PDP-1 PROGRAM LIBRARY

NUMBER: Digital - 1 - 3 - S

NAME: DEC Debugging Tape

AUTHOR: A. Kotok - DEC (MIT-1)

DATE: Writeup revised August 13, 1964

SPECS: MS, MB - SA 6000

ABSTRACT: DDT facilitates online debugging.

DDT-1 PROGRAM DESCRIPTION

Users of most computers, especially large-scale ones, are familiar with the procedure of submitting a new program for a run and receiving, along with the compilation and assembly listings, a dump and perhaps a storage map of the symbols used, together with a few remarks about the failure of the program to run properly. If the user is lucky enough to be present when his program is processed, he may get additional information from the console lights, motion of tapes, etc., but his correcting must be done away from the machine. Getting a program to work under these conditions takes a long time.

DDT helps shorten this debugging time by allowing the user to work on his program at the computer, to control its operation, and to modify the program or its data at will. Tracking down a subtle error in a complex piece of coding is a laborious and frustrating job by hand, but with DDT's breakpoint facility, the user can interrupt his program at any point and examine the state of the machine. In this way, he can quickly locate sources of trouble.

The programmer may insert corrections and patches and try them out immediately; those that work can be punched out on the spot in the form of loadable patch tapes, eliminating the necessity of creating new symbolic tapes and reassembling each time an error is found and remedied. DDT also maintains a symbol table, allowing a programmer to discuss matters with the computer in the language of his own program.

DDT occupies registers 6000 to 7750. The starting address is 6000. DDT carries with it a permanent table, starting at 5777 and extending toward 0, consisting of all standard defined PDP-1 mnemonics. This may be augmented by symbol definitions from the user's program tape or from the keyboard.

DEFINITIONS

A symbol is a string of from one to three letters or numerals. A number is a string of up to 6 digits.

An expression is a string of symbols and numbers separated by the following characters:

space	a separation character meaning arithmetic + (plus)
+	a separation character meaning arithmetic + (plus)
-	a separation character meaning arithmetic - (minus)
V	a separation character meaning boolean inclusive or.
Λ	a separation character meaning boolean and

All other characters are either used for control or are illegal.

When a register is opened, its contents are printed out and become available for modificaction.

When a register is closed, any modifications requested are made and further access to the register is denied until it is opened again.

Most DDT operations are specified by a single letter in upper case. Typing such a symbol followed by a carriage return causes DDT to perform the operation.

In the following discussion, all numbers are octal integers. In the examples, "ldc", "tab", and "beg" are symbols in the user's hypothetical program. "c(r)" means "the contents of register R". All underlined expressions in the examples are those typed by DDT; expressions without underlining are those typed by the user. DDT responds to errors by typing a "?" and ignoring the error. The user may cancel a line at any time before the carriage return, by typing × (multiplication sign).

USING DDT

The user first reads DDT into the computer. All subsequent operations, including loading the program to be debugged, are performed through DDT on the typewriter. All printed output appears on the typewriter.

Loading the Program

Z — All of memory through the highest register not used by DDT (i.e., to the bottom of the symbol table) is set to zero.

fa<laZ Zero memory between fa and la except that part, if any occupied by DDT.

K DDT's symbol table is restored to its initial list of permanent symbols. If any of these have been modified, the original value is not restored.

Y A binary tape in the reader is read into memory. No new symbols will be entered into DDT's table. The tape is read into storage between the limits in register M+1 and M+2. If a checksum error is encountered, the program will stop. It is then possible to move the tape back one block, restart the reader, and press Continue to continue reading.

T Read a Macro Symbol Table and merge it with the existing table. Definitions on tape take precedence over definitions in storage. The special symbols 1s, 2s, up to 9s, are not entered. The lowest register occupied by the symbol table is typed out upon completing reading the symbol section of the tape. This number may be found in register F. Checksum errors are handled as in Y.

Punching Operations

DDT will punch a standard Macro format tape with the title in readable format.

