DDT

INTERACTIVE MACHINE LANGUAGE DEBUGGING SYSTEM

REFERENCE MANUAL

Mark L. Greenberg

University of California, Berkeley

Document No. R-39
June 15, 1969
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. Futhermore, 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. In addition, their is a powerful conditional breaking facility that will allow the user to specify the exact condition under which he wants the program to break. 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) which under most circumstances act exactly like symbols. See the section on special symbols.

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
AB12B
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 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 would give an error if its argument is an undefined symbol.

Undefined symbols may become defined in several ways. They may be defined as external in the assembler (i.e., with EXT, or $) and read by DDT as part of a binary program. Alternatively, they may be defined by one of the symbol defintion 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 symbolically, 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 NARP, 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 pointer chains through the address fields of the words referring to the symbol. For each umdefined symbol, DDT will construct a different pointer chain for each 2K page in the users map which contains a reference to the symbol. The pointers to the beginnings of these chains are maintained in a 2K page acquired by DDT and not accessable by the user. A chain pointer is stored in the right eleven bits of the address field. Bit 10 marks the end of a chain and bit 11 indicates whether the symbol should be patched up on 18 bits or 24 bits when the symbol is defined.

From this description is 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 will often do 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. A chain may be destroyed by clobbering a word in

the chain or by removing a page containing undefined references from the user map. A word of warning, DDT's loader may become confused or go into a loop if a binary program is loaded which attempts to define an undefined symbol whose reference chains are not all in the map. 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 separately loaded 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 as part of a binary 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 tables for the two IDENTs will be merged. Conflicting symbol definitions will be overwritten.

If local symbols are defined in a binary program file, with no preceeding IDENT, then DDT will assign an IDENT name such as 03X. 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 entrire 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 primay.

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 ordinarly 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 the "special symbol" ;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 typing LDA =-1 causes -1 to be stored at 1000B, and is equivalent to LDA 1000B; the new value of ;F is 10001B. 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 it out in the same format in which it would be input; that is, as = followed by the numeric value of the literal. It should be noted that addresses specified as literals in a binary program file loaded by DDT will be printed as literals, however no entry into DDT's literal table is made for these addresses.

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 the 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 a 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.

1.5 Commands

A command is an order typed to DDT which instructs it to do 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
;&	logical AND
;<	less than
;=	equal to
;>	greater than
;%	logical OR
;+	x;+y means x;*3+y
;-	x;-y means x;*3-y
;:	remainder on (integer) division
;$	logical exclusive OR

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 all subsequent operands are truncated to 14 bits before being operated on the accumulated value of the expression. This allows one to compute a value which is loaded only into the address field of an instruction. The value of an expression is a 24 bit integer. An expression may be a single symbol or constant. The value of an expression after a relational operation is either 1 or 0 if the relation is true or false respectively.

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 iwth 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 all zeros or all ones, 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 'B5'.

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 or equal to 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 constants.

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 indexing and indirect address) 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 memeory 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 ;P, ;P+1,;P+2. Memory is never grabbed for examination of a register; however, entering information in cells which are in pages not in the map will cause a block of memory to be assigned for that page. If a cell is opened which is not assigned then DDT will type a ? and a tab, but the cell will remain open. Information may be filled into such a cell.

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 18 bits before it is used by DDT, while <W> will denote a full 24 bit expression. <A> will be used to denote an optional 18 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 futher 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. A floating point number may be inseted in the open cell and the following cell by typing a '%' followed by a floating point number and then closing the 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>" This command acts like /, except that the cell contents are typed an ASCII. Unprintable characters, as in QED, are preceded by &, e.g. 141 (control-A) prints out as &A.

<E>@ This command has the same effect as /, except that the contents of the cell opened are printed in formatted form. The format which is used is determined by the values of the two special symbols %M and %N. %M is a mask of bits to be included in fields. A bit turned on in %N indicates the right-most bit of a field. The formatting commands will act as though the right-most bit of %N is turned on whether it is or not. For example, if cell 400 contains 54356321B and %N=10101B and %M=77770777B, then we get

400@ 5435 3 21

Successive fields in the open cell and following cells may be changed by typing expressions for the new field values separated by ;, .

