Application Note

Timekeeping with Z8 Microcontrollers

AN003401-Z8X0500

ZiLOG WORLDWIDE HEADQUARTERS • 910 E. HAMILTON AVENUE • CAMPBELL, CA 95008
TELEPHONE: 408.558.8500 • FAX: 408.558.8300 • www.ZiLOG.com

This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

ZiLOG Worldwide Headquarters
910 E. Hamilton Avenue
Campbell, CA 95008
Telephone: 408.558.8500
Fax: 408.558.8300
www.ZiLOG.com

Windows is a registered trademark of Microsoft Corporation.

Information Integrity

The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

Document Disclaimer

© 2000 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.

Table of Contents

Acknowledgements

Project Engineer

Mark Thissen

Timekeeping with Z8 Microcontrollers

General Overview

Many microcontroller applications track time by using what is commonly called a Real Time Clock (RTC). For example, the controller may be required to turn a solenoid switch or a relay on or off at an appropriate time. Or, the controller may monitor an input and return a value at a defined time. Whatever the requirement, the use of an RTC can be the answer. From the code that develops an RTC using the microcontroller crystal, a battery-powered watch can be developed by using a different lower-powered device with a low-power oscillator and the watch-dog option.

Discussion

Theory of Operation

There are many ways to accomplish the task of real timekeeping. The first portion of this discussion focuses on implementing a real time clock based on the 60-Hz line frequency of a 120 VAC input.

In this example, a 120 VAC input develops the power supply and a digital input to Port 3 (2). The input to Port 3 (2) generates a falling edge interrupt (IRQ0) that becomes the time base. A transformer with a 6.3 VCT secondary generates the DC power and the AC input to Port 3 (2). There are different designs of power supplies, and the schematic pages of this application note illustrate two of those designs. Choose an elaborate or a simple one, depending on the requirements. Take the secondary waveform voltage, divide the voltage across a pair of 100-KΩ resistors, and feed that signal directly into the Port 3 (2) digital input. This signal generates 60 falling-edge interrupts per second. From this time base, increment the various registers to track the time (milliseconds, seconds, minutes, hours). The frequency on the AC lines is accurate and works well for this application. Not only does this application track the time, but it also tracks a.m. and p.m.

Each time a minute change is reached, clock new data into the serial LED display. Start with clocking a high data (START) bit, followed by 34 bits of data representing the four seven-segment displays, the four decimal points between the segments, and the two external LED option bits. Table 1 indicates the bit orders.

Table 1. LED Option Bits

Bit #	Digit #	Segment	Bit #	Digit #	Segment
1	1	A	18	3	B
2	1	B	19	3	C
3	1	C	20	3	D
4	1	D	21	3	E
5	1	E	22	3	F
6	1	F	23	3	G
7	1	G	24	3	DP
8	1	DP	25	4	A
9	2	A	26	4	B
10	2	B	27	4	C
11	2	C	28	4	D
12	2	D	29	4	E
13	2	E	30	4	F
14	2	F	31	4	G
15	2	G	32	4	DP
16	2	DP	33	—	PIN 1
17	3	A	34	—	PIN 2

Finally, clock out one low data (STOP) bit so the display synchronously generates the load signal, followed by the reset signal, to ready the device for the next 36 bits. Figures 1 and 2 illustrate the timing relationships of the Three Five Systems device. Figure 3 contains a block diagram of the device.

The next part of this discussion focuses on implementing a real time clock based on the internal frequency of the microcontroller. In this example, an 8-MHz crystal drives the Z8 microcontroller. The internal time base fed into the two counter timers is 1 MHz. To derive this time, divide the Z8's internal frequency by 2. The 4-MHz frequency is further divided internally by 4 before being fed into the counter timers. Use the counter timer registers, in combination with the prescaler registers, to further divide the timebase down to the interrupt timing required. In this case, timer0 interrupts every 5 ms, while timer1 interrupts every 1 ms.

- Timer0 (IRQ4) interrupts every 5 ms and tracks the time. The timer updates the time registers with additional subroutines that check the input switches for time change data and convert the raw binary data into binary coded decimal format for the display. The 16-bit interrupt vector for this routine is loaded into ROM at locations `0x08` and `0x09`.

- Timer1 (IRQ5) interrupts every 1 ms and returns data to the LED segments, one segment per interrupt. The 16-bit interrupt vector for this routine is loaded into ROM at locations 0x0A and 0x0B. The software provides this interrupt priority, so the muxed frequency display update is not delayed. Each seven-segment display is updated approximately every 4 ms.

The program counts in five-millisecond increments and, when a count of (`5ms x 100`) or 5/10 of a second is reached, updates the half-second count. From that data, update the seconds, minutes, and hours, as appropriate. Toggle the output to the colon every half-second. At the same time, interrupt the controller every one millisecond to refresh the output to the seven-segment LEDs, one display at a time.

The common anodes of the seven-segment LEDs are driven one segment per interrupt from Port 0, accomplished using the `display_pointer`, or r4. The `display_pointer` is rotated right after every update to align the data on Port 0 with the next segment to be turned on. The segment data lines are connected in series and driven from Port 2. To perform this process, use the pointer to get the raw number from memory, and use the raw number to increment the address in the LED data look-up table. Load the LED data to Port 2 using `led_data`, or r3. The colon bits are connected in parallel and driven from Port 2, bit 7. The switch inputs are monitored on Port 3.

