THE Z86C08 CONTROLS A SCROLLING LED MESSAGE DISPLAY

DISPLAY TEXT AND GRAPHICS WHILE SCROLLING THE LED MESSAGE WITH A MINIMUM OF HARDWARE. THE CHARACTERS ARE DISPLAYED USING A TIME-DIVISION MULTIPLEX TECHNIQUE WITH MORE THAN SIX CHARACTERS EASILY ADDED BY SOFTWARE.

INTRODUCTION

Designed around the Zilog Z86C08, 18-pin microcontroller, this LED display is capable of displaying text as well as graphics, with hardware being kept to a minimum. The present design is configured to display u to six characters, but additional characters are easily added with minimal software changes.

The display uses a TDM (Time-Division Multiplexed) technique to display the characters. That is, each character segment is turned on for a few hundred microseconds at a time, then is "refreshed" every 18 ms.

For demonstration purposes, the software routine displays the time in hours, minutes, and seconds. Once every sixty seconds, a "secret message" scrolls across the display. Then, after the message is displayed, the program reverts back to displaying the time.

THE HARDWARE

The Z86C08 (Figure 1) uses Port 2 for the row data and clocks the column data out of Port 0. PNP transistors are used to drive the rows, since the Z86C08 cannot source the required current directly. The characters are displayed in a 5x7 format, so only seven lines are needed out of Port 2. A logic Low turns on the transistors, while a logic High turns them off.

THE HARDWARE (Continued)

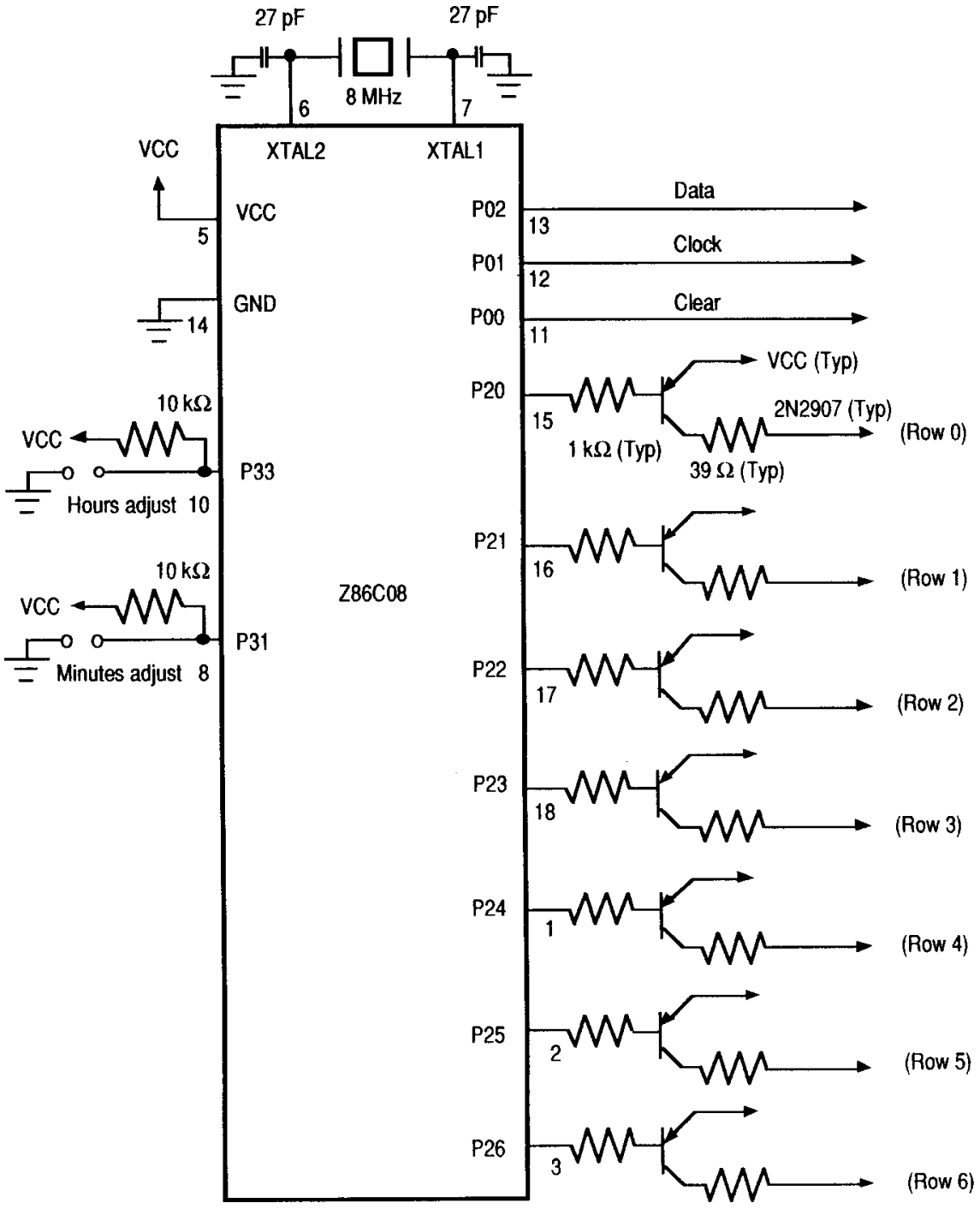

Figure 1. Z86C08 Circuit Functions

The columns are driven by six 74HCT164 shift registers (Figure 2). The 74HCT164s do not have the necessary sink capability to drive the LEDs, so the outputs of the shift registers drive six 75492 high-current buffers. These are capable of sinking 250 mA per pin continuously, with instantaneous current per column approaching 1.5 A. Each 74HCT164 drives six columns; five character segments, plus one space. The last Q output from the shift register is then fed to the DATA input of the next shift register, while all CLEAR and CLOCK lines are wired in parallel.

