DEC-08-CDDA-D

DDT PROGRAMMING MANUAL

PDP-8

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

DEC-08-CDDA-D

PDP-8

DDT PROGRAMMING MANUAL

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

PREFACE

The programs discussed in this manual, though written on the Programmed Data Processor-8 computer, can also be used without change on Digital's Programmed Data Processor-5. This compatability between the libraries of the two computers results in four major advantages:

1. The PDP-8 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-8 programming system takes advantage of the many man-years of field testing by PDP-5 users.

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

4. Programs written by users of the PDP-5 and submitted to the users' library (DECUS Digital Equipment Corporation Users' Society) are immediately available to PDP-8 users.

CONTENTS

CHAPTER 1

INTRODUCTION

Users of most computers, especially large-scale ones, are familiar with the procedure of submitting a new program for a computer run, waiting for it to be processed (which may take anywhere from a few hours to several days), and finally receiving the compilation and/or assembly listings, a list or dump of the contents of each core memory cell at the time the run was terminated, and perhaps a storage map giving the addresses of the symbols used in the programs. The user may get a few remarks from the computer operator regarding the failure of the program to run properly. If the user is present in the machine room 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 computer. 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 make corrections to the program or its data. For example, tracking down a subtle error in a complex section of coding is a laborious and frustrating job by hand; but with the breakpoint facility of DDT-8, the user can interrupt the operation of his program at any point and examine the state of the machine. In this way, sources of trouble can be located quickly.

Using DDT-8, the programmer can make corrections or insert patches in his program and try them out immediately. If his corrections work, the user can have the corrected sections and patches punched out on the spot in the form of tapes which can be loaded along with the program the next time it is run, thus eliminating the necessity of creating new symbolic tapes and reassembling or recompiling each time an error is found.

When working with DDT-8, the user has with him a listing of his program and of his symbols. In making corrections, he may refer to variables and tags by their symbolic names or by their octal values; he may add new symbols and delete old ones (for purposes of debugging, symbol

changes do not carry beyond the immediate debugging run). If he writes in the permanent corrections on his program listing, he can keep a record of the debugging and eventually make a new symbolic tape incorporating all his corrections and patches.

The first chapter of this manual explains a typical debugging run. The succeeding chapters describe the functions of DDT-8 in detail.

CHAPTER 2

USE OF DDT-8

This chapter is designed to introduce the basic operations of DDT-8 to the person unfamiliar with on-line debugging. Although some elementary concepts are explained in detail, any essential information which appears in this chapter about the DDT commands is also presented in Chapter 3, where it is accessible for easy reference.

Debugging a new program can be, and has been, done on varying levels of detail and sophistication. On the crudest level, the programmer can simply load the program and let it run until it stops unexpectedly. Then, using the console switches and keys, he can try to find the error(s) by interpreting the console lights. There are two hazards to this approach. First, by the time the program stops, the error may have caused all pertinent information, including itself, to be altered or eliminated; the program may have stopped by the simple process of self-destruction. Second, the program may not stop at all; it might continue to run in an infinite loop. Such loops are not always easy to detect.

If the programmer plans his debugging attack beforehand, he can, using the computer console, place strategic halts in his program before starting it. After each halt, he can examine various registers and alter their contents , again using the console. As long as he remembers to replace each halt with the original instruction before proceeding, he might find sources of error more readily than if he just let the program run on. However, errors seldom appear where expected, so a strategic halt may be of little real use.

Added to these problems (of console debugging) are the difficulties of interpreting binary console displays and translating them into symbolic expressions related to the user's program listing. Further, adding corrections to a program in the form of patches requires seemingly endless manipulation of switches and keys. In all this, the chance of programmer error at the console is large and is likely to obscure any real gain made from the debugging.

What is needed is a program which will assume the tasks which the programmer would have to perform if he used the console. Such a program would allow the user to examine registers, change their contents, and make corrections, without having to manipulate switches and keys.

It would allow the easy placement and removal of halts, even to the automatic restoration and execution of instructions which the halts had displaced. Most important, it would allow the programmer, using the keyboard and printer, to do all examination and to make all corrections in the symbolic language of the listing; the debugging program would perform all the necessary translation to and from the binary representation.

DDT-8 is such a debugging program. Descended from a line of programs that includes a version for every computer produced by DEC, it performs all the tasks described above, and many more, making the programmer's burden light enough to allow him to concentrate on the actual correction of his program.