The INIT subroutine is implemented one time (or initially) to initialize the device. The TIME_CONVERT subroutine is implemented to separate the 8-bit registers with eight bits of data into two 8-bit registers with four bits of data by ANDing away the unwanted four bits.

Summary

Both of these applications keep time effectively and are relatively simple to implement. They are also cost effective and can be adapted to several other microcontrollers with more I/Os and features. The applications can be adapted to any system that requires a low-cost way of tracking time.

Technical Support

Source Code

60Hz_Clock.asm Source File

```
;*****************************************************************
;*  This program demonstrates how to implement a simple
;*  Real Time Clock using a Z86E08 or another Z8 with at
```

```
* least five I/Os.  This code is from the Z8 App
* notes manual and was modified by Mark Thissen on 08/10/99.
*  This code has been modified to be compatible with ZDS vs. 2.12.
* The "60Hz_Clock.inc" must be assembled with this assembly file
*  in order to run.
*****************************************************************
*      Include section
*****************************************************************
include "60Hz_Clock.inc"
*****************************************************************
*      Interrupt Vectors
*****************************************************************

      vector reset = start        ;Start Vector 0Ch
      vector IRQ0  = time         ;16-bit address of IRQ0,
                                  ;labeled time
      vector IRQ1  = IRQ1         ;Dummy Vectors
      vector IRQ2  = IRQ2         ;  "      "
      vector IRQ3  = IRQ3         ;  "      "
      vector IRQ4  = IRQ4         ;  "      "
      vector IRQ5  = IRQ5         ;  "      "

      GLOBALS ON
```

