BOB

SECTION 3

DDT-10

T. H. Kerridge

UNIVERSITY OF QUEENSLAND
COMPUTER CENTRE

DDT (13)

TECHNICAL MANUAL NO 12

SECTION 3

DDT-10

T. H. Kerridge

MNT-12
9Aug71

CONTENTS

CHAPTER 1
INTRODUCTION

CHAPTER 2
BASIC DDT COMMANDS

CHAPTER 3
DDT COMMANDS

CONTENTS (Cont)

CONTENTS (Cont)

APPENDIX A
SUMMARY OF DDT FUNCTIONS

CONTENTS (Cont)

TABLES

ABSTRACT

DDT was developed at MIT for the PDP-1 computer in 1961. At that time DDT stood for 'DEC Debugging Tape'. Since then, the idea of an on-line debugging program has propagated throughout the computer industry. DDT programs are now available for all DEC computers. Since media other than tape are now frequently used, the more descriptive name 'Dynamic Debugging Technique' has been adopted, retaining the DDT acronym.

This manual describes the use of DDT-10 which is a debugging program designed specifically for the PDP-10. The manual is based on the original Digital DDT-10 documentation but the Computer Centre has rewritten and expanded Chapter 1 to make the manual more easily understood by the novice user. This chapter describes the basic functions of DDT and some of the more common and often used commands. For this reason some material may be repeated in later sections of the manual. Chapter 1 also includes a complete example of DDT being run on a FORTRAN program and subroutine.

As with all other manuals the Centre would appreciate comments on this section so that it may be improved by later revisions.

CHAPTER 1

INTRODUCTION

Under a batch system the only method of debugging a program, apart from desk checking, is to insert statements in the program at strategic points to print out the contents of important variables. This technique serves the dual purpose of checking the values of the variables and providing a trace of the execution path of the program.

When debugging a program from a terminal it is quite possible to use this technique. However, it can be an expensive approach. Each time a test run is set-up the source program has to be edited, the test statements inserted and the program recompiled. It is not always easy to determine where to place these statements as this will depend considerably on the progress of the program.

Because of these limitations a Dynamic Debugging Technique, referred to as DDT, has been developed. This technique is ideally suited to interactive operation. DDT enables a programmer to test a program by executing it, stopping it any stage to examine or change any location of the program. The FORTRAN user can examine and change the contents of variables during program execution. The MACRO user can change both data locations and program instructions.

DDT is invoked by the DDT option in the RUN command which causes the user's program and any associated subroutines to be loaded with a DDT program. The user's program is then run under the control of the DDT program. At the start of execution DDT requests instructions from the user. The repertoire of instructions is large and each of the instructions is explained in later chapters of this manual. However, generally a very limited subset of the repertoire is quite sufficient to enable programmers to derive great benefit from DDT.

Basically DDT provides three facilities:

(a) It allows a user to specify points in his program where he wishes execution to pause.

(b) Either before the program is started, or while it is in a 'pause' state any location may be examined and its contents output in any format (e.g. integer, floating point).

(c) The contents of any location may be changed while the program is in a 'pause' state or prior to execution commencing.

These points where the program is required to pause are called *breakpoints* and are specified to DDT by the user. This is done immediately prior to program execution commencing, or during any pause at a previously defined breakpoint. Up to eight breakpoints may be set simultaneously but as these can be cancelled and re-specified this is not a limitation. While the program is waiting at a breakpoint, the programmer may examine or change the contents of any location (by means of DDT commands).

The following example (sections 1.2.2 and 1.2.3) is intended to help the novice DDT user become conversant with the concepts and purpose of DDT. The example is very simple and shows how a FORTRAN program can be run with DDT. It is not intended to provide examples of all the DDT commands but rather show the basic commands so the user will understand the operation of DDT. When the user has experience in the use of DDT, Appendix A, which gives a summary of all the DDT commands, will prove a useful reference.

1.1 SPECIAL CHARACTERS AND DDT RESPONSES

(a) Rubout

DDT does not behave in the usual way when the 'RUBOUT' key is typed. It has the effect of cancelling the entire command being typed not just the character typed. DDT signals this by typing

```
          XXX
```

(b) Altmode

The symbol '$' refers to the character <altmode> (or <escape> for some Teletypes) throughout this document. This applies to both input from the Teletype and output from DDT to the Teletype.

(c) Return

The <cr> key may be used at the completion of any command to reposition the Teletype at the beginning of the next line.

(d) Responses from DDT

Generally DDT indicates that it is ready to receive commands by typing a <tab>. However this does not occur at the start or after a return and the user should assume DDT is ready and proceed to type a command.

If DDT cannot understand a command it responds with either '?' or 'U'. 'U' is typed if DDT cannot recognize a symbol typed by the user (e.g. a variable name) and '?' is used if the format of the command is not correct or if an attempt is made to set more than eight breakpoints.

1.2 ELEMENTARY DDT FOR FORTRAN USERS

1.2.1 General

Although in practice any location in a program can be referenced by DDT (whether it be a data location or a machine instruction), for those users not conversant with MACRO, it is suggested that only variables be examined or altered and that only statement numbers be used for specifying breakpoints. During the FORTRAN compilation process statement numbers are suffixed by the letter P. Thus statement 16 should be specified to DDT as 16P. Breakpoints may not be set at statement numbers referring to Format statements. This will cause an error to occur when the program is being executed. Although it is quite possible to specify locations before or after a particular statement number this is not advisable unless the user is conversant with the MACRO language and has a macro expansion listing of the FORTRAN program which is being debugged. To run a program using DDT it is necessary to have a relocatable version of the program, that is the program must have been previously compiled. The RUN command is used with DDT specified as an option, e.g.

```
RUN(DDT) MYPROG, MYSUBR
```

1.2.2 FORTRAN Program and Subroutine Listing

This section contains the listing of a FORTRAN program and a FORTRAN subroutine which will be used to explain the basic concepts of DDT.

The main program, named MYPROG, sets values for the variables COM, I, J, K and L, types out these values, calls the subroutine named MYSUBR and on return from the subroutine again outputs the values of COM, I, J, K and L.

The subroutine, MYSUBR, doubles the value of the common variable COM, negates the argument I and outputs these values. It then sums all integers from 1 to 300 into the variable ISUM, prints the value of ISUM and returns to the main program.

The listings of MYPROG and MYSUBR follow:

```
.FORTRAN(LIST) MYPROG<cr>



MYPROG/F4        F4Ø     V23-F3  3-AUG-71        15:17    PAGE 1
```

```
                   C      THIS IS A PROGRAM TO DEMONSTRATE THE USE OF DDT
                   C
                   C
                          COMMON    COM
                          COM=123.45
                    1     I=987
                          J='ABCD'
                          K=-567
                          L="112233445566
                    2     WRITE (6,1Ø) COM,I,J,K,L
                    1Ø    FORMAT ('ØMYPROG VALUES ARE ',/,' COM = ',F8.3,/,'
 I   = ',I8,/
                        1  ,'J  = ',A5,/,'K  = ',I8,/,'L  = ',012)
                    3     CALL MYSUB (I)
                    4     WRITE (6,1Ø) COM,I,J,K,L
                          END

CONSTANTS

        2Ø7755631463    1       4Ø6Ø5Ø3421ØØ    2       112233445566

COMMON

COM     /.COMM./+Ø

SUBPROGRAMS

FORSE.  JOBFF   FLOUT.  FLIRT.  INTO.   INTI.   ALPHO.  ALPHI.  OCTO.
OCTI.   MYSUB   EXIT

SCALARS

COM     Ø               I       7Ø              J       71
K       72              L       73


.F4(L) MYSUBR<cr>


MYSUBR/F4       F4Ø     V23-F3  3-AUG-71        15:24   PAGE 1

                          SUBROUTINE MYSUB(I)
                   C      THIS IS A SUBROUTINE TO DEMONSTRATE THE USE OF DDT
                   C
                   C
                          COMMON    COM
                          COM=COM+COM
                          I=-I
                    2     WRITE (6,1Ø) COM,I
```

```
                  1Ø   FORMAT ('ØMYSUBR VALUES ARE ',/,' COM = ',F8.3,/,'
I    = ',I8)
                       ISUM=Ø
                       DO 6 INDEX=1,3ØØ
                  4    ISUM=ISUM+INDEX
                  6    CONTINUE
                       WRITE (6,2Ø) ISUM
                  2Ø   FORMAT ('ØTHE SUM OF ALL INTEGERS FROM 1 TO 3ØØ =
',I5)
                       RETURN
                       END

GLOBAL DUMMIES

I       72

COMMON

COM     /.COMM./+Ø

SUBPROGRAMS

FLOUT.  FLIRT.  INTO.   INTI.

SCALARS

MYSUB   73             COM     Ø              I          72
ISUM    74             INDEX   75
```

The following section describes a run using DDT on the sample programs listed above.

In the description given below the actual Teletype I/O is given on the left hand side of the page. According to standard convention, the instructions which must be input by the user are underlined.

A description of the run, the use of DDT in the run and an explanation of some of the elementary features of DDT are given on the right hand side of the page, opposite the type-out to which they apply.

It is suggested that a user attempts the following example. Before typing the last $P command it is suggested that the user sets more breakpoints and does some experimentation with this example before using DDT in a 'live' situation with a large program.

Copies of the main program and subroutine may be copied to the user's disk area by the commands.

```
          COPY  $LEARN.MYPROG/F4, user's filename

          COPY  $LEARN.MYSUBR/F4, user's filename
```

It will be necessary for the user to compile both the main program and the subroutine before using the RUN command.

1.2.3 DDT Example

```
.RUN(DDT) MYPROG MYSUBR<cr>
LOADING

LOADER 4K CORE
EXECUTION
```

At this stage the program is waiting for a command from the user.

```
MAIN.$: $$1ØR   2P$B    $G
```

MAIN.$:
This command is in two parts.

(a) *MAIN.*
The name given automatically to the main FORTRAN program.

(b) *$:*
This is the command telling DDT that all symbols, i.e. statement numbers and variable names for the program specified in part (a) of the command, are required.

$$1ØR

Changes the output radix so that all numbers typed by DDT are in decimal, not octal.

