RDT

A Self-Relocating Debugging Tool
for Heath 8-bit Computers (HDOS System)
by Patrick Swayne

Distributed by the Heath Users' Group

****

Heath Users' Group

****

The Heath Users' Group is provided by Heath Company as a service to its members for the purpose of fostering the exchange of ideas to enhance their usage of Heath Equipment. As such, little or no evaluation of the program(s) is performed. The prospective user is hereby put on notice that the program may contain faults the consequences of which Heath Company in general and the Heath Users' Group (HUG) in particular cannot be held responsible. The prospective user is, by virtue of obtaining and using these programs, assuming full risk for all consequences.

****

These (this) program(s) were developed and tested on versions 1.6 (50.05) and 2.0 (50.06) of the Heath Disk Operating System (HDOS) and believed to be free of program errors. The Heath Users' Group cannot predict that the program will perform on future versions of HDOS. However, if a version dependent program surfaces, REMark will publish all known fixes for all products. Individual attention to problems of this nature is not possible with a group as large as ours.

****

PART OF 885-1092

CONTENTS:

Introduction

RDT is a machine language debugger for Heath computers using the HDOS system. It automatically moves itself up near the top of your system's memory when you load it, and thus allows you to debug programs that occupy the normal user RAM area. It also makes an excellent patch utility and can load system files as well as regular absolute files. Its built-in mnemonic disassembler will help you check your work when patching or debugging.

Loading and Running RDT

RDT is supplied in assembly source and assembled binary forms. The binary will run in any HDOS system. It occupies 6K of space at the top of your system's memory after loading in the overlays. It is not that large, but allows a buffer space for user programs that like to put their stack just below the overlays. If you wish to reduce the space occupied to 4K, change the value assigned to the label INCLUDE to 1 and re-assemble the source. The mnemonic disassembler feature will not be available if you do this.

To run RDT, just enter RDT as an HDOS command preceded by a device name (ex. SY1:RDT) if necessary. It will sign on by printing

```
SELF-RELOCATING DEBUGGING TOOL (RDT)  Version 1.0

Enter Printer Device Name: <none>
```

If you plan to use a printer with RDT, enter the name of your device driver at this point (LP:, AT:, etc.). You can leave off the colon (:) if you wish. Type a carriage return after you enter the device name. If you are not going to use a printer, type a carriage return without entering a device name. When you hit return, RDT prints

```
STARTS XXX.XXX ENDS   YYY.YYY
]
```

It displays the starting and ending addresses (XXX.XXX and YYY.YYY) (in split octal) that it will occupy after relocation, moves itself, and prompts for your commands with an inverse bracket (]).

The syntax ; for RDT commands is a command word followed by a number or numbers (addresses or data) separated by delimiter characters (for example, DISPLAY 42200,42237). Some commands need an additional command word before the numbers are entered (for example, SEARCH WORD 42200,67377,40010). The delimiters used in RDT are the space, the comma, and the dash, and they may

be interchanged in most cases (exceptions will be noted). There are no hard-to-remember special symbols or formats used in RDT. When you type the first letter (or first two letters) of a command, it completes itself and adds a following space if an argument is required. A carriage return is usually required to terminate a command. As you enter arguments to a command, you can correct mistakes with delete or backspace before you hit return. If you want to exit a command after invoking it, type control-A.

RDT expects you to enter numbers in octal/split octal but you can change the base to hex with the BASE command (explained later) if you wish. If you are in base octal, any number greater than three digits that you enter is considered split octal, if RDT expects an address. If you enter more than six digits for an address or more than three for a byte, only the last six or three digits entered are used. Leading zeros are not required in either case. The left digit in every group of three should be 3 or less. If it is not, it will be truncated to two binary digits. For example, if you enter 666, the left 6 (which is binary 110) will be considered 2 (which is binary 10), and RDT will "see" 266. If you enter a digit that is not octal (0-7) , a question mark will be printed when you terminate the entry (type carriage return), and you will be prompted for a new command. Here are some examples:

You enter	RDT response
111222333	"sees" 222333 (or 333 if a byte is needed)
477	"sees" 077
500637	"sees" 100237
338	prints "?"