60HZ Clock Program Main

```
*****************************************************************************
*                  60Hz Clock Program Main
*****************************************************************
start:
           di                ;Disable Interrupts
           ld    spl,#%80    ;Set Stack pointer to start @ 7Fh.
           clr   sph         ;Clear high byte of stack pointer
           call  INIT        ;Call init routine
main_loop:  ei               ;Enable interrupts
           jr    main_loop   ;Continuous loop


*****************************************************************
*  Function Name:           INIT
*  Returns:                 Initialized control registers and
*                           cleared memory registers
*  Entry Values (arguments): Include File data
*  Description:             This module simply initializes the
*                           device to have Port 0 as an output
*                           port and Port 3 as an input port.
*                           It also initializes several data mem-o
*                           ory locations, initializes interrupt
*                           IRQ0 as a digital falling edge from
*                           P32, initializes the time display to
*                           12:00 and initializes the segment
*                           of day to Post Meridiem (p.m.).
```

```
*
*  Notes
*****************************************************************

INIT:
           srp   #DELAY_REG       ;Set Register Pointer to 30h
           clr   slow_delay       ;Initialize slow_delay register
           srp   #TIME_REG        ;Set Register Pointer to 10h
           ld    p3m,#0           ;Set Port 3 inputs in digital
                                  ;mode
           ld    p01m,#04         ;Set Port 0 as output / internal
                                  ;Stack
           clr   p0               ;Clear Port 0 outputs
           clr   p3               ;Clear p3
           clr   irq              ;Ensure no pending interrupts
           ld    imr,#%01         ;Enable IRQ0
           ld    pointer,#CLEAR_MEM;start at ram location 18h
           ld    counter,#6       ;load counter with six
clear_reg: clr   @pointer         ;Clears six consecutive memory
                                  ;locations
           inc   pointer          ;inc pointer location
           djnz  counter,clear_reg ;decrement counter and clear
                                  ;ram location until 0
           ld    hours,#%12       ;Start at 12:00
           ld    AM_PM,#%FF       ;Start with PM (FF=PM,00=AM)
           ld    pointer,#hour_ptr;Start with hours register
           call  time_convert     ;Call time conversion subroutine
           call  load_time        ;Call time loading subroutine
           ret                    ;Return from subroutine to main

*****************************************************************
*     IRQ0 Interrupt Service Routine (time)
*     This interrupt service routine checks the time set switch,
*     regulates the speed at which the time is advance,
*     and updates the time registers.
*
*****************************************************************

time:      tm    p3,#2            ;Test time change switch
           jr    nz,inc_millisec  ;Switch pushed if zero
           srp   #DELAY_REG       ;Set Register Pointer to 30h
           cp    slow_delay,#35   ;36 seconds?
           jr    ugt,fast_run     ;yes, start faster time advance
           ld    r15,#%0    q     ;Init r15 to 0
           jp    next_run         ;go to next slow time advance
fast_run:  ld    r15,#%EF         ;start r15 with EFh for fast
                                  ;advance
next_run:  ld    r14,#%FF         ;Init r14 to FFh
slow_count:djnz  r14,slow_count   ;Decrement r14 until zero
           inc   r15              ;Increment r15
           cp    r15,#%FF         ;Is r15 FFh yet?
           jr    nz,next_run      ;r15!=FFh : Continue looping
           cp    slow_delay,#35   ;36 seconds yet?
           jr    ugt,no_inc       ;if greater than 35 do not
```

```
                                    ;increment
            inc   slow_delay        ;inc counter
no_inc:     srp   #TIME_REG         ;set register pointer back to 10h
            clr   millisec          ;Clear millisec register
            clr   seconds           ;Clear seconds register
            jr    inc_minutes       ;Increment the minutes register
                                    ;and continue
inc_millisec:srp #DELAY_REG         ;Set Register Pointer to 30h
            clr   slow_delay        ;clear slow_delay counter
            srp   #TIME_REG         ;set register pointer back to 10h
            inc   millisec          ;Add one to millisec count
            cp    millisec,#60      ;One Second?
            jr    ult,no_time_load;If less than, jump out
            clr   millisec          ;Else clear count
            inc   seconds           ;Increment seconds register
            cp    seconds,#60       ;60 Seconds?
                  jrult,no_time_load;If less exit
                  clrseconds        ;Else clear count
inc_minutes:add   minutes,#1        ;Add one to minutes count
                  daminutes         ;Convert to BCD
                  cpminutes,#%60    ;60 Minutes?
                 jrult,exit_time    ;If less exit
                  clrminutes        ;Else clear minutes count
            add   hours,#1          ;Add one to hours count
                  dahours           ;convert to BCD
                  cphours,#%12      ;Compare to 12
                  jrne,next_cp      ;Do not complement if less than
                  comAM_PM          ;Complement AM_PM register
next_cp:          cphours,#%13      ;1:00?
            jr    ult,exit_time     ;If not exit
            ld    hours,#1          ;Else make hours 1
exit_time: call   time_convert      ;Call conversion routine
            call  load_time         ;Call load time subroutine
no_time_load:iret                   ;Return from interrupt

;*****************************************************************
;  Function Name:           load_time
;  returns                  Properly loaded display
;  entry values (arguments) Raw time digits, data look up table
;  Description:             This subroutine fetches the raw LED
;                           numbers and uses a lookup
;                           table to convert the raw numbers into
;                           valid LED data, and stores that
;                           data in memory. It then formats
;                           the data routine to clock out the
;                           data and calls the clock_out routine
;                           to serially clock the data out.
;  Notes
;*****************************************************************

load_time:
            ld    pointer,#hour_ptr;Point to high hours register
            ld    BUFFER,#04        ;Load Buffer with 4
            ld    counter,#4        ;Load Counter with 4
```

```
            ld    address_hi,#HIGH led_table ;Get end address of
                                             ;LED table
load_table: ld    address_lo,#LOW led_table  ;Get start address
            ld    dat,@pointer               ;Load the high order hours number
            add   address_lo,dat             ;Add to offset the table address
            ldc   dat,@address               ;Load the segments data
            ld    @BUFFER,dat                ;Place the data at the location
                                             ;pointed to by buffer
            dec   pointer                    ;Decrement the pointer
            inc   BUFFER                     ;Increment the buffer
            djnz  counter,load_table;Continue until all 4 segment's
                                             ;data has been loaded
            ld    BUFFER,#4                  ;Load buffer with 4
            ld    p0,#START                  ;Make Data High
            call  clock_out                  ;Send one Start Bit
            ld    counter,#4                 ;Load counter with 4
            cp    hours_hi,#0                ;Is hours hi 0?
            jr    nz,next_digit              ;If != 0 then go clock out 4
                                             ;digits
            ld    bit_count,#8               ;Else clock out the first segment
                                             ;blank
            and   p0,#ZERO                   ;Make data low
zero_clock: call  clock_out                  ;Clock out data
            djnz  bit_count,zero_clock;Loop 8 times
            dec   counter                    ;Decrement the counter
            inc   BUFFER                     ;Point to the next digit's data
next_digit: ld    bit_count,#8               ;Load the bit counter with 8
            ld    dat,@BUFFER                ;Load the data from the buffer
rotate:     rcf                              ;Clear the carry flag
            rrc   dat                        ;Rotate bits right into the carry
                                             ;location
            jr    nc,zero                    ;If no carry then clock out a
                                             ;zero
            or    p0,#ONE                    ;Else prepare to clock out a one
            jr    clock_it                   ;Jump to clock it
zero:       and   p0,#ZERO                   ;Make data low
clock_it:   call  clock_out                  ;Go clock out bit
            djnz  bit_count,rotate;Loop until 8 bits clocked out
            inc   BUFFER                     ;Point to next digit
            djnz  counter,next_digit;Loop until all 4 digits
                                             ;clocked out
            ld    counter,#3                 ;Load counter with three
            cp    AM_PM,#%FF                 ;PM?
            jr    z,high_dat                 ;If yes prepare to clock out a
                                             ;high
            and   p0,#ZERO                   ;Else clock out zero
            jr    stopbits                   ;Jump to start clocking
high_dat:   or    p0,#ONE                    ;Make data high
stopbits:   call  clock_out                  ;Clock out last two data and one
                                             ;stop bit
            and   p0,#ZERO                   ;Make data low
            djnz  counter,stopbits;Loop until all 3 bits clocked
                                             ;out
            ld    p0,#ENABLE_HI              ;Disable display
```

```
           ret                    ;Return to caller

*************************************************************
*  Function Name:           time_convert
*  returns                  unpacked BCD data
*  entry values (arguments) Packed BCD data
*  Description:             This subroutine converts the seconds,
*                           minutes, and hours bcd
*                           data into units and tens of units for
*                           displaying by placing the
*                           data into a separate register and then
*                           and-ing out the appropriate
*                           4 bits.
*  Notes
*************************************************************
time_convert:ld  minutes_lo,minutes  ;Separate
                                     ;digits for display
           ld    minutes_hi,minutes  ;   "       "     "    "
           and   minutes_lo,#%0f     ;Get rid of upper 4 bits
           swap  minutes_hi          ;Swap nibbles
           and   minutes_hi,#%0f     ;Get rid of upper 4 bits
           ld    hours_lo,hours      ;Separate digits for display
           ld    hours_hi,hours      ;   "       "     "    "
           and   hours_lo,#%0f       ;Get rid of upper 4 bits
           swap  hours_hi            ;Swap nibbles
           and   hours_hi,#%0f       ;Get rid of upper 4 bits
           ret                       ;Return

*************************************************************
*  Function Name:           clock_out
*  returns                   Clocked out data bit
*  entry values (arguments)  Port zero data and enable set
*  Description:             This subroutine simply toggles the
*                           clock pin high and then low again
*                           after a short delay.
*  Notes
*************************************************************

clock_out: or    p0,#CLOCK_HI     ;Send high clock
           nop                    ;Wait
           nop                    ; "
           nop                    ; "
           and   p0,#CLOCK_LO     ;Send low clock
           nop                    ;Wait
           nop                    ; "
           nop                    ; "
           ret                    ;Return

*************************************************************
*   Unused IRQ Vectors placed to give the controller a place
*   to vector and return in case of a stray interrupt.
*************************************************************
IRQ1:                        ;Dummy interrupt routines
IRQ2:                        ;  "       "        "
```

```
IRQ3:                       ;  "       "        "
IRQ4:                       ;  "       "        "
IRQ5:                       ;  "       "        "
                            ;
            iret            ;Return from interrupt

****************************
*   LED Segment Tables
****************************

led_table: .byte   %3F            ; Data for number "0"
           .byte   %06            ; Data for number "1"
           .byte   %5B            ; Data for number "2"
           .byte   %4F            ; Data for number "3"
           .byte   %66            ; Data for number "4"
           .byte   %6D            ; Data for number "5"
           .byte   %7D            ; Data for number "6"
           .byte   %07            ; Data for number "7"
           .byte   %7F            ; Data for number "8"
           .byte   %67            ; Data for number "9"

************************************************************************

     end start              ;End of assembly
```

