TOPS-10
DDT Manual

AA–BH82B–TB

April 1986

This manual describes the use of TOPS–10 DDT, the
Dynamic Debugging Tool for MACRO–10 programs.

This document supersedes the previous version of the same
name, order number AA–BH82A–TB.

OPERATING SYSTEM:	TOPS–10 V7.03
SOFTWARE:	DDT V44

The following are trademarks of Digital Equipment Corporation:

DEC	MASSBUS	RSX
DECmate	PDP	RT
DECsystem–10	P/OS	UNIBUS
DECSYSTEM–20	Professional	VAX
DECUS	Q–BUS	VMS
DECwriter	Rainbow	VT
DIBOL	RSTS	Work Processor

digital

The postpaid READER'S COMMENTS form on the last page of this document requests
the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

FIGURES

TABLES

PREFACE

MANUAL OBJECTIVES AND AUDIENCE

This manual explains and illustrates the features of TOPS-10 DDT, the
debugger for MACRO-10 programs. Although TOPS-10 DDT can be used to
debug the compiled code of programs written in higher-level languages,
this manual illustrates the use of TOPS-10 DDT to debug programs
written in MACRO-10 only.

This manual is both an introduction to the basic functions of TOPS-10
DDT and a reference guide to all TOPS-10 DDT commands and functions.

This manual assumes that the reader is familiar with using TOPS-10,
has done some programming in MACRO-10, and is familiar with the format
of MACRO-10 instructions.

STRUCTURE OF THIS DOCUMENT

| This manual has 12 chapters, 1 appendix, and 1 glossary.

    o   Chapter 1 introduces the concept of symbolic debugging and
        describes the variants of TOPS-10 DDT.

    o   Chapter 2 describes loading TOPS-10 DDTwith your program,
        discusses basic TOPS-10 DDT commands, and illustrates a
        sample debugging session.

    o   Chapter 3 explains the syntax of a DDT command. Chapter 3
        also describes expressions to enter data and explains how
        TOPS-10 DDT evaluates expressions.

    o   Chapter 4 discusses how to examine and modify a program using
        TOPS-10 DDT.

    o   Chapter 5 describes the use of TOPS-10 DDT to control program
        execution: how to start, stop, and monitor the running of a
        program.

    o   Chapter 6 explains how to perform searches of a program's
        address space using TOPS-10 DDT.

    o   Chapter 7 discusses the manipulation of program symbols using
        TOPS-10 DDT.

    o   Chapter 8 describes how to use the TOPS-10 DDT patching
        function to insert and test a new series of instructions in
        your program without reassembling the program.

o   Chapter 9 describes the use of FILDDT.

o   Chapter 1Ø describes the use of EDDT.

o   Chapter 11 describes special-use commands that control
     physical and virtual addressing. These commands are useful
     primarily when running EDDT and FILDDT.

o   Chapter 12 describes the use of DDT in non-zero sections
     (NZS).

o   Appendix A explains DDT and FILDDT error messages.

o   The glossary defines important TOPS-1Ø DDT terms.

OTHER DOCUMENTS

Other documents to which the reader should have access are:

o   MACRO Assembler Reference Manual

o   LINK Reference Manual

o   TOPS-1Ø Operating System Commands Manual

o   DECsystem-1Ø/DECSYSTEM-2Ø Processor Reference Manual

o   TOPS-1Ø/TOPS-2Ø RSX-2ØF System Reference Manual

CONVENTIONS

The following conventions are used in this manual in the description
of DDT commands and concepts.

{}	Curly brackets (braces) indicate that the enclosed item is optional.
. (period)	The address contained in DDT's location counter; also called the current location.
addr	A symbolic location within a program, a symbolic or absolute address in memory, an AC, or ".", the current location.
c	A single ASCII or SIXBIT character.
expr	Any expression that is legal in DDT.
filnam	One or more components of a file specification.
instr	Any instruction in the PDP-1Ø machine instruction set.
location sequence stack	A circular stack of memory locations that is used to store the addresses of certain previously referenced locations.
n	A numeric argument.
page	A page in memory. A page equals 512 words of memory.

symbol	A symbol name of up to 6 RADIX50 characters.
text	Any string of ASCII or SIXBIT characters.
word	Any 36-bit value occupying one word of memory.
\<ESC\>	Represents pressing the ESCAPE or ALTMODE key once.
\<ESC\>\<ESC\>	Represents pressing the ESCAPE or ALTMODE key twice.
\<CTRL/X\>	Represents pressing a key (represented by X) at the same time as you press the key labeled CTRL.
\<BKSP\>	represents pressing the BACKSPACE key or \<CTRL/H\>.
\<LF\>	Represents pressing the LINE FEED key.
\<RET\>	Represents pressing the RETURN key.
\<TAB\>	Represents pressing the TAB key or \<CTRL/I\>.

Numbers are in octal radix unless otherwise specified.

Examples of interaction between the user and DDT show user input in lowercase and DDT output in uppercase.

The symbols \<BKSP\>, \<CTRL/x\>, \<ESC\>, \<LF\>, \<RET\>, and \<TAB\> always represent user input.

NOTE

The descriptions of many DDT commands list the actions and effects of those commands. The actions and effects may not occur in precisely the order specified, but this has no effect on the user.

CHAPTER 1

INTRODUCTION TO DDT

DDT is a utility program you can use that will help you debug your
MACRO-10 programs. This manual describes how to use the DDT utility.

1.1 SYMBOLIC DEBUGGING

It is sometimes difficult to understand precisely the operation of a
program by reading the source code. DDT is a tool for interactively
examining the operation of a MACRO-10 program while it is running.
DDT is useful for finding programming errors (bugs) in programs that
do not run correctly. You can also use DDT to analyze the flow of
control in a program that is to be revised or rewritten.

With DDT, you can interrupt the execution of your program at locations
(breakpoints) you choose, and then examine and modify the program's
address space as required. You can execute instructions one-by-one to
check whether the effect of each instruction is what is intended. You
can then set other breakpoints in your program before continuing
execution.

When you refer to program locations and values, DDT allows you to use
the symbols that are defined in the program rather than absolute
values and addresses. This makes it much easier to refer to the
source listing and to find specific locations in memory.

After modifying the program's instructions or data, you can exit DDT
and save (with the monitor-level SAVE command) the changed version of
the program for further testing.

1.2 TOPS-10 VARIANTS OF DDT

There are several variants of DDT, each useful under specific
circumstances or for specific tasks.

The variants of TOPS-10 DDT are:

    o   DDT.REL

    o   DDT.EXE

    o   VMDDT.EXE

    o   FILDDT.EXE

    o   EDDT.EXE

DDT.REL is the relocatable variant of DDT, and can be loaded directly with your MACRO-10 program by using (at TOPS-10 command level) the TOPS-10 command DEBUG command. For example, enter the command string:

        DEBUG filnam

where filnam is the name of your MACRO-10 program (with default extension .MAC). The DEBUG command causes LINK to retain your program symbol table and to merge DDT.REL with your program module(s). See the TOPS-10 Operating System Commands Manual for a description of the DEBUG command.

If you use the DEBUG command with a program written in a higher-level language, such as COBOL or FORTRAN, you will invoke the language-associated debugging tool, such as COBDDT or FORDDT, rather than DDT. To use DDT.REL to debug a program written in a higher-level language, you can explicitly run LINK and load DDT.REL with your program. See the LINK Reference Manual for a description of the /TEST and /DEBUG switches. You can also use the DEBUG command with the /DDT switch, as:

        DEBUG/DDT filnam

where filnam is the name of your program. The /DDT switch tells LINK that you wish to use DDT, rather than the associated debugging tool.

DDT.EXE is a stand-alone variant of DDT that you can use to enter MACRO-10 instructions directly for testing. You run DDT.EXE as a user program by using the TOPS-10 RUN command:

        R DDT

DDT.EXE has system symbols defined as in UUOSYM and MACTEN, and recognizes hardware instructions (for example, MOVE and ADDI) and TOPS-10 UUOs (for example, TTCALL and CORE).

VMDDT.EXE is the variant that is merged with your program when, at TOPS-10 command level, you enter:

        DDT

Your program must be loaded in memory, such as by using the TOPS-10 GET command, or the TOPS-10 RUN command. You can use the DDT command to load VMDDT, whether your program has just been loaded, if it ran to completion, if it crashed, or if you used <CTRL/C> to exit from your program. If JOBDAT location .JBDDT is zero, indicating that there is no other debugger (for example, FORDDT) loaded with your program, the DDT command loads and starts VMDDT. Pages 700-777 are reserved for VMDDT.EXE.

If, during the debugging session, your program executes a CORE UUO to shrink core, the monitor reclaims the memory used by VMDDT, and your program will get an Illegal memory reference error if it reaches a breakpoint.

If your program was not saved with symbols, VMDDT can display only numeric values and may not be of much use. See the LINK Reference Manual for a description of how to use the /SYMSEG and /LOCALS switches to save your program symbols when loading your program.

FILDDT.EXE is used to examine and modify disk files and structures, the monitor, and other running jobs. FILDDT is discussed in Chapter 9.

EDDT.EXE is used to debug and patch the monitor. EDDT is discussed in Chapter 10.

CHAPTER 2

GETTING STARTED WITH DDT

2.1 INTRODUCTION

This chapter is an introduction to using DDT.   It describes how to
load DDT.REL with your program and shows how to perform basic DDT
functions.   It then illustrates a sample session debugging a simple
MACRO-10 program, using basic DDT functions. You can use DDT to debug
programs, using only the commands described in this chapter.   Once you
are familiar with using these commands, you may wish to learn how to
use the commands and functions that are described in the rest of the
manual, to perform more sophisticated debugging.

The commands used in this chapter are described only in sufficient
detail for the debugging task being performed; all commands are
thoroughly described in Chapters 3 through 11 of this document.

The best way to learn is by doing. You will learn the commands and
techniques discussed in this manual if you use them as you read about
them.   If you have a MACRO-10 program that you wish to debug, use it
to practice the commands discussed here.   If not, type in the program
X.MAC listed in Figure 2-1.

2.2 LOADING DDT

It is much easier to debug a program when you can use the symbols that
are defined in the program.   For you to be able to use program
symbols, DDT must have access to your program's symbol table.   One way
to provide this access is to use the TOPS-10 DEBUG command to load
DDT.REL with your program and retain your program symbols.   Load an
existing MACRO-10 program with the TOPS-10 DEBUG command as follows:

        DEBUG filnam

where filnam is the name of your MACRO-10 program.   The following
appears on your terminal (if your .REL file is older than your .MAC
file, MACRO-10 reassembles your program, otherwise the second line
does not appear):

        .DEBUG PROG
        MACRO: filnam
        LINK: Loading
        [LNKDEB DDT execution]
        DDT

where filnam is the name of your MACRO-10 program (with default
extension .MAC).   The last line (DDT) indicates that DDT is loaded,
and is ready to accept your commands.

2.3   BASIC FUNCTIONS

You must be able to perform certain basic functions to interactively
debug a program. Basic DDT functions are:

   o   starting the program

   o   stopping the program at specified locations

   o   examining and modifying memory

   o   executing program instructions one-at-a-time

   o   continuing execution of the program

   o   deleting input

   o   exiting DDT

You must give DDT commands to tell DDT what functions to perform. DDT
does not wait for a line terminator (such as a carriage return) to
indicate the end of your command. Instead, DDT reads your commands
character-by-character as you enter them.   When you enter a DDT
command, you almost never have to press the RETURN key.   This manual
explicitly indicates the occasions when a command requires you to
press the RETURN key.

<div align="center">NOTE</div>

   You must press the ESCAPE key as part of entering many
   DDT commands.   This manual uses the symbol <ESC> to
   indicate where you press the ESCAPE key.   When you
   press the ESCAPE key, DDT displays a dollar sign ($)
   on the screen. DDT never displays <ESC> when you
   press the ESCAPE key.

<div align="center">NOTE</div>

   This manual uses the symbols <BKSP>, <ESC>, <LF>,
   <RET>, and <TAB> to indicate where you press the
   BACKSPACE, ESCAPE, LINE FEED, RETURN, and TAB keys,
   respectively.   This manual also uses the symbol
   <CTRL/X> to indicate where you simultaneously press
   the CONTROL key and the key indicated by X. These
   symbols ALWAYS indicate where you press the specific
   keys noted here. You need NEVER enter the characters
   <BKSP>, <ESC>, <LF>, <RET>, <TAB>, or <CTRL/X>, to
   enter a DDT command.

Your commands appear on the screen as you type them. Use the DELETE
key to delete partially entered commands character-by-character. If
you try to delete more characters than you have entered, DDT displays:

   XXX

You can delete an entire command line with <CTRL/U>. When you do, DDT
displays:

   XXX

To exit DDT, enter:

   <CTRL/Z>

The other basic DDT functions are described in the rest of this
chapter.

2.3.1 Error Conditions

If DDT cannot execute a command, it displays a message to let you
know. The message may be only a single character (such as M or U, for
Multiply-defined symbol or Undefined symbol), a question mark (?), or
a complete message string. For most errors, DDT also sets a pointer
to the error string, so that if DDT did not display it, you can enter
a command to display the error string. The error string is available
for display until another error occurs, when DDT changes the pointer.
To display the error string that describes the last DDT error, enter:

        <ESC>?

(press the ESCAPE key, followed by a question mark).

2.3.2 Basic Concepts

A very useful DDT concept is that of the current location. The
current location is a memory location that you have referenced, either
implicitly or explicitly, with your last command, and that is the
default point of reference of your next command. The current location
can be thought of as the location "where you are". The symbol "."
(period) refers to the address of the current location, and can be
used as an argument in DDT commands.

The location counter is a DDT pointer that contains the address of the
current location. The location counter performs a function similar to
that of a bookmark. You can enter a command to display the contents
of a specific location but not change the address of the current
location, in order to maintain a specific point of reference for your
next command. Most DDT commands change the address of the current
location, and therefore also change the location counter. The
commands that do not change the current location are so indicated.

The open location is a memory word that can be modified by your next
command. DDT "opens" the location as a result of a command you give
to examine or modify memory. There is never more than one location
open at any given time. The open location is usually also the current
location.

To find the symbolic address of the current location, enter:

        ._   (a period followed by an underscore)

This causes DDT to display the following:

        ADDR1+n

where ADDR1 is a label defined in your program, and n is the offset of
the current location from that label (if the current location is
ADDR1, DDT does not display +n).

Another useful DDT concept is that of the current quantity. This is a
value that is the contents of the last word that DDT displayed, or the
value that you last deposited in memory. The current quantity is the
most recent of those values. Many DDT commands use arguments that
default to the current quantity.

The <u>location sequence stack</u> is a DDT storage area used to store the addresses of previous current locations. Certain DDT commands store the address of the current location on the location sequence stack. Other DDT commands change the address of the current location to an address that has already been stored on the location sequence stack. The location sequence stack functions in a fashion similar to inserting place-markers in a source code listing, to be able to "get back" to prior references.

2.3.3 Starting and Stopping the Program

When your program is loaded and DDT is ready to accept your commands (as indicated by <u>DDT</u> appearing on the terminal display), you can begin execution of your program at its start address by entering:

    <ESC>G

Unless you set one or more <u>breakpoints</u> before you start the program, your program runs either to completion or until it commits a fatal error. A breakpoint is a location in a program's executable code that has been modified so that <u>if</u> the program attempts to execute the instruction at that location, control passes to DDT <u>before</u> the instruction is executed.

The command to set a breakpoint is:

    addr<ESC>B

where <u>addr</u> is the address at which to stop execution. If the user-program PC reaches addr, DDT interrupts execution of the program before the program executes the instruction at the specified address. When DDT interrupts program execution at a breakpoint, it changes the current location to the breakpoint and opens the current location (the breakpoint).

While program execution is stopped at a breakpoint, you can display and change the contents of instruction and data words, remove breakpoints, set new breakpoints, and execute instructions one at a time (single-step). As you examine memory, you may find an instruction that is incorrect, and modify it. You can also examine and modify data words in memory. After modifying incorrect instructions and data in memory, you can immediately execute the instructions to check the effects of the modifications, without having to reassemble the source code.

Once you have made your changes, you can continue program execution at the place where execution was interrupted, restart the program at the beginning, or start execution at any other location you choose. The program will run to completion, until it reaches a breakpoint, or until it gets a fatal error.

2.3.4 Examining and Modifying Memory

One command to examine memory is:

     addr/

where addr is the address of the memory word you wish to examine
(display), and can be numeric or symbolic. DDT displays the contents
of the word located at addr. If the opcode field (bits 0-8) of the
memory word matches a recognized instruction or user-defined OPDEF,
DDT displays the contents of addr as an instruction (or OPDEF).   If
DDT finds (in the symbol table) any of the values to be displayed, DDT
displays those symbols rather than the numeric values.   For example,
either of the following display lines might appear on your terminal
(depending on the address and contents of the word):

     ADDR1/   MOVE 2,SYM1

     ADDR1+5/   SYM1,,SYM2

where ADDR1, SYM1, and SYM2 have been defined in the program.

If you enter a symbol that DDT does not find in the symbol table, DDT
sounds the terminal buzzer or bell, and displays U on the screen. If
you enter a symbol that is defined as a local symbol in more than one
module, DDT sounds the terminal buzzer or bell and displays M. You
can eliminate the multiply-defined symbol problem by "opening" the
symbol table of the module in which the correct symbol is defined.
See Chapter 7 (Manipulating Symbols in DDT) for more information.

When searching for a symbol to display, DDT uses global symbols in
preference to local symbols. However, DDT searches the "open" symbol
table first, and treats local symbols found in the open symbol table
as global symbols.   If DDT finds only a local symbol that is not in
the open symbol table, DDT displays the symbol with a pound-sign (#)
appended to the symbol. For example, DDT might display:

     ADDR#/   MOVE 2,SYM1

See Chapter 7 (Manipulating Symbols in DDT) for more information on
symbols and symbol tables.

The command addr/ changes the current location to addr and opens the
word at addr.

If you omit addr from an examine-memory command, such as addr/, DDT
uses the current quantity to determine the address of the location to
display. For example, after DDT displays the contents of ADDR1+5 as
above, if you enter "/", DDT displays the contents of the word located
at SYM2. The display line then appears:

     ADDR1+5/   SYM1,,SYM2   /   value

where value is the contents of the word located at SYM2. By default,
DDT displays value symbolically if it can.

The command / by itself (without addr) does not change the current
location.   Both forms of the / command open the location displayed,
enabling you to modify the location with your next command.

Another very useful command for examining memory is <TAB>. This command starts a new display line before displaying the contents of addr, making the display easier to read. For example, if you enter <TAB> after DDT displays the address and contents of ADDR1+5 (as above) on your terminal, the terminal display appears:

```
ADDR1+5/    SYM1,,SYM2    <TAB>
SYM2/    value
```

where value is the contents of the word located at SYM2. <TAB> does not appear on the screen, but is shown above to indicate where you press the <TAB> key. <TAB> changes the current location to SYM2 and opens the word at SYM2. In this example, the current quantity becomes value.

<TAB> also stores the address of the current location (ADDR1+5) on the location sequence stack before changing the current location to the location just displayed (SYM2). DDT uses the location sequence stack to "remember" previous values of the location counter. To "get back" to the previous current location, enter:

```
<ESC><RET>
```

In the above example, after you press <TAB> at ADDR1+5, DDT displays the contents of SYM2 and changes the current location to SYM2. When you enter <ESC><RET>, DDT changes the current location to ADDR1+5, opens the location at ADDR1+5, and again displays the contents of ADDR1+5.

If you use the command addr<TAB>, DDT deposits addr in the open location and closes the location before opening the location at addr and displaying its contents. <TAB> by itself does not deposit anything, but does save the current location on the location sequence stack, making <TAB> more useful than / (slash by itself).

You can display and open the word after the current location by entering:

```
<LF>
```

DDT changes the current location to the next word in memory, starts a new line, and displays the address of the (new) current location (as a symbol or a symbol plus an offset, if it can find a corresponding symbol in the symbol table), displays the contents of the current location, and opens the current location. For example, to display the next word in memory after ADDR1+5, enter:

```
<LF>
```

DDT changes the current location to ADDR1+6, starts a new line, and displays the address and contents of ADDR1+6. The screen display then appears as follows:

```
ADDR1+5/    SYM1,,SYM2    <LF>
ADDR1+6/    -1,,SYM3
```

Note that DDT does not display the characters <LF>. <LF> does not affect the location sequence stack.

Entering another <LF> causes DDT to display and open the next word.