PREPARATIONS FOR A DEBUGGING RUN

By the time the programmer is ready to start debugging a new program at the computer, he should have the following materials:

1. The binary object tape of the program.
2. The symbol definition tape which was part of the assembly output.
3. A complete symbolic listing of the program.
4. A list of the symbols and their definitions.
5. A binary tape of DDT-8 (usually provided at the console).

To begin the debugging run, first ascertain that the BIN Loader is in memory. If it is not, load it using the procedure described on page 139 of the PDP-8 Users' Handbook (F-85), for loading RIM tapes. When the BIN Loader is in the computer, load the programmer's binary program tape(s) using the same BIN loading procedure. After it has been read successfully, load the DDT-8 binary tape.

> In the machine, there now is: 1) the DDT program, which occupies upper memory between registers 5240 and 7600, inclusive; 2) the User's programs which must not overlap the area occupied by DDT-8 or its permanent symbol table; 3) a table of symbol definitions, extending downward from location

5240 to 5000. This table includes the definitions for all of the PDP-8 memory reference instructions, operate class instructions, the ten basic IOT instructions, and the combined operations CIA and LAS.

Since DDT is to perform all translation between binary and symbolic representation, it must have access in memory to the user's symbol definitions. To load a symbol tape, perform the following steps. Note that the high-speed reader may not be used to load symbol tapes. Only the reader on the ASR 33 console may be used.

1. Turn the reader off; insert the symbol definition tape.

2. Type the characters, ALT MODE and R ([R), in that order, and then turn the reader on.

3. When the computer stops, turn off the reader; press CONTINUE. DDT will type out the address of the lowest register in memory which is occupied by a symbol definition.

The symbols for the user's program are stored in memory immediately below DDT's permanent table. These symbols, and any others which are entered from the console, comprise the external symbol table; these definitions may be removed at any time (see Chapter 3) without harming the permanent table.

With DDT, the program, and the symbol definitions now in memory, the programmer is ready to begin debugging. Figure 2-1 is a listing of a program ready for debugging; the remainder of this chapter will describe the process.

THE BASIC FUNCTIONS OF DDT: A SAMPLE RUN

As soon as DDT has typed the address of the lowest extension of the symbol table, it is ready for debugging work. The program to be checked out is a subroutine which accumulates the sum of the first n integers. For testing purposes, a short calling sequence has been included which provides the integer limit of the sum as an item of data. The first task is to place an integer in the register which holds this datum; namely, the register labeled INT in the calling

program. By typing the address of the register (which in this case can be done by typing the address tag), followed by a slash, the user indicates to DDT that he wishes to examine the contents of that register. Thus he types:

INT/

```
/INTEGER SUMMATION SUBROUTINE
INTSUM,     0
            CLA
            TAD I INTSUM        /GET DATA
            DCA N
            DCA PSUM
LOOP,       TAD PSUM            /MAIN COMPUTATION
            TAD N
            DCA PSUM
            ISZ N               /DECREMENT INDEX
            JMP LOOP            /NOT FINISHED
            TAD PSUM            /FINISHED.  PUT RESULT IN AC.
            ISZ INTSUM
IEXIT,      JMP I INTSUM        /RETURN

N,          0
PSUM,       0

*400

ITEST,      CLA                 /INTSUM TEST PROGRAM
            JMS INTSUM
INT,        0                   /PUT ARGUMENT HERE
RTN,        HLT

$
```

Figure 2-1 DDT Sample Program

DDT responds to the typing of the slash by typing an expression which has the value of the contents of the specified register. In this case, C(INT) = 0, and the line now appears as follows: (Note: In all of the examples below, information typed by DDT is underlined to make it distinguishable from that typed by the programmer. In actual operation, no underlining is present.)

INT/0000

After typing the contents of the register, DDT types five spaces and waits. The register is now open, which means that its contents are available for modification. The programmer decides

that the first test integer is 10. This must be an octal integer since DDT performs no decimal arithmetic. With the register open, he types the number 10. Then, to close the register, he types a Carriage Return (↵) immediately after the number.

INT/0000 10↵

Further access to this register is now denied until he opens it again.

Having provided his data, the programmer is ready to start the program. If it works, it should stop almost immediately with the sum of the first 10_8 integers, which is 44_8, displayed in the AC lights. To start the program from DDT, he types the following command.

ITEST[G

The left bracket ([) is the character printed when the ALT MODE key is struck; its function here is to identify the succeeding character as a DDT command. The letter G specifies the action to be performed, which in this case causes DDT to transfer control to the test program at location ITEST.

The programmer has typed the command; his program starts to function. Immediately he observes that something is wrong, since it does not stop almost instantaneously, but runs for a very short, but observable, time. When it does halt, the contents of the AC lights are definitely not equal to 44_8.

