Paul McJones

DDT
TIME-SHARING DEBUGGING SYSTEM
REFERENCE MANUAL

L. Peter Deutsch
Butler W. Lampson
University of California, Berkeley

Document No. R-11
Revised October, 1967
Contract SD-185
Office of Secretary of Defense
Advanced Research Projects Agency
Washington 25, D. C.

1.0 General

DDT is the debugging system for the SDS 930 Time-Sharing System. It has facilities for symbolic reference to and typeout of memory cells and central registers. Furthermore, it permits the use of literals in the same manner as in the assembler. It can also insert breakpoints into programs, perform a trace, and search programs for specified words and specified effective addresses. There is a command to facilitate program patching. Finally, DDT can load both absolute and relocatable files in the format produced by the assembler.

The system has a language for communication between DDT and its users. The basic components of this language are symbols, constants, and commands.

1.1 Symbols

A symbol is any string of letters, digits, and dots (.) containing at least one letter. (However, a digit string followed by B or D and possibly another digit is interpreted as an octal or decimal number respectively). In symbols of more than six characters, only the first six are significant: thus, ALPHABET is equivalent to ALPHAB. All opcodes recognized by the assembler are built-in symbols, except for some I/O instructions. Dot (.) is a built-in symbol with a special meaning explained in a later section. There are also some constructs like ;A (the A-register), ;F (the first cell beyond the end of the program), and ;M (the mask for memory searches) which behave like symbols under some circumstances. Their use will be detailed later.

Every symbol may have a value. This value is a 24-bit integer; for most symbols it will be either an address in memory or the octal encoding of an operation code. Examples:

ABC
AB124
12XYZ

The following are not symbols:

135B
AB*CD

Symbols may be introduced to DDT in two basically different ways:

(A) They may be written out by the assembler and read in from the binary program file by DDT.

(B) They may be typed in and assigned values during debugging.

It is possible for a symbol to be undefined. This may occur if a program is loaded which references an external symbol not defined in a previously loaded program. It may also occur if an undefined symbol is typed in an expression. In general, undefined symbols are legal input to DDT except when their values would be required immediately for the execution of a command. Thus, for example, the ;G (GO TO) command could not have an undefined symbol as its argument.

Undefined symbols may become defined in several ways. They may be defined as external in the assembler (i.e., with EXT, ENTRY, or $) and read by DDT as part of a binary program. Alternatively, they may be defined by one of the symbol definition commands available in DDT. When the definition occurs, the value of the symbol will be substituted in all the expressions in which the symbol has appeared.

If DDT types [U] after typing out the contents of a cell, it means that the cell contains an undefined symbol. The cell is closed at once so that its contents cannot be erroneously changed.

The only restriction on this facility is that, as for ARPAS, the undefined symbol must be the only thing in the address field of the word in which it appears. Incorrect uses of undefined symbols will be detected by DDT and will result in the error comment (U).

DDT keeps track of references to undefined symbols by building a pointer chain through the address fields of the words referring to the symbol. Thus, suppose that the symbol A is undefined and appears as follows

```
S1    LDA    A
       :
S2    STA    A
       :
S3    MRG    A
```

and nowhere else in the program. After loading, the entry for A in DDT's symbol table will contain a flag indicating that it is undefined and a pointer to 3. The above locations will contain:

```
S1    LDA    S1

S2    STA    S1

S3    MRG    S2
```

When the symbol is defined, DDT goes through the pointer chain and fills in the value. It recognizes the end of the pointer chain by an address which points to the cell in which it appears.

From this description it should be obvious what will happen if the pointer chain is destroyed. A probable consequence is that a search down the pointer chain will not terminate. DDT does such searches whenever it prints an address. If the chain it is searching has more than 256 links, it will print the symbol followed by (U) and continue. Fixing up an undefined symbol pointer chain which has been clobbered is an exercise which we leave to the reader.

1.2 Block Structure

A limited facility called the block structure facility is provided to simplify the referencing of local symbols which are defined in more than one program. Note that DDT's block structure has only a tenuous connection with the block structure of ALGOL. The block structure of a program is organized in the following manner: every IDENT read by DDT is part of a binary program file begins a new block. Any local symbol known to DDT has a block number associated with it; global symbols do not have a block number. Undefined symbols are always treated as global.