To display and open the word <u>previous</u> to the current location, enter:

        <BKSP>

DDT changes the current location to the previous word, starts a new
line, displays the address and contents of the (new) current location,
and opens the current location. <BKSP> does not affect the location
sequence stack. For example, if you enter <BKSP> to open and display
the location before ADDR1+5, the screen appears as follows:

        ADDR1+5/   SYM1,,SYM2   <BKSP>
        ADDR1+4/   -3,,SYM2

Note that <u><BKSP></u> does not appear on the screen.

To change the contents of the open location, enter:

        value<RET>

where <u>value</u> can be an instruction, a symbol, or a numeric expression.

For example, if you enter the command <u>LABL2/</u>, DDT displays the
contents of the memory word at LABL2, and "opens" that word. If the
word at LABL2 contains:

        MOVE 1,SYM1

and you wish to change SYM1 to SYM2, enter:

        MOVE 1,SYM2<RET>

DDT stores the new instruction in the location at LABL2 and "closes"
the location. DDT does NOT display <RET>. The terminal display
appears as follows (your input is in lowercase):

        labl2/   MOVE 1,SYM1   move 1,sym2<RET>

The current location is still LABL2, but there is no open location.
To check whether the instruction is now correct, you can enter:

        ./

to display the contents of the current location. The screen display
now appears (your input is in lowercase):

        labl2/   MOVE 1,SYM1   move 1,sym2<RET>
        ./   MOVE 1,SYM2

After entering a command to display and open a location, if you enter:

        value<LF>

DDT stores the new value, changes the current location to the next
location in memory, starts a new display line and opens and displays
the new current location. The example above would then appear as
follows (your input is in lowercase):

        labl2/   MOVE 1,SYM1   move 1,sym2<LF>
        LABL2+1/   CONTENTS

where <u>CONTENTS</u> is the value stored at LABL2+1.

2.3.5 Executing Program Instructions

When you have interrupted program execution at a breakpoint, you can
execute the next instruction (the one at the breakpoint), by entering:

        <ESC>X

DDT executes the instruction, displays the results of executing the
instruction, and displays the address and contents of the next
instruction to be executed. This command changes the current location
to the next instruction to be executed. For example, assume that the
next instruction to be executed is located at LABEL1, which contains:

        MOVE 1,VARIBL

If the word at VARIBL contains SYM1, when you enter <ESC>X, DDT starts
a new line and displays:

        1/   SYM1   VARIBL/   SYM1
        LABEL1+1/   instr

where instr is the contents of LABEL1+1, and is the next instruction
to be executed. You can continue to execute instructions
one-at-a-time by entering successive <ESC>X commands. This is known
as single-stepping.

To execute a subroutine, enter:

        <ESC><ESC>X

DDT executes the subroutine and returns control to you if the
subroutine returns to a location +1, +2, or +3 from the instruction
that calls the subroutine. DDT changes the current location to the
address of the next instruction to be executed.

To continue execution of the program until the next breakpoint or
until program completion, enter:

        <ESC>P

DDT starts the program running again, beginning with the next
instruction to be executed. If you did not single-step any
instructions, the program begins by executing the instruction at the
breakpoint. If you have executed any instructions by single-stepping,
the program continues where you stopped. The effect is as if the
program were running without DDT in control.

2.4 A SAMPLE DEBUGGING SESSION USING DDT

This section describes a debugging session using DDT.   The program
being debugged is X.MAC, shown in Figure 2-1. The program and the
sample session are for illustration only. There are many styles of
programming and debugging, and these examples are descriptive rather
than prescriptive in intent.

You will understand this section and learn the commands described more
easily if you type in the program listed in Figure 2-1 and use the
commands as they are described.

Figure 2-1: Sample Program X.MAC


```
            SEARCH    UUOSYM
            TITLE     X
R0=0                                    ;AC0
IDX=6                                   ;INDEX REGISTER
P=17                                    ;STACK COUNTER

START:: MOVE      P,PWORD               ;Set up stack counter
        MOVEI     IDX,TABLE1            ;Address of table with X & Y
        PUSHJ     P,ADDEM              ;Do the addition
        MOVEI     IDX,TABLE1            ;Address of table
        MOVE      R0,ANSWER(IDX)        ;Answer to R0
        JFCL      0
        EXIT                            ;All done!
ADDEM:  MOVE      R0,X(IDX)             ;Load X
        ADD       R0,Y(IDX)             ;X + Y
        MOVE      R0,ANSWER(IDX)        ;Store answer
        POPJ      P,                    ;Return

TABLE1: BLOCK     3                     ;3 words
X==0                                    ;Offset for X
Y==1                                    ;Offset for Y
ANSWER==2                               ;Offset for answer
STKSIZ==10                              ;Stack size
PWORD:  IOWD STKSIZ,STACK               ;Stack pointer
STACK:  BLOCK     STKSIZ                ;Stack
        END       START
```


Figure 2-2 is an annotated session debugging X.MAC, the program in
Figure 2-1.   In the annotated session, the DDT terminal display is on
the left, user input is in the center in lowercase, and explanatory
comments about the session are on the right. This is not always the
way it appears on the terminal. Figure 2-3 shows the session as it
actually appears on the terminal.

The program is designed to pass the address of a table to a
subroutine.   The table contains three elements. The subroutine is to
add the first two elements of the table and store the result in the
third element before returning to the main program. There are no
input or output routines in the program.   The table is initialized
using DDT, and the result is checked while in DDT.

NOTE

DDT does not display <LF>, <RET>, or <TAB>. These are
shown in the sample session to indicate user input.

NOTE

DDT does not display the AC field of an instruction if
it is zero. This means that if your program contains
the instruction MOVE R0,LABL1, where   R0=0,   DDT
displays the instruction as MOVE LABL1.

Figure 2-2: Annotated Debugging Session

SCREEN DISPLAY	USER INPUT	EXPLANATION
.		TOPS-10 prompt.
	debug x<RET>	Begin the session by entering "debug x<RET>", where x is the name of your MACRO program.
MACRO: X LINK: Loading [LNKDEB DDT execution] DDT		MACRO reassembles your program (if needed), and LINK loads your program with DDT. DDT displays the "DDT" prompt.
	start/	Begin examining code at label "START".
MOVE P,PWORD#		DDT displays the instruction at START.
	<LF>	Press <LF> to display the next instruction.
DDTEND+1/ MOVEI IDX,TABLE1#		The first symbol in this program happens to coincide with DDTEND, a DDT symbol. When DDT scans the symbol table, it finds DDTEND before it finds START, and displays DDTEND instead. DDT still accepts START as an input symbol. Also note the pound-sign (#) appended to TABLE1 and PWORD. PWORD and TABLE1 are local symbols that are not in the open symbol table.
	ddtend<ESC>k	Enter ddtend<ESC>k to suppress DDT typeout of symbol DDTEND. DDT will display START rather than DDTEND from now on.
	x<ESC>:	Enter the module name (X) followed by <ESC> and a colon to open the symbol table associated with X. DDT will not append any more pound-signs.

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY	USER INPUT	EXPLANATION
	<TAB>	Press <TAB> to start a new display line, evaluate the current quantity as if it were an instruction, and display the contents of the location addressed by the Y field of the instruction. (Entering / (slash) displays the same word as <TAB>, but does not start a new line.) <TAB> also saves your place (like a bookmark) on the location sequence stack, so you can get back here easily.
TABLE1/ Ø		When you enter the <TAB> command, DDT displays the address and the contents of the location. The first element of the table contains zero. The <TAB> command also opens the location.
	2<LF>	Enter "2" followed by <LF> to deposit the value "2" in the first element, and to open and display the second element.
TABLE1+1/ Ø		The second element contains zero.
	3<LF>	Enter "3" followed by <LF> to deposit the value "3" in the second element and open and display the third element. The addition to be performed by the program is 2+3.
TABLE1+2/ Ø		The third element (the answer) contains zero.
	<ESC><RET>	Press <ESC>, then press <RET> to return to the address you saved on the location sequence stack.
START+1/ MOVEI IDX,TABLE1		DDT displays the address and contents of the last location you displayed before you entered <TAB>.
	<LF>	Press <LF> to look at the next location.

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY   USER INPUT   EXPLANATION

START+2/   PUSHJ P,ADDEM   This is the call to the
                                                      subroutine that does the
                                                      computation.

                              .<ESC>b   Enter ".", press <ESC>, and
                                                      enter "b" to set a
                                                      breakpoint at the current
                                                      location.

                              <ESC>g   Enter <ESC>g to start
                                                      program execution.

$1B>>START+2/   PUSHJ P,ADDEM   DDT displays the breakpoint
                                                      number, the address of the
                                                      breakpoint, and the
                                                      instruction at the breakpoint.
                                                      This instruction has not yet
                                                      been executed.

                              <ESC><ESC>x   Press <ESC> twice, then
                                                      enter "x" to let DDT
                                                      execute the subroutine.

START+3/   MOVEI IDX,TABLE1   DDT returns from the
                                                      subroutine at the next
                                                      instruction, and displays the
                                                      address and contents of the
                                                      instruction. If there is a
                                                      "skip return", DDT displays
                                                      "<SKIP>" if the program
                                                      skipped one instruction. If
                                                      the program skips 2 or 3
                                                      instructions, DDT displays
                                                      "<SKIP n>", where n is the
                                                      number of instructions
                                                      skipped.

                              <ESC>x   Press <ESC> and enter "x"
                                                      to execute the instruction.

  IDX/   TABLE1   TABLE1   DDT displays the address and
                                                      contents of IDX (the result of
                                                      executing the instruction),
                                                      and also displays "TABLE1"
                                                      (the result of evaluating the
                                                      Y field of the instruction).

START+4/   MOVE 2(IDX)   DDT then starts a new line and
                                                      displays the address and
                                                      contents of the next
                                                      instruction. Note that
                                                      DDT does not display the
                                                      zero in the AC field of
                                                      the instruction.

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY	USER INPUT	EXPLANATION
	\<ESC>\<TAB>	Press \<ESC>, then \<TAB> to display the contents of the location addressed by the instruction, using any indexing and indirection. (If you omit \<ESC>, DDT uses only the Y field, without indexing and indirection.)
TABLE1+2/ 0		The location addressed by the instruction is TABLE1+2, and its contents is zero. This is the table element that contains the answer, which should be 5.
	\<BKSP>	Press \<BKSP> to see the previous element in the table.
TABLE1+1/ 3		This element contains 3. That is correct.
	\<BKSP>	Press \<BKSP> again to check the previous element.
TABLE1/ 2		This element contains 2. That is also correct. One way to find the error is to single-step through the program.
	start\<ESC>b	Enter "start", press \<ESC>, and enter "b" to set a breakpoint at the beginning of the program.
	\<ESC>g	Press \<ESC> and enter "g" to start the program again.
$2B>>START/ MOVE P,PWORD		DDT displays the breakpoint number, and the address and contents of the instruction at the breakpoint.
	\<ESC>x	Press \<ESC>, then enter "x" to execute the instruction. This instruction moves a memory word to a register.
P/ -10,,PWORD PWORD/ -10,,PWORD		DDT displays the address and new contents of the register, and the address and contents of the memory word.

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY	USER INPUT	EXPLANATION
START+1/ MOVEI IDX,TABLE1		DDT then displays the address and contents of the next instruction.
	<ESC>x	Press <ESC>, then enter "x" to execute this instruction, which moves an immediate value to a register.
IDX/ TABLE1 TABLE1		DDT displays the address and new contents of the register, and the immediate value.
START+2/ PUSHJ P,ADDEM		DDT then displays the address and contents of the next instruction.
	<ESC>x	Press <ESC>, then enter "x" to execute the instruction.
P/ -7,,STACK		DDT displays the address and new contents of the stack pointer used by the PUSHJ.
<JUMP>		DDT displays "<JUMP>" if the change in PC is less than one or greater than 4.
ADDEM/ MOVE 0(IDX)		DDT displays the address and contents of the next instruction to be executed.
	<ESC>x	Press <ESC> and enter "x" to execute the instruction.
0/ 2 TABLE1/ 2		The instruction moved the contents of the word at TABLE1 (which is 2) to AC0. Looks OK so far.
ADDEM+1/ ADD 1(IDX)		DDT displays the next instruction.
	<ESC>x	Press <ESC> and enter "x" to execute the instruction.
0/ 5 TABLE1+1/ 3		The instruction added the contents of the word at TABLE1+1 (which is 3) to AC0, which now contains 5. OK.
ADDEM+2/ MOVE 2(IDX)		DDT displays the next instruction.
	<ESC>x	Press <ESC> and enter "x" to execute the instruction.

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY	USER INPUT	EXPLANATION
0/ 0 TABLE1+2/ 0		The instruction moved the contents of the word at TABLE1+2 to AC0. The MOVE instruction at ADDEM+2 should be MOVEM.
ADDEM+3/ POPJ P,0		DDT displays the next instruction (as a result of the <ESC>x).
	<BKSP>	Press <BKSP> to display and open the location with the incorrect instruction.
ADDEM+2/ MOVE 2(IDX)		DDT displays the previous instruction. This is the incorrect instruction.
	movem r0,answer(idx)<RET>	Enter the new instruction and press <RET>.
	./	Check the current location to see what you deposited.
MOVEM 2(IDX)		Looks OK.
	.<ESC>b	Set a breakpoint at ".", the current location.
	<ESC>g	Restart the program at the beginning.
$2B>>START/ MOVE P,PWORD		DDT displays the breakpoint information.
	<ESC>p	Press <ESC> and enter "p" to proceed from breakpoint 2 to the next breakpoint.
$1B>>START+2/ PUSHJ P,ADDEM		DDT displays the breakpoint information.
	<ESC>p	Proceed from breakpoint 1.
$3B>>ADDEM+2/ MOVEM 2(IDX)		DDT displays the breakpoint information. This is the instruction you changed.
	<ESC>x	Single-step the instruction to watch what it does.
0/ 5 TABLE1+2/ 5		The instruction moves the contents of AC0 to the word at TABLE1+2. OK!!

Figure 2-2: Annotated Debugging Session (Cont.)

SCREEN DISPLAY	USER INPUT	EXPLANATION
ADDEM+3/ POPJ P,Ø		DDT also displays the address and contents of the next instruction.
	start+4<ESC>b	Set a breakpoint at START+4 to check the results.
	<ESC>p	Proceed from breakpoint 3.
$4B>>START+4/ MOVE 2(IDX)		DDT displays the breakpoint information.
	<ESC>x	Single-step the instruction.
Ø/ 5 TABLE1+2/ 5		The instruction moves the contents of the word at TABLE1+2 to ACØ. The new value of ACØ is 5. OK!
START+5/ JFCL Ø		DDT displays the address and contents of the next instruction.
	<CTRL/Z>	Quit.
		Back at TOPS-1Ø command level.

Figure 2-3 shows the session as it actually appears on the terminal
screen. Again, user input is in lowercase. Comments on the right
indicate where you enter characters that do not echo.

Figure 2-3: Terminal Display of Debugging Session


```
.debug x
MACRO:  X
LINK:   Loading
[LNKDEB DDT execution]
DDT
start/    MOVE P,PWORD#                                        Enter <LF>.
DDTEND+1/   MOVEI IDX,TABLE1#    ddtend$k    x$:    Enter <TAB>.
TABLE1/   0    2                                              Enter <LF>.
TABLE1+1/   0    3                                            Enter <LF>.
TABLE1+2/   0    $                                            Enter <ESC><RET>.
START+1/    MOVEI IDX,TABLE1                                   Enter <LF>.
START+2/    PUSHJ P,ADDEM    .$b    $g
$1B>>START+2/    PUSHJ P,ADDEM    $$x
START+3/    MOVEI IDX,TABLE1    $x
    IDX/    TABLE1    TABLE1
START+4/    MOVE 2(IDX)    $                                  Enter <ESC><TAB>.
TABLE1+2/    0                                                Enter <BKSP>.
TABLE1+1/    3                                                Enter <BKSP>.
TABLE1/    2    start$b    $g
$2B>>START/    MOVE P,PWORD    $x
    P/    -10,,PWORD    PWORD/    -10,,PWORD
START+1/    MOVEI IDX,TABLE1    $x
    IDX/    TABLE1    TABLE1
START+2/    PUSHJ P,ADDEM    $x
    P/    -7,,STACK
<JUMP>
ADDEM/    MOVE 0(IDX)    $x
    0/    2    TABLE1/    2
ADDEM+1/    ADD 1(IDX)    $x
    0/    5    TABLE1+1/    3
ADDEM+2/    MOVE 2(IDX)    $x
    0/    0    TABLE1+2/    0
ADDEM+3/    POPJ P,0                                          Enter <BKSP>.
ADDEM+2/    MOVE 2(IDX)    movem r0,answer(idx)    Enter <RET>.
./    MOVEM 2(IDX)    .$b    $g
$2B>>START/    MOVE P,PWORD    $p
$1B>>START+2/    PUSHJ P,ADDEM    $p
$3B>>ADDEM+2/    MOVEM 2(IDX)    $x
    0/    5    TABLE1+2/    5
ADDEM+3/    POPJ P,0    start+4$b    $p
$4B>>START+4/    MOVE 2(IDX)    $x
    0/    5    TABLE1+2/    5
START+5/    JFCL 0    ^z
.
```

2.5 PROGRAMMING WITH DDT IN MIND

There are a few MACRO-10 programming techniques that make debugging
with DDT easier. These techniques primarily concern the use of labels
and symbols.

Labels that meaningfully describe (perhaps mnemonically) the function
of the code are more helpful when examining code and setting
breakpoints than labels that are alphanumerically coded (such as
A0001).

When using symbols as offsets into tables, you can prevent DDT from
displaying the offset symbol in place of the symbol's numeric value if
you define the symbol in this way:

        symbol==expression

Symbol is still entered in the symbol table, and you can use symbol as
input to DDT, but DDT does not display symbol on output.

For example, if you have defined:

        OFFSET==3

DDT displays the contents of a word that contains the value of 3 as:

        addr/   3

rather than:

        addr/   OFFSET

where addr is the address of the word. See the MACRO Assembler
Reference Manual for more information about defining symbols.

CHAPTER 3

DDT COMMAND FORMAT

3.1 COMMAND SYNTAX

The complete syntax of a DDT command is:

        {arg1<}{arg2>}{arg3}{<ESC>{<ESC>}{arg4}}c{arg5}

where arg1, arg2, arg3, arg4, and arg5 are arguments to the command c.
Arg1, arg2, and arg3 can be any legal DDT expression. Arg1 must be
followed by a left angle bracket (<), and arg2 must be followed by a
right angle bracket (>). Arg4 can only be a number. Arg5 is a text
argument of the form:

        /text/   or   c<ESC>

where text is a string of characters, the slashes (/) are delimiters
that can be any character not contained in text, and c is a single
character.

DDT commands never use all five arguments. Each argument is optional
or required according to the syntax of the specific command. Most DDT
commands are not more complicated than:

        arg3<ESC>c   or   arg3<ESC>arg4c

You can enter alphabetic commands and text arguments in uppercase or
lowercase.

An argument to a command can be the result of executing another
command.   For   example, you can enter a command to evaluate a text
string, and then enter another command to deposit in memory the result
of evaluating the text string. The entire command line would be:

        "/abcd/<RET>

where /abcd/ is the argument to the command " (quotation mark).   The
function of the quotation mark command is to evaluate the string
(abcd) within the delimiters (/) as a left-justified ASCII string.
The left-justified ASCII string abcd is then the argument to the
command <RET> (entered by pressing the RETURN key). The function of
the <RET> command is to deposit an argument (in this case, the string
abcd) into the open location. The " command is described in this
chapter, and the <RET> command is described in Chapter 4 (Displaying
and Modifying Memory).

Most commands produce results that are immediately visible, such as commands that display the contents of memory locations. However, commands such as those that invoke search functions or those that evaluate text expressions (as above) may not produce immediately visible results. If you enter a question mark (?) while DDT is performing a function invoked by one of these commands, DDT displays a message that tells you what DDT is currently doing. For example, such a message might be:

        Searching: addr/   value

where addr is the address that DDT is to next test as part of a search, and value is the contents of the memory location at addr. Still other commands return values that DDT does not display, but can use as arguments to other commands.

3.2   INPUT TO DDT

You enter arguments to DDT as expressions. An expression can be a single value, or a combination of two or more values with one or more operators.