At this point, he knows something is wrong, but he is not sure where the error lies. If he could interrupt the program during its operation, he might get some idea of the nature of the difficulty. For instance, if he could verify that the data was transferred to the subroutine correctly, he could eliminate the calling sequence as a source of error.

The DDT facility which allows the programmer to interrupt the program at any time is called the breakpoint. As its name implies, it allows him to break into the program sequence at some point and return control to DDT. He can specify a breakpoint by typing the address of the instruction where he wants to interrupt the sequence; and after this address he types the breakpoint command. If he requests a breakpoint at location INTSUM+3, the program will be interrupted when the datum is in the AC, but before it is deposited in the working register N.

INTSUM+3[B

When this command is given, the information is retained by DDT until the start command is provided. At that time, the instruction in the register specified is removed and placed in a temporary storage location in DDT. In place of this instruction, a JMP is substituted which returns control to DDT.

To ascertain that the error did not destroy the item of data in the calling program, check it by opening the register.

INT/0010

Having ascertained that the datum is correct, again start the program.

ITEST[G

Almost immediately, the breakpoint is encountered. Control returns to DDT. When the break occurs, DDT saves the C(AC). It then types the address of the breakpoint, a right parentheses, and the contents of the AC which have been saved.

INTSUM+0003)0010

The programmer sees that the transfer is correct.

In similar fashion, he moves the breakpoint to the end of the subroutine at the location IEXIT/. He discovers that at this point the error has manifested itself. He knows now that the trouble is in the initialization or the main loop. He can investigate the loop by placing a breakpoint at LOOP, to discover that the datum is placed in register N as desired. Now he moves the breakpoint to the end of the loop.

LOOP+3[B
ITEST[G
LOOP+0003)0010

At the end of the first pass through the loop, the C(AC) are equal to the starting value of N. At this point, however, the C(N) itself have just been changed. If the subroutine is working properly, the C(N) should now be equal to 7_8. He investigates:

N/0011

By this time the programmer realizes what has been wrong with the program. In attempting to save space by using the datum as a counting index, he forgot that the ISZ instruction increments the contents of a register. What he needs is a counter that starts with a negative value. Realizing this, he ends the debugging run.

The sample program above was simple; the error was obvious. This is seldom the case; however, and with long or complex programs, several debugging runs may be required. However, DDT, with its facilities for handling symbolic expressions, allows the programmer to work entirely in the language of the Assembler (either MACRO-8 or PAL II), thus shortening the time required to arrive at a correct, workable program.

A detailed explanation of every function of DDT is provided in the next chapter. For those interested, a correct subroutine for performing the integer summation will be found in the PAL II Program Manual.

CHAPTER 3

THE FUNCTIONS OF DDT-8

STORAGE REQUIREMENTS

The operating portion of DDT-8 occupies storage in upper memory from location 5245 to location 7577, inclusive. The permanent symbol table extends downward in memory from location 5237 to location 5000, inclusive. This table contains the definitions of the mnemonics for all the basic memory reference instructions, the operate class instructions of both Group 1 and Group 2, the combined instruction CIA and LAS and the symbol I for indirect addressing, and the basic IOT instructions: KCC, KRS, KRB, KSF; TSF, TCF, TPC, TLS; ION and IOF. Appendix 2 lists all the symbols and definitions in the permanent table.

Space is reserved for the user's symbol table immediately below the permanent table. A maximum of 250 such external symbols is allowed; hence if the user's table is filled, the lower limit of space occupied by DDT is 3030. However, space not used for external symbols is available to the user. Each new symbol defined on line uses four locations in the external table.

During operation, DDT uses location 4 on page 0 for the breakpoint link; thus this register is not available to the user.

LOADING PROCEDURE

To load DDT, the BIN Loader must be in memory. Place the user's binary tape(s) in the reader, set the switches to 7777, then press LOAD ADDRESS and START in that order. When the tape has been read, the status of the AC lights will indicate any error in loading. If the lights are all out, loading was successful; if any lights are on, there was a checksum error and the tape must be reread.*

After the user's binary tape(s) is in memory, DDT may be loaded using the BIN Loader.

---

*For more information about the BIN Loader and binary tape format, see PDP-8 Users' Handbook, pp. 139-40.

SYMBOL TABLE TAPES

Part of the punched output of a MACRO-8 or PAL assembly is a tape containing the symbol definitions of the assembled program. The definitions from a symbol tape are entered into the DDT external table by the following procedure; only the ASR 33 tape reader may be used.

