Price: $1.00

DDT

REFERENCE MANUAL

for

SDS 940 TIME-SHARING COMPUTER SYSTEMS

May 1967

90 11 13A

SDS

SCIENTIFIC DATA SYSTEMS/1649 Seventeenth Street/Santa Monica, California

Printed in U.S.A.

RELATED PUBLICATIONS

Title	Publication Number
SDS 940 Computer Reference Manual	90 06 40
SDS 940 Time-Sharing System Reference Manual	90 11 16
SDS 940 Terminal User's Guide	90 11 18

NOTICE

The specifications of the software system described in this publication are subject to change without notice. The availability or performance of some features ma depend on a specific configuration of equipment such as additional tape units or larger memory. Customers should consult their SDS sales representative for details

CONTENTS

1. INTRODUCTION

The SDS 940 Dynamic Digital Debugging Tool (DDT) gives the user the capability of investigating a program's operation in terms of its original symbols. Because of the high degree of man-machine interaction permitted and the availability of the DDT subsystem, it allows the user to maintain vigilance over his program as it is executing. It also allows him to execute one, part, or all of his program's instructions and to investigate, between executions, the contents of any or all variables or locations.

A "patching" mechanism is provided for modifying programs and then continuing execution with negligible turn-around time. Other features of DDT include word pattern searches and execution halts at specified breakpoints.

Significantly, only a small fraction of the computer's time is actually employed during on-line debugging. The 940 Executive dismisses programs that are awaiting input and moves them to secondary storage. When the user makes his next move, the program resumes operation, usually within a second or two.

In addition to its on-line debugging capabilities, DDT allows the user to write machine-language programs directly. Although DDT does not save symbolics, dump files of programs can be created, so that many sessions of debugging can take place without the symbolics ever being reassembled.

This manual is intended to serve as a tutorial guide for the new DDT user and as a reference source for the experienced user. For clarity, several typographic conventions are used throughout the manual. These are explained below.

1. Underscored copy in an example represents that produced by the system in control of the computer. Unless otherwise indicated, nonunderscored copy in an example is that typed by the user.

2. The notation (RET) signifies a carriage return and (LF) signifies a line feed. The user signals the end of a DDT command by striking the RETURN key, and the system confirms the command with an automatic line feed. If either or both of these functions are initiated by the system, no notation is used.

3. Where the symbol "␣" appears in a DDT command, it represents a "blank" character.

OPERATING PROCEDURES

The standard procedure for gaining access to an SDS 940 time-sharing computer center from a remote Teletype terminal is described in the SDS 940 Terminal User's Guide, Publication No. 90 11 18. The user should also have a general understanding of the TAP macro assembler described in the SDS TAP Reference Manual, Publication No. 90 11 17. The following paragraphs summarize the standard procedures as they apply to DDT users.

LOG IN

To gain access to the computer, the following operating sequence is performed:

1. If the FD-HD (Full Duplex-Half Duplex) switch is present, turn the switch to FD. This is a toggle switch with two locking positions. When the Teletype is not connected to the computer (sometimes called the Local Mode), this switch must be in the HD position. Whenever the Teletype is connected to the computer, this switch must be in the FD position.

2. Press the ORIG (originate) key, which is usually located at the lower right corner of the console directly under the telephone dial. This key is depressed to obtain a dial tone before dialing the computer center.

3. Dial the computer center number. When the computer accepts your call, the ringing will change to a high-pitched tone. There will then appear on the Teletype a request that the user log in:

   PLEASE LOG IN!

4. The user must then type his account number, password, and name in the following format:

   number password;name (RET)

   Only persons who know all three elements (the account number, password, and name) may log in under that particular combination. The following examples all illustrate acceptable practice.

   PLEASE LOG IN! 123PASS;JONES (RET)
   PLEASE LOG IN! 151WORD;BROWN (RET)
   PLEASE LOG IN! 175PW;PSEUDO (RET)

   If the account number, password, and name are not recognized by the computer, it will print INVALID USER. The log-in procedure must then be repeated. If the user does not type his account number, password, and name within a minute and a half, a message is transmitted instructing him to call the computer center for assistance. The computer will then disconnect the user, and the dial and log-in procedure will have to be repeated.

5. If the account number, password, and name are accepted by the computer, it will print READY, the date, and the time on the next line, and a dash at the beginning of the following line:

   READY date, time
   -

   The dash indicates that the 940 Executive is ready to accept a command.[†]

[†] In some 940 time-sharing systems the commercial "at" sign, @, is used to indicate that the 940 Executive is ready to accept a command.

6. In response to the dash, the user types

    DDT

    When ready to accept commands, DDT responds with a carriage return and line feed, but does not print a confirming symbol.

ESCAPE

The ESCAPE (ESC) key[t] may be used at almost any time. It causes the DDT subsystem to abort the current operation and ask for a new command. Striking the (ESC) key before terminating a command with (RET) aborts the command.

[t] In some 940 time-sharing system configurations the RUBOUT or ALT MODE key is used instead of the ESCAPE key. Where (ESC) appears in this manual, RUBOUT or ALT MODE may be substituted.

EXIT AND CONTINUE

Striking the (RET) key twice in succession causes computer control to return to the Executive. The user may reenter DDT with program and execution status intact by typing

-CONTINUE (RET)

The system then prints a "DDT", to confirm the action, and returns control to DDT.

LOG OUT

When the user wishes to be disconnected from the computer, he types two consecutive escapes (to return control to the Executive) and then types

LOGOUT

The computer will respond by printing the amount of computer time and hook-up (line) time charged to the user's account since the previous log-in procedure was completed.

2. DDT'S LANGUAGE

The language available to DDT users consists of constants, symbols, expressions, and commands.

