Thoughts on the SWTP Computer System

The author continues his discussion of the "monitor to end all monitors."

Peter A. Stark
PO Box 209
Mt. Kisco, NY 10549

In this article we will continue our discussion of ROM monitor design and source listings of important routines from my "monitor to end all monitors" called HUMBUG. In part 13 (June 1980) we went over the principal design features, the organization of the monitor and its cold-start procedure. Let's examine the warm-start process.

Warm-Start

MIKBUG has two entry points—E0D0 and E0E3. The entry point at E0D0 initializes everything, whereas entering at E0E3 produces only a restart of the monitor, without full initialization. HUMBUG calls these two entry points cold-start and warm-start. They are actually at FC00 and FC03 in FCROM, but jumps at E0D0 and E0E3 in E0ROM go here too.

FCROM warm-start is shown in Listing 1. As in every entry, the stack pointer is initialized to the monitor stack area at D07F to make sure that the monitor stack never destroys part of the user's stack.

The next part of Listing 1 initializes the flags in RAM. First, a zero is stored in DSTAT, P0STAT and PASTAT. DSTAT indicates whether output on the optional port D is desired; a 0 means no. Clearing P0STAT means that output on port 0 is also turned off, while clearing PASTAT disables the pause mode, which pauses output every 15 lines.

Accumulator A is then decremented to FF. This is stored in P1STAT to turn on port 1 output and in VSTAT to turn on video board output. For all of these flags, 00 means off and FF means on.

Next, the address of the warm-start entry point at FC03 is placed into location RETADD. This address is then used whenever a program is stopped with a control-S and aborted with a return. This will normally lead the program back to HUMBUG's warm-start, but any program can modify this location to cause a return to itself. For instance, if BASIC is patched to put 0103 into RETADD, then an abort will go back to BASIC instead. Once control returns back to HUMBUG, this will again be reinitialized to FC03.

The pause counter PAUCTR is then initialized to 15, so that if the pause option is enabled, output will pause every 15 lines. Again, any program could change this to some other value while it is executing.

The next part of warm-start loads 8004, the address of control port 1, into location PORADD. This is compatible with SWTBUG and enables the control port to be moved around by software just by changing the number in location A00A/A00B.

```
                  * WARMSTART INITIALIZATION
FC52 8E D07F  WARMST LDS   #$D07F    SET STACK POINTER TO MONITOR AREA
FC55 4F              CLR A
FC56 B7 D003         STA A DSTAT     TURN OFF D
FC59 B7 D000         STA A P0STAT    TURN OFF PORT 0 OUTPUT
FC5C B7 D004         STA A PASTAT    TURN OFF PAUSE FUNCTION
FC5F 4A              DEC A
FC60 B7 D001         STA A P1STAT    TURN ON CONTROL PORT OUTPUT
FC63 B7 D002         STA A VSTAT     TURN ON VIDEO BOARD OUTPUT
FC66 CE FC03         LDX   #WARMV
FC69 FF D009         STX   RETADD    INITIALIZE PAUSE-RETURN ADDRESS
FC6C 86 0F           LDA A #$0F
FC6E B7 D00D         STA A PAUCTR    INIT PAUSE LINE COUNTER
FC71 CE 8004         LDX   #$8004
FC74 FF A00A         STX   PORADD    SET CONTROL PORT ADDRESS
FC77 86 15           LDA A #$15      ACIA INPUT INITIALIZATION
FC79 B7 D00C         STA A KBDINZ
FC7C 86 11           LDA A #$11      ACIA OUTPUT INITIALIZATION
FC7E B7 D00B         STA A PTRINZ
FC81 86 13           LDA A #$13      TURN READER OFF
FC83 BD FE67         JSR   OUTCHM
FC86 4C              INC A           TURN PUNCH OFF
FC87 BD FE67         JSR   OUTCHM

                  * SEE IF OTHER ROMS REQUIRE WARM START INITIALIZATION
FC8A B6 E003         LDA A $E003     CHECK ROM-E0
FC8D 81 7E           CMP A #$7E      IS THERE A JUMP?
FC8F 26 03           BNE   HOTST     NO
FC91 BD E003         JSR   $E003     YES, GO TO IT
```

Listing 1. FCROM warm-start initialization.

```
                     * HOTST - INITIALIZATION COMPLETE. READY FOR COMMAND
FC94 8E D07F  HOTST  LDS   #$D07F    RESET STACK POINTER TO MONITOR AREA
FC97 7F A00C         CLR   PORECH    TURN ON CONTROL PORT ECHO
FC9A BD FB79         JSR   PCRLF     PRINT CR/LF
FC9D 86 2A           LDA A #'*       PRINT PROMPT
FC9F BD FDFB         JSR   OUTEEE
FCA2 BD FB93         JSR   INEEE     GET FIRST COMMAND CHARACTER
FCA5 36              PSH A           SAVE FIRST CHARACTER OF COMMAND
FCA6 BD FB93         JSR   INEEE     GET SECOND COMMAND CHARACTER
FCA9 16              TAB             MOVE SECOND TO B
FCAA BD FB6B         JSR   OUTS
FCAD 32              PUL A           RESTORE FIRST COMMAND
FCAE 36              PSH A           AND SAVE IT ONCE MORE
              * CHECK COMMAND
FCAF 81 4A           CMP A #'J       CHECK FOR JU(MP)
FCB1 26 07           BNE   NOTJU
FCB3 C1 55           CMP B #'U
FCB5 26 03           BNE   NOTJU
FCB7 7E FB6F         JMP   JUMP      EXECUTE JUMP COMMAND
FCBA 81 4D    NOTJU  CMP A #'M       CHECK FOR ME(MORY CHANGE)
FCBC 26 04           BNE   HOTEND
FCBE C1 45           CMP B #'E
FCC0 27 0F           BEQ   CHANGE    EXECUTE CHANGE COMMAND
              * SEE IF OTHER ROMS HAVE COMMANDS
FCC2 B6 E006  HOTEND LDA A $E006
FCC5 81 7E           CMP A #$7E      IS THERE A JUMP
FCC7 27 02           BEQ   GOJUMP
FCC9 20 C9           BRA   HOTST     AND LOOK FOR MORE
FCCB 32       GOJUMP PUL A           GET FIRST CHARACTER
FCCC BD E006         JSR   $E006     AND JUMP TO NEXT ROM
FCCF 20 C3    GOHOT1 BRA   HOTST     THEN DO MORE COMMANDS
```

Listing 2. FCROM hot-start initialization.

The next four lines overcome the following problem in SWTBUG: each time SWTBUG inputs via INEEE, it initializes the ACIA to use only one stop bit; when doing an output via OUTEEE, it initializes the ACIA to output two stop bits. Unfortunately, if the user has previously initialized the ACIA in some other way, then this will reinitialize the port and destroy what has been done. This has been a particular problem in controlling the reader control line in the interface. HUMBUG does the same thing but puts the two initialization constants into locations KBDINZ and PTRINZ during warm-start and reads them out of these two locations in INEEE and OUTEEE, respectively.

Changing these locations before use allows complete user control over the ACIA. For instance, by changing the two constants from 15 and 11 to 16 and 12, the ACIA will change its baud rate to a quarter of its previous value. Since I have both a 1200 baud terminal and a 300 baud keyboard on the same port, I can change the baud rate from 1200 to 300 and back from the keyboard.

The last four steps of warm-start output $13 and $14 to the port to turn off the reader and punch, if they are controlled by ASCII codes.

Once all FCROM initialization is completed, the program tests to see whether there is a ROM at E000, and a JSR is made to it if it is there. As it turns out, neither E0ROM nor E4ROM require any, so they return to FCROM with an RTS. Their handling of warm-start is identical with that of cold-start, so I'm not including those listings here.

Hot-Start

Hot-start is my name for the command loop that looks for monitor commands and goes to execute them. The FCROM hot-start routine is shown in Listing 2.

As usual, the stack pointer is first reset to the monitor stack area at D07F. Then location PORECH is zeroed (it is used by INEEE to determine whether to echo keyboard input). In this one case, 00 means that echo is on and FF means that it is off. This is the opposite of the other flags, but is necessary to be compatible with SWTBUG. The program then jumps to a carriage-return/line-feed subroutine and outputs the prompt character (*). It then inputs the two-letter command, puts the two letters into the two accumulators and checks them.