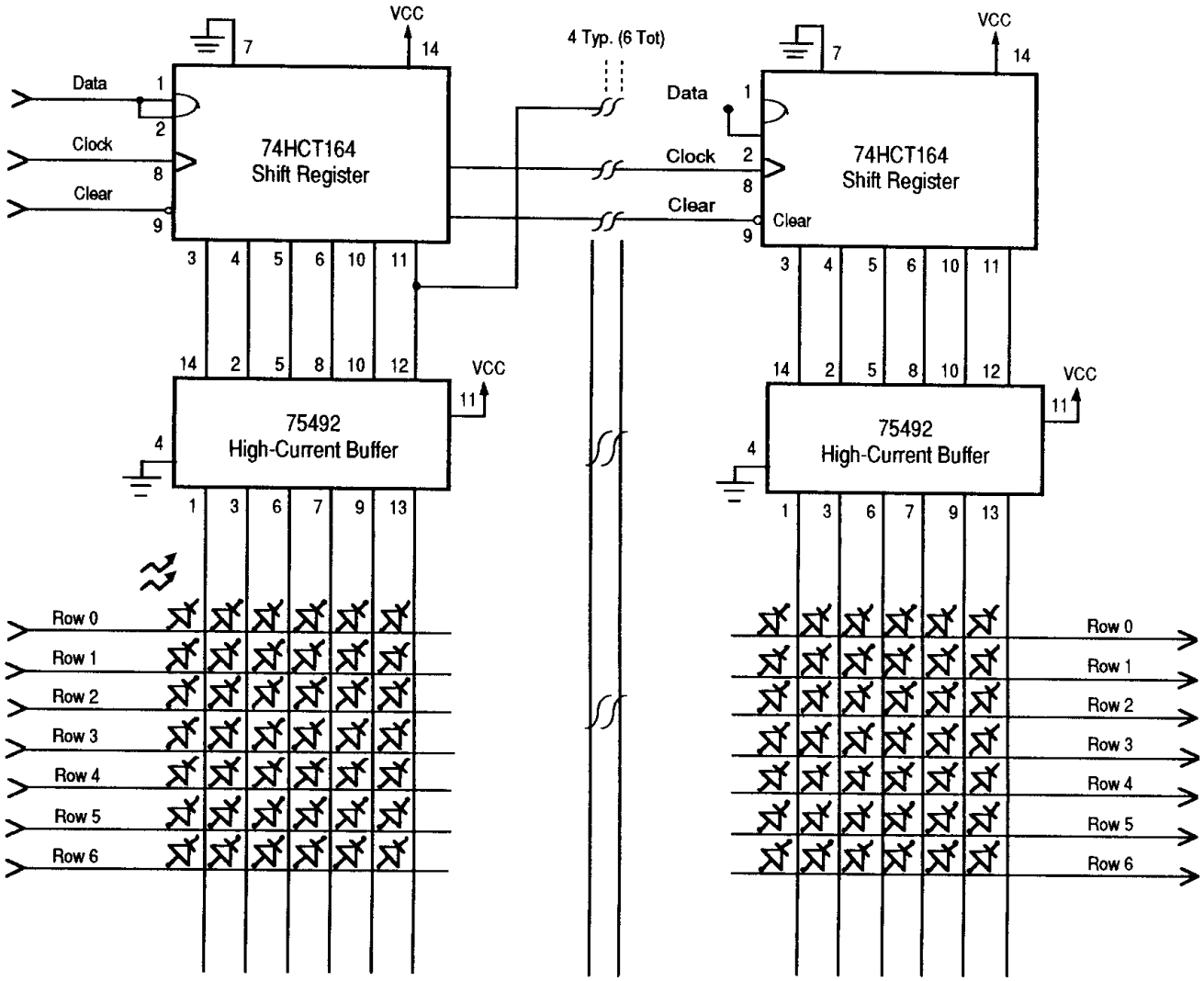

Figure 2. LED Circuitry

THE HARDWARE (Continued)

To scan the display, a logic 1 is output at P0-2. Next, P0-1 is taken High, then immediately taken Low, along with P0-2. This clocks a logic 1 into the first shift register (column 1). Next a character segment is output at P2. The transistors that are turned on will source current for the LEDs with the column providing a sink path. The columns are left on for about 500 µs and then turned off. Now, the column data is shifted one and a new character segment is output. After the last column has been scanned, the display is "refreshed" starting at the first column again.

To set the time, two push-button switches are connected to Port 3 to adjust the hours and minutes.

THE SOFTWARE

The initialization part of the program configures the ports, timers, interrupts, etc. The size of the display buffer (FIFO) is determined by the number of columns in the display. The bottom of the display buffer starts at 20H. The upper limit of the display buffer can extend to 70H. This translates into a sixteen-character display. There is no need to have a display buffer larger than the display itself, since only that many characters can be viewed at a time. A power-up, the display is configured for showing the time.

The flowchart for the display appears in Figures 3a and 3b. To keep time, the internal clock was divided down by T0 to provide an interrupt every 5 ms. The interrupt routine increments a counter, and when 200 counts is reached, the seconds register is incremented by one. Also, when ten seconds is reached, tens-of-seconds is incremented. This counter continues to increment minutes, tens-of-minutes, hours, and tens-of-hours. Upon power-up, the display shows 12:00. When it is around 9:00, the leading hours digit is blanked. Time is adjusted by two push-button switches. When pressed, one increments the minutes register every second, while the other increments the hours register every second. The time data is stored in locations 09H-0EH.

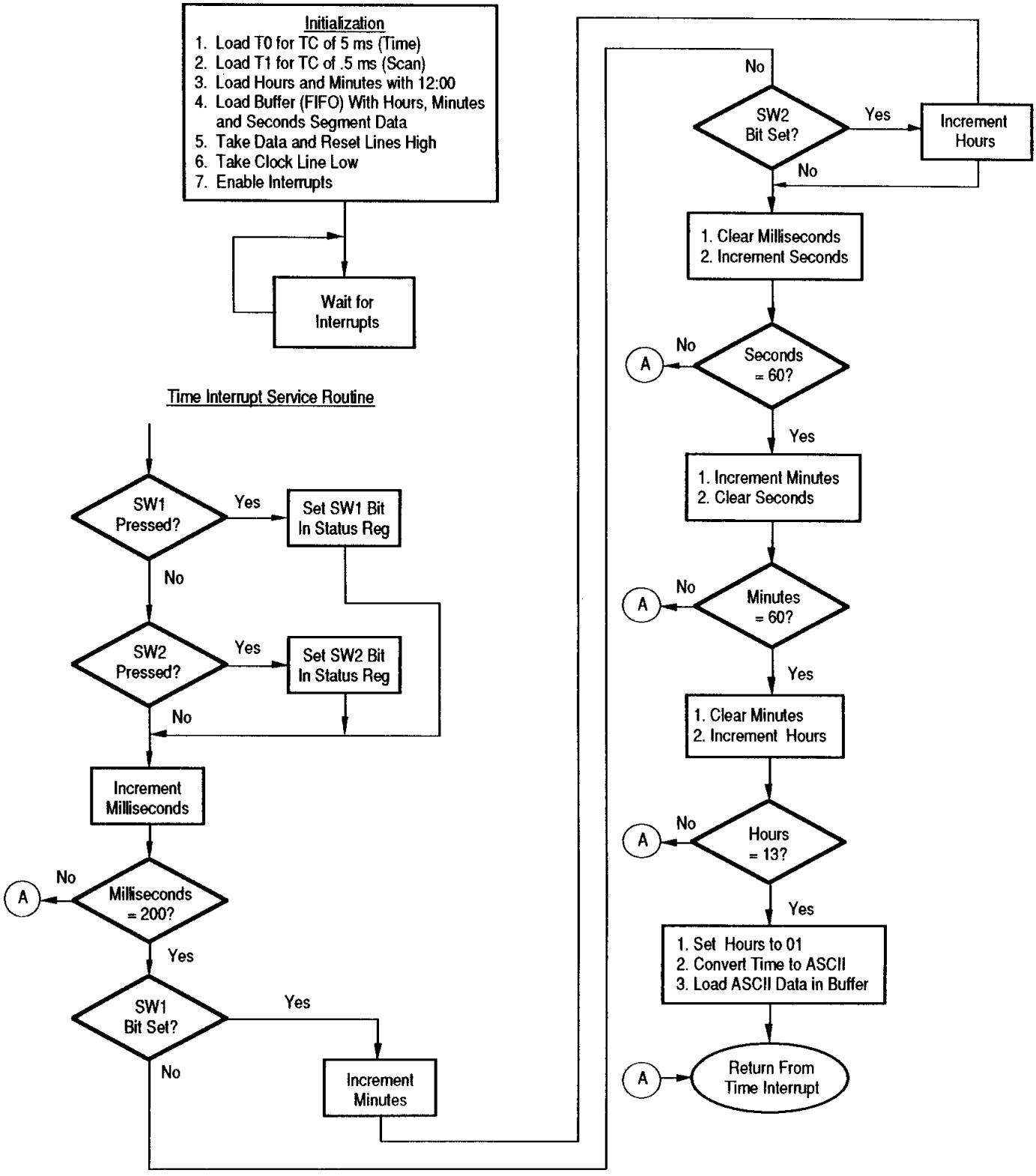

Figure 3a. Display Flowchart

THE SOFTWARE (Continued)