The name of a block is the symbol in the label field of the IDENT. If two IDENTs with the same symbol are read, the message (ALREADY LOADED) is printed, and the local symbol table from the former occurrence of the block will be deleted.

Global symbols must be unique within an entire program and are recognized at all times. If a multiple definition is encountered, the latest one takes precedence. Local symbols are recognized according to the following rules:

(1) At any given time one block is called the primary block. All local symbols associated with the primary block will be recognized.

(2) If a symbol is used which is neither global nor in the primary block, the entire symbol table is scanned for it. If it occurs in only one block, the symbol is recognized properly. If it occurs in more than one block, the error message (A) is printed.

(3) A symbol may be explicitly qualified by writing:

SYMA&SYMB

SYMA must be the name of a block. SYMB is then referenced as though the block whose name is SYMA were primary.

(4) When a cell is opened (see Section 2.1), the block to which the symbolic part of its location belongs becomes primary. Thus, NN&XYZ/ causes block NN to become primary; if ABC is a unique local symbol in block PQ, then ABC/ causes PQ to become primary.

1.3 Literals

Literals have the same format and meaning in DDT as in the assembler, i.e., the two characters' =' signal the beginning of a literal, which is terminated by any of the characters which ordinarily terminate an expression. In contrast to the assembler, the expression in a DDT literal must be defined.

The literal is looked up in the literal table. If it is found, the address which has been assigned to it is the value of the symbol. If it does not appear in the literal table, it is stored at the address which is the current value of ;F, and this address is taken as the value of the literal. ;F is increased by 1. For example, if the literal -1 does not already exist in the literal table and ;F is 1000B, then LDA =-1 causes -1 to be stored at 1000B, and is equivalent to LDA 1000B; the new value of ;F is 1001B. Exception: In patch mode (see Section 2.8), literals are saved and not stored until the patch is completed since otherwise they would interfere with the patch.

When DDT types out a symbol whose value is an address in the literal table, it will type out in the same format in which it would be input; that is, as = followed by the numeric value of the literal.

1.4 Constants

A constant is any string of digits, possibly followed by a B or D, in turn possibly followed by another digit. The number represented by the string is evaluated, truncated to 24 bits and then used just like the value of a symbol. The radix for numbers is normally 8 (octal), but may be changed arbitrarily by the commands described in Section 2.4 below. If a number is terminated by B or D, it is interpreted as octal or decimal respectively regardless of the current radix. A digit following the B or D is interpreted as a power of 8 or 10 respectively by which the number is to be multiplied. Thus 1750B=175B1=1000D=1D3. Constants are always printed by DDT in the current radix.

It is possible to enter numeric op codes by typing the number followed by an @ sign. Thus 100@=14400000B if the current radix is decimal (100D=144B).

1.5 Commands

A command is an order typed to DDT which instructs it to something. The commands are listed and their functions explained in Section 2 below.

1.6 Expressions

An expression is a string of numbers or symbols connected by any of a large number of operators. These operators have the following significance:

+	addition
-	subtraction
;*	(integer) multiplication
;/	(integer) division
;&	(AND) as in ARPAS
;<	(LSS) as in ARPAS
;=	(EQL) as in ARPAS
;>	(GTR) as in ARPAS
;%	(OR) as in ARPAS
;+	x;+y means x;*3+y, or (R)x+y in ARPAS
;-	x;-y means x;*3-y, or (R)x-y in ARPAS
;:	remainder on (integer) division
;$	(EOR)

Expressions are evaluated strictly left to right: all operators have the same precedence. Parentheses are not allowed. The first symbol or number may be preceded by a minus sign. Blank acts like plus, except that the following operand is truncated to 14 bits before being added to the accumulated value of the expression. The value of an expression is a 24-bit integer. An expression may be a single symbol or constant.

```
    Examples:        LDA     has the value      7600000
                     LDA 10  has the value      7600010 if the
                                                 radix is octal
                     LDA 10D has the value      7600012
    If SYM is a symbol with the value 1212, then
                     SYM     has the value      1212
                     SYM 10  has the value      1222
                     LDA SYM has the value      07601212
```

If this last expression were put into a cell and later executed by the program the effect would be to load the contents of SYM, register 1212, into the A register.