Since FCROM has only two commands, it is much faster to check the letters directly than to look them up in a command table. If the command is JU, then we jump to routine JUMP; if the command is ME, then we jump to routine CHANGE.

```
E006 7E E20F  COMMDV JMP   COMAND    COMMAND ENTRY POINT

E20F 36       COMAND PSH A           SAVE FIRST CHARACTER
E210 CE E240         LDX   #COMTAB-4 GET ADDR OF COMMAND TABLE
E213 08       LOOKUP INX
E214 08              INX
E215 08              INX
E216 08              INX
E217 8C E278         CPX   #TABEND   END OF TABLE?
E21A 27 10           BEQ   COMEND    YES
E21C A1 00           CMP A 0,X       NO, CHECK FIRST CHARACTER
E21E 26 F3           BNE   LOOKUP    WRONG
E220 E1 01           CMP B 1,X       CHECK SECOND CHARACTER
E222 26 EF           BNE   LOOKUP    WRONG, SKIP TO NEXT
E224 EE 02           LDX   2,X       GET ADDRESS IF OK
E226 32              PUL A           RESTORE STACK
E227 BD FC30         JSR   OUTS      PRINT A SPACE
E22A 6E 00           JMP   0,X       JUMP TO APPROPRIATE COMMAND ROUTINE
              * COMMAND NOT FOUND; SEE IF OTHER ROMS HAVE COMMANDS
E22C B6 E406  COMEND LDA A $E406     CHECK NEXT ROM
E22F 81 7E           CMP A #$7E      IS THERE A JUMP
E231 27 09           BEQ   COMND4
E233 B6 E806         LDA A $E806     CHECK ROM AFTER THAT
E236 81 7E           CMP A #$7E      IS THERE A JUMP
E238 27 06           BEQ   COMND8
E23A 32              PUL A           NO MORE ROMS; FIX UP STACK
E23B 39              RTS             AND RETURN TO FCROM
E23C 32       COMND4 PUL A           NEXT ROM EXISTS; RESTORE FIRST CHARACTER
E23D 7E E406         JMP   $E406     GO TO IT
E240 32       COMND8 PUL A           SECOND ROM EXISTS; RESTORE FIRST CHARACTER
E241 7E E806         JMP   $E806     GO TO IT

              * COMMAND TABLE

E244 4C       COMTAB FCC   'LO' LOAD MIKBUG TAPE
E246 E0 0C           FDB   LOAD
E248 50              FCC   'PU' PUNCH MIKBUG TAPE
E24A E1 10           FDB   PUNCH
E24C 46              FCC   'FD' FLEX DISK BOOT
E24E E2 DE           FDB   FLBOOT
E250 45              FCC   'EN' END OF TAPE FORMATTING
E252 E1 F9           FDB   PNCHS9
E254 47              FCC   'GO' GO TO USER PROGRAM VIA A048/9
E256 E1 A8           FDB   GOTO
E258 43              FCC   'CL' CLEAR SCREEN
E25A E0 58           FDB   CLEAR
E25C 46              FCC   'FI' FIND BYTES COMMAND
E25E E3 05           FDB   FIND
E260 48              FCC   'HD' HEX DUMP ROUTINE
E262 E0 D3           FDB   HEXDMP
E264 46              FCC   'FM' FILL MEMORY
E266 E3 81           FDB   FILL
E268 50              FCC   'PD' PERCOM DISK DOS-PLUS
E26A C0 00           FDB   MDOSPL
E26C 43              FCC   'CS' TWO-BYTE CHECKSUM
E26E E3 9A           FDB   SUM
E270 4D              FCC   'MT' MEMORY TEST
E272 E3 DA           FDB   ROBIT
E274 50              FCC   'PC' PRINT A048/A049
E276 E0 9F           FDB   PRNT48
   (E278)     TABEND EQU   *
```

Listing 3. E0ROM command lookup.

However, if the command is not recognized, then FCROM checks to see whether there is another ROM at E000. If so, it executes a JSR to the hot-start entry point of that ROM, carrying the two-letter command in accumulators A and B. If the command is not recognized by the other ROMs, they execute an RTS to return to the last line in Listing 2, which will return back to the beginning of the hot-start command loop. In this way, the command routine of all other ROMs (except FCROM) can be called as a subroutine by user programs.

Each of the other ROMs has more than two possible commands, so to more efficiently recognize the two-letter command, we should look it up in a table. Listing 3 shows how E0ROM does this; all other ROMs are done the same way.

In each case, there is a command table, COMTAB, which lists each two-letter command, followed by the address of the routine that executes that command. The program simply looks through that table—one entry at a time—and tries to match up the two letters in the A and B accumulators against the command entry in the table. If a match is found, then the program executes an indexed jump to the address listed in the table.

If no match is found, the routine checks whether there are any other ROMs. For instance, E0ROM checks for ROMs at E400 and E800, etc. If any are found, the program jumps to their command entry point; if not, then an RTS returns the pro-

```
                 • JUMP TO USER PROGRAM COMMAND
FD6F 8D A5      JUMP   BSR   BADDR      GET ADDRESS
FD71 8E A07F           LDS   #$A07F     INITIALIZE STACK TO USER AREA
FD74 AD 00             JSR   0,X        JUMP TO USER PROGRAM
FD76 7E FC52           JMP   WARMST     ON RTS, RETURN TO WARM START
```

Listing 4. JU command.

gram to FCROM without doing anything.

Back to FCROM

FCROM has all of the MIKBUG-compatible routines such as INHEX, BADDR and OUT2HS, as well as routines to change memory and jump to a user program. All of these are identical to MIKBUG (except that references to a PIA on port 1 have been changed to an ACIA). Only three routines—the jump-to-user-program routine, INEEE and OUTEEE—are substantially different.

Jump to User Program

As shown in Listing 4, the routine JUMP consists of just four steps. First, routine BADDR is called to get the jump address. Then the stack pointer is set to A07F, the user stack area, and JSR is executed to the address that has been input by BADDR and held in the index register.

This instruction is JSR rather than JMP so that subroutines can be executed and tested. A return to warm-start follows JSR so that when a subroutine returns to the monitor, it will neatly reenter the monitor.

Notice how a completely different user stack area—separate from the monitor stack at D07F—is set up. No locations in the scratchpad RAM at A000–A07F are used other than what SWTBUG used. The user program can thus redefine the stack area to a location compatible with SWTBUG or MIKBUG. On the other hand, if the user program does not redefine the stack, then a large area of the scratchpad is available for stack use.

INEEE

The new INEEE is shown in Listing 5. The last dozen lines of INEEE are the heart of the routine. INCH8 checks the ACIA on the control port for a character, waits for it if none is there and then returns to the calling routine with the character in the A accumulator. Note how PORADD is used to define the port address, while KBDINZ is used to configure it just before the input.

INCH8 returns a full 8-bit character, including the parity bit, which is required for some routines. However, most of the time, we want to strip off the parity bit and make the first bit of each character a 0. This is done by INCH7, which ANDs the character from INCH8 with a mask of $7F (a binary 01111111) to remove the first bit.

INEEE starts with saving the B accumulator and index register and then gets the character from INCH7. If it is not a control-S (or an ASCII $13), then it tests PORECH to see whether echoing is desired and prints it back via OUTEEE if PORECH is equal to 00.

If a control-S was detected, INEEE jumps to GOTCS and then to GETCMD to get the next character and perform the indicated command.

GETCMD starts by ringing the bell to signal that it is in control and then gets the next character

```
                 • INEEE - CHARACTER INPUT ROUTINE
FD93 37         INEEE  PSH B            SAVE B
FD94 FF D005           STX   INEEXR     SAVE REGISTERS
FD97 8D 4F      INRPT  BSR   INCH7      GET INPUT CHARACTER
FD99 81 13             CMP A #$13       IS IT CONTROL-S?
FD9B 27 0C             BEQ   GOTCS      YES
FD9D 7D A00C           TST   PORECH     NO; ECHO ON?
FDA0 26 02             BNE   INEXIT     NO, SO EXIT
FDA2 8D 59             BSR   OUTEEE     YES, SO ECHO
FDA4 FE D005    INEXIT LDX   INEEXR     RESTORE REGISTERS
FDA7 33                PUL B
FDA8 39                RTS              AND RETURN

                 • CONTROL-S DETECTED. GET AND INTERPRET COMMAND
FDA9 8D 02      GOTCS  BSR   GETCMD     DO COMMAND
FDAB 20 EA             BRA   INRPT

                 • SUBROUTINE TO GET AND DO COMMAND
FDAD 86 07      GETCMD LDA A #$07
FDAF BD FE67           JSR   OUTCHM     ECHO CONTROL-G (BELL) ON CTL PORT
FDB2 8D 34             BSR   INCH7      GET SECOND CHARACTER OF CMD
FDB4 81 30             CMP A #'0        PORT 0 COMMAND?
FDB6 26 04             BNE   NOT0       NO
FDB8 73 D000           COM   P0STAT     YES; FLIP PORT 0 STATUS
FDBB 39                RTS              AND RETURN
FDBC 81 31      NOT0   CMP A #'1        PORT 1 COMMAND?
FDBE 26 04             BNE   NOT1       NO
FDC0 73 D001           COM   P1STAT     YES; FLIP PORT 1 STATUS
FDC3 39                RTS
FDC4 81 44      NOT1   CMP A #'D        PORT D COMMAND?
FDC6 26 04             BNE   NOTD       NO
FDC8 73 D003           COM   DSTAT      YES; FLIP PORT D STATUS
FDCB 39                RTS
FDCC 81 50      NOTD   CMP A #'P        PAUSE COMMAND?
FDCE 26 09             BNE   NOTP       NO
FDD0 73 D004           COM   PASTAT     YES; FLIP PAUSE STATUS
FDD3 86 0F             LDA A #$F
FDD5 B7 D00B           STA A PAUCTR     RESET PAUSE LINE CNTR
FDD8 39                RTS              AND RETURN
FDD9 81 0D      NOTP   CMP A #$0D       CR COMMAND TO QUIT?
FDDB 26 0A             BNE   NOTCR      NO
FDDD 33                PUL B            YES; FIX UP STACK
FDDE 33                PUL B
FDDF 33         QUIT   PUL B            RESTORE B
FDE0 32                PUL A
FDE1 32                PUL A            FIX STACK SOME MORE
FDE2 FE D009           LDX   RETADD     GET RETURN ADDRESS
FDE5 6E 00             JMP   0,X        AND RETURN
FDE7 39         NOTCR  RTS              RETURN WITHOUT DOING ANYTHING OTHERWISE

                 • ACTUAL CONTROL PORT INPUT ROUTINES
FDE8 8D 03      INCH7  BSR   INCH8      GET 7-BIT CHARACTER
FDEA 84 7F             AND A #$7F       MASK OUT PARITY
FDEC 39                RTS
FDED FE A00A    INCH8  LDX   PORADD     GET 8-BIT CHARACTER
FDF0 B6 D00C           LDA A KBDINZ     CONFIGURE ACIA
FDF3 A7 00             STA A 0,X
FDF5 A6 00      ACIAIN LDA A 0,X
FDF7 47                ASR A
FDF8 24 FB             BCC   ACIAIN     WAIT FOR CHARACTER
FDFA A6 01             LDA A 1,X        GET IT
FDFC 39                RTS              AND RETURN
```