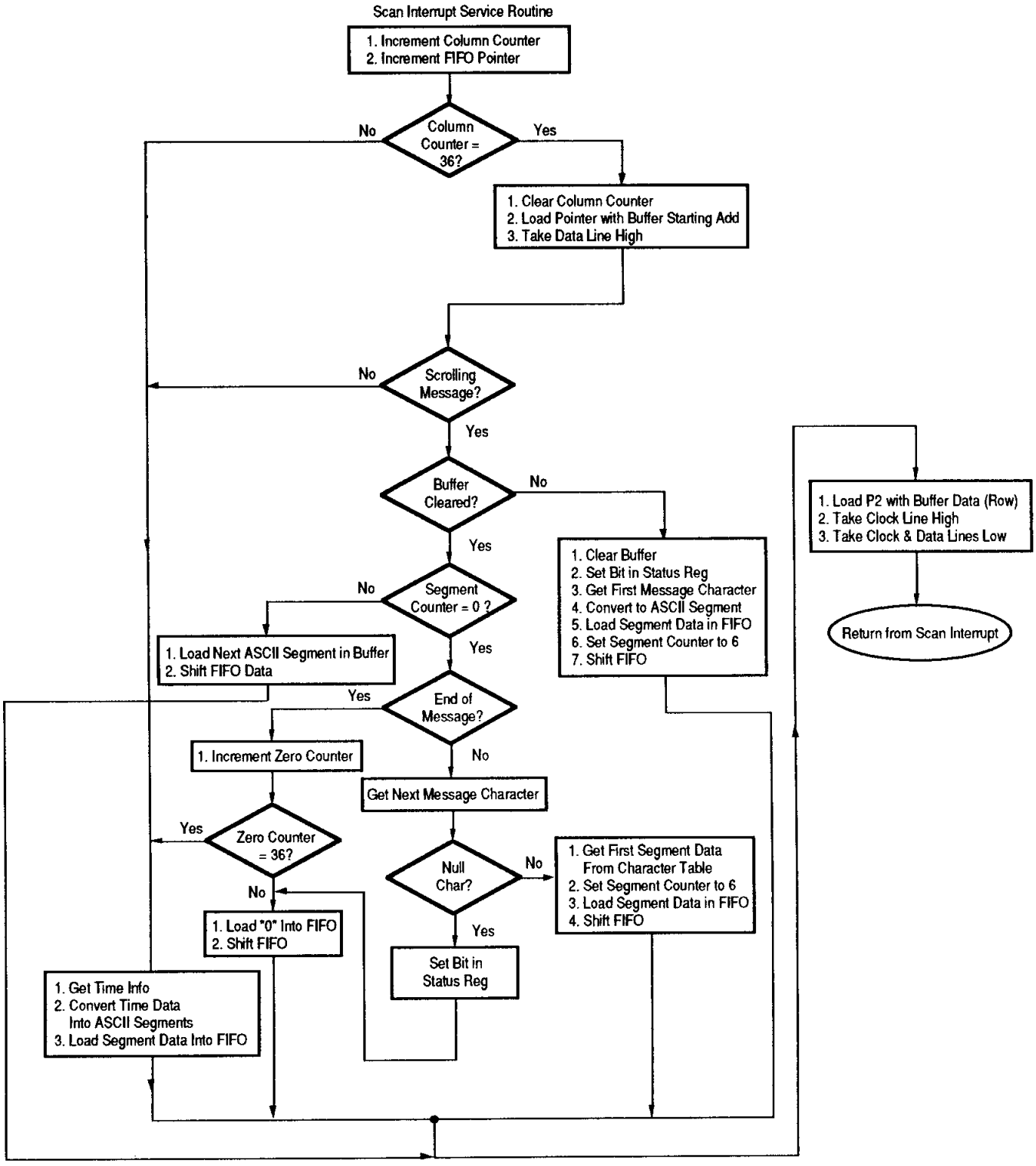

Figure 3b. Display Flowchart

At sixty-second intervals, the time display is blanked, and the internal message is scrolled across the display. The message is stored in an ASCII format. The individual ASCII characters index a look-up table, which converts the characters to a 6x7 format (first segment is a space). The software checks to see if all of the segments have been indexed at the beginning of the scan. If so, it then indexes the next character (Figures 4a and 4b).

