DIGITAL RESEARCH

Post Office Box 579, Pacific Grove, California 93950, (408) 373-3403

CP/M DYNAMIC DEBUGGING TOOL (DDT)

USER'S GUIDE

Table of Contents

Section Page

I. INTRODUCTION 1
II. DDT COMMANDS 3
1. The A (Assemble) Command 3
2. The D (Display) Command 4
3. The F (Fill) Command 4
4. The G (Go) Command 4
5. The I (Input) Command 5
6. The L (List) Command 6
7. The M (Move) Command 6
8. The R (Read) Command 6
9. The S (Set) Command 7
10. The T (Trace) Command 7
11. The U (Untrace) Command 8
12. The X (Examine) Command 8
III. IMPLEMENTATION NOTES 9
IV. AN EXAMPLE 10

CP/M Dynamic Debugging Tool (DDT)

User's Guide

I. Introduction.

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M Console Command level

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor (refer to the CP/M Interface Guide for standard memory organization), and thus resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints a sign-on message in the format

```
nnK DDT-s VER m.m
```

where nn is the memory size (which must match the CP/M system being used), s is the hardware system which is assumed, corresponding to the codes

```
D  -  Digital Research standard version
M  -  MDS version
I  -  IMSAI standard version
O  -  Omron systems
S  -  Digital Systems standard version
```

and m.m is the revision number.

Following the sign on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls

rubout	remove the last character typed
ctl-U	remove the entire line, ready for re-typing
ctl-C	system reboot

Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type

A	enter assembly language mnemonics with operands
D	display memory in hexadecimal and ASCII
F	fill memory with constant data
G	begin execution with optional breakpoints
I	set up a standard input file control block
L	list memory using assembler mnemonics
M	move a memory segment from source to destination
R	read program for subsequent testing
S	substitute memory values
T	trace program execution
U	untraced program monitoring
X	examine and optionally alter the CPU state

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. In all cases, the commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image using a SAVE command of the form

```
SAVE n filename.COM
```

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks can be determined by taking the high order byte of the top load address and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H then the number of pages is 12H, or 18 in decimal. Thus the operator could type a ctl-C during the debug run, returning to the Console Processor level, followed by

```
SAVE 18 X.COM
```

The memory image is saved as X.COM on the diskette, and can be directly executed by simply typing the name X. If further testing is required, the memory image can be recalled by typing

DDT X.COM

which reloads previously saved program from loaction 100H through page 18 (12FFH). The machine state is not a part of the COM file, and thus the program must be restarted from the beginning in order to properly test it.

II. DDT COMMANDS.

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (-) before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel (note that the rubout key should be used instead if the program is executing a T or U command). In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

1. The A (Assemble) Command. DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

As

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each sucessive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlayed by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used (refer to Section IV).

2. The D (Display) Command. The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

```
D
Ds
Ds,f
```

In the first case, memory is displayed from the current display address (initially 100H), and continues for 16 display lines. Each display line takes the form shown below

```
aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc
```

where aaaa is the display address in hexadecimal, and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are given to the right (represented by the sequence of c's), where non-graphic characters are printed as a period (.) symbol. Note that both upper and lower case alphabetics are displayed, and thus will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, except that the first line displayed is truncated so that the next line begins at an address which is a multiple of 16.

The second form of the D command shown above is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the rubout key.

3. The F (Fill) Command. The F command takes the form

```
Fs,f,c
```

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. The effect is as follows: DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f then the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4. The G (Go) Command. Program execution is started using the G comand, with up to two optional breakpoint addresses. The G command takes one ot the forms

```
G
Gs
Gs,b
```

```
Gs,b,c
G,b
G,b,c
```

The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Thus, if the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

```
*d
```

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

```
G,1234
```

and

```
G400,400
```

both produce an immediate breakpoint, without executing any instructions whatsoever.

5. The I (Input) Command. The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see the CP/M Interface Guide). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

```
Ifilename
```

or

```
Ifilename.filetype
```

If the second form is used, and the filetype is either HEX or COM, then subsequent R commands can be used to read the pure binary or hex format machine code (see the R command for further details).

6. The L (List) Command. The L command is used to list assembly language mnemonics in a particular program region. The forms are

```
L
Ls
Ls,f
```

The first command lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s, and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (see the G and T commands). Again, long typeouts can be aborted using the rubout key during the list process.

7. The M (Move) Command. The M command allows block movement of program or data areas from one location to another in memory. The form is

```
Ms,f,d
```

where s is the start address of the move, f is the final address of the move, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f then the move operation stops, otherwise the move operation is repeated.