60Hz_Clock.inc Include Source File

```
;*************************************************************
; 60Hz Clock Include File
;
;*************************************************************

hour_ptr     .equ      %1F  ;
START        .equ      02   ;
ENABLE_HI    .equ      01   ;}General purpose registers and ram
                            ;locations
ONE          .equ      02   ;
ZERO         .equ      %FD  ;
CLOCK_HI     .equ      04   ;
CLOCK_LO     .equ      %FB  ;
CLEAR_MEM    .equ      %18  ;

TIME_REG     .equ      %10  ;Working Register group 1
DELAY_REG    .equ      %30  ;Working Register group 3
slow_delay   .equ      r4   ;
AM_PM        .equ      %31  ;
millisec     .equ      r8   ;
seconds      .equ      r9   ;
minutes      .equ      r10  ;
hours        .equ      r11  ;
counter      .equ      r0   ;
bit_count    .equ      r1   ;General purpose registers and ram
                            ;locations
address_hi   .equ r4        ;
address_lo   .equ    r5     ;
address      .equ    rr4    ;
minutes_lo   .equ    r12    ;
minutes_hi   .equ    r13    ;
hours_lo     .equ    r14    ;
hours_hi     .equ    r15    ;
pointer      .equ    r6     ;
dat          .equ      r7   ;
BUFFER       .equ      r2   ;
```