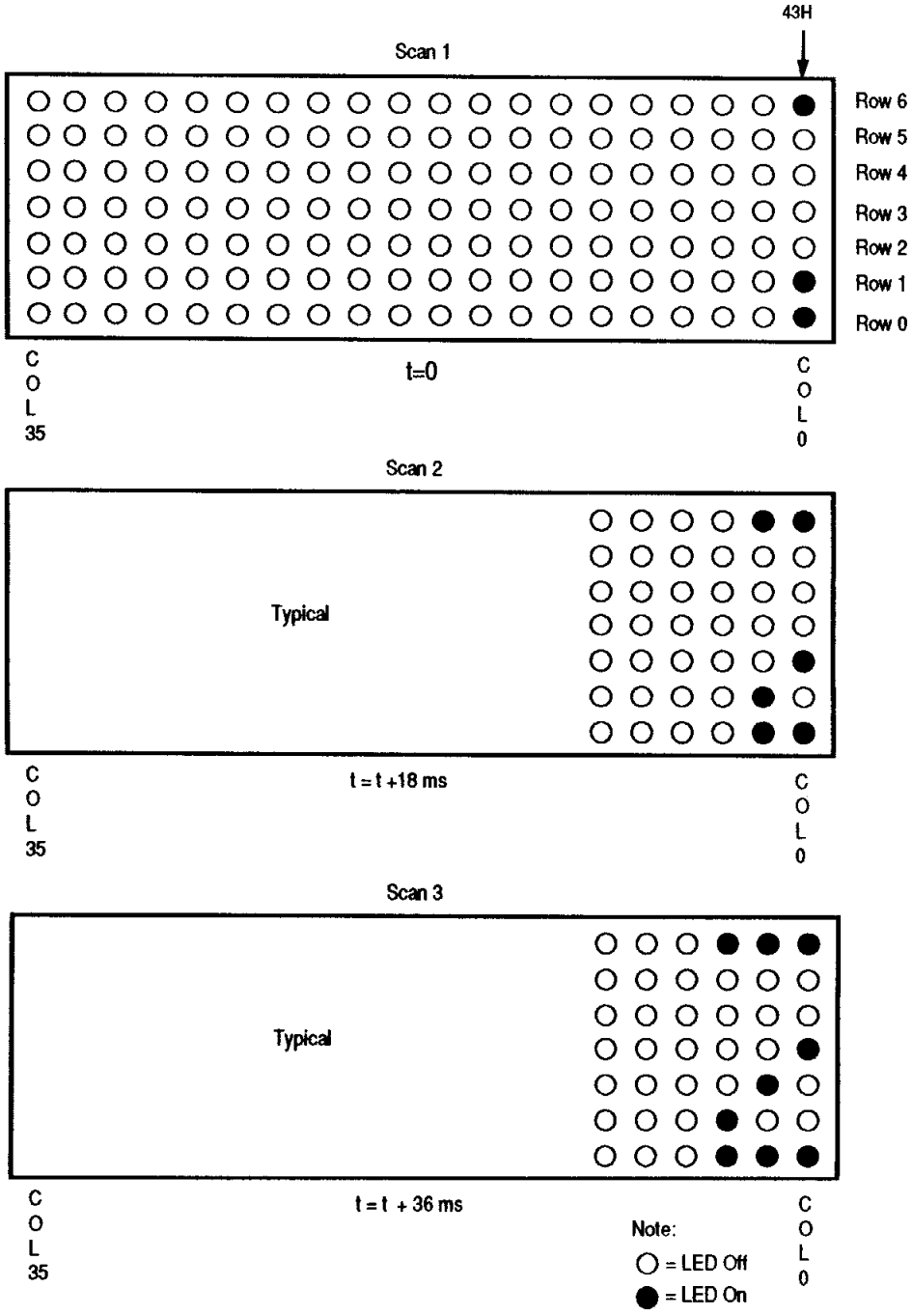

Figure 4a. Scrolling the Letter Z.

For scrolling messages, the display buffer acts like a FIFO (First In - First Out). The FIFO is cleared at power-up. When the internal message is being indexed, the character segments are queued up in the FIFO. The FIFO size is determined by the size of the display. At the end of each scan, the next character segment is indexed, and is stored at the bottom of the FIFO. The character segments are then shifted up the FIFO one location. This process continues until the entire message is displayed. At this time, a 0 is loaded into the FIFO at the beginning of each scan allowing the columns trailing the message to blank out. As the display is being scanned, the byte at each FIFO location is output at P2 as each column is turned on. The scrolling effect is created by shifting the FIFO data at the start of each scan (Figure 5).

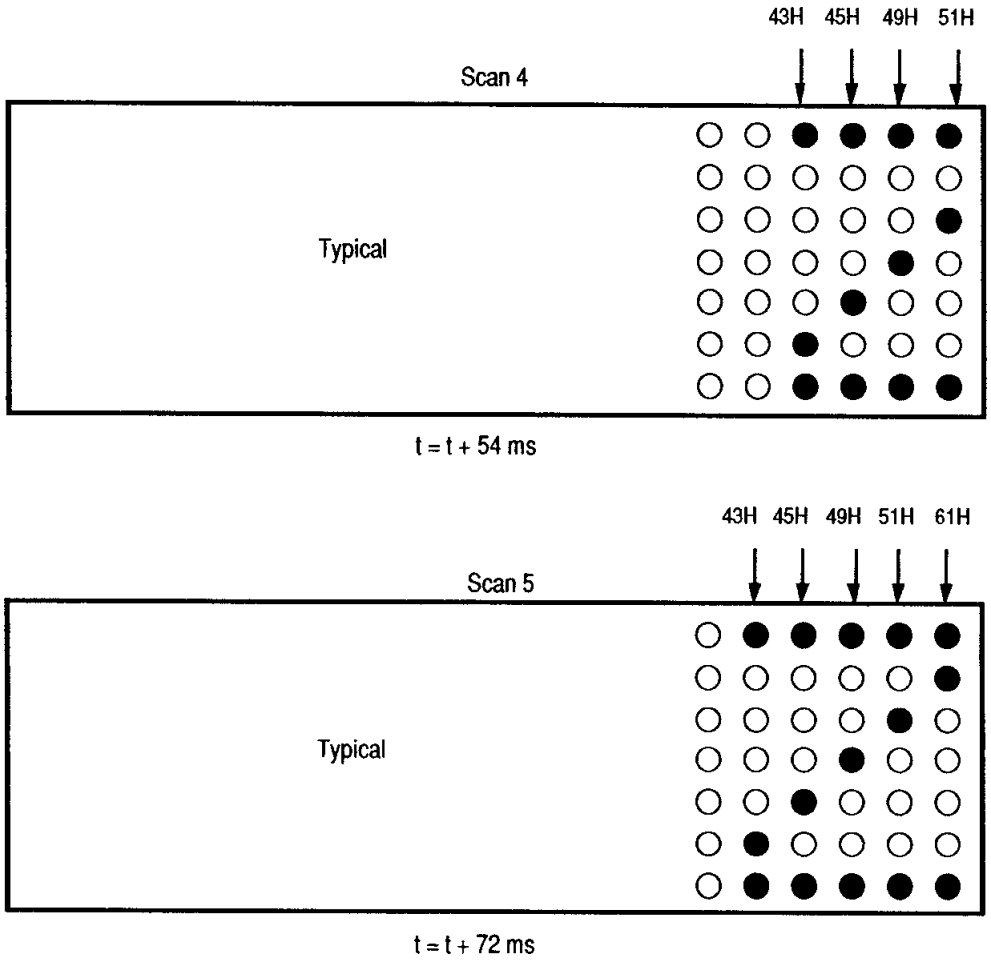

Figure 4b. Scrolling the Letter Z.