8. The R (Read) Command. The R command is used in conjunction with the I command to read COM and HEX files from the diskette into the transient program area in preparation for the debug run. The forms are

```
R
Rb
```

where b is an optional bias address which is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (i.e., the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is taken for COM files. Note that any number of R commands can be issued following the I command to re-read the program under test, assuming the tested program does not destroy the default area at 5CH. Further, any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command

```
DDT filename.filetype
```

which initiates the DDT program is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT  PC
nnnn pppp
```

where nnnn is the next address following the loaded program, and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

9. The S (Set) Command. The S command allows memory locations to be examined and optionally altered. The form of the command is

```
Ss
```

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

10. The T (Trace) Command. The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address displayed as

*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint is occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus, CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but it must be noted that commands which use the breakpoint facility (G, T, and U) accomplish the break using a RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the rubout key to get control back to DDT during trace, rather than executing a RST 7, in order to ensure that the trace for the current instruction is completed before interruption.

11. The U (Untrace) Command. The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

12. The X (Examine) Command. The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

X
Xr

where r is one of the 8080 CPU registers

C	Carry Flag	(0/1)
Z	Zero Flag	(0/1)

M	Minus Flag	(0/1)
E	Even Parity Flag	(0/1)
I	Interdigit Carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack Pointer	(0-FFFF)
P	Program Counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format

```
CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst
```

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

III. IMPLEMENTATION NOTES.

The organization of DDT allows certain non-essential portions to be overlayed in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

IV. AN EXAMPLE.

The following example shows an edit, assemble, and debug for a simple program which reads a set of data values and determines the largest value in the set. The largest value is taken from the vector, and stored into "LARGE" at the termination of the program

tab character *rubout* *rubout echo*

```
ED SCAN.ASM
*I
        ORG     100H     L-L;START OF TRANSIENT AREA
        MVI     B,LEN    ;LENGTH OF VECTOR TO SCAN
        MVI     C,0      ;LARGER_RST VALUE SO FAR
LOOP__P_O_O_L   LXI     H,VECT  ;BASE OF VECTOR
LOOP:   MOV     A,M      ;GET VALUE
        SUB     C        ;LARGER VALUE IN C?
        JNC     NFOUND   ;JUMP IF LARGER VALUE NOT FOUND
;       NEW LARGEST VALUE, STORE IT TO C
        MOV     C,A
NFOUND: INX     H        ;TO NEXT ELEMENT
        DCR     B        ;MORE TO SCAN?
        JNZ     LOOP     ;FOR ANOTHER
;
;       END OF SCAN, STORE C
        MOV     A,C      ;GET LARGEST VALUE
        STA     LARGE
        JMP     0        ;REBOOT
;
;       TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN     EQU     $-VECT   ;LENGTH
LARGE:  DS      1        ;LARGEST VALUE ON EXIT
        END
*B0P
        ORG     100H     ;START OF TRANSIENT AREA
        MVI     B,LEN    ;LENGTH OF VECTOR TO SCAN
        MVI     C,0      ;LARGEST VALUE SO FAR
        LXI     H,VECT   ;BASE OF VECTOR
LOOP:   MOV     A,M      ;GET VALUE
        SUB     C        ;LARGER VALUE IN C?
        JNC     NFOUND   ;JUMP IF LARGER VALUE NOT FOUND
;       NEW LARGEST VALUE, STORE IT TO C
        MOV     C,A
NFOUND: INX     H        ;TO NEXT ELEMENT
        DCR     B        ;MORE TO SCAN?
        JNZ     LOOP     ;FOR ANOTHER
;
```

Rubout deletes character

Create Source Program - underlined characters typed by programmer. "↵" represents carriage return.

```
;       END OF SCAN, STORE C
        MOV     A,C     ;GET LARGEST VALUE
        STA     LARGE
        JMP     0       ;REBOOT
;
;       TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN     EQU     $-VECT  ;LENGTH
LARGE:  DS      1       ;LARGEST VALUE ON EXIT
        END
*E      <- End of Edit
```

```
ASM SCAN      Start Assembler
CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY     Assembly Complete - Look at Program Listing