L — Put DDT into the title punch listen mode. Characters typed in are punched out in readable format on paper tape. Terminating characters are tab, carriage return, or back-space, which do the following:

- tab: Sets DDT to punch read-in-mode data blocks.
- carriage return: punches a standard input routine and sets DDT to punch standard checksum data blocks.
- back space: sets DDT to punch standard checksum data blocks but punches no input routine.

fa<laD — Punches data blocks from the first address to the last address in the format specified by L above. The first address and the last address are any symbolic expressions where fa is less than or equal to la.

adrJ — Punches the start (jmp) block to the address specified to denote the end of the binary tape.

.(center dot) — When a register is open, make the modification, if any, and punch it in the format specified by L.

Example: Punch out registers 4-10 and 100-350, in standard checksummed blocks. The program's name is MICRO and it starts at 100.

L (carriage return)
MICRO
4<10D 100<350D 100J

Program Examination

These operations allow any register in memory to be examined and modified.

/ This is the register examination character. The expression typed immediately preceding the / is the address of the register to be opened.

Example 1 When the user types

ldc 20/ DDT will immediately move to the next tab stop, print out the contents of the register ldc 20, and skip to the following tab stop. The resulting line might look like this:

ldc 20/ add 2653

If the user wishes to change the contents of the register, he types in the new information:

ldc 20/ add 2653 add 2663

Carriage Return: This causes DDT to place the new information in the open register and to close it. If no modifications were typed before the CR, the register is closed unchanged.

/ (alternate use) The slash may be used to open a register addressed by the currently opened one. If in example 1 the user wished to examine the contents of location 2653, he would have typed a "/" instead of the modifying instruction. The results may be looked like this.

Example 2

ldc 20/ add 2653 / isp 3062

Here, isp 3062 is the contents of register 2653, which is now open. The previously opened register, ldc 20, has been closed.

> The "greater than" sign works like / except any modifications to the opened register will be made before the register addressed by the new contents of the closed register is opened.

Example 3

ldc 20/ add 2653 add 2663> 7044

The contents of register 2663 is 7044.
Use of ">" does not alter the sequence of locations.

Back Space: This has the same effect as a carriage return, in that it closes an opened register. The next register in sequence is then opened. In Example 1, typing a backspace would cause register ldc 21 to be opened, thus:

Example 4

```
ldc 20/    add 2653        add 2663    (backspace)
ldc 21/    dac 600
```

Likewise in Example 2:

```
ldc 20/    add 2653  /     isp 3062    (backspace)
ldc 21/    dac 600
```

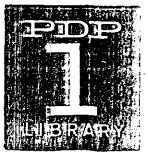
Note that the sequence of locations established by loc 20 is not altered by the use of "/" to open a chain of successively addressed registers.

↑ Up arrow has the same effect as backspace except that the preceding register in the sequence is opened.

Example 5

```
ldc 21/    dac 600     ↑
ldc 20/    add 2653
```

tab This causes an opened register to be closed after modification if any; the register which is now addressed by its contents is opened, establishing a new sequence. Substituting a tab for a backspace in Example 4 would have the following result:



Example 6

```
ldc 20/     add 2653        add 2663 (tab)
2663/       jda 235
```

a backspace would now result in:

```
ldc 20/     add 26653       add 2663 (tab)
2663/       jda 235         (backspace)
2664/       dzm ldc 35
```

The backspace may be used at any time. DDT always remembers the last register to be opened in normal sequence (that is, by any method except the alternate use of "/" and ">") and will open the next sequential register any time a backspace is typed. Intervening carriage returns or other operations have no effect on the sequence.

Verification of Program in Memory

To discover if a program in memory is unchanged, core may be checked against the binary tape by use of:

V reads a binary tape in Macro binary block format and compares it against memory between the locations specified in M+1 and M+2. No change is made to memory. Discrepancies are typed out in the form:

```
location/       memory     tape
```

Checksum errors may be handled as in Y.

Mode Control

The preceding examples show that information may be entered and typed out in several different forms. Expressions may be symbolic or absolute octal or decimal integers; register addresses may be relative or absolute. DDT can be conditioned to print information in any of these modes. The user, however, is never restricted and may always use the representation most convenient when typing input.

The first two operations below determine the form in which DDT types out register addresses; the second two determine the forms for other information.