Listing 5. INEEE routine.

via INCH7. If this character is either 0, 1, D or P, then it toggles P0STAT, P1STAT, DSTAT or PASTAT, respectively. Complementing is used, so that these flags will go from 00 to FF and back to 00 each time they are flipped. These four flags control output on port 0, port 1, optional port D and the pause mode. On a valid command, GETCMD ends with RTS, which goes back to GOTCS, which, in turn, leads back to INRPT to read the next character. Thus, the character following the control-S is neither echoed nor returned to the calling program.

On the other hand, if the character following the control-S was a carriage return, then the GETCMD fetches the return address from RETADD and jumps to it, thereby aborting whatever program had called it.

OUTEEE

Listing 6 shows the revised OUTEEE. This routine begins by saving some of the registers and then checks the control port for the presence of any character at the keyboard. If it detects a control-S, then it goes to GETCMD to execute it (as I described previously). Any other condition leads to NOTEST.

The next few steps check PASTAT to see whether the pause mode is on. If it is, then a series of decisions has to be made. If the current character is a clear-screen character (hex 10 or control-P in SWTP programs and terminals), then the pause line counter must be reset to allow a full screen after the clear-screen command is executed. Next, if the current character is a carriage return, then the line counter PAUCTR is decremented and checked to see if it is time to pause. If it is, then the program resets the pause line counter back to 15 (hex 0F) and waits for any character from the keyboard. If this character is another carriage return, then the program aborts; otherwise, it continues.

After all pause processing is over, OUTEEE checks each of the port flags (P0STAT, P1STAT, VSTAT and DSTAT). If any of these are nonzero, then the current character is output via that port. Note how VSTAT controls video board output. Although there is no monitor routine to control this flag (other than its being initialized), VSTAT allows other programs to turn off the video board—instead of straight echoing of OUTEEE

```
                    * OUTEEE - CHARACTER OUTPUT ROUTINE
FDFD 37             OUTEEE PSH B                SAVE B
FDFE FF B007               STX   OUTEXR         SAVE XR
FE01 36                    PSH A                SAVE CHARACTER
FE02 FE A00A               LDX   PORADD
FE05 A6 00                 LDA A 0,X            CHECK CONTROL PORT
FE07 47                    ASR A
FE08 24 0A                 BCC   NOTEST         NO CHARACTER
FE0A A6 01                 LDA A 1,X            CHARACTER; GET IT
FE0C 84 7F                 AND A #$7F           MASK OUT PARITY BIT
FE0E 81 13                 CMP A #$13           IS IT CONTROL-S?
FE10 26 02                 BNE   NOTEST         NO
FE12 8D 99                 BSR   GETCMD         YES; GET COMMAND AND DO IT
FE14 32             NOTEST PUL A                FINISHED TESTING FOR COMMAND
                    * CHECK FOR PAUSE
FE15 7D B004               TST   PASTAT         PAUSE STATUS ON?
FE18 27 24                 BEQ   NOPAUS         NO
FE1A 81 10                 CMP A #$10           CLEAR SCREEN?
FE1C 26 07                 BNE   NOCLR          NO
FE1E 86 0F                 LDA A #$0F           YES; RESET PAUSE COUNTER
FE20 B7 B00D               STA A PAUCTR
FE23 20 19                 BRA   NOPAUS
FE25 81 0D          NOCLR  CMP A #$0D           CR?
FE27 26 15                 BNE   NOPAUS         ONLY PAUSE AT END OF LINE
FE29 7A B00D               DEC   PAUCTR         DECR PAUSE LINE CNTR
FE2C 26 10                 BNE   NOPAUS         AND CHECK IT
FE2E 86 0F                 LDA A #$0F           MUST PAUSE. RESET CNTR
FE30 B7 B00D               STA A PAUCTR
FE33 8D B3                 BSR   INCH7          WAIT FOR RESTART CHAR
FE35 81 0D                 CMP A #$0D           QUIT IF IT'S A CR
FE37 26 03                 BNE   PCONT
FE39 7E FDBF               JMP   QUIT
FE3C 86 0D          PCONT  LDA A #$0D           CONTINUE WITH CR
FE3E 7D B000        NOPAUS TST   P0STAT         PRINT ON PORT 0?
FE41 27 02                 BEQ   NOTPT0         NO
FE43 8D 1D                 BSR   OUTCH0         YES
FE45 7D B001        NOTPT0 TST   P1STAT         PRINT ON CONTROL PORT?
FE48 27 02                 BEQ   NOTPTM         NO
FE4A 8D 1B                 BSR   OUTCHM         YES
FE4C 7D B002        NOTPTM TST   VSTAT          OUTPUT VIA VIDEO BOARD?
FE4F 27 04                 BEQ   NOTVID         NO
FE51 36                    PSH A                YES
FE52 8D 24                 BSR   OUTCHV         OUTPUT ON VIDEO
FE54 32                    PUL A
FE55 7D B003        NOTVID TST   DSTAT          PRINT ON D?
FE58 27 03                 BEQ   NOTDUR         NO
FE5A BD EC0C               JSR   OUTCHD         YES
FE5D FE B007        NOTDUR LDX   OUTEXR         RELOAD XR AND B
FE60 33                    PUL B
FE61 39                    RTS

                    * OUTPUT ON PORT 0
FE62 CE 8000        OUTCH0 LDX   #$8000         OUTPUT TO PORT 0
FE65 20 03                 BRA   OUTCHE

                    * OUTPUT ON CONTROL PORT
FE67 FE A00A        OUTCHM LDX   PORADD
FE6A F6 B00D        OUTCHE LDA B PTRINZ         ACIA INITIALIZATION
FE6D E7 00                 STA B 0,X            INITIALIZE FOR 8 BITS, 2 SB
FE6F E6 00          OUTM2  LDA B 0,X            WAIT UNTIL READY
FE71 57                    ASR B
FE72 57                    ASR B
FE73 24 FA                 BCC   OUTM2
FE75 A7 01                 STA A 1,X            PRINT IT
FE77 39                    RTS
```

Listing 6. OUTEEE routine.

```
E009 7E E278  FRMTOV JMP   FROMTO    FROMT-TO SUBROUTINE ENTRY

              * FROMTO SUBROUTINE - INITIALIZE BEGA AND ENDA ADDRESSES

E278 CE E04A  FROMTO LDX   #FROMST
E27B BD FC12         JSR   PDATA     PRINT "FROM "
E27E BD FC09         JSR   INEEE     GET CHARACTER
E281 81 0D           CMP A #$0D      IS IT A CR?
E283 26 03           BNE   GETFT     CONTINUE IF NOT
E285 7E FC0F         JMP   CRLF      ON CR, DO CRLF AND RETURN
E288 80 30    GETFT  SUB A #$30      CONTINUE .. CHECK FOR DIGIT
E28A 2B 2F           BMI   GONOTS    NOT HEX
E28C 81 09           CMP A #9
E28E 2F 0A           BLE   GOTONE
E290 81 11           CMP A #$11
E292 2B 27           BMI   GONOTS    NOT HEX
E294 81 16           CMP A #$16
E296 2E 23           BGT   GONOTS    NOT HEX
E298 80 07           SUB A #7        CONVERT A-F TO NUMBER
E29A 48       GOTONE ASL A           GOT FIRST DIGIT
E29B 48              ASL A
E29C 48              ASL A
E29D 48              ASL A
E29E 16              TAB             TEMP SAVE IT
E29F BD FC18         JSR   INHEX     GET SECOND DIGIT
E2A2 1B              ABA             COMBINE THEM
E2A3 B7 A002         STA A BEGA      STORE LEFT TWO DIGITS
E2A6 BD FC1B         JSR   BYTE      GET NEXT TWO
E2A9 B7 A003         STA A BEGA+1    STORE RIGHT TWO AS FROM ADDRESS
E2AC CE E06E         LDX   #TOSTR
E2AF BD FC12         JSR   PDATA     PRINT "TO "
E2B2 BD FC1E         JSR   BADDR     GET TO ADDRESS
E2B5 FF A004         STX   ENDA      STORE IT
E2B8 7E FC30         JMP   OUTS
E2BB 31       GONOTS INS             INVALID DIGIT; INCREMENT SP TO BYPASS
E2BC 31              INS             ...THE CALLING ROUTINE AND RETURN ONE LEVEL
E2BD 39              RTS             ...ABOVE (TO HOTSTART)
```

