. This value may be set to zero. For the 32 entries in the f table, (f_0 through f_{31}) f and f_{31} are set to the value typed after [base], α. Then the values of the entries in the f table are increased linerly until at f_{16}, they are equal to β at which point they are decreased linearly until at f_{31} they are equal to α again. After this is done, the random variables are added to this trangular like array.



[f] The control character f allows an entire table to be typed in, all 32 values, or just 1 value. Once the letter f is typed, the program will type out [f-table, a or 1?]. In order to change all of the f table, type [a]. To change any given member [1], the digit one.

When Changing One Value of f_m

When typing in just one value, the program will print [m=], this indicates that a value for m should be typed in, m is an integer, $0 \leq m \leq 31$. The value of f_m will be printed. The value of f_m may be changed by use of the slash. f_m is typed in floating point format, $-256 \leq f_m \leq 256$.

When Changing All of The f-table

The value of the subscript m ($0 \leq m \leq 31$) will be printed, then type in a value for f_m. The value of the subscript m will be increased by one and printed, etc. f_m is typed in floating point format, $-256 < f_m < 256$.

The Error System

The Vortex program, has a general and fairly elaborate error detection system. The types of errors detected are those caused by overflow. In order to speed the computation, all internal calculations are done in fixed point. This means that for each variable, the smallest change in value is equal to approximately 1/250,000 times the range of that variable. Should the result of any computation be a number outside the range of the variable that is being computed, then an overflow condition exists and an error print will occur. The error print is of the form the letters [err] in red followed by a two letter code, also in red, followed by the octal location of the offending instruction. At this point the computation freezes and various registers may be examined or changed or the computation restarted by means of the typewriter control features.

Facts About The Vortex Program

The Vortex program is coded in the DECAL language. The format used is almost entirely the instruction word statement, as this program needs to operate in real time and needs to be very fast. Features of DECAL were used to make this program compatible between machines that use multiply and divide as built-in instructions and those that use subroutines. The Vortex program itself consists of approximately 1,400 (1,400 decimal) instructions. Room for up to 32 vortices is contained in a table 352 registers long. Various subroutines and their tables used by the vortex system add an additional 2,200 registers to the program, for a total of about 4,000 registers (the memory contains 4,096 registers). The major subroutines used are the floating point system which is used for input and output and certain computations, the in-out subroutines, and the display subroutines.

SPACEWAR!
REAL-TIME CAPABILITY OF THE PDP-1

J. M. Graetz

Abstract

The game starts with each player in control of a spaceship (shown on PDP scope) equipped with propulsion rockets, rotation gyros, and space torpedos. The use of switches to control apparent motion of displayed objects amply demonstrates the real-time capabilities of the PDP-1.

Introduction

The demonstration program known as SPACEWAR! was first conceived in December, 1961 at an informal gathering of the Hingham Institute where Wayne Wiitanen, Stephen Russell, and the author were discussing some of the possibilities of the use of the large-screen CRT which was to be attached to the new PDP-1 computer at M.I.T. One idea that caught our fancy was the thought of a moving display under the control of the user. We thought that a simulation of ships in space would provide an excellent demonstration and the discussion developed into the Hingham Institute Study Group on Space Warfare, under whose auspices almost all of the work described here was done. The main control and computation programs were written and debugged in the first months of 1962 by Stephen R. Russell of Harvard.

The program is set up in the form of a game for two persons and a PDP-1. Each person has control over one of two displayed spaceship outlines. The object of the game is to destroy the opponent's ship by blasting him out of space with torpedos. Control is maintained over the ship's orientation by simulating rotational gyroscopes. All translation is achieved with the ship's main drive rocket; the ship will accelerate in the direction its nose is pointing as long as the rocket engines are turned on. Both ships are armed with ballistic missiles (torpedos) which are released from the nose of the ship with a velocity equal to the ship's velocity plus that imparted to the missile by the launcher. From then on, the torpedos are in true ballistic flight. Each ship has one other means of getting from one place to another, namely "hyperspace," which allows him to get out of the way quickly.

The display includes a background of stars and a bright, flickering star or "heavy star" in the center of the scope which maintains a rather fierce gravitational field.

The Game

At the beginning of the game two spaceships, equipped with 31 torpedos, are displayed in diagonally opposite quadrants of the scope face. The players operate switches for the purpose of maneuvering into position for joining the fray. (It is unwise to remain in a single position for a very long time, and also fruitless, for the torpedos have only a limited range.) The torpedos have two types of fuze: one is a proximity fuze which causes the torpedo to explode when it comes within a certain critical distance of any other collidable object which will also be caused to explode. The other is a time fuze which causes the torpedo itself to explode if it has not encountered another object after a given length of time.

The "heavy star" in the center is constantly exerting a strong gravitational influence on the two spaceships (torpedos are not affected by gravity). This star also has a very short capture radius; a ship with reasonably large intrinsic velocity can come in quite close to the star without fear of being captured. This maneuver is frequently used to change direction rapidly.

If a ship is captured by this star, it loses all velocity and is thrust into the "anti-point," that point on the surface of a topologically toroidal scope which is represented by the four corners of the face.

All collidable objects explode on coming into critical range. The current rules require that a game is won only if the remaining ship (after the opponent has exploded) can successfully avoid being blown up by any torpedos which may be left over. A tie is declared: when both ships collide (and explode); when an apparent victor is destroyed by a loose torpedo; or when both ships run out of torpedos. (Each ship has 31 torpedos at the start of each game).

The Spaceships

The two ships have different outlines making them more easily distinguishable on the scope face. Rotation is readily apparent and rocket blast is equally detectable. When the ship is blasting, a fiery tail is seen at the base of the ship, where the main rocket exhaust is placed. The spaceship outlines are generated and displayed by a program written by Daniel Edwards of M.I.T. This program provides a very fast and reasonably flicker-free display. Torpedos appear as single moving dots. They resemble stars rather closely.

The Heavy Star

A bright, flickering point in the center of the scope represents the massive star referred to as the "heavy star." This star has a strong effect (which approximates gravitation) on the two spaceships. The program for this was also written by Daniel Edwards. In the final version of SPACEWAR! he is going to provide an improved integration to eliminate some of the more unexpected, albeit interesting, properties of the "heavy star."

The Stars of the Heavens

To add verisimilitude to the display, a background of stars is provided. At first, this was merely a random display of dots. However, Peter Samson of M.I.T. has written a program which displays a star map of the sky as seen from the Earth's equator. The size of the scope limits the extent of the map to a 45° segment of the heavens. Stars down to just above fifth magnitude are displayed. The display moves imperceptibly across the face of the scope from left to right, and, given time, the complete band of stars of this section of the map will be displayed.

Hyperspace

This is an emergency device. It frequently happens that a ship cannot accelerate fast enough to get out of the way of an approaching torpedo. The player may send the ship into hyperspace then. The ship then will disappear and very shortly will reappear somewhere else on the scope. Since this is a way of getting from one place to another without traveling the distance between, the method used must be hyperspace! Each player has exactly three hyperspace jumps.

On most PDP-1s, the ships are controlled by switches in the Test Word. For the M.I.T. machine, however, two control consoles were devised by Robert A. Saunders and Alan Kotok, both of M.I.T. Each console has a double-throw switch to control rotation, a firing button, and a blast lever. Hyperspace is entered by pushing the blast lever forward and releasing.

Acknowledgement

Special thanks from the Hingham Institute are extended to: the various members of the Tech Model Railroad Club for help and encouragement; to Prof. Jack B. Dennis, director of the M.I.T. TX-0 and PDP-1 installations, whose assistance went beyond the generous allowance of time on the computer; and, to Digital Equipment Corporation without whose gift SPACEWAR! would still be wishful thinking at the Hingham Institute.

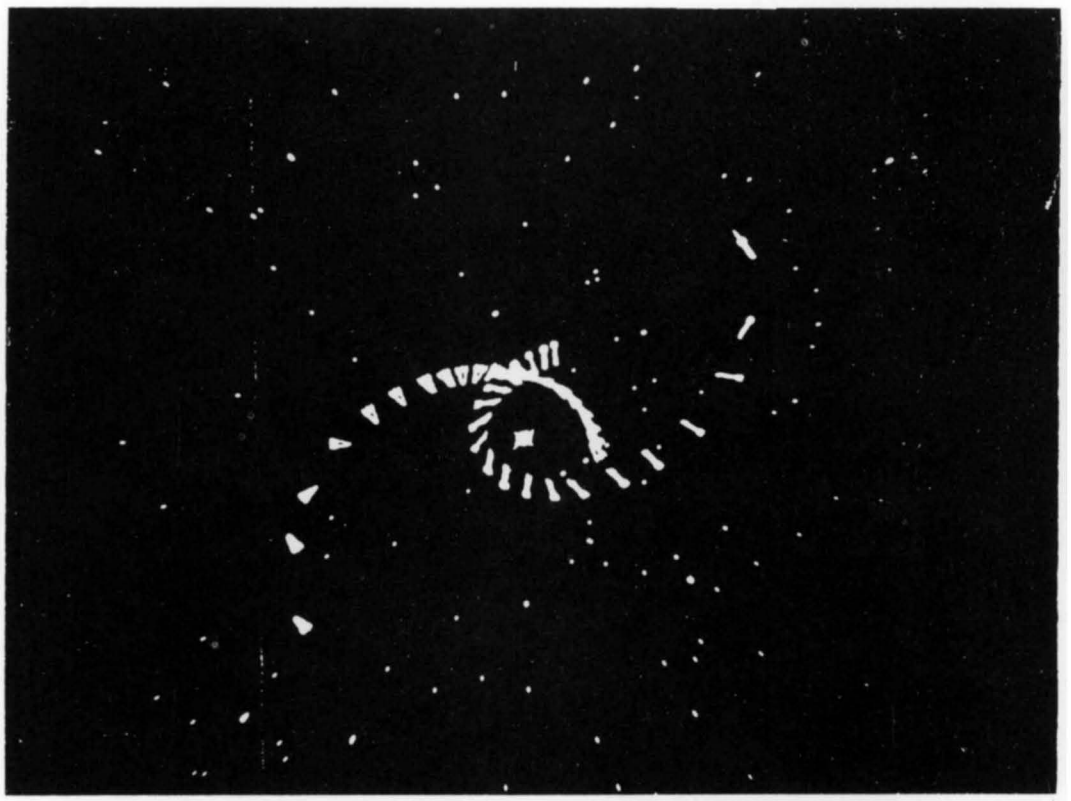


Figure 1 A Common Opening Maneuver

ON-LINE PROCESSOR-ORIENTED INVESTIGATION OF A CLASS OF DYNAMIC ATTRIBUTE EXTRACTION AND CLASSIFICATION PROCESSES

C. M. Walter

Abstract

A method for the comparative evaluation of a wide class of self-adaptive attribute extraction and classification procedures is described. This method involves the on-line use of a medium size digital computer coupled to a colored CRT display. Multichannel sensor information, or numerical test data, to be abstracted and classified, can be entered into the system from paper tape, or through an A-D converter, or manually, by use of light-pencil, CRT and keyboard. Both raw sensor data, its representation in the form of abstracted attributes, and its reconstruction from incomplete sets of attributes can be viewed simultaneously on the CRT output. On-line keyboard and light-pencil control provide an excellent mechanism for investigating the structure of the various complex transformation processes which are involved in this approach to the consolidation and interpretation of sensor information.

