INTRODUCING SMON

A TUTORIAL MONITOR

The S(imple)MON as its name suggests, is a very simple monitor. In fact it is only just enough to allow you to view, alter, and run your own programs.

The blandness of SMON is by design. This MONitor has been designed purely as a easy-to-understand tutorial, not as a fully functioned MONitor like JMON.

The JMON description lacked ease-of-understanding because JMON was not designed to be an easily read program. The SMON tutorial is a stepping stone to understanding (and writing) more advanced programs like JMON.

The SMON listing is more in the form of an assembler output listing. This means that it is symbolic and includes labels. You will be able to follow the program by reading English not HEX!

The typesetting is more open and you will find it a pleasure to read through.

RUNNING SMON

Before you can fully understand how the SMON program works, you need to see it working so you can understand the action of the program. To get a working SMON requires either a NVR or an extra RAM chip and a TEC mod that allows you to switch the expansion port to the 0000 (this is a highly recommended mod).

You also are required to sit down and type it in. This should not be much of a problem as it is less than a page (256) bytes long. JMON owners are laughing as they don't have to increment the RAM pointer after each byte and also can save it on tape for future use.

Once you have SMON up and running, get to know its actions. Notice that there are only two modes, the ADDRESS and DATA modes. Notice that the ADDRESS mode only effects the operation of the DATA keys not the control keys. Also notice that you can run a program straight from the ADDR mode by hitting "GO" and notice there is no auto zeroing as with MON-1.

Make sure you get to know the outside actions of SMON fully before trying to understand the inside working of the software.

SMON OVERVIEW

There are 3 main sections to SMON. The first is the main program. It is responsible for the set-up of variables, the calling of sub-routines and the processing of key strokes. The main body is located from 0000 to 0065 (it fits neatly under the NMI handler).

There is a slight difference between the main body of JMON and SMON. The difference is JMON calls a subroutine that scans the keyboard and display and returns when it finds a key press. SMON calls the scan routine and then looks for a key press itself.

The second section is the NMI handler at 0066. This is pinched straight from MON-1 except that SMON pushes AF and MON-1 doesn't.

The third section is collectively the sub-routines. The actions of the subroutines are to produce the tones, scan the display, set the dots and convert HEX values into display code. The subroutines are located at 0070.

A classic strategy is employed by SMON. Those of you who have read through the JMON listing will be familiar with it. The strategy is that control byte(s) are used to flag the current operating modes. Outside appearances, (Eg. the displays), are set-up by the master routine by referring to these bytes. This means that routines can change the operating mode of the software simply by changing a byte. The routines do not need to be concerned with up-dating the outside appearance, they leave this for the master.

Applying this to SMON, the initial variables are set-up from address 0000 through to 0014. The display up-dating is performed by the sub-routines called by the instructions at 0017 to 001D.

The key handler section, which alters these variable bytes starts at 0026. After the key handler is finished it jumps to 0010 (or 000D if the AD key was pressed as it must store the new control byte) and displays are up-dated.

SMON VARIBLES

SMON has only two varibles. They are the CONTROL BYTE and the RAM pointer. The control byte flags between the two operating modes, ADDRESS and DATA, by the state of bit 4. If bit 4 is a logic 1 then SMON is in the ADDRESS mode. If it is a zero then SMON is in the DATA mode. No other bits are used in the control byte. The control byte is stored at 0F08.

The RAM pointer holds the address of the RAM location the SMON is currently displaying. The RAM pointer is stored at 0F06.

POINTS OF INTEREST

If you are wondering why the stack pointer is loaded at 1000 and not 0FFF, the reason is the stack pointer is decremented by one BEFORE anything is pushed onto it. Therefore the first value pushed onto it will be stored at 0FFE and 0FFF.

Another point of interest is that there is a "SETDOTS" routine but no "RESET" dots routine, so how do the dots get erased from the display?