When DDT types out expressions, two mode switches control the format of the output. Commands for setting these modes are described in Section 2.4 below. The word printout mode determines whether quantities will be printed as constants or as symbolic expressions. In the latter case, the opcode (if any) and the address will be put into symbolic form. If the first nine bits of the value are 0 or 1, no opcode will be printed; in the latter case a negative integer will be printed. If the opcode is not recognizable as a symbol, it will be typed as a number followed by ~~an @ sign~~.
"35"

The address printout mode controls the format in which addresses are typed. DDT types addresses when asked to open the previous or the next cell, when it reports the results of word and address searches, and on breakpoints. In relative mode, addresses are typed in symbolic form, i.e., as the largest defined symbol smaller than the address plus a constant if necessary. If the constant is bigger than 200 octal, or if the value of the symbol is less than some minimum value (settable by the user, but normally the lowest location of the program) the entire address is typed as a constant. In absolute mode, addresses are always typed as constant.

1.7 The Open Cell

One other major ingredient of the DDT language is the open cell. Certain commands cause a cell to be "opened." This means that its contents are typed out (except in enter mode, for which see the \ command), followed by a tab. If the user types an expression followed by a carriage return, it will be inserted into the cell in place of the current contents, and the cell will then be closed. The current location is given by the symbol "." (dot) which always has as its value the address of the last cell opened, whether or not it is still open.

Note:

(1) Comma and star (for indirect addressing) may be used in expressions as they are used in the assembler; e.g. LDA* 0,2 has the value 27640000.

(2) DDT will respond to any illegal input with the character ? followed by a tab (if a cell is open) or carriage return (otherwise), after which it will behave as if nothing had been typed since the last tab or carriage return. The command ? also erases everything typed since the last tab or carriage return.

1.8 Memory Allocation and DDT

DDT may cause the time-sharing system to assign memory for use either by DDT itself or by the user's program. DDT's memory is used to hold the symbol table, which starts in page 0 and grows upward in memory. The symbol table contracts at the end of each load of a binary file and when symbols are killed; this contraction may cause memory to be released.

DDT acquires program memory when it is required for loading a binary file or when a ;U (execute) command is given and the value of ;F is such that a new block is needed to hold the instruction to be executed. For executing an instruction, DDT requires location ;F, ;F+1 and ;F+2. Memory is never grabbed for examination of a register; however, entering information with \ can cause memory to be assigned. Attempts to open locations not assigned will cause DDT to type ?. This means that upon initial entry to DDT no registers are available for examination. The easiest way to obtain memory is to simply start typing in a program using the \ command.

If an attempt to acquire or reference memory leads to a trap, DDT types (M) and abandons whatever it is doing. This can happen if the machine size is exceeded, or if an attempt is made to change read-only memory.

2.0 DDT Commands

In the following description of DDT commands, <S> will be used to denote an arbitrary symbol. <E> or <W> will be used to denote an arbitrary expression which may be typed by the user: <E> will be used when the value of this expression is truncated to 14 bits before it is used by DDT, while <W> will denote a full 24-bit expression. <A> will be used to denote an optional 14-bit expression. If none is typed, the last expression printed out will usually be used; deviations from this rule will be described under the individual commands. <F> will denote a file name followed by a dot: DDT will type a tab whenever it expects a file name.

2.1 Cell Opening Commands

<A> / This opens the cell addressed by the value of <A>. DDT will give a tab, type an expression whose value is equal to the contents of the register, give another tab and await further commands. The precise form of the expression typed is dependent on the setting of the word and address printout modes. If the user types in an expression, DDT will insert its value into the cell. Typing another command closes the cell, unless it is a type value or symbol definition command. If another / is given as the next command with no preceding expression, the contents of the cell addressed by the expression typed by DDT are typed out. A further / repeats this process. Note, however, that the original cell opened remains the open cell; any changes made will go into that cell.

carriage return This command does not necessarily have any effect. If the specified conditions are present, however, any of the following actions may occur:

(1) If there is an open cell, the cell is closed.
(2) If DDT is in enter mode, it leaves it.
(3) If DDT is in patch mode, the patch is terminated (for a fuller description of this effect, see the patch command in Section 2.8).