1. Introduction

The development of increasingly more elaborate sensors and measuring devices for obtaining empirical information about the environment, has led to a continuing crisis in the information processing domain. Improvements in radar, radio, and seismic detection devices, together with the advent of a variety of energy detecting satellites, is creating growing mountains of unprocessed data. The potential of this data, for purpose of military surveillance, and also as a source of scientific knowledge, often diminishes rapidly with its age in a raw, unprocessed state.

Unfortunately, the cost of directly operating on this sensor data with the present generation of digital "data processors" (actually logical symbol manipulating devices) is proving to be too costly. Much of the current effort in pattern recognition and on the application of artificial intelligence to this domain, seems to be addressed primarily to the symbol manipulating aspects of the problem. The more fundamental matter of extracting appropriate sets of intrinsic attributes, together with both the syntactic and semantic rules governing their behavior, has only been treated by a few groups.

In large measure, the difficulty arises because the canonical attributes, which behave according to reasonably simple logical or mathematical rules, have to be obtained from the raw sensor data by a process of statistical and dynamical abstraction. These attributes are usually not the measured outputs of the various sensors, and are generally related to these raw measurements in a complex multivariate statistical-and dynamical-manner, which depends upon both the process under observation and also the measurement process itself.

Thus, the primary problem is not so much one of symbol manipulation according to specified rules, as that of determining what the rules should be, and of establishing meaningful interpretations for the symbols. Under this viewpoint, the problem of effective dynamic processor design is inseparably tied to the general measurement problem.

It is, therefore, only reasonable that we should investigate in detail various important classes of dynamic attribute extraction procedures, together with possible mechanisms for efficiently realizing these procedures. A step in this direction is the on-line simulation of various classes of these sensors attribute extraction procedures, using an orthodox digital data processor, to which suitable graphic input-output devices have been added.[1] The slowness and inefficiency of this type of simulation, on sequential processors, clearly calls for the development of a parallel processing capability which at least covers the manipulation of the linear associative matric algebras needed to characterize the basic measurement and attribute extraction process.

[1] Walter, C. M., "The Experimented Dynamic Processor DX-1," Part 9, 1962 IRE International Convention Record, March, 1962.

Since appropriate physical realizations must be provided for the matrix elements, in terms of the non-canonical coordinates associated with the raw sensor measurement processes, this is not an easy task. However, the use of electro-optical storage mechanisms and parallel processing techniques should materially aid in these realizations.

2. The General Problem

The emphasis in this report will be on the development of on-line data processing techniques for evaluating a class of dynamic attribute extraction and classification procedures. The particular class of procedures to be implemented is based on a synthesis of methods from multivariate analysis, statistical mechanics and the quantum theory of measurement processes, as applied to the non-atomic domain, and is discussed in detail elsewhere.[2]

The categories of data to which these techniques are most relevant are those in which the important attributes are not clearly discernible from direct examination of the raw measurement data. This is particularly true of target signature identification problems in the surveillance area, whether the basic mechanism be radar, I.R., radio forward-scatter or back-scatter, or sonar. It is also true of bioelectric signals such as EEG and ECG data. In all of these examples the significant, structural attributes are highly interrelated, in terms of the raw sensor data, in both a statistical and a dynamical manner. Any process which attempts to unscramble this data must seek the natural invariants of the system and determine a suitable basis for the exhibition of these invariant attributes.

3. Approach

The attribute extraction processes under investigation may be resolved into six interrelated phases: 1) raw sensor data description; 2) intrinsic attribute derivation; 3) interpreted data description in terms of intrinsic attributes; 4) truncation of interpreted data; 5) resynthesized sensor data from truncated interpreted data; 6) comparison of raw and resynthesized data.

3.1 The first phase is concerned with the consolidation of the raw sensor data, x, into a family of descriptors, δ, from which an interaction function, F is formed, along the lines delineated in Appendix A, Section 1.1 (A-1.1). The tabular description of raw sensor input data is outlined in A-4.3. A similar format of colored points is used to exhibit the x data on the color CRT attached to the DX-1 System. Means are provided for inputing the data, either through the scope under light pencil control, or from mag tape, with light pencil editing. In the scope display the t-coordinate is horizontal, rather than vertical, as in the printout.

3.2 The second phase involves the construction of a transformation, u, from the sensor data coordinate basis, to a new basis of intrinsic attributes, in terms of which the interaction function possesses the simplest structure in the sense of least interaction between isolated attributes. The specific mechanism chosen to achieve this goal is the eigenfunction procedure specified in A-1.2. The tabular description of the eigenvector attribute filters is discussed in A-4.2.

3.3 Then, using the eigenbasis determining procedure, it is possible to provide an interpreted data presentation, φ, in which the intrinsic attributes characterizing the process under observation are ordered in such a manner that the least significant attributes can be most readily neglected. This is particularly important for purposes of information transmission and storage. Procedures for constructing this interpreted information are discussed in A-1.3, and the tabular mode of description is presented in A-4.4. The associated graphic CRT display is arranged with t-coordinate horizontal and aligned directly under the associated x data on the CRT for viewing convenience.

3.4 At this point the interpreted data is arranged in a format (cf. A-4.4) which permits three kinds of controlled information degradation. We can eliminate interpreted data in all channels whose associated eigenvalues are less than a given value, as outlined in A-1.4. Another information reduction process which can be investigated at this point, but which has not been explicitly implemented in the present program, is to quantize the range of values of the φ function in a progressively coarser manner. The third mechanism for controlled information reduction, is to select $\varphi_n(t)$ for values of t, modulo some number τ, where $1 < \tau \leq N_0$. This is possible because of the interpolation feature of the resynthesis transformation from φ to x, as indicated in section 3.5 below.

[2] Huggins, W. H., et al., "Representation and Analysis of Signals," Department of Electrical Engineering Monographs, Parts I to X, The John Hopkins University, 1957-1962.

3.5 The next phase of the attribute extraction process treats the problem of resynthesis of an approximation to the original sensor data. In problems of information transmission, the effectiveness of the system is often measured by the fidelity with which this resynthesis can be achieved from the smallest amount of intermediate, interpreted (i.e., suitably encoded) data. The resynthesis process is described in A-1.5 and is greatly simplified by the unitary property of the eigenbasis transformation.

Basically, the inverse transformation, from φ to δ, and hence to x, is the transpose of the transformation from δ to φ. Moreover, for a given value of t, each $\varphi_n(t)$ is formed by looking ahead in the x data up to N_0 units along t, hence in the resynthesis process, the estimate $\hat{\varphi}'_n(t)$ can be used to estimate a new x' through a process of interpolation and extrapolation. Several tabular modes of description are outlined in A-4.6 and 4.7 and follow a format similar to that used for the raw x data, except that the intensity level of the resynthesized data points is now a measure of the confidence that the resynthesized data points belong to the signal pattern. The use of multi-level intensity, under program control on the color CRT, has proved highly effective in providing a "feel" for the manner in which the resynthesis process is altered as the "bandwidth" of the interpreted, intermediate data, is compressed.

3.6 Finally, the consequences of carrying out the various kinds of controlled information degradation cited in section 3.4 above can be examined through an on-line comparison of the raw sensor data, x, with the resynthesized approximation, x' . The initial efforts in this phase have been concerned primarily with means for conveniently obtaining a direct visual comparison between x and x' data. However, only problems of computation speed limit the extent to which a variety of analytical error criteria can be examined and compared.

APPENDIX A

FUNCTIONAL DESCRIPTION OF A CLASS OF DYNAMIC ATTRIBUTE EXTRACTION PROCEDURES

1. Functions to be Evaluated

1.0 The Basic Measurement Descriptor

Let $\delta_a(t)$ be the basic measurement descriptor associated with the stored measurement data $x_\alpha(t + \alpha_0)$, defined, for $a = (x, \alpha, \alpha_0)$, and t, specified over suitable domains as

$$\delta_a(t) = \begin{cases} 0 & \text{if } x \neq x_\alpha(t + \alpha_0), \\ C & \text{if } x = x_\alpha(t + \alpha_0), \end{cases}$$

and $$C = 1 / \sqrt{N_0 N_1}$$

where $$\sum_{a=1}^{N} \delta_a^2(t) = 1$$

for each t.

We shall also write

$$\delta_a(t) = \delta(x, \alpha, \alpha_0; t),$$

and, for convenience, interpret the ordered triplet $(x, \alpha, \alpha_0) = a$ as the integer,

$$a = N_0 N_1 X + N_1 (\alpha_0 - 1) + \alpha .$$

where

$$x,\ y = 0,\ 1,$$
$$\alpha,\ \beta = 1, \ldots, N,$$
$$a_0, \beta_0 = 1, \ldots, N_0,$$
$$N = 2 N_0 N_1,$$
$$n = 1, \ldots, N,$$
$$a,\ b = 1, \ldots, N.$$

1.1 Weighted Bivariate Interaction Matrix

Let

$$F_{ab} = < \delta_a (t)\ \delta_b (t) >,$$

where

$$< \delta_a (t)\ \delta_b (t) > = \frac{K (a_0, \beta_0)}{N(S)} \sum \delta_a (t)\ \delta_b (t).$$

Here S in the set of values of t over which the finite averaging operation is carried out, and N(S) is the number of elements in S.