If you select base hex, all hex digits are valid (0-F). You do not need leading zeros for addresses less than 4 digits or bytes less than two digits. As above, if you enter too many digits, only the last ones entered are used. Here are some examples:

You enter	RDT response
ABCDE	"sees" BCDE (or DE if a byte is needed)
abed	"sees" ABCD (lower case is OK)
123X	prints "?"

RDT Commands

Below is a description of the commands available in RDT. Before each command's description the syntax of the command is shown using general terms in parentheses instead of actual numbers or delimiter characters. The expression (del) means a delimiter character, and (cr) means a carriage return. An example is

FILL (start addr.) (del) (end addr.) (del) (constant) (cr)

and an actual use of the FILL command might be

FILL 60000,77377,0

followed by a carriage return. In that example, the memory space from 60000 to 77377 is filled with zeros.

The ADd/subtract command:

NOTE: The letters you need to type to invoke the above command are capitalized. The rest are shown lower case. This procedure will be followed when each command is introduced.

ADD/SUBTRACT (number) (del) (number) (cr)

With this command, you can add and subtract in octal/split octal and hex. The sum and difference of the two numbers you enter are printed on the next line (the second number is subtracted from the first). If the result of* the addition is over 377377A (0FFFFH), the carry is dropped. If you subtract a number from one smaller than itself, a borrow is supplied. Here is *an* example of this command:

```
]ADD/SUBTRACT 43177,42200
105.377 000.377
```

The AScii command:

ASCII (address)(cr)

The ASCII command lets you examine memory and make changes to it using ASCII characters instead of numbers. After you enter the command, address, and carriage return, RDT prints your address on the next line, spaces over, and prints an ASCII representation of the byte at that address followed by a slash (/). Control characters are printed as normal characters preceded by a "^" (example: control-E is ^E). At this point you can type a carriage return to end the command, or a slash (/) to look at the next addresss, or just about any other ASCII character to replace what is there. The character you enter can be a control character as long as it is not one used by HDOS or RDT (^A to ^D, ^0, ^P, ^S, ^Q, or ^Z and CR, LF). RDT does not check for delete or backspace characters in the ASCII command, so you can enter those characters as well. After you enter the character, you again have the option of ending the command with a carriage return, or going on to the next address with a slash. If at any time you wish to back up to a previous address, type a dash (-). You should take care when using this command not to change an

address below 42200A (2280H) or above the starting address of RDT, displayed when it signed on. You should also make sure that what you are changing is ASCII data, not part of your program's machine code. Example: the word TEXT was misspelled as TEST.

```
]ASCII 112045

112.045  T//            type a slash to move forward
112.046  E//
112.047  S/X/           replace S with X and move
112.050  T/             type a (cr) here
]                       back to RDT
```

The Base command:

BASE HEX

BASE OCTAL

This command sets the number base for RDT input and output. After you type B to invoke the command, you only need to type H or 0 to complete it (no carriage return needed). At start up, the base is always set to octal/split octal.

The CHange command:

CHANGE (address)(cr)

The CHANGE command is used to examine or make changes to memory. Its operation is similar to the ASCII command. After you enter an address and a carriage return, RDT prints the address on the next line, spaces, and prints the byte there followed by a slash. At this point you can enter a comma or space to look at the next address, a dash to look at the previous address, a carriage return to end the command, or a number to replace the one there. If you enter a number, you can use the above procedure to move forward, back up, or end the command. In the CHANGE command, the replacement is not actually done until you enter a delimiter character or carriage return after you enter your number. (In the ASCII command, the change is done as soon as you enter the new character.) If you have typed in a number in the CHANGE command and decide that you did not want to make that change, you can exit without changing by typing an illegal character. RDT enters the character input mode while in the CHANGE command so it can check each character entered, so backspace and delete will not work. Since RDT accepts only the last byte entered, you can correct mistakes by retyping the number before you go on to the next address.