<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>%# The contents of locations <E> and <E>+1 are treated as a floating point number which is printed. Cell <E> is opened.

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 the 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 characters [,], $, ", are also preserved by line feed, up arrow and (.

;⁄ This command supresses 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 <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 value of ;Q in formatted form. (see the <A>@ command).

Example:

```
LDA=            7600000

LDA 10=         7600010

LDA<            LDA

7600000<        LDA

-1=             77777777

-1#             -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. Also, the current location is set to the word address of the character.

;' This command types the string pointed to by the contents of the current location and the following cell, consided 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.

%# This command types the contents of the current location and the following cell considered as a floating point number.

<E>,<E>%# This command types the pair of expressions considered as a floating point number.

2.3 Symbol Definition and Killing Commands

These commands all define the symbol as global.

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

<W> < <S>: This defines <S> to have the value of <W>.

<W> < <S>;O This defines <S> as an opcode with value <W>.

;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) This command removes only the symbol <S> from the symbol table.

<S>&;K (KILL) This command removes all symbols local to the block named <S> from the symbol table, as well as removing the block name itself. This command will type back --OK and expect a confirming dot.

%K This command will remove all undefined symbols from DDT's symbol table. This implies that all references to undefined symbols will be lost. DDT will type --OK and expect a confirming dot.

2.4 Mode Changing Commands

" This command is followed by a string of arbitrary characters terminated by 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 and the result treated as a constant. For example, if no register is open, "ABCDEcontrol-D= yields 10221043.

;D (DECIMAL) This command changes the current radix to decimal (see section 1.4).

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

<E>;R (RADIX) This command sets the current radix to the value of the expression, which must be greater than or equal to 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 / eqivalent to].

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

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

;@ (FORMATTED) This command makes / equivalent to @.

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

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

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

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

2.5 Breakpoint Commands

There are four breakpoints in DDT. The first one is called the special breakpoint. The remaining 3 are called regular breakpoints. If a program attempts to execute the instruction at an address at which a breakpoint is set, control returns to DDT which will print a break message and await further commands. The break occurs before execution of the instruction in the breakpoint location. ;L is set to the location at which the break occured. The break message will normally print the address of the break followed by the contents of any of the central registers which have changed since the last break. If it is the first break after a ;G then all registers will be typed. Furthermore, if a register has not been printed in the last ten break messages, then it will be printed anyway in the form ;A=<expression>, instead of the normal form ;A←<expression>. The contents of the break location will also be typed in the break message if the special symbol %I is set to a non-negative value. The typing of the break address can be suppressed by setting the special symbol %P to a non-negative value. If one instruction is executed by the ;N or ;S commands, then the break message will include the address and new contents of any cell modified as a result of executing that instruction. In addition to

these breakpoints there is the conditional breakpoint facility (see the section on conditional breaking).

<E>! (SET SPECIAL BREAKPOINT) This command sets the special breakpoint to the value <E>. The previous value of the special breakpoint will of course be lost.

! (CLEAR SPECIAL BREAKPOINT) This command clears the special breakpoint.

<E>;! (SET REGULAR BREAKPOINT) This command sets a regular breakpoint to the value <E>. If all the regular breakpoints are already set then DDT will type back FULL?. If a breakpoint already exists with that value then DDT will type a ?.

<E>%! (CLEAR REGULAR BREAKPOINT) This command will clear a regular breakpoint that has the value <E>. If no such breakpoint exists, then DDT will type ?.

%! (CLEAR ALL BREAKPOINTS) This command clears all 4 breakpoints.

;! (list breakpoints) This command lists all breakpoints, the special breakpoint first.

2.6 Program Execution Commands