CONSTANTS

A constant is any string of digits which may be followed by a B or D to signify an octal or decimal number, respectively. The number represented by the constant is evaluated and truncated to 24 bits. The base or radix for numbers is normally eight (octal), but the base may be changed by a DDT command. Constants are always accepted and printed by DDT in the current radix. However, if a constant is terminated by B or D, it is interpreted as octal or decimal regardless of the current radix.

Octal Constants	Decimal Constants
105B	982D
77777777 (where radix is 8)	46890 (where radix is 10)

SYMBOLS

A symbol is any legal string of letters and numerals containing at least one letter. In symbols of more than six characters only the first six are significant; thus, "ALPHABET" is equivalent to "ALPHAB". TAP instruction mnemonics are legal DDT symbols.

Legal Symbols	Illegal Symbols
ABC	135B
AB124	AB*CD
12XYZ	

Note that "135B" is not a legal symbol but rather an octal constant; "AB*CD" is not a legal symbol because "*" is neither a letter nor a numeral.

EXPRESSIONS

A DDT expression is a string of numbers or symbols connected with blanks or any of the operators shown below. These operators have the following significance:

- + addition
- - subtraction
- ;* (integer) multiplication
- ;/ (integer) division
- ;& logical AND
- ;< less than
- ;= equals
- ;> greater than

Expressions are always evaluated from left to right. All operators have the same precedence and parentheses are not allowed. The first symbol or number may be preceded by a minus sign. A blank is equivalent to a plus sign, except that the operand that follows is truncated to 14 bits before being added to the accumulated value of the expression.

The value of an expression is an 8-digit octal integer. An expression may be a single symbol or a constant.

Examples:

Expressions	Octal Value
10D+20D	00000036
5-2	00000003
LDA	07600000
LDA+10	07600010
LDA 10	07600010
SYM	00001212 (previously defined)
SYM 10	00001222
LDA SYM	07601212

COMMANDS

A command is an order typed by the user instructing DDT to perform a function. Commands in DDT generally take the form

e;c

where e is an expression and c is a character.

A question mark (?) may be typed at any time to abort the current command.

3. COMMANDS CONCERNING THE RADIX

As mentioned previously, all expressions are evaluated according to the current base or radix. DDT normally calculates expression values using base 8. To alter the current radix the user may type one of the commands listed below.

Command	Purpose
e;R	Sets the radix to 3, where e must be greater than 1.
;D	Sets radix to 10. All expressions will be calculated and printed in base 10.
;O	Sets radix to 8. All expressions will be calculated in base 8.

Until the radix is changed, all subsequent expressions are evaluated in the specified radix.

Example:

```
-DDT
2;R
111 + 101 = 1100
11111 + 1 = 100000
5;R
1234 + 1  = 1240
```

Note that an equals sign following an expression causes DDT to evaluate the expression and print the answer in a constant form.

The following is an example of expression evaluation using octal and decimal radixes.

Example:

```
;O
56+44    = 122 (RET)
32-11    = 21 (RET)
34;*56   = 2410 (RET)
12;/2    = 5 (RET)
= 5

;D
56+44    = 100 (RET)
32-11    = 21 (RET)
34;*56   = 1904 (RET)
12;/2    = 6 (RET)
= 6
```

Note that if only the equals sign is typed, the value of the last expression typed is printed. The last expression typed may be referred to as ;Q.

Example:

```
;D
1000+986 = 1986 (RET)
;Q       = 1986 (RET)
;Q+4     = 1990 (RET)
```

4. COMMANDS TO EVALUATE EXPRESSIONS

In addition to the equals sign, there are other commands that cause DDT to evaluate and print the value of an expression. These commands are listed below.

Command	Purpose
#	Types the value of ;Q as a signed integer.
←	Types the value of ;Q in symbolic form.
'	Types the value of ;Q as text.
@	Types the address part of ;Q in symbolic form.
@=	Types the value of ;Q as a numeric operation code.

Examples:

77777777#	-1 RET
12345670#	12345670 RET
40000000#	40000000 RET
76000000←	WIO 0,2 RET
07600000←	LDA 0 RET
07500010←	LDB 10 RET
41'	A RET
10220043'	A@C RET
144@=	14400000 RET

5. MEMORY ALLOCATION

Upon entering DDT, the user does not have memory automatically allocated to him. DDT obtains program memory only when (1) it is required for loading a binary file, (2) the execute command (;U) is given and a new page is needed to hold the instruction to be executed, and (3) a reference is made to a memory location that is not in a page previously obtained.

If another subsystem was being used prior to entering DDT, the user may still have some pages of memory assigned to him. In this case, he should release all previous memory by issuing the Executive command, CLEAR.[†]

Examples:

(1) -DDT RET
CAX;U

(2) -DDT RET
LDA 4000;U

In example 1, the command ;U causes the instruction CAX (04600400) to be loaded in location 240_8; DDT obtains the core page containing this location, which is page zero (all locations between 240_8 and 3777_8).

In example 2, the ;U command used in conjunction with LDA 4000 will cause DDT to obtain two pages of memory; the first to hold the instruction and the second because it contains the location 4000_8 referred to by the command.

Note that if DDT has already obtained some memory when the ;U command is issued, the instruction to be executed is loaded into the first unused memory location. "Unused memory" refers to memory that has been defined. The first unused memory location is usually referred to as ;F. Only if there is no unused memory beyond ;F will a new page be obtained. DDT always loads two BRS 10 instructions following the instruction to be executed.

The user may also use the backward slash \ (shift L) to obtain memory and enter data.

Example:

-DDT RET
240\ LDA 4000

This sequence causes only one page of memory to be obtained (i.e., page zero, which holds the instruction). Note that the page containing location 4000 is not obtained until the instruction is executed.