We shall require that

$$F_{ab} = F_{ba},$$

and that

$$\sum_{a=1}^{N} F_{aa} = 1.$$

Hence a constraint on the function K is that

$$\sum_{x=0}^{1} \sum_{a=1}^{N_1} \sum_{a_0=1}^{N_0} K (a_0, a_0) = 2 N_0 N_1 = N$$

1.1.1 Unweighted Case

(Relative interaction is a bilinear function of δ-buffer is independent of α_0, β_0 components of positions.)

$$K (a_0, \beta_0) = 1$$

1.1.2 First Weighted Case

(Relative interaction in bilinear function of α_0, β_0 components of positions.)

$$K (a_0, \beta_0) = \begin{cases} \dfrac{6 \beta_0 (N_0 - a_0 + 1)}{(N_0 + 1)(N_0 + 2)} & a_0 \leqq \beta_0 \\ K (\beta_0, a_0) & a_0 \geqq \beta_0 \end{cases}$$

1.1.3 Second Weighted Case

(Relative interaction is inversely proportional to number of positions which a signal of length $|\beta_0 - \alpha_0|$ can occupy in δ -buffer.)

$$K(\alpha_0, \beta_0) = \frac{1}{2N_1 (N_0 - |\beta_0 - \alpha_0|)} .$$

1.2 Eigenfunctions, $u_n(a)$, and Eigenvalues, λ_n, of F_{ab}

Let

$$\sum_{b=1}^{N} F_{ab}\, u_n(b) = \lambda_n\, u_n(a) ,$$

where the values λ are ordered so that

$$\lambda_n \geqq \lambda_{n+1}$$

for all n.

Let the u's be normalized so that

$$\sum_{a=1}^{N} u_n(a)\, u_m(a) = \delta_{nm} ,$$

for all n, m, and

$$\sum_{n=1}^{N} u_n(a)\, u_n(b) = \quad ab ,$$

for all a, b.

Thus, u will be a unitary tranformation.

Note that

$$\lambda_n = \sum_{a=1}^{N} \sum_{b=1}^{N} F_{ab}\, u_n(a)\, u_n(b) \geqq 0,$$

for each n, and that

$$\sum_{n=1}^{N} \lambda_n = \sum_{a=1}^{N} F_{aa} = 1;$$

hence, for each n

$$0 \leqq \lambda_n \leqq 1 .$$

1.3 Transformation to Interpreted Descriptor, $\phi_n(t)$

Let $\phi_n(t)$ be related to $\delta_a(t)$ by the transformation

$$\phi_n(t) = \sum_{a=1}^{N} u_n(a)\, \delta_a(t) .$$

Note that

$$\phi_n^2(t) = \sum_{a=1}^{N} \sum_{b=1}^{N} u_n(a)\, u_n(b)\, \delta_a(t)\, \delta_b(t) ,$$

hence

$$\sum_{n=1}^{N} \phi_n^2(t) = \sum_{a=1}^{N} \delta_a^2(t) = 1 ,$$

for all t.

Also, we may note that

$$< \phi_n^2(t) > = \sum_{a=1}^{N} \sum_{b=1}^{N} < \delta_a(t)\, \delta_b(t) > u_n(a)\, u_n(b) ,$$

$$= \sum_{a=1}^{N} \sum_{b=1}^{N} F_{ab}\, u_n(a)\, u_n(b) = \lambda_n .$$

The inverse transformation, since $u_n(a)$ is unitary, will be

$$\hat{\delta}_a(t) = \sum_{n=1}^{N} u_n(a)\, \phi_n(t) \tilde{=} \delta_a(t)$$

where $\hat{\delta}_a(t)$, as given by the summation, will, in general, only be an approximation to the descriptor $\delta_a(t)$, which must, by definition, be 0 or C for each a and t.

Hence, for pusposes of estimating δ in the presence of errors, we may take it to be C if $\hat{\delta} \geq 0.5$ C and 0 otherwise. A final check on the validity of the estimation will be the requirement that

$$CK = \sum_{a} \delta_a^2(t) = 1 ,$$

for all t.

1.4 Reinterpretation of Data Descriptors, $\phi_n(t)$, by Partial Elimination and Reconstruction

Let the vector $\phi_n(t)$ be subjected to a transformation to new data $\phi_n'(t)$, where

$$\phi_n'(t) \begin{cases} \phi_n(t) \text{ for } 1 \leq n \leq m, \\ z_n(t) \text{ for } m < n \leq N. \end{cases}$$

Here the synthesized data, $z_n(t)$, may either be deterministic or nondeterministic data,

subject to the constraint $| z_n (t) | \leq 1$ for all n, t.

1.5 Resynthesis of Basic Measurement Data from Reinterpreted Data

Let

$$\hat{\delta}'_a (t) = \sum_{n=1}^{N} u_n (a) \; \phi'_n (t),$$

where, by definition, we take the resynthesized raw data $x'_\alpha (t)$ to be

$$x'_\alpha (t + \alpha_0) = x \text{ if } \hat{\delta}' \geq 0.5 \; C.$$

This gives a prediction for up to N_0 units beyond t.

Note that $\hat{\delta}$, suitably quantized, may be taken as the measure of confidence we have in the decision:

$$x'_\alpha (t + \alpha_0) = x,$$

for a given α, α_0, t.

For resynthesis based on data always measured relative to the center of the storage $\frac{N_0 + 1}{2}$ along the time-like storage coordinate a_0,

$$\delta' \left(x_j \; \alpha_j \; \frac{N_0 + 1}{2} ; \; t \right) = \sum_{n=1}^{N} u_n \left(x_j \; \alpha_j \; \frac{N_0 + 1}{2} \right) \phi'_n (t).$$

2. External Parameters and Functions

2.1 Input Data, $x_a (t)$

Binary valued range, $\{0, 1\}$, for x;

Integer valued domain, $\{ 1, \ldots, T + N_0 \}$, for t;

Integer valued parameter set, $\{ 1, \ldots, N_1 \}$, for α.

2.2 Synthesized Data, $z_n (t)$

Continuous range of values in closed interval, $[-1, 1]$, for z;

Integer valued domain, $\{ 1, \ldots, T + N_0 \}$, for t;

Integer valued parameters set, $\{m + 1, \ldots, N\}$, for n;

Values of $z_n (t)$ will be specified in both a deterministic manner and also statistically.

3.0 Internal Parameters and Functions

3.1 Length of Transient Data Storage, N_0, where N_0 is odd. (Required for center of storage to be at an integer value $\frac{N_0 + 1}{2}$), for parameter α_0.

3.2 Total Size of Transient Data Storage $N = 2 N_0 N_1$.

3.3 Span of Transformed Data, N.

3.4 Restricted Span of Transformed Data, m.

3.5 Matrix for Eigenfunction Transformation, F_{ab}.

3.6 Interpreted Descriptor, $\Phi_n(t)$

Continuous range of values in interval [-1, 1];

Integer valued domain { 1, ..., T } for t;

Integer valued domain { 1, ..., N } for n.

4.0 Format

4.1 Matrix Fab

x = 0					x = 1					$a_0 = 1, \beta_0 = 1$
α/β	1	2	...	N_1	α/β	1	2	...	N_1	
1	—	—		—	1	—	—		—	
2	—	—		—	2	—	—		—	
.										
.										
.										
N_1	—	—		—	N_1	—	—		—	
...										
...										

x = 0					x = 1					$a_0 = N_0, \beta_0 = N_0$
α/β	1	2	...	N_1	α/β	1	2	...	N_1	
1	—	—		—	1	—	—		—	
2	—	—		—	2	—	—		—	
.										
.										
.										
N_1	—	—		—	N_1	—	—		—	

4.2 Eigenfunctions and Eigenvalues

u_1 $\lambda_1 = \_\_$

$x = 0$					$x = 1$				
α_0/α	1	2	...	N_1	α_0/α	1	2	...	N_1
1	—	—		—	1	—	—		—
2	—	—		—	2	—	—		—
.									
.									
.									
N_0	—	—		—	N_0	—	—		—
...									
...									

u_N $\lambda_N = \_\_$

$x = 0$					$x = 1$				
α_0/α	1	2	...	N_1	α_0/α	1	2	...	N_1
1	—	—		—	1	—	—		—
2	—	—		—	2	—	—		—
.									
.									
.									
N_0	—	—		—	N_0	—	—		—

4.3 Raw Data, $x_\alpha(t)$

t/α	1	2	...	N_1
1	x	x		x
2	x	x		x
.				
.				
.				
T	x	x		x

NOTE:

Value of x	Symbol to be Recorded
0	Blank
1	Asterisk

4.4 Interpreted Data, $\phi_n(t)$

t/n	1	2	...	N
1	—	—		—
2	—	—		—
.				
.				
.				
T	—	—		—

NOTE: Data is specified to one decimal digit plus sign.

4.5 Reinterpreted Data, $\phi'_n(t)$

Same as in 4.4)

4.6 Resynthesized Characteristic Function, $\hat{\delta}'\left(x, \alpha, \frac{N_0 + 1}{2}; t\right)$

Essentially same format as in 4.3 except for type of symbols)

t/a	1	2	...	N_1
1	—	—	—	—
2	—	—	—	—
.				
.				
.				
T	—	—	—	—

NOTE:

Value of $\hat{\delta}$	Symbol to be Recorded
$\hat{\delta} \leq 0.25\ C$	Blank
$0.25\ C < \hat{\delta} \leq 0.5\ C$	Minus Sign
$0.5\ C < \hat{\delta} \leq 0.75\ C$	Plus Sign
$0.75\ C < \hat{\delta}$	Asterisk

4.7 Resynthesized Characteristic Function, $\hat{\delta}'(x, \alpha, \alpha_0; t)$

α_0/α	1	2	...	N_1		
1	—	—		—	$t = K$	$CK = ——$
2	—	—		—		
.						
.						
.						
N_0	—	—		—		
1	—	—		—	$t = N_0 + K$	$CK = ——$
2	—	—		—		
.						
.						
.						
N_0	—	—		—		
...						
...						
1	—	—		—	$T = J N_0 + K$	$CK = ——$
2	—	—		—		
.						
.						
.						
N_0	—	—		—		

where J, K are positive integers such that

$J N_0 + K \leq T$.