3.2.1   Values in DDT Expressions

Values in DDT expressions can be:

        o   octal or decimal integers

        o   floating point numbers

        o   symbols

        o   values that are returned by commands

        o   text

To enter an octal integer value, simply enter the integer in octal digits. For example:

        70707065

To enter a decimal integer value, enter the integer in decimal digits and follow the value with a decimal point. For example:

        9876.

To enter a floating point number, use regular or scientific notation. For example, you can enter the value .034 as one of the following:

        .034
        3.4E-2

Note that 1. is a decimal integer, while 1.0 is a floating point number.

To enter a symbol as a value in an expression, type in the symbol name
as defined in your program. To enter an undefined symbol that you can
define later, enter:

        symbol#

where symbol is the symbol that you will later define. See Chapter 7
(Manipulating Symbols in DDT) for more information about using
undefined symbols.

You can enter a command that returns a value as a value in an
expression. DDT commands that return values and the values they
return are listed in Table 3-1.

Table 3-1: Commands That Return Values

COMMAND	VALUE RETURNED	VALUE ALSO KNOWN AS
.	The address of the current location.	.
<ESC>.	The address of the next user program instruction to be executed.	$.
<ESC><ESC>.	The previous value of "<ESC>.".	$$.
<ESC>nB	The address of the DDT location that contains the address of breakpoint n.	$nB
<ESC>nI	The address of the DDT location that contains the saved machine state flags (user-program context).	
<ESC>nM	The address of DDT "mask" n.	
<ESC>Q	The current quantity.	$Q
<ESC><ESC>Q	The current quantity, with halves swapped.	$$Q
<ESC>nU	The address of the DDT location that contains the argument (or default) that was given in the virtual addressing command: expr<ESC>nU.	

The commands <ESC>nB, <ESC>nI, <ESC>nM, and <ESC>nU, return values
that are the addresses of locations internal to DDT, which contain
information that you can use and modify. For brevity, these commands
are said to address those internal DDT locations.

For example, the command <ESC>nB returns (but does not display) the
address of the DDT location that contains the address of breakpoint n,
and the command addr/ (address followed by slash) displays the
contents of the location at addr.   To display the address of
breakpoint n, enter:

        <ESC>nB/

where you enter the command <ESC>nB as the expression for DDT to
evaluate as addr.

You can enter <u>text</u> to be interpreted in the following ways:

- o   left-justified ASCII strings

- o   left-justified SIXBIT strings

- o   single right-justified ASCII characters

- o   single right-justified SIXBIT characters

- o   RADIX50 words

You can enter text expressions in uppercase or lowercase. DDT translates strings to uppercase for SIXBIT or RADIX50 text as required.

The term <u>long text string</u> refers to an expression in a DDT command that is a string of text characters that requires more than one 36-bit expression for full evaluation. You can enter long text strings in SIXBIT and ASCII as DDT expressions. If you use a long text string as an expression, DDT assumes that you will enter a command that deposits the expression in memory.

DDT evaluates the string one 36-bit expression at a time. After evaluating the first 36-bit expression, DDT deposits the expression in the open location, closes the open location, and opens the next location.

DDT then evaluates the next 36-bit expression contained in the string, and deposits that expression in the (new) open location. This process continues until you enter <u>c</u>, the command. If you enter a command that does deposit to memory, DDT deposits the final 36-bit expression in the open location, and updates the location counter according to the rules of that particular command. The current quantity is the last 36-bit expression that DDT evaluated.

If you do not enter a command that deposits to memory, DDT uses, as the argument to the command, the 36-bit expression that was last evaluated. All other 36-bit expressions that were evaluated as part of the string have been deposited, and the current and open locations were updated accordingly. The current quantity is then the last 36-bit expression that DDT evaluated.

If there is no open location when you begin typing the long text string, DDT evaluates only the first 36-bit expression, ignores the rest of the string, and uses the first 36-bit expression as the argument to the command. The current quantity is then the first 36-bit expression that DDT evaluated in the string. If you enter a command that deposits to memory, it has no effect because there was no open location.

The syntax to enter an <u>ASCII string</u> is:

    "/text/

where <u>text</u> is the string, and the slashes (/) represent any printing character that is not contained within <u>text</u>. DDT evaluates the string as a series of 36-bit expressions, each in 7-bit ASCII format (left-justified), with all unused bits reset.

For example, if you enter:

    "+abc/def+

DDT evaluates one 36-bit expression as the 7-bit ASCII string abc/d in
bits 0-34, and bit 35 reset. If there is no open location, DDT uses
that expression as the argument to the command, and that expression
becomes the current quantity.

If there is an open location, DDT deposits abc/d in the open location,
closes it, and opens the next location in memory. DDT then evaluates
a second 36-bit expression as the 7-bit ASCII string ef in bits 0-13,
and bits 14-35 reset. The last 36-bit expression evaluated becomes
the current quantity.

                                NOTE

         You cannot use this format to enter an ASCII string
         that begins with the ESCAPE character, because <ESC>
         terminates the command that enters a single
         right-justified ASCII character (in this case, your
         intended delimiter).

The syntax to enter a SIXBIT string is:

    <ESC>"/text/

where text is the string,and the slashes (/) represent any printing
character that is not contained within text. DDT evaluates the string
as a series of 36-bit expressions, each in SIXBIT format
(left-justified), with any unused bits in the last 36-bit expression
reset. DDT translates lowercase characters to uppercase; all other
non-SIXBIT characters cause DDT to sound your terminal buzzer or bell
and display a question mark.

For example, if you enter:

    <ESC>">qwertyu>

DDT evaluates one 36-bit expression as the SIXBIT string QWERTY in
bits 0-35. If there is no open location, DDT uses that expression as
the argument to the command, and that expression becomes the current
quantity.

If there is an open location, DDT deposits QWERTY in the open
location, closes it, and opens the next location in memory. DDT then
evaluates a second 36-bit expression as the SIXBIT character U in bits
0-5, with bits 6-35 reset. The last 36-bit expression evaluated
becomes the current quantity.

                                NOTE

         You cannot use this format to enter a SIXBIT string
         that begins with the ESCAPE character, because <ESC>
         terminates the command that enters a single
         right-justified SIXBIT character (in this case, your
         intended delimiter).

The syntax to enter a right-justified ASCII character is:

    "c<ESC>

where c is the character. DDT evaluates this as one 36-bit expression
with the 7-bit ASCII character c in bits 29-35, and bits 0-28 reset.

The syntax to enter a right-justified <u>SIXBIT</u> <u>character</u> is:

    <ESC>"c<ESC>

where c is the character. DDT evaluates one 36-bit expression with the SIXBIT character <u>c</u> in bits 30-35, and bits 0-29 reset. DDT translates lowercase characters to uppercase; all other non-SIXBIT characters cause DDT to sound your terminal buzzer or bell and display a question mark.

The syntax to enter a <u>RADIX50</u> <u>word</u> is:

    text<ESC>5"

where <u>text</u> is any string of RADIX50 characters up to six characters long. DDT evaluates one 36-bit expression with bits 0-3 reset and the RADIX50 string <u>text</u> in bits 4-35. DDT ignores any characters in <u>text</u> after the sixth.

For example, if you enter:

    poiuytr<ESC>5"

DDT evaluates one 36-bit expression with bits 0-3 reset and the RADIX50 string <u>POIUYT</u> in bits 4-35. DDT ignores the character <u>r</u>. DDT translates lowercase characters to uppercase. Characters in <u>text</u> not in the RADIX50 character set that are DDT commands use, as an argument to the command, any characters already entered. Characters in <u>text</u> not in the RADIX50 character set that are not DDT commands cause DDT to sound your terminal buzzer or bell and display a question mark.

3.2.2 Operators in DDT Expressions

When you enter an expression, DDT evaluates the expression to create a 36-bit quantity but does not necessarily use all 36 bits when it executes the command. For example, you can enter a complete MACRO instruction when giving an argument to a command that requires an address, but DDT uses only the address specified by the instruction (and ignores the rest of the evaluated expression) when it executes the command.

Table 3-2 lists DDT's expression operators and the effects those operators produce on the evaluation. The term <u>value</u> <u>so</u> <u>far</u> represents the accumulated 36-bit value resulting from evaluation of the expression to that point.

Table 3-2:   Effects of Operators When Evaluating Expressions

OPERATOR	EFFECT ON EVALUATION
+	Add the 36-bit value on the left to the 36-bit value on the right, using two's complement addition.
-	Subtract the 36-bit value on the right from the 36-bit value on the left, using two's complement subtraction.
*	Multiply the 36-bit value on the left by the 36-bit value on the right, using PDP-10 full-word integer multiplication. DDT uses only the low-order 36 bits of the result.
' (apostrophe)	Divide the 36-bit value on the left by the 36-bit value on the right, using PDP-10 full-word integer division. DDT ignores any remainder. NOTE Apostrophe is DDT's division operator. / (slash) is a DDT command to examine memory, and is never used in DDT to indicate division.
space	Add the previous expression (normally an opcode) to the value so far, and add the low-order 18 bits of the value at the right of the space to the low-order 18 bits of the value so far. DDT ignores carries resulting from the addition, and does not change the left half of the value so far.
, (comma)	If you are entering an I/O instruction, shift the low-order 18 bits of the expression at the left of the comma 26 bits to the left (to the device field of the instruction), otherwise shift the low-order 18 bits of the expression at the left of the comma 23 bits to the left (to the A field of an instruction). Then logically OR the result into the value so far. NOTE DDT does not check whether the value at the left of the comma is a legitimate device or AC address, and may overwrite other parts of the instruction.

Table 3-2: Effects of Operators When Evaluating Expressions (Cont.)

OPERATOR	EFFECT ON EVALUATION
()	Swap the halves of the expression within the parentheses and add the resulting expression to the value so far. This makes it possible to enter an instruction that uses an index register. NOTE DDT does not check whether the value within the parentheses is a legitimate AC address, and may overwrite other parts of the instruction.
@	Assume the expression is an instruction and set the indirect bit (bit 13) of the value so far.
,, (two commas)	Move the low-order bits of the expression at the left of the commas to bits 0-17 and build a new 18-bit expression in the right half.

The nonarithmetic operators allow you to enter expressions in instruction format as well as in data format.

To enter an instruction, format the instruction as you would in a MACRO-10 program. For example:

MOVE R4,@VAR1+OFFSET(R5)

NOTE

Follow an opcode (such as MOVE) with a space, not a <TAB>.

To enter halfwords, enter the values (numbers or symbols) separated by two commas (,,). The halfwords can be symbolic or absolute values. For example:

-1,,SYM1

NOTE

DDT is not designed to evaluate complicated arithmetic expressions. The nonarithmetic operators are implemented to enable DDT to evaluate expressions you enter as MACRO-10 instructions and halfwords. Using values and operators for other purposes may not produce the results you intend.

3.3 COMMAND FILES

If you frequently use DDT or FILDDT to debug the same file (such as the monitor), and execute the same commands each time (such as loading symbols, suppressing symbols, applying patches, and setting virtual addressing conditions), you can save time by creating a file that contains these commands. At any time that you are in DDT you can give the command:

    <ESC>Y

to indicate that you wish to load a command file. DDT prompts:

    File:

Enter the file specification, followed by <RET>. You can enter a complete TOPS-10 file specification. DDT reads the file and executes the commands. If you append /A after the file specification, DDT stops executing the command file if it encounters any errors while executing the commands in the file. If you do not use the /A switch, DDT executes all legal commands in the control file regardless of any errors.

An alternate form of the command is:

    filnam<ESC>Y

where filnam is a SIXBIT file name up to six characters long, with default device DSK: and default extension .DDT. For example, to have DDT execute the DDT commands contained in file DSK:PROG.DDT, enter:

    <ESC>"/PROG/<ESC>Y

The delimiters (/) can be any character not in filnam.

If bit 15 of the TTY control mask is reset, DDT displays the commands and resulting output as it executes them. If bit 15 is set, DDT suppresses the display. The command <ESC>1M returns the address of the DDT location that contains the TTY control mask. See Chapter 4 (Displaying and Modifying Memory) for more information about the TTY control mask.

CHAPTER 4

DISPLAYING AND MODIFYING MEMORY

4.1 DISPLAY MODES

A major function of DDT is displaying the contents of memory words, both data and instructions. You can choose whether to display the contents of memory words as symbols or as numeric values. You can also select the radix in which DDT displays numeric values.

DDT displays symbols, labels, and most messages in uppercase.

4.1.1 Default Display Modes

There is no sure way for DDT to distinguish between instruction and data words, or between data words of different formats.

DDT displays memory words in symbolic mode by default. Symbolic mode is described in Table 4-1. DDT tests for the condition on the left, and if the condition is met, displays the word in the format described on the right. DDT performs the tests in descending order.

Table 4-1: Evaluation of Symbolic Display Mode

CONDITION	DDT DISPLAYS	EXAMPLE
Bits 0-18 are all set.	A negative number in the current radix.	-45
The 36-bit value is defined in the user program symbol table.	The symbol.	SYMBL1 HALT
The opcode field is zero.	Halfwords.	345,,-27
The opcode and I, X, and Y fields, or the opcode and A fields match an OPDEF in the user program symbol table.	The OPDEF.	CORE 6,
The opcode matches a definition in DDT's internal hardware instruction table.	The instruction.	MOVE 3,SYMBL
No match.	Halfwords.	3445,,-23

By default, DDT displays numeric values in radix 8. Leading zeros are always suppressed.

4.1.2 Selecting Display Modes

You can select display modes to control:

- o the format in which DDT tries to interpret the contents of memory locations; for example, as instructions, or as floating-point numbers.

- o whether addresses are displayed as symbolic or numeric values.

- o the radix in which numeric values are displayed.

In addition, you can specify these modes on a short-term (temporary mode) or long-term (prevailing mode) basis.

A prevailing display mode remains in effect until you select another prevailing mode, but may be overridden by a temporary mode until you enter a command that restores the prevailing display mode. DDT commands that restore the prevailing display mode are:

- o {expr}<RET>   (deposit expr and close location)

- o <ESC>G   (start program execution)

- o <ESC>P   (proceed from a breakpoint)

- o <ESC>W, <ESC>E, <ESC>N   (perform a search)

- o <ESC>Z   (zero memory)

- o instr<ESC>X   (execute instr)

- o <ESC>Y   (execute DDT command file)

- o <ESC>V   (watch a location)

The syntax of commands that set the prevailing mode is:

      <ESC><ESC>mode

where mode is one of the display modes shown in Table 4-2.

The syntax of commands that set a temporary mode is:

      <ESC>mode

where mode is one of the display modes shown in Table 4-2.

The current display mode is the mode (prevailing or temporary) in which DDT will display the next word (unless you enter a command to change the display mode).

DDT has two "masks" that control the action of two of the display modes.

<ESC>3M is a command that addresses a DDT location that contains the output byte size mask. When the current display mode is O, each bit that is set in the mask indicates the position of a low order bit of a byte in the word being displayed. In this mode, bit 35 is always assumed to be set. For example, if the output byte size mask contains:

        510410100400   (octal)

the byte sizes specified are, from left to right, 1, 2, 3, 4, 5, 6, 7, and 8. When displaying a word in O mode that contains 777777,,777777, and the current radix is 8, DDT displays:

        1,3,7,17,37,77,177,377

The default value of the output byte size mask is zero, specifying one 36-bit byte.

You can set the output byte size mask with the command:

        expr<ESC>3M

where expr evaluates to the bit pattern required.

You can also examine and change the output byte size mask with the examine and deposit commands described later in this chapter.

<ESC>2M is a command that addresses a DDT location that contains the maximum symbolic offset. When DDT displays an address in R(elative) mode, it displays the address symbolically, that is, as a symbol, or as a symbol + the numeric offset of the address from that symbol. The maximum symbolic offset (minus 1) determines the maximum offset address that DDT displays symbolically, and defaults to 1000 (octal). DDT displays addresses beyond that offset in A(bsolute) mode. For example, assume that the maximum symbolic offset is 2, and that you are examining subroutine ADDEM in program X.MAC (Fig 2-1), using <LF> to display instructions in sequence. DDT displays:

        ADDEM/   MOVE 0(6)
        ADDEM+1/   ADD 1(6)
        addr/   MOVE 2(6)

where addr is the absolute address (for example, 14414) of the location.

You can set the maximum symbolic offset with the command:

        expr<ESC>2M

where expr evaluates to the offset required.

You can also examine and change the maximum symbolic offset with the examine and deposit commands described later in this chapter.

DDT display modes and the commands that select them are described in Table 4-2.

Table 4-2: DDT Display Modes

FORMAT MODES	
MODE	EFFECT
C	Display memory word as numbers in the current radix (see Radix Modes).
F	Display memory word as a floating point decimal number.
H	Display memory word as two halfword addresses (see Address Modes) separated by two commas (,,).
O	Display memory word as numeric bytes of sizes that are specified by the <ESC>3M mask.
n0	Display memory word as n-bit numeric bytes, (with trailing remainder byte, as required).
S	Display memory word in symbolic mode (default).
1S	Search DDT's internal hardware opcode table before searching the user's symbol table, otherwise follow rules for symbolic mode.

nT	Display memory word as ASCII text, using n-bit bytes.
	
	    n=5: RADIX50
	
	    n=6: SIXBIT
	
	    n=7 through 36:
	        Specifies the number of bits per byte. The default is 7-bit ASCII.
	
	    n=0: ASCIZ
	
	    (Stop ASCIZ typeout by typing any character.)

A	Display addresses as absolute values in the current radix.
R	Display addresses as values relative to symbols (default). DDT displays the offsets in the current radix. The maximum offset is controlled by the value stored in the <ESC>2M mask, and defaults to 1000 (octal).
RADIX MODES	
MODE	EFFECT
nR	Display numeric values in radix n (default=8), where n is a decimal number greater than 1. If n=8, DDT displays the word as octal halfwords, otherwise DDT displays the word as one number.

4.2 DISPLAYING EXPRESSIONS

DDT has three commands you can use to display expressions in different
modes. They are:

- ; (semicolon)

- = (equal sign)

- _ (underscore)

The syntax of these commands is:

    {expr}c

where <u>expr</u> is the expression to display (<u>expr</u> defaults to the current
quantity), and <u>c</u> is one of the above commands. These commands are
useful for redisplaying the current quantity without affecting the
current display mode.   Table 4-3 lists the commands to display
expressions and their effects.

Table 4-3:   Commands to Display Expressions

COMMAND	EFFECT
;	Display the current quantity in the current display mode.
expr;	Display expr in the current display mode.
=	Display the current quantity as a number in the current radix.
expr=	Display expr as a number in the current radix.
-	Display the current quantity in 1$ mode.
expr_	Display expr in 1$ mode.

4.3 DISPLAYING BYTE POINTERS

If you set the display mode to 1T, DDT displays the contents of the
memory location as a byte pointer. DDT can display one-word local,
one-word global, and two-word byte pointers. DDT displays the P and S
fields, and the address as determined by the I, X, and Y fields of the
byte pointer.

In section zero, DDT displays only one-word byte pointers (local and
global).

For example, if the contents of the location at ADDR2 is 100702,,addr,
where addr is the value of symbol LABL2, the following illustrates
one-word local byte pointer display:

    addr2/   100702,,addr   <ESC>1T;   10 7 LABL2(2)

The following illustrates one-word global byte pointer display, where
addr is the value of symbol LABL2:

    1,,addr2/   610002,,LABL2   <ESC>1T;   44&7 2,,LABL2

The following illustrates two-word global byte pointer display, where
addr is the value of symbol LABL2 (DDT echoes <BKSP> as ^H):

    1,,addr2/   440740,,0   <LF>
    1,,addr2+1/   3,,addr   <ESC>1T^H
    1,,addr2/   44 7 3,,MAIN. <2>

4.4 DISPLAYING AND DEPOSITING IN MEMORY

DDT allows you to display the contents of memory locations and deposit
a new value in the open location. In performing these functions, you
must understand the concept of the open location, the current
location, the location sequence stack, and the current quantity.

The open location is a memory location (or AC) that is "open" for
modification by the next command.   There is never more than one
location open at a time.   DDT always closes the open location before
opening another.

The location counter contains the address of a word in memory that has
been referenced (implicitly or explicitly) by the previous command,
and that is the default point of reference for the next command. That
word is known as the current location. DDT uses the address of the
current location as the default address in most commands. The current
location is often, but not always, the open location.

Most DDT commands change the current location to a word specified by
an address given (explicitly or by default) in the command. Commands
that do not are so indicated.

"." (period) is a command that returns (but does not display) the
address of the current location.

When you first enter DDT, the current location is zero.