2P$B

This command is in two parts.

(a) *2P*
This specifies the statement numbered 2 in the main program.

(b) *$B*
This tells DDT to set a breakpoint in this position. The effect of a breakpoint is that the execution of the program will be held up immediately prior to the execution of the instruction at this statement number.

Breakpoints are numbered 1 to 8. When using this command a breakpoint is automatically allocated the lowest free breakpoint number.

$G

Instructs DDT to start the user's program. The program is now executed until the instruction at statement 2 is reached. Note that if it is not reached for any reason the program will just continue.

```
$1B>>2P COM/    MOVSS 15.,2Ø9715.(13.)  COM$F/  123.44999  <cr>
```

$1B>>2P
This typeout from DDT indicates a breakpoint has been reached.
1B indicates it is breakpoint 1.
2P indicates the location of the breakpoint.
At this stage DDT waits for further commands from the user.

COM/

DDT always attempts to type out the contents of a location as an instruction. If the contents are not a valid instruction it types out the contents as a numeric field.

COM$F/

Instructs DDT to type out the value of the variable COM in floating point format.

```
I/      987.    <cr>
```

I/

Type the contents of field I.

```
J/      ANDM 1.,115776.(8.)     J$T/    ABCD    <cr>
```

J/

Type the contents of field J.

J$T/

Instructs DDT to type the field out as ASCII characters.

```
K/      -567.   <cr>
```

K/

Type the contents of field K.

```
L/      38Ø43.,,15Ø39Ø. $8R:?   L$8R/   112233,,445566  <cr>
```

L/

Type the contents of field L.

$8R:

This is an invalid DDT command and DDT responds with '?'.

L$8R/

Type out contents of field L as an octal number.

```
COM/    MOVSS 15.,2Ø9715.(13.)   =18248839987.   <cr>
```

COM/

Note after a field has been typed out in any format it can be repeated in numeric form by typing an equals sign '='.

```
MYSUB$: 4P$B    $P
```

MYSUB$:

Instructs DDT to refer to symbols in the subroutine MYSUB.

4P$B

Instructs DDT to set a breakpoint at statement 4 in the subroutine.

$P

Proceeds with program execution.
The program and subroutine proceed to output the following results until the breakpoint in the subroutine is reached.

```
MYPROG VALUES ARE
COM =  123.45Ø
I   =       987
J   = ABCD
K   =      -567
L   = 112233445566

MYSUBR VALUES ARE
COM =  246.9ØØ
I   =      -987
$2B>>4P INDEX,,4P$2B    $P
```

$2B>>4P

Typeout from DDT signifying that statement 4 has been reached.

INDEX,,4P$2B

When setting a breakpoint it is possible to tell DDT to type out the contents of a variable automatically. This command tells DDT to reset breakpoint 2 at statement 4P and on reaching this breakpoint type out the contents of variable INDEX.

$P

Proceed with execution.

```
$2B>>4P INDEX/  2.      3$P
```

3$P

Instructs DDT to stop at the current breakpoint on the third occasion the breakpoint is reached.

```
$2B>>4P INDEX/  5.      3$$P
```

3$$P

The double $ instructs DDT to proceed automatically from the breakpoint. In this example the breakpoint is active initially for the third time and thereafter every time the statement number is reached. To get out of this automatic operation type any character while DDT is typing out the breakpoint information and then re-assign the breakpoint as required.

```
$2B>>4P INDEX/  8.

$2B>>4P INDEX/  9.

$2B>>4P INDEX/  1Ø.

$2B>>4P INDEX/  11.

$2B>>4P INDEX/  12.

$2B>>4P INDEX/  13.

$2B>>4P INDEX/  14.
```

```
$2B>>4P  INDEX/  15.

$2B>>4PPPP       INDEX/  16.     PPØ$1BU Ø$1B    $B      $P
```

PPØ$1B

This is illegal, and DDT responds with 'U'.

Ø$1B

This command to DDT removes breakpoint 1. Note that breakpoints can be reassigned by respecifying a particular breakpoint.

$B

Removes all specified breakpoints.

$P

Program will proceed to termination as no breakpoints are specified.

```
THE SUM OF ALL INTEGERS FROM 1 TO 3ØØ = 4515Ø

MYPROG VALUES ARE
COM =  246.9ØØ
I   =     -987
J   = ABCD
K   =     -567
L   = 112233445566

EXECUTION TIME:          1.88 SEC.
TOTAL ELAPSED TIME:      11 MIN. 29.ØØ SEC.

NO. OF ERRORS            ERROR TYPE
3                INTEGER OVERFLOW

?
EXIT
↑C

.
```

1.3 SETTING BREAKPOINTS

LOCA$B	sets next available breakpoint to location LOCA
LOCA$nB	n = 1 - 8 sets the specific breakpoint n
LOCA$$B LOCA$$nB	stops at breakpoint but proceeds immediately after typing breakpoint identified.

```
LOCB,,LOCA$B         when breakpoint is reached the field LOCB is typed out.
LOCB,,LOCA$$B
LOCB,,LOCA$nB
LOCB,,LOCA$$nB
```

If the format $B is used and all breakpoints are already in use then DDT types '?'. If the format $nB is used and breakpoint n is already in use, the breakpoint is reassigned with the new value.

If the breakpoint is accepted DDT responds with a horizontal tab.

When a breakpoint is reached, DDT types

```
$nB>>LOCA

$nB>>LOCA       LOCB/xxxxx
```

Breakpoints can be removed in the following ways:

(a) A specific breakpoint can be removed by
  (i) Respecifying it with a different value
    e.g. LOCA$6B
  (ii) Typing Ø$6B

(b) All breakpoints may be removed by the command $B.

1.4 TYPE OUT MODES

DDT is initialized to type out the contents of locations as 'symbolic Macro instructions'. That is it analyzes the location as if it were an instruction. If however, it decides that because of the value of the location it could not possibly be an instruction it then assumes that the word is holding two constants, that is one constant in each half of the word, and it prints these out separated by two commas ',,'.

It is possible to change the mode of typeout, either permanently or temporarily, and this procedure will now be detailed. The general format of commands to set type out modes is as follows:

(a) (i) $symbol

  (ii) $$symbol

  Command (i) sets the particular mode specified for all locations typed out until a return is typed.

  Command (ii) sets the mode permanently (i.e. until another permanent assignment is made).

  Note that temporary assignments always take priority over the permanent mode assignment.

(b) The output radix of any numeric field typed by DDT can be set as follows:

$nR
$$nR   where $2 \geqslant n \geqslant 10$

In particular, n=10 gives decimal output and DDT prints a period following the number to indicate that the field is in decimal.

(c) To type out a field as a floating point number.

$F
$$F

e.g. $F   LOCB/   123.45

(d) To type out a field as characters (7-bit ASCII codes).

$T
$$T

e.g. $T   LOCA/   ABCDE

(e) In addition to these a field can always be typed as a constant by typing '=' immediately after the contents have been printed.

e.g. $T   LOCA/   ABCDE   =4Ø6Ø5Ø,,342212

(f) There are several other modes which can be specified and these are given in Chapter 4 of this manual.

1.5 TYPE IN MODES

Contents of a field can be altered by typing in the new value immediately after the original contents have been typed.

e.g.   LOCB/   423   546<cr>

in the example the original value of 423 in LOCB has been replaced by the value 546.

The new value to be placed in the particular location can be specified in many different modes. The type in mode is not affected by the current 'type out' mode discussed above. The full list of 'type in' modes is given in a later chapter but the more useful ones are listed below.

(a) To type in an octal value

e.g.   LOCA/   983.   1234<cr>

(b) To type in a fixed point decimal number

e.g.   LOCA/   983.   1Ø869.<cr>

(c) To type in up to 5 characters (7-bit) left justified

e.g.   LOCA/   983.   "/ABCDE/<cr>

Note that the '/' is a delimeter and may be any character that is not in the string to be inserted.

CHAPTER 2
BASIC DDT COMMANDS

The DDT commands most frequently used by programmers are described in this chapter. Many programs are debugged successfully using only these basic commands.

This chapter introduces the main features of DDT to the uninitiated user. Later chapters describe in detail these basic commands, less frequently used commands and other more complex options.

2.1 EXAMINING STORAGE WORDS

By using DDT, a programmer may examine the contents of any storage word by typing the address of the desired word followed immediately by a slash (/). For example, to type out the contents of a location whose symbolic address is CAT, the user types,

```
CAT/
```

DDT now types out the contents (preceded and followed by tabs) on the same line[1].

```
CAT/    MOVEM AC,DOG+21
```

The word labeled CAT is now considered to be opened, and DDT has set its location pointer to point to this address.

2.2 TYPE-OUT MODES

The preceding example showed DDT typing out the contents of location CAT as a symbolic instruction with its address field also relative to a symbol. This is the type-out mode in which DDT is initialized. It is also initialized to type all numbers in the octal radix. The user may ask DDT to retype the preceding quantity as a number in the current radix by typing an equal sign (=). For example[2],

```
CAT/    MOVEM AC,DOG+21  = 202400,,6736
```

DDT has numerous commands which reset the type-out mode permanently, temporarily, or for only one typeout. The modes that can be selected include numeric constants, floating point numbers, ASCII and SIXBIT text modes, and half-word format. Absolute or relative addressing and different radixes may similarly be selected. For example, to change the current type-out mode to ASCII text, the user types the command[3]

```
$T
```

[1] In this manual information typed out by the user is underlined to distinguish DDT output from user-typed input.

[2] The two commas indicate that 202400 is in the left half of CAT, and 6736 is in the right half.

