DIGITAL 7-4-S

PROGRAMMING MANUAL

DDT DEBUGGING SYSTEM PDP-7

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

DIGITAL-7-4-S

PDP-7 PROGRAMMING MANUAL

DDT

PROGRAM DESCRIPTION

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

PREFACE

The program discussed in this manual, though written for use on the Programmed Data Processor-7 computer, can also be used without change on Digital's Programmed Data Processor-4. This compatibility between the libraries of the two computers results in three major advantages:

1. The PDP-7 comes to the user complete with an extensive selection of system programs and routines making the full data processing capability of the new computer immediately available to each user, eliminating many of the common initial programming delays.

2. The PDP-7 programming system takes advantage of the many man-years of field testing by PDP-4 users.

3. Each computer can take immediate advantage of the continuing program developments for the other.

CONTENTS

INTRODUCTION

Users of most computers, especially large-scale ones, are familiar with the procedure of submitting a new program for a run and receiving along with the compilation and assembly listings a dump and perhaps a storage map of the symbols used together with a few remarks about the failure of the program to run properly. If the user is present when his program is processed, he may get additional information from the console lights, motion of tapes, etc., but his correcting must be done away from the machine. Getting a program to work under these conditions takes considerable time.

DDT (DEC Debugging Tape) helps shorten this debugging time by allowing the user to work on his program at the computer, to control its execution, and to modify the program or its data. Tracking down a subtle error in a complex section of coding is a laborious and frustrating job by hand, but with DDT's breakpoint facility, the user can interrupt his program at any point and examine the state of the machine. In this way, sources of trouble can be quickly located. In case the program does not reach a breakpoint and halts, DDT will have to be restarted. To do this, place 6000 in the ADDRESS switches and depress START.

The programmer may make corrections or insert patches and try them out immediately. Those that work can be punched out on the spot in the form of loadable patch tapes, eliminating the necessity of creating new symbolic tapes and reassembling each time an error is found. DDT also maintains a symbol table, allowing a programmer to refer to his own program symbols when working through DDT.

In both 4K and 8K machines, DDT occupies the highest 2000_8 registers of memory. Immediately below this, extending downward into memory is DDT's permanent symbol table consisting of all standard defined PDP-7 mnemonics. This may be augmented by symbol definitions from the user's program tape or from the keyboard.

User's Language

A symbol is a string of letters or numerals, the first of which must be a letter or the character period (.) which is treated as a letter. Only the first six characters of a symbol are interpreted by DDT.

A number is a string of up to six octal digits.

An expression is a string of symbols and numbers separated by either of the following two characters:

space	a separation character meaning arithmetic plus
-	a separation character meaning arithmetic minus

All other characters either are used for control or are illegal.

Definitions

When a register is opened, its contents are printed out, and it becomes available for modification.

When a register is closed, any modifications requested are made and further access to the register is denied until it is opened again.

Most DDT operations are specified by symbols whose final character is $. Typing such a symbol followed by a carriage return (↓) causes DDT to perform the operation.

DDT responds to operator errors by typing a ? and ignoring the error.

In ensuing examples, C(R) means the contents of register R.

In the following discussion, all numbers are octal integers. In the examples, LOC, LIST, and BGIN are symbolic address tags in the user's hypothetical program. All underlined expressions in the examples are those typed by DDT; expressions not underlined are those typed by the user.

USING DDT

The user reads DDT into the computer by setting the ADDRESS switches to 17770 and depressing START. The RIM loader is assumed to be in memory. The program is self-starting. When loaded, it will initialize itself and respond by printing a carriage return signifying it is ready to accept commands. All subsequent operations, including loading the program to be debugged, are performed through DDT from the teleprinter keyboard. All printed output appears on the teleprinter.

Before loading a program to be debugged, the user may wish to erase from DDT's symbol table any symbols from a prior program. It may also be desirable to clear memory not used by DDT. To specify these operations, use:

KILL$	DDT's symbol table is restored to its initial list of permanent symbols. If any of these have been redefined from the teleprinter, the original value is not restored. If any symbols were redefined by appending a user's symbol tape (see Appendix D), the original value will be restored.
ZERO$	All of memory to the bottom of DDT's symbol table is set to zero.