[†]RESET in 940 RAD/Tape Systems

6. OPENING REGISTERS

A major feature in the DDT language is the "opened register". A register (core location) may be opened at any time and its contents printed for examination and possible alteration. However, the page in which the location exists must have been obtained by DDT prior to opening the register.

To open a register, the user types the address and a slash. DDT responds by printing the contents of the location and issuing a tab. At this point the user may close the register without disturbing its contents, by striking a carriage return. However, if he wants to change the contents, he may type whatever he wishes at the tab stop, followed by a carriage return. The new information will then be stored in the opened register.

Example:

```
-DDT (RET)
NOP;U (RET)
240/     NOP (RET)
241/     BRS 12     CLA (RET)
242/     BRS 12 (RET)
243/     0 (RET)
244/     0 (RET)
4000/    ? (RET)
```

The slash can be used repeatedly to examine the contents of the contents of the contents, ad infinitum, of the opened register. If at any time during this procedure an expression is typed, that expression is stored in the original opened register.

Example:

```
-DDT (RET)
NOP;U (RET)
240/     NOP     300 (RET)
300/     0       400 (RET)
240/     300     /     400     /     0 (RET)
```

The other methods of opening a register are the line feed and the upward arrow. The line feed will open the register that follows the last one opened. A dot (typed as a period) always refers to the currently open register. Therefore, the line feed opens the .+1 register. Similarly, the upward arrow will open the register immediately preceding the last one opened; i.e., .-1.

Example:

```
-DDT
240      NOP (RET)
241/     BRS 12 (LF)
.=241    (RET)
242/     BRS 12 ↑
241/     BRS 12 ↑
240/     NOP (RET)
100+60=  160 (RET)
(LF)
241/     BRS 12 (RET)
```

Note that the request for the value of the expression 100+60 does not make DDT "forget" which location was inspected last.

Another command available is semicolon blank (;□). This is equivalent to a line feed except that nothing is printed. Its main use is in entering programs or data.

Example:

```
1000\    1;□ 2;□ 3 (RET)
```

is equivalent to

```
1000\    1 (RET)
1001\    2 (RET)
1002\    3 (RET)
```

Using the information covered so far, the user can enter a program and data into DDT by (1) instructing DDT to obtain memory and (2) opening the registers needed and altering their contents

Example:

```
-DDT (RET)
CLA;U
300;F
240/     CLA      LDX     =1000 (RET)
241/     BRS 12   LDA     0, 2 (RET)
242/     BRS 12   ADD     1, 2 (RET)
243/     0        STA     2, 2 (RET)
```

Note that literals and tag references are recognized by DDT just as they are by TAP. This is also true for "*", which indicates indirect addressing.

7. COMMANDS CONCERNING MODES

When the user enters DDT, the general mode is symbolic. Thus, if a register is opened with the slash, the contents will be printed in symbolic form. However, the user can change the current mode by issuing one of the commands listed in the table below. DDT will then print the contents of a register in the mode most recently set.

Command	Purpose
;]	Sets the current mode to "symbolic".
;[	Sets the current mode to "constant".
;$	Sets the current mode to "signed integer".
;"	Sets the current mode to "ASCII".

Example:

```
240/     LDA     SYM (RET)
;[
240/     7600456 (RET)
;]
240/     LDA     SYM (RET)
;$
240/     7600456 (RET)
;"
240/     ?!N (RET)
```

Although a mode is established by one of the above commands, the contents of a particular location may be printed in another mode by using one of the following commands:

Command	Purpose
e]	Opens location e and prints contents in symbolic form.
e[	Opens location e and prints contents as a constant.
e$	Opens location e and prints contents as a signed integer.
e"	Opens location e and prints contents in ASCII (control characters are preceded by &).
e\	Opens location e and does not print the contents.

Examples:

```
-DDT (RET)
NOP;U
240/     NOP     ADM     SYM (RET)
240]     ADM     SYM (RET)
240[     6300456 (RET)
240$     6300456 (RET)
240"     9!N (RET)
240\
```

If a location is opened by using one of these five commands, a subsequent line feed or upward arrow will cause the contents of the register thus opened to be printed in the specified mode rather than in the general mode.

Example:

```
;[
240"     ABC (LF)
241"     DEF (LF)
242"     GHI
```

In this example, the general mode is the "constant" mode, but the contents of locations 240, 241, and 242 are printed in ASCII.

The user can also control the form in which the address of a location is printed when it is generated by an expression, line feed, or upward arrow. The commands are as follows:

Command	Purpose
;R	Address will be printed in the relative mode.
;V	Address will be printed in the absolute mode.

Examples:

```
;R
240/     LDX ADR+5 (LF)
MS1/     LDB 0,2 (LF)
MS1+1/   LSH 10 (LF)
MS1+2/   STA X (LF)
MS1+3/   SKE .COLON (RET)

;V
240/     LDX 304 (LF)
241/     LDB 0,2 (LF)
242/     LSH 10 (LF)
243/     STA 273 (LF)
244/     SKE 274 (LF)
245/     BRU 247 (LF)
246/     0 (RET)
```

8. INDIRECT ADDRESSING

DDT provides a command for addressing a location indirectly. To open the location whose address is the last 14 bits of the value of the last expression typed, the user types a left parenthesis.

Example:

```
-DDT (RET)
;D
240\     LDA     400 (RET)
400\     1234
.
:
240/     LDA     400 (RET)
(
400/     1234    5678 (RET)
400/     5678
```

If another slash had been typed following the 5678 typed by the user, any expression following the slash would have been stored in the location that was opened first (in this case, location 400).