<A>] This command has the same effect as /, except that the contents of the cell opened are always typed in symbolic form.

<A> [This command has the same effect as /, except that the contents of the cell opened are typed in constant form.

<A> $ This command has the same effect as/, except that the contents of the cell opened are typed as a signed integer.

<E>@ (open (print formatted)

<E> " This command acts like /, except that the cell constants are typed in ASCII. Unprintable characters, as in QED, are preceded by &, e.g. 141 (control-A) prints out as &A.

<E>;' The contents of locations <E> and <E>+1 are treated as an SPS string pointer, and the string is printed. Cell <E> is opened.

<E>%# fl. pt. no.

line feed This command opens the cell whose address is the current location plus one, i.e. the cell after the one just opened. The output of DDT on this command is carriage return, location (format controlled by the address printout mode), /, tab, value of contents, tab.

; (=space) This is equivalent to line feed except that nothing is printed. Its main use is in entering programs or data, e.g.

```
        1000    1; 2; 3         (carriage return)
```

is equivalent to

```
        1000    1       (carriage return)
        1001    2       (carriage return)
        1002    3       (carriage return)
```

↑ This command opens the cell whose address is the current location minus one, i.e. the previous cell. The output is the same as for the line feed command.

Example:

```
        ABC/     LDA   ALPHA                   (line feed)
        ABC+1/   STA   BETA     STA GAMMA      (line feed)
        ABC+2/   LDB   DELTA    ↑
        ABC+1/   STA   GAMMA
```

(This command opens the cell whose address is the last 14 bits of the value of the last expression typed. The output is the same as for line feed.

\ This command is the same as /, except that the contents of the cell are not typed. DDT goes into enter mode, in which the contents of cells opened by line feed, ↑, or (are not typed. Most other commands cause DDT to go out of enter mode. In particular, carriage return has this effect. When a cell has been opened with \ , DDT thinks that it has typed out the contents. The type value commands will, therefore, work on the contents of the cell.

The type register in special mode character [,], $ (type as a ~~negative~~ signed integer), " (type in ASCII) are also preserved by line feed, up arrow and (.

;\ This command suppresses typeout of cell addresses during line feed, up arrow and (chains. Carriage return cancels the command.

2.2 Type Value Commands

= This command types the value of the last expression typed (;Q) in constant form. It may appear in the form <W> =, in which case the value of the <W> is typed. Otherwise, the expression referred to is the one most recently typed, either by DDT or by the user.

# This command types the value of ;Q as a signed integer.

← This command types the value of ;Q in symbolic form.

' This command types the value of ;Q typed as a word of text (see " command on previous page).

@ This command types the address part of ;Q in symbolic form. If, for instance, the program has executed BRM X, then X\@ will cause DDT to print the address of the BRM.

Example:

LDA=	7600000
LDA 10=	7600010
LDA ←	LDA
7600000←	LDA
-1=	77777777
-1#	-1
77777777#	-1
10221043'	ABC

;← This command types ;Q as a character address, e.g. if the value of the symbol X is 1000, then 3002;← yields X;+2.

;' This command types the string pointed to by the contents of the current location and the following cell, considered as an SPS string pointer.

<E>, <E>;' This command types the string pointed to by the pair of expressions considered as an SPS string pointer.

2.3 Symbol Definition Commands

These commands all define the symbol as global.

<S> : This command defines the value of the symbol <S> to be the current location.

<S> @ This command defines the value of <S> to be the address of the last expression typed by DDT or the user.

<E> ⟨<S>⟩ This defines <S> to have the value of <E>.

2.4 Mode Changing Commands

" This command is followed by a string of arbitrary characters terminated by D^c (control D). If a cell is open, the string will be inserted into successive locations packed 3 characters per word; otherwise, characters beyond the third will be thrown away. For example, if no register is open, "ABCDED^c= yields 10221043.

;D (DECIMAL) This command changes the current radix (see Section 1.4).

;O (OCTAL) This changes the current radix to octal.

<E> ;R (RADIX) sets the current radix to the value of the expression, which must be ≥ 2.

;[(CONSTANT) This command changes the word printout mode to constant, i.e. makes / equivalent to [.