Section III

HARDWARE AND INPUT-OUTPUT TECHNIQUES

FILM READING USING A COMPUTER

A. M. Cappelletti

Abstract

The advantages of using movie film as a medium for recording and storing data, which might be produced by a space vehicle, or by a wind or current measuring device, are quite impressive, particularly in view of the limited storage space and input power required. The main problem is devising a means to "read" the data after it has been recorded. Because of certain inevitable fluctuations of the film, simple mechanical and electrical devices used for this purpose were found to be unsatisfactory. In order to enlist the aid of a computer, it was necessary to develop a suitably sophisticated light-sensitive input device, optimal logic systems and appropriate programs.

Introduction

Scientists over the past 50 years or so have often found it advantageous to use 16 mm and 35 mm film as a medium of data storage. Until recently, however, the only methods of retrieving this data involved manual procedures and the human eye. Many and varied mechanical electrical devices have been designed for this task, but in general, they were proved not to be adequate. The reasons were obvious to those who tried to make such devices and it is generally agreed that a machine with the sophisticated logical capabilities of a digital computer would be required for retrieving data stored on film.

Binary Film

Data film can be categorized as binary film or analog film. Binary type film may be depicted by first describing an instrument that produces such data. The Richardson Oceanographic Transducer, otherwise known as a current meter, is a cylindrical shaped instrument about three feet high and eight inches in diameter made of very strong cast aluminum built to withstand the immense pressures of the ocean depths. On top of this instrument is a vane which functions much the same as a weather vane. Within the instrument the direction of this vane is converted across the film into fourteen gray binary bit positions, called channels. On the bottom of the current meter there is what is called a Savonios rotor which functions like a wind anemometer that measures the speed of the current. This measurement is recorded as pulses in one or both of two channel positions. There is also a channel position for a clock pulse, and another for a continuous reference line which is always "on." Thus, across the width of the film is a total of 18 on-off channel positions somewhat analogous to the eight channels on PDP-1 punched tape. This binary type film can be read by straightforward procedures using logic and memory tables to keep track of the eighteen channel positions, which may vary as much as one tenth the width of the film. We are presently designing a system which will allow 10^{12} binary bits on a 100 foot roll of 16 mm film.

Analog Film

Analog film is completely different from binary film, especially in the reading process. In its most general sense, analog film is simply a picture of something; e.g., a radar scope trace, a microscope slide, or perhaps an x-ray of human lungs. The process of "reading" these pictures amounts to various levels of pattern recognition.

The Film Reading System

Information International, Inc. has recently completed the development of a film reading system suitable for reading both binary and analog film. The film reading system is based on three major elements: a PDP-1 digital computer, together with a visual display scope; a film reading device; and computer programs for using the computer and film reader.

The film reading process involves the scanning of film by a rapidly moving light point on the visual display scope. The output of this scanning operation is detected by a photo-sensitive device in the film reader and relayed to the digital computer for further processing and analysis. In addition to translating the data itself into a more desirable format, the film reading system can also furnish summaries and analyses of the data as may be required.

The flexibility of the film reading system in two respects should be emphasized. First, almost any format of data on film can be read, with appropriate modifications to the basic computer program. This includes data represented in the form of lines, graphs (e.g., radar pulses), points, and other similar forms of data. Second, almost any type of desired output may be obtained once the basic data is obtained from the film. Forms of output which are available include the following:

1. A print-out or listing of data on paper
2. A record of the data on magnetic tape
3. Visual representations of data. These may take the form of a continuous graph (using a digital x-y plotting device). Or they may take the form of photographs -- still or motion -- of scope displays. This latter format is particularly flexible in that computer programs may be designed and written to provide many types of useful and informative data representations.

In addition to data recorded on film, data recorded on any light-permeable medium (such as light or medium weight paper) can also be read by means of the film reading system.

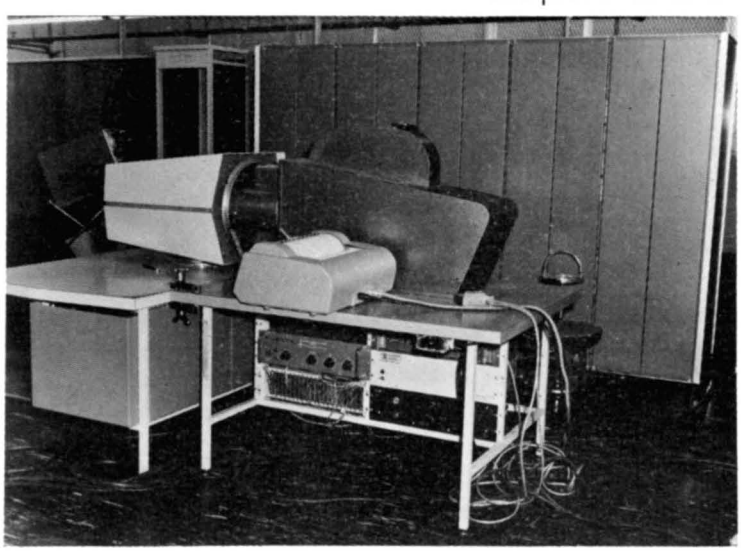
Applications of the film reading system include the following:

1. Analysis of data produced by oscillographs or other types of graphic recorders
2. Tracking and analysis of objects for which motion pictures are available (e.g., missile tracking studies)
3. Reading of astronomical or astrophysical data recorded on film (e.g., analysis of stellar configurations)
4. Reading photographs of cloud chambers, bubble chambers, and spark chambers
5. Counting of particles (such as blood cells or bacteria) in photographs

Reading of Analog Film

A film reading system to read analog data representing missile tracking studies recorded in the form of radar pulses on 35 mm film was completed. The film reading system was developed with the following capabilities:

1. Approximately 500 readings per frame of the amplitude of the radar signals
2. Computation of the median amplitude value for each frame
3. Count of the number of radar pulses
4. For each radar pulse, a measurement of pulse width, average amplitude during the pulse, and measurement of the location of the leading edge of the pulse
5. Recording on magnetic tape of all original and processed data



A FUNCTIONAL DESCRIPTION OF THE ITEK DISPLAY SYSTEM

Earle W. Pughe, Jr.

Abstract

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

Introduction

In the development of the Electronic Drafting Machine (EDM), sometimes referred to as the Digital Graphic Processor (DGP), the need for a suitable real-time input-output computer display was immediately apparent. A mechanical X-Y plotter was deemed too slow. A conventional point by point computer display has objectionable flicker and consumes a substantial amount of computer time when displaying several hundred inches of lines. Thus there was a need for a flicker-free real-time display that did not require appreciable computer time to operate and which could display a meaningful drawing while accepting a light pen input as the drafting pencil.

Requirements

To meet the flicker-free requirement a frame rate of 30 per second was selected. This rate is compatible with a conventional 1800 RPM drum or disc, the same as TV frame frequency and close to the 16 frame or 32 fields per second of home movies. If a higher frame rate is used, the amount of display is correspondingly reduced. Conversely, reducing the frame rate would enable more data to be displayed; but the flicker would become very apparent and, objectionable.

To display a useful amount of data, i.e., more than 500 inches of lines on a 10" x 10" display area at 30 cycles, a point by point display is too slow and a faster scan such as TV, requires too many bits of storage, so a continuous incremental line drawing technique was selected. The basic method employed is to specify the initial $X_o Y_o$ position and then to leave the beam on and give incremental values of ΔX and ΔY which are added to the current $X_n Y_n$ each increment of time. Thus a line drawing technique much like that used on many mechanical X-Y plotters is used.

A 2^{10} or 1024 position scope such as the DEC Type 30B is used with the normal increment being 4 adjacent positions or approximately 1/25 inch. The period is 1.67×10^{-6} seconds, i.e., 20,000 increments for 1/30 second, giving 800 linear inches of drawing. Certain bookkeeping functions tend to reduce the 800 inches in a 1/1 scale. However, scales of 1/4, 1/2, 1, 2, 4, 8, and 16 are available so several thousand linear inches may be displayed. On a 10" x 10" display, 1000 inches of drawing represents 10 lines per inch across the face of the scope; an amount far in excess of any usual line drawing. About 2000 characters can be displayed on the scope, which gives the display system use as an editing tool.

Block Transfer

To meet the problem of releasing the computer for other uses when the display is not changing, the display data is blocked onto a drum or disc in 6-bit parallel from the computers' core memory. A special 1-0 order (720061) has been added to the PDP-1 at Itek and to one PDP-1 at AFCRL to block transfer the data from the core memory onto the disc or drum, or to read the data back into the core memory from the drum. Since 20,000 bits per track for 6 tracks in parallel represents almost 7000 words, several blocks are required to fill up the drums display tracks. A special feature of the block transfer is that no extra counters are needed. If the block order is given in register n, registers n+1 through the end of memory are blocked on the drum (or read from the drum), the program counter end carry terminates the order and the next order is taken from register zero. Thus, a typical transfer would be:

```
n-1,  220010   register 10 contains the drum address
n,    720061
```

n+.,
--
7777 } display data
0 601000 jump to next computer routine

When it is desired to use the computer display to perform such functions as tracking the light pen or displaying control points, the computer display operates in the normal way and takes precedence over the drum display. When the computer display is done, the drum display continues with that portion of the picture erased which would have been displayed at the time of the computer display. Two techniques are available to the programmer to keep the computer display from apparently interfering with the drum display: one is to display from the computer at a time allotted for this purpose on the drum, and the second is to erase a different portion of the display each time.

Display Processor

A block diagram of the system is given in Figure 1. The computer blocks the display information onto the drum. Between the drum and the display CRT is a display processor. The function of the display processor is to decode the 6-bit display instruction, perform the necessary arithmetic to calculate the new X-Y position and to transfer this information into the X and Y decoder registers in the display console. The display processor is a small special purpose computer. The 6-bit display byte is decoded as an instruction which in turn, with seven timing pulses, controls the X accumulator and Y accumulator. The 6-bit display byte is not added directly to the X and Y accumulators. There are codes for such functions as ignore, beam intensity, point plot etc.

In the display processor there is a drum sector counter (20 sectors) and 8-bit segment counter. The sector counter is cleared by the drum index pulse and counts the 20 drum sectors as the drum revolves. The programmer can number each line or character by giving a segment command at the start of each pulse. The sector pulses clear the segment counter.

Light Pen Response

If it is desired to use the light pen to select a line or character displayed from the drum, the segment-sector counter is read into the computer when the light pen sets its flag. Each line is given a segment count by the programmer before blocking the display data onto the drum and this data is transferred onto the drum along with the picture. The light pen response is well under one microsecond so no ambiguity results.