[3] The Teletype keys ALTMODE (ALT), PREFIX (PREFIX), or ESCAPE (ESC) are all equivalent and echo as $.

or, to change the current type-out mode to half-word format, he types

```
$H
```

or, to select decimal numbers in his typeouts, he types

```
$1ØR
```

Using these commands (and others described in Chapter 3), a programmer may examine any location in the mode most appropriate to the information stored there. A semicolon (;) commands DDT to retype the preceding quantity in the current mode. Combining this command with a mode change gives results such as the following:

```
   CAT/  MOVEM AC,DOG+21    $1ØR;  MOVEM AC,DOG+17
or CAT/  MOVEM AC,DOG+21    $H;    2Ø24ØØ,,DOG+21
or TEXT/  ANDM 1,342212(1Ø)  $T;   ABCDE
```

2.3 MODIFYING STORAGE WORDS

Once a word has been opened, its contents may be changed by typing the desired new contents immediately following the typeout produced by DDT. A carriage return will command DDT to make the indicated modification and close the word. For example,

```
CAT/  MOVEM AC,DOG+21   MOVNM AC2,DOG+21↵
```

The carriage return simply closes the previously examined register without opening another[1]. The line feed (↓) may also be used to close a word after examining (and optionally modifying) it. The line feed commands DDT to (1) echo a carriage return, (2) close the current word (making a modification if one was typed), (3) add one to DDT's location pointer, and (4) type out the new pointer value and the contents of that address. Thus, if a line feed had been used in the previous example, the result would be:

```
CAT/  MOVEM AC,DOG+21   MOVNM AC2,DOG+21↓
CAT+1/  AOBJN XR6,LOOP5
```

Location CAT+1 is now open and may be modified if desired.

The vertical arrow (↑) is similar to the line feed command except that the location counter is decremented by one. Therefore, if the user continued the previous example by typing ↑ the result would be

```
CAT+1/  AOBJN XR6, LOOP5↑
CAT/  MOVNM AC2,DOG+21
```

[1] The carriage return command has the additional property of causing temporary type-out modes to revert to permanent mode.

Location CAT is thus displayed and shows the result of the modification made in the previous example.

The tab (⇥) and backslash (\ ) both close the current register and open the address last typed (whether typed by DDT or the user). However, tab sets DDT's location pointer (.) to this new address while backslash leaves it unaltered. A more complex example may clarify the usefulness of these commands.

```
CAT+1/        AOBJN XR6,LOOP5 ⇥

LOOP5/        CAMGE AC2,TABL(XR6)  CAMG AC2,TABL+1(XR6)\SETZI 0=401000,,0↓

LOOP5+1/      JUMPL AC3,FAULT JUMPL AC2,FAULT⇥

FAULT/        JRST 4,FAULT
```

2.4 TYPE-IN MODES

The examples in the preceding section showed modifications made as symbolic instructions in a form identical to MACRO-10 machine language. It is also possible to enter various numbers and forms of text.

Octal values may be typed in as octal integers with no decimal point. Numeric strings with numbers following the decimal point imply decimal floating-point numbers. The E-notation may also be used on floating-point numbers. Some examples are:

Octal:	1234	777777777777	-6	0
Decimal integers:	6789	99999999.	-25.	0.
Floating-point numbers:	78.1	0.249876E-10	-4.00E+20	0.0
Incorrect formats:	76E+2	76.E+2 (instead write 76.0E+2)		

To enter ASCII text (up to five characters, left justified in a word), type a double quote (") followed by any printing character to serve as a delimiter, then type the one to five ASCII characters and repeat the delimiter. For example:

```
"/ABCDE/          (/ is the delimiter)
"ABCDA            (A is the delimiter)
```

Note that the mode of a quantity typed in is determined by the user's input format and is unaffected by any type-out mode settings.

2.5 SYMBOLS

The user's symbol tables are loaded by the Linking Loader when it loads programs and DDT. However, initially DDT is set to treat only global symbols (created by INTERNAL and ENTRY pseudo-ops in MACRO-10) as being defined. This means that only global symbols will be used for relative address typeouts and, likewise, only these globals can be referenced when typing in symbolic modifications. In order to make the local symbols within a particular program available to DDT, the user types the program name (this comes from the MACRO-10 TITLE statement or the FORTRAN IV SUBROUTINE or FUNCTION statement) followed by ALTMODE and a colon ($:). For example, the command

```
ARCTAN$:
```

will unlock the local symbols in the program named ARCTAN. This provision in DDT permits the user to debug several related subroutines simultaneously and reference the local symbol table of each independently without fear of multiply-defined local symbols. If the user's program is not titled, the command MAIN.$: will unlock the local symbol table.

NOTE

> DDT is not quite so stringent on the use of local symbols as indicated above (see Section 5.6). However, the user is advised to unlock symbols with $: until he is fairly familiar with DDT.

The user may also insert symbols into the symbol table. To insert a symbol with a particular value, type the value, followed by a left angle bracket (<), the symbol, and a colon (:). Some examples are

```
707<CONS: 27<S:    12.1E+<NUMB: ADR+12<ADRX:
```

To assign a symbol with a value equal to DDT's location pointer, simply type the symbol followed by a colon. For example,

```
XREF+4/    JRST @ TABL(3)  BRNCH:
```

will cause BRNCH to be defined with the value XFER+4.

2.6 EXPRESSIONS

DDT permits the user to combine symbols and numeric quantities into expressions by using the following characters to indicate arithmetic operators.

+ The plus sign indicates 2's complement addition

- The minus sign indicates 2's complement subtraction

* The asterisk indicates integer multiplication

' The single quote or apostrophe indicates integer division (remainder discarded)--slash cannot be used to indicate division since it has another use in DDT.

As usual in arithmetic expressions, the evaluation proceeds from left to right with multiplication and division performed before addition and subtraction.

2.7 BREAKPOINTS

The breakpoint facility in DDT provides a means of suspending program operation at any desired point to examine partial results and thus debug a program section by section. The simpler facts about breakpoints are presented next; the use and control of conditional breakpoints is deferred to Paragraph 4.2.

2.7.1 Setting Breakpoints

The programmer can automatically stop his program at strategic points by setting as many as eight breakpoints. Breakpoints may be set before the debugging run is started, or during another breakpoint stop. To set a breakpoint, the programmer types the symbolic or absolute address of the word at the location point in which he wants the program to stop, followed by $B. For example, to stop when location 6004 is reached, he types,

```
6ØØ4$B
```

Breakpoint numbers are normally assigned by DDT in sequence from 1 to 8. The user may instead assign breakpoint numbers himself when he sets a breakpoint by typing,

```
$NB
```

when n is the breakpoint numoer ($1 \leq n \leq 8$), for example,

```
CAT+3$4B   DOG+1$7B   6ØØ4$8B
```

When the programmer sets up a breakpoint he may request that the contents of a specified word be typed out when the breakpoint is reached. To do this, the address of the word to be examined is inserted, followed by two commas, before the breakpoint address. Some examples are

```
DOG,,CAT$3B AC1,,LOOP+2$B X,,6ØØ4$8B
```

2.7.2 Breakpoint Restrictions

The locations where breakpoints are set may not

a. be modified by the program
b. be used as data or literals
c. be used as part of an indirect addressing chain
d. contain the user mode monitor command INIT
e. be accumulator 0.

2.7.3 Breakpoint Type-Outs

When the breakpoint location is reached, DDT suspends program execution without executing the instruction at the breakpoint location. DDT then types the breakpoint number and the Program Counter value at the time the breakpoint is reached (this value will differ from the typed-in breakpoint address if the breakpoint is executed by an XCT instruction elsewhere in the program). The format of this typeout is as shown in the following examples:

```
$4B >> CAT+3 $7B >> DOG+1 $8B >> 6ØØ4
```

If the user requested that a specified address be examined at that breakpoint, it will be opened; for example,

```
$3B >> CAT DOG/ SOJGE 3,GOAT+6
```

2.7.4 Removing and Reassigning Breakpoints

The user may remove a breakpoint by typing,

```
Ø$NB
```

where n is the number of the breakpoint to be removed. For example,

```
Ø$2B
```

removes the second breakpoint. All assigned breakpoints are removed by typing

```
$B
```

The user may reassign a breakpoint without formally removing it. Thus, if he has set breakpoint No. 2 at location ADR (via the command ADR$2B) he may reassign No. 2 to LOC+6 by typing LOC+6$2B.

2.7.5 Proceeding From a Breakpoint

Program execution may be resumed (in sequence) following a breakpoint stop by typing the proceed command, $P.

If the user does not wish to stop until the nth time that this breakpoint is encountered he types,

```
N$P
```

Then this breakpoint will be passed n-1 times before a break occurs.

2.8 STARTING THE PROGRAM

The program is started by typing

```
$G
```

This starts the program at the previously specified starting address in location JOBSA. (Typically this is the address from the MACRO-10 END statement.) The programmer may start at any other location by typing that address followed by $G. For example,

```
4ØØØ$G
```

starts the program at the instruction stored at location 4000. BEGIN$G starts the program at the symbolic location BEGIN.

The start command may also be used to restart from a breakpoint stop when it is not desired to continue in sequence from the point where program execution was suspended.

2.9 DELETING TYPING ERRORS

Any partially typed command may be deleted by pressing the RUB OUT key. This causes DDT to ignore any preceding (unexecuted) partial command, and DDT types XXX. The correct command may then be retyped.

2.10 ERROR MESSAGES

If the user types an undefined symbol which cannot be interpreted by DDT, U is typed back. If an illegal DDT command is typed, or a location outside the user's assigned memory area is referenced ? is typed back.

2.11 SUMMARY

As was said in the beginning, these basic commands are sufficient for debugging many programs. Complete descriptions of all DDT commands are explained in the following chapters.

CHAPTER 3
DDT COMMANDS

When DDT is initialized, it is set to type out in the symbolic instruction format with relative addresses, and to type out numbers in octal radix.