8MHz.asm Source File

```
;*****************************************************************
;*  Simple Real Time Clock
;*
;*  This program demonstrates how to implement a simple
;*  Real Time clock using a Z86E40 or another Z8 with at
;*  least 14 I/O's.  This code was taken from the Z8 Application
;*  notes manual and modified by Mark Thissen on 08/08/99.
;*  This code is compliant with ZDS v.2.12. The include file
;* "8MHz_Clock.inc" must be assembled with this source code
;*
;*  Counter Timer 0 is set up to count down to zero every
;*  5ms with an 8 MHz XTAL. The formula is:
;*
;*     i = t x p x v
;*
;*     i = desired time interval until end of T/C count
;*     t = input clock period (8 divided by the XTAL frequency)
;*     p = prescaler value (1-64 decimal 01-00 Hex)
;*     v = T/C value (1-256 decimal 01h,02h,...00h)
;*
;*     Therefore,  5ms = 1us x 20 x 250
;*
;*     ** This interval can be lengthened somewhat to let the
;*        clock run a little slower or shortened to have the
;*        opposite effect but should keep time within +/- the
;*        crystal tolerance.
;*
;*  Counter Timer 1 is set up to count down to zero every
;*  roughly 1ms with an 8 MHz XTAL. The formula is:
;*
;*     i = t x p x v
;*
;*     i = desired time interval until end of T/C count
;*     t = input clock period (8 divided by the XTAL frequency)
;*     p = prescaler value (1-64 decimal 01-00 HEX)
;*     v = T/C value (1-256 decimal 01h,02h,...00h)
;*
;*     Therefore,  1ms = 1us x 4 x 256
;*
;*
;*****************************************************************

;*****************************************************************
;*      Include section
;*****************************************************************

        include "8MHz_Clock.inc"

;*****************************************************************
;*      Interrupt Vectors
;*****************************************************************
```

```
      vector reset = Start        ;Start Vector 0Ch
      vector IRQ0 = IRQ0          ;Empty Vector Locations
      vector IRQ1 = IRQ1          ;  "     "      "
      vector IRQ2 = IRQ2          ;  "     "      "
      vector IRQ3 = IRQ3          ;  "     "      "
      vector IRQ4  = time         ;16 bit Address of IRQ4, labeled
                                  ;time
      vector IRQ5  = load_time    ;16 bit Address of IRQ5, labeled
                                  ;load_time

*****************************************************************
*     MAIN:  Timekeeping based on the 8-MHz crystal
*****************************************************************

      GLOBALS ON

Start:
           di                     ;Disable interrupts
           ld    spl,#%80         ;Set Stack pointer to start
                                  ;@ 7Fh.
           clr   sph              ;Clear high byte of stack pointer
                callINIT          ;Initialize device
main_loop: ei                     ;Enable interrupts
           jr    main_loop        ;Continuous loop

*****************************************************************
*  Function Name:           System initialization
*  returns                  Various Initialized registers
*  entry values (arguments) NA
*  Description:             Initializes the system
*  Notes
*****************************************************************

INIT:      srp   #WORKING_REG     ;Set Register Pointer to 10h
           ld    t0,#250          ;Load T0 Count with 250 Decimal
           ld    pre0,#01010001b  ;Set for 5mS period and
                        |    |    ;Modulo Counting
                    ; |    ||
                    ; |    ||___ Set for Modulo Count
                    ; |    ||___ Reserved, Must be 0
                    ; |    |
                    ; |    |
                    ; |____|
                     ;   |________ 14h or 20d for prescaler value.

           ld    t1,#0            ;Load T1 Count with 256 Decimal
           ld    pre1,#00010011b  ;Set for roughly 1mS period and
                                  ;Modulo Counting
                    ; |    ||
                    ; |    ||___Set for Modulo Count
                    ; |    ||____Internal Clock Source
                    ; |    |
                    ; |    |
```

```
                    ; |_____|
                    ;   |_______ 4 for prescaler value.


            ld    p2m,#0           ;Set Port 2 as 8 bit output port.
            ld    p3m,#1           ;Set Port 2 Push-Pull.
            ld    p01m,#04         ;Set Port 0 as output / internal
                                   ;Stack
            clr   p0               ;Clear Port 0 outputs
            clr   p3               ;Clear p3 outputs
            clr   p2               ;Clear Port 2 outputs
            clr   switch_count     ;Clear GPR
            clr   STATUS           ;Clear GPR
            clr   HALF_SECOND      ;Clear GPR
            ld    ipr,#%08         ;IRQ5 has priority over all IRQ's
            ld    imr,#%30         ;Enable IRQ4 and IRQ5
            ld    tmr,#%cf         ;init timer mode register
            ld    pointer,#04      ;start at ram location 04
            ld    r15,#12          ;Clear 12 bytes of Ram from-
clear_reg:  clr   @pointer         ;-04h to 0fh.
            inc   pointer          ;inc pointer location
            djnz  r15,clear_reg    ;decrement counter and clear ram
                                   ;location until 0
            ld    HOURS,#%12       ;Start at 12:00
            ld    pointer,#HOURS_HI;Start with hours register
            ld    display_pointer,#%88 ;Enable first segment
                 ret               ;Return to main

*********************************************************************
*  Function Name:     Time Convert
*  returns            Data in tens and ones registers for
*                     displaying.
*  entry values (arguments)     Minutes and Hours BCD
*  Description:       This subroutine converts the minutes and
*                     hours (packed) bcd data into units and
*                     tens of units (unpacked bcd) for
*                     displaying.
*  Notes
*********************************************************************

time_convert:ld MINUTES_LO,MINUTES     ;Load minutes to separate
                                       ;8-bit registers
            ld   MINUTES_HI,MINUTES  ;
            and  MINUTES_LO,#%0f       ;Dispose of upper 4bits
            swap MINUTES_HI            ;Swap nibbles
            and  MINUTES_HI,#%0f       ;Dispose of upper 4bits
            ld   HOURS_LO,HOURS        ;Same procedure for hours
            ld   HOURS_HI,HOURS        ;
            and  HOURS_LO,#%0f         ;
            swap HOURS_HI              ;
            and  HOURS_HI,#%0f         ;
            ret                        ;return

*********************************************************************
```

```
*  Function Name:     Test Switch
*  returns            Status, switch_count (Debounce register)
*  entry values (arguments)     P3 Data, switch_count
*  Description:       This subroutine checks to see if the time
*                     set switches are pressed
*  Notes
*****************************************************************

test_switch:    push  rp            ;Save register pointer on
                                    ;the stack
           srp  #WORKING_REG        ;set register pointer to 10h
           ld   led_data,p3         ;read switch data
           com  led_data            ;Invert data
           and  led_data,#%03       ;get rid of unused bits
           cp   led_data,#0         ;any switch pushed?
           jr   eq,clear_sw         ;no? clear switch_count and
                                    ;status and return
           tm   led_data,#%1        ;else, minutes switch pressed?
           jr   z,test_hrs          ;no, jump and test next switch
           inc  switch_count        ;increment counter
           cp   switch_count,#%2    ;This is the debounce must be
                                    ;pressed for two passes
           jr   ult,exit_sw         ;if less than two jump out
           or    STATUS,#%1         ;else, set status bit and return
           jr    exit_sw            ;jump out
test_hrs:  tm    led_data,#%2       ;hours switch pressed?
           jr    z,clear_sw         ;no, clear switch_count and
                                   status and return
           inc   switch_count       ;else inc switch_count
           cp    switch_count,#%2 ;This is the debounce must be
                                    ;pressed for two passes
           jr    ult,exit_sw        ;if less than two jump out
           or    STATUS,#%2         ;else, set status bit and return
exit_sw:poprp                       ;reinstate register pointer
                                    ;from stack
           ret                      ;return
clear_sw   clr   STATUS             ;clear status
           clr   switch_count       ;clear switch_count
           pop   rp                 ;reinstate register pointer from
                                    ;stack
           ret                      ;return

*****************************************************************
* Dummy IRQ Vectors placed to give the controller a place
* to vector and return in case of a stray interrupt.
*****************************************************************

IRQ0:                               ;Dummy vectors
IRQ1:                               ;   "      "
IRQ2:                               ;   "      "
IRQ3:                               ;   "      "
           iret                     ;Return from interrupt

*****************************************************************
```

```
* This interrupt routine updates the time and tests the switches
*****************************************************************

time:       push  rp                ;Save Register Pointer
            srp   #TIME_REG         ;Set RP to 00
            call  test_switch       ;Test inputs
            inc   millisec          ;add one millisec
            cp    millisec,#100     ;Half Second?
            jr    ult,no_convert    ;If less than, jump out
            tm    STATUS,#%01       ;Switch 1 pressed?
            jr    z,test_sw2        ;If not check sw2
            add   MINUTES,#%1       ;add one minute
            da    MINUTES           ;Convert to BCD
            cp    MINUTES,#%60      ;Are we at 60 minutes?
            jr    ult,clear_count   ;Jump out if less than
            clr   MINUTES           ;else clear minutes
clear_count:clr   millisec          ;clear millisec counter
            clr   HALF_SECOND       ;clear half second counter
            clr   seconds           ;clear seconds counter
            jr    exit_time         ;jump out
test_sw2:   tm    STATUS,#%02       ;switch 2 pressed?
            jr    z,inc_half_sec    ;if not jump out
            add   HOURS,#%1         ;else add hours+1
            da    HOURS             ;convert to BCD
            cp    HOURS,#%13        ;1:00?
            jr    ult,inc_half_sec  ;jump out if not
            ld    HOURS,#%1         ;change hours to 1
inc_half_sec:inc  HALF_SECOND       ;inc half sec count
            xor   p2,#%80           ;toggle colon
            clr   millisec          ;clear millisec counter
one_second:cp     HALF_SECOND,#%2   ;is this one second?
            jp    ult,exit_time     ;if not exit
            clr   HALF_SECOND       ;else clear counter
inc_seconds:add   seconds,#%1       ;inc sec count
            cp    seconds,#60       ;60 Seconds?
            jr    ult,exit_time     ;if less exit
            clr   seconds           ;else clear counter
            add   minutes,#%1       ;inc minutes count
            da    minutes           ;convert to BCD
            cp    minutes,#%60      ;60 Minutes?
            jr    ult,exit_time     ;if less exit
            clr   minutes           ;else clear minutes counter
            add   hours,#%1         ;inc hours count
            da    hours             ;convert to BCD
            cp    hours,#%13        ;1:00?
            jr    ult,exit_time     ;if not exit
            ld    hours,#%1         ;else make hours 1
exit_time:  call  time_convert      ;call conversion routine
no_convert:pop    rp                ;reinstate register pointer
            iret                    ; Return from interrupt

*****************************************************************
* This interrupt routine Outputs the time data to the LED bank
*****************************************************************
```

```
load_time: push  rp                ;Save register pointer
           srp   #WORKING_REG      ;set new pointer to 10h
load_table:ld    r12,@pointer      ;Load time register
           ld    address_hi,#HIGH led_table ;get start address of
                                   ;LED table
           ld    address_lo,#LOW led_table  ;get end address
           cp    r12,#0            ;start at beginning of table? 0?
           jr    eq,no_index       ;yes to previous question
index_num: incw  address           ;else inc address
           djnz  r12,index_num     ;inc address until equal to
                                   r12 contents
no_index:  lde   led_data,@address ;load led segment value to
                                   ;display
           tm    p2,#%80           ;check status of colon
           jr    z,colon_off       ;if off, jump
           and   p2,#%80           ;else clear p2 bits d0-d6
           jr    continue          ;jump
colon_off: and   p2,#%00           ;clear all p2 bits
continue:       orp2,led_data      ;output segment data
           ld    p0,display_pointer  ;turn on display
           dec   pointer           ;dec pointer
           cp    pointer,#SECONDS_HI ;compare pointer with 0ah
           jr    ugt,load_time_ret;get out on greater values
           ld    pointer,#HOURS_HI;else start with beginning digit
           cp    @pointer,#0       ;is the beginning digit a 0?
           jr    ne,load_time_ret  ;if not get out
           dec   pointer           ;dec the pointer
           rr    display_pointer   ;rotate to point to first digit
load_time_ret:rr display_pointer   ;rotate display pointer to turn
                                   ;on next segment
           pop   rp                ;reinstate reg pointer
           iret                    ;return from interrupt



;*****************************
;* LED Segment Tables
;*****************************

led_table:.byte  %01               ; number "0"
          .byte  %79               ; number "1"
          .byte  %12               ; number "2"
          .byte  %06               ; number "3"
          .byte  %4c               ; number "4"
          .byte  %24               ; number "5"
          .byte  %20               ; number "6"
          .byte  %0f               ; number "7"
          .byte  %00               ; number "8"
          .byte  %0c               ; number "9"
          .byte  %7f               ; Off
```

```
        end Start               ;End of assembly
```