The location sequence stack is a "ring" of seventeen words, each
containing the address of a prior current location, or of a match
found during a search. The present value of the current location is
not placed in the ring.

Entries are made to and retrieved from the location sequence stack in
a last-in, first-out manner. Most commands that change the location
counter by values other than +1 and -1 cause DDT to place the address
of the current location (before the change) on the location sequence
stack. Addresses of matching locations found during searches are also
placed on the location sequence stack. When DDT enters a new value in
the next word on the stack, the new value becomes the current location
stack entry. This is similar to PUSHing entries on a stack. When the
current location stack entry is the last location on the location
sequence stack, DDT enters a new value on the stack by "wrapping
around" to the beginning of the stack and overwriting the value in the
first location on the stack. The first location on the stack then
contains the current location stack entry.

Certain DDT commands change the address of the current location to the current location stack entry, and then change the current location stack entry to the previous entry. This is similar to POPping entries off a stack, and allows you to "return" to locations that have previously been the current location. When the first location on the location sequence stack contains the current location stack entry and DDT changes the address of the current location to the current location stack entry, DDT "wraps around" to the end of the stack, and the value contained in the last word of the stack becomes the current location stack entry (whether or not the stack was previously "full").

The current quantity is a value that is the most recent of:

- o the last 36-bit quantity that DDT displayed (an expression or the contents of a memory location)

- o the last expression that you entered as an argument to a command that deposits to memory

This value is also known as the last value typed. <ESC>Q is a command that returns (but does not display) the current quantity. DDT issues an implicit <ESC>Q to return this value for use as the default argument for some commands.

You can give the current quantity as an argument to a command by entering the command <ESC>Q as the argument.

The command <ESC><ESC>Q returns the current quantity with the right and left halves swapped.

This manual uses the term $Q to refer to the value that is returned by the command <ESC>Q, and the term $$Q to refer to the value that is returned by the command <ESC><ESC>Q.

Some commands calculate the address of the location to be opened from an expression given or defaulted in the command. Other commands use the address of the current location or entries on the location sequence stack.

The general syntax of these commands is:

    {expr}{<ESC>}c

where expr is any legal DDT expression, and c is the command.

NOTE

    See Values in DDT Expressions in Chapter 3 for a
    discussion of long text strings as values in DDT
    expressions.

Table 4-4 summarizes the commands and their effects. Complete descriptions of the commands follow the table.

Table 4-4: DDT Commands to Display Memory

COMMAND	DISPLAY CONTENTS	MODE OF DISPLAY	OPEN THE LOCATION	CHANGE CURRENT LOCATION	DEPOSIT EXPR
/	Yes	Current	Yes	Yes(1)	No
[	Yes	Numeric	Yes	Yes(1)	No
]	Yes	Symbolic	Yes	Yes(1)	No
!	No	Suppress	Yes	Yes(1)	No
\	Yes(2)	Current	Yes	No	Yes(1)
<TAB>	Yes(2)	Current	Yes	Yes	Yes(1)
<RET>	No	Restore	No	No	Yes(1)
<LF>	Yes(2)	Current	Yes	Yes(.+1)	Yes(1)
<BKSP> or ^	Yes(2)	Current	Yes	Yes(.-1)	Yes(1)

(1) If you enter expr.

(2) If not suppressed by !.

4.4.1 Commands That Use the Current Location

The commands <RET>, <LF>, and <BKSP> use the address of the current location to determine the next address of the current location.

These commands do not make entries to the location sequence stack.

{expr}<RET> does the following:

   o deposits expr (if given) in the open location

   o closes the open location

   o resets the current typeout mode to the prevailing typeout mode

   o does not change the address of the current location

April 1986

{expr}<LF> does the following:

- o deposits expr (if given) in the open location

- o closes the open location

- o increments the location counter

- o opens the current location

- o displays the open location (unless display has been suppressed by !)

{expr}<BKSP> and {expr}^ do the following:

- o deposit expr (if given) in the open location

- o close the open location

- o decrement the location counter

- o open the current location

- o display the open location (unless display has been suppressed by !)

4.4.2 Commands That Use the Location Sequence Stack

The commands <ESC><RET>, <ESC><LF>, and <ESC><BKSP> use the current location stack entry to determine the next address of the current location.

Repetitions of these commands refer to successively earlier entries on the stack, until you again address the most recent entry.

These commands do not make entries to the location sequence stack.

{expr}<ESC><RET> does the following:

- o deposits expr (if given) in the open location

- o closes the open location

- o changes the value contained in the location counter to the current location stack entry

- o opens the current location

- o starts a new line and displays the address and contents of the open location in the current display mode

- o causes the previous entry on the location sequence stack to become the current location stack entry

NOTE

If display is suppressed as a result of using the ! command, the command {expr}<ESC><RET> restores the current display mode, which can be either a temporary or prevailing display mode.

{expr}<ESC><LF> does the following:

- o deposits expr (if given) in the open location

- o closes the open location

- o changes the value contained in the location counter to the current location stack entry

- o increments the location counter

- o opens the current location

- o starts a new line and displays the address of the open location

- o displays the contents of the open location (unless display has been suppressed by !)

- o causes the previous entry on the location sequence stack to become the current location stack entry

{expr}<ESC><BKSP> and {expr}<ESC>^ do the following:

- o deposit expr (if given) in the open location

- o close the open location

- o change the value contained in the location counter to the current location stack entry

- o decrement the location counter

- o open the current location

- o display the address of the open location

- o display the contents of the open location (unless display has been suppressed by !)

- o cause the previous entry on the location sequence stack to become the current location stack entry

4.4.3 Commands That Use an Address Within the Command

The commands:

```
/        (slash)
[        (left square bracket)
]        (right square bracket)
!        (exclamation point)
\        (backslash)
<TAB>
```

use an expression given in the command (either explicitly or by default) to determine the addresses of the current location and the open location.

DISPLAYING AND MODIFYING MEMORY

The complete syntax of these commands is:

    {expr}{<ESC>{<ESC>}}c

where expr may be an address, ".", a symbol, or any expression that is legal in DDT, and c is the command.

When you use the commands /, [,], !, \, and <TAB>:

- o If you omit expr

  - > DDT uses the current quantity as a default.

  - > <TAB> enters the address of the current location on the location sequence stack and changes the current location to the address determined from the current quantity.

- o If you enter expr, DDT enters the address of the current location on the location sequence stack (except \).

- o DDT treats expr (whether given or defaulted) as if it were in instruction format and performs the effective address calculation as follows:

  - > If you omit <ESC>, DDT does not perform indexing or indirection.

  - > If you include one <ESC>, DDT treats expr as an IFIW (instruction format indirect word), and uses the I and Y fields of expr to perform indexing and indirection when appropriate.

  - > If you use <ESC><ESC>, DDT utilizes EFIWs (extended format indirect words), as appropriate, when performing effective address calculations, and can thereby calculate 30-bit addresses.

  - > In section zero, when you include <ESC><ESC>, it is treated as one <ESC>.

These commands always do the following:

- o close the open location

- o open the location at the address indicated by expr

- o change the current quantity to the value displayed (all commands except !)

The following is a list that gives a complete description of the effects of each command.

COMMAND   EFFECTS

/

- o closes the open location

- o opens the location at the address calculated from the current quantity

- o displays the contents of the open location in the current display mode

- o sets the current quantity to the value displayed

expr/

- o   closes the open location

- o   opens the location at the address calculated from expr

- o   enters the address of the current location on the location sequence stack

- o   changes the current location to the location at the address calculated from expr

- o   displays the contents of the open location in the current display mode

- o   sets the current quantity to the value displayed

[

- o   closes the open location

- o   opens the location at the address calculated from the current quantity

- o   displays the contents of the open location in numeric mode in the current radix

- o   sets the current quantity to the value displayed

expr[

- o   closes the open location

- o   opens the location at the address calculated from expr

- o   enters the address of the current location on the location sequence stack

- o   changes the current location to the location at the address calculated from expr

- o   displays the contents of the open location in numeric mode in the current radix

- o   sets the current display mode to numeric mode in the current radix

- o   sets the current quantity to the value displayed

]

- o   closes the open location

- o   opens the location at the address calculated from the current quantity

- o   displays the contents of the open location in symbolic mode

- o   sets the current display mode to symbolic mode

- o   sets the current quantity to the value displayed

expr]

      o   closes the open location

      o   opens the location at the address calculated from expr

      o   enters the address of the current location on the location
         sequence stack

      o   changes the current location to the location at the address
         calculated from expr

      o   displays the contents of the open location in symbolic mode

      o   sets the current display mode to symbolic mode

      o   sets the current quantity to the value displayed

!

      o   closes the open location

      o   opens the location at the address calculated from the current
         quantity

      o   does not display the contents of the open location

      o   suppresses display of the open location by the \, <TAB>,
         <LF>, and <BKSP> commands (any other display command restores
         the current display mode)

      o   does not change the current quantity

expr!

      o   closes the open location

      o   opens the location at the address calculated from expr

      o   enters the address of the current location on the location
         sequence stack

      o   changes the current location to the location at the address
         calculated from expr

      o   does not display the contents of the open location

      o   suppresses display of the open location by the \, <TAB>,
         <LF>, and <BKSP> commands (any other display command restores
         the current display mode)

      o   does not change the current quantity

\

      o   closes the open location

      o   opens the location at the address calculated from the current
         quantity

      o   displays the contents of the open location in the current
         display mode (unless display has been suppressed by !)

      o   sets the current quantity to the value displayed

expr\

    o   deposits expr in the open location

    o   closes the open location

    o   opens the location at the address calculated from expr

    o   does not change the address of the current location (and does not enter the address of the current location on the location sequence stack)

    o   displays the contents of the open location in the current display mode (unless display has been suppressed by !)

    o   sets the current quantity to the value displayed

<TAB>

    o   closes the open location

    o   opens the location at the address calculated from the current quantity

    o   enters the address of the current location on the location sequence stack

    o   changes the current location to the location at the address calculated from the current quantity

    o   starts a new line and displays the address of the open location (which is also the current location)

    o   displays the contents of the open location in the current display mode (unless display has been suppressed by !)

    o   sets the current quantity to the value displayed

expr<TAB>

    o   deposits expr in the open location

    o   closes the open location

    o   opens the location at the address calculated from expr

    o   enters the address of the current location on the location sequence stack

    o   changes the current location to the location at the address calculated from expr

    o   starts a new line and displays the address of the open location (which is also the current location)

    o   displays the contents of the open location in the current display mode (unless display has been suppressed by !)

    o   sets the current quantity to the value displayed

You can treat expr as an IFIW (instruction format indirect word), and use any indexing and indirection specified by expr to compute the effective address of the location to be opened. Use the command form:

        {expr}<ESC>c

where c is /, [,], !, \, or <TAB>.

For example, assume the following conditions as indicated by the display commands:

COMMAND   DISPLAY   EXPLANATION

LABL1/   SYM1   Display contents of LABL1.
LABL1+1/   SYM2   Display contents of LABL1+1.
SYM2/   SYM3   Display contents of SYM2.
2/   1   Display contents of AC 2.
@LABL1(2)/   SYM1   DDT uses Y field only.
@LABL1(2)<ESC>/   SYM3   <ESC> causes indexing and indirection.

Note that DDT does not start a new line unless you enter <TAB>, <RET>, <LF> or <BKSP>, or until the display wraps around the end of the line. DDT also displays three spaces (or a tab, depending on the TTY control mask) before and after its output. Thus, an actual DDT terminal display might be the following (user input is lowercase; <LF> and <TAB> do not appear on the screen, but are shown to indicate where you pressed the corresponding keys):

2/   1   labl1/   SYM1   <LF>
LABL1+1/   SYM2   <TAB>
SYM2/   SYM3   sym4/   MOVE 1,@LABL1(2)   <ESC><TAB>
SYM2/   SYM3

You can treat expr as an EFIW (extended format indirect word) and use any indexing and indirection specified by expr to compute the (global) effective address of the location to be opened. Use the command form:

        {expr}<ESC><ESC>c

where c is /, [,], !, \, or <TAB>.

4.5 DISPLAYING ASCIZ STRINGS

You can display memory as an ASCIZ string. The command

        addr<ESC>0T

where addr defaults to the open location (if there is one, otherwise addr defaults to the current location), displays memory, beginning with addr, as an ASCIZ string. The display stops when DDT finds a zero byte, or when you type in any character, which DDT displays, but otherwise ignores. The current location remains unchanged.

4.6 ZEROING MEMORY

To deposit the same value in each of a string of memory words (useful for initializing memory to zero), enter:

        addr1<addr2>{expr}<ESC>Z

where expr is any legal DDT expression, addr1 is the first word to receive expr, and addr2 is the last. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>). Both addr1 and addr2 are required. If you omit expr, it defaults to zero. Prior to execution, DDT enters the address of the current location on the location sequence stack and closes the open location. When DDT has completed execution of the command, the current location is the word at addr2 + 1. There is no open location. This command restores the prevailing display mode.

If you enter:

        ?

while DDT is executing the <ESC>Z command, DDT displays:

        Depositing: addr/   value

where addr is the location where DDT will make the next deposit, and value is the contents of addr before the deposit.

If you enter any other character, DDT stops executing the <ESC>Z command, and waits for your next command. The character that you enter to terminate the <ESC>Z command is otherwise ignored.

4.7 AUTOMATIC WRITE-ENABLE

If you attempt to change a word that is write-protected, DDT removes the protection, makes the change, and then reinvokes the protection.

To prevent DDT from changing write-protected memory, enter:

        <ESC><ESC>{Ø}W

If you attempt to modify write-protected memory while automatic write-enable is turned off, DDT returns the message:

        ?NOT WRITABLE

To allow DDT to change write-protected memory, enter:

        <ESC>{Ø}W

This is the default condition.

The zero in the above commands is optional and has no effect on the operation of the commands.

| 4.8 AUTOMATIC PAGE CREATION

If you attempt to deposit a value in a word within a nonexistent page,
DDT sounds your terminal buzzer or bell and displays a question mark
(?).

To allow DDT to create a page when you attempt to deposit a word
within a nonexistent page, enter:

>   <ESC>1W

If DDT tries to create the page and fails, DDT displays:

>   ?CAN'T CREATE PAGE

To prevent DDT from creating a page when you attempt to deposit a
value within a nonexistent page, enter:

>   <ESC><ESC>1W

This is the default condition.

4.9 DISPLAYING PAGE ACCESSIBILITY INFORMATION

You can get information about the access requirements of the pages and
sections in the program you are debugging, using the $L and $$L
commands. The complete format for this command is:

>   {{arg1<}arg2}{<ESC>}<ESC>L

where arg1 and arg2 are sections numbers. Using one <ESC> causes DDT
to display access information about the section and about individual
pages. Using <ESC> twice causes DDT to display access information
only about the section(s).   If you include both arg1 and arg2, DDT
displays the information for all sections that your program and DDT
are using, in the range arg1 to arg2, inclusive. If you include only
arg2, DDT displays access information for that section only.   If you
omit both arguments, DDT displays access information for all sections
that your program and DDT are using.

The page and section accessibility bits and their meanings are:

Read	Page can be read.
Write	Page can be written.
AA	Page access is allowed.
Sharable	Page can be shared.
Hiseg	Page is part of high segment.
Zero	Page is allocated but zero.
Spying	Page is spying on someone.
Cannot page	Page cannot be paged out.
Paged out	Page has been paged out.
Locked	Page is locked in core.
Not cached	Page is not cached.

For example, the command <ESC>L might produce the following display:

```
Section 0
000              Read, Write, AA, Cannot page
001-030          Read, Write, AA
577              Read, Write, AA
600-656          Read, AA, Sharable, Hiseg

Section 1 indirect to section 0
000              Read, Write, AA, Cannot page
001-030          Read, Write, AA
577              Read, Write, AA
600-656          Read, AA, Sharable, Hiseg

Section 3
000-777          Read, Write, AA

Section 4
000-354          Read, Write, AA
```

And the command <ESC><ESC>L might product a display like the following:

```
Section 0
Section 1 indirect to section 0
Section 3
Section 4
```

4.10   WATCHING A MEMORY LOCATION

If you wish to have DDT monitor or "watch" a memory location while your program is running, and display the location whenever its contents change, enter:

       addr<ESC>V

where addr is the address of the location to be watched, and defaults to the current location. When you enter the command, DDT starts a new line and displays:

       addr/   value

where addr is the address of the location being watched, and value is the contents of the location. This command also restores the prevailing display mode.

DDT checks addr every "jiffy" (about 20 milliseconds), and displays the address and contents of addr whenever those contents change. (Executive mode EDDT watches addr continuously.)

If you enter a question mark (?) while DDT is watching, DDT displays:

       Watching: addr/   value

where addr is the address of the location being watched, and value is the contents of addr.

To terminate the watch, enter any other character. DDT stops monitoring the word, starts a new display line, echoes the character you enter, starts another line, and waits for more input. The character that you enter to terminate the watch is otherwise ignored.

Because any input character terminates the watch, you cannot continue
execution and watch your own user program. The <ESC>V command is
useful to watch activity in a separate process (such as the running
monitor or other job, for which you must be using EDDT or FILDDT).
The page that contains the word you wish to watch must be mapped into
your own process (the one that contains DDT and your program).

4.11 TTY CONTROL MASK

You can control certain aspects of DDT's display by setting DDT's TTY
control mask. The command <ESC>1M returns a value that is the address
of the DDT location that contains this mask. Table 4-5 summarizes the
features controlled by the bits in the TTY control mask.

Table 4-5: TTY Control Mask

BIT	VALUE	EXPLANATION
15	Ø	Display the commands (and results) from the file executed by the <ESC>Y command (default).
	1	Do not display the commands (or results) from the file executed by the <ESC>Y command.
16	Ø	When interrupting program execution at a breakpoint, display the address and contents of the breakpoint (default).
	1	When interrupting program execution at a breakpoint, display only the address of the breakpoint.
17	Ø	Display 3 spaces when spacing DDT output (1).
	1	Display DDT output fields at tab stops (1).
34	Ø	The terminal does not have a tab mechanism (2).
	1	The terminal has a tab mechanism (2).
35	Ø	Echo deleted characters (3).
	1	Backspace over deleted characters (3).

(1) If bit 17 is reset (default), DDT displays 3 spaces between
output fields (such as between the address of a location and the
contents of the location), and at the end of display lines. If
bit 17 is set, DDT lines up the output fields in columns
beginning at tab stops (see bit 34).

Figure 4-1 illustrates the two different modes.

(2) If bit 34 is set, DDT displays a tab character (<CTRL/I>)
between fields. If bit 34 is reset, DDT displays enough spaces
to start the field at the next tab stop. When starting up, DDT
checks whether your terminal can handle TAB characters
(<CTRL/I>), and sets this bit accordingly.

(3) When starting up, DDT checks whether your terminal can
backspace to delete characters, and sets this bit accordingly.

To change the settings of the TTY control mask, use the command:

    expr<ESC>1M

where expr evaluates to the required bit pattern.

You can also open the location addressed by <ESC>1M with one of the DDT display commands, and deposit an expression that contains the new bit settings.

Figure 4-1 is an illustration of the effects of bit 17 in the TTY control mask. The code being examined is the first few lines of X.MAC, listed in Figure 2-1. The example is not a complete debugging session; only enough is shown to illustrate the effects of bit 17 of the TTY control mask. The numbers at the left of the DDT display lines are to assist you in following the commentary that follows the display. User input is in lowercase.

Figure 4-1: DDT Session Showing Columnar Output

                          SCREEN DISPLAY

```
1.          DDT
2.          start/    MOVE P,PWORD    x$:    .$b    $g
3.          $1B>>START/    MOVE P,PWORD    $x
4.             P/   -1Ø,,STACK    PWORD/    -1Ø,,STACK
5.          START+1/    MOVEI IDX,TABLE1    $x
6.             IDX/    TABLE1    TABLE1    $1m/    2    1,,2
7.          start$g
8.          $1B>>START/        MOVE P,PWORD    $x
9.                 P/        -1Ø,,STACK        PWORD/   -1Ø,,STACK
1Ø.         START+1/         MOVEI IDX,TABLE1         $x
11.               IDX/      TABLE1    TABLE1
```


                          COMMENTARY

    Line 1:

    o DDT is loaded and waiting for a command.

    Line 2:

    o Enter start/ to examine location start.

    o Enter x<ESC>: to open the symbol table for module X.

    o Enter .<ESC>b to set breakpoint at location START.

    o Enter <ESC>g to begin execution.