As each cell is changed, RDT checks to see that it was changed correctly. If not, a question mark is printed and

you are prompted for a new command. This feature can help you find bad memory cells, or let you know that your address is way out in left field. As with ASCII, be sure you do not change memory below 42200A (2280H) or above RDT's starting address.

Example: write a mini loop program.

```
]CHANGE 100000

100.000  127/303,          replace 127 with 303
100.001  032/000,
100.002  311/100-          let's back up
100.001  000/-             back up one more
100.000  303               type (cr) here
]                          back to RDT
```

The COmpare command:

COMPARE (start addr .) (del) (end addr.) (del) (test addr.)(cr)

This command is used to compare two blocks of memory. The first two addresses you enter are the start and end of the control block, and the third address is the start of the test block. Each byte in the test block is compared with a corresponding byte in the control block, and if there is a mis-match, the current test address is printed. Comparison continues until the end of the control block is reached, and control returns to RDT. This command is useful for checking newly programmed EPROMS.

The Display memory command:

DISPLAY (start addr.)(del)(end addr.)(cr)

With this command, you can display the contents of a block of memory of any size. After you enter the required addresses, RDT prints the starting address on the next line, spaces twice, and prints data bytes separated by spaces. When the address is divisible by 10H or 10Q (depending on the base selected), RDT prints a carriage return-line feed, the current address, and continues printing data bytes. This process continues until the end address is reached. You can abort the DISPLAY command by typing control-A.

```
]DISPLAY 30000,30037    (example of DISPLAY command)

030.000   303 014 037 041 300 377 071 353
030.010   041 100 040 166 042 076 040 066
030.020   000 043 315 216 030 302 017 030
030.030   006 000 052 076 040 004 064 176
]
```

The Examine memory command:

EXAMINE (start addr.) (del) (end addr.)(cr)

This command is a mnemonic disassembler. Its output resembles the output of the HDOS assembler (ASM). On each line of disassembly, it prints the current address, the machine code, its mnemonic representation, and a "comment" based on the ASCII value of the instruction and data. Disassembly continues until the end address is reached. If the end address falls within a two or three byte instruction, the entire instruction is printed. EXAMINE is useful for checking patches or other hand-entered code. It will not be available if you re-assemble RDT for the short (4k) version. You can abort EXAMINE with control-A. Here is a sample run of EXAMINE:

```
]EXAMINE 30000,30017

030.000   303 014 037     JMP    037014A         *C
030.003   041 300 377     LXI    H,377300A       *!@
030.006   071             DAD    SP              *9
030.007   353             XCHG                   *k
030.010   041 100 040     LXI    H,040100A       *!@
030.013   166             HLT                    *v
030.014   042 076 040     SHLD    040076A        *">
030.017   066 000         MVI    M,000Q          *6
```

If there is a RST 7 instruction in your program, it is printed by RDT as SCALL with a one byte argument, as if it were a two byte instruction.

The Fill memory command:

FILL (start addr.) (del) (end addr) (del) (constant) (cr)

This command allows you to fill a block of memory with a constant. RDT will fill your specified area with any 8-bit value you supply, and check each cell as it is filled. If a cell checks out bad, RDT prints a question mark and prompts for a new command. You can check your program's effect on an area of memory by filling it with a known constant, running your program, and then using the DISPLAY command to see if anything changed. You can also perform a simple RAM test by filling the test area with 125Q (55H) and then with 252Q (AAH) . Since the binary pattern for 125Q is 01010101 and for 252Q is 10101010, the test checks each bit in each cell, but you may need a more comprehensive test for some RAM problems. As with the CHANGE and ASCII commands, care should be taken not to write into HDOS or RDT memory.