Loading the Program

DEBUG$	An object tape in FF format in the reader will be read into memory. The symbol definitions will be read and the symbols appended to DDT's symbol table, extending it down into memory. After the program is loaded, DDT will type out the address of the first free register above the program and of the lowest register in the symbol table.
LOAD$	An object tape in the reader will be read into memory. However, no new symbols will be entered into DDT's table. The typed information is as described for DEBUG$.
TABLE$	Only the symbol definitions on an object tape will be read as the tape passes through the reader. These symbols are appended to DDT's table, extending it down into memory. The typed information is as described for DEBUG$.

Mode Control

Information may be entered and typed out in several different forms. Instructions may be typed as symbols or absolute octal integers or as a combination of the two; register addresses may be typed as relative or absolute. DDT can be set to print out data in any one of these modes. The user can specify the mode best suited to the information he expects to examine. Input, however, is not restricted to one mode; the user may always use the representation most convenient.

The following commands determine the form in which DDT types out register addresses: (In these examples LOC is the tag for location 2642.)

OCTAL$	Conditions DDT to type out locations in absolute octal form:
	2662
RELAT$	Sets the mode to print location addresses relative to a program symbol, thus:
	LOC 20

These commands determine the mode in which the contents of registers are printed:

ABSOL$	Causes DDT to type out in absolute address form. That is, the instruction code is printed as a mnemonic, the address part as an octal integer:
	ADD 2653
CONST$	Causes DDT to print information as octal integers (constants), thus:
	302653
SYMBO$	Causes DDT to print information as symbolic expressions:
	ADD LOC 11

While operating in ABSOL$ or SYMBO$ modes, the user sometimes wants information in octal integer form. He can obtain this representation without leaving the current mode, by using the colon (:).

:	Causes the preceding expression typed by DDT or by the user to be printed as an octal integer. Its use is illustrated below.
	`LOC 20/      ADD 2653      :      302653`

Initial output modes are RELAT$ and SYMBO$.

Program Examination and Modification

These operations allow any register in memory, exclusive of DDT itself, to be examined and modified.

/	This is the register examination character. Typed immediately after the address of a register, it will cause DDT to open that register.

Example: When the user types

```
LOC 20/
```

DDT will immediately move to its next internal tab stop, print out the contents of the register LOC 20, and skip to the following tab stop. The resulting line might look like this:

```
LOC 20/      ADD 2653
```

If the user wishes to change the contents of the register, he types in the new information:

```
LOC 20/      ADD 2653      ADD 2607
```

Carriage Return

This causes DDT to close the opened register after placing any new information specified in it. If no modifications were typed before the carriage return, the register is closed unchanged.

Line Feed

This has the same effect as a carriage return, in that it closes an opened register. The next register in sequence is then opened. In the first example, typing a line feed would cause register LOC 21 to be opened, thus:

```
LOC 20/      ADD 2653      (lf)
LOC 21/      DAC 600
```

/ (alternate use)

The slash may be used to open a register addressed by the currently open unmodified register. If, in the first example, the user wished to examine the contents of location 2653, he would have typed a / instead of modifying information or a carriage return. The results may have looked like this:

```
LOC 20/      ADD 2653      /      ISZ 3062
```

Here, ISZ 3062 is the contents of register 2653, which is now open. The previously opened register, LOC 20, has been closed. The sequence of locations established by LOC 20 is not altered by the use of the slash.

```
LOC 20/      ADD 2653      /      ISZ 3062   (lf)
LOC 21/      DAC 600
```

)

Used without a preceding left parenthesis. A right parenthesis works like the alternate slash except that any modifications to the opened register will be made before that register is closed. DDT then opens the register addressed by the modified contents.

Example:

```
LOC 20/        ADD 2653        ADD 2607)     LOC 4202   :   7044
```

Here 7044 is the contents of register 2607.

Use of) does not alter the sequence of locations.

& This causes an opened register to be closed after modification, if any. The register which is now addressed by the contents is opened, establishing a new sequence. Substituting an ampersand for a right parenthesis in the example above would have the following result:

```
LOC 20/        ADD 2653        ADD 2607&
2607/          LOC 4202
```

A line feed would now result in:

```
LOC 20/        ADD 2653        ADD 2607&
2607/          LOC 4202    (lf)
2610/          DZM LOC 35
```

The line feed may be used at any time. DDT always remembers the last register to be opened in a sequence and will open the following register whenever a line feed is typed. Intervening carriage returns or other operations have no effect on the sequence.