    Line 3:

    o DDT displays breakpoint information.

    o Enter <ESC>x to execute the next instruction.

    Line 4:

    o DDT displays results of executing the instruction.

DISPLAYING AND MODIFYING MEMORY

Line 5:

o   DDT displays the next instruction.

o   Enter <ESC>x to execute the instruction.

Line 6:

o   DDT displays the results of executing the instruction.

o   Enter <ESC>lm/ to display and open the TTY control mask.

o   DDT displays the mask. Bit 34 is set.

o   Enter 1,,2<RET> to set bits 17 and 34.

Line 7:

o   Enter start<ESC>g to restart the program.

Line 8:

o   DDT displays the breakpoint information.

o   Enter <ESC>x to execute the instruction.

Line 9:

o   DDT displays the results of executing the instruction.

Line 10:

o   DDT displays the next instruction.

o   Enter <ESC>x to execute the next instruction.

Line 11:

o   DDT displays the results of executing the instruction.

CHAPTER 5

CONTROLLING PROGRAM EXECUTION

5.1 BEGINNING EXECUTION

To begin execution of your program, enter:

    <ESC>G

Your program will run, beginning at its start address. If you have
not set any breakpoints, your program runs to completion, or until it
makes a fatal error. At TOPS-10 command level, you can then use the
DDT command to reenter DDT and examine your program.

You can start or continue program execution at any address with the
command:

    addr<ESC>G

5.2 USING BREAKPOINTS

A breakpoint is a program location that has been altered such that if
your program PC reaches the address of the breakpoint, your program
transfers control to DDT.

When you set a breakpoint with DDT, DDT stores the address of the
breakpoint in an internal table. When you command DDT to begin or
continue program execution, DDT stores the instructions from all
breakpoints in the table, and replaces them with JSRs into a DDT entry
table.

While program execution is suspended at a breakpoint, you can examine
and modify memory, remove breakpoints, insert new breakpoints, execute
individual instructions, and continue program execution.

During this time, the command "<ESC>." returns the value that is the
address of the next instruction to be executed. The command
"<ESC><ESC>." returns a value that is the previous value returned by
"<ESC>.".   When you first receive control at the breakpoint, "<ESC>."
returns the address of the breakpoint and "<ESC><ESC>." returns zero.
Before you start execution with <ESC>G, "<ESC>." and "<ESC><ESC>." are
illegal commands (if you try to execute them, DDT sounds the terminal
buzzer or bell and displays a question mark).

                              NOTE

        This manual uses the term "$." to represent the value
        returned by the command "<ESC>.", and the term "$$."
        to represent the value returned by the command
        "<ESC><ESC>.".

You can set up to 12 breakpoints at a time (this is a DDT assembly parameter) in your program. These breakpoints are numbered 1 through 12. There is also one breakpoint (the <u>unsolicited breakpoint</u>, numbered zero) that can be used by your MACRO program to "call" DDT.

Each breakpoint has several internal DDT locations associated with it, which contain information to control DDT action with respect to the breakpoint. You can examine and modify these DDT locations with the same DDT commands that you use to examine and modify locations in your user program. <ESC>nB is a command that returns the value that is the address of the first DDT word associated with breakpoint n. The symbol $nB is used here to represent that address.

Table 5-1 contains a list of the breakpoint locations of interest to the user, and their contents.

Table 5-1:   Breakpoint Locations of Interest

LOCATION	CONTENTS
$nB	Address of breakpoint n.
$nB+1	Instruction for conditional breakpoint n.
$nB+2	Proceed count for conditional breakpoint n.
$nB+3	Address of a location to be opened and displayed when the breakpoint is reached.
$nB+4	Address of an ASCIZ DDT command string to be executed when the breakpoint is reached.

When your user-program PC reaches a breakpoint, your program executes the JSR into DDT. When this occurs, DDT does the following:

o   saves your user-program context

o   replaces the JSR instructions at all breakpoints with the original program instructions

o   displays the breakpoint number, breakpoint address, and the contents of the breakpoint (depending on bit 16 of the TTY control mask)

o   sets "$." to the breakpoint address

o   sets "$$." to zero

o   enters the address of the current location (set before you started the program or proceeded from a breakpoint) on the location sequence stack

o   changes the current location to the breakpoint

o   waits for you to give a DDT command

When you command DDT to restart or continue program execution, DDT does the following:

- o saves the program instructions from all breakpoints

- o replaces the program instructions at all breakpoints with JSR instructions to DDT

- o if you have not executed the instruction at the breakpoint with <ESC>X, DDT simulates execution of the instruction at the breakpoint

- o restores your user-program context

- o DDT performs a JRSTF (if in section zero, otherwise XJRSTF) to the next instruction to be executed

5.2.1 Setting Breakpoints

To set a breakpoint, enter:

    addr<ESC>{n}B

where addr is the address where you want to suspend execution (addr can be ".", the command that returns the address of the current location), and n is the number of the breakpoint (and defaults to the lowest unused breakpoint number).

If you do not specify n, it defaults to the lowest available (unset) breakpoint. If you have already set twelve breakpoints, DDT displays "?" and sounds the terminal buzzer or bell.

If you specify n, it must be greater than zero and less than 13. DDT restores the original contents of any (previously set) breakpoint designated as breakpoint n before setting new breakpoint n.

You cannot set more than one breakpoint at the same address. DDT simply sets the same breakpoint again, even if you explicitly specify a breakpoint number the second time.

You cannot set a breakpoint at AC zero.

Assume the following conditions:

- o location LABL1+3 contains the instruction MOVE 1,LABL2

- o breakpoint 2 is set at LABL1+3

If your program reaches LABL1+3 it executes the JSR to DDT, and DDT does the following:

- o saves your user-program context

- o restores the original program instructions to the breakpoints

- o sets "$." to LABL1+3

- o sets "$$." to zero

- o enters the address of the current location on the location sequence stack

o   changes the current location to LABL1+3 (the breakpoint)

o   opens location LABL1+3

o   displays: $2B>>LABL1+3/   MOVE 1,LABL2

To <u>set a breakpoint and have DDT display an additional location</u> when your program reaches the breakpoint, enter:

   addr1<addr2<ESC>{n}B

where addr1 is the location to be displayed, and addr2 is the location of the breakpoint. Follow addr1 with a left angle bracket (<).

Assume the following conditions:

o   location LABL1+3 contains the instruction MOVE 1,LABL2

o   location LABL3 contains value SYMBL1

o   breakpoint 2 was set by the command:

   LABL3<LABL1+3<ESC>B

If your program reaches LABL1+3 it executes the JSR to DDT, and DDT does the following:

o   saves your user-program context

o   restores the original program instructions to the breakpoints

o   sets "$." to LABL1+3

o   sets "$$." to zero

o   enters the address of the current location on the location sequence stack

o   changes the current location to LABL1+3 (the breakpoint)

o   enters the address of the current location (the breakpoint) on the location sequence stack

o   changes the current location to LABL3

o   opens location LABL3

o   displays: $2B>>LABL1+3/   MOVE 1,LABL2   LABL3/   SYMBL1

Note that, because DDT placed the breakpoint address on the location sequence stack, you can enter <ESC><RET> to change the current location back to the breakpoint.

When invoked at a breakpoint, DDT can also <u>execute a string of DDT commands</u> that is stored as an ASCIZ string. To invoke this function, enter:

   {addr1<}addr2>addr3<ESC>{n}B

where addr1 is an optional location to be displayed, addr2 is the address of the ASCIZ string containing the DDT commands, and addr3 is the address of the breakpoint. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>).

This command is legal only if feature-test FTYANK was turned on when DDT was assembled.

If your program reaches addr3 it executes the JSR to DDT, and DDT does the following:

    o saves your user-program context

    o restores the original program instructions to the breakpoints

    o sets "$." to addr3

    o sets "$$." to zero

    o enters the address of the current location on the location sequence stack

    o changes the current location to addr3 (the breakpoint)

    o displays the breakpoint number

    o displays the address and contents of the location at addr3

    o if you give addr1< in the command, DDT does the following:

        > enters addr3 on the location sequence stack

        > changes the current location to addr1

        > opens the location at addr1

        > displays the contents of the location at addr1

    o executes the DDT commands stored as an ASCIZ string at addr2

If you give addr1< in the command, DDT places the breakpoint address on the location sequence stack, and you can enter <ESC><RET> to change the current location back to the breakpoint.

To display the address of any breakpoint, enter:

    <ESC>nB/

where n is the address of the breakpoint. DDT displays the address of breakpoint n, and you can use the examine commands to open and display the instruction at breakpoint n. If breakpoint n is not set, DDT displays zero.

To remove breakpoint n, enter:

    0<ESC>nB

To remove all breakpoints, enter:

    <ESC>B

5.2.2 Proceeding from Breakpoints

After your program has reached a breakpoint, you can continue
execution at "$." by entering:

    <ESC>P

DDT saves the program instructions from all breakpoints, replaces the
program instructions with JSRs to DDT, restores your user-program
context, and if you have not executed any program instructions with
the <ESC>X command, simulates execution of the instruction at the
breakpoint. DDT then executes a JRSTF (in section zero, otherwise DDT
executes an XJRSTF) to the next instruction to be executed.

You can cause the program to start execution at a different location
with the {addr}<ESC>G command, where addr defaults to the program's
start address.

Once your program has reached a breakpoint and DDT has interrupted
execution, you can cause DDT to continue execution but NOT stop at
that breakpoint until your program has reached that breakpoint a
specified number of times. To do this, enter:

    expr<ESC>P

where expr is the proceed count. DDT places expr at location $nB+2,
where n is the number of the breakpoint at which your program has
stopped. DDT resumes execution of your program. Each time your
program reaches breakpoint n, DDT decrements the proceed count stored
at $nB+2. Your program continues execution until:

    o it reaches a different breakpoint

    o it terminates normally

    o it commits a fatal error

    o the proceed count reaches zero

Each breakpoint has an associated automatic proceed flag. If this
flag is set and the program reaches the breakpoint, DDT decrements the
proceed count at $nB+2 (where n is the number of the breakpoint) and
displays the breakpoint information if the proceed count is less than
one. DDT then automatically continues program execution.

The <ESC>P command resets (clears) the automatic proceed flag
associated with the breakpoint at which DDT has suspended program
execution.

To set a breakpoint and set the associated automatic proceed flag,
enter:

    {addr1<}addr2<ESC><ESC>{n}B

where addr2 is the address of the breakpoint and may be ".", addr1 is
an (optional) additional location to be displayed, and n is optional
and defaults to the lowest unused breakpoint.

Each time your program reaches breakpoint n, DDT decrements the associated proceed count, and if the result is less than one, displays:

        $nB>>addr2/   instr

where n is the breakpoint number, addr2 is the address of the breakpoint, and instr is the contents of the word at addr2.

If you entered addr1< when you gave the command, DDT displays:

        $nB>>addr2/   instr   addr1/   contents

where n is the breakpoint number, addr2 is the address of the breakpoint, instr is the contents of the word at addr2, addr1 is the additional location to be displayed, and contents is the contents of the word at addr1.

DDT then automatically continues program execution until:

        o   your program reaches a different breakpoint

        o   your program terminates normally

        o   your program commits a fatal error

        o   you enter any character while your program is at breakpoint n

You can interrupt the automatic proceed function if you enter a character while your program is at breakpoint n. DDT then resets the automatic proceed flag and suspends program execution at the breakpoint. DDT echoes the character that you entered, which is otherwise ignored.

To have DDT execute a DDT command stored as an ASCIZ string when you interrupt the automatic proceed cycle, enter:

        {addr1<}addr2>addr3<ESC><ESC>{n}B

where addr1 is an optional location to be displayed, addr2 is the address of the ASCIZ string containing the DDT commands, and addr3 is the address of the breakpoint. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>).

If your program reaches breakpoint n, DDT displays the requested information and automatically continues program execution. DDT executes the ASCIZ DDT command string stored at addr2 only when you interrupt the automatic proceed cycle by entering any character.

If feature-test FTYANK was not turned on when DDT was assembled, DDT does not execute the command string stored at addr2.

To proceed from a breakpoint and set the associated automatic proceed flag, give the command:

        {expr}<ESC><ESC>P

where expr is the proceed count. DDT stores the proceed count at $nB+2.

5.2.3 Conditional Breakpoints

To cause DDT to interrupt program execution at a breakpoint only if a specific condition is satisfied, you must store a single test instruction or a call to a test routine in DDT's breakpoint table. You can use a test routine in your program, or one that you enter in DDT's patching area. See Chapter 8 (Inserting Patches with DDT) for more information about the patching area. To enter the test instruction (or the call to the test routine), open the DDT location addressed by the command <ESC>nB+1 by entering:

    <ESC>nB+1/

where n is the number of the breakpoint. You must enter n, or DDT interprets the command as <ESC>B, and removes all breakpoints. Deposit the test instruction or the call to the test subroutine. If your program reaches breakpoint n, DDT executes the instruction at $nB+1. DDT then proceeds as follows:

  o If the instruction does not cause a program counter skip, DDT decrements the proceed count at $nB+2. If the result is zero or less, DDT interrupts execution at breakpoint n.

  o If a program counter skip of 1 does occur, DDT interrupts execution at breakpoint n.

  o If the conditional instruction is a call to a subroutine that returns by skipping over two or more instructions, DDT does not interrupt program execution.

If DDT interrupts execution because the test instruction resulted in a program counter skip, DDT displays only one angle bracket after the breakpoint identification, as:

    $3B>LABL1/   MOVE 1,LABL2

5.2.4 The "Unsolicited" Breakpoint

You can cause your MACRO program to "call" DDT by inserting the following instruction in your program:

    JSR $0BPT##

The two pound-signs (##) appended to $0BPT in your MACRO program declare the symbol as EXTERNAL.

                              NOTE

        "$" represents the dollar sign character, which is
        part of the symbol, and is not the DDT echo of the
        ESCAPE key.

You must load DDT.REL with your program or you will get a LINK error (?LNKUGS undefined global symbol) when you load your program. Load DDT.REL with your program as follows (your input is in lowercase; the last line indicates that DDT is loaded and ready to accept your commands):

```
.r link
*/debug filnam/go
DDT
```

where filnam is the name of your MACRO-10 program. You can start your program running with the <ESC>G command. If your program executes the JSR instruction, DDT interrupts program execution and displays:

```
$ØB>>addr+1/    instr
```

where addr+1 is the first location after the JSR $ØBPT instruction, and instr is the contents of that location.

The following sequence of instructions provides another way to have your MACRO program call DDT:

```
CALDDT: SKIPN     .JBBPT
        JRST      [OUTSTR ASCIZ\%DDT not loaded\
                  POPJ P,]
        JSR       @.JBBPT
        POPJ      P,
```

If DDT is loaded in memory (as indicated by a nonzero value in JOBDAT location .JBBPT), you can interrupt your program and transfer control to DDT by entering:

```
<CTRL/D>
```

You must have first used the TOPS-10 command SET DDT BREAKPOINT ON (see the TOPS-10 Commands Reference Manual). The monitor interrupts your program and transfers control to DDT, which displays:

```
$ØB>>CALDDT+3/    POPJ P,
```

You can use the sequence of TOPS-10 commands shown in the following terminal display to get DDT loaded into memory with your program (the commands you enter are in lowercase):

```
.get filnam
.ddt
DDT
<ESC>g
```

If you did not load and save DDT.REL with your program, you will get VMDDT instead of DDT.

5.3   EXECUTING EXPLICIT INSTRUCTIONS

To execute a specific instruction, enter the instruction followed by <ESC>X:

```
instr<ESC>X
```

For example:

```
MOVE 1,@LABL1(3)<ESC>X
```

After executing the instruction, DDT starts a new line and displays:

    o   <>   if in-line execution of instr would result in
                    skipping no instructions.

    o   <SKIP>   if in-line execution of instr would result in
                    skipping 1 instruction.

    o   <SKIP 2>   if in-line execution of instr would result in
                    skipping 2 instructions.

    o   <SKIP 3>   if in-line execution of instr would result in
                    skipping 3 instructions.

<div align="center">NOTE</div>

"In-line execution" means execution of the instruction
as part of normal program flow. The execution of
instructions with this command has no effect on your
user-program PC.

This command restores the prevailing display mode.

5.4 SINGLE-STEPPING INSTRUCTIONS

After your program has transferred control to DDT from a breakpoint,
you can execute program instructions one at a time. This is called
"single-stepping."

"<ESC>." is a command that returns the address of the next instruction
to be executed.

To execute the instruction whose address is returned by "<ESC>.",
enter:

    <ESC>X

For example, breakpoint 3 is set at LABL1+3. If your program PC
reaches LABL1+3, control passes to DDT, which displays:

    $3B>>LABL1+3/   ADD 1,LABL2(2)

Examining the environment, you learn the following:

    o   AC 1 contains 1

    o   AC 2 contains 3

    o   LABL1+4 contains MOVEM 1,@LABL2(3)

    o   LABL2+3 contains SYM3

as shown by the following terminal display (DDT does not display <LF>
or <ESC>):

    $3B>>LABL1+3/   ADD 1,LABL2(2)   <ESC>\   SYM3   <LF>
    LABL1+4/   MOVEM 1,@LABL2(3)   1/   1   <LF>
    2/   3

If you now enter the command <ESC>X, DDT does the following:

    o   changes "$$." to LABL1+3

    o   executes the instruction at LABL1+3

    o   changes "$." to LABL1+4

    o   changes the current location to LABL1+4

    o   opens LABL1+4

    o   displays:

            1/   SYM3+1   LABL2+3/   SYM3
        LABL1+4/   MOVEM 1,@LABL2(3)

If single-stepping an instruction results in a value of ($. minus $$.)
not equal to 1, DDT also begins a new line and displays:

    o   <SKIP>   if ($. minus $$.) = 2

    o   <SKIP 2>   if ($. minus $$.) = 3

    o   <SKIP 3>   if ($. minus $$.) = 4

    o   <JUMP>   if ($. minus $$.) is greater than 4 or less than 1

before displaying the address and contents of the next instruction to
be executed. For example, the following shows a typical terminal
display where you enter <ESC>X to single-step the first instruction at
a breakpoint (DDT echoes <ESC> as $):

    $4B>>LABL1+5/   AOSN 3   /   Ø   <ESC>x
        3/   1

    <SKIP>
    LABL1+7/   MOVEM 1,LABL2

5.5   EXECUTING SUBROUTINES AND RANGES OF INSTRUCTIONS

To execute a series of n instructions beginning with the   instruction
whose address is returned by the command "<ESC>.", enter:

    n<ESC>X

where n is the number of instructions to execute.

DDT then does the following for each instruction:

    o   starts a new display line

    o   executes the instruction

    o   displays the address of any register or memory location
        referenced by the execution of the instruction, and the
        contents of those locations after execution of the
        instruction

    o   changes the current location to the next instruction to be
        executed

o   opens the current location

o   displays the address and contents of the next instruction to be executed

o   changes "$." to the address of the next instruction to be executed

o   changes "$$." to the address of the instruction just executed

To suppress typeout of all but the last instruction executed, use the command:

    n<ESC><ESC>X

where n is the number of instructions to execute.

To continue program execution until the PC (program counter) enters a range of instructions, enter:

    {addr1<}{addr2>}<ESC><ESC>X

where addr1 is the lower end of the range, and addr2 is the upper end. Addr1 defaults to 1 + "$." and addr2 defaults to addr1 + 3. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>).

This command also indicates skips and jumps.

This command is useful for executing a loop or a subroutine call quickly and without typeout.

For example, breakpoint 3 is at location LABL1.

    $3B>>LABL1/   PUSHJ 17,SUBRTN   <ESC><ESC>X   ;Enter <ESC><ESC>X
    <SKIP>   ;SUBRTN returns + 2
    LABL1+2/   ADD 1,2

If you enter a question mark (?) while DDT is executing an <ESC><ESC>X command, DDT displays:

    Executing: addr/   instr

where addr is the address of the next instruction to be executed, and instr is the instruction.

To terminate the execution of the series of instructions, enter any character other than ? (question mark).   DDT does the following:

o   echoes the character

o   displays <SKIP>, <SKIP 2>, <SKIP 3>, or <JUMP>, as appropriate

o   starts a new display line

o   changes the current location to the address of the next instruction to be executed

o   displays the address and contents of the current location

o   opens the current location

o   waits for your next command

5.5.1 Single-Stepping "Dangerous" Instructions

DDT classifies the following as "dangerous" instructions:

   o   instructions that can modify memory

   o   instructions that can cause an arithmetic trap

   o   instructions that can cause a stack overflow

   o   a monitor call or I/O instruction

Before single-stepping one of these instructions, DDT saves and replaces the original instructions at the breakpoints with JSRs to DDT, and restores the full user-program context (including interrupt system and terminal characteristics) before executing the instruction. After executing the instruction, DDT replaces the JSRs at the breakpoints with the original program instructions, and saves the full user-program context.