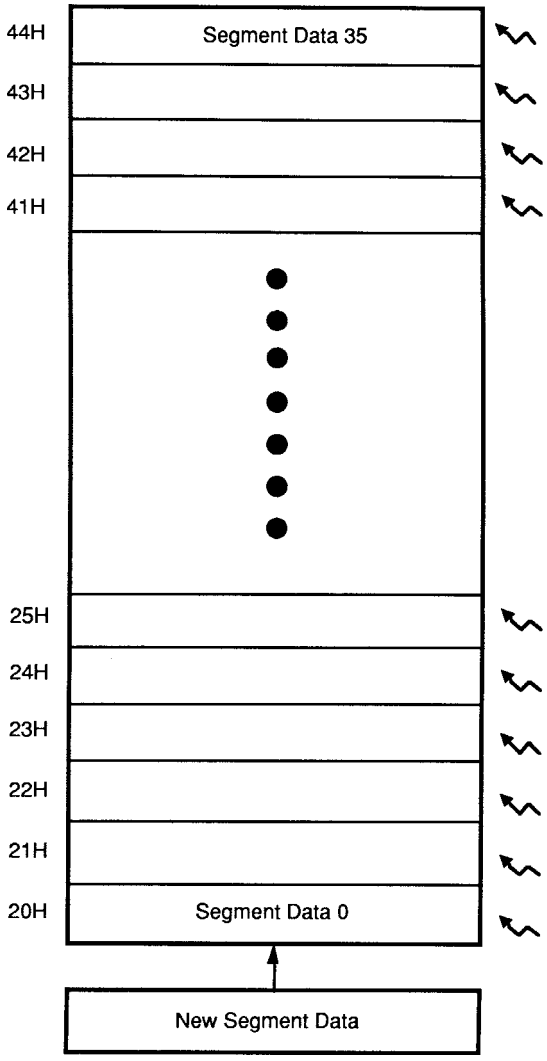

Figure 5. Shifting FIFO Data

APPENDIX

```
                        SCROLL1
                    LINE# --- SOURCE ---
                       1 ;*****************************************************************************;
                       2 ;                                                                             ;
                       3 ;   This is a scrolling LED display routine using the Zilog Z86C08, 'C17     ;
                       4 ;  18-pin CMOS processor. The processor's P2 port outputs the row data, and  ;
                       5 ;  the column data is clocked out of P0 into cascaded shift registers. It    ;
                       6 ;  has a real-time clock in software, and displays the time in hours, min-   ;
                       7 ;  utes, and seconds. At every 60-second interval, it blanks the screen and  ;
                       8 ;  displays an internal message. The message is stored in ROM, and can be up ;
                       9 ;  to 127 characters in length. After scrolling the message, the program re- ;
                      10 ;  sumes the time display, until the next 60 seconds.                         ;
                      11 ;   The size of the display is 36-columns long, enough to display six char-  ;
                      12 ;  acters. The display can be made longer, but the refresh time may need ad- ;
                      13 ;  justing to eliminate flicker. Current refresh time is 500 microseconds,   ;
                      14 ;  using a crystal frequency of 8.00 MHz.                                   ;
                      15 ;   There are two pushbutton switches that are read to adjust the hours and  ;
                      16 ;  minutes. On power-up, the display will show 12:00.                        ;
                      17 ;   This program was written by Don Owen Newquist on 12-1-91.               ;
                      18 ;*****************************************************************************;
0000000000000000010   19 WORK_REG       .equ    10h
  abs 00000000        20 address_hi      .equ    r0
  abs 00000001        21 address_lo      .equ    r1
  abs 00000000        22 address         .equ    rr0
  abs 00000002        23 pointer         .equ    r2
  abs 00000004        24 zero_count      .equ    r4
  abs 00000005        25 temp_1          .equ    r5
  abs 00000006        26 temp_2          .equ    r6
  abs 00000007        27 temp_3          .equ    r7
  abs 00000008        28 col_count       .equ    r8
  abs 00000009        29 seq_count       .equ    r9
0000000000000000000   30 TIME_REG        .equ    00h
  abs 00000005        31 millisec        .equ    r5
  abs 00000006        32 seconds         .equ    r6
  abs 00000007        33 minutes         .equ    r7
  abs 00000008        34 hours           .equ    r8
  abs 00000009        35 seconds_lo      .equ    r9
  abs 0000000a        36 seconds_hi      .equ    r10
  abs 0000000b        37 minutes_lo      .equ    r11
  abs 0000000c        38 minutes_hi      .equ    r12
  abs 0000000d        39 hours_lo        .equ    r13
  abs 0000000e        40 hours_hi        .equ    r14
  abs 0000000f        41 sw_count        .equ    r15
0000000000000000004   42 STATUS          .equ    %04
0000000000000000005   43 MILLISEC        .equ    %05
0000000000000000006   44 SECONDS         .equ    %06
0000000000000000007   45 MINUTES         .equ    %07
0000000000000000008   46 HOURS           .equ    %08
0000000000000000009   47 SECONDS_LO      .equ    %09
000000000000000000a   48 SECONDS_HI      .equ    %0a
000000000000000000b   49 MINUTES_LO      .equ    %0b
000000000000000000c   50 MINUTES_HI      .equ    %0c
000000000000000000d   51 HOURS_LO        .equ    %0d
000000000000000000e   52 HOURS_HI        .equ    %0e
0000000000000000020   53 BUFFER          .equ    %20
                      54 ;
                      55 ;              STATUS  REG:    d4 d3 d2 d1 d0
                      56 ;                                |  |  |  |  |_ displaying message
                      57 ;                                |  |  |  |___ buffer cleared
```

```
                        58 ;                                  |  |  |_____ end of message
                        59 ;                                  |  |________ min sw pressed
                        60 ;                                  |___________ hrs sw pressed
                        61 ;
00000000                62                .org    00
00000000 0000           63                .word   00
00000002 0000           64                .word   00
00000004 0000           65                .word   00
00000006 0000           66                .word   00
00000008 Wwww           67                .word   time
0000000a Wwww           68                .word   shift
0000000c                69                .org    000ch
                        70 ;*******************************************************************************;
                        71 ;                Initialization                                                 ;
                        72 ;*******************************************************************************;
0000000c 3110           73 init:          srp     #WORK_REG
0000000e 8f             74                di                              ; disable int
0000000f e6f464         75                ld      t0,#%64                 ; 100 decimal
00000012 e6f5c9         76                ld      pre0,#11001001b         ; set t0 for 5 mS period
00000015 e6f27d         77                ld      t1,#125                 ;
00000018 e6f313         78                ld      pre1,#00010011b         ; set t1 for .5 mS period
0000001b e6f600         79                ld      p2m,#0                  ; outputs on p2
0000001e e6f701         80                ld      p3m,#1                  ; active on p2
00000021 e6f800         81                ld      p01m,#0                 ; outputs on p0
00000024 b000           82                clr     p0                      ; p0 low
00000026 b0ef           83                clr     sw_count                ; sw count = 0
00000028 e6f908         84                ld      ipr,#00001000b          ; make irq5 > irq3
0000002b e6fb30         85                ld      imr,#%30                ; enable irq4,irq5
0000002e e6ff80         86                ld      spl,#%80                ; set stack pointer
00000031 e6f10f         87                ld      tmr,#%0f                ; load and enable counters
00000034 2c04           88                ld      pointer,#%04            ; point to time regs
00000036 fc0c           89                ld      r15,#12                 ; six locations
00000038 b1e2           90 clear_reg:     clr     @pointer                ; clear ram
0000003a 2e             91                inc     pointer                 ;
0000003b fafb           92                djnz    r15,clear_reg           ; continue until all zero
0000003d d6Wwww         93                call    clear_buffer            ; clear buffer
00000040 e60812         94                ld      HOURS,#%12              ; start time at 12:00
00000043 d6Wwww         95                call    load_time               ; load starting time
00000046 2c20           96                ld      pointer,#BUFFER         ; start at top of buffer
00000048 e60003         97                ld      p0,#00000011b           ; take data and clear hi
0000004b 9f             98 main_loop:     ei                              ; enable interrupts
0000004c 8bfd           99                jr      main_loop               ;
                       100
                       101 ;*******************************************************************************;
                       102 ;            This interrupt routine updates the time                            ;
                       103 ;*******************************************************************************;
0000004e 70fd          104 time:          push    rp                      ; save current reg pointer
00000050 3100          105                srp     #TIME_REG               ; point to time reg group
00000052 d6Wwww        106                call    test_sw                 ; look at time-set switches
00000055 5e            107                inc     millisec                ; increment millisec reg
00000056 a6e5c8        108                cp      millisec,#200           ; one second?
00000059 7dWwww        109                jp      ult,exit_time           ; exit if not
0000005c b0e5          110                clr     millisec                ; set to zero
0000005e 760408        111                tm      STATUS,#00001000b       ; sw 1 pressed?
00000061 6b**          112                jr      z,test_sw2              ; no
00000063 060701        113                add     MINUTES,#1              ;
00000066 4007          114                da      MINUTES                 ; convert to bcd
00000068 a60760        115                cp      MINUTES,#%60            ; sixty minutes?
0000006b 7b**          116                jr      ult,inc_seconds         ;
0000006d b007          117                clr     MINUTES                 ;
```