Special Registers

There are several registers in the DDT program itself that may be opened and modified by the user; they are the only ones in DDT that may be so accessed. The tags for these registers can be used like any other symbols, remembering that $ is part of the name.

A$	Holds the C(AC) at the time a breakpoint occurs.
L$	Holds the C(L) at the time a breakpoint occurs (0 or 1).
B$	Contains the address in the user's program of the currently effective breakpoint.
F$	Contains the XOR instruction whose address is the lowest memory location occupied by DDT's symbol table.

M$ Contains the mask used in word searches. The two registers immediately following M$ contain the limits of the search. Initially, M$ contains 777777; M$+1 contains 0; and M$+2 contains 17777. Searching always terminates at the address specified by C(F$) or C(M$+2), whichever is smaller.

Running the Program Breakpoints and Traps

The operations in this group allow the user to control the running of his program by starting and interrupting it wherever he wishes.

k' This causes machine control to go to the location specified by the expression k. The C(A$) and the C(L$) are placed in the AC and link, respectively; if a breakpoint has been requested, it will be inserted. The most common use of ' is to start the user's program:

BGIN'

This causes program execution to start at location BGIN. The character ' may be used by itself to start the program at the location indicated in the start block of the last program tape loaded.

k" This causes DDT to insert a breakpoint at location k when control is passed to the user's program. When location k is encountered, the contents of k are saved, and a JMS to DDT is substituted. Instead of the instruction in k being executed when k is encountered, control returns to DDT. The C(AC) and the C(L) are placed in the registers A$ and L$, respectively. Through these registers, the AC and link may be modified before proceeding. The address of the break location is printed out, followed by a right parenthesis, tab, and the contents of the AC. The user may examine and modify his program, and then return control to it.

Since only one breakpoint may be in effect at one time, the break location may be removed simply by requesting a new one.

If " is typed without an argument, any existing breakpoint is removed.

RESTRICTIONS: The breakpoint will not work successfully with any program that operates in the program interrupt. Also the user must not place a breakpoint at an instruction which is modified during execution of the program, nor may he place a breakpoint at a subroutine call which is followed by arguments to be picked by the subroutine, since the user's call will be executed from within DDT. Note that one may successfully break on skips as well as a normal subroutine JMS, as long as the breakpoint is not moved until after exiting from the subroutine.

! After a breakpoint has occurred, this allows the user to continue his program from the point of the break. The contents of A$ and L$ are placed in the AC and link; and if a new breakpoint has been requested, it is inserted. The instruction substituted by the original breakpoint is executed, and the program continues.

Frequently, the user will want to insert a breakpoint in a loop in his program. If so, he probably will not want a break to occur every time the program passes through that location. He may delay the break until the program has encountered the break location a specified number of times by typing an expression before the !, thus:

250!

The break will not occur until the break location has been encountered 250 times.

Example:

LOC 30"	Request a breakpoint at LOC+30.
BGIN'	Start the program at BGIN.
LOC 30) 27305	DDT types the break location and the C(AC) when the break was encountered. LOC 30 is not opened.
...	
...	
!	After examination or modification, the program proceeds. The breakpoint is still at LOC 30 unless changed.

(etc.)

TRAP$ This is a special breakpoint used to trap CAL instructions. A break is inserted at location 21; if a CAL is executed during the operation of the user's program, DDT will trap it, print the location of the CAL instruction, and save the C(AC) and C(L). To continue the program properly, a JMP I 20 must be executed using k#. (See Miscellaneous Operations.)

Symbol Definition

Quite often, it is desirable to define new symbols for program uses, for instance in naming the first location of a patch. DDT will accept new definitions, appending each one to the lower end of the table. Similarly, any existing symbol may be redefined including those in the permanent table. Each definition requires three registers of memory.

, New location symbols may be defined with the comma in a way similar to that used in the assembler.