8MHz_Clock.inc Include Source File

```
;*****************************************************************
;*           8MHz. Clock include file
;*****************************************************************

WORKING_REG     .equ    %10     ;Working Register group 1
address_hi      .equ    r0      ;
address_lo      .equ    r1      ;
address         .equ    rr0     ;
pointer         .equ    r2      ;General purpose registers and
                                ;ram locations
led_data        .equ    r3      ;
display_pointer .equ    r4      ;
HALF_SECOND     .equ    %16     ;
TIME_REG        .equ    %00     ;Working Register group 0
millisec        .equ    r5      ;
seconds         .equ    r6      ;
minutes         .equ    r7      ;
hours           .equ    r8      ;
seconds_lo      .equ    r9      ;
seconds_hi      .equ    r10     ;
minutes_lo      .equ    r11     ;
minutes_hi      .equ    r12     ;
hours_lo        .equ    r13     ;
hours_hi        .equ    r14     ;
switch_count    .equ    r15     ;General purpose registers and
                                ;ram locations
STATUS          .equ    %04     ;
MILLISEC        .equ    %05     ;
SECONDS         .equ    %06     ;
MINUTES         .equ    %07     ;
HOURS           .equ    %08     ;
SECONDS_LO      .equ    %09     ;
SECONDS_HI      .equ    %0a     ;
MINUTES_LO      .equ    %0b     ;
MINUTES_HI      .equ    %0c     ;
HOURS_LO        .equ    %0d     ;
HOURS_HI        .equ    %0e     ;
```