While the display system was designed for the Electronic Drafting Machine, it has general use as a computer output device. At Itek the display has been extensively used as the output for lens design[1] and has been used to a lesser extent as an alphanumeric display.

---

1 See paper by Shannon, Morello, Rimmer and Radkowski - "Use of Displays in Optical Design."

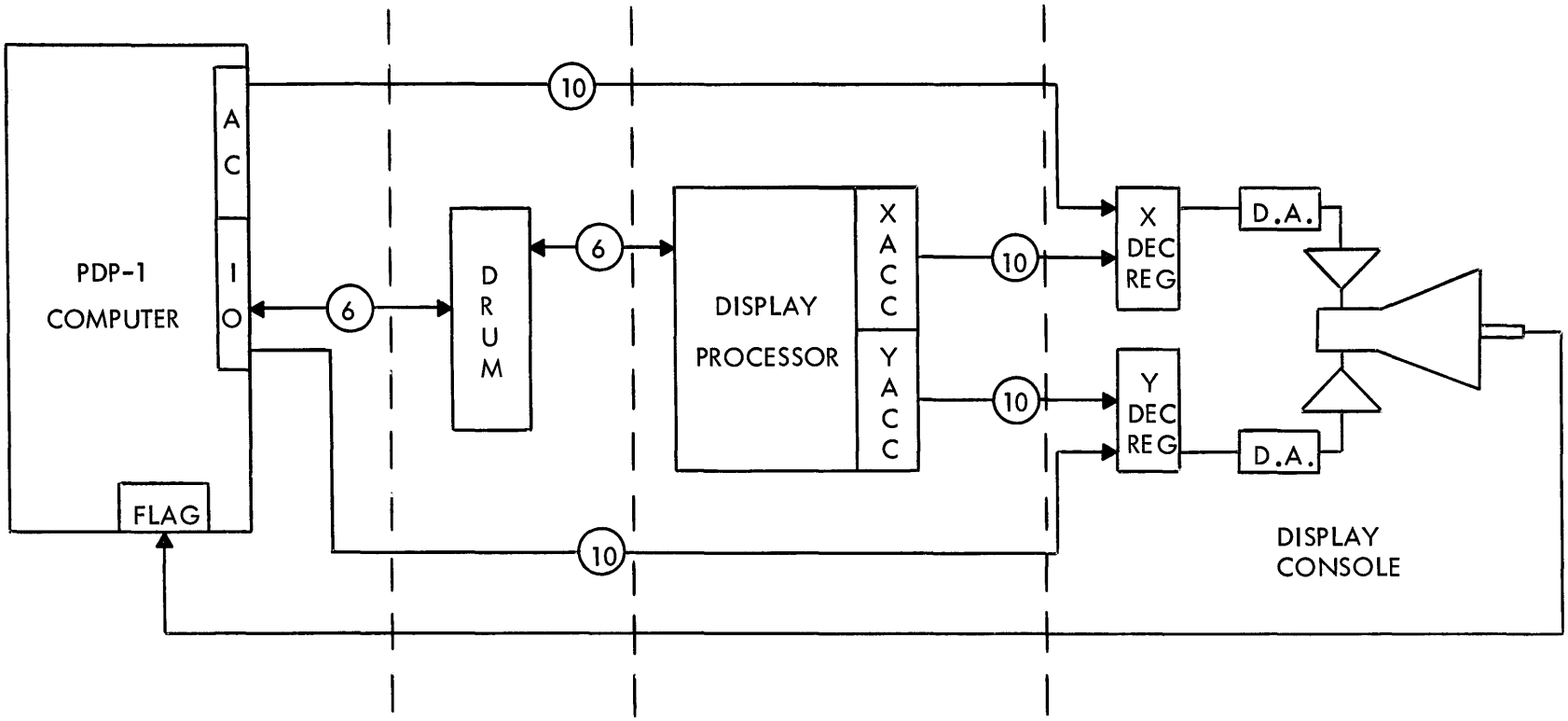


Figure 1 Display System Block Diagram

A TIME-SHARING SYSTEM FOR THE PDP-1 COMPUTER *

John E. Yates

Abstract

A system for time sharing of the PDP-1 digital computer with seven typewriters, two paper tape punches, two paper tape readers and two CRT Displays is described. The additional hardware required for the system and the modification required to a basic PDP-1 are described and a program is presented to handle the monitor of "executive" functions of the system. A System using two typewriters, one punch, one reader and one display based on this design is currently being installed at M.I.T.

Introduction

A time sharing system for the PDP-1 at M.I.T. has been designed and is in the process of construction. It allows for the use of seven typewriters, two paper tape punches, two readers, and two CRT displays simultaneously, by up to seven users. Every effort has been made to make as many features of the basic machine available to users as possible, although some sacrifices must be made to make the computing capacity available to several users simultaneously.

The System

The seven consoles which comprise the system each consist of a typewriter, six sense switches, a console ON switch, a display lever which allows lengthened quantas, a debugging button, and two lights indicating the console is active in core and it is permissible to type in. The two punches, two readers and two displays are shared among the users on an assigned basis. The test word switches are also assigned.

Programming for the System in Two Parts

The programming for the time-sharing system consists of two parts, the executive routine and the administrative routine. The executive routine is a permanent part of core memory (approximately 512 registers) which will handle the needs of the time sharing system on a second-to-second basis. It will handle the so-called instruction traps and time-out interrupts. 2. The administrative routine is a separate program brought into memory on request to perform such jobs as: assignment of equipment, regulation of memory protection, provision for services such as an assembling, debugging routines, editing programs, error indication for illegal instructions, and other miscellaneous jobs. Let us assume several users are using the computer, a particular program is in core and is being executed. Since one does not wish the computer to stop because of a user's errors, (and thus keep others from executing) certain provisions must be made. All halt instructions, illegal operation codes, requests for manual run, and illegal instruction cause a trap to the executive routine, ER (See Figure 1).

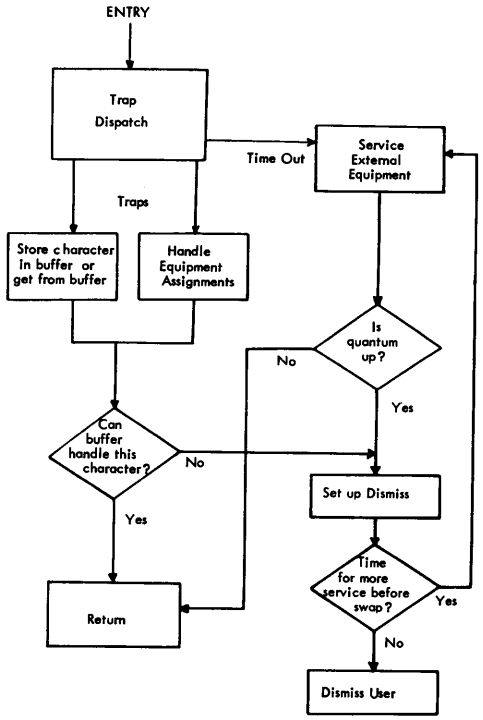


Figure 1 General Flow Diagram of Executive Routine

Similarly certain IOT commands must trap as the program does not know if the equipment has been assigned to it, or which one to address if one has been assigned. The ER then executes the command using the correct assignment, or puts out an error indication thru the administrative routine

Maximum Efficiency

The program may well compute or require characters faster than the I/O equipment can take care of or supply them. Normally, the computer waits in an in-out halt for the completion pulse before processing the next character. Under the time-sharing system it goes to another program while waiting. For maximum efficiency, several characters are computed at once and stored in a buffer in the ER. Then the next program is brought in. At frequent intervals a time-out interrupt occurs where in control is momentarily transferred to the ER. Here one character is taken from each buffer and transmitted, if the I/O device is ready to accept. If not, it is skipped. Control then returns to the program in core. When a certain maximum time has elapsed, or if the ER buffer becomes full, or if the program runs into an error, the program is dismissed and another brought in. A magnetic drum capable of holding twenty - two memories is used as auxiliary storage for the programs not currently active in core. In this way no time is wasted and each user's program is in memory often enough for the user to think he has the computer to himself.

Several instructions have been added to the machine which are valid during the time the executive routine has control. They are decoded from the IOT 77 class and are used by the executive routine to test the states of the consoles, to make equipment assignments, and to provide the proper information to the status bits for the user current in memory.

*This paper is based on a thesis prepared by the author in partial fulfillment of the requirements for the degree of MS in Electrical Engineering, M.I.T. The complete thesis is available as report ESL-R-140 from:

Publications Department
Electronic Systems Laboratory
Building 32
Massachusetts Institute of Technology

PROCESS CONTROL APPLICATIONS OF PDP-4*

C. G. Bell

Abstract

The PDP-4 is designed to operate as a module for a large majority of process control applications. In terms of these applications the interface capabilities allow PDP-4 to be connected to the process, or to Input/Output equipments, with a minimum amount of extra hardware in a relatively straightforward manner.

The PDP-4 configuration, in terms of the above design constraints is described as well as the specific interface terminals, their polarities, timing and operation with the program. The terminals include Device Selection, Information Collection, Information Distribution, Program Interrupt, Data Interrupt and Real-Time Clock.

*The Editor regrets that the complete paper was not received in time to be included in these proceedings.

The PDP-4 data processor was introduced at the May 17 meeting. Users of this processor were formally welcomed at the DECUS Annual Meeting in October, 1962. Future plans by DECUS Executive Board include a special meeting in 1963 for PDP-4 users.

Section IV

PANEL DISCUSSION

MACRO, DECAL, and the PDP-1

Moderator: Dr. John Hayes*

Panel: (MACRO) Professor Jack Dennis, M.I.T.
Harrison Morse, DEC
Alan Kotok, M.I.T.

(DECAL) Edward Fredkin, Information International, Inc.
Ted Strollo, BBN
Roland Silver, Mitre Corp. (not present)

Dr. Hayes: For some time now, there have been rumblings among programmers about DECAL versus MACRO for the PDP-1. It began to look as though perhaps people wouldn't talk to each other who talked different programming languages. From the discussion today, we should learn a good deal about both of these programming languages.

Among a number of things, I'd like to remind our panelists that some of us know DECAL, some of us know MACRO, some of us don't know either. But very few of us know both MACRO and DECAL. So, I hope that statements may be explained where there may be a language difficulty. In the process of discussion, I would hope that we would develop first of all, for the purpose of new users who are perhaps trying to decide which language to use, the relative advantages of one over the other or perhaps the relative advantages of using both. For old users of MACRO or DECAL, such as the members of my laboratory, we would like to find out enough advantages of one language over the other to justify the time and expense (which may well be considerable) of retraining people to use the other language.