The answer is the dots are removed from the display buffer when a new value is written in by the HEX-to-display code routine. The HEX-to-display code routine is always called before the SETDOTS routine so when the SETDOTS routine is reached all the dots are cleared.

Have a look at the AD key handler at 0048. Address 0F08 and press the AD key. Watch what happens to the value at 0F08 each time you hit the AD keythat . You should now see how the ADDR and DATA modes are toggled between.

Now look at the set dots routine at 00AB. Do you see how the difference between the ADDR and DATA modes is detected and then acted upon?

Another feature to look closely at is the method used to shift a new data nibble into the current RAM location. The code to do this is at 0062.

The code that enters a new nibble into the RAM pointer buffer, as required when in the ADDR mode, is at 005C.

Grab your copy of issue 13 and turn to page 16. You will find my three digit counter (I wrote the counter program but not the comments).

Look at convert to display code subroutine at 0A00. It is identical in operation to the one used here in SMON. If you look closely, you will notice the CALL and RETurn instructions at 0A09 are missing from SMON. I will leave it to you to discover why the CALL and RETurn are not needed. (Hint: look at the CALL address).

Also look at the differences between the scanning routines, SMON's scans 6 digit while the three digit counter only scans 3. Do you see how this difference is achieved without the use of a counter?

Finally, a slight change to the SMON's tone routine makes SMON's shorter and easier to understand than JMON's.