1. Turn the reader off; insert the symbol tape.

2. Type ALT MODE, then R ([R) on the keyboard and turn the reader on.

3. When the computer stops, turn off the reader, then press CONTINUE. DDT will type out the address of the lowest register used by the external symbol table.

4. If more tapes are to be entered, bit 0 of the switch register (SR) must be down; repeat steps 1-3, for each tape.

Reading will continue until the end of the tape is reached or until a total of 250 symbol definitions have been read. If this maximum limit is reached, no further symbols may be added to the table until some have been deleted. Even if the limit is reached in the middle of a tape, however, the user may proceed with debugging by typing EOT, then turning the reader off and pressing CONTINUE. The remaining symbols left unread will not be in the table.

DEFINITIONS

A symbol is a string of up to six letters and numerals, the first of which must be a letter. The following are legal symbols: FIMAGE, K2, X464PQ, PMLA. The following are not aceptable:

4WD	Does not begin with a letter
F2.8	Contains an illegal character
AN PRC	A space cannot be imbedded in a symbol
GANDALF	More than six characters

A number is a string of up to four octal digits (integers). Hence, a number may have a maximum value of 7777_8. The digits 8 and 9, however, may be used only as characters in a symbol.

An expression is a symbol, an integer, or a sequence of symbols and integers separated by any of the following operators:

+	An operator designating addition (arithmetic plus).
-	An operator designating subraction (arithmetic minus).
space	An operator which indicates that the remainder of the expression is to be treated as the address part of an instruction (see the MACRO-8 User's Manual.)

All other characters, except those used for DDT control commands, are illegal.

If two or more spaces appear in succession, all but the first are ignored. Thus, TAD TEM and TAD TEM are identical expressions.

DDT will respond to an extra CR with CR, LF; the extra CR's are otherwise ignored.

The following errors will cause DDT to type a question mark (?) and ignore all the information typed between the point of the error and the previous tab or CR.

1. Undefined symbol; illegal symbol.
2. Illegal character.
3. Undefined control command.
4. Cross-page addressing.

MODE CONTROL

Any expression containing a symbol is symbolic; an expression containing only integers is octal. The user of DDT is free to use whichever mode is most convenient for the information he is typing in. On output, DDT will type exclusively in one mode or the other, as determined by one of the commands described below.

> NOTE: When DDT is first set into operation, the output mode is symbolic.

[0 This command causes DDT to print any subsequent item of information as an octal integer. Typed input may be symbolic or octal. If LOC=2642:

Example:

1. LOC/1263
2. 2642/1263

[S This command causes DDT to print any subsequent item of information as a symbolic expression. Typed input may be symbolic or octal. If LOC=2642 and C(LOC)= TAD DATA+4:

Example:

1. LOC/TAD DATA+0004
2. 2642/TAD DATA+0004

If the user wishes to find the octal value of a symbolic expression typed by himself or by DDT without changing the prevailing output mode, he may use the following command.

= Typed immediately after a symbolic expression, this will cause DDT to print the value of the expression as an octal integer.

Example:

1. LOC=2642
2. LOC/TAD DATA+0004 =1263

In the second example above, the prevailing output mode is symbolic and remains so after the use of the equal sign.

PROGRAM EXAMINATION AND MODIFICATION

These commands and operations allow the user to examine and change the contents of any register in the PDP-8 core memory.

CAUTION

Be careful not to open and modify any register within the DDT symbol table or program itself. DDT does not protect itself against such intrusions, which will inevitably cause errors in operation.

/ This is the register examination character. Typed immediately after an expression, it causes DDT to print the contents of the register whose address is specified by that expression.

Example: If the user types:

LOC/

DDT will type out the contents of LOC, followed by 5 spaces, thus:

LOC/TAD DATA+0004

The user may now change the contents of the register if he wishes:

LOC/TAD DATA+0004 JMP LOC+10

) (CR) This causes DDT to close the opened register after making the specified changes (if any) in its contents.

Example:

LOC/TAD DATA+0004 JMP LOC+10)

Typing additional CR's will have no effect on the operation of DDT.

LF If, after examining and/or modifying the contents of a register, the user wishes to open the next register in sequence, he types a line feed instead of a CR. The open register is

closed, and DDT then opens the succeeding register, typing the address, a back slash to indicate that the register was not opened by the user, the contents of the new register, and another five spaces.

Example: After examining and changing the contents of LOC, the user wishes to examine the contents of LOC+1.

LOC/TAD DATA+0004 JMP LOC+10 (LF)
LOC+1\ DCA DATA

The register LOC+1 is now open.