APPENDIX (Continued)

```
0000006f 8b**           118                 jr      inc_seconds             ;
00000071 760410         119 test_sw2:       tm      STATUS,#00010000b       ; sw 2 pressed?
00000074 6b**           120                 jr      z,inc_seconds           ; no
00000076 060801         121                 add     HOURS,#1                ;
00000079 4008           122                 da      HOURS                   ;
0000007b a60813         123                 cp      HOURS,#%13              ; 1:00?
0000007e 7b**           124                 jr      ult,inc_seconds         ;
00000080 e60801         125                 ld      HOURS,#1                ;
00000083 06e601         126 inc_seconds:    add     seconds,#1              ; increment sec
00000086 40e6           127                 da      seconds                 ; convert to bcd
00000088 a6e660         128                 cp      seconds,#%60            ; sixty seconds?
0000008b 7b**           129                 jr      ult,exit_time           ; no
0000008d 460401         130                 or      STATUS,#00000001b       ; set message flag
00000090 b0e6           131                 clr     seconds                 ; set to zero
00000092 06e701         132                 add     minutes,#1              ; inc minutes
00000095 40e7           133                 da      minutes                 ; convert to bcd
00000097 a6e760         134                 cp      minutes,#%60            ; sixty minutes?
0000009a 7b**           135                 jr      ult,exit_time           ; not yet
0000009c b0e7           136                 clr     minutes                 ; set to zero
0000009e 06e801         137 set_hrs:        add     hours,#1                ; inc hours
000000a1 40e8           138                 da      hours                   ; convert to bcd
000000a3 a6e813         139                 cp      hours,#%13              ; 1:00?
000000a6 7b**           140                 jr      ult,exit_time           ; exit
000000a8 8c01           141                 ld      hours,#1                ; set to 1:00
000000aa d6wwww         142 exit_time:      call    time_convert            ; convert to individual chars
000000ad d6wwww         143                 call    load_time               ; load new values into buffer
000000b0 50fd           144                 pop     rp                      ; return to orig reg pointer
000000b2 bf             145                 iret                            ; return from int
                        146
                        147 ;*******************************************************************************;
                        148 ;               This routine converts the seconds, minutes, and hours bcd     ;
                        149 ;               data into units and tens-of-units for displaying              ;
                        150 ;*******************************************************************************;
000000b3 e40609         151 time_convert:   ld      SECONDS_LO,SECONDS      ; transfer contents
000000b6 e4060a         152                 ld      SECONDS_HI,SECONDS      ;
000000b9 56090f         153                 and     SECONDS_LO,#%0f         ; keep only lower bits
000000bc f00a           154                 swap    SECONDS_HI              ; swap nibbles
000000be 560a0f         155                 and     SECONDS_HI,#%0f         ; keep only lower bits
000000c1 e4070b         156                 ld      MINUTES_LO,MINUTES      ; transfer contents
000000c4 e4070c         157                 ld      MINUTES_HI,MINUTES      ;
000000c7 560b0f         158                 and     MINUTES_LO,#%0f         ; keep only lower bits
000000ca f00c           159                 swap    MINUTES_HI              ; swap nibbles
000000cc 560c0f         160                 and     MINUTES_HI,#%0f         ; keep only lower bits
000000cf e4080d         161                 ld      HOURS_LO,HOURS          ; transfer contents
000000d2 e4080e         162                 ld      HOURS_HI,HOURS          ;
000000d5 560d0f         163                 and     HOURS_LO,#%0f           ; keep only lower bits
000000d8 f00e           164                 swap    HOURS_HI                ;
000000da 560e0f         165                 and     HOURS_HI,#%0f           ; keep only lower bits
000000dd af             166                 ret                             ; return
                        167
                        168 ;*******************************************************************************;
                        169 ;           This subroutine loads the time data into the RAM buffer           ;
                        170 ;*******************************************************************************;
                        171
000000de 70fd           172 load_time:      push    rp                      ; save current reg pointer
000000e0 3110           173                 srp     #WORK_REG               ; point to working reg
000000e2 ac09           174                 ld      r10,#%09                ; load starting time reg
000000e4 bc20           175                 ld      r11,#BUFFER             ; load starting buffer reg
000000e6 e3ca           176 load_table:     ld      r12,@r10                ; load contents
000000e8 0c**           177                 ld      address_hi,#^hb number_table; load hi address of table
```

```
000000ea 1c**           178                ld      address_lo,#^lb number_table; load lo address of table
000000ec a6ec00         179                cp      r12,#0                ; is it zero?
000000ef 6b**           180                jr      eq,no_index           ; if yes, don't step thru table
000000f1 a0e0           181 index_num:     incw    address               ; step thru table
000000f3 a0e0           182                incw    address               ;
000000f5 a0e0           183                incw    address               ;
000000f7 a0e0           184                incw    address               ;
000000f9 a0e0           185                incw    address               ;
000000fb a0e0           186                incw    address               ;
000000fd caf2           187                djnz    r12,index_num         ; index if not zero
000000ff cc06           188 no_index:      ld      r12,#6                ; load no of segments
00000101 c3b0           189 load_time_reg: ldci    @r11,@address         ; load into reg
00000103 cafc           190                djnz    r12,load_time_reg     ; keep going if not zero
00000105 ae             191                inc     r10                   ; inc reg location
00000106 a6ea0f         192                cp      r10,#%0f              ; at ending reg?
00000109 7bdb           193                jr      ult,load_table        ; go again
0000010b e62b3a         194                ld      %2b,#%3a              ; put colon here
0000010e e6373a         195                ld      %37,#%3a              ; and here too
00000111 a60e00         196                cp      HOURS_HI,#0           ; zero?
00000114 eb**           197                jr      ne,load_time_ret      ;
00000116 ac00           198                ld      r10,#0                ; load zeros on last 5 columns
00000118 bc3e           199                ld      r11,#%3e              ; starting here
0000011a cc05           200                ld      r12,#5                ; blank out leading zero
0000011c f3ba           201 leading_loop:  ld      @r11,r10              ;
0000011e be             202                inc     r11                   ; blank last five columns (hrs)
0000011f cafb           203                djnz    r12,leading_loop      ; step thru ram
00000121 50fd           204 load_time_ret: pop     rp                    ; return to time regs
00000123 af             205                ret                           ; return to caller
                        206
                        207
                        208 ;*******************************************************************************;
                        209 ;      This subroutine checks to see if the time-set switches are pressed.     ;
                        210 ;*******************************************************************************;
00000124 a803           211 test_sw:       ld      r10,p3                ; load sw data
00000126 60ea           212                com     r10                   ; 1's complement
00000128 56ea03         213                and     r10,#03               ; mask off upper bits
0000012b 760301         214                tm      p3,#1                 ; min pressed?
0000012e 6b**           215                jr      z,test_hrs            ; no
00000130 fe             216                inc     sw_count              ; inc counter
00000131 a6ef02         217                cp      sw_count,#2           ; debouncd?
00000134 7b**           218                jr      ult,exit_sw           ; not yet
00000136 460408         219                or      STATUS,#00001000b     ; set bit
00000139 8b**           220                jr      exit_sw               ; exit
0000013b 760304         221 test_hrs:      tm      p3,#4                 ; hrs pressed?
0000013e 6b**           222                jr      z,clear_sw            ; no
00000140 fe             223                inc     sw_count              ; inc debounce counter
00000141 a6ef02         224                cp      sw_count,#2           ; debounced?
00000144 7b**           225                jr      ult,exit_sw           ; not yet
00000146 460410         226                or      STATUS,#00010000b     ; set bit
00000149 af             227 exit_sw:       ret                           ; return to caller
0000014a 5604e7         228 clear_sw:      and     STATUS,#11100111b     ; reset sw status bits
0000014d b0ef           229                clr     sw_count              ; reset debounce counter
0000014f af             230                ret                           ; return
                        231
                        232 ;*******************************************************************************;
                        233 ;      This is the timer interrupt routine.  When T1 hits TC, column data is   ;
                        234 ;     shifted one.                                                              ;
                        235 ;*******************************************************************************;
00000150 70fd           236 shift:         push    rp                    ; save reg pointer
00000152 3110           237                srp     #WORK_REG             ; point to working reg
```