Listing 7. FROMTO routine.

```
              * 'HD' HEX DUMP COMMAND

E0D3 BD E278  HEXDMP JSR   FROMTO
E0D6 FE A002         LDX   BEGA      GET STARTING ADDRESS
E0D9 FF B020         STX   SAVEX     SAVE DUPLICATE
E0DC 20 08           BRA   HEXCON    AND SKIP OVER NEXT VECTOR

              * FREE TO E0E2 (5)

              * WARMST WARM START
  (E0E3)             ORG   $E0E3
E0E3 7E FC03  E0E3   JMP   WARMST    VECTOR TO FC ROM

              * CONTINUATION OF HEX DUMP
E0E6 B6 B021  HEXCON LDA A SAVEX+1
E0E9 84 F0           AND A #$F0      ROUND DOWN TO NEXT 0
E0EB B7 B021         STA A SAVEX+1
E0EE BD FC0F  HEX    JSR   CRLF
E0F1 CE B020         LDX   #SAVEX    GET LOCATION OF STARTING ADDR
E0F4 BD FC2B         JSR   OUT4HS    PRINT IT
E0F7 BD FC30         JSR   OUTS      EXTRA SPACE
E0FA C6 10           LDA B #16       SET COUNTER TO 16
E0FC FE B020         LDX   SAVEX
E0FF BD FC2A  HEX1   JSR   OUT2HS    PRINT NEXT BYTE
E102 09              DEX             BACKUP POINTER
E103 BC A004         CPX   ENDA      LAST ADDRESS?
E106 26 01           BNE   HEX2      CONTINUE IF NOT
E108 39              RTS             OTHERWISE END
E109 08       HEX2   INX             RESTORE POINTER
E10A 5A              DEC B           DECREMENT COUNTER
E10B 26 F2           BNE   HEX1      CONTINUE LINE IF NOT FINISHED
E10D FF B020         STX   SAVEX     SAVE CURRENT POINTER
E110 20 DC           BRA   HEX       GET READY FOR NEXT LINE
```

Listing 8. Hex dump routine.

output—whenever memory-mapped output or graphics are desired.

OUTCH0 and OUTCHM are two character output routines that output to port 0 and the control port, respectively. The actual port address used depends simply on the address loaded into the index register.

FROMTO Subroutine

MIKBUG's P, or Punch, routine used locations BEGA (A002-3) and ENDA (A004-5) to hold the beginning and ending addresses of memory to be punched to tape. In a similar way, HUMBUG uses these same two locations, not just for the PU command, but for other commands as well. The FROMTO subroutine in Listing 7 is used by these commands to ask for these two addresses from the control port.

This routine is easy to understand but has two special operating modes. After INEEE is called for the first digit of the "from" address in the third line, that character is checked for a carriage-return character. If a CR is detected, then the routine returns to the calling program without changing BEGA and ENDA. Next, even if this character is not a return, if it is not a valid hex digit, then the subroutine returns to the program one level above the calling program; that is, it returns to the program that called the program that called FROMTO. In the case of these monitor routines, this will always mean a return to the hot-start location.

Although FROMTO is buried in E0ROM, there is an entry vector to it in location E009, so that its calling address does not change even if E0ROM is modified.

Monitor Commands

Except for the ME and JU commands in FCROM, all other commands are subroutines that

```
E305 CE E0AD  FIND   LDX    #MANYST
E308 BD FC12         JSR    PDATA     ASK "HOW MANY BYTES"
E30B BD FC09         JSR    INEEE     GET NUMBER
E30E 80 30           SUB A #$30       CONVERT FROM ASCII
E310 27 6C           BEQ    FIND5     IF = 0
E312 2B 6A           BMI    FIND5     IF LESS THAN 0
E314 81 03           CMP A #$3
E316 2E 66           BGT    FIND5     IF GREATER THAN 3
E318 B7 D025         STA A FINDNO     STORE NUMBER OF BYTES
E31B BD FC30         JSR    OUTS
E31E CE E1EA         LDX    #WHATST
E321 BD FC12         JSR    PDATA     ASK "WHAT BYTES"
E324 F6 D025         LDA B FINDNO     GET NUMBER
E327 CE D022         LDX    #WHAT
E32A 37       FIENTR PSH B
E32B BD FC1B         JSR    BYTE      ENTER A BYTE
E32E 33              PUL B            RESTORE COUNTER
E32F A7 00           STA A 0,X        STORE IT
E331 08              INX
E332 5A              DEC B
E333 26 F5           BNE    FIENTR    ENTER MORE, IF NEEDED
E335 BD E278         JSR    FROMTO    GET BEGA AND ENDA
E338 FE A002         LDX    BEGA      GET READY TO LOOK
E33B F6 D025  FIND1  LDA B FINDNO     MAIN FIND LOOP
E33E A6 00           LDA A 0,X        GET FIRST BYTE
E340 B1 D022         CMP A WHAT
E343 26 31           BNE    FIND4     WRONG BYTE
E345 5A              DEC B
E346 27 11           BEQ    FIND2     FOUND ONE CORRECT BYTE
E348 A6 01           LDA A 1,X        GET SECOND BYTE
E34A B1 D023         CMP A WHAT+1
E34D 26 27           BNE    FIND4     WRONG
E34F 5A              DEC B
E350 27 07           BEQ    FIND2     FOUND TWO CORRECT BYTES
E352 A6 02           LDA A 2,X        GET THIRD BYTE
E354 B1 D024         CMP A WHAT+2
E357 26 1D           BNE    FIND4     WRONG BYTE
E359 FF D020  FIND2  STX    SAVEX     FOUND CORRECT BYTES
E35C 8D 20           BSR    FIND5     PRINT CRLF VIA VECTOR AT FIND5
E35E CE D020         LDX    #SAVEX    POINT TO ADDRESS WHERE FOUND
E361 BD FC2D         JSR    OUT4HS    PRINT IT
E364 BD FC30         JSR    OUTS      ONE MORE SPACE
E367 FE D020         LDX    SAVEX
E36A 09              DEX              BACKUP ONE BYTE
E36B C6 04           LDA B #4         READY TO PRINT FOUR BYTES
E36D BD FC2A  FIND3  JSR    OUT2HS    PRINT BYTE
E370 5A              DEC B
E371 26 FA           BNE    FIND3     PRINT FOUR BYTES
E373 FE D020         LDX    SAVEX     RESTORE INDEX REGISTER
E376 BC A004  FIND4  CPX    ENDA      SEE IF DONE
E379 27 03           BEQ    FIND5     YES
E37B 08              INX              NO
E37C 20 BD           BRA    FIND1     KEEP LOOKING
E37E 7E FC0F  FIND5  JMP    CRLF      DO LAST CRLF AND RETURN TO FCROM WHEN DONE
```

Listing 9. Find routine.

normally return to the hot-start entry point and are also user callable. Some of them are to the point, such as PU and LO, which are similar to MIKBUG's P and L routines, except for the use of an ACIA instead of a PIA.

Let's look at the other routines.

Listing 8 shows the HEXDMP routine. As with several other routines in E0ROM, this one is sandwiched between MIKBUG-compatible calls. In this case, the monitor restart vector at E0E3 splits it in two parts.

This listing shows how FROMTO is called at the beginning to allow beginning and ending addresses to be specified. The beginning address is moved from BEGA to temporary location SAVEX, but the second byte of that address is ANDed with $F0 to force the last digit to always be 0. Thus, the 16 bytes printed on a line will always start with a location ending with 0.

Subroutines to perform the FI (find), FM (fill memory), CS (checksum memory), AI (ASCII input), AO (ASCII output) and MO (move memory) commands are shown in Listings 9 through 14, respectively. Most of these are easily understandable.

Note how the move memory routine checks the old and new addresses to see whether memory contents are being moved to lower or higher addresses. This is necessary to avoid erasing data if the new locations overlap the old locations. If the memory contents are being moved to lower addresses, then the move starts with the lower address. But if the move is to higher addresses, then the highest locations are moved first. In this way, even if the old and new locations overlap, data will be moved out of the way before it is written over.