Timing Diagrams and Tables

Figures 1 and 2 illustrate two TSM6234B timing relationships.

Figure 1. TSM6234BTiming

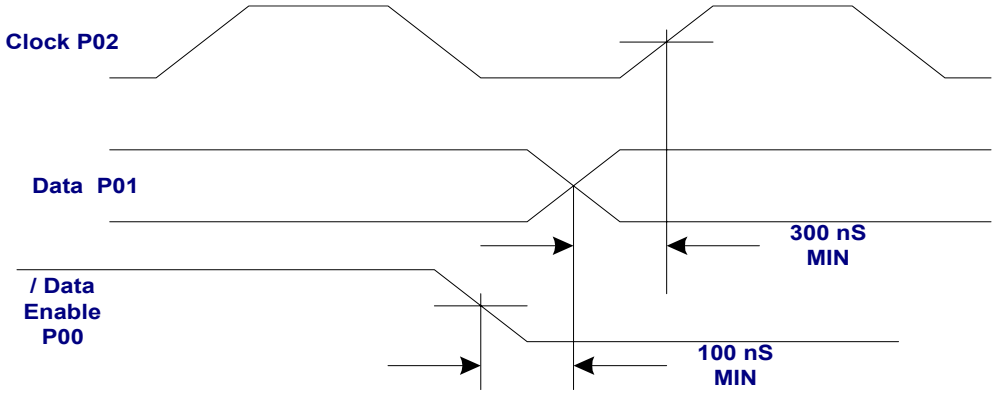
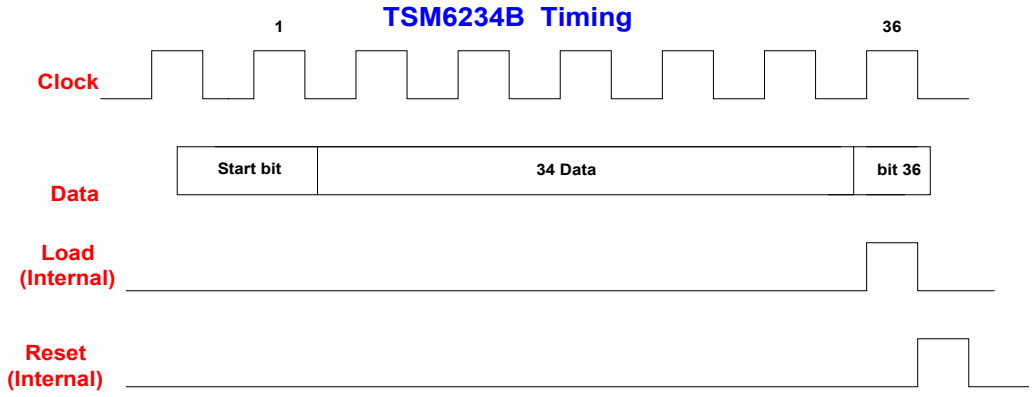