There is a lot of information and reference material packed into this simple-to-understand program.

```
0000  31 00 10                      LD SP,1000              ;set the stack to top of RAM+1
0003  21 00 08                      LD HL,0800              ;load HL with first RAM location
0006  22 06 0F                      LD (PTR_BUFF),HL        ;and put it in RAM pointer buffer
0009  CD C1 00                      CALL RESET_TONES        ;call double reset tone
000C  AF          CLR_CON_BYT:      XOR A                   ;clear control byte

      ;When the AD key is pressed, the MONitor jumps to here to store a new control byte provided by the AD key handler.

000D  32 08 0F    STR_CON_BYT:      LD (CONT_BUFF),A        ;store control byte

      ;MON jumps here to clear the key buffer in the interrupt reg after key processing (except for AD see above).

0010  3E FF       CLR_KEY_FLG:      LD A,FF                 ;set interrupt vector register to FF to signify no
0012  ED 47                         LD I,A                  ;key press:
0014  2A 06 0F                      LD HL,(PTR_BUFF)        ;put RAM pointer into HL and call
0017  CD 8B 00                      CALL CON_HL_A           ;HL and (HL) to display code conversion
001A  CD AB 00                      CALL SET_DOTS           ;call set dots
001D  CD 70 00    KEY_LOOP:         CALL SCAN               ;call scan: Key loop jumps here until key press
0020  ED 57                         LD A,I                  ;get byte from interrupt register
0022  FE FF                         CP FF                   ;test for FF: If it is then no key is
0024  28 F7                         JR Z, KEY_LOOP          ;pressed so keep looping else continue
0026  F5                            PUSH AF                 ;on and process key: save key value
0027  CD C4 00                      CALL KEY_TONE           ;call key press tone
002A  F1                            POP AF                  ;recover key value
002B  2A 06 0F                      LD HL, (PTR_BUFF)       ;put RAM pointer into HL
002E  CB 67                         BIT 4,A                 ;if the key +, -, go or AD then bit 4 is
0030  28 1D                         JR Z, DAT_KEY_PROC      ;set: jump if data key
0032  FE 10                         CP 10                   ;else process control key here: Is it "+"
0034  20 06                         JR NZ, CP_MINUS         ;jump if not
0036  23                            INC HL                  ;else increment RAM pointer
0037  22 06 0F    PTR_UPDATE:       LD (PTR_BUFF), HL       ;place new RAM pointer in its buffer
003A  18 D4                         JR CLR_KEY_FLG          ;jump to set key buffer and up-date display
003C  FE 11       CP_MINUS:         CP 11                   ;is key "-"?
003E  20 03                         JR NZ, CP_GO            ;jump if not
0040  2B                            DEC HL                  ;decrement RAM pointer
0041  18 F4                         JR PTR_UPDATE           ;jump to up-date its buffer
0043  FE 12       CP_GO:            CP 12                   ;is key the "GO" key?
0045  20 01                         JR NZ, AD_KEY           ;jump if not
0047  E9                            JP (HL)                 ;else jump to current RAM location
0048  3A 08 0F    AD_KEY:           LD A, (CONT_BUFF)       ;key MUST BE "AD": GET control byte in A
004B  EE 10                         XOR 10                  ;toggle the mode Eg. if ADDR now DATA and
004D  18 BE                         JR STR_CON_BYT          ;vica-versa: Jump to store new control byte
004F  47          D_KEY_PROC:       LD B,A                  ;DATA KEY HANDLER: save key value in B
0050  3A 08 0F                      LD A, (CONT_BUFF)       ;get control byte in A
0053  CB 67                         BIT 4,A                 ;test for which mode
0055  78                            LD A,B                  ;put key value back in A
0056  20 04                         JRNZ, D_KEY_AD_MD       ;jump if ADDR mode
0058  ED 6F                         RLD                     ;else shift nibble into RAM location
005A  18 B4                         JR CLR_KEY_FLG          ;jump to up-date display
005C  21 06 0F    D_KEY_AD_MD:      LD HL, PTR_BUFF         ;DATA key in ADDR mode: point HL at RAM
005F  ED 6F                         RLD                     ;pointer buffer and shift
0061  23                            INC HL                  ;the new nibble in
0062  ED 6F                         RLD                     ;and shift the carry out nibble into second
0064  18 AA                         JR CLR_KEY_FLG          ;byte: Jump to up-date displays
0066  F5          NMI_HANDLER:      PUSH AF                 ;NMI HANDLER: Save A
0067  DB 00                         IN A,(00)               ;get key value
0069  E6 1F                         AND 1F                  ;mask off junk bits
006B  ED 47                         LD I,A                  ;save it in interrupt vector register
006D  F1                            POP AF                  ;recover AF
006E  C9                            RET                     ;return
006F  FF                            RST 38                  ;ignore
0070  06 20       SCAN:             LD B,20                 ;SCAN: load B with scan bit
0072  21 00 0F                      LD HL, DISP_BUFF        ;point HL at display buffer
0075  7E          SCAN_LOOP:        LD A,(HL)               ;get first display digit
0076  D3 02                         OUT (02),A              ;output to port 2
0078  78                            LD A,B                  ;put scan bit in A
0079  D3 01                         OUT (01),A              ;output to port 1
007B  06 80                         LD B,80                 ;use B for a short
007D  10 FE       D_LOOP:           DJNZ, D_LOOP            ;display delay
007F  23                            INC HL                  ;point HL to next display byte
0080  47                            LD B,A                  ;put scan bit back into B
0081  AF                            XOR A                   ;clear the last port
```

```
0082  D3 01                        OUT (01),A            ;switched on to prevent "ghosting"
0084  CB 08                        RRC B                 ;shift scan bit to next digit
0086  30 ED                        JR NC SCAN_LOOP       ;jump if scan bit didn't "fall" into carry
0088  D3 02                        OUT (02),A            ;else all digits scanned: (unnecessarily)
008A  C9                           RET                   ;clear port 2 and return
```

;HL is converted to display code via the convert A routine. After H is converted the corresponding two display bytes are at the lower end of the display buffer. The next two bytes in the display buffer are for the display codes for L.

```
008B  01 00 0F    CON_HL_A:        LD BC, DISP_BUFF      ;HL CONVERT: point BC to display buffer
008E  7C                           LD A,H                ;get high byte
008F  CD 97 00                     CALL CON_A            ;call convert A
0092  7D                           LD A,L                ;get low byte
0093  CD 97 00                     CALL CON_A            ;call convert A
```