9. CENTRAL REGISTERS

Because time-sharing requires that several simultaneous users share the computer's central registers, the contents of the central registers are assigned as the values of special symbols. Thus, to interrogate the contents of the central registers, the user has available the following commands:

Command	Purpose
;A=	Prints the contents of the A register
;B=	Prints the contents of the B register
;X=	Prints the contents of the X register
;L=	Prints the contents of the location counter

Other appropriate commands (see "Commands to Evaluate Expressions", above) may be used with these symbols instead of the equals sign.

Example:

```
;A=    0
;B=    1
;X"    ABC
;L=    280
```

The following commands are available to alter the contents of the central registers.

Command	Purpose
e;A	Stores the value of e in the A register
e;B	Stores the value of e in the B register
e;X	Stores the value of e in the X register
e;L	Stores the value of e in the location counter

Example:

```
5;A
;A=5 (RET)
```

10. SYMBOL DEFINITION

A symbol may be defined by using a DDT command, or it may enter DDT predefined (via the symbol table that accompanies an object program).

To define a symbol by using a DDT command, the user may type

e < s >

where e is any expression and s is any legal symbol. This will cause DDT to assign the value of the expression to the symbol.

Example:

```
400 <DATA1>
DATA1/  0   1 (RET)
DATA1/  1 (RET)
```

The first line of the example assigns DATA1 to location 400. The second line puts a "1" in location 400.

Similarly, typing a symbol followed by a colon will assign the current location (the address of the opened register) as the value of the symbol.

Example:

```
-DDT
240\        START:   LDA     500 (RET)
START=240 (RET)
START/      LDA      500
```

The use of the "@" sign following a symbol will cause the last expression printed by DDT or typed by the user to be assigned as the value of that symbol.

Example:

```
-DDT
400\        777 (RET)
.
.
400/        777      BEGN@    CAX (RET)
BEGN=       777 (RET)
777/        CAX (RET)
```

11. ERASING SYMBOLS

The user can "undefine" a symbol by using the kill (;K) command. If the ";K" is preceded by a symbol, just that symbol is "killed"; ";K" alone will kill all defined symbols. The use of ";K" causes the computer to print "OK" and wait for a carriage return. This prevents inadvertent erasures.

The command ";U" will cause all undefined symbols to be listed on the Teletype.

The following example illustrates the use of the kill command.

Example:

```
-DDT
NOP;U
240/  0          START: (RET)
500/  0          ITEMNO: (RET)
START=240 (RET)
ITEMNO=500 (RET)
START;K
ITEMNO=500 (RET)
START=? (RET)
;K - - OK (RET)
ITEMNO= ?(RET)
START=?
```

12. BLOCK STRUCTURE

A limited facility called the "block structure" is provided to simplify referencing local symbols that are defined in more than one program. The block structure of a program is organized in the following manner: Every binary program file loaded by DDT constitutes a separate block. Also, there is an intrinsic block called "block zero". Any symbol input to DDT has a block number associated with it. It also has a type: (1) external (global) or (2) local.

All instruction mnemonics are associated with block zero and are external. When a binary file written by TAP is loaded by DDT, it defines a new block; all symbols defined during the assembly and written on the binary file are associated with that block. Any symbols that were declared by TAP to be external are external symbols. All other symbols are local symbols.

External symbols must be unique within an entire program, since they are recognized at all times. Local symbols are recognized according to the following rules:

1. At any given time, there is one block that is called the primary block. A block becomes the primary block when one of its registers is opened. All other blocks are secondary. All symbols associated with the primary block will be recognized.
2. If a symbol is used which is neither external nor in the primary block, the entire symbol table is scanned for it. If it occurs in only one block, that block becomes primary and the symbol is recognized properly. If it occurs in more than one block, an error is indicated.
3. A symbol may be explicitly qualified by

   s_1 & s_2

   where s_1 is the name of the block (the name appearing in the label field of the IDENT statement in TAP) and s_2 is the symbol in question.

Symbols defined by the "@" or ":" commands are local to the block that is primary when the command is given. Symbols defined by "<>" are external and not local to any block. Not all symbols appearing in a block will necessarily be local to that block.

To obtain a listing of all block names, the user may type "%&".

13. WORD SEARCH

DDT gives the user the capability of searching memory, between any given limits, for a specified word. Also, a mask may be specified to determine which bits of the word are to be compared. The commands used are listed below.

Command	Purpose
e;M	Mask – sets the mask to the value of e.
e;1	Lower bound – sets the lower bound to the value of e.
e;2	Upper bound – sets the upper bound to the value of e.
e_1, e_2;L	An alternative command to set lower (e_1) and upper (e_2) bounds.
e;W	Word search – searches memory between limits ;1 and ;2 for locations which match e when both are masked by the value of ;M. All matching words are printed.
e;E	Effective word search – searches memory between limits ;1 and ;2 for effective address equal to e.
e;#	Not-word search – same as e;W except that all words not matching e are printed.

The following two examples illustrate the word search commands. In both examples the information in Part I represents a DDT listing of a program segment; the information in Part II illustrates various word search operations by the user.