The Go to command:

GO TO (address) (del) (breakpoint) (del) (breakpoint) (cr)

With this command, you can execute your programs and set one or two breakpoints if desired. You have the option of typing (cr) instead of the first or second delimiter character to eliminate one or both breakpoints. When the command is terminated with (cr), control is transferred to your program with all registers except PC having the values indicated by the REGISTER command (described later). The PC register will take the value of the first address entered. You can eliminate the first address if the PC register is already where you want it. Here are some examples:

]G0 TO 60000,67123,64324	start at 60000 and set breakpoints at 67123 and 64324
]G0 TO 42200	start at 42200, set no breakpoints
]G0 TO ,52100	start at existing PC value set breakpoint at 52100

If your program hits a breakpoint, control is returned to RDT, which saves the values of all registers and prints them out in the format used by the REGISTER command. The breakpoint hit is cleared along with the other one, if it was set. If your breakpoints are not hit, or you did not set any, you can still return to RDT by typing control-A. As with breakpoints, RDT will print all the registers and prompt for a new command, and clear any breakpoints that were set. In either case the PC value indicated is the next executable instruction.
WARNING: the HDOS System Programmer's Guide states that you should not single step through a call to HDOS. This restriction also applies to breakpoints. You should not put a breakpoint in HDOS, or type control-A while your program is calling HDOS. Results are unpredictable. You should try to use only breakpoints and not control-A if your program uses HDOS a lot, to ensure breaking only in the program. Even .SCIN and .SCOUT can cause trouble if interrupted.

The Hex command:

HEX (number) (cr)

The HEX command expects a number in octal/split octal regardless of the base you have set and prints the number in hex on the next line. The number is printed using 4 digits, with leading zeros if necessary. Example:

```
]HEX 360
00F0
]
```

The Input mode command:

INPUT MODE CHARACTER

INPUT MODE LINE

This command lets you set the input mode HDOS will be in when you go to a user program. Normally a program will take care of setting up HDOS for the mode of console operations it will use. But if you interrupt a program in the middle (breakpoint, etc.), return to RDT, then return to the program, it may not reset the console mode. RDT will do it for you. After you type I to invoke the INPUT MODE command, just type C to select character mode without echo, or L for line mode with echo (these are the modes usually used). RDT will set up the mode selected just before it jumps to your program. The default mode (at start up) is Line.

The Load command:

LOAD (<dev:>fname.ext)(cr)

LOAD is used to load absolute binary files from disk into memory. The default device is SY0:, and the default extension is .ABS. The program can occupy any space from 42200A (2280H) to the starting address of RDT. It can be a system or non-system file, as long as it is the binary type (RDT will not load "CODE PIC" files such as device drivers). If you try to load a non-binary file, the message FILE IS NOT MACHINE CODE is printed and the load terminated. You can, however, use RDT to patch PATCH or other absolute system files. If the file you try to load is too large for the available space, RDT prints FILE TOO BIG. If the load is successful, RDT prints the starting and ending addresses of the file. The user PC register is set to the entry point of the file, so you run the program by typing

GO TO (cr) or GO TO ,(breakpoints)(cr)

Here is an example of the LOAD command:

```
]LOAD SY1:GAME  (load GAME.ABS from SY1:)

STARTS 042.200
ENDS   051.233
]
```

The Move command:

MOVE (start addr.) (del) (end addr.) (del) (destination) (cr)

This command copies the contents of a specified block of memory to a destination address. The movement does not fix three byte instructions for execution at the new address. MOVE is useful for temporarily saving a program while its space is used by another one. The program can later be moved back into its operating area.

The Octal command:

OCTAL (number)(cr)

OCTAL is the opposite of the HEX command. It expects a number in hex, converts it to octal, and prints it on the next line. The number is printed using six digits with leading zeros if necessary.

The POrt commands:

PORT IN (port address)(cr)

PORT OUT (port address)(cr)

These commands allow you to get data from or send it to any of the 256 8080 or Z80 ports. After you invoke the command, type I to input or 0 to output, then enter the port address (0 to 377 octal or 0 to 0FF hex). Type (cr) to terminate the entry. If you are inputting (IN) , RDT will read your port and print what it reads. You can read the same port again by typing a delimiter character, or exit the command with a carriage return. If you are outputting (OUT), RDT will print an equal sign (=) and wait for a response. You can either enter a number to be sent to the port, or a (cr) to exit the command. If you enter a number, it is not actually sent until you terminate it with a delimiter or a carriage return. If you use a delimiter, RDT will send your data and prompt for more data with an equal sign. If you enter a (cr), RDT will send your data and exit the command. Here are some examples:

```
]PORT IN 360

377,              read the port again
377               type (cr) to exit
]PORT OUT 340

= 101,            send 101Q and prompt for another input
= 102             type (cr) to exit and send 102Q
]
```