The line feed may be used at any time, even if the last register examined has been closed or if other operations have intervened. For example, if the following sequence of operations occurs:

LOC/TAD DATA+0004 JMP .+10)
[0
.+5[B
(LF)

DDT will still open register LOC+1. The breakpoint address has no effect on the counter within DDT which keeps track of the last opened register.

If instead of changing the contents of a register, the user wishes to examine the register addressed by those contents, he types ↑, as follows:

LOC/TAD DATA+0004 ↑
DATA+4\ OPR+337

The register DATA+4 is now open.

Note that this operation is intended for use with unmodified registers. If the user types it after typing some modifying information, the register addressed will be the one which is changed. For example, if the following sequence occurs:

LOC/TAD DATA+0004 JMP LOC+10↑

the information will be placed in DATA+4, so that the next line, printed by DDT, will look like this:

DATA+4\JMP LOC+0010

The register LOC will not be changed.

An indirect address modifier will not be interpreted by the ↑ operation. If, for example, the register LOC contained TAD I DATA+4, and the user typed ↑ as in the previous example, DDT would still open the register DATA+4.

. (period) The period is used as a symbol whose value is the address of the last previous register opened. It can be used in several ways.

Example 1: To check the results of a modification.

LOC/TAD DATA+0004 JMP LOC+10)
./JMP LOC+0010

2. To refer to the currently open register.

LOC/TAD DATA+0004 JMP .+10

3. To execute any command starting at an address relative to the last opened register.

LOC/TAD DATA+0004 JMP.+10)
.-5[G

← (back arrow) An error may be deleted by typing a back arrow. All information between the ← and the previous tab or CR is ignored; DDT responds by typing a tab. For example:

LOC/TAD DATA+0004 JMP LC←_____JMP .+10

CROSS-PAGE ADDRESSES

When the user types an instruction to be placed in an open register, the address of that instruction must be in the same page as the address which contains the open register.
If such a cross-page address is attempted, DDT will signal an error by typing ? and ignoring the information.

Example: If LOC = 2642 and XPAG =2770, the following sequence would result in an error indication:

LOC/TAD DATA+0004 DCA XPAG+20)
?

The expression XPAG+20 = 3010, which is outside the page containing LOC.

The register LOC will be closed without modification.

Conversely, an expression containing symbols defined outside the page is acceptable if its value is in the current page.

Example: If LOC = 2642 and XPEG = 3010, the following sequence is acceptable, since XPEG-20 has a value which brings it within the current page:

LOC/TAD DATA+0004 DCA XPEG-20)

USING COMBINED OPERATE OR IOT CLASS INSTRUCTIONS

Except for CIA and LAS, combined Operate Class and IOT instructions are not defined in the DDT-8 permanent symbol table. To enter such instructions into an open register, the combination must contain no more than two mnemonics, the second of which must be CLA. Any other combination will be treated as an error, and the information will be ignored.

Example: The following attempt is an error.

```
XPAG/CLA      CLA CMA)
?
```

This attempt is correct.

```
XPAG/CLA      CMA CLA
```

If the desired combination does not include CLA, the user may do one of two things. He can define the combined operation as a new symbol (see Symbol Definition) whose value is the combined operation code. For example, the operation CLL RAR can be defined as a symbol, say, CLAR, whose value is 7110.

Alternatively, the user may enter the combined operation as an expression containing the symbol OPR. For example, the operation CLL RAR can be entered as OPR+110. He may similarly use the symbol IOT in entering new I/O combinations.

OUTPUT

When operating in symbolic mode, DDT-8 will always attempt to make a symbolic expression out of the contents of an opened register, regardless of whether the contents are intended to be such or not. For example, if register DATA contains the number 6115, opening the register will result in the following line:

```
DATA/IOT+0115
```

The user can use the equal sign to ascertain the octal value:

```
DATA/IOT+0115      =6115
```

SPECIAL REGISTERS

There are five registers contained within DDT which hold information of interest to the user. These registers may be opened and their contents may be changed.

[A	When a breakpoint is encountered, the C(AC) at that point are placed in this register.
[Y	When a breakpoint is encountered, the C(L) at that point are placed in this register.

[L — This register contains the address of the lower limit of a word search. Initially, C([L) = 0001.

[U — This register contains the address of the upper limit of a word search. Initially, C([U) = 5000.

[M — This register contains the mask used in a word search. Initially, C([M) = 7777.

PROGRAM EXECUTION AND CONTROL

The commands described in this section allow the user to control the execution of his program.

k[G — This command causes DDT to begin the execution of the user's program, starting with the instruction in the register whose address is specified by the expression k. If a breakpoint (see below) has been requested, it is inserted just before control is passed to the user's program.