O conditions DDT to type out locations in absolute octal form, e.g:

2663/

R sets the mode to print location addresses relative to a program symbol, thus:

ldc 21/

C sets the information mode to print octal integers (constants), thus/

302653

S sets the information mode to print symbolic expressions:

dzm ldc+35

H puts DDT into the hoctal mode. All numeric printouts are in octal.

U puts DDT into the unhoctal decimal mode. All numeric printouts are in decimal.

' causes the last numeric value typed to be taken as decimal on input. The ' must immediately follow the number.

" causes the last three characters typed in to be taken as their concise code value. This applies only to letters or numbers.

Example 7

ldc 50/ 0 abc" (c.r.)

ldc 50/ 616263

While operating in one mode, the user sometimes wants information in another form. He can force this representation without leaving the current mode, by using the following operators.

= causes the last previous expression typed by DDT or the user to be printed as an integer. Its use is illustrated below.

Example 8

ldc 20/ add 2653 = 402653

→ types out the last quantity as an instruction

Example 9

ldc 20/ 402652 → add 2763

~ types out the last quantity as concise code characters in the order, left, middle, right. Spaces are deleted.

Example 10

ldc+50/ 616263 ~ abc

[the same as /, but forces printout as an octal constant.

Example 11

```
ldc 20[     302653
```

] the same as / but forces printout as an instruction.

Example 12

```
2664 ]      dzm ldc 35
```

Special Registers

There are several registers in DDT that hold information of interest to the user. These registers may be opened and modified; they are the only ones in DDT that may be so accessed. The names for these registers can be used like any other symbols; only remember that they are always capitalized.

A holds the C(AC) any time DDT is running.

I holds the C(IO) any time DDT is running.

F contains the address of the lowest memory location occupied by DDT's symbol table. Its contents will decrease by 2 every time a new symbol is defined.

M contains the mask used in word searches. The two registers immediately following M contain the limits of the search. When DDT is first read into memory, M contains 777777, M+1 contains 0, and M+2 contains 7777. Searching always terminates the address specified by C(F) or C(M+2), whichever is smaller. Register M immediately follows register I in DDT.

Running the Program; Breakpoints and Traps

The operations in this group allow the user to control the running of his program by starting and interrupting it whenever he wishes.

kG — this command causes machine control to go to the location specified by the address part of expression k. The C(AC) and the C(IO) are placed in the AC and IO respectively and program flag 1 and the sequence break system status are restored. If a breakpoint has been requested, it will be inserted.

The most common use of G is to start the user's program:

```
begG
```

Typing G alone is an error.

kB — This command causes DDT to insert a breakpoint at location k when control is passed to the program. At that time, the contents of k are saved, and a jda is substituted. When the user's program reaches the break location, control returns to DDT. The C(AC), the C(IO), and the status of program flag 1 and the sequence break system are saved (DDT uses only program flag 1); the address of the break location is printed out, followed by a right parenthesis, tab, and the contents of the AC. The user may now examine and modify his program, and then return control to it.

Since there is only one breakpoint, the location may be moved simply by requesting a new one.

If B is typed without an argument, any existing breakpoint is removed.

RESTRICTIONS:

The user must not place a breakpoint at an instruction which is modified during execution of the program, nor at any instruction which is used as data by the program, nor may he place a breakpoint at a subroutine call which is followed by arguments to be picked by the subroutine, since the call will be executed from DDT. After breaking at a subroutine call(on a jda or a jsp) the user may not place the next breakpoint within the subroutine being called. Note that one may successfully break on skips as well as normal subroutine jmps.

P after a breakpoint has occurred, this command allows the user to continue his program from the point of the break. The C(AC) and C(IO), program flag 1 and sequence break status are restored, and if a new breakpoint has been requested, it is inserted. The instruction at the location of the original breakpoint is executed, and the program continues. Frequently, the user will want to insert a breakpoint in a loop in his program. If so, he probably will not want a break to occur every time the program passes through that location. He may delay the break until the program has encountered the break location a specified number of times by typing an expression before the P, thus:

250P

The break will then not occur until the location has been encountered 250 times.