TYPE SCAN.PRN
```

Code Address, Machine Code, Source Program

```
 0100                    ORG     100H    ;START OF TRANSIENT AREA
 0100 0608               MVI     B,LEN   ;LENGTH OF VECTOR TO SCAN
 0102 0E00               MVI     C,0     ;LARGEST VALUE SO FAR
 0104 211901             LXI     H,VECT  ;BASE OF VECTOR
 0107 7E        LOOP:    MOV     A,M     ;GET VALUE
 0108 91                 SUB     C       ;LARGER VALUE IN C?
 0109 D20D01             JNC     NFOUND  ;JUMP IF LARGER VALUE NOT FOUND
                ;        NEW LARGEST VALUE, STORE IT TO C
 010C 4F                 MOV     C,A
 010D 23        NFOUND:  INX     H       ;TO NEXT ELEMENT
 010E 05                 DCR     B       ;MORE TO SCAN?
 010F C20701             JNZ     LOOP    ;FOR ANOTHER
                ;
                ;        END OF SCAN, STORE C
 0112 79                 MOV     A,C     ;GET LARGEST VALUE
 0113 322101             STA     LARGE
 0116 C30000             JMP     0       ;REBOOT
                ;
                ;        TEST DATA
 0119 0200040305VECT:    DB      2,0,4,3,5,6,1,5
 0008 =         LEN      EQU     $-VECT  ;LENGTH
 0121           LARGE:   DS      1       ;LARGEST VALUE ON EXIT
 0122                    END
A>
```

Code/data listing truncated

Value of Equate

```
DDT SCAN.HEX↵          Start Debugger using hex format machine code
16K DDT VER 1.0
NEXT  PC
0121 0000
-X↵       ← last load address +1
                                                                  next instruction
                                                                  to execute at
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT  7F       PC=0
-XP↵               Examine registers before debug run
P=0000 100↵   Change PC to 100
-X↵  Look at registers again                            PC changed.
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,00
-L100↵                                                  Next instruction
                                                        to execute at PC=100
0100  MVI   B,00    }
0102  MVI   C,00    }
0104  LXI   H,0119  }
0107  MOV   A,M     }
0108  SUB   C       }
0109  JNC   010D    } Disassembled Machine
010C  MOV   C,A     } Code at 100H.
010D  INX   H       } (See Source Listing
010E  DCR   B       } for comparison)
010F  JNZ   0107    }
0112  MOV   A,C     }
-L↵
0113  STA   0121    }
0116  JMP   0000    }
0119  STAX  B       }
011A  NOP           }
011B  INR   B       } A little more
011C  INX   B       } machine code
011D  DCR   B       } (note that Program
011E  MVI   B,01    } ends at location 116
0120  DCR   B       } with a JMP to 0000)
0121  LXI   D,2200  }
0124  LXI   H,0200  }
-A116↵  enter inline assembly mode to change the JMP to 0000 into a RST 7, which
            will cause the program under test to return to DDT if 116H
0116  RST 7↵   is ever executed.
0117↵  (single carriage return stops assemble mode)
-L113↵  List code at 113H to check that RST 7 was properly inserted
0113  STA   0121    ← in place of JMP
0116  RST   07
```

```
0117  NOP
0118  NOP
0119  STAX B
011A  NOP
011B  INR  B
011C  INX  B
-

-X,   Look at registers

C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
-I,   Execute Program for one step.     initial CPU state, before , is executed
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08*0102
-I,  Trace one step again (note 08H in B)          automatic breakpoint
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00*0104
-I,  Trace again (Register C is cleared)
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119*0107
-T3, Trace three steps
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC  010D*010D
-D119,  Display memory starting at 119H.         automatic breakpoint at 10DH
0119 02 00 04 03 05 06 01   Program data
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 08 78 81   ..."!..~.W.#..x.   (Lower case x)
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00   .'...)..........
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   Data is displayed
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   in ASCII with a "."
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   in the position of
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   non-graphic
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   characters
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................
-X,            Current CPU state

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
-T5,  Trace 5 steps from Current CPU State
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR  B      Automatic
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ  0107   Breakpoint
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB  C*0109
-U5,  Trace without listing intermediate states
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC  010D*0108
-X,  CPU State at end of U5,

C0Z0M0E1I1 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB  C
```

```
-G
```
Run Program from current PC until completion (in real-time)

```
*0116
```
breakpoint at 116H, caused by executing RST 7 in machine code

```
-X
```
CPU state at end of Program

```
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST  07
-XP
```
examine and change Program counter

```
P=0116 100

-X

C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
-T10
```
Trace 10 (hexadecimal) steps

first data element — current largest value — subtract for comparison A<C

```
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0102 MVI  C,00
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI  H,0119
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC  010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB  C
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC  010D
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX  H
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR  B
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ  0107
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV  A,M*0108
-A109
0109  JC 10D
010C
-G0
```
Insert a "hot patch" into the machine code to change the JNC to JC

Program should have moved the value from A into C since A>C. Since this code was not executed, it appears that the JNC should have been a JC instruction

Stop DDT so that a version of the patched program can be saved

```
SAVE 1 SCAN.COM
```
Program resides on first Page, so save 1 page.