Now I'd like to introduce the panel: For MACRO, Professor Jack Dennis of M.I.T., Harrison Morse of DEC, and Alan Kotok of DEC and M.I.T. For DECAL, Ted Strollo of AFCRL and BBN, and Edward Fredkin of Information International, Inc. I'm sorry that Roland Silver, who was to speak for DECAL could not attend. I am moderator, but I plan to moderate only in the case of severe physical danger to one or more participants. Professor Dennis.

Prof. Dennis: Am I correct in the understanding that so far in this meeting, there has been no presentation of MACRO? How much time do I have for my initial presentation?

Dr. Hayes: The time will be five minutes.

Prof. Dennis: I will start with a brief description of what MACRO is. MACRO is an assembly program (as opposed to a compiler program) and was originally developed in 1958 and 1959 for the TX-O computer at M.I.T. The needs of the users of the TX-O computer, at that time, were the determinants of the features that were placed in the MACRO assembly program. The original version of the MACRO

*Psychologist, Operational Applications Laboratory, Air Force Electronics Systems Division, AFSC, Bedford, Mass.

assembly program was based quite a bit on previous experience in the Whirlwind Laboratory at M.I.T. and the experience of the people who participated in the Whirlwind group. When the TX-O computer was brought to M.I.T. in 1958, we had need for creating a new programming system for the machine. At that time, we asked for an assembly language so that the machine could be used by students and research people at M.I.T. We discovered many features which should be in an assembly program for the type of uses which were being made of it - for example, the automatic macro-instruction feature, whereby a user may assign a name to a sequence of instructions or words and later on in his program use that name to specify the sequence to be placed in the object program. Since then additional features have been added to TX-O MACRO, such as automatic reservation of storage for constants, for variable automatic storage, and automatic reservation of table space by using a dimension statement. Various people here helped with this work. Among them, we should mention Bob Saunders in particular (now with Information International), Bob Wagner who is working for the Rand Corporation, and Alan Kotok who is on this panel. When the PDP Computer was donated to M.I.T. by the Digital Equipment Corporation in the fall of 1961, there was quite a bit of concern about the kind of language that should be provided for the PDP for use by students and staff in the M.I.T. work and after some time we decided to translate the MACRO program so it could be used on the PDP. This was a rather easy job because of the great similarity of the two machines, and I believe it was accomplished in something like three weeks of work on the part of about four people, working part time, which was quite an accomplishment. As to the reasons I think that MACRO is a very useful assembly program, I have the feeling that for the PDP Computer, an assembler is more desirable than a compiler. I feel this way because applications made of the PDP-1 are such that using a compiler would lead to object programs which are relatively inefficient and require considerably more space than required by a program hand-coded for translation by an assembly program. When I say this, I don't mean that a compiler can't be constructed which would be suitable for the PDP-1, but I believe that compilers which are based on the kind of compiling techniques which are now in existence would lead to programs which are long and time-consuming in their operation on this machine. So my feeling is that for many classes of problems for which the PDP is used, an assembly language is the more important language to be concerned with. I believe that the MACRO source language has all of the more important useful features of any assembly language in existence and is very flexible in its use. I think I'll wait until later for any further comments.

Dr. Hayes: Maybe we should now turn to a DECAL representative: Mr. Fredkin.

Mr. Fredkin: Well, I guess I've been labeled a DECAL representative. Let me say something about MACRO, though. I like MACRO and I think it's fine, but I don't think it's fine for the PDP-1 because I don't think the PDP cares what you put in the reader as long as you don't make it shudder too much. I think MACRO is fine for a group of people and there are members of that group here. The fact is, that we could be sitting up here arguing whether we should speak English or some other language and I don't think you can argue this on the merits of the language so much as by looking at the language in terms of its practical uses. DECAL and MACRO are two very different languages. DECAL is a very complicated system, MACRO is a simple system. Simplicity is very nice sometimes and the PDP-1 is perhaps a simple computer, but if you describe the two systems by listing their properties, DECAL includes more of the desirable features of MACRO than vice versa by a big margin.

1. DECAL has one important thing and this is really best described as growth potential. The language is increasing in capability with time. The fact that it's changing may be a disadvantage, but still it is including more and more of the ALGOL language features.

2. DECAL has a library feature. It allows groups, large organizations to set up systems with various individuals' programs. It allows you to use library programs and library tapes and allows you to relocate binary - in general, those things that are oriented for systems programming.

Now MACRO, on the other hand, is a beautiful language for one person who wants to sit down, write his program, and make it work. This is characteristic of many users of MACRO; in particular, I would say, characteristic of M.I.T. students who generally don't get together to write large systems, but write their own programs. They want to assemble, get something out, run it, debug it, etc. That is very different from the way many other organizations use computers. So, I really feel that there are a set of users for which MACRO is better. On the other hand, I feel that there is a much larger set for which DECAL is better because most organizations have invested in systems and these systems are large and quantitative.

Another point...Sure enough, MACRO does result in efficient object codes, but normally I don't care. What I care about is the lapse of time between when I start writing and when I have a finished program. Generally I'm going to write the program over five times and maybe the last time I'll do it in machine code. I want the amount of time I spend doing this to be minimum; I don't care about the machine. I usually write programs for hours that only amount to milliseconds and so sometimes it takes 10 milliseconds (instead of one) for having used the DECAL algebraic compiler. On the other hand, use of the DECAL compiler may cut hours off the time of actually writing the program itself. So, I think that there are very specific issues involved in the choice of a programming language, and I think I'll defer getting down to them until we've heard from each of the people.

Mr. Kotok: I don't think I can say as much as the two gentlemen who preceded me, but I think they did outline the issues pretty well. One thought, which might be germane, is that maybe we shouldn't be arguing whether MACRO versus DECAL. Instead, whether either one of them or FRAP. It seems I've been informed that a large number of users are still using FRAP for one reason or another and maybe if we come out no where else, at least users will know something about one of these two systems that we are discussing here.

I'm certainly not trying to claim that MACRO is as general a system as DECAL, especially, the new DECAL described in the paper that was presented before this discussion.* I think that you obtain through this generality, the facility of the use of DECAL (as was shown in the discussion that preceded this one) the description of instruction generators and action operators, which caused Mr. McQuillin to indicate that even he gets confused occasionally. The macro-instruction feature of MACRO is somewhat akin to the instruction generator feature of DECAL and our system is, we think, somewhat easier to use. I think that most of the complaints against MACRO and why people say DECAL

*DECAL-BBN - Symbolic Version of DECAL - by R. J. McQuillin - p. 19 of these proceedings.

over MACRO as an assembler is that, first of all, MACRO has a limited set of symbols: 1, 2, and 3 characters. I can certainly see that people can get unhappy with this. Maybe I'm on the wrong side of the fence, too, but, like Ed, I can see where there might be objection to this. Also, in MACRO there is an absence of a linking facility between programs. However, the linking facility, it seems, as provided by DECAL, is a mixed blessing. Since it is a one pass system, there is no way to directly get a loadable binary type which can be read in right away. The second pass of the assembly that MACRO does do is often necessary in DECAL if you do wish just a self-loading tape causing you to go through two passes of punching. These are just a few of the points.

Mr. Strollo: I'd like to say that I don't think it's different types of programmers who should use MACRO or DECAL. It would seem to me it depended on the type of program to be written. Sometimes when writing a short program which one would like to get into the machine as quickly as possible, MACRO has advantages. But, if one were working on a long system and expected to link several short programs together then I think DECAL is the better system to use because programs which other people have already written can easily be incorporated. A library tape, for example, could be used. Thus available programs which have already been debugged can be linked with the recently written program. Certainly this is a lot easier than recompiling all programs over again and going through a new process with each program and then reading it into the computer.

I think there are two philosophies you can have when writing programs. You can either write one very large program (and you would almost have to do it with MACRO where all the symbols are linked together and where you stand a good chance of not getting the program debugged for quite a while) or you can write several short programs debugging each program as you write it. When you are certain that programs A & B are working, then you can write a program C and get it working and then try it in conjunction with A and B. I think the latter is a strong point of DECAL. You can take all of your shorter programs which you know are working now and link them together with your recently written program and be pretty much certain that you're not going to have a major debugging problem.

Mr. Morse: I would like, first of all, to make one thing very clear. An impression, I think, Ed Fredkin and Ted Strollo have given is that it's difficult for more than one person to work on one program in MACRO and it's difficult to write a program that isn't one big chunk of coding. This is not true. I have many times written programs which consist of a big glob of subroutines (literally 20 or 40 to 100) and a large control program, with the subroutines on separate symbolic tapes. The subroutines are assembled and checked out separately, prior to putting together the whole system. Once the subroutines are checked out, these are punched out on a binary tape and a symbol punch gotten from MACRO. The subroutines sit in a fixed place and remain there while you go to work on a control program. One great advantage of this is that you can use MACRO instructions as a means of calling these subroutines. In particular, one person can write all the subroutines, define how they're used, and another person, not knowing a thing about this big black box, can use MACRO instructions to call the subroutines. This is one way of performing the same function that DECAL does with the relocatable subroutines which are called by system symbols when the main program is loaded. Just this to counteract the impression that MACRO

is for one-man programs only. There are advantages to both systems. If you're doing mainly arithmetic processing, DECAL does have the advantage that you can write a program much more quickly and possibly get it debugged much more quickly. A disadvantage here is that at the present time DECAL's programs must be debugged in octal. This will eventually be counteracted by using DDT and symbols from DECAL for debugging. Another disadvantage is that if the DECAL program is large and has many systems symbols which must be stored in memory while loading the program, then you also have storage problems that are alleviated by using MACRO since you can use all of core except the last 27 registers or so.

Mr. Strollo: If I understand MACRO correctly all symbols are three characters in MACRO. Is that correct? For example if someone else were working on a program could you say to them don't use the symbol A because I'm using the symbol A in my program and you can't use it in yours? Is this what you would have to do? I think there should be a feature for external symbols because there are a certain group of symbols that I use over and over again in several of my programs and even if I were working on a system I like to use these symbols within the program like "move" or something like that.

I think the communications problem when you're writing a long system would be enormous if you had to eliminate all the symbols you use and pass it on and say don't use these symbols in your program.