Example 13

```
ldc 30B
begG
ldc 30)     27305
...
...
P
(etc.)
```

In this example, a breakpoint is requested for register loc 30. The user starts his program at beg, and when the break location is reached, its address is printed out, followed by the C(AC). (Note that ldc 30 is not opened.) After any examination, the user asks the program to proceed by typing a P. The break at ldc 30 is still in effect.

Symbol Definition

Quite often, it is desireable to define new symbols for program uses, for instance, in naming the first location of a patch. DDT will accept new definitions, appending each one to the lower end of the table. Each definition requires two registers of memory. Any existing symbol may be redefined, including those in the permanent table.

New location symbols may be defined in a way similar to that used in MACRO.

Example 14

```
2663/    jda 235          her,
```

The symbol her is assigned the value 2663. DDT types a tab to indicate that the symbol has been accepted; the register remains open.

⊃ when preceded by a legal symbol, causes that symbol to be defined as the address part of the last quantity typed by DDT or the user.

Example 15

```
her/     jda 235          temɔ
her/     jda tem
```

(def) New symbols may be defined at other times using parantheses as follows:

Example 16

```
ldc 30(aeg)
```

The new symbol, aeg, is assigned the value of the expression ldc 30, where ldc must have been previously defined.

Searching

These operations disclose if a word or address is or is not present in a given section of memory. They also allow a search for certain parts of a word (for instance, all isp instructions, regardless of address). Using the mask in M and the limits in the following two registers, the user may search any part of memory except that occupied by DDT itself. Only those word positions which correspond to those containing ones in M are considered in the search. The lower limit is determined by the C(M+1), the upper limit by the C(M+2) or C(F), whichever is smaller.

kW DDT will search for registers whose contents have the value of the expression k, masked by the C(M). The location and contents of every such register are printed out. Using W without an argument is an error.

kN acts as W but searches for those registers whose contents are not equal to k.

kE causes DDT to search for those registers whose contents have an effective address equal to the expression k. For this purpose, indirect addressing chains are followed to a depth of 100. The mask is in effect here also.

Example 17

Assuming that M contains 777777, we wish to search for all registers between 500 and 1000 which contain the instruction lac 650. First we set the limits; then we request the search.

```
M 1/     0        500  (backspace)
M 2/     17777    1000 (cr)
lac 650W

501/     lac 650
704/     lac 650
```

to search the same section of memory for all isp instructions would now require a change of the mask, thus:

```
M/        777777  760000(cr)
ispW
555/      isp 1604
620/      isp 1107
727/      isp 1604
```

The mask causes only the instruction part of the words to be examined.

Miscellaneous Operation

kX — The instruction k will be executed. If it is not a jump to some part of the user's program, control will remain with DDT. Any instruction including skips and subroutine calls may be used with X.

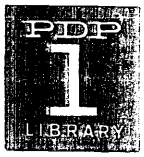
Example 18

```
claX
```

Q — This always has the value of the last previous quantity typed. Its usefulness is best illustrated by an example.

Example 19

```
ldc 30/    add list 25
```



Suppose we wished only to change the address to list + 24. Instead of typing the whole expression Q can be used:

Example 20

```
ldc 30/     add list 25    Q-1 (cr)
```

. used by itself, that is, not as part of a symbol, the period always refers to the last location opened by DDT in the normal sequence.

A common use is illustrated:

Example 21

```
ldc 20/     add 2653          add 2663 (cr)
.-1/        lac list 30
```

The contents of ldc 17 is lac list 30.

The period, like the backspace may be used at any time.

HINTS AND KINKS

E will find effective address of all instructions except jda. It is principally useful for locating incorrect instructions which are modifying the program. If a jda is suspected, try jda adrW in addition to adrE.

Breakpoints are extremely useful for investigating misbehavior of long programs. Do not try to break at program-modified instructions, or jsp's or jda's followed by program parameters to be picked up by subroutines. You may break at skip instructions whether they skip or not. Do not break at an instruction which is in the middle of a chain of indirect addressing. A breakpoint addressed by an xct will cause a break, and proceed will be from the xct.