3.1 EXAMINING THE CONTENTS OF A PROGRAM STORAGE WORD

To type out the contents of a storage word, the programmer types the address, followed immediately by a slash (/). For example, to examine the contents of a word whose symbolic address is ADR, the user types,

```
ADR/
```

DDT types out the contents on the same line. In this manual, information typed out by DDT is underlined.

```
ADR/   MOVE A,CC1
```

The word labeled ADR is now considered to be opened, and DDT continues to point to this address. The point, or period, character (.) represents DDT's location pointer, and may be used to type out its contents, as in the following command.

```
./   MOVE A, CC1
```

Since we did not change the contents, they are the same, but we used the location pointer to re-examine the currently opened word. Similarly, the programmer may use the period (.) as an arithmetic expression component, such as

```
.+5/   SOJGE  2,ADR+3
```

DDT's location pointer is set to a new value by the / command when immediately preceded by an address. For example,

```
2Ø1/   Ø
```

sets the location pointer to 201. If the user types / without typing an address, the contents of the location addressed in the last typeout are typed.

```
667/   MOVE 1,6   /   Ø
./     MOVE 1,6
```

Location 667 contains the instruction MOVE 1,6. The second slash displays the contents of Accumulator 6, which is zero. This does not change the location pointer, which is still pointing to location 667.

```
ADR/   MOVE A,CC1   /   ADD 2,SUM+7
```

It should also be noted that the spaces, which occur after DDT completes the typing of the contents of ADR, are automatically produced by DDT, not the user.

The left square bracket ([)[1] has the same effect as the slash, (the address immediately preceding the [will be opened). However, [forces the typeout to be in numbers of the current radix.

```
ADR[   11   (OCTAL)
ADR]   9.   (DECIMAL)
```

The right bracket (])[1] has the same effect as the slash except that it forces the typeout to be in symbolic instructions.

```
ADR+23]      MOVE   15,LIST+2
```

The exclamation point (!) works like the slash except that it suppresses type out of contents of locations until either /, [, or] is typed by the user. The LINE FEED (↓) commands DDT to type out the contents of ADR+1.

```
ADR!   MOVE AC,555↓          (1)
ADR+1!↵                      (2)
ADR/   MOVE AC,555           (3)
```

Thus, in step (1) of the example the contents of ADR are not typed out, but the address is opened to modification and MOVE AC,555 has been typed in by the user.

Step (2) of the example shows that the location pointer has been incremented by one and the contents of ADR+1 are not typed out. This is because the exclamation point is still in effect and will continue to take effect until /, [, or] is typed in by the user. In this case, the slash terminates the effect of the exclamation point.

Step (3) shows that the modification (MOVE AC,555) of ADR typed in Step (1) has been accomplished.

3.2 CHANGING THE CONTENTS OF A WORD

After a word is opened, its contents can be changed by typing the new contents following the type out by DDT, followed by a carriage return. For example,

```
ADR/   MOVE A,CC1   MOVE A,CC2↵
```

The carriage return closes the open word, but does not move the location pointer. A LINE FEED (↓) command could also be used to make this modification. A LINE FEED causes a carriage return, adds

[1] On Teletype Models 33 and 35 the left square bracket ([) is produced by holding the SHIFT key down and striking the K key. The right square bracket (]), is produced by holding the SHIFT key down and striking the M key.

one to DDT's location counter (moves the pointer), types out the resulting address and the contents of the new address. Thus, if we conclude our last example with a LINE FEED

```
ADR/      MOVE A,CC1  MOVE A,CC2↓
ADR+1/    ADD 3,CC3
```

ADR+1 is now open, and may be modified by the user.

The vertical arrow (↑)[1] works similarly, except that one is subtracted from the location pointer. The open word is closed (modified if a change is given) and the new address and contents are typed out.

```
ADR+1/     ADD 3,CC3↑
ADR/       MOVE A,CC2
```

Since the vertical arrow subtracts one from the pointer, the resulting address is ADR, and the contents now show the change made in the previous example.

3.3 INSERTING A CHANGE, AND EXAMINING THE CONTENTS OF THE LAST TYPED ADDRESS

The horizontal tab (⇥) causes a carriage-return line feed, then sets the location pointer to the last address typed (the new address if a modification was made) of the instruction in the register just closed. Then DDT types this new address, followed by a slash and the contents of that location, as shown below.

```
ADR5/    JRST ADR1    JRST ADR⇥
ADR/     MOVEM B,CC2 ⇥
CC2/     666
```

The backslash (\)[2] opens the word at the last address typed and types out the contents. However, backslash does not change the location pointer. The backslash closes the previously opened word and causes it to be modified if a new quantity has been typed in.

```
ADR/   MOVE A,CC2   JRST X\ MOVE AC,3
```

The use of the backslash accomplishes two things. First it changes ADR by replacing its contents with JRST X. Second, the backslash causes DDT to type out the contents of X, namely, MOVE AC,3. The location pointer continues to point to ADR, but now location X is open and may be modified if desired.

[1] ↑ is produced by SHIFT-N on Teletype Models 33 and 35. The backspace key may be used instead of ↑ on Teletype Model 37.

[2] \ is produced by SHIFT-L on Teletype Models 33 and 35.

If the line-feed control character and the vertical arrow were used in conjunction with the backslash, the results would be as follows.

```
ADR/       MOVEM B,CC2   MOVE A,CC1\ 1Ø5776 ↓
ADR+1/     MOVE A,C      ↑
ADR/       MOVE A,CC1    \   1Ø5776
```

The following is a summary in table form of these special control characters and their corresponding functions. For example, the chart shows that the forward slash (/) will examine the contents of an address, type out in the current mode, open the address, change the location pointer to the address just opened, but it does not cause a new quantity to be inserted in that address.

Table 3-1
Special Character Functions

Command Character	Type Out Contents	Mode	Address Opened	Change Location Pointer	Insert New Qty If New Qty Has Been Typed
/	Yes	Current	Yes	Yes[1]	No
[	Yes	Numeric			
]	Yes	Symbolic			
!	No	None			
\	Yes[2]	Current	Yes	No	Yes
TAB (⇥)	Yes[2]	Current	Yes	Yes	Yes
↑ or backspace	Yes[2]	Current	Yes	Yes (.-1)	Yes
Line-feed (↓)	Yes[2]	Current	Yes	Yes (.+1)	Yes
Carriage return (↵)	No	None	No (closes)	No	Yes

A ? typed by DDT when examining a location indicates that the address of the location is outside the user's assigned memory area. A ? typed when depositing indicates that the location cannot be written in, because it is either outside the assigned memory area or inside a write-protected memory segment.

[1] If a user-typed quantity preceded.
[2] If ! has not suppressed typeout.

3.4 STARTING THE PROGRAM

The program is started by typing

```
            $G
```

This starts the program with the instruction beginning at the user's previously specified starting address taken from location JOBSA. The programmer may start at any other instruction by typing the address of that instruction followed by $G. For example,

```
            4000$G   OR   ADR+5$G
```

starts the program at the instruction stored at location 4000 or, in the second part, at the symbolic address ADR+5. The start command may also be used to restart from breakpoints when the user does not wish to proceed to the next instruction.

3.5 ONE-TIME TYPEOUTS

These commands cause a single typeout of the opened word in the mode indicated.

3.5.1 Type Out Numeric

Although DDT is initialized to type out in symbolic mode, it is often useful to change to numeric typeout. When the programmer types the equal sign (=), the last expression typed is retyped by DDT in the current radix (initially octal). This is useful when a symbolic typeout is meaningless. Since this usually indicates that numeric data is stored in that word, the user can verify this by typing = and checking the value.

3.5.2 Type Out Symbolic

If a typeout is numeric, and the user wants to examine it in symbolic mode, he types the left arrow (←). The last typed quantity is retyped as a symbolic instruction. The address mode is determined by $A or $R.

3.5.3 Type Out in Current Mode

To retype a typeout in the current mode, the user types a semicolon (;). This may be used, for example, if the user has changed the typeout mode. For example,

```
            TEXT/   ANDM   1,342212 (10)   $T;   ABCDE
            -----                          ---
```

3.6 SYMBOLS

Before DDT commands can be used to reference local symbols in the program Symbol Table, the user should type the program name as specified in the MACRO-10 TITLE statement, or the FORTRAN IV

SUBROUTINE or FUNCTION statement, followed by an ALTMODE and a colon. For example,

```
                    MAIN$:
```

makes the local symbols in the program called MAIN available. Since the user can debug several related subroutines simultaneously, reference to several independent symbol tables is permitted, each of which may use the same local symbols with different values. DDT allows the user to reference unique local symbols in other programs without respecifying the program name with $: (see Section 5.6.2). However, to access a local symbol that is used in several programs, the user must specify the program name to remove the ambiguity. Global symbols, such as those specified in MACRO-10 INTERNAL statements, may always be referenced.

The user may insert (or redefine) a symbol in the symbol table by typing the symbol, followed by a colon. The symbol will have a value equal to the address of the location pointer (.).

```
                    X/   ADD1 3,N   TAG:
```

causes TAG to be defined with the same value as X. All user defined symbols are global.

The user may also directly assign a value to a symbol by typing the value, a left angle bracket (<) and the symbol, terminated by a colon. This is the equivalent of a MACRO-10 direct assignment statement. Some examples are,

```
                    707<CONS:   12.1E+2<NUMB:
                     27<X:      101<MIL:
```

3.7 TYPING IN

To change or modify the contents of a word, the user may type symbolic instructions, numbers, and text characters. Type-ins are interpreted by DDT in context. That is, DDT tests the data typed in to determine whether it is to be interpreted as an instruction, a number (octal or decimal), or text. Typeout mode settings, such as $S, $C, and $nR, do not affect typed input.

The user may type the following:

a. Symbolic Instructions

b. Numbers

   (1) Octal integers
   (2) Fixed-point decimal integers
   (3) Floating-point decimal mixed numbers

c. Text

   (1) Up to five PDP-10 ASCII characters, left justified in a word
   (2) Up to six SIXBIT characters, left justified in a word
   (3) A single PDP-10 ASCII character, right justified in a word
   (4) A single SIXBIT character, right justified in a word

d. Symbols

   Anything that is not a number or text is interpreted by DDT as a symbol.