;] (SYMBOLIC) This command changes the word printout mode to symbolic, i.e. makes / equivalent to].

;" (ASCII) This makes / equivalent to ".

;$ (SIGNED INTEGER) This makes / equivalent to $.

;R (RELATIVE) This command changes the address printout mode to relative (symbolic). This mode determines the format for the output of addresses, both in symbolic expressions and when generated by line feed and ↑.

;V (ABSOLUTE) This command changes the address printout mode to absolute.

;3 (3 CHARS/WORD) This sets the " and ' commands to operate on 8-bit characters packed 3 per word.

;4 (4 CHARS/WORD) This sets the " and ' commands to operate on 6-bit characters packed 4 per word.

2.5 Breakpoint Commands

<E>! (BREAKPOINT) <E>! sets a breakpoint at the address given by the value of the expression E. (If the maximum of 4 breakpoints has been reached, DDT will type FULL.) The effect is that if the program executes the instruction at this address control returns to DDT, which will print the address and the contents of the A, B and X registers and await further commands (see below). The break occurs before execution of the instruction in the breakpoint location. ;L is set to the location at which the break occurred.

! (CLEAR ALL BREAKPOINTS). ! alone causes all breakpoints to be cleared.

<E>;! (CLEAR BREAKPOINT) The breakpoint at <E>, if any, is removed. If no such breakpoint is present, ? is typed and no other action is taken.

;! (LIST BREAKPOINTS) The breakpoints are listed.

<A>;P (PROCEED) This command restarts the program after a break. The program executes the instruction at the break and goes on from there. No breakpoint is removed unless this is specifically done

by ! or ;! so that, if the program arrives at this location again, another break will occur. If <E>;P is given, another break will not occur until some breakpoint has been reached that many times.

<A>;N (NEXT) This command executes the instruction at ;L and breaks. This program provides a trace facility in that repeated executions of ;N will provide a running print out of the contents of the significant internal registers, instruction by instruction. The function is essentially the same as that of the step switch on the console. <E>;N will cause <E> instructions to be executed before the next break occurs.

The ;N command follows the flow of control in the user's program. In particular, it will normally trace the execution of users' POPs (see ;O below). The execution of SYSPOPs, however, is not traced. In other words, a SYSPOP such as FAD (floating add) is regarded as one instruction by ;N. Cells ;F, ;F+1, and ;F+2 are used by ;N and ;P.

<E>;S (STEP). This is equivalent to <E> repetitions of ;N. Note that this is not the same as <E>;N.

<E>;V (ADVANCE). This is equivalent to <E> repetitions of ;P, and is not the same as <E>;P.

<N>;O (POP TRACE MODE). If <N>>0, programmed operators (POPs) together with their associated subroutines will be treated like machine instructions for the ;N and ;S commands, i.e. the break will not occur until control returns to the location following the POP. Since DDT determines when it should break by counting POPs, BRMs, SBRMs, BRRs and SBRRs, it can be fooled by POPs which do sufficiently peculiar things. If <N>≤0, POP subroutines will be traced, i.e. the first break after the POP will be at the first instruction of the subroutine.

<N>;U (SUBROUTINE TRACE MODE). If <N>=1, BRMs or SBRMs together with the subroutine called will be treated as single instructions by ;N. The same algorithm is used as in ;O to determine when to break. If <N>=0, subroutines will be traced explicitly.

Attempts to proceed through certain instructions having to do with forks will produce erroneous results, and breakpoints encountered when the program is running in a fork will not do the right thing. Attempts to proceed through unreasonable instructions will cause the error comment

$ > > .

Also, when control returns to DDT from a breakpoint or rubout, the interrupt mask for the program is cleared.

2.6 Input/Output Commands

<A>;Y<F> DDT expects to find a binary program on the file <F>. If the program is absolute, it is read in. If it is relocatable, it is read in and relocated at the location specified by <A>. If the expression is omitted, relocatable loading commences at location 240B and continues by beginning each program in the first available location after the preceding one, i.e. at the value of ;F at the time. After reading is complete, the first location not used by the program is typed out. Any local symbols on the binary file are ignored.

<A>;T<F> This command is identical to ;Y except that it also reads local symbols from the file and adds them to DDT's symbol table. Any symbols on the file will be recognized by DDT thereafter.