If the operator types an undefined symbol, DDT will respond with a U. All typed input up to that point is deleted automatically.

If when attempting to type out a word as concise code, the typewriter should hang, hitting the space bar will clear it.

Symbols are stored in the following format:

word n ···concise symbol
word n+1 .its value

If it is desired to completely remove the last symbol defined, change register F to its old value +2. F must never contain an even number. When trying to determine the best symbol to fit a given value, and given two equally good symbols, DDT will pick the one last defined for its printout.

To return to typing in black, type any illegal character.

There are two ways to print a block of registers. Either set the mask to zero, set up M+1 and M+2 to enclose the area to be printed and search for any value; or, if irrelevant parts of memory happen to contain zero, merely do an N-search for zero. If you change the mask or search limits, it is well to set them back to their usual values when you are through.

DDT takes advantage of the status bits. Typeouts happen only if the typewriter status bit is on, and DDT will compute during the typeout time. When leaving DDT, the typewriter status bit will always be on.

The sequence break mode indicator is preserved upon entry to DDT, via a breakpoint or "X" return. When leaving, the sequence break system will always be cleared, and no breaks will occur as a consequence of leaving DDT.

SUMMARY OF COMMANDS

Character	Action
0-9	octal or decimal numerals and/or symbol constituents
a-z	symbol constituents
'	take as decimal
"	take as concise code
~	print as concise code
⊃	define symbol as address typed
∨	inclusive or
∧	and
<	first argument delimiter in D and Z
>	modify and open register
↑	modify and open previous register
→	print as instruction
(	set symbol definition value
)	define symbol
[	examine register as octal constant
]	examine register as instruction
-	minus
+	plus
,	define symbol equal to .
=	print as integer
.	current location
×	delete typed input
/	examine register
tab	modify an open addressed register
bk sp	modify and open next register
car ret	modify and close register
uc,lc	set case
space	plus

Character	Action
Mode Control	
S	sets the mode in which DDT types out Words to symbolic.
C	sets the mode in which DDT types out words to octal constants.
R	sets the mode in which DDT types out locations to relative (symbolic).
O	sets the mode in which DDT types out locations to octal.
U	decimal mode output.
H	Octal mode output.
Word Searches	
W	search for all occurrences of the expression preceding W.
N	search for all words not equal to the preceding expression.
E	effective address search.
Storage	
K	resets the symbol table to the initial list. Modified initial definitions are not restored.
Z	clears memory from 0 to C(F).
Breakpoint	
B	insert a breakpoint at the locations specified before the B, or remove any breakpoint if no address is specified.
P	proceed from a breakpoint.
G	go to the location specified before the G.

Character	Action
Loading Tapes	
Y	load a binary tape.
T	load the symbols only from a binary tape.
Punching	
L	listen for title punch.
D	punch data blocks.
J	punch a jump block.
Miscellaneous	
X	execute the preceding instruction.
Q	last quantity typed out by DDT.

The following symbols have as their values the addresses of certain registers in DDT whose contents are available to the user.

A	accumulator storage (for breakpoints).
I	IO storage.
M	mask used in search: M+1 and M+2 contain first and last address of the area to be searched.
F	contains the lower limit of DDT.

APPENDIX:
Extend DDT-1

Introduction

Note: The following description of the program contains only those details which differ from DDT-1. Most of the work on this version of DDT was done by William Mann of Bolt, Beranek and Newman, Cambridge. Massachusetts.

Extend DDT (DEC Debugging Tape) is a symbolic debugging program for the PDP-1 with Memory Extension Control (Type 15) and up to 16 Memory Modules (Type 12). In addition extend DDT requires a PDP-1 configuration having hardware multiply and divide routines. It occupies the upper section any Memory Module starting at register 4600. In addition it may use registers 7765-7777 in other modules. The initial symbol table contains all basic PDP-1 instructions. The program is loaded by a self-contained loader occupying registers 7751-7777 (this is the standard MIDAS absolute loader).