Example: If the user types

BGIN[G

DDT will transfer control to location BGIN.

Likewise,

FILI-5[G

will cause the user's program to start in the fifth register preceding the one labeled FILI.

Using [G without an argument is an error. DDT will ignore the command, and type ? to indicate the mistake.

k[B — This causes DDT to insert a breakpoint at the location specified by the expression k. The breakpoint is not placed immediately, however. When this command is typed, DDT

stores the value of the address indicated by k. Then, when the user next types either a [G or a [C (see below) command, the breakpoint is placed just before control passes to the user's program. At that time the sequence of operations performed by DDT is as follows:

1. The contents of location k are saved in a special register.

2. In place of the instruction in location k, DDT substitutes the instruction, JMP I 4. Location 4 contains the address of a special breakpoint handling subroutine within DDT.

3. After the breakpoint has been placed, DDT passes control to the user's program.

When, during execution, the user's program encounters the location containing the breakpoint, control is immediately passed (via location 4) to the breakpoint subroutine in DDT. The C(AC) and C(L) at the point of interruption are saved in the special registers [A and [Y, respectively. DDT then types out the address of the register containing the breakpoint, followed by a right parentheses and the contents of A as an octal number. Control has now returned to DDT, and the user is free to examine and modify his program.

Only one breakpoint may be in effect at one time. As soon as the user requests a new breakpoint using the B command, any previous existing breakpoint is removed. To eliminate the breakpoint entirely, the command is typed without an argument, thus:

[B

When the breakpoint is removed, the original contents of the break location are restored.

After the breakpoint has occurred and the user has examined his program and made the changes he wishes, he can cause his program to continue from the point of the break by means of the following command:

[C This continue command causes DDT first to execute the instruction which was originally in the break location, and then pass control to the next location in the user's program. The breakpoint remains in effect.

Example: This example illustrates the use of the three commands just described. The comments explain the events.

FILI+7[B	Breakpoint request.
BGIN[G	Program execution is initiated at BGIN. Program runs until breakpoint location is encountered.
FILI+0007)7721)	DDT types the address of the break location and the contents of the AC at the time of the break. Note that location FILI+7 is not opened.
...	The user performs such examination and modification as he
...	desires.
[C	The user's program continues, beginning with the execution of the instruction originally in FILI+7. The breakpoint remains in effect.

Oftentimes, the user would like to place a breakpoint at a location within a loop in his program. Since loops can run to thousands of repetitions, some means must be available to prevent a break from occurring every time the location is encountered. This is done using the [C command; after the breakpoint is encountered the first time, the user can specify how many times the loop must be executed before another break is to occur, as follows:

Example: After the first breakpoint occurrence, the user wishes to wait for 250_8 repetitions before the next break.

FILI+7[B	The break is requested.
BGIN[G	The break is placed; the program begins.
FILI+0007)7721)	The first break occurs.
250[C	The program continues. The next break will not occur until the location FILI+7 has been encountered 250 times.

FILI+0007)2534) The next break occurs after 250 times through the loop.

Restrictions on the Use of Breakpoints

The user must not place a breakpoint at any of the following places in his program:

1. Within any section of the program which operates with the program interrupt enabled.

2. At any location that contains an instruction which is modified during the course of the program. For example, if the program contains a sequence which includes the following instructions:

```
        ...
        ISZ B
        ...
        ...
B,      TAD A
        ...
```

a breakpoint may not be inserted at location B.

When the user's program comes to a halt, control may be returned to DDT by setting the Switch Register to 5400 and pressing LOAD ADDRESS and START, in that order.

3. In a register containing a subroutine jump (JMS) which is followed by one or more arguments for that subroutine.

A breakpoint may be inserted at the point of a subroutine call if the JMS instruction is not followed by any subroutine arguments, but the breakpoint may not be removed until control has returned from the subroutine to the calling program.

WORD SEARCHES

The searching operations are used to determine if a given quantity is present in any of the registers of a particular section of memory. The search is initiated by the following command:

k[W — DDT will perform a word search and print the address and contents of every register in the desired section of memory whose contents are equal to the value of the expression k. If the expression k is omitted, a search for the quantity 0000 masked by C([M) is assumed.

The conditions for any search are set by the following criteria:

1. The contents of every register searched are masked by the contents of the special register M, using the Boolean AND operation. The resulting logical product is then compared with the value of k. If the two quantities are identical, the address and contents of the examined register are printed on the teletype-writer.

2. The search is conducted over that section of memory whose lower limit is given by the C([L), and whose upper limit is given by the C([U), except for the special case described in the next paragraph.