The routine for the DE, or "DEsemble," command is shown in Listing 15. It consists of a short calling program named DESEMB and a subroutine called PRNTOP, which does most of the work.

DESEMB begins by calling

```
                      * "FM" COMMAND - FILL MEMORY WITH CONSTANT

E381 BD E270  FILL   JSR   FROMTO    GET FROM-TO ADDRESSES
E384 CE E1C5         LDX   #WITHST
E387 BD FC12         JSR   PDATA     ASK FOR DATA
E38A BD FC1B         JSR   BYTE
E38D FE A002         LDX   BEGA      GET STARTING ADDRESS
E390 09              DEX
E391 08       FILOOP INX
E392 A7 00           STA A 0,X       STORE THE BYTE
E394 BC A004         CPX   ENDA      SEE IF DONE
E397 26 F8           BNE   FILOOP    CONTINUE OF NO
E399 39              RTS             QUIT WHEN DONE
```

Listing 10. Fill memory routine.

```
              * SUM - MEMORY CHECKSUM

E39A BD E270  SUM    JSR   FROMTO    GET ADDRESS LIMITS
E39D FE A002         LDX   BEGA      GET STARTING ADDRESS
E3A0 4F              CLR A
E3A1 5F              CLR B
E3A2 EB 00    SUMLP  ADD B 0,X       ADD TO CHECKSUM
E3A4 89 00           ADC A #0        ALSO ADD CARRY TO SECOND BYTE
E3A6 BC A004         CPX   ENDA      LAST ADDRESS?
E3A9 27 03           BEQ   SUMDON    YES
E3AB 08              INX             NO, SO INCREMENT AND
E3AC 20 F4           BRA   SUMLP
E3AE B7 D020  SUMDON STA A SAVEX     STORE SUM WHEN DONE
E3B1 F7 D021         STA B SAVEX+1
E3B4 CE D020         LDX   #SAVEX    POINT TO CHECKSUM
E3B7 7E FC2B  VEC4HS JMP   OUT4HS    OUTPUT CHECKSUM AND RETURN WHEN DOZL
```

Listing 11. Checksum routine.

```
              * 'AI' COMMAND - ASCII INPUT ROUTINE

E525 BD E009  ASCIN  JSR   FROMTO    GET ADDRESS RANGE
E528 BD FC0F         JSR   CRLF
E52B FE A004         LDX   ENDA      GET LAST EMPTY ADDRESS
E52E FF D02C         STX   SAVEX     SAVE IT
E531 FE A002         LDX   BEGA      GET STARTING ADDRESS
E534 09              DEX
E535 08       ASCI2  INX
E536 BD FC09         JSR   INEEE     GET NEXT CHARACTER
E539 A7 00           STA A 0,X       STORE IT
E53B A1 00           CMP A 0,X       SEE IF IT STORED OK
E53D 26 08           BNE   ASCI3
E53F FF A004         STX   ENDA      STORE ENDING ADDRESS
E542 BC D02C         CPX   SAVEX     CHECK IF RUN OUT OF MEMORY
E545 26 EE           BNE   ASCI2     NO, SO GET MORE
E547 CE E54F  ASCI3  LDX   #ESTR     MEM FULL OR BAD, SO..
E54A BD FC12         JSR   PDATA     PRINT ERROR
E54D 20 F8           BRA   ASCI3     GO TO REPEAT

E54F 20       ESTR   FCB   ' ,'E,'R,'R,'O,'R,4
```

Listing 12. ASCII input routine.

```
                    * 'AO' COMMAND - ASCII OUTPUT ROUTINE

E556 BD E009  ASCOUT JSR   FROMTO    GET ADDRESS RANGE
E559 BD FC0F         JSR   CRLF
E55C FE A002         LDX   BEGA      GET STARTING ADDRESS
E55F A6 00    ASCO2  LDA A 0,X       GET NEXT CHARACTER
E561 BD FC0C         JSR   OUTEEE    OUTPUT IT
E564 BC A004         CPX   ENDA      SEE IF DONE
E567 27 03           BEQ   ASCO3     YES
E569 08              INX
E56A 20 F3           BRA   ASCO2     REPEAT IF NOT
E56C 39       ASCO3  RTS             RETURN WHEN DONE
```

Listing 14. Move routine.

```
E56D 45       OLDSTR FCC   'ENTER OLD ADDRESSES:'
E581 04              FCB   4
E582 45       NEWSTR FCC   'ENTER NEW ADDRESS:  '
E596 04              FCB   4
E597 CE E56D  MOVE   LDX   #OLDSTR
E59A BD FC12         JSR   PDATA     ASK FOR OLD ADDRESSES
E59D BD E009         JSR   FROMTO
E5A0 BD FC0F         JSR   CRLF
E5A3 CE E582         LDX   #NEWSTR
E5A6 BD FC12         JSR   PDATA     ASK FOR NEW ADDRESS
E5A9 BD FC1E         JSR   BADDR
E5AC FF D042         STX   NEWLOC    SAVE
              * NOW CHECK FOR FORWARD MOVE OR BACKWARD MOVE
E5AF B6 A002         LDA A BEGA
E5B2 B0 D042         SUB A NEWLOC
E5B5 25 2E           BCS   BACK      IF NEW>OLD
E5B7 26 0B           BNE   FORWRD    IF <>
E5B9 B6 A003         LDA A BEGA+1    IF =, CHECK THE REST
E5BC B0 D043         SUB A NEWLOC+1
E5BF 25 24           BCS   BACK      IF NEW>OLD
E5C1 26 01           BNE   FORWRD
E5C3 39       MEXIT  RTS             NO MOVE IF NEW=OLD
              * FORWARD MOVE
E5C4 FE A002  FORWRD LDX   BEGA
E5C7 FF D02C         STX   SAVEX     SAVE COPY OF STARTING ADDRESS
E5CA FE D02C  FWD1   LDX   SAVEX
E5CD 09              DEX
E5CE BC A004         CPX   ENDA      CHECK FOR END
E5D1 27 F0           BEQ   MEXIT     EXIT IF DONE
E5D3 08              INX
E5D4 A6 00           LDA A 0,X       GET NEXT BYTE
E5D6 08              INX             BUMP FROM-POINTER
E5D7 FF D02C         STX   SAVEX
E5DA FE D042         LDX   NEWLOC
E5DD A7 00           STA A 0,X       SAVE BYTE
E5DF 08              INX             BUMP TO-POINTER
E5E0 FF D042         STX   NEWLOC
E5E3 20 E5           BRA   FWD1      AND REPEAT
              * BACKWARD MOVE
E5E5 B6 A004  BACK   LDA A ENDA      COMPUTE END OF NEW AREA
E5E8 F6 A005         LDA B ENDA+1
E5EB F0 A003         SUB B BEGA+1
E5EE B2 A002         SBC A BEGA      LENGTH OF OLD
E5F1 FB D043         ADD B NEWLOC+1
E5F4 B9 D042         ADC A NEWLOC
E5F7 B7 D042         STA A NEWLOC
E5FA F7 D043         STA B NEWLOC+1  STORE LAST LOC OF NEW
E5FD FE A004         LDX   ENDA
E600 FF D02C         STX   SAVEX     SAVE COPY OF LAST LOC
E603 FE D02C  BACK1  LDX   SAVEX
E606 08              INX
E607 BC A002         CPX   BEGA      CHECK FOR END
E60A 27 B7           BEQ   MEXIT     EXIT IF DONE
E60C 09              DEX
E60D A6 00           LDA A 0,X       GET NEXT BYTE
E60F 09              DEX             BUMP FROM-POINTER
E610 FF D02C         STX   SAVEX
E613 FE D042         LDX   NEWLOC
E616 A7 00           STA A 0,X       SAVE BYTE
E618 09              DEX             BUMP TO-POINTER
E619 FF D042         STX   NEWLOC
E61C 20 E5           BRA   BACK1     AND REPEAT
```

Listing 13. ASCII output routine.

FROMTO to get beginning and ending addresses for the dump. The beginning address is then saved in SAVEX. Next, PRNTOP is called.

PRNTOP uses a method of analyzing the length of an instruction known as the Thompson Lister, named after its originator, Noel Thompson. It begins by printing the address in SAVEX. Then it gets the op code of the instruction and, through a series of comparisons, determines the length of that instruction in bytes. Finally, it prints the operation code plus any following bytes and stores in SAVEX the address of the following instruction.

The rest of DESEMB simply checks to see whether all the data requested has been printed and branches back to print more if not. PRNTOP is an important subroutine because it is also used in single-stepping.

Debugging Functions

HUMBUG's strong point is its debugging facility. Let's look at each of the routines used in debugging commands such as BR (used for setting and resetting breakpoints) and SS (for single-stepping).

When the system was first started, the cold-start routine in E4ROM filled each of the twelve locations of BKTAB with FF. BKTAB is used to store the current four breakpoints as shown in Table 1.

In other words, the first three bytes are used for the first breakpoint, the next three are used for the second breakpoint, and so on.