Operation of DDT is from the on-line typewriter. Lower case letters, numerals and period are symbol constituents; other characters, and characters typed in upper case either are control characters or are illegal. A symbol consists of a string of letters, numerals or periods, at least one of which must be a letter. Only the first six are significant. A string of digits alone is an octal number. A string of digits followed by a period is a decimal number. A period alone means the present location. This symbol syntax is the same as that of MIDAS. DDT will read binary tapes and symbol punches produced by MIDAS or MACRO.

While waiting for a type-in Extend DDT displays the current memory module it is examining in the AC lights of the console. Extend DDT itself is located in the memory module indicated by the I/O lights.

REGISTER EXAMINATION CHARACTER

| Same as /, but takes a 16 bit address and resets the current memory module indicator (displayed in the AC while waiting for type-in).

EVALUATING CHARACTER

' Types out the right 16 bits of the last quantity as internal symbol format (sqoze code as used in the MIDAS Assembler).

MODE CHANGING CHARACTERS

¯(overbar) Sets the word type-out mode to concise code characters.

R Sets the location type-out mode to relative symbolic addresses.

nR Sets the numeric type-out to radix n.

PAPER TAPE COMMANDS

Y — Reads a MIDAS or MACRO binary tape into memory between the limits contained in M+1 and M+2. Checksum errors cause a halt.

T — Read a MACRO or MIDAS symbol table merging it with the old table; redefinitions may occur. The contents of F (lower bound of symbol table) are typed upon completing reading the symbol section of the tape.
Symbols are defined only for the 4K memory module with which DDT is concerned at the time the table is read. Symbols defined for module 40_8 are defined for all modules. The module can be specified using vertical bar.

nT — For a MIDAS relocatable symbol table n is the relocation constant.

L — Punches following characters in readable format on paper tape. Terminates on:

1) TAB. Sets DDT to punch in read-in mode.

2) CAR.RET. Punches MIDAS input routine and sets DDT to punch checksummed data blocks.

3) BACKSPACE Sets DDT to punch checksummed blocks.

a<bD — Punch out registers a through b exclusive. a and b are 16 bit addresses.

The MIDAS checksummed loader occupies 7751-7777 and will load either MIDAS or MACRO tapes into a single 4K memory module. The loader will load MIDAS blocks, using 16 bit addresses, into any memory module if the machine is in extend mode.

OTHERS

Z — No argument. Clear the 4K memory block DDT is currently examining. Memory containing DDT itself is not affected.

symK — The symbol sym is removed from DDT's symbol table.

The symbols 1s - 9s may be typed in and will be used for typeout.

A breakpoint may be used at a JDA or JSP call to subroutine which is followed by a calling sequence.

The effective address search will find those locations where a JDA to the effective address exists.

SUMMARY OF CONTROL CHARACTERS

A	Accumulator storage
B	Insert a breakpoint
C	Set word print mode to constants
D	Punch data blocks
E	Effective address search
F	Lowest location in DDT
G	Go to
H	Set number output mode to octal
I	I/O storage
J	Punch start (jump) block
K	Kill defined symbols
L	Listen for title punch
M	Mask register
N	Not-word search
O	Set location print mode to numeric
P	Proceed
Q	Last quantity
R	Set location print mode to relative, or change output radix
S	Set word print mode to symbolic
T	Read symbol table
U	Set number output mode to decimal
V	Verify tape against memory
X	Execute as instruction
Y	Read binary tape
Z	Zero memory

0-9	Number and/or symbol constituents
a-z	Symbol constituents
"	Take as concise code
'	Print as sqoze code
~	Print as concise code
⊃	Define symbol as address typed
∨	Inclusive or
∧	And
<	Argument setup character
>	Modify and open addressed register
↑	Modify and open previous register
→	Print as instruction
(	Set symbol definition value
)	Define symbol
[	Examine register as constant
]	Examine register as instruction
¯(overbar)	Set word print mode to concise
|	Set module and examine register
-	Minus
+	Plus
•(center dot)	Punch present location
,	Define symbol as
=	Print as number
.	Current location
×	Delete typed input

/	Examine register
tab	Modify and open addressed register
bksp	Modify and open next register
car ret	Modify and close register
uc,lc	Set case
space	Plus

all others ignored, but respond with a ?