;After HL is converted, the byte at (HL) is converted. This is the value that appears on the DATA displays.

```
0096  7E                           LD A,(HL)             ;get contents of RAM location
0097  F5          CON_A:           PUSH AF               ;save byte for second nibble convert
0098  07                           RLCA                  ;shift
0099  07                           RLCA                  ;high nibble to
009A  07                           RLCA                  ;low nibble spot
009B  07                           RLCA                  ;for ease of convertion
009C  CD A0 00                     CALL CON_NIBBLE       ;call nibble convert
009F  F1                           POP AF                ;recover A for second nibble Convert
00A0  E6 0F       CON_NIBBLE:      AND 0F                ;mask off unwanted bits
00A2  11 E0 00                     LD DE, DISP_COD_TAB   ;point DE to conversion table
00A5  83                           ADD A,E               ;add nibble value to table pointer
00A6  5F                           LD E,A                ;put new table pointer low byte into DE
00A7  1A                           LD A,(DE)             ;get display code and put in display buffer
00A8  02                           LD (BC),A             ;any set dot is cleared by this operation
00A9  03                           INC BC                ;point BC to next display buffer
00AA  C9                           RET                   ;done
```

The set dots routine causes either the DATA or ADDR dots to be set on the LED display. This is achveived by setting bit 4 of the DATA or ADDR section of the display buffer. Because the DATA section is at the higher end of the display buffer, HL is loaded to point the end of the diplay buffer and is decremented down two bytes in the case of ADDR mode. This is more efficent than loading HL several times.

```
00AB  21 05 0F    SETDOTS:         LD HL, DISP_BUFF_END  ;point HL to data end of display buffer
00AE  06 02                        LD B,02               ;set B for 2 dots
00B0  3A 08 0F                     LD A, (CONT_BUFF)     ;get control byte
00B3  CB 67                        BIT 4,A               ;test mode
00B5  28 04                        JR Z, DO_SET          ;jump if it is DATA mode
00B7  2B                           DEC HL                ;else move HL to lowest
00B8  2B                           DEC HL                ;ADDR display buffer
00B9  06 04                        LD B,04               ;set B for 4 dots
00BB  CB E6       DO_SET:          SET 4,(HL)            ;set dot
00BD  2B                           DEC HL                ;point to next digit
00BE  10 FB                        DJNZ, DO_SET          ;do until B=0
00C0  C9                           RET                   ;done
```

The double reset tone is created by calling the key tone then returning to the key press tone.

```
00C1  CD C4 00    RESET_TONES:     CALL KEY_TONE         ;call key tone
00C4  0E 40       KEY_TONE:        LD C,40               ;set C for half cycle count
00C6  AF                           XOR A                 ;turn off speaker bit
00C7  D3 01       TONE_LOOP:       OUT (01),A            ;on bit 7 of port 1
00C9  06 40                        LD B,40               ;put delay period into B
00CB  10 FE       TONE_DELAY:      DJNZ, TONE_DELAY      ;and do delay
00CD  EE 80                        XOR 80                ;toggle speaker bit in A
00CF  0D                           DEC C                 ;count down cycles
00D0  20 F5                        JR NZ, TONE_LOOP      ;toggle speaker bit until L is 0
00D1  C9                           RET                   ;done

00E0              DISP_COD_TAB     DEFB EB, 28, CD, AD   ;display codes for 0, 1, 2, 3
00E4                               DEFB 2E, A7, E7, 29   ;display codes for 4, 5, 6, 7
00E8                               DEFB EF, 2F, 6F, E6   ;display codes for 8, 9, A, B
00EC                               DEFB C3, EC, C7, 47   ;display codes for C, D, E, F

                  PTR_BUFF         EQU 0F06              ;set PTR_BUFF to 0F06
                  CONT_BUFF        EQU 0F08              ;set CONT_BUFF to 0F08
                  DISP_BUFF        EQU 0F00              ;set DISP_BUFF to 0F00
                  DISP_BUFF_END    EQU 0F05              ;set DISP_BUFF_END to 0F05
```