The following two points should be noted in connection with ;Y and ;T commands.

1) The use of an expression before ;T or ;Y when the file is absolute (i.e. SAVE file or self-loading paper tape) is an error.
2) The block read in becomes the primary block.

;W<F> Causes all global symbols to be written on the specified file, in a format which can be read back in with ;T.

;C<F> Causes all symbols to be written on the specified file.

2.7 Search Commands

<W>;W (WORD SEARCH) <W>;W searches memory between the limits ;1 and ;2 for cells whose contents match <W> when both are masked by the value of ;M. The locations and contents of all such cells are typed out. $<W_1><<W_2>$;W will perform this search, and in addition performs the following replacement: if Q is the address of a cell such that $(Q)\wedge;M=W_2$, then (Q) will be replaced by $(Q)\wedge\overline{;M}+W_1$. (Note that this is not quite the same as masked substitution.) Both old and new contents of the cell will be typed out.

<W>;# (NOT-WORD SEARCH). This is the same as ;W, except that all cells which do not match <W> will be printed. This is useful, for example, in finding and printing all non-zero cells in a given part of memory.

<E>;E (EFFECTIVE ADDRESS SEARCH). <E>;E searches memory between the limits ;1 and ;2 for effective addresses equal to <E>. Indexing, if specified, is done with the value of ;X. Indirect address chains are followed to a depth of 64. The addresses and contents of all words found are typed out. When ;W or ;E is complete, . is left pointing to the last cell where the expression was found.

2.8 The Patch Command

<A>) <A>) causes a patch to be inserted before the instruction at ".". If an expression is given, the expression is used instead of the current contents of ".". DDT inserts in this location a branch to the current value of ;F. When the patch is done, ;F is updated. It then gives a carriage return and a) and waits for the user to type in the patch. Legal input consists of a series of expressions whose values are inserted in successive locations in memory. Each of these expressions should be terminated by line feed or ;▲, exactly as though the program were being typed in with the \ command instead of as a patch. The ↑ command may be given in place of the line feed and has its usual meaning, except that the contents of the previous location are not typed. Two other commands are legal in patch mode. They are:

(1) Colon, which may be used to define a symbol with value equal to the current location.

(2) Carriage return, which terminates the patch. When the patch is terminated, DDT inserts in the next available location the original contents of the location at which the patch was inserted. It then inserts in the following two locations branch instructions to the first and second locations following the patch. This means that if the patch command is a skip instruction, the program will continue to operate correctly. Any other command given in patch mode may cause unpredictable errors.

<A>;I Is identical to the) command except that it puts the instruction being patched before the new code inserted by the programmer instead of after.

2.9 Miscellaneous Commands

;? and ? This commands erase everything typed since the last tab or carriage return. It is always legal.

<E>;G (GO TO) <E>;G restores the A, B and X registers which were saved when DDT was entered (unless they have been modified) and transfers to the location specified by the value of the expression.

;K (KILL) This command resets DDT's symbol table to its initial state. DDT will type back--OK and wait for a confirming dot. Any other character will abort the command.

<S>;K (KILL). Removes only the symbol <S> from the table.

<S>&;K (KILL). Removes all symbols local to the block named <S> from the table, as well as the block name itself.

<E>,<E>;L Sets ;1 and ;2 (the lower and upper bounds for searches) to the values of the first and second expressions respectively.

;U (UNDEFINED). This command causes all undefined symbols to be listed.

<E>;U (EXECUTE). This causes the value of the expression to be executed as an instruction. If it is a branch, control goes to the location branched to. In all other cases control remains with DDT. A single carriage return is typed before execution of the instruction. If the instruction does not branch and does not skip, or returns to the following location, a $ and another carriage return are typed after its execution. If the instruction does skip, two dollar signs ($$) are typed followed by a carriage return.

;Z (ZERO) <E>,<E>;Z sets to zero all locations between the value of the first expression and that of the second. <E><<E>,<E>;Z sets to the value of the first expression all locations between the values of the second and third. ;Z alone releases all memory accessible to the user's program. DDT will type back --OK and wait for a confirming dot. Any other characters will abort the command. If this memory is returned, due to later access by DDT or a program, it will be cleared to zero.