The PRinter command:

PRINTER (Y OR N) Y

PRINTER (Y OR N) N

With this command, you can send listings (from DISPLAY, EXAMINE, etc.) to a printer. You must have supplied a valid device driver name when RDT signed on. When you invoke the command and answer Y, everything sent to the screen is also sent to your printer. It is sent in blocks of 256 bytes, so the listing on the screen will stop from time to time while the printer "catches up". The 256 byte buffer might be partially full when your listing finishes, so it may not all appear on the paper, but when you use the PRINTER command again and answer N, the buffer is emptied, your printed listing is finished, and the printer driver is closed. You can abort a listing while the printer is on with control-A, but do not type it while the printer is actually printing, because you will interrupt HDOS.

The Register display and modify command:

REGISTER (cr)

REGISTER (identifier) (number) (cr)

This command is used to display and/or alter user registers. If you type a carriage return after REGISTER, the values of all the (8080) registers are displayed as follows:

```
]REGISTER
A=000 F=000 B=000.000 D=000.000 H=000.000 P=000.000 S=042.200
```

Note that B indicates the BC register pair, D means DE, H means HL, P means PC (program counter) and S means SP (stack pointer). If you are in base hex, the display looks like this:

```
]REGISTER
A=00 F=00 B=0000 D=0000 H=0000 P=0000 S=2280
```

At start up, all the registers are initially set to zero, except for the stack pointer, which is set to the usual 042200A (2280H). If you want to change the value of a register, enter its identifier after the word REGISTER instead of a (cr). Use the abbreviations used by RDT in the full register display to identify registers. After you enter the identifier, the current value is displayed followed by a slash. At this point you can enter a (cr) to exit the command, or a number to replace the value currently in the register. After you enter a new value, type return to make the change and exit the command.

RDT considers the BC, DE, and HL register pairs as single 16-bit registers (B, D, and H). If you change the value of a 16 bit register, any number you enter will affect all 16 bits. If you only want to change half of the 16 bits, you may have to enter the existing other half as well. Here is an example. We want to change the B register to 111, and the E register to 222.

```
]REGISTER B 000.000/111000

]REGISTER D 000.000/222
```

Note that we had to enter 111000 to change only the B register and leave the C register at 000, but we only had to enter 222 to change the E register because it is the lower half, and the upper half was zero.

The SAve command:

SAVE (<dev :>fname.ext) (comma) (start) (del) (end) (del) (entry) (cr)

This command is used to save a block of memory on disk as an absolute binary file, using the standard HDOS binary format. The default extension is .ABS, and the default device is SY0:. After you enter the file name, you must supply the starting address, the ending address, and the entry address. You must use a comma as the delimiter character after the file name before you enter the addresses. If you are patching a file, use the REGISTER command to see what the entry address is (PC register). Here is an example:

```
]SAVE SY1:GAME,42200,51233,42200
(This saves GAME.ABS on SY1:)
```

The SEarch command:

SEARCH. BYTE (start addr.) (del) (end addr.) (del) (byte) (cr)

SEARCH WORD (start addr.) (del) (end addr.) (del) (word) (cr)

The SEARCH command will look for a number within a specified block of memory. After you invoke the command, enter B to search for a byte (8 bits) or W to search for a word (16 bits). Then enter the starting and ending addresses of the block to search, and the number to search for. RDT will print the address of the number each time it finds it. When it reaches the end of the block, it will prompt for a new command. If no addresses are printed, it means that your number was not found.