For each breakpoint, the first two bytes contain the address of that breakpoint, while the third byte holds the operation code of the instruction at that location. A breakpoint is set up by substituting an SWI instruction (3F) for the instruction originally there, so that the program will return to the monitor when it reaches the breakpoint. Since putting in the SWI would erase the first byte of the instruction supposed to be there (the op code), this op code is stored in the BKTAB table so it can be

```
                * 'DE' COMMAND - DESEMBLER DUMP

E4B6 BD E009  DESEMB JSR    FROMTO    ASK FOR ADDRESSES
E4B9 FE A002         LDX    BEGA
E4BC FF D02C         STX    SAVEX
E4BF BD E4D1  DES2   JSR    PRNTOP    GO TO PRINT CURRENT LINE
E4C2 B6 A004         LDA A ENDA       SUBTRACT NEXT FROM LAST
E4C5 F6 A005         LDA B ENDA+1
E4C8 F0 D02D         SUB B SAVEX+1
E4CB B2 D02C         SBC A SAVEX
E4CE 24 EF           BCC    DES2      RETURN IF NEXT <= LAST
E4D0 39              RTS              OTHERWISE EXIT

                * PRNTOP - SUBROUTINE TO PRINT ADDRESS AND CURRENT INSTRUCTION

E4D1 BD FC0F  PRNTOP JSR    CRLF
E4D4 CE D02C         LDX    #SAVEX    GET LOCATION OF NEXT ADDRESS
E4D7 BD FC2B         JSR    OUT4HS    PRINT IT
E4DA BD FC30         JSR    OUTS
E4DD FE D02C         LDX    SAVEX     GET ADDRESS OF INSTRUCTION
E4E0 A6 00           LDA A 0,X        GET OPERATION CODE
E4E2 B7 D044         STA A INSTR      SAVE IT
E4E5 BD FC2A         JSR    OUT2HS    PRINT IT
E4E8 FF D02C         STX    SAVEX     INCREMENT SAVEX
E4EB 5F              CLR B            BYTE COUNTER
E4EC B6 D044         LDA A INSTR
E4EF 81 8C           CMP A #$8C       ANALYZE OP CODE FOR NO OF BYTES
E4F1 27 18           BEQ    LENTH3
E4F3 81 8E           CMP A #$8E
E4F5 27 14           BEQ    LENTH3
E4F7 81 CE           CMP A #$CE
E4F9 27 10           BEQ    LENTH3
E4FB 84 F0           AND A #$F0
E4FD 81 20           CMP A #$20
E4FF 27 0B           BEQ    LENTH2
E501 81 60           CMP A #$60
E503 25 08           BCS    LENTH1
E505 84 30           AND A #$30
E507 81 30           CMP A #$30
E509 26 01           BNE    LENTH2
E50B 5C       LENTH3 INC B            3-BYTE:8C,8E,CE,7X,BX,FX
E50C 5C       LENTH2 INC B            2-BYTE:2X,6X,8X,9X,AX,CX,DX,EX
E50D F7 D046  LENTH1 STA B COUNT      1-BYTE:1X,3X,4X,5X
E510 01              NOP
E511 01              NOP
E512 27 10           BEQ    POP3
E514 7A D046         DEC    COUNT
E517 27 05           BEQ    POP1
E519 BD FC2B         JSR    OUT4HS    PRINT 2 BYTES
E51C 20 03           BRA    POP2
E51E BD FC2A  POP1   JSR    OUT2HS    PRINT ONE BYTE
E521 FF D02C  POP2   STX    SAVEX     INCREMENT NEXT
E524 39       POP3   RTS
```

Listing 15. DEsemble routine.

```
                * 'BP' COMMAND - PRINT BREAKPOINT LOCATIONS

E68B C6 30    BPRINT LDA B #'0        BREAKPOINT NUMBER IN ASCII
E68D CE D036         LDX    #BKTAB
E690 FF D02C         STX    SAVEX
E693 5C       BPR1   INC B
E694 C1 35           CMP B #'5        STOP AT 5 BREAKPOINTS
E696 26 01           BNE    BPR2
E698 39              RTS              RETURN WHEN DONE
E699 BD FC0F  BPR2   JSR    CRLF      PRINT CR
E69C 17              TBA              GET BP NUMBER
E69D BD FC0C         JSR    OUTEEE    PRINT BREAKPOINT NUMBER
E6A0 FE D02C         LDX    SAVEX     GET ITS LOCATION IN TABLE
E6A3 A6 00           LDA A 0,X        GET BP ADDRESS
E6A5 81 FF           CMP A #$FF       IS THERE ONE?
E6A7 26 05           BNE    BPR3      YES, GO PRINT IT
E6A9 08              INX
E6AA 08              INX              NO, UPDATE POINTER
E6AB 08              INX
E6AC 20 0C           BRA    BPR4      AND REPEAT
E6AE BD FC30  BPR3   JSR    OUTS      PRINT SPACE
E6B1 FE D02C         LDX    SAVEX
E6B4 BD FC2B         JSR    OUT4HS    PRINT ADDRESS OF BREAKPOINT
E6B7 BD FC2A         JSR    OUT2HS    PRINT OP CODE
E6BA FF D02C  BPR4   STX    SAVEX     SAVE BKTAB LOCATION OF NEXT
E6BD 20 D4           BRA    BPR1      AND REPEAT
```

Listing 16. Print breakpoints routine.

restored later.

When the table is first initialized, it is filled with FFs. Since a breakpoint can never be placed at location FFFF (which is in ROM and contains a vector, rather than an instruction), having an FFFF as the address of each of the breakpoints is an impossible condition used to signify that the breakpoint doesn't exist.

BP Command

The BP monitor command prints out the locations and operation codes of the current breakpoints. For instance, if breakpoint number 2 is at location 1000, the operation code that belongs in that location is 86, and all other breakpoints are unused, then the printout would be as follows:

```
1
2 1000 86
3
4
```

Listing 16 lists the BPRINT subroutine, which prints the breakpoints. It simply scans through BKTAB and prints out the contents for each breakpoint that doesn't have an address of FFFF. The only unusual part of the routine is that the loop counter, which counts up to four breakpoints, is maintained in ASCII. It goes from 31 (the ASCII code for a 1) up to 34 (the ASCII code for a 4) so that it functions both as a counter as well as the number printed at the start of each line.

BR Command

Setting and resetting breakpoints is done with the BR command, which is executed by the BREAK subroutine shown in Listing 17.

For example, if the BR command is used to set up breakpoint number 2 at location 1000,

```
                * 'BR' COMMAND - SET/RESET UP TO FOUR BREAKPOINTS

E61E 8D 45      BREAK  BSR   BKNUM     GET NUMBER OF DESIRED BREAKPOINT
E620 FF B02C           STX   SAVEX     SAVE ADDRESS
E623 8D 22             BSR   BERASE    GO ERASE OLD ONE
E625 CE E582           LDX   #NEWSTR   PRINT "ENTER NEW ADDRESS: "
E628 BD FC12           JSR   PDATA
E62B BD FC1E           JSR   BADDR     GET ADDRESS
E62E FF B042           STX   NEWLOC
E631 E6 00             LDA B 0,X       GET PRESENT OP CODE
E633 86 3F             LDA A #$3F      GET SWI INSTRUCTION
E635 A7 00             STA A 0,X       SUBSTITUTE IT.
E637 FE B02C           LDX   SAVEX     GET POINTER TO BRKTAB AGAIN
E63A B6 B042           LDA A NEWLOC
E63D A7 00             STA A 0,X       STORE ADDRESS IN TABLE
E63F B6 B043           LDA A NEWLOC+1
E642 A7 01             STA A 1,X
E644 E7 02             STA B 2,X       STORE DELETED OP CODE
E646 39                RTS             AND RETURN
                * ERASE PREVIOUS BREAKPOINT, IF ANY, AND RESTORE OP CODE
E647 E6 02      BERASE LDA B 2,X       GET OP CODE
E649 A6 00             LDA A 0,X       GET PART OF ADDRESS
E64B 81 FF             CMP A #$FF      WAS THERE A BREAKPOINT?
E64D 27 0B             BEQ   BEEXIT    NO, EXIT
E64F EE 00             LDX   0,X       YES, GET ADDRESS OF BREAK
E651 E7 00             STA B 0,X       RESTORE OP CODE
E653 FE B02C           LDX   SAVEX
E656 86 FF             LDA A #$FF
E658 A7 00             STA A 0,X       ERASE BREAKPOINT TABLE ENTRY
E65A 39         BEEXIT RTS             AND RETURN

                * BKNUM ROUTINE - GET NUMBER OF DESIRED BREAKPOINT AND POINT
                * TO ITS LOCATION IN BKTAB TABLE

E65B 20         BNSTR  FCC   ' NUMBER: '
E664 04                FCB   4
E665 CE E65B    BKNUM  LDX   #BNSTR
E668 BD FC12           JSR   PDATA
E66B BD FC09           JSR   INEEE     GET BREAKPOINT NUMBER
E66E 80 30             SUB A #$30      CONVERT FROM ASCII
E670 2B 16             BMI   NGEXIT    IF NEGATIVE
E672 27 14             BEQ   NGEXIT    IF ZERO
E674 81 04             CMP A #$4
E676 2E 10             BGT   NGEXIT    IF GREATER THAN 4
E678 36                PSH A
E679 BD FC30           JSR   OUTS
E67C 32                PUL A
E67D CE B036           LDX   #BKTAB
E680 4A         BKN1   DEC A
E681 27 07             BEQ   OKEXIT    EXIT WHEN INDEX POINTS CORRECTLY
E683 08                INX
E684 08                INX             BUMP INDEX BY 3
E685 08                INX
E686 20 F8             BRA   BKN1      AND REPEAT
E688 31         NGEXIT INS             FIX STACK TO BYPASS CALLING ROUTINE ON ERROR
E689 31                INS
E68A 39         OKEXIT RTS             RETURN WHEN DONE
```