3.7.1 Typing In Symbolic Instructions

In general, a symbolic instruction is written for insertion by DDT, in the same way the instruction is written as a MACRO-10 source program statement. For example,

```
X/   Ø   ADD   AC1,DATE
```

where a space terminates the operation field, and a comma terminates the accumulator field. For example: (1) In DDT, the operation code determines the interpretation of the accumulator field. If an I/O instruction is used, DDT inserts the I/O device number in the correct place, and (2) indirect and indexed addresses are written, as in MACRO-10 statements, where @ precedes the address to set the indirect bit, and the index register specified follows in parentheses.

```
X/Ø   ADD   4,@NUM(17)
```

To type in two 18-bit halfwords, the left and right expressions are separated by two commas. For example,

```
X/   Ø  A,,B
```

This is similar to the MACRO-10 statement

```
XWD   A,B
```

3.7.2 Typing In Numbers

A typed-in number is interpreted by DDT as octal if it does not contain a decimal point. The following examples are octal type-ins:

```
1234   -1Ø1Ø1
772     777777777777
```

Fixed-point decimal integers must contain a decimal point with no digits following.

```
1234.   -99.   877.
```

Floating-point numbers may be written in two formats. With a decimal point and a digit following the decimal point:

```
1Ø1.1   1234.5   999.Ø   -2.71828
```

Or as in MACRO-10, with E indicating exponentiation:

```
12.ØE+2   77.ØE+5   12.34E2   31.4159E-1
```

3.7.3 Typing In Text Characters

To type in up to five PDP-10 ASCII characters, left justified in an opened word, the user types a quotation mark, followed by any printing delimiting character, then the text characters, and terminated by the delimiting character. The following examples are legal:

```
"/TEXT/    "ABCDEFA
```

In these cases, / and A are the delimiting characters

To type in up to six SIXBIT characters, left justified in an opened word, the user types ALTMODE quotation mark ($"), followed by any delimiting character, then the text characters, and terminated by repeating the delimiting character. Lower case letters are converted to upper case. Characters outside the SIXBIT set are illegal, and DDT types a question mark. The two examples below are SIXBIT type ins.

```
$"/DIVIDE/   $"EXXXXXXE
```

To type in a single PDP-10 ASCII character, right justified in an opened word, the user types a quotation mark, followed by a single ASCII text character, then by an ALTMODE.

```
"Q$   "/$   "?$
```

To type in a single SIXBIT character, right justified in an opened word, the user types an ALTMODE, followed by a quotation mark, a single SIXBIT text character and terminated by an ALTMODE.

```
$"Q$   $"M$   $"$$
```

3.7.4 Arithmetic Expressions

Numbers and symbols may be combined into expressions using the following characters to indicate arithmetic operations.

+ The plus sign means 2's complement integer addition.

- The minus sign means 2's complement integer subtraction.

* The asterisk means integer multiplication.

' The single quote means integer division with any remainder discarded. (The slash has another function.)

Symbols and numbers are combined by +, -, *, ' to form expressions. Examples:

```
6+2
S'2.51+BASE
2*3+1
```

3.8 DELETE

Any partially typed command may be deleted by pressing the RUB OUT or DELete key. This causes DDT to ignore any preceding (unexecuted) partial command and DDT types XXX. The correct command may then be retyped.

3.9 ERROR MESSAGES

If the user types an undefined symbol which cannot be interpreted by DDT, U is typed back. If an illegal DDT command is typed, ? is typed back. Examining or depositing into a location outside the user's assigned memory area causes DDT to type a ?. Depositing in a write-protected high memory segment also results in a ? typeout.

CHAPTER 4
MORE DDT-10 COMMANDS

This chapter describes other type-out modes, conditional breakpoints, searches and additional features. Commands are available to change modes from the initial settings so that numeric data can be typed out in a radix chosen by the user, in floating-point format, in RADIX50 format, as halfwords (two addresses) and as bytes of any size. The contents of a storage word may also be typed out as 7-bit PDP-10 ASCII text, or SIXBIT text characters. (See MACRO-10 Manual, Appendix E.)

Searches can be made in any part of the program for any word, not-word (inequality), or effective address. The user specifies the instruction or data to be searched for and the limits of the search.

Breakpoints can be set conditionally, so that a program stop occurs if the condition is satisfied. In addition, a counter can be set up allowing the user to specify the number of times a breakpoint is passed before a program stop occurs.

4.1 CHANGING THE OUTPUT RADIX

Any radix (≥2) may be set by typing $nR, where n is the radix for the next typeout only, and n is interpreted by DDT as a decimal value. The radix is permanently changed when the double ALTMODE is used in the command $$nR. To change the type-out radix permanently to decimal, the user types,

```
$$1ØR
```

When the output radix is decimal, DDT follows all numbers with a point.

4.2 TYPE-OUT MODES

When DDT-10 is loaded, the type-out modes are initialized to produce symbolic instructions with addresses relative to symbolic locations. For numeric typeouts, the radix is initially set to octal.

These modes may be changed by the user. The duration, or lasting effect of a type-out mode change is also set by the user. Prevailing modes, which are semipermanent, are preceded by two ALTMODEs. Temporary modes are preceded by a single ALTMODE. In addition, some mode changes effect only one typeout, such as the equal sign, which causes DDT to retype the last typed quantity in numeric mode.

In general, prevailing modes are changed by replacing them with another prevailing mode or by reinitializing the system. Temporary modes remain in effect until the user types a carriage return (↵), or re-enters DDT. One-time modes apply only to a single typeout.

4.2.1 Primary Type-out Modes

$S (OR $$S) — Type out symbolic instructions. The address part interpretation is set by $R or $A.

```
$S ADR/   ADD   AC1,TABLE+3
```

$A (OR $$A) — Type out the address parts of symbolic instructions, and both addresses when the mode is halfword, as absolute numbers in the current radix.

```
$A   ADR/   ADD   4Ø02
```

$R (OR $$R) — Type out addresses as relative addresses.

$C (OR $$C) — Type out constants, i.e., as numbers in the current radix.

```
$C ABLE/   254111,,4Ø5Ø
```

If the output radix is octal and the left half is not 0, the word will be divided into halves separated by commas.

$F (OR $$F) — Type out the contents of storage words as floating-point numbers.

```
$F   X/    Ø.1751623ØE-45
```

Unnormalized numbers are typed out as signed decimal integers.

$T (OR $$T) — Type out as 7-bit ASCII text characters. Left-justified characters are assumed unless the leftmost character is null. If the leftmost character is null, then right-justified characters are assumed.

```
$T   REX/   ABCDE
```

$6T (OR $$6T) — Type out as SIXBIT text characters.

```
$6T   HEX/   ABCDEF
```

$5T (OR $$5T) — Type out symbols in radix 50 mode. (See MACRO-10 Manual, Appendix 6.)

```
$5T   13774/   4 CREF = 4ØØØ3,,26155Ø
```

$H (OR $$H) — This command causes the typeout to be in halfwords, the left half separated from the right half by double commas. The address mode interpretation is determined by $R or $A.

```
$A   $H   Z/   45Ø3,,45Ø2
$R   $H   Z/   TABL+14,,TABL+13
```

$NO (OR $$NO) — Type out in n-bit bytes, where n is decimal. (Use the letter O, not zero).

```
$6O BYTS/   22,23, 1, 73, 51, 46
```

As in all DDT typeouts, leading zeros are suppressed.

4.3 BREAKPOINTS

4.3.1 Setting Breakpoints

The programmer can automatically stop his program at strategic points by setting up to eight breakpoints. Breakpoints may be set before the debugging run is started, or during another breakpoint stop. To set a breakpoint, the programmer types the symbolic or absolute address of the word at the location which he wants the program to stop, followed by $B. For example, to stop when location 4002 is reached, he types,

```
4ØØ2$B
```

If all eight breakpoints are in use, DDT will type a question mark. The user may assign breakpoint numbers when he sets a breakpoint by typing ADR $nB, where n is the breakpoint number (1<n<8). For example,

```
SYM$3B   ADR$7B
```

If n is not entered DDT will assign 1 through 8 in sequence. In the previous example, when ADR is reached, DDT types,

```
$7B >> ADR
```

indicating that the break has occurred at location ADR, and breakpoint No. 7 was encountered. The break always occurs before the instruction at the breakpoint address is executed.

If the instruction at the breakpoint location is executed by an XCT instruction, the typeout will show the address of the XCT instruction, not the location of the breakpoint. The program stops at each breakpoint address, and the programmer can then type other commands to examine and debug his program.

When the programmer sets a breakpoint, he may request that the contents of a word be typed out when a breakpoint is reached. To do this, the address of the word to be examined is inserted, followed by two commas, before the breakpoint address.

```
X,,4ØØ2$2B
```

When address 4002 is reached, DDT types out,

```
$2B>>4ØØ2   X/   ADD   AC,Y+2
```

where ADD AC, Y+2 is the contents of X. Location X is left open at this point. Location 0 may not be typed out in this way because a zero argument implies no typeout.

4.3.2 Removing Breakpoints

The user may remove a breakpoint by typing,

```
Ø$NB
```

where n is the number of the breakpoint to be removed. Therefore,

```
Ø$2B
```

removes the second breakpoint. All assigned breakpoints are removed by typing

```
$B
```

The user may reassign a breakpoint. If he has set breakpoint No. 2 at location ADR (ADR$2B), he may reassign No. 2 to ADR+1 by typing ADR+1$2B.

4.3.3 Restrictions for Breakpoints

Breakpoints may not be set on instructions that are

a. Modified by the program

b. Used as data or literals

c. Used as part of an indirect addressing chain

d. The user mode monitor command, INIT

> A breakpoint at any other monitor command will operate correctly, except that if the monitor command is in error, the monitor will type out an error and the Program Counter, but the Program Counter will be internal to DDT and meaningless to the user.

e. A breakpoint may not be assigned to accumulator 0.

4.3.4 Restarting After a Breakpoint Stop

To resume the program after stopping at a breakpoint, the user types the proceed command,

```
$P
```

The program is restarted by executing the instruction at the location where the break occurred. If the user types n$P, this breakpoint will be passed n-1 times before a break can occur; the break will occur the nth time. If n is not specified, it is assumed to be one. If the user proceeds by typing $$P (or n$$P), the program will proceed automatically when the program breaks again. If DDT encounters an XCT loop or the monitor command INIT when proceeding, a question mark will be typed.