Example A

Part I

```
240/    LDX  304 (LF)
241/    LDB  0,2 (LF)
242/    LSH  10 (LF)
243/    STA  273 (LF)
244/    SKE  274 (LF)
245/    BRU  247 (LF)
246/    BRU  270 (LF)
247/    TCO  273 (LF)
250/    CLA (LF)
251/    LSH  10 (LF)
252/    STA  273 (LF)
253/    SKE  274 (LF)
254/    BRU  256 (LF)
255/    BRU  270 (LF)
256/    TCO  273 (LF)
257/    CLA (LF)
260/    LSH  10 (RET)
```

Part II

```
240,255;L

CLA;W
250/    CLA

777;M

273;W
243/    STA  273
247/    TCO  273
252/    STA  273
```

Example B

Part I

```
MSG/    LDX  ADR+5 (LF) (RET)
MS1/    LDB  0,2 (LF)
MS1+1/  LSH  10 (LF)
MS1+2/  STA  X (LF)
MS1+3/  SKE  COLON (LF)
MS1+4/  BRU  MS1+6 (LF)
MS1+5/  BRU  MS2 (LF)
MS1+6/  TCO  X (LF)
MS1+7/  CLA (LF)
MS1+10/ LSH  10 (LF)
MS1+11/ STA  X (LF)
MS1+12/ SKE  COLON (LF)
MS1+13/ BRU  MS1+15 (LF)
MS1+14/ BRU  MS2 (LF)
MS1+15/ TCO  X (LF)
MS1+16/ CLA (LF)
MS1+17/ LSH  10 (LF)
MS1+20/ STA  X (LF)
MS1+21/ SKE  COLON (LF)
MS1+22/ BRU  MS1+24 (LF)
MS1+23/ BRU  MS2 (LF)
MS1+24/ TCO  X (LF)
MS1+25/ EAX  1,2 (LF)
MS1+26/ BRU  MS1 (LF)
MS2/    TCO  LF (LF)
MS2+1/  TCO  CR (LF)
MS2+2/  BRS  12 (LF)
X/      0 (LF)
COLON/  32 (LF)
LF/     152 (LF)
CR/     155 (RET)
```

Part II

```
MSG;1
X;2
COLON;E
MS1+3/  SKE COLON
MS1+12/ SKE COLON
MS1+21/ SKE COLON
```

14. PROGRAM ALTERATION

In debugging a program, it is often necessary to insert or delete instructions. In DDT this can be done easily without reassembling the program.

INSERTIONS

To insert instructions (a procedure often referred to as patching), the user issues one of the commands listed in the table below. In response, DDT issues a carriage return and line feed, prints a right parenthesis, and waits for the user to type the insertion.

Command	Purpose
e;I	Causes instructions to be inserted after location e.
e)	Causes instructions to be inserted before location e.

After one of the patch commands has been issued, 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 a line feed or ; □, exactly as though the program were being typed in with the "\" command instead of as a patch. Two other commands are legal in the patch mode:

1. The colon, which may be used to define a local symbol with a value equal to the current location.
2. The carriage return, which terminates the patch.

The e) patch command followed by the instructions to be inserted will cause the following sequence to occur.

1. The instructions to be inserted are loaded sequentially beginning with ;F (;F being the first unused memory location).
2. Location ;F is updated.
3. The instruction at the point of insertion is then copied into location ;F.
4. Location ;F is updated.
5. Two return branches are loaded into the new ;F and ;F+1 so that, if the patch is a skip instruction, the program will operate correctly.
6. Location ;F is updated.

Example:

Before	After
⋮	⋮
250/ LDA A1	250/ LDA A1
251/ STA A2	251/ BRU 1000
⋮	⋮
;F=1000	1000/ ETR =77
	1001/ STA A2
	1002/ BRU 252
	1003/ BRU 253
⋮	
251)	
)ETR =77	

The command ";I" would cause the new instructions (see example above) to be inserted also, but STA A2 would appear in location 1000 rather than 1001.

LITERALS

As mentioned previously, literals have the same format and meaning in DDT as in the assembler; i.e., an equals sign following a blank signals the beginning of a literal that is terminated by any of the characters which ordinarily terminate an expression. In contrast to the syntax of the assembler, the expression in a DDT literal must be defined.

The literal is looked up in the literal table. If it does not appear in the table, it is stored in the current ;F location and ;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 a -1 to be stored at 1000B. Then, the instruction is equivalent to "LDA 1000B", and the new value of ;F is 1001B; however, in the patch mode, literals are saved and are not stored until the patch is completed. Otherwise, they would interfere with the patch.

DELETIONS

To delete instructions, using DDT, the user may open a register and replace its contents by a NOP instruction.

Example:

```
240/  LDA  Z  (LF)
241/  CLB        NOP (LF)
242/  LSH  6     NOP (LF)
243/  STA  Z+1 (LF)
```

PSEUDO RELABELING†

Command	Purpose
$e_1, e_2;R$	Sets pseudo relabeling for a program according to the value of e_1 and e_2.

†See SDS 940 Time-Sharing System Reference Manual.

15. PROGRAM EXECUTION

Program execution is of primary importance to the DDT user. Therefore, DDT provides the user with sophisticated execution capabilities. For example, the user may start and stop program execution at any given address. Also, he may execute one instruction at a time or N instructions at a time. He can list up to four breakpoints, or he can specify how many times to pass a breakpoint before stopping.

BREAKPOINTS

The term "breakpoint" simply means "the address at which program execution is to stop". The break always occurs before the execution of the instruction at the breakpoint location. DDT then prints this address and the contents of the central registers A, B, and X. DDT has four breakpoints which can be set simultaneously. They are numbered 0-3. The commands to specify, set, clear, or list breakpoints are listed below.

Command	Purpose
e!	Sets breakpoint 0 at the address e.
n;e!	Sets breakpoint n (where $0 \leq n \leq 3$) at the address e.
!	Clears all breakpoints.
;!	Lists all breakpoints.
n;!	Clears breakpoint n.

EXECUTION COMMANDS

To control execution, the following commands are available:

Command	Purpose
e;G	Starts execution at location e.
;P	Restarts execution at the value of the location counter (;L).
e;P (when e is an integer)	Restarts execution at ;L and breaks after e breakpoints have been reached.
;N	Executes the next instruction and then breaks.
e;N (when e is an integer)	Executes e instructions and breaks.
e;S	Executes the next instruction, breaks, and repeats this sequence e times.