3. If the C([M) = 7777 and the expression k contains any symbol in its address part (for instance, ISZ FILI+5; FILI is the symbol), the search will be conducted only on the page for which that symbol is defined, regardless of the search limits specified by C([L) and C([U). For any other case, including that where the address tag of k is defined for page 0, the search is conducted according to the limits set.

A search never alters the contents of any register examined.

Addresses and register contents are printed as symbolic expressions or octal integers, according to the prevailing mode at the time of the search.

Example: Search locations 2600 to 3000 for all occurrences of the expression TAD DATA, where DATA = 2740. The C([M) = 7777. The C([L) and C([U) are at their initial values.

```
[L\0001    2600 )
[U\5000    3000 )
TAD DATA [W

LOC\TAD DATA
LOC+0015\TAD DATA
FILI+0002\TAD DATA
```

Note that in this example, the C([L) and C([U) could have been left alone, since the expression k contained the symbol DATA in the address part. Had the user requested a search for the expression TAD 2740, he would have had to set the limits as shown for the desired search.

Example: Search locations 2000 to 4000 for all occurrences of an ISZ instruction.

```
[L\0001    2000 )
[U\5000    4000 )
[M\7777    7000 )
ISZ[W
```

The addresses rather than tags are typed out when symbols are not defined.

```
2002\ISZ 2135
2053\ISZ 2135
2111\ISZ 0017
...
(etc.)
```

The search will continue until all registers containing an ISZ are found. Note that the setting of the mask limits the investigation to the first three bits of each register, so that only instruction codes are considered.

Example: Obtain a dump of any section of memory. The search is conducted between the limits set, and the addresses and contents of all registers in the searched section are printed.

```
[L\0001    2600 )
[U\5065    3000 )
[M\7777    0    )
[W
```

2600\ 0000
2601\ CLA
2602\ TAD 2610
...
(etc.)

The search will continue to the specified limit, printing the contents of every register. Note the following points: The mask is set to 0 to insure that results of every comparison are the same, i.e., 0. The search is conducted for all registers containing 0, so that the results of each comparison are equal to the desired quantity, 0. Always remember that the contents of the registers themselves are not altered.

DEFINING NEW SYMBOLS

Often, during the course of a debugging run, the user will want to add new symbols to the external table. This is especially so when he adds a sequence of instructions to his program as a patch elsewhere in memory. The patch is usually identified by a symbol which is the address tag of the first instruction in the patch. In order to use the symbol in subsequent debugging operations, he must add its definition to the external table as follows:

1. Set bit 0 of Switch Register down.

2. Type [R (ALT MODE, R).

3. Type carriage return, line feed, in that order.

4. Type the symbol, at least one space, and an octal integer whose value is the definition of the symbol.

5. If more than one symbol is to be defined, repeat steps 3 and 4 for each definition.

6. After the last definition, type carriage return, line feed, EOT, in that order.

7. Press CONTINUE. DDT will type out the new lower limit of the external table.

Example: To define the symbols PATCH1 and PATCH2, the operations will appear as follows (Assume that the current limit of the table is 4775):

↓ (If)
PATCH1 610 ↓ (If)
PATCH2 620 ↓ (If)
(EOT)

(Press CONTINUE)

4665 (new limit of the table)

If the user makes an error while typing a definition, he cannot use ← to eliminate the information. The erroneous definition must be entered.

A symbol already in the table may not be redefined. Only new symbols can be added.

> NOTE: Extra carriage return, line feed pairs may not be inserted between definitions; they will cause errors in subsequent table look-up when DDT is operating.

To completely expunge the external symbol table (for instance, when starting a new debugging run with DDT already in memory), the following command is used:

[X

On receipt of this command, DDT removes all definitions in the external table. The permanent table is unaffected.

MAKING A NEW SYMBOL TAPE

DDT may be used to make a new symbol definition tape. If a number of new symbols have been defined in the course of a debugging run, the user can put these definitions on tape for future debugging purposes. The procedure is as follows. Only the ASR 33 console punch may be used.

1. Place the ASR 33 console OFF LINE; turn on the punch.

2. Punch a length of leader tape by the following method: strike and hold down in order the keys SHIFT, CNTRL, REPEAT, @ . When enough tape has been punched with leader-trailer code, release the keys in reverse order.

3. Type RUBOUT.

4. Type CR, LF.

5. Type the symbol, at least one space, and the definition (an octal number).

6. Repeat steps 4 and 5 for each definition required.

7. After the last definition, type CR, LF, EOT. (Note that steps 4-7 are identical with steps 3-6 described in the preceding section, Defining New Symbols.)