Alternatively, the user may restart at any location by typing the start command,

```
ADR$G
```

where ADR is any program address, or $G, which restarts at the previously specified starting address in location JOBSA.

4.3.5 Automatic Restarts from Breakpoints

If the user requests DDT to type out the contents of a word and then continue program execution without stopping, he types two ALTMODES when specifying the breakpoint address.

```
AC,,ADR$$B
```

When ADR is encountered, the contents of AC are typed out and program execution continues. To get out of the automatic proceed mode, type any Teletype key during the typeout, and then remove the breakpoint or reassign it with a single ALTMODE.

4.3.6 Checking Breakpoint Status

The user may determine the status of a breakpoint by examining locations $nB, $nB+1, and $nB+2.

$nB contains the address of the breakpoint in the right half and the address of the location to be examined in the left half. If both halves equal zero, the breakpoint is not in use.

$nB+1 contains the conditional breakpoint instruction. (See Paragraph 4.3.7.)

$nB+2 contains the proceed count.

4.3.7 Conditional Breakpoints

Breakpoints may be set up conditionally in two ways. The user may provide his own instruction or subroutine to determine whether or not to stop, or he may set a proceed counter which must be equal to or less than zero in order for a break to occur.

When a breakpoint location is reached, DDT enters its breakpoint analysis routine consisting of five instructions.

```
SKIPE     $NB+1            ; Is the conditional break instruction 0?
XCT       $NB+1            ; No, execute conditional break instruction
SOSG      $NB+2            ; Decrement and test the proceed counter
JRST      break routine
JRST      proceed routine
```

If the contents of $nB+1 are zero (indicating that there is no conditional instruction), the proceed counter at $nB+2 is decremented and tested. If it is less than or equal to zero, a break occurs; if it is greater than zero the execution of the user's program proceeds with the instruction where the break occurred.

If the conditional break instruction is not zero, it is executed. If the instruction (or the closed subroutine) does not cause a program counter skip, the proceed counter is decremented and tested as above. If a program counter skip does occur, a break occurs. If the conditional instruction is a call to a closed subroutine which returns skipping over two instructions, execution of the user's program proceeds.

If the user wishes a break to occur based only on the conditional instruction, he should set the proceed counter to a large positive number so that the proceed counter will never reach zero.

4.3.7.1 Using the Proceed Counter - If the user wishes to proceed past a breakpoint a specified number of times, and then stop, he inserts the number of passes in $nB+2, which contains the proceed count.

The proceed counter may be set in two ways. The first way is by direct insertion. For example,

```
        $NB+2/   Ø   2Ø
```

sets the counter to 20. The second method is as follows. After stopping at a breakpoint, the proceed count may be set (or reset) by typing the count before the proceed command:

```
        2Ø$P
```

($P will proceed from the interrupted instruction sequence even if the breakpoint has been removed or reassigned.)

4.3.7.2 Using the Conditional Break Instruction - The user inserts a conditional instruction, or a call to a closed subroutine at $nB+1. For example,

```
        $3B+1/   Ø   CAIGE   ACC,15↵
```

or

```
        $4B+1/   Ø   JSA     16, TEST↵
```

When the breakpoint is reached, this instruction or subroutine is executed. If the instruction does not skip or the subroutine returns to the next sequential location, the proceed counter is decremented and tested, as explained in Paragraph 4.2.7. If the instruction skips or the subroutine returns skipping over one instruction, the program breaks. If the subroutine causes a double skip return, the program proceeds with the instruction at the breakpoint address.

Examples of Conditional Breakpoints

If address 6700 is reached and DDT's No. 4 breakpoint registers are as follows:

```
        $4B/        AC1,,67ØØ
        $4B+1/      CAIE AC1,1ØØ
        $4B+2/      2ØØ
```

AC1 contains 100, and DDT types

```
        $4B>67ØØ   AC1/   1ØØ
```

Since AC1 contains 100, the compare instruction skips and the program breaks. If AC1 did not contain 100, $4B+2 would be decremented by one and the user's program would continue running.

If the conditional break instruction transfers to a subroutine which, after the subroutine is executed, returns to the calling location +3, a break will never occur regardless of the proceed counter. Example: If the internal DDT breakpoint registers ($2B and $2B+1) have the following contents, a break would not occur unless accumulator 3 contains 100.

```
$2B/        ADR
$2B+1/      JSR TEST
TEST/       0              (contains PC when JSR to subroutine
                            TEST is made)
TEST+1/     AOS TEST
TEST+2/     CAIE 3,100
TEST+3/     AOS TEST
TEST+4/     JRST @ TEST
```

The subroutine TEST causes a double skip (the return is to the third instruction after the call) in DDT if accumulator 3 does not equal 100. A break will never occur at address ADR (regardless of the proceed counter) unless accumulator 3 contains 100.

4.3.8 Entering DDT from a Breakpoint

When a break occurs, the state of the user's program is saved, the JSR breakpoint instructions are removed, and the programmer's original instructions are restored to the breakpoint locations. DDT types out the number of the breakpoint and a symbol indicating the reason for the break, > for the conditional break instruction, >> for the proceed counter and the address in the user's program where the break occurred.

Example: If address ADR is reached in the user's program and DDT's breakpoint registers contain:

```
$2B/        ADR
$2B+1/      0
$2B+2/      0              (proceed counter contains zero)
```

DDT stops the program and types,

```
$2B>>ADR
```

4.4 SEARCHES

There are three types of searches: the word search, the not-word search, and the effective address search.

Searches can be done between limits. The format of the search command is,

```
            ⎧ W       Word search
a<b >c$     ⎨ N       Not-word search
            ⎩ E       Effective address search
```

where:

a Is the lower limit of the search; 0 is assumed if this argument and its delimiter are not present.

b Is the upper limit of the search. The lower numbered end of the symbol table is assumed if this argument and its delimiter are not present.

c Is the quantity searched for.

The effective address search (E) will find and type out all locations where the effective address, following all indirect and index-register chains to a maximum depth of 64_{10} levels, equals the address being searched for.

Examples:

```
4517<5ØØØ>X$E

INPUT <5ØØØ>7ØØ$E
```

Examples of DDT output, when searching for X in the above example, are as follows.

```
4517/   SETZM X

4721/   MOVE 2,X

5ØØØ/   MOVE 3, @ 4721
```

(indirectly addresses X through address 4721)

The word search (W) and the not-word search (N) compare each storage word with the word being searched for in those bit positions where the mask, located at $M, has ones. The mask word contains all ones unless otherwise set by the user. If the comparison shows an equality, the word search types out the address and the contents of the register; if the comparison results in an inequality, the word search will type out nothing. The not-word search types nothing if an equality is reached. It types the contents of the register when the comparison is an inequality.

Examples:

```
INPT<INPT+1Ø>NUM$W

INPT<INPT+1Ø>Ø$N
```

`$M/` This command types out the contents of the mask register, which is then open. The contents of the mask register are ordinarily all ones unless changed by the user.

`N$M` Inserts n into the mask register.

`Ø$M` `FIRST<LAST>ØW` Lists a block of locations by setting the MASK to zero then performing a word search for zero.

4.5 MISCELLANEOUS COMMANDS

$Q — $Q represents the value of the last quantity typed.

```
ADR/   100   $Q+1↵

ADR/   101
```

INST$X — Causes the instruction INST to be executed.

Example:

JRST ADR$X would cause the user's program to be started at ADR.

There are a number of circumstances when the user will want to zero out certain memory location(s). The following command provides this capability:

FIRST<LAST $$Z — This command will zero out the memory locations between the indicated FIRST address and LAST address inclusively. If the FIRST address is not present, the location 0 is assumed. If the LAST address is not present, the location before the low-numbered end of the symbol table is assumed. In no case will locations 20-137 nor any part of DDT or DDT's symbol table be zeroed.

Example:

```
400/   ADD 2, 12012   X:
```

This puts the symbolic tag X into DDT-10's symbol table and sets X equal to 400, the address of the last register opened.

5.2 DELETING SYMBOLS

There are times when the user will want to restrict or eliminate the use of a certain few defined symbols. The following three ways give the user of DDT-10 these capabilities.

SYMBOL $$K — SYMBOL is killed (removed) in the user's symbol table. SYMBOL can no longer be used for input or output.

Example:

```
X$$K
```

This command removes the symbol X from the symbol table.

SYMBOL $K — This command prevents DDT from using this symbol for typeout; it can still be used for typein. For example, the user may have set the same numeric value to several different symbols. However, he does not wish certain symbol(s) to be typed out as addresses or accumulators.

```
X/  MOVE J, SAV  J$K ← MOVE N, SAV  N$K ← MOVE AC,SAV
```

Since the user does not wish J to be typed out as an accumulator, he types in J$K, followed by a left arrow to type out the contents of X again and MOVE N,SAV is typed out. He then repeats the above process until the desired result, namely AC, is typed out. Any further symbolic typeouts with the same number in the accumulator field of the instruction will type out as AC.

$D — The last symbol typed out by DDT has $K performed on it. The value of the last quantity output is then retyped automatically. For example,

```
A/  MOVE AC,LOC  $D MOVE AC,ABC+1
```

5.3 DDT ASSEMBLY

When improvising a program on-line to the PDP-10 on a Teletype, the user will want to use symbols in his instructions in making up the program. In this and in other situations, undefined symbols may be used by following the symbol with the number sign (#). The symbol will be remembered by DDT from then on. Until the symbol is specifically defined by the use of a colon, the value of the symbol is taken to be zero. Successive use of the undefined symbol causes DDT to type out #. Appending # to all subsequent uses of the symbol enables the user to readily identify undefined (not yet defined by a colon) symbols. When an undefined symbol is finally defined, all previously tagged (#) occurrences of the symbol will be filled in.

Example:

```
        MOVE 2,VALUE#
```

VALUE is now remembered by DDT and may be used further without the user appending the #. If subsequent instructions are given involving VALUE, DDT appends a # automatically to that symbol. Thus VALUE will always appear as VALUE followed by the # (until VALUE is defined).