<A>,<A>;G (GO TO) This command allows transfer of control to the user program. The first argument, if given, specifies the starting address. The second argument, if given, specifies the number of breakpoints the program will pass through before the program halts. The first argument, if missing, is assumed to be the current location. The second argument, if missing, is assumed to be one. If there are any undefined symbols in DDT's symbol table, then DDT will type '--OK' and expect a confirming dot to be typed before it will allow transfer of control to the user program. This is true also of all the other progam execution commands.

<A>,<A>;P (PROCEED) This command also causes transfer of control to the user program, but it is designed to restart a program after a breakpoint. It is identical to the ;G command, with the following exceptions. The ;P command will not break on the first instruction executed if a breakpoint is set at that address, whereas the ;G command will break. The arguments of the ;P command are interpreted in the opposite order of the ;G command. All central registers will be printed

at a break after a ;G command, while only the changed central registers are printed after the ;P command.

<A>,<A>;V (ADVANCE) This command is identical to the ;P command except that a break message is printed at every breakpoint encountered during execution of the program, whereas ;P will print a break message only after the last breakpoint.

<A>,<A>;N (NEXT) This command causes the number of instructions specified by the first argument to be exectuted starting at the address specified by the second argument. If the first argument is omitted then 1 is assumed. If the second argument is omitted then ;L is assumed. A break message is printed at the end of the execution.

<A>,<A>;S (STEP) This command is identical to the ;N command except that a break message is printed after the execution of every instruction.

<E>;U (EXECUTE) This command 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 instrucition 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.

DDT may be put in pop trace mode by setting the special symbol %O to a non-negative value. A negative value will cause DDT to leave pop trace mode. In pop trace mode all programmed operators 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.

DDT can be put is subroutine trace mode by setting the special symbol %U to a non-negative value and removed from this mode by setting %U to a negative value. In subroutine trace mode BRMs and SBRMs together with their associatied subroutines will be treated as single instructions by the ;N and ;S commands.

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

2.6 Input/Ouput 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 ;F. ;F is updated when the file is loaded. After reading is complete, the first location not used by the program is typed out. Any local symbols or opcode definitions on the binary file are ignored.

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

<A>%Y <F> This command is identical to the ;Y command except that it will also read in opcode definitions and put them in DDT's symbol table, but it will still ignore local symbol definitions.

The following two points should be noted in connection with ;Y, %Y, or ;T commands.

1) The use of an expression before ;T, or %Y or ;Y when the file is absolute (i.e. SAVE file) is an error.

2) The block read in becomes the primary block.

Several files can be loaded in a row with greater speed if a semi=colon is used for the file name terminator instead of a dot. Using a semi-colon causes the resorting of the DDT symbol table to be suppressed at the end of loading.

;W <F> Causes all global symbols with their values 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) This command searches memeory 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.

<W1>< <W2>;W will perform the same search, and in addition performs the following replacemnt: if Q is the address of a cell such that (Q) (and) ;M = W2, then the masked part of W1 will replace the masked part of (Q).

<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) This command 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 typed out.

2.8 Patch Commands

;) This command causes a patch to be inserted before the instruction in the open cell. A cell must be open for this command to be legal. 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 ;(space), exactly as though the program were being typed in with the ↗ commadnd 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 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 inseted. It then insets in the following two locations branch instructions to the first and second locations following the patch. This means that if the patched instruction is a skip instruction, the program will continue to operate correctly. Any other command given in patch mode may cause unpredictable errors.

;(This command is identical to ;) command except that it puts the instruction being patched before the new code inserted by the user instead of after.

2.9 Miscellaneous Commands

;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. If this memory is returned, due to later acces by DDT or a program, it will be cleared to zero.

%& (LIST BLOCKS) This command causes all blocks known to DDT to be listed. If printing of symbols in that block has been suppressed then a ;] will be typed following the block name.