NOTE: The 8080 and Z80 microprocessors store 16 bit words with the low byte first, so if you ask RDT to search for 111222 (split octal), it looks for the byte pair 222111. This is no concern when you are searching for addresses since you can enter them normally, but if you are searching for a two byte instruction such as MVI A, 100Q, you will have to reverse the two bytes. MVI A,100Q is 076 100 in octal, but you would have to enter 100076 to look for it. Keep in mind that when you are looking for single byte instructions you may also find the instruction's numeric value in a data field. Not every 303Q in a program may be a JMP instruction. Here is an example of SEARCH:

```
]SEARCH WORD 42200,53122,40010

47325 50102
```

RDT found the word 40010A in two places, 47325A and 50102A.

The Single step command:

SINGLE STEP (start addr.) (del) (count) (cr)

This command allows you to single step through a program you are debugging. After each instruction is executed, RDT prints out the registers in the format used by the REGISTER command. It will step for as many instructions as specified in the count you supply. Remember that the count will be in the number base you are using (octal or hex). The maximum count allowed is 256 (decimal, which is specified by entering zero, or leaving off the count.) You can halt the stepping with control-S and continue it with control-Q. You can also abort the stepping with control-A. Here is an example. In memory we have the following:

```
ORG     42200A
MVI     A,111Q
LXI     B,222222A
LXI     H,333333A
STC
```

When we single step through those instructions, we get:

```
]SINGLE STEP 42200,4

 A-Ill F=000 B=000.000   D=000.000  H=000.000  P=042.201  S=042.200
 A=lll F=000 B=222.222   D=000.000  H=000.000  P=042.204  S=042.200
 A=lll F=000 B=222.222   D=000.000  H=333.333  P=042.207  S=042.200
 A=lll F=001 B=222.222   D=000.000  H=333.333  P=042.210  S=042.200
]
```

The above shows the output for a Z80 machine. For an 8080, the F (flag) register would be 002 for the first three lines and 003 for the fourth line (because bit 1 is always high).

WARNING: You should not single step through calls to HDOS. Examine the area where you are single stepping (with EXAMINE) for SCALLs if you are not familiar with it.

NOTE: RDT uses the single step hardware in the computer for single stepping. In H8*s (with an 8080), the 8080*s INTE output is used in single stepping. In H89's, the Z80 used does not produce an INTE output so it is simulated with external hardware. It does not always work. Sometimes it will not allow a single instruction to be executed so you may see two lines with the same PC register value. This does not harm your program or hinder your debugging, though, since no instruction was executed, but the attempt is counted by RDT, so your stepping may end before you wanted it to.

The Tape Load command:

TLOAD (cr)

This command loads Heath format binary cassette tapes. The PC register is set to the entry address of the program loaded. If there is a load error, a question mark is printed and RDT prompts you for a new command. Do not load tapes that start lower than 42200A (2280H), or you will wipe out HDOS's lower storage area.

The Tape Save command:

TSAVE (start addr.) (del) (end addr.) (del) (entry addr.)(cr)

This command writes a block of memory on cassette in Heath binary format. The entry address is stored on tape for later use by PAM-8, MTR-88, or RDT.

RDT Sample Run

The following is a sample work session with RDT. We are going to use it to patch the HDOS assembler (ASM Issue #104.05.00) to print the address and data columns in hex instead of octal. After starting RDT, the first thing we are going to do is load the assembler. It is on our system disk.

```
]LOAD ASM

STARTS  042.200
ENDS    075.005
```

In addition to the starting and ending addresses, we will need the entry address, so we use the REGISTER command.

```
]REGISTER P 070.055/
```

We don't want to change it, so we just hit return after the value is displayed. The routine we want to change is called $TOD (type octal digits) and is located at addresses 61234A through 61260A. Let's look at it.

```
]EXAMINE 61234,61260

061.234  305               PUSH    B           *E
061.235  006  003           MVI     B,003Q      *
061.237  247               ANA     A           *'
061.240  027               RAL                 *
061.241  027               RAL                 *
061.242  027               RAL                 *
061.243  365               PUSH    PSW         *u
061.244  346  007           ANI     007Q        *f
061.246  306  060           ADI     060Q        *F0
061.250  167               MOV     M,A         *w
061.251  043               INX     H           *#
061.252  361               POP     PSW         *q
061.253  005               DCR     B           *
061.254  302  240  061      JNZ     061240A     *B 1
061.257  301               POP     B           *A
061.260  311               RET                 *I
```