Listing 17. Breakpoint set/reset routine.

```
                • BREAKPOINT RE-ENTRY POINT AFTER SWI IN MAIN PROGRAM

E6BF BF A008  BKRETN STS   SP         SAVE USER STACK POINTER
E6C2 30              TSX              TRANSFER TO INDEX
E6C3 8E D07F         LDS   #$D07F     RESET TO MONITOR STACK
E6C6 6D 06           TST   6,X        DECREMENT USER PC TO POINT...
E6C8 26 02           BNE   RONLY      ...TO SWI, NOT PAST IT
E6CA 6A 05           DEC   5,X        DECR LEFT BYTE
E6CC 6A 06    RONLY  DEC   6,X        DECR RIGHT BYTE, AND CONTINUE TO PRINT REG

                • 'RE' COMMAND - PRINT USER REGISTERS FROM STACK

E6CE BD FC0F  REGIST JSR   CRLF
E6D1 FE A008         LDX   SP         POINT TO USER STACK
E6D4 E6 01           LDA B 1,X        GET CC REGISTER
E6D6 58              ASL B
E6D7 58              ASL B            READY FOR SHIFTING INTO CARRY
E6D8 CE 0006         LDX   #6         SET COUNTER
E6DB 58       RELOOP ASL B            MOVE NEXT BIT INTO CARRY
E6DC 86 30           LDA A #$30
E6DE 89 00           ADC A #0         CONVERT TO ASCII
E6E0 BD FC0C         JSR   OUTEEE     PRINT IT
E6E3 09              DEX              BUMP COUNTER
E6E4 26 F5           BNE   RELOOP     PRINT NEXT BIT
E6E6 BD FC30         JSR   OUTS       PRINT SPACE
E6E9 FE A008         LDX   SP         POINT TO USER STACK AGAIN
E6EC 08              INX              STEP PAST CC REGISTER
E6ED 08              INX              POINT TO B ACCUMULATOR
E6EE BD FC2A         JSR   OUT2HS     PRINT B
E6F1 BD FC2A         JSR   OUT2HS     PRINT A
E6F4 BD FC2D         JSR   OUT4HS     PRINT INDEX
E6F7 BD FC2D         JSR   OUT4HS     PRINT PC
E6FA B6 A008         LDA A SP
E6FD F6 A009         LDA B SP+1       GET CURRENT USER STACK
E700 CB 07           ADD B #7
E702 89 00           ADC A #0         CHANGE BACK TO VALUE IT HAD IN USER PGM
E704 B7 D02C         STA A SAVEX
E707 F7 D02D         STA B SAVEX+1    TEMP SAVE IT
E70A CE D02C         LDX   #SAVEX     POINT TO IT
E70D BD FC2D         JSR   OUT4HS     PRINT IT
E710 7E FC06         JMP   HOTST      AND RETURN TO FCROM
```

Listing 18. Breakpoint reentry and register print routines.

the whole exchange with the monitor would be:

BR NUMBER: 2 ENTER NEW ADDRESS: 1000
(user's entries are underlined).

Only a number from 1 to 4 is allowed for a breakpoint number; any other entry will return to the command loop without doing anything.

As soon as a valid breakpoint number is entered, the old breakpoint (if any) is restored and erased from the table. If the new address is valid, then the new breakpoint is set up; but if the new address is a carriage return or any other invalid character, then no new breakpoint is entered. This is, therefore, a good way of erasing breakpoints.

Listing 17 first goes to the subroutine BKNUM, which asks for the breakpoint number and points the index register at the corresponding entry in the BKTAB table. This pointer is then saved in SAVEX.

Next, subroutine BERASE erases the old breakpoint (if any) from the table. It looks at the first byte of the breakpoint address in the table. If this byte is not FF (no breakpoints can exist at locations FF00 through FFFF, since this is all ROM), then it gets the op code from the table, puts it back into the original address and puts an FF into BKTAB to make the address invalid.

Finally, the program asks for the new address and then pulls a switch. The op code is yanked out of the breakpoint location, a 3F is substituted, and the breakpoint address and the op code are placed into BKTAB.

SWI Reentry

What happens when a user program runs and hits a breakpoint? You may remember from last month's article that FCROM has an address of FFED in the SWI interrupt vector at location FFFA. When an SWI interrupt occurs, the 6800 will look into location FFFA to get the address to go to. In this case, it will start executing a program at FFED.

But there were two instructions starting at FFED that loaded into the index register the number in location A012 and then executed JMP 0,X. Hence, the number in A012 is a pointer to the real starting point of the SWI service routine. This pointer is in RAM so it can be changed by user programs.

A012 is initialized during the initial power-up sequence to point to BKRETN, so an SWI interrupt eventually winds up at BKRETN. This routine is shown in Listing 18.

When an SWI gets us to BKRETN, the contents of the stack pointer are stored at location SP, or location A008. At this point, the stack pointer points to the next empty location of the user stack, just under the seven bytes that hold all the register data that was dumped into the stack by the 6800 when it performed the SWI.

The next instruction following BKRETN transfers the contents of the stack pointer to the index register. However, the 6800 adds 1 to this number before it loads it into the index register. Thus, now the index register points to the last of the seven bytes, instead of the next empty location.

The stack now has the following seven bytes:
Program counter (low)
Program counter (high)
Index register (low)
Index register (high)
A Accumulator
B Accumulator
CC Reg.—IX now points here
Empty—SP now points here

In the next step, the stack pointer is loaded with the address of the monitor stack at D07F, so that all following operations use a different stack area.

The next four instructions subtract one from the PC (program counter) contents stored in the user stack. The PC, as stored after the SWI, points to the next instruction after the SWI itself. Subtracting one points it back to the SWI, so that when the contents of the PC are printed, it will indicate the address where the breakpoint occurred, rather than the address of the next byte. This is essential, so that when we continue from the breakpoint we resume at the instruction which had been replaced by the breakpoint, rather than the next byte after it.

After this is done, the program continues into the same routine that is executed for the RE, or register, dump command.

This REGIST routine uses the contents of SP to point to the user's stack. Its function is similar to SWTBUG's R command, but it does it in a slightly different way. First, it separates the bits of the condition code register and prints them separately, instead of printing them as a hex number, as SWTBUG does. Second, it adds 7 to the stack pointer before printing it. For instance, if SWTBUG printed a register dump as

C4 BB AA 1234 5678 4321

HUMBUG would print it as

000100 BB AA 1234 5678 4328.

Why the difference in the stack pointer? SWTBUG prints the stack pointer the way it exists after the breakpoint SWI instruction; HUMBUG prints it the way it was just before the breakpoint.

Listing 19 shows the steps used for executing the CO command. SWTBUG has a G command that is used both for starting programs as well as for continuing after a breakpoint; HUMBUG has separate GO and CO commands.

GO is used just for starting a program. It always uses the contents of A048 and A049 for a starting address. CO, on the other hand, is used only for continuing after a breakpoint or single-step. It can't be used to start a program, since the contents of SP are undefined at the beginning.

SS—Single-Stepping

Executing the single-step command was shorter and simpler than I expected. The entire single-step routine is shown in Listing 20.

```
                • 'CO' COMMAND - CONTINUE AFTER A BREAKPOINT

E713 BE A008  CONT   LDS   SP         GET USER STACK POINTER
E716 3B              RTI              AND RETURN TO HIS PROGRAM
```

Listing 19. Continue from breakpoint routine.

The SS command uses the contents of the SP, or stack pointer, location, which is initialized only upon reentering after a breakpoint, so SS can only be used after breakpoints. This is a minor annoyance at first, but you'll get used to it. (E8ROM actually has an ST, or STart, command to get around this, but that is not necessary for our purposes.)

When the SS command is called, the STEP routine of Listing 20 uses the user stack pointer to get the current user program counter and saves it in USERPC and also in SAVEX. Then it goes to PRNTOP, which uses SAVEX to find the instruction, prints it and then updates SAVEX to point to the next instruction. This pointer is also left in the index register when PRNTOP finishes.