8. Punch a length of trailer by repeating step 2.

9. Turn the punch off; place the console ON LINE. Remove the new tape from the punch.

A tape punched in the above manner can be read into DDT's external table by the method described under Symbol Table Tapes (page 11).

Punching Binary Tapes

After making the desired corrections and changes, the user may punch out a new binary tape of his program. This allows the debugged program to be used immediately, without waiting for the programmer to incorporate the corrections in a new symbolic tape and reassemble the program. The punching procedure given below may be used for either the Teletype console punch or the optional high-speed punch. The device is indicated by the setting of bit 0 of Switch Register.

Bit 0 of Switch Register	Device to be Used
up	Console punch (low speed)
down	High-speed punch

In the following description, instructions in parentheses apply to the use of the console punch. If the high-speed punch is used, these instructions for turning the punch off and on may be ignored.

[T	This command is used to obtain a segment of leader-trailer.
a;b[P	This command causes DDT to punch a block of binary tape with the information contained in the section of core memory designated by the expressions a (lower limit) and b (upper limit), inclusive. a and b may be any kind of acceptable terms.
[E	This command is used at the end of punching operations and causes DDT to punch a checksum block, followed by a length of trailer tape.

The punching procedure, using these three commands, is as follows:

1. (Turn the punch off), type [T.

2. Turn punch on and press CONTINUE.

3. When punching is completed (turn punch off) type the lower limit, a semicolon, the upper limit, and [P.

4. (Turn on punch), press CONTINUE.

To punch more blocks, repeat steps 3 and 4 for each block.

5. After the last block has been punched, (turn off punch) type [E.

6. (Turn on punch), press CONTINUE. The computer will punch a checksum block and a length of trailer.

7. When the entire operation is finished, (turn off punch) remove the tape, depress CONTINUE. This binary tape may be loaded by the BIN Loader.

CAUTION

The user should not try to punch the section of memory between 5000 and 7600 which contains DDT.

If the user wishes to restart DDT before he has punched a complete tape (i.e., between data blocks) he must set the console switches to 5401 to preserve the checksum. Subsequent restarts must also be to 5401 until the checksum block has been punched.

APPENDIX 1

SUMMARY OF COMMANDS

Character	Action
space	Separation character.
+	Arithmetic plus.
-	Arithmetic minus.
/	Register examination character. When it follows the address register, it causes the register to be opened and its contents printed.
carriage return	Make modifications, if any, and close register.
line-feed	Make modifications, if any, close register, and open next sequential register.
↑	When it immediately follows a register printout, it causes the register addressed therein to be opened.
=	Type last quantity as an octal integer.
. (period)	Current location
←	Delete the line currently being typed.
[S	Sets DDT to type out in symbolic mode.
[O	Sets DDT to type out in octal mode.
N[W	Word search for all occurrences of the expression N masked with C([M).
k[B	Insert a breakpoint at the location specified by k. If no address is specified, remove any breakpoint.
n[C	Continue from a breakpoint n times automatically. If n is absent, it is assumed to be 1.
k[G	Go to the location specified by k.
[R	Read symbol table into external symbol table or define symbols on line.
[T	Punch leader-trailer code.

a;b[P	Punch binary tape from memory bounded by the addresses a and b.
[E	Punch end of tape: checksum and trailer.

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

[A	Accumulator storage (at breakpoints).
[Y	Link storage (at breakpoints).
[M	Mask used in search.
[L	Lower limit of search.
[U	Upper limit of search.

APPENDIX 2

INTERNAL SYMBOL TABLE

AND	=	0	CMA	=	7040
TAD	=	1000	CML	=	7020
ISZ	=	2000	RAR	=	7010
DCA	=	3000	RAL	=	7004
JMS	=	4000	RTR	=	7012
JMP	=	5000	RTL	=	7006
IOT	=	6000	IAC	=	7001
OPR	=	7000	SMA	=	7500
CLA	=	7200	SZA	=	7440
KCC	=	6032	SPA	=	7510
KRS	=	6034	SNA	=	7450
KRB	=	6036	SNL	=	7420
TSF	=	6041	SZL	=	7430
TCF	=	6042	SKP	=	7410
TPC	=	6044	OSR	=	7404
TLS	=	6046	HLT	=	7402
ION	=	6001	CIA	=	7041
IOF	=	6002	LAS	=	7604
KSF	=	6031	I	=	400
CLL	=	7100			

DIGITAL EQUIPMENT CORPORATION • MAYNARD, MASSACHUSETTS

Printed in U.S.A. 8/67