The following example demonstrates the use of the above commands in a typical debugging session. The program to be debugged is a message printer.

Part I is a DDT listing of the program. Part II is the debugging and execution of the program.

Note that in Part II the first three characters of the message appear on lines labeled 1, 2, and 3. The remaining characters of the message, plus some extraneous characters, appear on line 4.

By reexecuting the program, it is discovered that a CLA instruction is missing at the beginning of the program. Line 5 inserts the missing instruction and line 6 changes the address of the BRU in MSG+26 so that the CLA instruction is included in the loop.

Example:

Part I

```
MSG/      LDX  ADR+5 (LF)
MS1/      LDB  0,2 (LF)
MS1+1/    LSH  10 (LF)
MS1+2/    STA  X (LF)
MS1+3/    SKE  COLON (LF)
MS1+4/    BRU  MS1+6 (LF)
MS1+5/    BRU  MS2 (LF)
MS1+6/    TCO  X (LF)
MS1+7/    CLA (LF)
MS1+10/   LSH  10 (LF)
MS1+11/   STA  X (LF)
MS1+12/   SKE  COLON (LF)
MS1+13/   BRU  MS1+15 (LF)
MS1+14/   BRU  MS2 (LF)
MS1+15/   TCO  X (LF)
MS1+16/   CLA (LF)
MS1+17/   LSH  10 (LF)
MS1+20/   STA  X (LF)
MS1+21/   SKE  COLON (LF)
MS1+22/   BRU  MS1+24 (LF)
MS1+23/   BRU  MS2 (LF)
MS1+24/   TCO  X (LF)
MS1+25/   EAX  1,2 (LF)
MS1+26/   BRU  MS1 (LF)
MS2/      TCO  LF (LF)
MS2+1/    TCO  CR (LF)
MS2+2/    BRS  12 (LF)
X/        0 (LF)
COLON/    32 (LF)
LF/       152 (LF)
CR/       155 (LF)
ADR/      102@21043  '  ABC (LF)
ADR+1/    110@22446  '  DEF (LF)
ADR+2/    116@24051  '  GHI (LF)
ADR+3/    124@25454  '  JKL (LF)
ADR+4/    MUL  0  '  :  (LF)
ADR+5/    ADR (LF)
ADR+6/    0 (RET)
```

Example:

Part II

```
        ;C
        MS1+1!
        MSG;G
        MS1+1          11223000    10221043    277
        ;N
        MS1+2          11400041    10421400    277
        ;N
        MS1+3          11400041    10421400    277
        X/             11400041
        MS1+6!
        ;P
        MS1+6          11400041    10421400    277
        ;N
(1) --> AMS1+7         11400041    10421400    277
        MS1+15!
        ;P
        MS1+15         42          10600000    277
        ;N
(2) --> BMS1+16        42          10600000    277
        MS1+24!
        ;P
        MS1+24         43          0           277
        ;N
(3) --> CMS1+25        43          0           277
        !
        ;P
(4) --> DEFGHIJKL: (ESC)
        MS2!
        MSG;G
        ABCDEFGHIJKL: (ESC)
        MS1+25!
        MSG;G
        ABCMS1+25      43          0           412
        ;P
        DEFMS1+25      46          0           413
        ;P
        GHIMS1+25      51          0           414
        ;P
        JKLMS1+25      54          0           415
        ;N
        MS1+26         54          0           416
        ;N
        MS1            54          0           416
        ;N
        MS1+1          54          6400000     416
        ;N
        MS1+2          26032       0           416
        X/             54
        ;J
(5) --> MSG;I
        )              CLA
        ADR+5/         ADR (LF)
        ADR+6/         LDX  ADR+5 (LF)
        ADR+7/         CLA (RET)
        MS1/           LDB  0,2 †
        MSG/           BRU  ADR+6 (RET)
(6) --> MS1+26/        BRU  MS1    BRU   ADR+7
        MSG;G
        ABCDEFGHIJKL
```

ADDITIONAL EXECUTION COMMANDS

Four other commands are provided, to make the debugging process as simple as possible. These are listed below.

Command	Purpose
e;O (when e = 1)	Causes POP's to be treated as one instruction for :N and :S.
e;O (when e = 0)	Causes POP's to be treated as part of user's code.
e;U (when e = 1)	Causes subroutines to be treated as one instruction.
e;U (when e = 0)	Causes subroutines to be treated explicitly.

16. PANICS

In conjunction with program execution, 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 ESCAPE button.
4. Panics generated by the execution of BRS 10 in the user's program.

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

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

The other two types of panics cause DDT to ring the bell and issue a carriage return. Both ";L" and "." will 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 are not available. Therefore, DDT types a "?".

Two other panic conditions are possible in DDT.

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

Moreover, 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 perform properly. Attempts to proceed through unreasonable instructions will cause the error comment

S > >

to be typed.

17. INPUT

To execute and debug a program, using DDT, the object code of the program is usually located on a disc file prepared by TAP. TAP automatically writes a list of symbols along with binary object code that DDT can access.

To instruct DDT to read and load the file containing the object code and symbol table, the user types

;T /file/ (RET)

This will cause the relocatable program to be loaded, beginning at location 240_8. After input has been completed, the first location not used by the program (;F) is printed on the Teletype. A subsequent ";T" command would cause loading to begin at this location.

In the event that an alternative loading position is desired, the user may type

e;T /file/ (RET)

where e is any expression (evaluated in octal). The block is then loaded beginning with location e.

In addition to the loading procedure, ";T" causes DDT to read the list of symbols prepared by TAP and to append this list to its own (which simply consists of the instruction mnemonics recognized by TAP).

If the user wishes to have a program loaded by DDT but does not want the local symbols on the binary file added to the symbol table, he may use the command

e;Y /file/ (RET)

As before, if e is omitted, loading will begin with location 240_8 or the first location available.