We can replace the above with another routine as long as it is not larger. Fortunately, the routine we are going to use is one byte shorter:

```
        $THD    PUSH    PSW         SAVE BYTE
                RRC                 MOVE HIGH NIBBLE DOWN
                RRC
                RRC
                RRC
                CALL    CHEX        CONVERT TO HEX
                POP     PSW         RESTORE BYTE
        CHEX    ANI     17Q         ISOLATE NIBBLE
                ADI     220Q        CONVERT TO ASCII
                DAA
                ACI     100Q
                DAA
                MOV     M,A         STORE NUMBER
                INX     H           INCR POINTER
                RET
```

There are two ways to put in a patch with RDT. The obvious way is by using the CHANGE command to put it in a byte at a time. But if it is a large patch, you can type in the source with an editor, assemble it, and then use RDT to merge it with the program to be patched. To assemble the above routine, you would have to add an ORG statement (ORG 61234A), and an END statement (END $THD). Let's say we have assembled the patch and placed the resulting .ABS file on SY1:, and it is named HEX.ABS. We will load it into memory:

```
]LOAD SYl:HEX

STARTS  061.234
ENDS    061.260
```

We just wiped out a 256 byte section of the assembler that was already in memory, since that is the minimum block size in HDOS. It is not possible to merge files by simply loading one on top of another, since they are rounded to 256 byte increments when they are stored on disk. But we can use the MOVE command in RDT to merge them. First, we move the patch we just loaded up more than 256 bytes beyond the assembler's end address:

```
]MOVE 61234,61257,101234
```

Note that we subtracted one from the end address (61257). We did it because the assembler always adds an extra byte at the end. Now look at the destination address we used (101234). When you move a block somewhere temporarily, it is always a good idea to use a destination address with the same lower half as the source address. That makes it easier to figure the end address when you move it back. Now that we've moved the patch, we can load in a fresh copy of the ASM with the load command, and merge the patch into it:

```
]LOAD ASM

STARTS 042.200
ENDS   075.005
]MOVE 101234,101257,61234
```

The patch is now in place in the assembler. If we ran it now, it would output numbers in hex, but the addresses would still have a period between the upper and lower bytes, which is not normal practice in hex notation. The code that does it is only 3 bytes long and can simply be replaced with NOP instructions, so we will do it with the CHANGE command.

The address of the code we are going to patch is 57133A.

```
]CHANGE 57133

057.133  066/0,
057.134  056/0,
057.135  043/0
```

The first patch could also have been done with the CHANGE command, but we would have had to calculate the address after the CALL instruction (CALL CHEX). It is easier to let the assembler do it for you. Now that the assembler is patched, we can save it on disk. We want to keep the octal assembler, so we will call this one ASMH.ABS and save it on SY1:.

```
]SAVE SY1:ASMH,42200,75005,70055
```

We now have two assemblers, one with hex output and one with octal. You can also give a patched file the same name as the original and save it on the original disk to replace the original (if the original is not write protected), but your disk must have as much free space as the size of the file, because RDT will write the new file before it deletes the old one. If there is not enough space, you will get an error message, and the old file is not changed. The ability to give a patched file a new name and/or save it on a different disk allows you to patch files that are write protected and locked.
We are through patching for now. If we need the assembler right away, we can run it with the GO TO command, or we can exit to HDOS by typing control-D:

```
]^D
ARE YOU SURE? Y
>
```

Debugging with RDT

RDT was designed primarily for the assembly language programmer. When you assemble a program with one of Heath's assemblers, it produces a binary file that can be loaded by RDT. The PC register will be set to the address specified by the END statement in the source. You can run the program by entering GO TO (cr), or step through it with SINGLE STEP, (count) (cr) or SINGLE STEP (cr).