%& (LIST BLOCKS). The names of all blocks are printed.

%F (FINISHED). Control returns to the executive.

%R (PRINT MAP). The current program map (pseudo-relabeling) is printed.

<E>,<E>;R(SET MAP). The program map is set as indicated. This is equivalent to putting the expressions in A and B respectively and executing BRS 44.

2.10 Special Symbols

The value of "." is the current location, i.e. the address of the last register opened.

The following constructs refer to various special registers of the machine. They act like symbols in that in most contexts they are synonymous with their values. Their value is the contents of these registers as saved by DDT: ;X= will print the saved contents of the X register. To change the contents of a register, a command of the form <E>;A is used. This command sets the A register to the value of the expression. Whenever DDT executes any command involving execution of instructions in the user's program, it restores the values of all machine registers. If any of these values have been changed by the user, it is the changed value which will be restored.

;A The value of this symbol is the contents of the A register.

;B The value of this symbol is the contents of the B register.

;X The value of this symbol is the contents of the X register.

;L The value of this symbol is the contents of the program counter. The only reason for changing ;L is to set the location from which ;N will begin execution.

The values of the following special symbols are used by DDT in certain commands or are available to the programmer for his general enlightenment. These values may be changed in the same way that the values of the symbols for the central registers of the machine may be changed.

;M The value of this symbol is the mask for word searches.

;Ø The value of this symbol is the smallest address which DDT will ever attempt to print in symbolic form.

;1 The value of this symbol is the lower bound for word and effective address searches. It may also be set by the ;L command.

;2 The value of this symbol is the upper bound for word and effective address searches. It may also be set by using ;L.

;Q This symbol has a value equal to the value of the last expression typed by DDT or the user. It is useful, for instance, if the programmer wishes to add one to the contents of the open register; he need only type ;Q + 1.

;F The value of this symbol is the address of the lowest location in core not used by the program. New literals and patches are inserted starting at this address. Note: like all other special symbols, ;F may be changed by the command <E>;F. It is also updated as necessary by patches and literal definitions.

2.11 Panics

DDT recognizes four kinds of panic conditions:

(1) Illegal instruction panics from the user's program.
(2) Memory allocation exceeded panics from the user's program.
(3) Panics generated by pushing the rubout button.
(4) Panics generated by the execution of BRS 10 in the user's program.

For the first two of these conditions DDT prints out a message, the location of the instruction at which the panic occurred, and the contents of this location. The messages are as follows:

(1)	Illegal instruction panic	I > >
(2)	Memory allocation exceeded	M > >

(3) The other two types of panics cause DDT to type bell and carriage return. ;L and . will both be equal to the location at which the panic occurred.

If a memory allocation exceeded panic is caused by a transfer to an illegal location, the contents of the location causing the panic is not available and DDT, therefore, types a ?.

Two other panic conditions are possible in DDT.

(1) If the rubout button is pushed twice with no intervening typing by the user, control returns to the executive.
(2) If the rubout button is pushed while DDT is executing a command, execution and typeout are terminated and DDT types carriage return and bell and then awaits further commands.

2.12 Multiple Program Debugging

It is occasionally desirable to hold several programs with different maps and symbol tables in DDT simultaneously. This situation could be approximated using the DUMP and RECOVER commands in the time-sharing executive, but several commands are provided in DDT itself to facilitate the process.

<W_1>,<W_2>;R (SET MAP). The pseudo-relabeling for the program is set according to the value of <W_1> and <W_2>. This command is essentially equivalent to executing BRS 44 with <W_1> in A and <W_2> in B.

%E (ERASE). DDT types --OK and waits for a confirming dot. Any other character will abort the command. DDT then resets itself to its initial state, i.e. the symbol table, program map, breakpoints and modes are all reset. The program memory, however, is not released.

%D (DUMP). This command also requires a confirming dot. The entire state of DDT is saved away and a number typed out which will allow this state to be retrieved by the %R command (see below). DDT then resets itself as described under %E above.

<E>%R (RECOVER). This command requires a confirming dot. If the present state of DDT has ever been dumped (i.e. was produced by %R), it is dumped again. Then the state is restored exactly as it was when the %D was given, whose number was the value of <E>. Using an illegal number for %R can lead to chaos.