DDT does not check whether the instruction actually results in one of these conditions, only whether the opcode is in the class of instructions that can cause these effects. This can make executing subroutines and ranges of instructions under DDT control extremely time-consuming.

To execute a subroutine or series of instructions without checking for dangerous instructions, use the command:

       {addrl<}{addr2>}<ESC><ESC>1X

where addrl is the lower end of the range, and addr2 is the upper end. Addrl defaults to 1 + "$." and addr2 defaults to 3 + addrl. Follow addrl with a left angle bracket (<), and addr2 with a right angle bracket (>). This command executes much faster than <ESC><ESC>X, but if the execution of an instruction causes a software interrupt, the error and trap handling mechanism may not function correctly. In addition, program instructions that change or rely on terminal or job characteristics that are also used by DDT can cause unpredictable results.

5.6 USER-PROGRAM CONTEXT

When DDT interrupts your program's execution at a breakpoint, and after it has executed a dangerous instruction during an <ESC>X or <ESC><ESC>X command, it saves the user-program context. The command <ESC>nI, where $0<=n<=10$ (octal), returns the address of the word that contains the information for "function" n. You can use this address to display and modify these values. Most of these values are useful only in executive mode. DDT displays the address of the word containing the information for function n as:

       $I+n

where $1<=n<=10$ (octal). If n = 0, DDT displays only $I.

Table 5-2 lists the functions.

Table 5-2:   User-Program Context Values

FUNCTION	VALUE
0	Executive mode CONI PI.
1	Executive mode PI channels turned off.
2	Executive mode CONI APR.
3	User PC flags.
4	User PC address.
5	EPT page address.
6	UPT page address.
7	CST base virtual address.
10	SPT base virtual address.

DDT restores the user-program context whenever you execute <ESC>G, <ESC>P, and when you execute <ESC>X, or <ESC><ESC>X of dangerous instructions.

Functions 5 through 10 (octal) affect DDT's interpretation of your program's virtual address space. You can alter DDT's interpretation of your program's virtual address space with the physical and virtual addressing (<ESC>nU) commands described in Chapter 11 (Physical and Virtual Addressing Commands). However, any alterations that you make do not become part of your user-program context, and do not affect TOPS-10's interpretation of your program's virtual address space.

DDT also saves and restores the user-program ACs as part of the user-program context. DDT stores the contents of the ACs in an internal "register" block. Any references you make to addresses 0-17 refer to the relative locations in DDT's internal register block. These actions are totally transparent to you.

CHAPTER 6

SEARCHING FOR DATA PATTERNS IN DDT

With DDT you can search for memory locations that contain a specific
value, and conversely, for words that do not contain a specific value.
You can also set a mask to indicate to DDT that only specified bits
are to be considered when performing the search. In addition, you can
search for words that reference a specific address. You can specify a
range within which to perform the search, or default the range to all
of your program's address space. In either case, DDT compares the
contents of each location within the range with the specified value.

To search for words that match a specific value, enter:

        {addrl<}{addr2>}expr<ESC>W

where expr is the value for which DDT is to search, and addrl and
addr2 delimit the range in which the search is to be conducted.
Follow addrl with a left angle bracket (<) and addr2 with a right
angle bracket (>). Addrl defaults to zero and addr2 defaults to
777777 in the current section. Expr can be any legal DDT expression.

DDT does the following:

    o   compares each location (after ANDing it with the search mask)
        within the search range with the 36-bit value resulting from
        evaluating expr

    o   starts the search by comparing the contents of addrl with
        expr

    o   stops the search after comparing the contents of addr2 with
        expr

    o   displays (on a new line) the address and contents of each
        location that matches expr

    o   enters the address of each matching location on the location
        sequence stack

    o   sets the current location to addr2

    o   displays a blank line to indicate the search is over

    o   restores the prevailing display mode

                            NOTE

        If DDT finds more matching locations than there are
        words on the location sequence stack, the earlier
        entries are overwritten.

NOTE

When you use the DDT search functions while running FILDDT, addr2 defaults to 777777 (in the current section) unless:

o   the target is the running monitor or other running job and you are using physical addressing

o   the target is an .EXE file and you are using normal virtual addressing

o   the target is a disk structure or data file

In these cases, addr2 defaults to the last word of the target. See Chapter 9 (FILDDT), and Chapter 11 (Physical and Virtual Addressing Commands), for more information.

To search for words that do NOT match a specified value, enter:

{addrl<}{addr2>}expr<ESC>N

where expr is the value which is not to be matched, and addr1 and addr2 delimit the range within which DDT is to search. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>). Addr1 defaults to zero and addr2 defaults to 777777 in the current section. (If you are using FILDDT, see the note following the description of the <ESC>W search command.) Expr is any legal DDT expression.

DDT functions as for the <ESC>W command, except:

o   DDT searches for and displays the address and contents of any word within the address range that does NOT match the 36-bit value resulting from evaluating expr.

o   DDT enters the locations of non-matching words on the location sequence stack.

To search for references to an address, enter:

{addrl<}{addr2>}expr<ESC>E

where addr1 and addr2 delimit the range of the search, and expr contains the address for which DDT is to search. Follow addr1 with a left angle bracket (<) and addr2 with a right angle bracket (>). Addr1 defaults to zero and addr2 defaults to 777777 in the current section. Expr is any legal DDT expression.

DDT performs an IFIW effective address calculation on the expression contained in each word within the range, and uses the 18-bit result to determine whether there is a match.

Thus, if bits 14-17 (the X field of an instruction) or bit 13 (the I field of an instruction) are nonzero, indexing or indirection may result in DDT finding different search results at different times.

DDT does not check whether the expression is actually an instruction before performing the effective address calculation.

If you enter a question mark (?) while DDT is performing any of the above searches, DDT displays:

      Searching: addr/   value

where addr is the address of the location that will next compare, and value is the contents of addr.

To abort the search, enter any character other than question mark (?). DDT stops searching, and waits for more input. The character that you enter to terminate the search is otherwise ignored.

Each of the above search commands restores the prevailing display mode.

<ESC>M is a command that addresses a DDT location that contains a search mask used to prevent specified bits in the memory word from being considered during the search. This mask is used only by <ESC>W and <ESC>N, not by <ESC>E. DDT logically ANDs the search mask with the memory word before making the comparison, but does not change the memory word. If DDT finds a match, it displays the entire word.

DDT sets the search mask to 777777,,777777 (compare all 36 bits) by default.

To set the search mask, enter:

      expr<ESC>M

where expr evaluates to the required bit pattern.

For example, to search for all of the RADIX50 references to MAIN. (user input is in lowercase):

      <ESC><ESC>5t   ;Set typeout mode to RADIX50.
      37777,,777777<ESC>m   ;Ignore the left 4 bits.
      main.<ESC>5"<ESC>w   ;Enter RADIX50 symbol, start search.

      4112/   4 MAIN.   ;DDT displays match found.
      4775/   0 MAIN.   ;DDT displays match found.
                                  ;Search over, DDT displays blank line.

You can also examine and modify the search mask with the examine and deposit commands described in Chapter 4 (Displaying and Modifying Memory).

CHAPTER 7

MANIPULATING SYMBOLS IN DDT

7.1 OPENING AND CLOSING SYMBOL TABLES

Each separate program module has its own symbol table. When
displaying a value symbolically, if more than one symbol is defined
with that value, DDT displays the first global symbol found. When
searching for a symbol, DDT searches the "open" symbol table first.
For display purposes, DDT treats local symbols found in the open
symbol table as global symbols. DDT appends a pound-sign (#) to local
symbol names that it finds in a symbol table that is not open. For
example:

    SYMBL1#

where SYMBL1 is a local symbol that DDT found in a symbol table that
is not open.

If you enter an expression that contains a symbol that is defined in
more than one of your program modules, DDT uses the value of the
symbol that is contained in the open symbol table. If the symbol is
not defined in the open symbol table, or if there is no open module
and there is not a global definition of the symbol, DDT displays:

    M

To <u>open</u> <u>the</u> <u>symbol</u> <u>table</u> of a program module, enter:

    name<ESC>:

where <u>name</u> is the name of the program module as specified by the TITLE
pseudo-op in your MACRO-10 program (or the equivalent mechanism in a
higher-level language program). DDT closes any currently open symbol
table and opens the symbol table associated with module <u>name</u>.

To <u>close</u> <u>the</u> <u>open</u> <u>symbol</u> <u>table</u>, enter:

    <ESC>:

7.2 DEFINING SYMBOLS

To redefine a symbol or to create a new symbol in the current symbol
table, enter:

        expr<symbol:

where _expr_ is any legal DDT expression, and _symbol_ is the symbol name.

To define _symbol_ as the address of the open location, enter the
command:

        symbol:

If there is no open location, DDT uses the address of the last
location that was open. DDT defines _symbol_ as a global symbol. If
you previously used _symbol_ as an undefined symbol, DDT inserts the
correct value in all the places you referenced _symbol_, and removes
symbol from the undefined symbol table.

7.3 SUPPRESSING SYMBOL TYPEOUT

To prevent a symbol from being displayed, enter:

        symbol<ESC>K

where _symbol_ is the symbol to be suppressed. DDT still accepts _symbol_
as input, but no longer displays _symbol_ as output.

To suppress the last symbol that DDT displayed (in an address, in the
contents of a memory word, or in the evaluation of an expression),
enter:

        <ESC>D

DDT suppresses the last symbol displayed, and then redisplays the
current quantity. DDT does not display its usual three spaces between
the command and the displayed value.

In the following example, assume that symbol SIZE is defined as 3.
User typein is lowercase (<LF> does not appear on the terminal
screen).

        start/   JFCL 0   <LF>
        LOOP/   AOS 1   <LF>
        LOOP+1/   MOVE 2,1   <ESC>dMOVE 2,1   <LF>
        START+3/   MULI 2,SIZE   <ESC>dMULI 2,3

To _reactivate a symbol for typeout_, redefine the symbol. For example,
to reactivate the display of symbol SIZE, above, enter:

        size<size:

Note that SIZE is now defined as a global symbol, even if it was
previously a local symbol.

7.4 KILLING SYMBOLS

To remove a symbol from the symbol table, enter:

    symbol<ESC><ESC>K

DDT removes symbol from the symbol table, and no longer displays
symbol or accepts symbol as input.

7.5 CREATING UNDEFINED SYMBOLS

It is sometimes convenient to use symbols that have not yet been
defined. To create an undefined symbol, enter:

    symbol#

where symbol is the undefined symbol name. DDT enters symbol in the
undefined symbol table. When you later define the symbol, DDT enters
it into the defined symbol table, removes it from the undefined symbol
table, and enters the correct value in all locations where you
referenced the symbol.

You can use undefined symbols only as parts of expressions that you
are depositing to memory. Undefined symbols can be either fullword or
right-halfword values; they cannot be used as the A or X fields of an
instruction, or as the left-halfword of an expression.

7.6 FINDING WHERE A SYMBOL IS DEFINED

To determine the modules in which a symbol is defined, enter:

    symbol?

where symbol is the name of the symbol.   DDT displays the name of each
program module in which symbol is defined. If the symbol is a global
symbol, DDT displays a "G", as:

    sym?
    MAIN. G

DDT does not display G following a local symbol found in the open
symbol table.   When DDT has searched the entire symbol table, it
displays a blank line.

7.7 LISTING UNDEFINED SYMBOLS

To get a list of all currently undefined symbols, enter:

    ?

DDT displays a list containing each undefined symbol.

7.8 LISTING SYMBOLS

To get a list of all symbols starting with a certain character or set
of characters, type:

    sym<ESC>?

where sym is the set of characters that all the symbols start with.
DDT lists all symbols starting with sym, the modules where they are
defined, and the values that symbols are defined as.

For example, typing the following command will list all symbols
starting with TECO:

    TECO<ESC>?

    TECO   TECO   600030
    TECO01   TECO   600040
    TECO02   TECO   600047
    TECO03   TECO   600055

CHAPTER 8

INSERTING PATCHES WITH DDT

To replace the instruction at the open location with a series of instructions and test the new instructions without reassembling your program, you can use the DDT patch function. DDT deposits (in a patching area) the replaced instruction, the new series of instructions, and one or more JUMPA instructions back to the main line of your program. DDT also deposits (in the location that contains the replaced instruction) a JUMPA instruction to the first word of the patch.

To insert a patch that will be executed before the instruction at the open location, enter:

        {expr}<ESC><

where expr is the start of the patching location, and defaults first to PAT.., then to PATCH. EDDT defaults to PAT (an area created during the monitor build), PAT.., and PATCH, in that order. If you do not enter expr, and DDT finds none of the default symbols, DDT uses the value contained in JOBDAT location .JBFF as the address to begin the patch. If expr is a symbol (or the default), DDT updates the symbol table when you terminate the patch, so that the symbol identifies the first word after the patch that you just terminated.

If there is no open location when you initiate the patch, DDT displays "?" and sounds the terminal buzzer or bell.

                                NOTE

        If expr is an AC address, or resolves to a value less
        than 0,,140, DDT displays "?" and sounds the terminal
        buzzer or bell.

When you issue a command to start a patch, DDT saves the address of the open location, closes the open location, changes the current location to the first word in the patching area, and opens that word. DDT also displays the address and contents of the first word of the patching area. For example:

        <ESC><
        PAT../   0

You can now enter the patch, using deposit instructions (the expr<LF> format is probably most useful). DDT updates the current and open locations according to the rules for the command that you use.

To terminate the patch, enter:

{expr}<ESC>{n}>

where expr is the last word of the patch you are entering, and n is the number of returns possible from execution of the patch. The default for n is 2, allowing for a return to 1 + the address of the instruction being replaced, and for a "skip return" to 2 + the address of the instruction being replaced.

When you terminate the patch, DDT deposits the instruction being replaced into the first location following the current location, unless:

- o display is not suppressed by ! AND

- o the current location is zero AND

- o the current location is closed OR you omitted expr

in which case DDT deposits the instruction being replaced into the current location. This prevents the patch from containing unintended null words.

DDT deposits n JUMPA instructions in the locations immediately following the one in which it deposited the original program instruction. The first JUMPA instruction has 1 in its A field, and jumps to 1 + the address of the replaced instruction, the second JUMPA instruction has 2 in its A field and jumps to 2 + the address of the replaced instruction, and so on. The AC numbers are used for identification purposes only. Any JUMPA instruction beyond the sixteenth contains 17 in its A field.

DDT then changes the current location to the location that was open when you initiated the patch, deposits in the current location a JUMPA instruction to the first word of the patch that you entered, and displays the address, original contents, and new contents of the current location. The current location is "open", and can be modified by your next command.

If you default expr, or enter a symbol in the {expr}<ESC>< command, when you terminate the patch, DDT redefines the symbol that identifies the start of the patch. If DDT used the value contained in JOBDAT location .JBFF as the address of the patching area, DDT changes the values contained in .JBFF and the left half of JOBDAT location .JBSA. In all cases, the new value is the address of the memory location after the last word of the patch.

By default, there are 100 (octal) words in the patching area. DDT does not check whether your patch overflows the patching area. You can control the size of the patching area with the /PATCHSIZE switch in LINK.

NOTE

DDT allows you to use other DDT commands while you are in the process of entering a patch. DDT does not check whether the current and open locations are in the patching area, or whether you are entering patch instructions in sequence. When you terminate the patch, DDT deposits the instruction being replaced in the current location regardless of whether the current location is in the patching area.

To <u>insert</u> <u>a</u> <u>patch</u> <u>that</u> <u>will</u> <u>be</u> <u>executed</u> <u>after</u> <u>the</u> <u>instruction</u> at the
open location, enter:

    {expr}<ESC><ESC><

where <u>expr</u> is the address of the patching location (PAT.. is the
default).   The results are the same as inserting the patch before the
instruction as above, except:

    o When you open the patch DDT deposits the replaced instruction
       in the first word of the patch.

    o When you terminate the patch, DDT deposits the first JUMPA
       instruction (rather than the instruction being replaced) in
       the first location following the current location unless:

        > display is not suppressed by ! AND

        > the current location is zero AND

        > the current location is closed OR you omitted <u>expr</u>

       in which case DDT deposits the first JUMPA instruction in the
       current location. This is to prevent the patch from
       containing unintended null words.

<div align="center">NOTE</div>

    If <u>expr</u> is an AC address, or resolves to a value less
    than 0,,140, DDT displays "?" and sounds the terminal
    buzzer or bell.

Figure 8-1 illustrates the patching function. The program being
patched is X.MAC (see Figure 2-1). The patch inserts a SKIPN
instruction that is to be executed after the instruction at START+4.

Figure 8-1: Annotated Patching Session

DDT OUTPUT	USER INPUT	EXPLANATION
START+4/ MOVE 2(IDX)		As a result of your last command, DDT displays the contents of START+4.
	<ESC><ESC><	Enter <ESC><ESC>< to start the patch.
PAT../ 0 MOVE 2(IDX)		DDT displays the address and contents of the first word word of the patch area, and deposits the instruction from START+4 in the first word of the patch.
PAT..+1/ 0		DDT displays the address and contents of the next word of the patch area.
	pat..=	Check the address of PAT.." (the first word of the patch area).

Figure 8-1: Annotated Patching Session (Cont.)

DDT OUTPUT	USER INPUT	EXPLANATION
14432		DDT displays the current address of "PAT..".
	skipn 1,0<ESC>2>	Enter the new instruction, and terminate the patch with a normal return and one skip return by entering <ESC>2>.
PAT..+2/ 0 JUMPA 1,START+5		DDT displays the next word of the patch area, then deposits a JUMPA instruction to 1 + the address of the replaced instruction.
PAT..+3/ 0 JUMPA 2,START+6		DDT displays the address and contents of the next word of the patch area, then deposits a JUMPA instruction to 2 + the address of the replaced instruction.
START+4/ MOVE 2(IDX) JUMPA STACK+10		DDT displays the address and original contents of the replaced instruction, then deposits and displays a JUMPA instruction to the first word of the patch. START+4 is the current location, and is "open".
	pat..=	Check the address of the patch area.
14436		DDT updated "PAT..".

Figure 8-2 shows the terminal display as it actually appears when you insert the patch described above. Your input is in lowercase.

Figure 8-2: Terminal Display of Patching After an Instruction

```
START+4/    MOVE 2(IDX)    $$<
PAT../    0    MOVE 2(IDX)
PAT..+1/    0    pat..=14432    skipn 1,0$2>
PAT..+2/    0    JUMPA 1,START+5
PAT..+3/    0    JUMPA 2,START+6
START+4/    MOVE 2(IDX)    JUMPA STACK+10    pat..=14436
```

INSERTING PATCHES WITH DDT

Figure 8-3 shows the terminal display when inserting the same patch before the instruction at START+4. You enter the instruction in the form expr<LF> (user input is lowercase). Note the use of the patch termination command without expr and without n.

Figure 8-3: Terminal Display of Patching Before an Instruction


```
        START+4/    MOVE 2(IDX)    $<
        PAT../    Ø    .=14432    skipn 1,Ø
        PAT..+1/   Ø   $>
        PAT..+1/   Ø   MOVE  2(IDX)
        PAT..+2/   Ø   JUMPA 1,START+5
        PAT..+3/   Ø   JUMPA 2,START+6
        START+4/    MOVE 2(IDX)     JUMPA STACK+1Ø    pat..=14436
```


To abort the patch you are entering, enter:

        <ESC>Ø<

DDT displays 3 spaces (or a tab, depending on the TTY control mask) and changes the current location to the location that was open when you initiated the patch. The symbol that denotes the start of the patching area is unchanged. Any deposits that you made as part of the patch remain in the patching area. This allows you to restart the same patch, or to write over the patch with a new one.

CHAPTER 9

FILDDT

9.1 INTRODUCTION

FILDDT is a utility used to examine and change disk files and physical
disk blocks. You can also use FILDDT to examine monitor crash dumps,
and to examine and change the running monitor or running jobs. With
FILDDT, you can look at .EXE files as if they had been loaded with the
monitor GET command, or as if they were binary data files.