```
A>DDT SCAN.COM
```
Restart DDT with the saved memory image to continue testing

```
16K DDT VER 1.0
NEXT  PC
0200 0100
-L100
```
List some code

```
0100  MVI   B,08
0102  MVI   C,00
0104  LXI   H,0119
0107  MOV   A,M
0108  SUB   C
0109  JC    010D
```
Previous Patch is Present in X.COM

```
010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C
-XP
P=0100
-T10    Trace to see how patched version operates          Data is moved from A to C
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR  B
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ  0107*0107
-X                                                      breakpoint after 16 steps
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV  A,M
-G,108   Run from current PC and breakpoint at 108H
*0108
-X                      next data item
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C
-T                      Single step for a few cycles
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB  C*0109
-T
C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC   010D*010C
-X
C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV  C,A
-G     Run to completion
*0116
-X
C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST  07
-S121    look at the value of "LARGE"
0121 03   Wrong Value!
```

```
0122 00
0123 22
0124 21
0125 00
0126 02     End of the S command
0127 7E .
-L100
0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119
0107  MOV  A,M
0108  SUB  C
0109  JC   010D
010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C          Review the code
-L
0113  STA  0121
0116  RST  07
0117  NOP
0118  NOP
0119  STAX B
011A  NOP
011B  INR  B
011C  INX  B
011D  DCR  B
011E  MVI  B,01
0120  DCR  B
-XP
P=0116 100     Reset the PC
-T     Single step, and watch data values
C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI  B,08*0102
-T
C0Z1M0E1I1 A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI  C,00*0104
-T                     Count set   largest set
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0121 S=0100 P=0104 LXI  H,0119*0107
-T                                     base address of data set
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M*0108
```

```
-T
        first data item brought to A
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C*0109
-T

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D*010C
-T

C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A*010D
-T                      first data item moved to C correctly
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H*010E
-T

C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B*010F
-T

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107*0107
-T

C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M*0108
-T              second data item brought to A
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C*0109
-T              subtract destroys data value which was loaded!!!
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D*010D
-T

C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H*010E
-L100

0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119
0107  MOV  A,M
0108  SUB  C  <------ This should have been a CMP so that register A
0109  JC   010D        would not be destroyed.
010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C
-A108

0108  CMP C        hot patch at 108H changes SUB to CMP

0109

-G0   stop DDT for SAVE
```

```
SAVE 1 SCAN.COM        Save memory image
A>DDT SCAN.COM         Restart DDT
16K DDT VER 1.0
NEXT  PC
0200 0100
-XP
P=0100
-L116
0116  RST   07 }
0117  NOP      }
0118  NOP      } Look at code to see if it was properly Loaded
0119  STAX B   } (long typeout aborted with rubout)
011A  NOP      }
- (rubout)
-G,116   Run from 100H to completion
*0116
-XC   Look at Carry (accidental typo)
C1
-X   Look at CPU state
C1Z1M0E1I1 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST  07
-S121   Look at "Large" - it appears to be correct.
0121 06
0122 00
0123 22 .
-G0   Stop DDT
ED SCAN.ASM   Re-edit the source program, and make both changes
*NSUB
*0LT           ctl-z
        SUB      C         ;LARGER VALUE IN C?
*SSUB↑ZCMP↑Z0LT
        CMP      C         ;LARGER VALUE IN C?
*
        JNC      NFOUND    ;JUMP IF LARGER VALUE NOT FOUND
*SNC↑ZC↑Z0LT
        JC       NFOUND    ;JUMP IF LARGER VALUE NOT FOUND
*E
```

```
ASM SCAN.AAZ     Re-assemble, selecting source from disk A
                                 hex to disk A
CP/M ASSEMBLER - VER 1.0         Print to Z (selects no print file)

0122
002H USE FACTOR
END OF ASSEMBLY

DDT SCAN.HEX     Re-run debugger to check changes

16K DDT VER 1.0
NEXT  PC
0121 0000
-L116

0116  JMP  0000   check to ensure end is still at 116H
0119  STAX B
011A  NOP
011B  INR  B
- (rubout)

-G100,116     Go from beginning with breakpoint at end

*0116   breakpoint reached
-D121   Look at "LARGE" — correct value computed

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1  .."!..^.W.#..X.
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 .'...)..........
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

- (rubout)    aborts long typeout

-G0    stop DDT, debug session complete
```