Mr. Fredkin: Well, when writing with FRAP we used to break things up. We used to say I'll start all my symbols with my initials. This is sort of hard when you're limited to three characters because it doesn't leave too many initials.

Dr. Hayes: Especially if you have a long name.

Mr. Fredkin: That's right. If you have four initials.

Prof. Dennis: Take the example that Mr. Morse gave in which you compile a set of subroutines and then define a set of MACRO instructions to be the calling sequences for the subroutines. After you've got to that stage, you may dispense with the symbols which are involved in the subroutines and simply use the MACRO instructions in your main control program. So once you have coded the subroutines and defined the calling sequences and debugged them you may dispense with all of the symbols involved in these programs in the subroutines and refer to them only through the MACRO instructions.

Mr. Fredkin: Isn't that true only if you know the binary locations?

Prof. Dennis: No.

Mr. Fredkin: Dispensing with all the symbols?

Prof. Dennis: One way of doing this is to define the calling sequences as macro-instructions on a separate tape which is assembled with the subroutines. Then a definitions tape is obtained containing only the macro-definitions. If you have used the system correctly, the macro-instructions defined on the tape provide the correct calling sequences for the subroutines, but the tape will not have any of the symbol definitions of the subroutines.

Mr. Fredkin: However, these subroutines will not be able to link if you get rid of their definitions, unless you pick the binary location.

Prof. Dennis: That is correct.

Mr. Fredkin: Now for instance on a DECAL library tape you might have 20,000 instructions worth of program. You can't fix the binary locations and pick any subset so it's impossible to have such facility in MACRO where you do in DECAL. By the way, when we talk about library tape I thought I'd mention one thing. One of the advantages of MACRO is the ease of tape handling. With DECAL, especially with this library tape and such, we had in mind from the very beginning that this should be a magnetic tape feature eventually. It should work with paper tape, and in addition it should get into magnetic tape. It is on magnetic tape here and there are people who put in a little paper tape, maybe about 20 fanfolds, where they crowd a whole slew of things in the library and just go whizzing through this mag tape and they pick up all of these routines so that you do get access easily in a relocatable form.

Prof. Dennis: I would like to point out that for the TX-O computer there is a relocating version of MACRO assembly program. The relocating features were not translated into the PDP version because of space limitations in memory of the PDP. However, I expect that this is something that the Digital Equipment Company would be interested in doing, but we don't have the manpower at this time.

Another way, is to store the symbolic version of each subroutine on tape and add to MACRO a facility which could be done with about as much trouble as putting in the DECAL library tape to call the subroutines wanted in symbolic, assembling these at that time. This means a double tape handling, but when you're handling magnetic tape the extra time needed is still so much less than the time used to handle the paper tape it becomes a very workable scheme and does not entail large changes to the MACRO system itself.

Mr. Kotok: Just to comment on the thing Prof. Dennis was talking about before - assembling a large number of subroutines and using the MACRO instructions with these subroutines as calling sequences may be done by using the symbol punch facility in MACRO. The symbols may be punched for use with DDT, or the MACRO instructions without the symbols may be punched for use at a later date. The MACRO calling sequences would be absolute addresses of the subroutines for later use.

Mr. Fredkin: Three things occur to me: First, how about the length of symbols because we can't name everything you want with three letters? Second, what about relocation? Third, what about library tapes? These are all features of DECAL now and they could be a part of MACRO.

Dr. Hayes: Yes, our discussion seems to have boiled down to the properties of future programs. Are there any further comments from the panel or is it now time to entertain questions from the floor?

Note: (Questions from the audience were not audible for purposes of recording them. One question to Dit prompted the explanation of macro-instructions.)

Mr. Morse: I would like to give a brief description of how to use macro-instructions. The

macro-instruction facility is a way of naming a series of instructions which are commonly used in the program, which can be put in the program by writing the name of the macro-instruction.

For instance to define the MACRO instruction load:

```
define      load B, A
            lac (A
            dac B
            terminate
```

This MACRO instruction is commonly used to load register B with the constant A.

Now to use the instruction in the program to load 3 with register zzz one need only write:

```
            load zzz, 3
```

I may also use other MACRO instructions within a MACRO definition:

```
define      load 2 z, one, two
            load z, one
            load z+1, two
            terminate
```

The use of this

```
            load 2 g, 4, 20
```

will cause the following instruction to be assembled

```
            lac (4
            dac g
            lac (20
            dac g+1
```

This operation will be performed many times. The argument A will be cycle lac 9 times and that can be used as part of the later work. For example, this is essentially the MACRO feature.

Mrs. Newman: A good, brief description of MACRO appeared in the May 1962 issue of DECUSCOPE.

Mr. Morse: Thank you.

Prof. Dennis: In programs written in a large interpretive system (for example, a system for floating point computation), the interpreted instructions may be given names with mnemonic significance by parameter assignments or macro-instruction definitions. With macro-instructions, specifying the parameters of an interpreted instruction is far more convenient. Of course, an interpreted instruction may occupy two or three registers, depending on how many arguments must be specified to the interpreter. This makes no difference when you are using macro-instructions. The macro-instruction may have a length of 1, 2, or 3

registers depending on the particular instruction it represents.

Editor's Note: There was a comment from the floor about the ease of writing macros.

Mr. Fredkin: There is one thing about macros. They are easy to write, but I would rather work with "instruction generators" which are easy to use. You use things more often than you write them and since you are only going to write it once you don't need MACRO to do it. Let me give you an example of this. What do you do when you want 39 in register? In DECAL you write: 39⇒A but in MACRO you have to remember whether it is: LOAD A, 39 or: LOAD 39, A (which goes into which). I guess Dit made a mistake in the definition and you can write into A, put 39. His results will involve the equivalent law 47, and dac into A. Taking Dit Morse's example in the May DECUSCOPE; I showed him a program in DECAL which did the same thing and it was easily 1/8 as long and he said that's not fair because I used existing subroutines. I didn't use anything but a single DECAL library tape. So the program was shorter and much easier to write.

Mr. Morse: This is true, but first of all, the example was to illustrate the use of macro-instructions and was not intended to compare MACRO's virtues with those of any other programming system. However, let's use it for that as Ed has, and compare the effort needed to run the programs. Using MACRO, you need only do two passes on the symbolic tape and you have a binary tape which may be loaded and run. Using DECAL, you must first assemble the program, then load the linking loader, load the program, load the library tape, and if you do not wish to do this every time the program is run, you must load punch-off and punch out a binary tape.

Editor's Note: There was much reaction in the audience, especially from DECAL users.

Dr. Hayes: I think the audience is getting jittery because they cannot participate. Are there any questions from the audience?

Questions from the floor: One of the features of DECAL is the instruction generator. I think this is equivalent to definitions. Is this correct?

Prof. Dennis: Yes, in the form of macro-instruction definitions.

Mr. Saunders: What you can do, for instance, is to have additional MACRO instructions written into the programs. What one cannot do is have the MACRO instructions written in duplicate on certain substructures depending on the value.

Mr. Strollo: If you can't get all of the instructions in on DECAL, you can insert a new tape in DECAL. Can you do this in MACRO?

Mr. Morse: Yes, it is possible.

Dr. Hayes: Any comments from the floor?

Question: Not audible but Moderator repeated.

Dr. Hayes: The question has to do with the use of magnetic tape with DECAL.

Mr. Fredkin: When you use it, my experience with DECAL is that even paper tape tears much less. DECAL definitely has growth potential with respect to magnetic tape.

(A question was directed to Mr. Fredkin about writing programs.)

Mr. Fredkin: The thing is that DECAL has facility for doing things. In MACRO you write the programs over and over, but in DECAL we only do it once. A very important thing is the joining of binary programs. You can do it in MACRO, but in DECAL we put them in locations and never bother with them again. In general, if you have a very complicated mathematical thing and you have to be fast, you can do parts of it in DECAL algebraic language and then maybe convert.

Prof. Dennis: The language I would use would depend on whether my program could be divided into subroutines. Certain programs are impossible to divide into subroutines. Then the question of MACRO versus DECAL depends on whether the macro-instruction feature of MACRO turns out to be useful with reference to what you are doing, and in most cases it is. The advantage of using MACRO for programs with many subroutines is that you can give nice names to their calling sequences and refer to them by convenient names. You have the advantage in DECAL which is given by the linking loader feature. I prefer the coding format of MACRO to the coding format of DECAL. This, of course, is something outside of what either program can do for you and I admit that this is a matter of opinion and my personal bias. It may also have something to do with my experience with MACRO.

Editor's Nore: Discussion from the floor became more lively but speakers were heard by those sitting close by. A question was raised about the effect of DECAL on the PDP causing strain on input-output devices and it was pointed out that the M.I.T. machine had been modified for MACRO and didn't accept DECAL. Jackson Wright repeated that the format of MACRO was easier for a program writer. A little excitement was engendered at this point. It was obvious that the audience was having a good time and that the DECAL users thought it more advantages for them in its present form.

Dr. Hayes: Yes, Mr. Kotok.

Mr. Kotok: I saw Ted Strollo working on a program on the flexowriter. I didn't see any algebraic statements in it at all. He mentioned the manipulations which you will have to go through to type the DECAL program, some of which have to do with just which characters to choose. All the upper cases were troubling him. Also, a system where you have to put in information as to where you are assembling and not compiling has many difficulties such as the difficulty of putting in addresses alone.

Prof. Dennis: It depends on whether you are talking about compiling. If you are doing your own programming and have no typist, the more characters you have the more chance for errors.

Mr. Strollo: This could be remedied by the action operators in DECAL.

Mr. Fredkin: Ease of typing should not be the basis for evaluating a system.

Dr. Hayes: It is difficult to evaluate on the basis of how many keys you have to strike to make a comma.

Mr. Fredkin: There was a time when I, too, used to program in MACRO. I liked MACRO instructions but I've made progress. The algebraic statement is the best although I'd like to have a combined system.

Mr. Kotok: We ought to ask the audience what they like, we have been talking mainly about what we have to offer. It would be interesting to find out what they use and what they like. (Many voices and affirmative nods.) Who are DECAL users?

Editor's Note: The moderator asked for a show of hands. The number of people using MACRO and the number of people using DECAL were about the same. The count for each system is given below.

16 DECAL
16 MACRO - (M.I.T. programmers)
9 FRAP

Dr. Hayes: About one-half of the audience did not indicate a preference. That is very interesting. Yes, Ed.