Example:

```
2663/          JMS 235          HERE,
```

The symbol HERE is assigned to location 2663. DDT types a tab to indicate that the symbol has been accepted; the register remains open.

(...) New symbols may be defined at other times using parentheses as follows:

LOC 30(NLOC)

The new symbol, NLOC, is assigned the value of the expression LOC 30, where LOC must have been previously defined.

Searching

These operations disclose if a word or operand address is or is not present in a given section of memory. They also allow a search for certain parts of a word (for instance, all ISZ operations, regardless of address). Using the mask in M$ and the limits in the succeeding two registers, the user may search any part of memory except that occupied by DDT itself. Only those word positions which correspond to those containing ones in M$ are considered in the search. The lower limit is determined by the C(M$+1); the upper limit, by the C(M$+2) or C(F$), whichever is smaller.

k WORD$ DDT will search for registers whose contents, masked by C(M$), have the value of the expression k. The location and contents of every such register are printed out.

If WORD$ is used without an argument, 0 (zero) is assumed.

k NOT$ Acts as WORD$ but searches for those registers whose contents when masked are not equal to k.

k ADDRE$ Causes DDT to search for those registers containing instructions whose effective address masked by C(M$) are equal to the expression k. For this purpose, indirect addressing chains are followed one level. (Because the defer bit is set in a law instruction, DDT computes an effective address for it. This occasionally causes an undersirable printout, which is immediately apparent.)

Assuming that M$ contains 777777, we wish to search for all registers between 500 and 1000 which contain the instruction LAC 650. First we set the limits; then we request the search.

Example:

```
M$ 1/              0               500 (lf)
MS 2/              17777           1000 (↲)
LAC 650 WORD$

501/               LAC 650
704/               LAC 650
```

To search the same section of memory for all isz instructions would now require a change of the mask.

```
M$/                777777          740000 (↲)
ISZ WORD$

555/               ISZ 1604
620/               ISZ 1107
727/               ISZ 1604
```

The mask causes only the instruction part of the words (leftmost four bits) to be examined during searching.

Miscellaneous Operations

k# The instruction k will be executed. If it is not a jump to some part of the user's program, control remains with DDT. Before execution, the contents of A$ and L$ are placed in the AC and the link; the AC and link are saved again after execution.

Example:

```
JMP I 20#          (to continue after a CAL trap.)
```

Q$ This symbol has the value of the previous quantity typed. Its usefulness is best illustrated by an example.

```
LOC 30/            ADD LIST 25
```

Suppose we wished only to change the address to LIST 24. Instead of typing the whole expression, Q$ can be used:

```
LOC 30/            ADD LIST 25     Q$-1 ↲
```

Used by itself, that is, not as part of a symbol, the period always refers to the last location opened by DDT in the normal sequence. In the example above, for instance, . would refer to register LOC 30 at the completion of the operations. A common use is illustrated:

```
LOC 20/            ADD 2653        ADD 2663 ↲
.-1/               LAC LIST 30
```

The contents of LOC 17 is LAC LIST 30.
The period, like the line feed, may be used at any time.

Punching Operations

The following commands enable the user to punch an object tape of his corrected program directly from DDT. The information in DDT is punched out in FF format arranged in blocks. (See PDP-7 Assembler, Digital 7-3-S for a further explanation.)

PUNCH$ A block of information is punched on tape. One or two arguments must be specified:

n PUNCH$	a one-word block is punched from register _n_
n;m PUNCH$	the contents of registers _n_ through _m_, inclusive, are punched

INPUT$ A termination block is punched to signal the end of input. When this block is encountered by the FF Loader the jump in the start block will be executed.

k START$ A start block consisting of a jump to register _k_ and an input block consisting of a jump to the FF Loader is punched on tape.

The above operations must be performed in the order, INPUT$, PUNCH$, START$, to produce a loadable tape. The tape is then loaded in the opposite direction; that is the last block punched is read first. Tapes punched by DDT can be loaded by it; otherwise, a FF Loader must be in memory.

Example:

To punch the contents of register 1277 to 1471, and the contents of register 143 to form a tape which will transfer control to location 1310 when loaded:

```
INPUT$
1277; 1471 PUNCH$
143 PUNCH$
1310 START$
```

To load the resulting tape through the FF Loader associated with DDT LOAD$ can be used. Otherwise the tape must be loaded by setting the AC Switches to 17770, the starting address of the FF Loader, and pressing START. To place the FF Loader in memory if it's not already there simply load any main program object tape.