In selecting a disk file, the monitor, or a running job, with FILDDT,
you are really establishing the virtual address space that FILDDT
accesses. When discussing the contents of that virtual address space,
where the contents can be any of the above objects, this chapter uses
the term target.

Once you have accessed a target you can examine and modify it with the
DDT   examine   and   modify   commands,   and   then   save   it with your
modifications. You can use all of DDT's commands for examining and
modifying memory, but you cannot use any commands that cause the
execution of program instructions, such as <ESC>X, <ESC>G, and so on.
If you attempt to execute a program instruction, DDT sounds the
terminal buzzer or bell.

9.2 USING FILDDT

There are two command levels in FILDDT. This document refers to these
two levels as FILDDT command level and DDT command level.

FILDDT command level accepts FILDDT commands to control session
parameters and to select the target.   When at this level, FILDDT
displays the prompt:

      File:

Once you access a target, FILDDT enters DDT command level. At   this
level, use DDT commands to examine and modify the target.

The syntax to use at FILDDT command level is:

        dev:file.ext[path]/switch/switch

where <u>dev:file.ext[path]</u> is the TOPS-10 file specification, and <u>switch</u> is a <u>FILDDT</u> command. The commands are described below. With a <u>FILDDT</u> command you can:

    o   request HELP on FILDDT

    o   specify the target to be examined

    o   establish certain parameters about the operations that you
        can perform (enable patching, for example)

    o   enter DDT command level

A FILDDT command can have more than one of the above effects.

At TOPS-10 command level, start FILDDT by entering:

        R FILDDT

FILDDT enters FILDDT command level and prompts:

        File:

9.2.1 FILDDT Commands

There are two classes of FILDDT-level commands; those that select the
target that FILDDT is to access and those that establish what function
FILDDT is to perform for the target (enable patching, extract symbols,
and treat an .EXE file as data).

The following are the targets (virtual address spaces) that FILDDT can
access, and the commands that select them:

    o   disk files   /F (default)

    o   disk structures   /U

    o   the running monitor   /M

    o   running jobs   /J

To examine a running job or the running monitor, you must have PEEK or
SPY privileges.

The following are the functions you can select for FILDDT to perform
on the target and the commands that select them:

    o   load the file (used with /S)   /F

    o   treat file as pure binary data   /D

    o   enable patching   /P

    o   load symbol table only from file   /S

To patch the running monitor or a running job, you must have POKE
privileges.

To get HELP, enter:

    /H

FILDDT displays a very brief description of the FILDDT commands and
redisplays the File: prompt.

9.2.2 Symbols

To enhance performance, FILDDT uses a symbol table which it builds in
its own address space, rather than one which exists in the target
address space.

FILDDT automatically extracts symbols from the first .EXE file it
loads during a session to build its internal symbol table. Once
FILDDT has an internal symbol table, it ignores any symbols in
subsequently loaded .EXE files unless you use the /S command.

9.2.3 Establishing Formats and Parameters

/F

        Use the /F command in conjunction with the /S command, to load
        the same file from which FILDDT copies the symbol table, as:

            file-spec/S/F

/D

        By default, FILDDT loads .EXE files in virtual memory as if they
        were to be executed. When you give the /D command you can look
        at an .EXE file as if it were a data file. FILDDT then loads the
        entire file (including the .EXE directory blocks) as a binary
        file, starting at virtual location zero.

/P

        The /P command enables patching (lets you modify) the target. If
        you do not enter /P, you can only examine the target.

/S

        The /S command tells FILDDT to copy the symbol table from the
        file you named in the command. FILDDT then again prompts you
        with File:

9.2.4 Selecting the Target

To select a disk file, you need only name the file. This is the
default, and requires no explicit command switch. Enter:

        file-spec<RET>

FILDDT loads the file. If FILDDT does not already have an internal
symbol table and file-spec names an .EXE file, FILDDT builds an
internal symbol by default. FILDDT then enters DDT command level.

/J

> To examine a running job, enter:
>
>> number/J
>
> where number is the job number. You must have PEEK or SPY
> privileges to examine a running job, and POKE privileges to
> modify the monitor or a running job.

/M

> To examine the running monitor, enter the /M command with no file
> specification, as:
>
>> /M
>
> You must have PEEK or SPY privileges to examine the running
> monitor, and POKE privileges to modify the running monitor.

/U

> To examine a disk structure, use the /U command:
>
>> disk-name/U
>
> where disk-name is the logical name of the disk structure. If
> you use the logical name of a multi-disk structure, you can
> examine and patch the entire logical disk. If you use a physical
> disk name, you are restricted to that physical disk.

9.2.5 Exiting FILDDT

When you are through examining and modifying the target, save the
modified file by entering:

> <CTRL/E>

FILDDT closes the file, saving any changes that you have made, and
again prompts you for a file name.

Any symbol table that you have loaded (explicitly or by default)
remains loaded until you specify another with the /S command.

If you have modified symbols, FILDDT also modifies the symbol table of
the disk file, if one of the following occurred:

> o FILDDT automatically loaded the symbol table.
>
> o you loaded the symbol table and entered DDT command level by
>   entering:
>
>> file-spec/S/F

FILDDT sometimes runs out of memory when you use the <CTRL/E> command
to save files without exiting FILDDT. If FILDDT runs out of memory
while loading a file, it displays the message:

> ? Not enough memory for file pages

If FILDDT runs out of space while building a symbol table, it displays the message:

    ? Not enough memory for symbols

To reclaim all of your available memory, exit FILDDT with the <CTRL/Z> command, and then restart FILDDT with the TOPS-10 command R FILDDT. Note that this technique restores standard virtual addressing conditions, as if you had used the <ESC>U command. See Chapter 11 (Physical and Virtual Addressing Commands) for more information about virtual addressing conditions.

To close the file, save all modifications (as with <CTRL/E>, above) and exit from FILDDT, enter:

    <CTRL/Z>

If you exit FILDDT by entering <CTRL/C>, changes that you make to a disk file can still be in FILDDT's output buffer; if so, they will NOT be saved. However, modifications to the monitor and running jobs take effect as you enter them.

When you exit FILDDT with <CTRL/Z>, you can save FILDDT with its internal symbol table. This saves time if you often use FILDDT to debug a specific target (such as the monitor) that has a very large symbol table.

Start FILDDT, load the symbol table, then exit with <CTRL/Z>. Use the TOPS-10 SAVE command to create a copy of FILDDT to be used with that specific file.

CHAPTER 10

EDDT

EDDT is used to debug the monitor. You can run EDDT in executive mode
to debug the running monitor, or in user mode to examine and patch the
monitor .EXE file.

10.1 EXECUTIVE MODE

When you use EDDT in executive mode, it appears as if you were running
the monitor as a user job with DDT merged in. You can set
breakpoints, examine and modify memory, and perform all other DDT
functions. When at a breakpoint, timesharing is suspended, and you
have total control of the system.

You can run EDDT is executive mode only from the CTY. To obtain
faster response than from a printing terminal, you can redirect the
CTY (only on a KL-10) to a DH-11 line that is connected to a video
terminal (see the TOPS-10/TOPS-20 RSX-20F System Reference Manual).

To use EDDT in executive mode, you must invoke it when booting the
monitor. EDDT is built and loaded with the monitor, but if you do not
invoke it when booting the system, it is discarded to save space.
Invoke EDDT by using the /E switch when giving BOOT the name of the
monitor file:

        BOOT>file-spec/E

where file-spec is the name of the monitor file that BOOT is to load.

After receiving the EDDT prompt, enter:

        DEBUG<ESC>G

This tells EDDT to start the monitor in debugging mode, in which EDDT
remains in memory. The monitor goes through a minimal start-up dialog
and then begins timesharing in "DEBUG" mode.

Once timesharing has begun, enter EDDT by typing <CTRL/D> on the CTY.

The monitor traps <CTRL/D>, and calls an internal subroutine that
executes a JSR to DDT location $0BPT (dollar sign, zero, BPT). This
causes EDDT to suspend execution at the unsolicited breakpoint, and to
display:

        $0B>>addr/   instr

where addr is the monitor location following the JSR instruction and
instr is the instruction at addr.

Timesharing is suspended and you can perform any DDT functions.

If the system is hung (and EDDT has been kept in memory), you may be able to enter EDDT through the PARSER. On the CTY, type:

    <CTRL/\>   (control backslash)

When you receive the following prompt:

    PAR>

enter:

    HALT

to freeze the machine state. The parser again prompts PAR>. Now enter:

    EXAMINE KL

to type out the machine state at the time of the HALT. At the parser prompt, enter the command:

    JUMP 401

to enter EDDT. The monitor is frozen in its state at the time of the HALT. The start address of EDDT is 401.

The dialog on the CTY after you enter <CTRL/\> might appear as follows (your input is in lowercase):

    PAR>halt
    PAR>examine kl
    PC/ 1450
    VMA/ 405452
    PI ACTIVE: ON, PI ON: 000, PI HOLD: 020, PI GEN: 000
    NO KL PC FLAGS ARE SET
    PAR>jump 401
    EDDT

You can now use DDT commands to debug the monitor.

                    NOTE

        The <ESC>Y command in executive mode EDDT reads from
        the console paper tape reader.

10.2 USER MODE

You can use EDDT in user mode, to examine and patch the monitor .EXE file, and then use the TOPS-10 SAVE command to save (and rename, as appropriate) the new version of the file. You can then boot and test the patched version later, during stand-alone time.

Invoke user-mode EDDT by first using the TOPS-10 GET command:

    GET filnam

where filnam is the name of the monitor .EXE file. Then use the TOPS-10 DDT command. EDDT prompts:

    DDT

EDDT does not prompt EDDT because you are in user mode rather than executive mode. You can now use DDT commands to set breakpoints, examine, and patch the monitor .EXE file. When you exit EDDT, you can use the TOPS-10 SAVE command to save the .EXE file. If you left breakpoints set when you exited, they are still active when you boot the monitor using the saved .EXE file.

CHAPTER 11

PHYSICAL AND VIRTUAL ADDRESSING COMMANDS

All TOPS-1Ø DDTs (including FILDDT) can do their own page mapping.
The commands described in this chapter allow you to set parameters to
govern the interpretation of the address space which you are
examining. You can control the mapping of the address space you are
examining by choosing to use or bypass the user process table (UPT) or
the executive process table (EPT). You can choose which special pages
table (SPT) to use, and which hardware register block to use. Other
commands allow you to emulate either KI-paging or KL-paging, control
address relocation, and set memory protection limits. In each of the
following commands, the argument (page, addr, n) defaults to zero.

NOTE

The DDT commands <ESC>G, <ESC>P, and <ESC>X have side
effects that affect your control over physical and
virtual addressing. In addition to their normal
functions, these commands also do the following:

o   restore normal virtual addressing as if <ESC>U had
    been given (<ESC>X does NOT do this)

o   set the FAKEAC flag (as if <ESC>U had been given)

o   clear the relocation factor (as if Ø<ESC>8U had
    been given)

o   reset the address-protection address to infinity
    (377777,,777777)

o   restore the active hardware register block to the
    one in use before any <ESC>4U command was given

COMMAND   EXPLANATION

<ESC>U

This command enables memory mapping by standard TOPS-1Ø virtual
addressing. When you give this command, DDT restores the virtual
addressing conditions that were in effect before any
{<ESC>}<ESC>nU (where Ø<=n<=2) commands were given, and sets
DDT's FAKEAC flag, thereby forcing DDT to interpret memory
addresses Ø-17 as DDT's own internal "registers", in which the
user's registers were saved.

<ESC><ESC>U

>This command enables DDT to use actual physical addresses when accessing memory, and clears DDT's FAKEAC flag, causing DDT to interpret memory addresses 0-17 as the hardware registers 0-17. This command is meaningful only when using EDDT in executive mode, or when using FILDDT to look at the running monitor. Although DDT accepts <ESC><ESC>U at other times, this command then produces the same effect as <ESC>U.

The general syntax of the following virtual addressing commands is:

>arg<ESC>nU

where n is the function number of the command, and <u>arg</u> is dependent on the function (see the function descriptions below).

Functions 0, 1, and 2 enable you to control memory mapping by selecting the executive process table (EPT), user process table (UPT), or the section map through which mapping occurs. Setting a mapping condition with any one of these functions (0, 1, and 2) also has the effect of clearing the effects of any prior use of one of these functions (0, 1, and 2).

You can also specify the offset into the special pages table (SPT) with functions 0, 1, and 2 by using the following command:

>arg<ESC><ESC>nU

where arg is the SPT offset, and 0<=n<=2. This form is legal only if KL-paging is in effect.

NOTE

>All forms of <ESC>B and <ESC>X are illegal if you have used the page mapping functions (0, 1, or 2) and have not restored standard mapping with the <ESC>U command.

COMMAND   EXPLANATION

page<ESC>0U

>This command causes memory mapping to occur through the executive process table (EPT) that is located at physical page <u>page</u>.

offset<ESC><ESC>0U

>This command produces the same effect as page<ESC>0U (above), except that <u>offset</u> is an offset (in words) into the SPT.

page1<page2<ESC>0U

>This command is an exception to the general syntax, and is legal only under KI-paging. You can select both the user page table (UPT) and the executive page table (EPT) with this command, where <u>page1</u> is the page number of the UPT, and <u>page2</u> is the page number of the EPT. Follow page1 with a left angle bracket (<).

page<ESC>1U

>This command causes memory mapping to occur through the user process table (UPT) that is located at physical page <u>page</u>. With this command, you can bypass the EPT.

offset<ESC><ESC>1U

    This command produces the same effect as page<ESC>1U (above),
    except that offset is an offset (in words) into the SPT.

page<ESC>2U

    This command causes mapping to occur through the section map at
    physical page page. This command is legal only if KL-paging is
    in effect.

offset<ESC><ESC>2U

    This command produces the same effect as page<ESC>2U (above),
    except that offset is an offset (in words) into the SPT. This
    command is legal only if KL-paging is in effect.

n<ESC>3U

    This command determines whether DDT interprets references to
    memory locations 0-17 as references to hardware registers, or to
    DDT's own internal "registers" (which normally contain the
    user-program ACs), by setting or resetting DDT's FAKEAC flag.

    If n=0, reset FAKEAC flag (use the hardware registers 0-17).
    If n is nonzero, set FAKEAC flag (use DDT's internal registers
    0-17).

    If you enter a nonzero value for n, DDT stores the value -1.

n<ESC>4U

    This command tells DDT to copy hardware register block n
    (0<=n<=7) to its own internal register block, set the FAKEAC
    flag, and use hardware register block n as its own registers. If
    the FAKEAC flag is set when you give this command, DDT first
    restores the contents of its internal register block to the
    hardware register block from which they were copied. This
    command is legal in executive mode EDDT only. Note that the
    microcode uses register block 7, and any attempt to use this
    block produces an almost immediate system crash.

addr<ESC>5U

    This command copies the 20 (octal) word block located at addr to
    DDT's internal "registers" and sets the FAKEAC flag.

addr<ESC>6U

    This command sets the special pages table (SPT) to addr.

addr<ESC>7U

    This command sets the core status table address (CST) to addr.

addr<ESC>8U

    This command sets the address relocation factor to addr. DDT
    adds addr to all user addresses that you enter.

addr<ESC>9U

    This command read-and-write-protects all addresses above addr
    (before adding relocation factor).

n<ESC>10U

> This command controls whether KI paging is enabled or cleared.
>
> If n is nonzero, KI paging is enabled.
> If n=0, KI paging is cleared.
>
> If you enter a nonzero value for n, DDT stores the value -1.
>
> This command is illegal in executive mode EDDT.

n<ESC>11U

> This command controls whether KL paging is enabled or cleared.
>
> If n is nonzero, KL paging is enabled.
> If n=0, KL paging is cleared.
>
> If you enter a nonzero value for n, DDT stores the value -1.
>
> This command is illegal in executive mode EDDT.

| 22<ESC>U
| 23<ESC>U
|
|
|
| These commands specify the type of CPU on which the program is
| being debugged. 22<ESC>U refers to a KL processor. 23<ESC>U
| refers to a KS processor.
|
| For DDT, this command is meaningless, because DDT uses the
| current CPU type. However, these commands may be useful for
| FILDDT.

You can interrogate DDT to determine the last virtual addressing
command that was given for a specific function. The command:

> <ESC>nU

where 0<=n<=11, returns the address of a DDT location that contains
the argument that was given if the command for that function was used,
and returns the default value if that function was not used.   If you
entered a nonzero argument to a command that requires zero or nonzero
values (or if the default is nonzero), this location contains -1. You
can use DDT commands to examine this location.

The command:

> <ESC><ESC>nU

where 0<=n<=2, returns the address of a DDT location that contains
information that indicates which function you used, and whether you
set a page address or an offset. You can use DDT commands to examine
this location. This command is illegal for all functions where n>2.
If you did not enter any commands affecting functions 0-2 since the
last <ESC>U command, the right half of this DDT location word contains
zero. Otherwise, the right half contains n+1, where n is the number
of the command function you used. If you set a page address (with
arg<ESC>nU), bit 1 of this word is reset. If you set an offset (with
arg<ESC><ESC>nU), bit 1 of the word is set.

CHAPTER 12

EXTENDED ADDRESSING

You can load your program with LINK so that the program resides in
extended sections (sections other than Section 0).

If you are debugging a program with extended sections (also called
non-zero sections, or NZS), DDT is automatically loaded into Section
0, started at location 700000. To access non-zero sections, you must
issue the <ESC>4M command to enable intersection (global) breakpoints.
After you use the <ESC>4M command, you can set intersection
breakpoints and execute instructions that cross section boundaries.
(Refer to Section 12.2 for more information about intersection
breakpoints.)

You can load a normal, Section 0 program into a non-zero section (NZS)
using one of the following monitor commands:

      .RUN prog/USE:n
      .GET prog/USE:n

where n is the section number in which to load the program. These
commands are documented in the TOPS-10 Operating System Commands
Manual.

If the program does not already have DDT loaded, then a DDT command
will merge SYS:VMDDT with your program, starting at address 700000 (in
Section 0). In this case, DDT will be able to examine and deposit to
locations in the program, but you cannot set breakpoints or
single-step the program. You can merge VMDDT into a NZS section using
the monitor command:

      .MERGE SYS:VMDDT/USE:n

A subsequent DDT command will then start DDT in section "n". When you
merge DDT into the same section that you specified in the GET/RUN
command, you can immediately set breakpoints and single-step the
program.

When DDT is first started in Section 0, if it detects an extended-section symbol table pointer, it will attempt to map itself into an extended section so that it can access the symbol table. For example, the monitor works this way. DDT is loaded into Section 0 along with the rest of the monitor's low and high segments, and the symbol table is placed in Section 2. The section chosen is the entry-vector section, if any; otherwise, the section that contains the symbol table is used, or Section 1. If DDT can find a section to use, DDT displays the following message:

    [DDT - Section 0 mapped into section n for NZS symbol table access]

where n is the section number that is actually used. If DDT is unable to find a section where it can be mapped, it displays the following message and continues running:

    [DDT - Can't access NZS symbol table(s) from Section 0]

When your program is running in extended sections, DDT searches for the symbol table first in the entry vector section, if any; otherwise it searches JOBDAT (locations .JBSYM, .JBUSY, and .JBHSM) in Section 0.

Although TOPS-10 does not support Program Data Vectors (PDVs), you can set up your own PDV, and use the $5M command to specify the address of a PDV to use for the symbol table vector(s).

12.1 BREAKPOINTS

If DDT is running in a non-zero section, breakpoints can be set in any section.

12.1.1 The Breakpoint Block

To set breakpoints in a section other than the section containing DDT, DDT requires an area of storage in the section containing the breakpoint. This storage area, also known as the "breakpoint block," is required for saving global addresses for transferring control between your program and DDT, and for executing single-stepped instructions that refer to memory locations outside their section.

Each section in your program's memory space that contains a breakpoint must have one breakpoint block, located at the same relative local address within the section, of 100 octal words in length. (If two or more sections of the program are mapped together, the breakpoint blocks must also be mapped together.)

Each breakpoint block is contiguous within the section. Breakpoint blocks cannot be extended across section boundaries, and do not wrap around the end of the section to the beginning of the section.

Your program can refer to locations in the breakpoint block; remember, however, that DDT can overwrite this information.

12.1.2 Enabling and Disabling Intersection Breakpoints

The section-relative (18-bit) address of the breakpoint block is stored in an internal DDT location. The command <ESC>4M returns the address of that location. The symbol $4M refers to the DDT location at the address returned by <ESC>4M. Intersection breakpoints are enabled when $4M contains the locations of the breakpoint block.