Mr. Fredkin: MACRO is 5 years old and has reached some maturity. It has a good write-up. DECAL hasn't reached the same state of maturity but seems to be getting there. I think that within the not too distant future we will see DECAL with a good up-to-date Symbolic and a good write-up.

Prof. Dennis: DECAL as it is presently offered, does not have the possibility of subscripted variables -- the most important feature of the algebraic language. I understand a version of DECAL is being prepared now which does offer subscripts but I have the feeling that putting subscripts in DECAL is going to increase the inefficiency of object programs over programs created with the absence of subscripts and I think it is possible to create a compiler language for a computer like the PDP-1 which could compile efficient object programs better than any today in that it would not be a one to one translation between source programs and object representations. I believe that such a program is possible and I would like to see one prepared and I would then be sure to use a compiler for any program I would write, but until such time I will use the assembler.

Dr. Hayes: Yes. Would the other members of the panel like to give some conclusions now?

Mr. Morse: I believe MACRO is a better system for writing programs in which you need close control over the resulting object code and storage allocation, for example a real-time control program. In contrast, DECAL is more efficient from the point of view of the lapse time of beginning a program and getting it running.

Mr. Strollo: It's a matter of what type of program one is writing and whether it is desirable to use programs other people have worked out. When linking a group of programs together, one saves time with the DECAL system.

Mr. Wright: Can you link programs with different symbols and different programs?

Mr. Fredkin: Yes! DECAL does it! BBN has it - In summary; an interesting thing happened some time ago - Elsa Newman got after me. (She's the greatest weapon DECUS has!) With reference to outlining virtues for DECAL or MACRO, her idea was to do something like this debate, but in written form for the DECUSCOPE. So Dit Morse and I got together to have a debate and what happened was that I agreed with nearly every statement he made and I think, vice versa. We got so bored with this, that after three-quarters of an hour, we went home. On the panel today, I decided that I would argue more strongly in behalf of DECAL, but my feeling is that both systems are good, for the reasons I've mentioned earlier.

Mr. Kotok: I must agree with Ed. I argued for MACRO, but I feel as Ed does that both systems are worthwhile. I would have liked to find out about what others like. If one sees something that neither of these systems has or can find a compromise that you think is better drop a line to DECUSCOPE and we'll start something like the ACM debates.

Audience: (laughed)

Prof. Dennis: The discussion this afternoon served a very good purpose in bringing to light the features of these two systems to the audience. If this is so, it has served its purpose.

Dr. Hayes: I hope, in spite of the good-fellowship and gemütlichkeit we have generated, that the audience will have gained some appreciation of the differences between these two systems and that they will now be able to ask better questions about them for their own applications.

Section V

APPENDIX

ANNUAL MEETING

Place: Air Force Cambridge Research Laboratories
L. G. Hanscom Field
Bedford, Massachusetts

Date: October 10, 11, 1962

PROGRAM

October 10 - Wednesday

0900	Registration
0930	Introductory Remarks - Charlton M. Walter, President of DECUS
0945	The PDP-4 Programming System - H. Morse, DEC
1030	Reading Film with a Computer - M. Cappelletti, Information International, Inc.
1100	A World Oceanographic Data Display System - Edward Fredkin, Information International, Inc.
1215	Lunch - Officers' Club
1330	Minimax Detection Station Placement - Richard D. Smallwood, AFCRL
1400	Displays -

Group I: to the DX-1 Experimental Dynamic Processor Room

Display of Minimax Detection Station Placement

Dynamic Attribute Extraction Display & Discussion - Charlton M. Walter, AFCRL

Display - Steven Bernstein, AFCRL

Group II: to the Operations Applications Laboratory

Displays & Discussion

1600 Reconvene in Main Conference Room, Building 1105A - General Discussion and Security Check

October 11 - Thursday

0900 Matrix Package for the DX-1 System - Carmine Caso, Wolf R & D

0920 Lawrence Radiation Laboratory's PDP-1

1. A Peripheral Processor for Large Computers - Mrs. Dorothy Monk
2. A PDP Systems Tape - Fraser Bonnell
3. Translation Problems of a Peripheral Computer in a Multilingual House - R. P. Abbott and L. E. Mish

1100 Playing Music in Real Time - Peter R. Samson, MIT

1115 Business, Introduction of Newly Elected Officers

1962-63 Officers

Edward Fredkin, President
Elsa Newman, Secretary

Committee Chairman

Eunice Cronin, Meetings
William Fletcher, Equipment
John R. Hayes, Programming
Elsa Newman, Publications

1230 Lunch

1330 The BBN Symbolic Version of DECAL - R. J. McQuillin, Bolt, Beranek & Newman, Inc.

1530 DECAL, MACRO and the PDP-1 (Panel Discussion)

Moderator		John Hayes, OAL, Air Force Systems Command
Panel	(for MACRO)	Professor Jack Dennis, Massachusetts Institute of Technology Harrison Morse, Digital Equipment Corporation Alan Kotok, Massachusetts Institute of Technology
	(for DECAL)	Edward Fredkin, Information International, Inc. Theodore Strollo, AFCRL, BBN

1730 Concluding Remarks

Edward Fredkin, Decus President, (1962-1963)

ATTENDANCE

ANNUAL MEETING

October 10 and 11, 1962

Air Force Cambridge Research Laboratories

CHARLES W. ADAMS ASSOCIATES
Bedford, Massachusetts
John Gilmore - D
Mary Lanahan
Al Rousseau
Paul Rodenhiser

AIR FORCE CAMBRIDGE RESEARCH LABS.
Bedford, Massachusetts
Frank Balzer, Jr.
B. Bernstein - pd
Harry Blum
Roger E. Bove
Eunice C. Cronin
Robert Duncan
Donald Easterday
Stuart Gygi
Edward LeFebvre
Philip Lieberman
John Mott-Smith
Vera Pless
Eugene Prange
Richard D. Smallwood P, pd
Charlton M. Walter - P, D
Weiant Wathen-Dunn - D

AIR FORCE SYSTEMS COMMAND
(Electronic System Division)
Bedford, Massachusetts
Charles R. Brown - pd, D
Donald W. Connolly - pd
James Duva
Ira Goldstein
John B. Goodenough
John R. Hayes - P, D
Sylvia Mayer
Raymond Nickerson
Anne Story
Paul Wein
Robert Westfield
Major John T. Willis

ATOMIC ENERGY OF CANADA, LIMITED
Chalk River, Canada
J. Quarrington - D

BIO-DYNAMICS
Cambridge, Massachusetts
Avery Johnson

BOLT, BERANEK & NEWMAN, INC.
Cambridge, Massachusetts
Los Angeles, California
M. Breen
Lucy Darley
Thomas Evans
William Mann
Thomas Marill - D
Richard J. McQuillin - P
David Park
Theodore Strollo - pd

JET PROPULSION LABORATORY
(California Institute of Technology)
Pasadena, California
William Sholey

DATA PROCESSING, INC.
Waltham, Massachusetts
Richard Mills - D

DIGITAL EQUIPMENT CORPORATION
Maynard, Massachusetts
Harlan Anderson
Robert Beckman
Gordon Bell
Peter Bonner
Martin Graetz
Benjamin Gurley
John Koudela
Nancy Lambert
Harrison Morse - D
Elsa Newman
George Rice

GEOTECHNICAL INFORMATION
Garland, Texas
Gerald Clawson - D

INFORMATION INTERNATIONAL, INC.
Maynard, Massachusetts
Michael Cappelletti - P
Edward Fredkin - P, D
John Wood

INFORONICS, INC.
Maynard, Massachusetts
Lawrence Buckland - D
William Nugent

INFORMATION SYSTEMS DIVISION
(International Telephone & Telegraph)
Paramus, New Jersey
H. Gould - D

ITEK CORPORATION
Lexington, Massachusetts
William Blotnick
Charles Burgess
Terrence R. Cullen
Doris Gagnon
Richard Hagan
H. P. Peterson
Earle Pughe
Edward Radkowski
Robert Rizzo
Edward Spignise
T. R. Stansfield

LAWRENCE RADIATION LABORATORY
Livermore, California
Frazer Bonnell - D
Lloyd Mish - P
Dorothy T. Monk - P, D

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
Cambridge, Massachusetts
Professor Jack B. Dennis - P, D
Alan Kotok
Peter Samson - P
Robert Saunders
Jackson Wright

MASSEY DICKENSEN COMPANY
Waltham, Massachusetts
W. J. Lennon

OREGON PRIMATE RESEARCH CENTER
Beaverton, Oregon
Robert W. Coffin - D

RAYTHEON COMPANY
Wayland, Massachusetts
Ralph W. Zaorski

SYSTEMS RESEARCH LABORATORIES
Dayton, Ohio
W. Fahle - D

UNITED AIRCRAFT RESEARCH LABORATORIES
East Hartford, Connecticut
Gerard A. Paquette - D
David Sirota

WOLF RESEARCH & DEVELOPMENT CORP.
West Concord, Massachusetts
D. B. Brzezenski - D
Carmine Caso - P
Norman Hirst
Janet Seltzer

Notes: D - DECUS Delegate or designated representative.

P - Speaker or Panel member.

pd - Displayed CRT showing special programs.

AUTHOR INDEX

Radkowski, E. J.	Itek Corporation Lexington, Massachusetts	25
Rimmer, M. P.	Itek Corporation Lexington, Massachusetts	25
Seltzer, J. N. (Miss)	Wolf Research & Development Corporation Concord, Massachusetts	17
Sexton, J. M.	Wolf Research & Development Corporation Concord, Massachusetts	17
Shannon, R. R.	Itek Corporation Lexington, Massachusetts	25
Smallwood, R. D., Lt.	Air Force Cambridge Research Laboratories Cambridge, Massachusetts	23
Walter, C. M.	Air Force Cambridge Research Laboratories Cambridge, Massachusetts	29-41

PANEL MEMBERS

Dennis, J. (Professor)	Massachusetts Institute of Technology Cambridge, Massachusetts
Fredkin, E.	Information International, Inc. Maynard, Massachusetts
Hayes, J.	Operational Applications Laboratory Air Force Electronic Systems Division Air Force Systems Command Bedford, Massachusetts
Kotok, A.	Massachusetts Institute of Technology Cambridge, Massachusetts
Morse, H.	Digital Equipment Corporation Maynard, Massachusetts
Strollo, T.	Air Force Cambridge Research Laboratories Cambridge, Massachusetts