APPENDIX A

SUMMARY OF COMMANDS

Character	Action
space	Separation character meaning arithmetic plus.
-	Separation character meaning arithmetic minus.
/	Register examination character: When following the address of a register, it causes the register to be opened and its contents printed. Immediately following a register printout, slash will cause the register addressed therein to be opened.
carriage return	Make modifications, if any, and close register.
line feed	Make modifications, if any, close register, and open next sequential register.
& (ampersand)	Make modifications, if any, and open addressed register. (Establishes a new sequence.)
: (colon)	Type last quantity as an octal integer.
. (period)	Current location.
k#	Execute the expression _k_ as an instruction.
Q$	Last _quantity_ typed out by DDT.
k,	Define the symbol _k_ as the tag of the currently open register.
)	Make modification and open addressed register. (The sequence is not changed.)
(...)	Define the enclosed symbol as the value preceding the (.
SYMBO$	Sets the mode in which DDT types out words to _symbolic_.
CONST$	Sets the mode in which DDT types out words to octal _constants_.
ABSOL$	Sets the mode in which DDT types out words to _absolute_. That is, the instruction code is typed as symbolic while the address is typed in octal.

Character	Action
RELAT$	Sets the mode in which DDT types out locations to relative (symbolic).
OCTAL$	Sets the mode in which DDT types out locations to octal.
N WORD$	Search for all occurrences masked with M$ of the expression N.
N NOT$	Search for all words not equal to the expression N after masking with M$.
N ADDRE$	Search for all words masked with M$ with the same effective address as N.
KILL$	Resets the symbol table to the initial list. Modified definitions are retained only if altered on line. Definitions added from a user's symbol table tape are restored to their original values.
ZERO$	Clears memory available to the user.
k" (double quote)	Insert a breakpoint at the location specified by k. If no address is specified, remove any breakpoint.
! (exclamation)	Proceed from a breakpoint.
k' (single quote)	Transfer control to the location specified by k, or to the address in the start block on tape if no address is specified.
LOAD$	Load a FF format tape (storage words only).
TABLE$	Load only the symbols from a FF format tape.
DEBUG$	Load both storage words and symbols.
N PUNCH$	Punch the contents of N.
N; M PUNCH$	Punch N to M, inclusive.
INPUT$	Punch the input block.
START$	Punch a start block.
TRAP$	Place a trap location at 21 for CAL.

The following symbols are the address tags of certain registers in DDT whose contents are available to the user.

A$	Accumulator storage (at breakpoints).
L$	Link storage (at breakpoints).
M$	Mask used in search; M$+1 and M$+2 contain first and last address of the area to be searched.

Character	Action
F$	Contains the lower limit of DDT as the address part of an XOR instruction.
B$	Contains the current breakpoint location.

APPENDIX B

BASIC SYMBOLS

The DDT-7 tape labeled DDT-7 (Basic Symbols) contains the following symbols in its permanent symbol table.

DAC	040000
JMS	100000
DZM	140000
LAC	200000
XOR	240000
ADD	300000
TAD	340000
XCT	400000
ISZ	440000
AND	500000
SAD	540000
JMP	600000
IOT	700000
OPR	740000
CAL	0
LAW	760000
LAM	777777
I	020000
NOP	740000
CLA	750000
CLL	744000
CMA	740001
CML	740002
CLC	750001
CCL	744002
RAL	740010
RAR	740020
RTL	742010
RTR	742020
RCR	744020
RCL	744010
OAS	740004
LAS	750004
LAT	750004
HLT	740040
SPA	741100
SMA	740100
SPL	741400
SML	740400
SZA	740200
SNA	741200
SKP	741000
SZL	741400
SNL	740400
GLK	750010
STL	744002
XX	740040