Example:

```
-TAP
BINARY      /BIN/ (RET)
OLD FILE
INPUT       /TEST/ (RET)

45          CELLS USED BY PROGRAM

-DDT
;T          /BIN/ (RET)
    305
;F = 305
```

18. OUTPUT

Commands are available that will cause symbols to be written on a specified file for permanent storage. These commands are listed below.

Note that ";W" will also cause all IDENT labels to be printed.

Command	Purpose
;W /file/	Causes all global symbols to be written on the specified file, in a format that can be read with ";T".
;C /file/	Causes all symbols to be written on the specified file.

APPENDIX A. CHARACTER CODES

SDS 940 INTERNAL, ASCII, TELETYPE, LINE PRINTER, AND CARD CODES

INT	ASCII	TTY	LP	CARDS	INT	ASCII	TTY	LP	CARDS
00	40				40	100	@	↙	78
01	41	!	!	-0	41	101	A	A	+1
02	42	"	'	84	42	102	B	B	+2
03	43	#	≠	+78	43	103	C	C	+3
04	44	$	$	-38	44	104	D	D	+4
05	45	%	⌣	085	45	105	E	E	+5
06	46	&	△	-78	46	106	F	F	+6
07	47	'	'	84	47	107	G	G	+7
10	50	(	(	048	50	110	H	H	+8
11	51	)	)	+48	51	111	I	I	+9
12	52	*	*	-48	52	112	J	J	-1
13	53	+	+	+	53	113	K	K	-2
14	54	,	,	038	54	114	L	L	-3
15	55	-	-	-	55	115	M	M	-4
16	56	.	.	+38	56	116	N	N	-5
17	57	/	/	01	57	117	O	O	-6
20	60	0	0	0	60	120	P	P	-7
21	61	1	1	1	61	121	Q	Q	-8
22	62	2	2	2	62	122	R	R	-9
23	63	3	3	3	63	123	S	S	02
24	64	4	4	4	64	124	T	T	03
25	65	5	5	5	65	125	U	U	04
26	66	6	6	6	66	126	V	V	05
27	67	7	7	7	67	127	W	W	06
30	70	8	8	8	70	130	X	X	07
31	71	9	9	9	71	131	Y	Y	08
32	72	:	:	58	72	132	Z	Z	09
33	73	;	;	-68	73	133	[	[	+58
34	74	<	<	+68	74	134	\	\	068
35	75	=	=	38	75	135	]	]	-58
36	76	>	>	68	76	136	↑	≠	082
37	77	?	?	+0	77	137	←	≠	087

SPECIAL CODES

INTERNAL	ASCII	CONTROL	FUNCTION
141	1	A	SOM
142	2	B	EOA
143	3	C	EOM
144	4	D	EOT
145	5	E	WRU
146	6	F	RU
147	7	G	BELL
151	11	I	TAB
152	12	J	LINE FEED
153	13	K	VT
154	14	L	FORM
155	15	M	RETURN
161	21	Q	X - ON
162	22	R	TAPE
163	23	S	X - OFF
164	24	T	TAPE
165	25	U	ESCAPE
166	26	V	SPACE
167	27	W	
170	30	X	
171	31	Y	
172	32	Z	

APPENDIX B. SDS 940 INSTRUCTIONS

MNEMONIC	CODE	NAME
LOAD/STORE		
LDA	76	Load A
STA	35	Store A
LDB	75	Load B
STB	36	Store B
LDX	71	Load Index
STX	37	Store Index
EAX	77	Copy Effective Address into Index
XMA	62	Exchange Memory and A
ARITHMETIC		
ADD	55	Add
ADC	57	Add with Carry
ADM	63	Add to Memory
MIN	61	Memory Increment
SUB	54	Subtract
SUC	56	Subtract with Carry
MUL	64	Multiply
DIV	65	Divide
LOGICAL		
ETR	14	Extract (AND)
MRG	16	Merge (OR)
EOR	17	Exclusive OR
REGISTER CHANGE		
CLA	0 46 00001	Clear A
CLB	0 46 00002	Clear B
CLAB	0 46 00003	Clear AB
CLX	2 46 00000	Clear Index
CLEAR	2 46 00003	Clear A, B, and Index
CAB	0 46 00004	Copy A into B
CBA	0 46 00010	Copy B into A
XAB	0 46 00014	Exchange A and B
ABC	0 46 00005	Copy A into B, Clear A
BAC	0 46 00012	Copy B into A, Clear B
CAX	0 46 00400	Copy A into Index
CXA	0 46 00200	Copy Index into A
XXA	0 46 00600	Exchange Index and A
CBX	0 46 00020	Copy B into Index
CXB	0 46 00040	Copy Index into B
XXB	0 46 00060	Exchange Index and B
CNA	0 46 01000	Copy Negative into A
BRANCH		
BRU	01	Branch Unconditionally
BRX	41	Increment Index and Branch
BRM	43	Mark Place and Branch
BRR	51	Return Branch
TEST/SKIP		
SKE	50	Skip if A Equals Memory
SKG	73	Skip if A Greater Than Memory
SKM	70	Skip if A Equals Memory on B Mask
SKA	72	Skip if A and Memory do not Compare Ones
SKB	52	Skip if B and Memory do not Compare Ones
SKN	53	Skip if Memory Negative
SKR	60	Reduce Memory; Skip if Negative
SKD	74	Difference Exponents and Skip