APPENDIX (Continued)

```
00000154 2e              238                 inc     pointer              ; point to next location in ram
00000155 8e              239                 inc     col_count            ;
00000156 a6e824          240                 cp      col_count,#36        ; thirty-six columns?
00000159 7b**            241                 jr      ult,test_flag        ; end of screen?
0000015b b0e8            242                 clr     col_count            ; reset col number
0000015d 2c20            243                 ld      pointer,#BUFFER      ; start at beginning of buffer
0000015f 460002          244                 or      p0,#00000010b        ; take data line high
00000162 760401          245 test_flag:      tm      STATUS,#00000001b    ; time to scroll message?
00000165 6b**            246                 jr      z,continue           ; no, display time
00000167 d6Wwww          247                 call    load_message         ; load message data
0000016a 8b**            248                 jr      load_row             ; load row data
0000016c d6R000+00de,    249 continue:       call    load_time            ; get time data
0000016f e352            250 load_row:       ld      temp_1,@pointer      ; load contents
00000171 60e5            251                 com     temp_1               ;
00000173 5902            252                 ld      p2,temp_1            ; out to port
00000175 460004          253 clock_it:       or      p0,#00000100b        ; take clock hi
00000178 560001          254                 and     p0,#00000001b        ; take clock and data low
0000017b 50fd            255                 pop     rp                   ; restore reg pointer
0000017d bf              256                 iret                         ;
                         257 ;*****************************************************************************;
                         258 ;               This routine loads the message into the buffer                ;
                         259 ;*****************************************************************************;
0000017e 760402          260 load_message:   tm      STATUS,#00000010b    ; clear buffer?
00000181 eb**            261                 jr      nz,req_scroll        ;
00000183 d6Wwww          262                 call    clear_buffer         ; clear contents of buffer
00000186 460402          263                 or      STATUS,#00000010b    ; set bit
00000189 0c**            264                 ld      address_hi,#^hb mess_beg     ;
0000018b 1c**            265                 ld      address_lo,#^lb mess_beg     ;
0000018d d6Wwww          266                 call    get_ascii            ; get first character
00000190 9c06            267                 ld      seg_count,#6         ; start out with six segs
00000192 8b**            268                 jr      load_nert_seg        ;
00000194 a6e900          269 req_scroll:     cp      seg_count,#0         ; six segments loaded?
00000197 eb**            270                 jr      ne,load_next_seg     ; no
00000199 760404          271                 tm      STATUS,#00000100b    ; end of message?
0000019c 6b**            272                 jr      z,load_next_char     ; load next ascii char
0000019e 6c00            273                 ld      temp_2,#0            ;
000001a0 d6Wwww          274                 call    load_fifo            ; load data
000001a3 4e              275                 inc     zero_count           ; inc no of locations cleared
000001a4 a6e424          276                 cp      zero_count,#36       ; 36 locations yet?
000001a7 7b**            277                 jr      ult,scroll_return    ; load segment data
000001a9 b004            278                 clr     STATUS               ; display time now
000001ab b0e4            279                 clr     zero_count           ; clear counter
000001ad 8b**            280                 jr      scroll_return        ; return
000001af 9c06            281 load_next_char: ld      seg_count,#6         ; load 6 segments/character
000001b1 a0e0            282                 incw    address              ; point to next ascii char
000001b3 d6Wwww          283                 call    get_ascii            ; get next character
000001b6 c26c            284 load_next_seg:  ldc     temp_2,@rr12         ; load seg from ascii table
000001b8 d6Wwww          285                 call    load_fifo            ; shift the data
000001bb a0ec            286                 incw    rr12                 ; point to next seg in table
000001bd 00e9            287                 dec     seg_count            ; decrement segment count
000001bf af              288 scroll_return:  ret                          ;
                         289
                         290 ;*****************************************************************************;
                         291 ;       This subroutine loads and shifts the RAM buffer                       ;
                         292 ;*****************************************************************************;
000001c0 fc24            293 load_fifo:      ld      r15,#36              ; load number of columns
000001c2 2c20            294                 ld      pointer,#BUFFER      ; load starting buffer reg
000001c4 e372            295 shift_fifo:     ld      temp_3,@pointer      ; get data
000001c6 f326            296                 ld      @pointer,temp_2      ; load new data
000001c8 68e7            297                 ld      temp_2,temp_3        ; transfer bytes
```

```
000001ca 2e              298              inc     pointer              ; next buffer address
000001cb faf7            299              djnz    r15,shift_fifo       ; load all columns
000001cd af              300              ret                          ; return to caller
                         301 ;*****************************************************************************;
                         302 ;  This subroutine indexes the character table and fetches the segment data. ;
                         303 ;*****************************************************************************;
000001ce cc**            304 get_ascii:   ld      rr12,#^hb char_table ; load starting add of table
000001d0 dc**            305              ld      rr13,#^lb char_table ;
000001d2 c2ea            306              ldc     r14,@rr10            ; load ascii data
000001d4 a6ee00          307              cp      r14,#0               ; end of message?
000001d7 eb**            308              jr      ne,load_next         ; no
000001d9 460404          309              or      STATUS,#00000100b    ; set bit to mark mess end
000001dc 8b**            310              jr      mess_return          ; return
000001de 26ee20          311 load_next:   sub     r14,#%20             ; subtract 20h
000001e1 a6ee00          312              cp      r14,#0               ; is it a space?
000001e4 6b**            313              jr      eq,mess_return       ; if yes, don't index table
000001e6 a0ec            314 index_table: incw    rr12                 ; index table
000001e8 a0ec            315              incw    rr12                 ;
000001ea a0ec            316              incw    rr12                 ;
000001ec a0ec            317              incw    rr12                 ;
000001ee a0ec            318              incw    rr12                 ;
000001f0 a0ec            319              incw    rr12                 ;
000001f2 eaf2            320              djnz    r14,index_table      ; keep going if not zero
000001f4 af              321 mess_return: ret                          ;
                         322
                         323 ;*****************************************************************************;
                         324 ;           This subroutine clears the RAM buffer.                            ;
                         325 ;*****************************************************************************;
000001f5 fc24            326 clear_buffer: ld     r15,#36              ; get no of columns
000001f7 2c20            327              ld      pointer,#BUFFER      ; starting point of buffer
000001f9 b1e2            328 clear_loop:  clr     @pointer             ; clear contents
000001fb 2e              329              inc     pointer              ; next location
000001fc fafb            330              djnz    r15,clear_loop       ;  till out of columns
000001fe af              331              ret                          ; return to caller
                         332
                         333 ;*****************************************************************************;
                         334 ;          Message data area. Message can be up to 80 ASCII characters       ;
                         335 ;*****************************************************************************;
00000200                 336              .org    200h
00000200                 337 mess_beg:
00000200 5448495320444953 338             .asciz  'THIS DISPLAY IS POWERED BY ZILOG!'
00000208 504c415920495320
00000210 504f574552454420
00000218 4259205a494c4f47
00000220 2100
                         339
                         340 ;*****************************************************************************;
                         341 ;       This is the ASCII character look-up table.                           ;
                         342 ;*****************************************************************************;
00000280                 343              .org    280h
00000280                 344 char_table:
00000280 000000000000    345              .byte   0,0,0,0,0,0          ; space
00000286 0000007d00      346              .byte   0,0,0,7DH,0          ; !
0000028b 000070007000    347              .byte   0,0,70H,0,70H,0      ; "
00000291 00147f147f14    348              .byte   0,14H,7FH,14H,7FH,14H ; #
00000297 00122a7f2a24    349              .byte   0,12H,2AH,7FH,2AH,24H ; $
0000029d 006264081323    350              .byte   0,62H,64H,08H,13H,23H ; %
000002a3 003649350205    351              .byte   0,36H,49H,35H,02H,05H ; &
000002a9 000000700000    352              .byte   0,00,00,70H,00,00    ; '
000002af 001c22410000    353              .byte   0,1CH,22H,41H,0,0    ; (
```