<block name>;] (SUPPRESS) This command causes symbols in the given block to be ignored when DDT prints symbolic address.

<block name>;[This command reverses the action caused by the ;] command.

%F This command causes control to be returned to the fork which called DDT.

%R (PRINT MAP) The current program map 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.

%A This command lists all ambiguous symbols in the DDT symbol table and lists the blocks that each ambiguous symbol is in.

2.10 Special Symbols

DDT has built into it a number of special symbols. these symbols can have their values set with the following construct. <E>;A where ;A is a special symbol. if a special symbol is used in any other context then it is treated like any other symbol in DDT. 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 contents of the B register

;X contents of the X register

;L contents of the program counter

;M mask used by word search commands

;0 the value of this symbol +1 is the smallest address which DDT will ever attempt to print in symbolic form.

;1 lower limit for searches using search commands

;2 upper limit for searches

;Q value of the last expression typed by DDT or the user.

;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. It is updated by patches, literal definitions, and load commands.

%N The value of this symbol is the field descriptor for type out and loading of cells in formatted form.

%M The value of this symbol is the mask for type out and loading of cells in formatted form.

%V The value of this symbol is the opcode number which will next be used in automatic opcode definitions using the opcode linking feature of the DDT loader. (see the manual on binary file format for the DDT loader)

The remaining special symbols are used to control modes in DDT. The specified action will occur if the symbol is set to a non-negative value.

%B Turn on conditional breaking

%T Print the instruction at ;L as part of a break messsage.

%P Suppress printing of the value of the program counter as a part of the break message.

%O Suppress tracing of POP subroutines.

%U Suppress tracing of BRM and SBRM subroutines.

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 users'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 panic 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 memory allocation exceed panic is caused by a transfer to an illegal location, the contents of the location causing the panic is not avialabe and DDT, therefore, types a ?.

Two other paic conditions are possible in DDT.

(1) If the rubout button is pushed twice with no intervening typing by the user, control returns to the calling fork.

(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 Conditional Breaking

Conditional breaking a feature which allows the user to run a program and have it break on the exact instruction where a specified condition becomes true. Conditional breaking is implemented by loading an interpreter at ;P when control is transfered to the user program. Therefore, if conditional breaking is used, approximately 200 cells following ;P will be clobbered. Conditional break mode is entered by setting the symbol %B to a non-negative number. A negative value of %B returns DDT to normal break mode. To use conditional breaking, put DDT in conditional break mode and specify a condition to break on using the %B command. Thereafter, any transfer of control to the user program will cause conditional breaking to take effect. A user program will execute about 15 times slower in conditional break mode.

To specify a break condition type %E. DDT will then type carriage return and expect the user to type a logical expression which obeys the following syntax. A conditional expression is terminated with a control-D. Control-A may be used to delete characters typed. When the value of this expression becomes true then the condition is satisfied and DDT will cause the user program to break.

Syntax:

<expr> ← any number of <term> separated by !

<term> ← any number of primary separated by *

<prim> ← <operand1> <relational> <operand2>

<relational> ← =/#/>/</<=/>=

<operand1> ← any number of <operand2> separated by <operator>

<operaor> ← +/-/@/blank

<operand2> ← [$] (<constant>/<symbol>/<special symbol>

<special symbol> ← ;A/;B/;X/;L/;E/;O

A constant may be any legal DDT constant.

A symbol may be any defined symbol optionally preceded by a block name and &.

Semantics:

Expressions are compiled left to right. The stregnth of binding of operators is as flollows:

<operators>,<relationals>,*,!.

! means OR

* means AND

@ means MASK i.e. extract second operand from first operand.

$ means literal as opposed to contents of

blank means address add e.g. b 3 is the contents of b+3

+ means add

- means subtract

;A,;B,;X are the contents of the central registers.

;L is the program counter

;E is the effective address of the current instruction.

;O is the 7-bit opcode field of the current instruction right justified.

Examples:

A 3=;E*;O=$43!B+3=$123

This expression will cause conditional breaking if the contents of A+3 is equal to the effective address of the current instruction at the same time that the opcode of the current instruction is 43 or if the contents of B plus the contents of 3 equals 123.