Example:

```
START!      MOVE 2,VALUE#↓           (user types the #)
START+1!    ADDI 2, 50↓
START+2!    MOVEM 2, VALUE↓
#                                    (DDT types #)
START+3!    JRST VALUE+#1↓           (DDT types # after the plus sign be-
START+4!                             cause only at that point does DDT
                                     realize the symbol VALUE is complete.)
```

Undefined symbols can be used only in operations involving addition or subtraction. The undefined symbols may be used only in the address field.

Example:

```
        MOVEI 2,3*UNDEF#
```

This is an illegal operation - multiplication with a symbolic tag (UNDEF) which has not previously been defined.

The question mark (?) is a command to DDT to list all undefined symbols that have been used in DDT up to that point in the program.

Example:

```
        ?
        VALUE
        UNDEF
```

5.4 FIELD SEPARATORS

The storage word is considered by DDT to consist of three fields: the 36-bit wholeword field; the accumulator or I/O device field; and the address field. Expressions are combined into these three fields by two operators:

Space — The space adds the expression immediately preceding it (normally an op code) into the storage word being formed. It also sets a flag so that the expression going into the address field is truncated to the rightmost 18 bits.

Single Comma	The comma does three things: the left half of the expression is added into the storage word; the right half is shifted left 23 bits (into the accumulator field) and added into the storage word. If the leftmost three bits of the storage word are ones, the comma shifts the right half expression left one more place (I/O instructions thus shift device numbers into the device field). The comma also sets the flag to truncate addresses to 18 bits.
Double Comma	Double commas are used to separate the left and right halves of a word with contents expressed in halfword mode.

The address field expression is terminated by any word termination command or character.

5.5 EXPRESSION EVALUATION

Parentheses are used to denote an index field or to interchange the left and right halves of the expression inside the parentheses. DDT handles this by the following generalized procedure.

A left parenthesis stores the status of the storage-word assembler on the pushdown list and reinitializes the assembler to form a new storage word. A right parenthesis terminates the storage word and swaps its two halves to form the result inside the parentheses. This result is treated in one of two ways:

a. If +, -, ', or * immediately precede the left parenthesis, the expression is treated as a term in the larger expression being assembled and therefore may be truncated to 18 bits if part of the address field.

b. If +, -, ', or * did not immediately precede the left parenthesis, this swapped quantity is added into the storage word.

Parentheses may be nested to form subexpressions, to specify the left half of an expression, or to swap the left half of an expression into the right half.

5.6 SYMBOL EVALUATION

5.6.1 Order of Symbol Table Search

DDT references two symbol tables: (1) a built-in operation table containing the machine language instructions and monitor UUOs (e.g., MOVE, JRST, and INIT) and (2) a symbol table constructed by LOADER during the loading process, containing all the user-defined symbols. When a user types into DDT a symbol, which must be converted into a binary value, DDT has two places to look for the symbol. If the expression (see Section 5.5) constructed has a zero value (the normal case when typing in the operation code of an instruction such as the JRST part of a JRST ADDRESS instruction), DDT looks for the symbol first in its internal operation table, and then, if the symbol is not found, in the LOADER constructed symbol table. If the expression constructed is non-zero, DDT searches the LOADER constructed table first, and then the internal operation table. This method of searching the tables allows instructions such as JRST JRST to work correctly (the first JRST is an operation code, and the second JRST is a user-defined address location).

5.6.2 Order of Symbol Table Search for Symbol Evaluation

When DDT searches the LOADER constructed symbol table to evaluate a symbol typed in, it begins the search by looking through the symbols specified by <program name>$: (see Section 2.5). DDT searches the table in the following order:

1. Looks for the symbol as a local or global symbol in the currently unlocked (by $:) program symbols.
2. Looks for the symbol as a global symbol anywhere in the symbol table.
3. Looks for the symbol as a local symbol in the symbol table of one and only one program.
4. Looks for the symbol as a local symbol that appears in the symbol table of more than one program, but with the same value in each table. (If the symbol appears with different values in different tables, it will not be recognized as defined because there is no way to resolve the ambiguity.)
5. If all the above fail, the symbol is undefined unless it appears in the internal operation table of the DDT.

Fortunately, the searching is accomplished with a single pass over the symbol table.

If one of the several identical local symbols (in step 4) is redefined, it becomes a global, and the symbol is then found at either step 1. or step 2.

This procedure relaxes the requirement of Sections 2.5, 3.6, and the beginning of Chapter 5 on the use of $: to unlock local symbols.

5.7 SPECIAL SYMBOLS

The @ sign sets the indirect bit in the storage word being formed.

Example:

```
MOVE AC,@X
```

5.8 BINARY VALUE INTERPRETATION

When DDT is typing the symbolic equivalent of a binary word or address, it looks for the symbol with a value that best matches the binary. DDT looks through the symbol values in the following order:

1. Searches the symbols of the currently unlocked (by $:) program for a local or global symbol with a value that exactly matches the binary to be interpreted.
2. Searches for a global symbol outside the currently unlocked program with a value that exactly matches the binary to be interpreted.

3. Searches all the other local symbol tables for one or more entries with values that match the binary to be interpreted. If more than one symbolic equivalent is found, the DDT does not use any of them but goes on to step 4. If exactly one symbolic equivalent is found (this includes the case of the same symbol with the same value in more than one local symbol table), then this symbol is used. However, the symbol has a # appended to it to warn the user that this symbol might have a different value in some other local symbol table.

4. Searches the currently unlocked program symbols for a local symbol, and searches the entire symbol table for a global symbol, with the value closest to but less than the binary to be interpreted. The closest symbol is then used for typeout if it is not more than 64 smaller than the binary being interpreted.

If a usable symbol is not found in any of the above steps, the binary is typed out as an integer in the current output radix.

The purpose of this complicated procedure is to output the best symbol without forcing the user to continually respecify the program symbol table names by using $:.

APPENDIX A
SUMMARY OF DDT FUNCTIONS

A.1 TYPE-OUT MODES

The following are used to set the type-out mode:

	Type	Sample Output(s)
Symbolic instructions	$S	ADD 4, TAG+1 ADD 4, 4ØØ2
Numeric, in current radix	$C	69. 1Ø5
Floating point	$F	Ø.125E-3
7-bit ASCII text	$T	PQRST
SIXBIT text	$6T	TSRQPO
RADIX50	$5T	4 DDTEND
Halfwords, two addresses	$H	4ØØ2,,4ØØ5 X+1,,X+4
Bytes (of n bits each)	$NO	$8Ø COULD YIELD Ø,14,237,123,Ø

A.2 ADDRESS MODES

The following are used to set the address made for typeout of symbolic instructions and halfwords (see examples above):

Relative to symbolic address	$R	TAG+1
Absolute numeric address	$A	4ØØ5

A.3 RADIX CHANGE

The following is used to change the radix of numeric type-outs

to n (for n$\geq$2):	$NR	$2R COULD YIELD 11Ø1Ø11ØØØØØØ1ØØØØØØØØØØØ111ØØ1Ø11ØØ

A.4 PREVAILING VS. TEMPORARY MODES

The following are used in prevailing vs. temporary modes:

To set a temporary type-out or address mode or a temporary radix as shown in the commands above, type	$	$C $1ØR

	Type	Sample Output(s)	
To set a prevailing type-out or address mode on a prevailing radix, in the commands above, substitute	$$	$$C $$1ØR	
To terminate temporary modes and revert to prevailing modes, type a carriage return	↲		
Initial prevailing (and temporary) modes are	$$S $$R $$8R		

A.5 STORAGE WORDS

The following are used to examine storage words:

	Type	Sample Output(s)	
To open and examine the contents of any address in current type-out mode	adr/	LOC/	254Ø2Ø,,DDTEND
To open a word, but inhibit the type out of contents	adr!	LOC!	
To open and examine a word as a number in the current radix	adr[	LOC[	254Ø2Ø,,3454
To open and examine a word as a symbolic instruction	adr]	LOC]	JRST @DDTEND
To retype the last quantity typed (particularly used after changing the current type-out mode)	;	$6Ø;	25,4Ø,2Ø,ØØ,34,54
		$6T;	5%Ø <L

A.6 RELATED STORAGE WORD

The following are used to examine related storage words:

To close the current open word (making any modification typed in) and to open the following related words, examining them in the current type-out mode:

To examine ADR+1	↓ (line feed)
To examine ADR-1	↑ (or backspace, on the Teletype Model 37)

	Type	Sample Output(s)
To examine the contents of the location specified by the address of the last quantity typed, and to set the location pointer to this address	→| (TAB)	
To examine the contents of address of last quantity typed, but not change the location pointer	\ (backslash)	
To close the currently open word, without opening a new word, and revert to permanent type-out modes	↵ (carriage return)	

A.7 ONE-TIME ONLY TYPEOUTS

The following typeouts occur only one time:

To repeat the last typeout as a number in the current radix	=	
To repeat the last typeout as a symbolic instruction (the address part is determined by $A or $R)	←	
To type out, in the current type-out mode, the contents of the location specified by the address in the open instruction word, and to open that location, but not move the location pointer	/	
To type out, as a number, the contents of the location specified by the open instruction word and to open that location, but not move the location pointer	[	
To type out, as a symbolic instruction, the contents of the location specified by the open instruction word, and to open that word, but not move the location pointer	]	

A.8 TYPING IN

Current type-out modes do not affect typing in; instead, the following are performed:

To type in a symbolic instruction	`ADD AC1,@DATE(17)`
To type in half words, separate the left and right halves by two commas	`4Ø2,,4Ø3`
To type in octal values	`1234`
To type in a fixed-point decimal integer	`99.`

	Type	Sample Output(s)
To type in a floating-point number	1Ø1.11 77.ØE+2	
To type in up to five 7-bit PDP-10 ASCII characters, left justified, delimited by any printing character	"/ABCDE/	(/ is delimiter)
To type in one PDP-10 ASCII character, right justified	"A$	($ must be ALTMODE)
To type in up to six SIXBIT characters, left justified, delimited by any printing character	$"ABCDEFGA	(A is delimiter)
To type in one SIXBIT character, right justified	$"Q$	($ must be ALTMODE)

A.9 SYMBOLS

The following are DDT symbols:

To permit reference to local symbols within a program titled name	name$:	MAIN.$:
To insert or redefine a symbol in the symbol table and give it the value n	n<symbol:	14<TABL3:
To insert or redefine a symbol in the symbol table, and give it a value equal to the location pointer (.)	symbol:	SYM:
To delete a symbol from the symbol table	symbol$$K	LPCT$$K
To kill a symbol for typeouts (but still permit it to be used for typing in)	symbol$K	TBITS$K
To perform $K on the last symbol typed out and then to retype the last quantity	$D	
To declare a symbol whose value is to be defined later	symbol#	JRST AJAX#
To type out a list of all undefined symbols (which were created by #)	?	