To specify the address of the breakpoint block, use the following DDT command:

        n<ESC>4M

where n is the address of the breakpoint block, and can be any valid DDT expression (from 20 to 777700). DDT uses only the right half of n, and changes only the right half of the DDT location at $4M.

To display the location of the breakpoint block, use the following DDT command:

        <ESC>4M/

When this location contains 0, breakpoints are disabled. Therefore, you can use the following command to disable intersection breakpoints:

        0<ESC>4M

While intersection breakpoints are disabled, you cannot set a breakpoint in a section external to DDT, and any breakpoints already set in such a section are lost when you begin program execution with <ESC>P or <ESC>G. For each breakpoint that is lost, DDT displays the following message:

        % CAN'T INSERT $nB - IN NON-DDT SECTION

where n is the breakpoint number.

While intersection breakpoints are disabled, DDT cannot execute the <ESC>X command under the following circumstances:

    o When you try to execute the instr<ESC>X command, and the default section is not the section that contains DDT.

    o When you try to single-step a dangerous instruction and the user program PC is not in the section that contains DDT.

In these cases, when you try to use <ESC>X, DDT rings the terminal bell or buzzer and sets its error message text to:

        Intersection reference and no $4M global breakpoint/execute block

12.2 DISPLAYING SYMBOLS IN NON-ZERO SECTIONS

DDT normally uses right-halfword values when searching symbol tables for symbols to display. However, code linked in a non-zero section has symbols defined with the section number in the left-halfword. DDT uses a 30-bit value when searching for a symbol in the following circumstances:

    o when displaying the address of a location

    o when displaying the contents of a location as an address

    o when displaying the Y field of an instruction

When displaying an address, DDT searches for a symbol defined with the 30-bit value of the address. If such a symbol is not found, DDT displays the address in halfword format.

When displaying the Y field of an instruction, DDT searches for a symbol defined with a 30-bit value consisting of:

    o the section number of the address of the word being displayed

    o the section-relative address contained in the Y field of the instruction

If DDT does not find a symbol defined with that 30-bit value, it looks for a symbol defined with the 18-bit value contained in the Y field of the instruction.

Assume a program with the following conditions:

        Symbol LABL1 is defined as 0,,300
        Symbol LABL2 is defined as 3,,300
        Location 1,,300 contains 3,,300
        Location 1,,301 contains 2,,300
        Location 3,,400 contains 200040,,300
                (MOVE contents of location 300 to AC 1)

When displaying the contents of location 1,,300, DDT displays:

    1,,LABL1/   LABL2

When displaying the contents of location 1,,301, DDT displays:

    1,,LABL1+1/   2,,LABL1

When displaying the contents of location 3,,400, DDT displays:

    LABL2+100/   MOVE 1,LABL2

12.3 DEFAULT SECTION NUMBERS

To reduce the need to type in the section number as part of the address when you specify a location, DDT uses a default section number when you do not specify one. DDT has two section defaulting options:

    o Permanent default section

    o Floating default section

The command <ESC>6M returns the address of an internal DDT location that contains section default information. The symbol $6M refers to the DDT location at the address returned by the command <ESC>6M.

When DDT is in Section 0, the default section number is always 0, regardless of the contents of $6M.

12.3.1 Permanent Default Section

If the value contained in $6M is positive (Bit 0 is reset), the permanent default section option is in effect. DDT then takes the left half of $6M as the section number of any address that you type in without a section number.

Set the permanent default section by typing the following DDT command:

    n,,0<ESC>6M

where n is the section number, and can be any legal DDT expression.

12.3.2 Floating Default Section

If the value contained in $6M is negative (Bit 0 is set), the floating default section option is in effect. This is the default option (at startup, DDT initializes $6M to -1). DDT selects the floating default section as follows:

 o If you enter DDT from its normal start address, DDT sets the default section to one of the following:

    a. the section that contains the program entry vector (if there is an entry vector)

    b. Section 0

 o If you enter DDT from a breakpoint, DDT sets the default section to the section that contains the breakpoint.

 o If you open a local address between 20 and 777777, DDT sets the default section to the section that contains the open address.

 o If you type in an address that contains a section number (including a symbol that is defined with a section number), DDT sets the default section to the one in the address you typed in.

If you exit DDT with <CTRL/C> or <CTRL/Z>, and then reenter DDT, the current location does not change. If you give a command that takes the current location as its default address argument, DDT sets the floating default section to the section of the current location.

In the following example, the DDT screen display is on the left, and explanatory comments are on the right. The entry vector is in section 1. Symbol START is not defined with a section number. User typein is in lowercase.

Screen Display	User Input	Explanation
	3,,place/	Examine location 3,,PLACE.
LABL1		DDT displays the contents.
	<LF>	Type in <LF> to examine the next location.
3,,PLACE+1/ LABL1+2		DDT displays the next location. The floating default section = 3.
	<CTRL/C>	Exit with <CTRL/C>. The current location is 3,,PLACE+1.
@		TOPS-20 prompts you.
	@ddt	Reenter DDT.
DDT		DDT is loaded and ready for your command. The floating default section is 1, because the entry vector is in section 1.
	<LF>	Type in <LF> to examine the next location.
3,,START+2/ LABL1+4		DDT displays the address and contents of the next location. DDT doesn't use the floating default section, because your <LF> command defaults addr to the current location, and uses its section number (3).
	start/	Examine location START. DDT uses the floating default section number because symbol START is defined with no section number.
JFCL 0		DDT displays the contents.
	<LF>	Type in <LF> to examine the next location.
1,,START+1/ MOVE 1,LABL1		DDT displays the address and contents of the location.

12.4 EXECUTING SINGLE INSTRUCTIONS

Instructions that are executed by means of the command:

    instr<ESC>X

where instr is the instruction for DDT to execute, are executed within the current default section. If that section is not the one that contains DDT, DDT uses the breakpoint block in that section to execute instr. If the floating default section option is in effect, and you are unsure of the current default section, use the addr/ command to open a location in the section in which you wish DDT to execute instr. This sets the default section to the section specified by addr.

Instructions that are executed by means of the command:

    instr<ESC><ESC>nX

where instr is the instruction for DDT to execute, are executed within the section specified by n. If you type this command without specifying the section number, DDT uses its current section.

If DDT is to execute the instruction in a section other than the one that contains DDT, intersection breakpoints must be enabled.

If you try to execute instr outside DDT's section while intersection breakpoints are disabled, DDT sounds the terminal buzzer or bell, displays "?", and sets its error string to:

    Intersection reference and no $4M global breakpoint/execute block

12.5 ENTERING PATCHES IN EXTENDED SECTIONS

You cannot type in a patch if a patching area does not exist in the section that contains the word to be replaced. To ensure that there is a patching area for each section that contains user-program code, do one of the following:

    o Reserve the same part of each section for patches, and define
       the patch symbol as 0,,addr, where addr is the local address
       of the patching area.

    o Use only one patching area, and map it into all the sections
       that contain user-program code. Define the patch symbol as
       0,,addr, where addr is the local address of the patching
       area.

    o Define a different symbol for each section's patching area,
       and use the symbol appropriate to the section being patched.

If the left half of expr is 0, DDT defaults the section to the section that contains the open location. If the left half of expr is a value that is not the section that contains the open locations, DDT displays the following message:

    ?CAN'T PATCH ACROSS SECTIONS

APPENDIX A

ERROR MESSAGES

DDT and FILDDT display error messages to indicate the results of your
commands.   DDT sometimes (and FILDDT usually) displays these messages
on the screen, and at other times displays only a question mark.   When
only a question mark is displayed, a location internal to DDT usually
points to a text string that is the error message.   To display the
error message, enter the command:

        <ESC>?

Following is a list of DDT messages together with explanations of what
the messages indicate.

? ABOVE PROTECTION REGISTER LIMIT

        The address of the location you tried to display or modify is
        above the protection register limit, which is set by n<ESC>9U.

? ACTUAL REFERENCE FAILED

        A memory reference failed unexpectedly (the page exists and is
        readable, but the reference failed anyway).

? ADDRESS GREATER THAN 777777

        An address to be mapped through a section table has a nonzero
        section number.   This can occur only if you specified a section
        table with the n<ESC>{<ESC>}2U command.

? ADDRESS BEYOND END OF PHYSICAL MEM

        You attempted to examine a physical memory location beyond the
        end of physical memory. This error occurs only if you have used
        the <ESC><ESC>U command to enable physical addressing.

? BAD FORMAT FOR .EXE FILE

        You specified a file that appears to have an .EXE directory, but
        the directory is badly formatted or DDT cannot read it because of
        some other reason.

? BAD $4M VALUE

        You used the n<ESC>4M command where 777700<n<20.

? BAD POINTER ENCOUNTERED

   DDT does not recognize the type code contained in a page map
   pointer. This can occur only if you are trying to do your own
   virtual address mapping, and used the expr<ESC>{<ESC>}nU command,
   where 0<=n<=2.

? CAN'T BE WRITE ENABLED

   Even though you have automatic write-enable turned on, DDT is
   unable to write-enable a page that exists and is write-protected.

? CAN'T CREATE PAGE

   DDT attempted to create a page and failed, or else is unable to
   attempt to create the page (see the <ESC>1W command).

? CAN'T DEPOSIT INTO SYMBOL TABLE BECAUSE .....

   You tried to define or kill a symbol, but DDT was unable to
   modify the symbol table. Look up the second part of the error
   message in this appendix.

? CAN'T DEPOSIT INTO SYMBOL TABLE BECAUSE DEPOSIT FAILED

   You tried to define or kill a symbol, but DDT was unable to
   modify the symbol table, and cannot identify the specific reason.

% CAN'T INSERT $nB BECAUSE .....

   DDT is not able to access the location where you inserted your
   breakpoint. Look up the second part of the error message in this
   appendix. This occurs before DDT tries to execute <ESC>G,
   <ESC>P, <ESC>X, or <ESC><ESC>X.

% CAN'T INSERT $nB BECAUSE BREAKPOINT IS IN DIFFERENT SECTION

   DDT is not able to access the location where you inserted your
   breakpoint because inter-section breakpoints are not enabled
   (<ESC>4M contains zero). This error occurs before DDT tries to
   execute <ESC>G, <ESC>P, <ESC>X, or <ESC><ESC>X. To enable
   inter-section breakpoints, deposit the breakpoint block address
   in the location addressed by the command <ESC>4M.

% CAN'T INSERT $nB BECAUSE MEM REF FAILED

   DDT is not able to access the location where you inserted your
   breakpoint. DDT is not able to identify the reason. This occurs
   before DDT tries to execute <ESC>G, <ESC>P, <ESC>X, or
   <ESC><ESC>X.

% CAN'T REMOVE $nB BECAUSE .....

   DDT is not able to access the location where you inserted your
   breakpoint. Look up the second part of the error message in this
   appendix. This error occurs when your program enters DDT from a
   breakpoint.

%CAN'T REMOVE $nB BECAUSE BREAKPOINT IS IN DIFFERENT SECTION

> DDT is not able to access the location where you inserted your
> breakpoint, because inter-section breakpoints are not enabled
> (<ESC>4M contains zero). This error occurs when your program
> enters DDT from a breakpoint. To enable inter-section
> breakpoints, deposit the breakpoint block address in the location
> addressed by the command <ESC>4M.

% CAN'T REMOVE $nB BECAUSE MEM REF FAILED

> DDT is not able to access the location where you inserted your
> breakpoint. DDT is not able to identify the reason. This error
> occurs when your program enters DDT from a breakpoint.

%CAN'T SET BREAKPOINT, $4M NOT SET

> You attempted to set a breakpoint in a section other than the one
> containing DDT while inter-section breakpoints were not enabled.

? Device must be a disk unit or a file structure

> The device that you specified to the /U command is not a disk.

? Explicit structure required with /U

> You used the /U command without specifying a device.

? FAILURE ON SWITCHING ADDRESS SPACE

> EDDT (Executive mode EDDT only) encountered an error while trying
> to access the virtual address space where monitor symbols are
> kept.

? FILOP. failure (n) for input file

> The FILOP. call to open the specified file or unit failed with
> error code n.

?Garbage at end-of-command

> FILDDT encountered extra text at a place in the command where
> there should have been only <RET>.

Intersection reference and no $4M global breakpoint/execute block

> Inter-section breakpoints are not enabled, and:
>
> o you tried to execute the command instr<ESC>X but the default
>   section is not the section that contains DDT, or
>
> o you tried to single-step a dangerous instruction but the
>   user-program PC is not in the section that contains DDT.

ERROR MESSAGES

? I/O error

 An I/O error occurred when FILDDT attempted to read or write to
 the file or unit.

? I/O error reading command file

 DDT encountered an I/O error when reading the command file that
 you specified to the <ESC>Y command.

? Illegal job number

 You entered an illegal job number with the /J command.

? Illegal switch "c" specified

 You entered a command other than /D, /F, /H, /J, /M, /P, /S, or
 /U.

? Incorrect symbol table pointer

 FILDDT is unable to read the symbol table specified by the symbol
 table pointer in the file.

? Input device must be a disk

 The device you specified is not a disk.

? Insufficient memory to read EXE file directory

 FILDDT does not have enough free memory to read in the directory
 section of the .EXE file that you specified.

? INVALID DDT INTERNAL ADDRESS

 You addressed an internal location that is not defined. This is
 most likely to occur after you use a command that returns a value
 (such as <ESC>M) to examine a DDT location and then use <LF> or
 <BKSP> to look at nearby memory.

? /M illegal with /D, /F, /S, or /U

 You entered an illegal command along with /M.

M

 You entered a symbol that is defined in more than one module.
 You can select the correct symbol by opening the symbol table
 associated with that module, using the module<ESC>: command.

? NO READ ACCESS

 You tried to display a word in a page to which you do not have
 read access.

? No such file structure

    COMND% and DEVST% think you supplied a disk name in a STRUCTURE command, but no unit with that name was returned by MSTR%.

? Not enough memory for file pages

    FILDDT does not have enough free memory for its file page buffers.

NOTE

        FILDDT sometimes runs out of memory when you use the <CTRL/E> command to save files without exiting FILDDT. If this is the case, exit with the <CTRL/Z> command, and then restart FILDDT with the TOPS-10 command: R FILDDT.

? Not enough memory for symbols

    FILDDT does not have enough free memory to read in the symbol table from the specified .EXE file. See the note above.

? NOT IN CORE

    You tried to map through a page map pointer (in a UPT, SPT, or section table) that addresses a page that is swapped out. This can occur only if you are trying to do your own virtual address mapping, and used the expr<ESC>{<ESC>}nU command, where 0<=n<=2.

% Not in .EXE format -- Data file assumed.

    A file was specified without /D which is not in .EXE file format. FILDDT assumes it is a data file.

? NOT WRITABLE

    You tried to modify a word in a write-protected page. To enable writing on protected pages, use the <ESC>0W command.

? Null filename illegal

    You did not enter a file specification to a command that requires one.

? Null file spec with /F

    You entered a /F command without a file specification.

? OUTPUT error on CLOSE

    An error occurred when FILDDT tried to CLOSE the file.

? PAGE DOES NOT EXIST

   You tried to display a word in a nonexistent page.

? Patching is not enabled

   You attempted to modify a file, a disk, the monitor, or a job but
   did not give the /P command.

% Patching the running monitor is illegal

   You entered an ENABLE PATCHING command and then gave a PEEK
   command.

? PEEK FAILED

   You tried to PEEK at the monitor, but do not have PEEK, POKE, or
   [1,2] privileges enabled.

? /S illegal with /D or /U

   You used the /S command with a command other than /F.

% Symbols cannot be extracted from a data file

   You entered the command filnam/S/F, where the file specified by
   filnam is not in .EXE format. FILDDT assumes it is a data file
   and does not attempt to read in any symbols from the file.

? Symbols cannot be extracted from a data file

   You entered the command filnam/S, where the file specified by
   filnam is not in .EXE format. FILDDT prompts for a new command.

U

   You entered a symbol that DDT cannot locate in any symbol table.
   Cure this by entering the correct symbol, or by defining the
   symbol with the {expr<}symbol: command.

? UNEXPECTED MOVEM FAILURE

   DDT could not deposit to memory even though the page exists and
   is write-enabled.

% Update of file's symbol table failed

   FILDDT was unable to write the modified symbol table back to the
   file after you gave a <CTRL/Z> or <CTRL/E> command.

GLOSSARY

bit

    Bit is a contraction of "binary digit". A bit is the smallest
unit of information in a binary system of notation. It is the
choice between two possible states, usually designated as zero
and one. Bits of data are often used as flags to indicate on/off
or yes/no conditions.

breakpoint

    A breakpoint is a location in a program's executable code that
has been modified so that if the program attempts to execute the
instruction at that location, control passes to DDT before the
instruction is executed.

current display mode

    The current display mode is the mode in which DDT displays the
next word (unless there is an intervening command that changes
the current display mode). Also known as the current typeout
mode.

current quantity

    The current quantity is the most recent of:

    o the last 36-bit quantity that DDT displayed

    o the 36-bit evaluation of the last expression that you entered
       as an argument to a command that deposits to memory

    This value is often used as the default argument for the next
command. Also known as the last value typed.

current typeout mode

    See current display mode.

current location

    The current location is a memory word that has been referenced by
an earlier DDT command. The address of the current location is
the default address for most DDT commands.

current location stack entry

    The location that will become the current location as a result of
the next <ESC><RET> command.

GLOSSARY

current radix

> The current radix is the radix in which DDT displays values.

debugging

> Debugging is the process of finding and removing program errors from programs.

EDDT

> EDDT is the DDT variant that is used to debug executive-mode programs.

FILDDT

> FILDDT is the DDT variant that is used to examine and modify disk files and disk structures. FILDDT is also used to examine and modify the running monitor and other running jobs.

jiffy

> A jiffy is a unit of time defined as one AC (alternating current) cycle. If your line power has a frequency of 60 Hz., a jiffy is one sixtieth of a second (about 16 milliseconds). If your line power has a frequency of 50 Hz., a jiffy is one fiftieth of a second (20 milliseconds).

last value typed

> See current quantity.

location

> A location is a numbered or named place in storage or memory where a unit of data or an instruction can be stored. This manual also uses the terms word and memory word.

location counter

> The location counter is a memory word that contains the address of the current location.

location sequence stack

> The location sequence stack is a stack in which DDT stores the addresses of locations used earlier. DDT uses the stack to access these locations again without having you explicitly enter the address of each of the locations. DDT references these addresses in a last-in, first-out manner.

open location

> The open location is a memory word that you can modify with your next DDT command.

prevailing display mode

> The prevailing display mode is a user-defined default display mode. DDT displays memory words in the prevailing mode unless you specify a temporary display mode. You can restore the prevailing mode with the <RET> command. See Chapter 4 (Displaying and Modifying Memory) for a list of other commands that restore the prevailing display mode.

reset

> Reset refers to the zero condition of a bit or flag. A bit that is zero is said to be reset. To reset is the verb that refers to the act of turning the bit off, "clearing" the bit, or making it zero.

set

> Set refers to the nonzero condition of a bit or flag. A bit that is nonzero is said to be set. To set is the verb that refers to the act of turning the bit on, or making it nonzero.

single-stepping

> Single-stepping is the process of executing program instructions one-at-a-time using DDT, to verify the result of each instruction.

target

> Target refers to the contents of the virtual address space that FILDDT is accessing. The virtual address space may contain a disk structure, a disk file, a running job, or the running monitor.

temporary display mode

> The temporary display mode is a short-term, user-selected display mode which overrides the prevailing display mode. Temporary display mode remains in effect until you enter <RET>, <LF>, <BKSP>, or <TAB>. Also known as the temporary typeout mode.

temporary typeout mode

> See temporary display mode.

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

_____

_____

_____

_____

_____

_____

_____

_____

Did you find errors in this manual? If so, specify the error and the page number.

_____

_____

_____

_____

_____

_____

_____

Please indicate the type of user/reader that you most nearly represent:

☐ Assembly language programmer
☐ Higher-level language programmer
☐ Occasional programmer (experienced)
☐ User with little programming experience
☐ Student programmer
☐ Other (please specify) _____
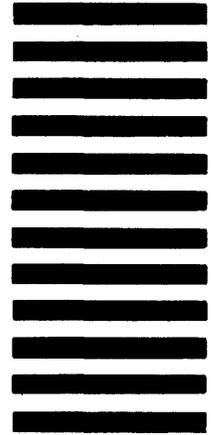
Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code_____
                                                      or Country