APPENDIX (Continued)

```
000002b5 00000041221c    354                .byte    0,0,0,41H,22H,1CH     ; )
000002bb 0022147f1422    355                .byte    0,22H,14H,7FH,14H,22H ; *
000002c1 0008083e0808    356                .byte    0,08H,08H,3EH,08H,08H ; +
000002c7 000001060000    357                .byte    0,0,1,6,0,0           ; ,
000002cd 000808080808    358                .byte    0,8,8,8,8,8           ; -
000002d3 000000010000    359                .byte    0,0,0,1,0,0           ; .
000002d9 000204081020    360                .byte    0,2,4,8,10H,20H       ; /
                         361 ;numbers
000002df 003e4549513e    362 number_table:  .byte    0,3EH,45H,49H,51H,3EH ; 0
000002e5 0000217f0100    363                .byte    0,0,21H,7FH,01,0      ; 1
000002eb 002345494931    364                .byte    0,23H,45H,49H,49H,31H ; 2
000002f1 004241495966    365                .byte    0,42H,41H,49H,59H,66H ; 3
000002f7 000c14247f04    366                .byte    0,0CH,14H,24H,7FH,04H ; 4
000002fd 00725151514e    367                .byte    0,72H,51H,51H,51H,4EH ; 5
00000303 001e29494946    368                .byte    0,1EH,29H,49H,49H,46H ; 6
00000309 004047485060    369                .byte    0,40H,47H,48H,50H,60H ; 7
0000030f 003649494936    370                .byte    0,36H,49H,49H,49H,36H ; 8
00000315 003149494a3c    371                .byte    0,31H,49H,49H,4AH,3CH ; 9
                         372 ;     MORE SPECIAL CHARACTERS
0000031b 000000140000    373                .byte    0,0,0,14H,0,0         ; :
00000321 000001160000    374                .byte    0,0,1,16H,0,0         ; ;
00000327 000814224100    375                .byte    0,8,14H,22H,41H,0     ; <
0000032d 001414141414    376                .byte    0,14H,14H,14H,14H,14H ; =
00000333 000041221408    377                .byte    0,0,41H,22H,14H,08H   ; >
00000339 0020404d5020    378                .byte    0,20H,40H,4DH,50H,20H ; ?
                         379 ;     AT SIGN AND UPPERCASE LETTERS             ;
0000033f 003e415d4d39    380                .byte    0,3EH,41H,5DH,4DH,39H ; @
00000345 001f2444241f    381                .byte    0,1FH,24H,44H,24H,1FH ; A
0000034b 007f49494936    382                .byte    0,7FH,49H,49H,49H,36H ; B
00000351 003e41414122    383                .byte    0,3EH,41H,41H,41H,22H ; C
00000357 007f4141413e    384                .byte    0,7FH,41H,41H,41H,3EH ; D
0000035d 007f49494941    385                .byte    0,7FH,49H,49H,49H,41H ; E
00000363 007f48484840    386                .byte    0,7FH,48H,48H,48H,40H ; F
00000369 003e41414547    387                .byte    0,3EH,41H,41H,45H,47H ; G
0000036f 007f0808087f    388                .byte    0,7FH,08H,08H,08H,7FH ; H
00000375 0000417f4100    389                .byte    0,00H,41H,7FH,41H,00H ; I
0000037b 00020101017e    390                .byte    0,02H,01H,01H,01H,7EH ; J
00000381 007f08142241    391                .byte    0,7FH,08H,14H,22H,41H ; K
00000387 007f01010101    392                .byte    0,7FH,01H,01H,01H,01H ; L
0000038d 007f2018207f    393                .byte    0,7FH,20H,18H,20H,7FH ; M
00000393 007f1008047f    394                .byte    0,7FH,10H,08H,04H,7FH ; N
00000399 003e4141413e    395                .byte    0,3EH,41H,41H,41H,3EH ; O
0000039f 007f48484830    396                .byte    0,7FH,48H,48H,48H,30H ; P
000003a5 003e4145423d    397                .byte    0,3EH,41H,45H,42H,3DH ; Q
000003ab 007f484c4a31    398                .byte    0,7FH,48H,4CH,4AH,31H ; R
000003b1 003249494926    399                .byte    0,32H,49H,49H,49H,26H ; S
000003b7 0040407f4040    400                .byte    0,40H,40H,7FH,40H,40H ; T
000003bd 007e0101017e    401                .byte    0,7EH,01H,01H,01H,7EH ; U
000003c3 007c0201027c    402                .byte    0,7CH,02H,01H,02H,7CH ; V
000003c9 007f020c027f    403                .byte    0,7FH,02H,0CH,02H,7FH ; W
000003cf 006314081463    404                .byte    0,63H,14H,08H,14H,63H ; X
000003d5 0060100f1060    405                .byte    0,60H,10H,0FH,10H,60H ; Y
000003db 004345495161    406                .byte    0,43H,45H,49H,51H,61H ; Z
000003e1 007f7f414141    407                .byte    0,7FH,7FH,41H,41H,41H ; [
000003e7 002010080402    408                .byte    0,20H,10H,08H,04H,02H ; \
000003ed 004141417f7f    409                .byte    0,41H,41H,41H,7FH,7FH ; ]
000003f3 000408100804    410                .byte    0,04H,08H,10H,08H,04H ; ^
000003f9 000101010101    411                .byte    0,01H,01H,01H,01H,01H ; -
                         412
                         413                .end
```

Zilog, Inc. 210 East Hacienda Ave.
Campbell, CA 95008-6600
Telephone (408) 370-8000
Telex 910-338-7621
FAX 408 370-8056
Internet: http://www.zilog.com