A.10 SPECIAL DDT SYMBOLS

The following are special DDT symbols:

To represent the address of the location pointer	. (point)	
To represent the last quantity typed	$Q	

	Type	Sample Output(s)
To represent the indirect address bit	@	
To represent the address of the search mask	$M	
To represent the address of the saved flags, etc., (see Appendix D)	$I	
To represent the pointers associated with the nth breakpoint	$nB	

A.11 ARITHMETIC OPERATORS

The following arithmetic operators are permitted in forming expressions:

Two's complement addition	+
Two's complement subtraction	-
Integer multiplication	*
Integer division (remainder discarded)	' (apostrophe)

A.12 FIELD DELIMITERS IN SYMBOLIC TYPE-INS

The following are field delimiters:

To delimit op-code name	one or more spaces	JRST SUBRTE
To delimit accumulator field	, (comma)	
To delimit two halfwords	left, right	-6,,BEGIN-1
To delimit index register	( )	
To indicate indirect addressing	@	

A.13 BREAKPOINTS

The following are used for breakpoints:

To set a specific breakpoint n(1<n<8)	adr$nB	CAR$8B
To set the next unused breakpoint	adr$B	303$B
To set a breakpoint with automatic proceed	adr$$nB adr$$B	CAR$$8B 303$$B
To set a breakpoint which will automatically open and examine a specified address, x	x,,adr$nB x,,adr$B x,,adr$$nB x,,adr$$B	AC3,,Z+6$5B AC4,,ABLE$B AC3,,Z+6$$5B AC4,,ABLE$$B

	Type	Sample Output(s)
To remove a specific breakpoint	0$nB	Ø$8B
To remove all breakpoints	$B	$B
To check the status of breakpoint n	$nB/	
To proceed from a breakpoint	$P	$P
To set the proceed count and proceed	n$P	25$P
To proceed from a breakpoint and thereafter proceed automatically	$$P n$$P	$$P 25$$P

A.14 CONDITIONAL BREAKPOINTS

The following are used for conditional breakpoints:

To insert a conditional instruction (INST), or call a conditional routine, when breakpoint n is reached	$nB+1/ $2B+1/ 0	INST CAIE 3,1ØØ

If the conditional instruction does not cause a skip, the proceed counter is decremented and checked. If the proceed count ≤ 0, a break occurs

If the conditional instruction or subroutine causes one skip, a break occurs.

If the conditional instruction or subroutine causes two skips, execution of the program proceeds.

A.15 STARTING THE PROGRAM

The following commands are used to start the program:

To start at the starting address in JOBSA	$G	$G
To start, or continue, at a specified address	adr$G	LOC$G
To execute an instruction	inst$X	JRST 2, @JOBOPC$X returns to program after ↑C and DDT commands

A.16 SEARCHING

The following commands are used for searching:

To set a lower limit (a), an upper limit (b), a word to be searched for (c), and search for that word	a<b>c$W	2ØØ<25Ø>Ø$W

	Type	Sample Output(s)
To set limits and search for a not-word	a<b>c$N	351<731>Ø$N
To set limits and search for an effective address	a<b>c$E	4Ø1<471>LOC+6$E
To examine the mask used in searches (initially contains all ones)	$M/	$M/ -1
To insert another quantity n in the mask	n$M	777ØØØ777777$M

A.17 UNUSED FUNCTIONS

The following are unused:

$U

$Y

A.18 ZEROING MEMORY

The following are used for zeroing memory:

To zero memory, except DDT, locations 20-137, and the symbol table	$$Z
To zero memory locations FIRST through LAST inclusive	FIRST<LAST $$Z

A.19 SPECIAL CHARACTERS

The following special characters are used in DDT typeouts:

Breakpoint stops		
Break caused by conditional break instruction	>	
Break because proceed counter ≤ 0	>>	
Undefined symbol cannot be assembled	U	
Half-word type-outs	left,,right	4Ø1,,4Ø2
Unnormalized floating-point number	#1.234E+27	#1.234E+27
To indicate an integer is decimal. The decimal point is printed	$10R 77=63.	
Illegal command	?	
If all eight breakpoints have been assigned	?	
RUBOUT echo	XXX	

APPENDIX B

STORAGE MAP FOR USER MODE DDT

See Figure B-1. The permanent symbol table, which contains all PDP-10 instructions and monitor UUOs, is an integral part of DDT.

If the user's symbol table is overwritten DDT can still interpret all instructions and UUOs. It will not interpret I/O device mnemonics, internal $ symbols ($M, $I, $1B through $8B, DDT and DDTEND or the following:

JOV
JEN
HALT

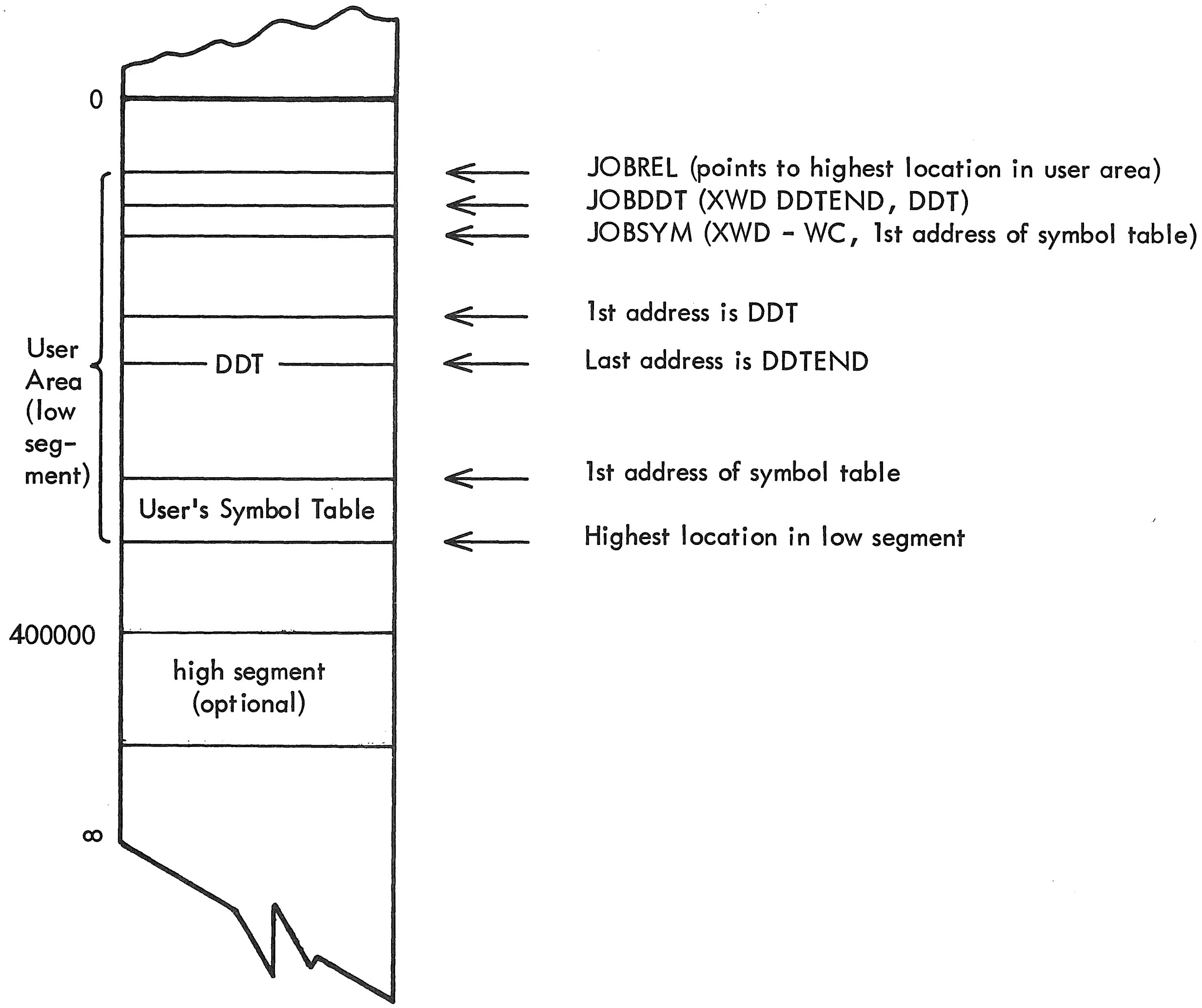
0
JOBREL (points to highest location in user area)
JOBDDT (XWD DDTEND, DDT)
JOBSYM (XWD - WC, 1st address of symbol table)
1st address is DDT
DDT
Last address is DDTEND
User Area (low segment)
1st address of symbol table
User's Symbol Table
Highest location in low segment
400000
high segment (optional)
∞

Figure B-1 Storage Map for User Mode DDT

INDEX

COMMENTS

Title of Publication: PDP-10 UTILITY PROGRAMS MNT-12

The Computer Centre welcomes any suggestions that will assist in improving their publications. Please comment on the usefulness and readability of this manual. Suggest additions and deletions and indicate any specific errors and omissions. Please provide page and section references where relevant.

Name : ______________________________

Position : ______________________________

Address : ______________________________

____

ERRORS

COMMENTS

Please return to the Technical Writer,
Computer Centre,
University of Queensland,
St Lucia.
QUEENSLAND 4067.