The next part of STEP, starting at location E725, uses this pointer to pull out the op code of this instruction, save it in memory and replace it with a 3F or SWI. It then checks whether this 3F was stored. If not, it goes to NOGOOD to print the error message NO! This prevents single-stepping through ROM or nonexistent memory.

Eventually, the monitor will jump to the instruction to be performed and execute it. Right after this instruction is an SWI, which will return to the monitor immediately after the one instruction being executed. But what if that instruction is a jump or branch, so that the following SWI is never executed? The next part of the monitor, starting at OK1, checks for that.

If the instruction about to be stepped through is a jump or branch, then another SWI is placed at the location where the computer will jump. There are now two SWI instructions, so

Listing 20. Single-step routine.

```
                  * 'SS' COMMAND - SINGLE STEP AFTER BREAKPOINT
E717 FE A008      STEP   LDX   SP        GET USER STACK POINTER
E71A EE 06               LDX   6,X       GET USER PC
E71C FF D02E             STX   USERPC    SAVE IT
E71F FF D02C             STX   SAVEX
E722 BD E4B1             JSR   PRNTOP    PRINT ADDRESS AND INSTRUCTION
                  * REPLACE NEXT INTRUCTION WITH SWI
E725 FF D030             STX   NEXT      SAVE ADDRESS
E728 A6 00               LDA A 0,X       GET INSTRUCTION
E72A B7 D032             STA A NEXT+2    SAVE IT
E72D 86 3F               LDA A #$3F      GET SWI
E72F A7 00               STA A 0,X
E731 A1 00               CMP A 0,X       CHECK IT
E733 27 02               BEQ   OK1       IT STORED OK
E735 20 35               BRA   NOGOOD    ABORT IF ERROR
                  * NEXT, SEE IF A BRANCH OR JUMP IS INVOLVED
E737 B6 D044      OK1    LDA A INSTR     GET OP CODE
E73A 81 20               CMP A #$20
E73C 25 04               BCS   NOBR      NO BRANCH
E73E 81 30               CMP A #$30
E740 25 6E               BCS   YESBR     YES
E742 81 39        NOBR   CMP A #$39      CHECK FOR RTS
E744 26 03               BNE   NOTRTS    NO
E746 7E E7EF             JMP   RTSIN     YES
E749 81 3B        NOTRTS CMP A #$3B
E74B 27 1F               BEQ   NOGOOD    DON'T DO RTI
E74D 81 3F               CMP A #$3F
E74F 27 1B               BEQ   NOGOOD    DITTO FOR SWI
E751 81 6E               CMP A #$6E
E753 26 03               BNE   NOTJIN
E755 7E E7DE      JINV   JMP   JINDEX    OK FOR INDEXED JUMPS
E758 81 AD        NOTJIN CMP A #$AD
E75A 27 F9               BEQ   JINV      DITTO
E75C 81 7E               CMP A #$7E
E75E 27 77               BEQ   JEXT      OK FOR EXTENDED JUMPS
E760 81 BD               CMP A #$BD
E762 27 73               BEQ   JEXT      DITTO
E764 81 8D               CMP A #$8D
E766 27 48               BEQ   YESBR     BSR IS A BRANCH TOO
E768 81 3E               CMP A #$3E
E76A 26 15               BNE   NORMAL    OK IF NOT WAI
                  * REFUSE TO DO SOME INSTRUCTIONS
E76C CE E77D      NOGOOD LDX   #NOSTR
E76F BD FC12             JSR   PDATA     PRINT "NO!"
E772 FE D030             LDX   NEXT
E775 B6 D032             LDA A NEXT+2
E778 A7 00               STA A 0,X       RESTORE NEXT INSTR ON ERROR
E77A 7E FC06             JMP   HOTST
E77D 4E           NOSTR  FCC   'NO!'
E780 04                  FCB   4
                  * NORMAL INSTRUCTIONS ARE EASY
E781 86 FF        NORMAL LDA A #$FF      ERASE ALT ADDRESS LOC
E783 B7 D033             STA A BRANCH
E786 CE E790      GOUSER LDX   #SSRETN   REDIRECT SWI RETURN
E789 FF A012             STX   SWIJMP
E78C BE A008             LDS   SP        GET USER STACK
E78F 3B                  RTI             GO TO USER
```

that if a conditional branch is involved, we'll stop whichever way we go. (And, of course, the deleted instruction is saved.) This is somewhat complex for relative branches and indexed JMPs and JSRs, but this is handled by routines that add or subtract offsets.

There are other instructions that need checking. An RTS is executed by fetching the return address from the stack. HUMBUG doesn't attempt to execute the difficult RTI, SWI and WAI instructions.

Once everything is set up, the program advances to GOUSER at location E786, ready to do an RTI to go to the user program. But first we must initialize the RAM location SWIJMP at A012 with the return address of SSRETN (instead of BKRETN) just before we go to the user program. Otherwise, the SWI, which will return to HUMBUG, will return us to the breakpoint routine instead of back to the single-step routine.

After the single-step is performed, the computer returns back to the single-step program at SSRETN. This part of the program now resets SWIJMP to point back to BKRETN, erases the SWI instruction and replaces it with the original byte, erases the alternate SWI, which had been placed into the program for jumps and branches, and then goes to BKRETN to save the stack pointer and print registers as it does after a normal breakpoint.

Conclusion

With this information, you can now construct your own version of HUMBUG. If you prefer to obtain complete source code on disk or cassette, or burned EPROMs, contact Star-Kits, PO Box 209, Mt. Kisco, NY 10549. ■

```
                * RETURN POINT FROM SINGLE STEP
E790 CE E6BF    SSRETN LDX   #BKRETN   RESTORE BREAK ADDRESS
E793 FF A012           STX   SWIJMP
E796 FE B030           LDX   NEXT      RESTORE NEXT OP CODE
E799 B6 B032           LDA A NEXT+2
E79C A7 00             STA A 0,X
E79E B6 B033           LDA A BRANCH    CHECK BRANCH ADDRESS
E7A1 81 FF             CMP A #$FF
E7A3 27 08             BEQ   NONE
E7A5 FE B033           LDX   BRANCH    RESTORE IT
E7A8 B6 B035           LDA A BRANCH+2
E7AB A7 00             STA A 0,X
E7AD 7E E6BF    NONE   JMP   BKRETN    STORE STACK PTR AND PRINT REGISTERS
                *
                *HANDLE EFFECTIVE ADDRESS OF BRANCH
E7B0 FE B02E    YESBR  LDX   USERPC
E7B3 E6 01             LDA B 1,X       GET OFFSET
E7B5 27 06             BEQ   ZEROOF
E7B7 2B 18             BMI   MINOFF
                * PLUS OFFSET
E7B9 08         PLUSOF INX             ADD OFFSET TO INSTR ADDRESS
E7BA 5A                DEC B
E7BB 26 FC             BNE   PLUSOF
E7BD 08         ZEROOF INX             POINT TO NEXT INSTR
E7BE 08                INX
E7BF FF B033    GOTADD STX   BRANCH    SAVE ADDRESS
E7C2 A6 00             LDA A 0,X       GET INSTRUCTION
E7C4 B7 B035           STA A BRANCH+2  SAVE IT
E7C7 86 3F             LDA A #$3F
E7C9 A7 00             STA A 0,X       SUBSTITUTE SWI
E7CB A1 00             CMP A 0,X       CHECK THAT IT WENT IN
E7CD 27 B7             BEQ   GOUSER    GO TO USER IF OK
E7CF 20 9B             BRA   NOGOOD    IF IT DIDN'T STORE PROPERLY
                * MINUS OFFSET
E7D1 09         MINOFF DEX             SUBTRACT OFFSET
E7D2 5C                INC B           FROM INSTR ADDRESS
E7D3 26 FC             BNE   MINOFF
E7D5 20 E6             BRA   ZEROOF
                *
                * HANDLE EXTENDED JUMP ADDRESS
E7D7 FE B02E    JEXT   LDX   USERPC
E7DA EE 01             LDX   1,X       GET EXTENDED JUMP ADDRESS
E7DC 20 E1             BRA   GOTADD    GO TAKE CARE OF IT
                *
                *HANDLE INDEXED JUMP
E7DE FE B02E    JINDEX LDX   USERPC
E7E1 E6 01             LDA B 1,X       GET OFFSET
E7E3 FE A008           LDX   SP
E7E6 EE 04             LDX   4,X       GET USER INDEX REGISTER
E7E8 09                DEX
E7E9 09                DEX             POINT TO 2 BYTES UNDER
E7EA 5D                TST B
E7EB 27 D0             BEQ   ZEROOF    IF OFFSET IS ZERO
E7ED 20 CA             BRA   PLUSOF    IF OFFSET IS NONZERO
                *
                * HANDLE RTS INSTRUCTION
E7EF FE A008    RTSIN  LDX   SP        GET USER STACK POINTER
E7F2 EE 08             LDX   8,X       GET RETURN ADDRESS FROM USER'S STACK
E7F4 20 C9             BRA   GOTADD    AND TREAT IT AS A JUMP
```