MNEMONIC	CODE	NAME
SHIFT		
RSH	0 66 00xxx	Right Shift AB
LRSH	0 66 24xxx	Logical Right Shift AB
RCY	0 66 20xxx	Right Cycle AB
LSH	0 67 00xxx	Left Shift AB
LCY	0 67 20xxx	Left Cycle AB
NOD	0 67 10xxx	Normalize and Decrement X
CONTROL		
NOP	20	No Operation
EXU	23	Execute
INPUT/OUTPUT		
EOM	02	Energize Output M
SKS	40	Skip if Signal Not Set
PIN	33	Parallel Input
POT	13	Parallel Output
OVERFLOW		
OTO	0 22 00100	Overflow Indicator Test Only
REO	0 22 00010	Record Exponent Overflow
ROV	0 22 00001	Reset Overflow Indicator
OVT	0 22 00101	Overflow Indicator Test and Reset

APPENDIX C. SYSTEM PROGRAMMED OPERATORS

MNEMONIC	NUMBER	FUNCTION
BIO	176	Block input/output
TCO	175	Teletype character output
TCI	174	Teletype character input
BRS	173	Branch to system
CTRL	172	Input/output control
SBRR	171	System branch and return
SBRM	170	System subroutine call
STP	167	Store pointer
LDP	166	Load pointer
GCI	165	Get character and increment
WCH	164	Write character
SKSE	163	Skip on string equal
SKSG	162	Skip on string greater
CIO	161	Character input/output
WIO	160	Word input/output
WCI	157	Write character and increment
FAD	156	Floating add
FSB	155	Floating subtract
FMP	154	Floating multiply
FDV	153	Floating divide
EXS	152	Execute instruction in the system mode
OST	151	Output to specified Teletype
IST	150	Input from specified Teletype
DWO	145	Disc word output (random)
DWI	144	Disc word input (random)
DBO	143	Disc block output (random)
DBI	142	Disc block input (random)
ISC	141	Internal to string conversion (floating output)
SIC	140	String to internal conversion (floating input)
GCD	137	Get character and decrement
STI	136	Simulate Teletype input
WCD	135	Write character and decrement

APPENDIX D. DDT SUMMARY

COMMANDS CONCERNING THE RADIX

e;R	Sets radix to e
;D	Sets radix to 10
;O	Sets radix to 8

COMMANDS TO EVALUATE EXPRESSIONS

=	Types value of ;Q (where ;Q is the last expression typed)
#	Types value of ;Q as a signed integer
←	Types value of ;Q in symbolic
'	Types value of ;Q as text
@	Types the address part of ;Q in symbolic
@=	Types the value of ;Q as a numeric op code

OPENING REGISTERS

e/	Opens location e and prints its contents in the current mode
↑	Opens preceding location and prints its contents
Line Feed	Opens next location and prints contents
;□	Opens next location but does not print contents

COMMANDS CONCERNING MODES

;]	Sets current mode to "symbolic"
;[	Sets the current mode to "constant"
;$	Sets current mode to "signed integer"
;"	Sets current mode to "ASCII"
e]	Opens location e and prints contents in symbolic
e[	Opens location e and prints contents as a constant
e$	Opens location e and prints contents in ASCII
e\	Opens location e and does not print contents
;R	Prints address in relative mode
;V	Prints address in absolute mode

INDIRECT ADDRESSING

(	Opens location whose address is the last 14 bits of the value of the last expression typed

CENTRAL REGISTERS

;A=	Prints the contents of the A register
;B=	Prints the contents of the B register
;X=	Prints the contents of the X register
;L=	Prints the contents of the Location Counter
e;A	Stores the value of e in the A register
e;B	Stores the value of e in the B register
e;X	Stores the value of e in the X register
e;L	Stores the value of e in the Location Counter

SYMBOLIC DEFINITION

e <s>	Assigns the value of e as the value of the symbol
s:	Assigns the current location as the value of the symbol
s@	Assigns the value of ;Q as the value of the symbol

ERASING SYMBOLS

s;K	Kills symbol "s"
;K	Kills all symbols
;U	Lists all undefined symbols

WORD SEARCH

e;M	Sets the mask to the value of e
e;1	Sets the lower bound to the value of e
e;2	Sets the upper bound to the value of e
e_1, e_2;L	Sets the lower bound to the value of e_1 and the upper bound to the value of e_2.
e;W	Searches memory between lower and upper bounds for locations which match e when both are masked by the value of ;M
e;E	Searches memory between lower and upper bounds for effective address equal to e
e;#	Same as e;W except that all words not matching e are printed

PROGRAM ALTERATION

e;I	Causes instructions to be inserted after location e
e)	Causes instructions to be inserted before location e
e_1, e_2;R	Relabeling

PROGRAM EXECUTION

e!	Sets breakpoint 0 to the address e
n;e!	Sets breakpoint n (where n can be 0-3) to the address e
!	Clears all breakpoints
;!	Lists all breakpoints
n;!	Clears breakpoint n
e;G	Starts execution at location e
;P	Restarts execution at the value of the location counter
e;P	Restarts execution at ;L and breaks after e breakpoints have been reached
;N	Executes the next instruction and then breaks
e;N	Executes e instructions and breaks
e;S	Executes the next instruction, breaks, and repeats this sequence e times
e;O (when e=1)	Causes POP's to be treated as one instruction for ;N and ;S
e;O (when e=0)	Causes POP's to be treated as part of user's code
e;U (when e=1)	Causes subroutines to be treated as one instruction
e;U (when e=0)	Causes subroutines to be treated explicitly

INPUT/OUTPUT

;T /file/	Loads binary file and symbol table beginning with location ;F
e;T /file/	Loads binary file and symbol table beginning with location e
;Y /file/	Loads binary file and external symbols beginning with location ;F
e;Y /file/	Loads binary file and external symbols beginning with location e
;W /file/	Causes all global symbols to be written on the specified file
;C /file/	Causes all symbols to be written on the specified file