Using Breakpoints

Breakpoints can be used to locate crash points and test conditional branches. If your program crashes when you run it, you can locate the crash point by placing breakpoints "early" in the program and moving them farther in until the point is found. Keep in mind that crashes can destroy everything in memory, including your program and RDT. Make sure you have a good copy of the program on disk or tape, and update it frequently.

You can determine the direction a conditional branch takes by single stepping through it, or by placing breakpoints at the condition met and condition not met destinations. You can force a conditional jump to take a particular direction by placing a breakpoint at the address of the jump instruction. When it is hit, use the REGISTER command to alter the flag register to the condition you want, then enter GO TO (cr) to take the jump.

When using breakpoints, be sure to place them at the addresses of instructions, not data. A breakpoint in a data area will never be hit, and could cause a crash. However, if you hit another breakpoint or control-A out of the program, the bad breakpoint will be cleared.

RDT and the Type-Ahead Buffer

RDT clears the console type-ahead buffer when it completes a command word so that you cannot backspace into the command while you enter arguments. You can enter a command during the delay from when you start RDT until it prompts for input, but you cannot enter arguments during that time. The buffer is used when you input arguments (address, etc.) and RDT gets them from it when you type (cr).
You can terminate the address you enter for the CHANGE command with a comma as well as with a return, and that allows you to do some tricks with the type-ahead buffer. For instance, after you enter the address and a comma, you can enter the changes you wish to make right on the command line. The example on page 5 could be entered as CHANGE 100000,303,0,100 followed by a (cr). When you hit return, it would look like this:

```
]CHANGE 100000,303,0,100    (you type this)

100.000  127/303,           (RDT prints all of this)
100.001  032/0,
100.002  311/100
]
```

You can also use this method to examine a few memory cells. Just type as many commas as the number of bytes you want to see after the address. For example:

```
]CHANGE 100000,,,   (you type this)

303/,       (RDT prints this)
000/,
100/
]
```

When you use this method to make changes or look at memory, you can use backspace or delete to correct errors in any part of the entry.

RDT Command Summary

ADd/subtract (number) (del) (number) (cr)
AScii (address)(cr)
Base Hex
Base Octal
CHange (address)(cr)
Compare (start addr.) (del) (end addr.) (del) (test addr.) (cr)
Display (start addr.) (del) (end addr.) (cr)
Examine (start addr.) (del) (end addr.) (cr)
Fill (start addr.) (del) (end addr.) (del) (constant) (cr)
Go to (address) (del) (breakpoint) (del) (breakpoint) (cr)
Hex (number)(cr)
Input mode Character
Input mode Line
Load (<dev:>fname.ext)(cr)
Move (start addr.) (del) (end addr.) (del) (destination) (cr)
Octal (number)(cr)
POrt In (port address)(cr)
POrt Out (port address)(cr)
PRinter (y or n) Y
PRinter (y or n) N
Register (cr)
Register (identifier)(number)(cr)
SAve (<dev:>fname.ext) (comma) (start) (del) (end) (del) (entry) (cr)
SEarch Byte (start addr.) (del) (end addr.) (del) (byte) (cr)
SEarch Word (start addr.) (del) (end addr.) (del) (word) (cr)
Single step (address) (del) (count) (cr)
TLoad (cr)
TSave (start addr.) (del) (end addr.) (del) (entry addr.)(cr)

In this summary, the letters needed to invoke each command are shown as capital letters and the rest as lower case letters. In actual practice, all capitals are used, and if you enter lower case letters, they are converted to capitals. The expression (del) means a comma space or dash which can usually be used interchangeably, and (cr) is a carriage return.

INDEX

The disk HUG 885-1092 contains the following:

RDT.ABS -- This is the assembled RDT program. To run it, enter the name RDT as a command in HDOS, prefixing it with a device name if necessary. Example: SY1:RDT

RDTS.ABS -- This is the short version of RDT. It does not contain the EXAMINE command (the disassembler).

RDT.ASM — This is the assembly source for RDT. All HDOS definitions used in the program are in this source.

DIS.ACM and DIST.ACM — These files make up the disassembler that is used by the EXAMINE command.