Interrupt

IOF	700002
ION	700042
ITON	700062
CAF	703302

I/O States

IORS	700314
SKP7	703341

Clock

CLSF	700001
CLOF	700004
CLON	700044

Perforated Tape Reader

RSF	700101
RSA	700104
RSB	700144
RRB	700112
RCF	700102

Tape Punch

PSF	700201
PLS	700206
PCF	700202
PSA	700204
PSB	700244

Keyboard

KSF	700301
KRB	700312

Teleprinter

TSF	700401
TLS	700406
TCF	700402
TTS	703301

Display 30D

DXL	700506
DXS	700546
DYL	700606
DYS	700646
DLB	700706
DXC	700502
DYC	700602

Light Pen Type 370

DSF	700501
DCF	700601

APPENDIX C

EXTENDED SYMBOLS

The DDT-7 tape labeled DDT-7 (Extended Symbols) contains the following symbols in its permanent table, in addition to the basic symbols.

Type 57A Mag Tape

MTS	707006
MTC	707106
MCD	707042
MNC	707152
MRC	707244
MRD	707204
MTRS	707314
MCEF	707322
MEEF	707342
MIEF	707362
MCWF	707222
MEWF	707242
MIWF	707262
MSEF	707301
MSWF	707201
MSCR	707001
MSUR	707101
MCC	707401
MCA	707405
MWC	707402
MRCA	707414
MDEF	707302
MDWF	707202

Card Punch

CPSF	706401
CPSE	706444
CPLR	706406
CPCF	706442

Symbol Generator Type 33

GPL	701002
GPR	701042
GLF	701004
GSF	701001
GCL	700641
GSP	701034

Card Reader

CRSF	706701
CRSA	706704
CRSB	706744
CRRB	706712

Line Printer

LPSF	706501
LPCF	706502
LPLD	706542
LPSE	706506
LSSF	706601
LSCF	706602
LSLS	706606
LPD-1	706504
LPB-2	706524
LPB-3	706544
PRI	706604
PAS	706624

DECtape

MMRD	707512
MMWR	707504
MMSE	707644
MMLC	707604
MMRS	707612
MMDF	707501
MMBF	707601
MMEF	707541

Extended Arithmetic Element Type 177

EAE	640000
LRS	640500
LRSS	660500
LLS	640600
LLSS	660600
ALS	640700
ALSS	660700
NORM	640444
NORMS	660444
MUL	653122
MULS	657122
DIV	640323
DIVS	644323
IDIV	653323
IDIVS	657323
FRDIV	650323
FRDIVS	654323
LACQ	641002
LACS	641001
CLQ	650000
ABS	644000
GSM	664000
OSC	640001
OMQ	640002
CMQ	640004

Automatic Priority Interrupt Type 172

CAC	705501
ASC	705502
DSC	705604
EPI	700004
DPI	700044
ISC	705504
DBR	705601

Precision Incremental Display Type 340

IDLA	700606
IDSE	700501
IDSI	700601
IDSP	700701
IDRS	700504
IDRD	700614
IDRA	700512
IDRC	700712
IDCF	700704

Memory Extension Control Type 148

SEM	707701
EEM	707702
LEM	707704
EMIR	707742

Serial Drum Type 24

DRLR	706006
DRLW	706046
DRSS	706106
DRCS	706204
DRSF	706101
DRSN	706201
DRCF	706102

Multiplexer Control Type 139

ADSM	701103
ADIM	701201

A-to-D Converter Type 138B

ADSC	701304
ADRB	701312
ADSF	701301

APPENDIX D

ADDING SYMBOLS TO DDT'S PERMANENT TABLE

Symbols may be added to DDT's permanent table in the following manner.

1. Prepare a symbolic tape consisting of the definitions of the desired symbols. For example:

   ```
   DEFINE ABC AND DEF
   ABC = 47132
   DEF = -4
   START
   ```

2. Assemble the tape normally. In particular, do not put AC switches 0 and 2 up, as this would suppress punching of symbols.

3. Remove the title, loader, and the 3-word binary block which follows the loader from the beginning of resulting tape.

4. Remove the 2-word binary start block from the end of the binary tape of DDT.

5. Splice the torn edges together.

DDT's entire permanent symbol table may be replaced by performing the same process except removing the entire last data block from the DDT tape at step 4.

digital
EQUIPMENT
CORPORATION
MAYNARD, MASSACHUSETTS

5228 PRINTED IN U.S.A. 20-1/65