Figure 2. TSM 6234B Timing



The 34 data bits include 8 segments x 8 bits plus two bits for external LEDs (x or don't care). The Start bit must be accompanied by a high data bit, and the stop bit must be accompanied by a low data bit.

Technical Drawings

Figure 3. TSM6234B Block Diagram

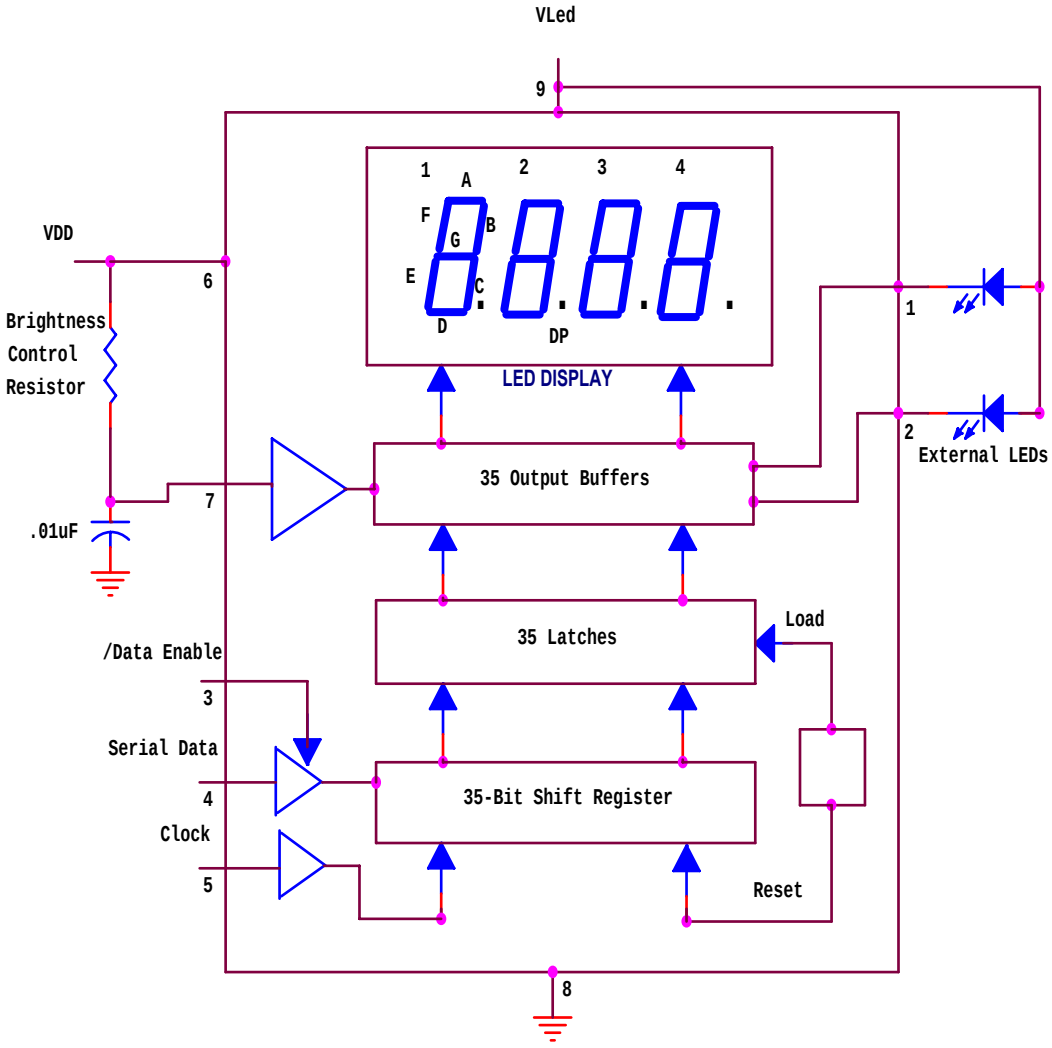


Figure 4. 60-Hz Interrupt Service Routine

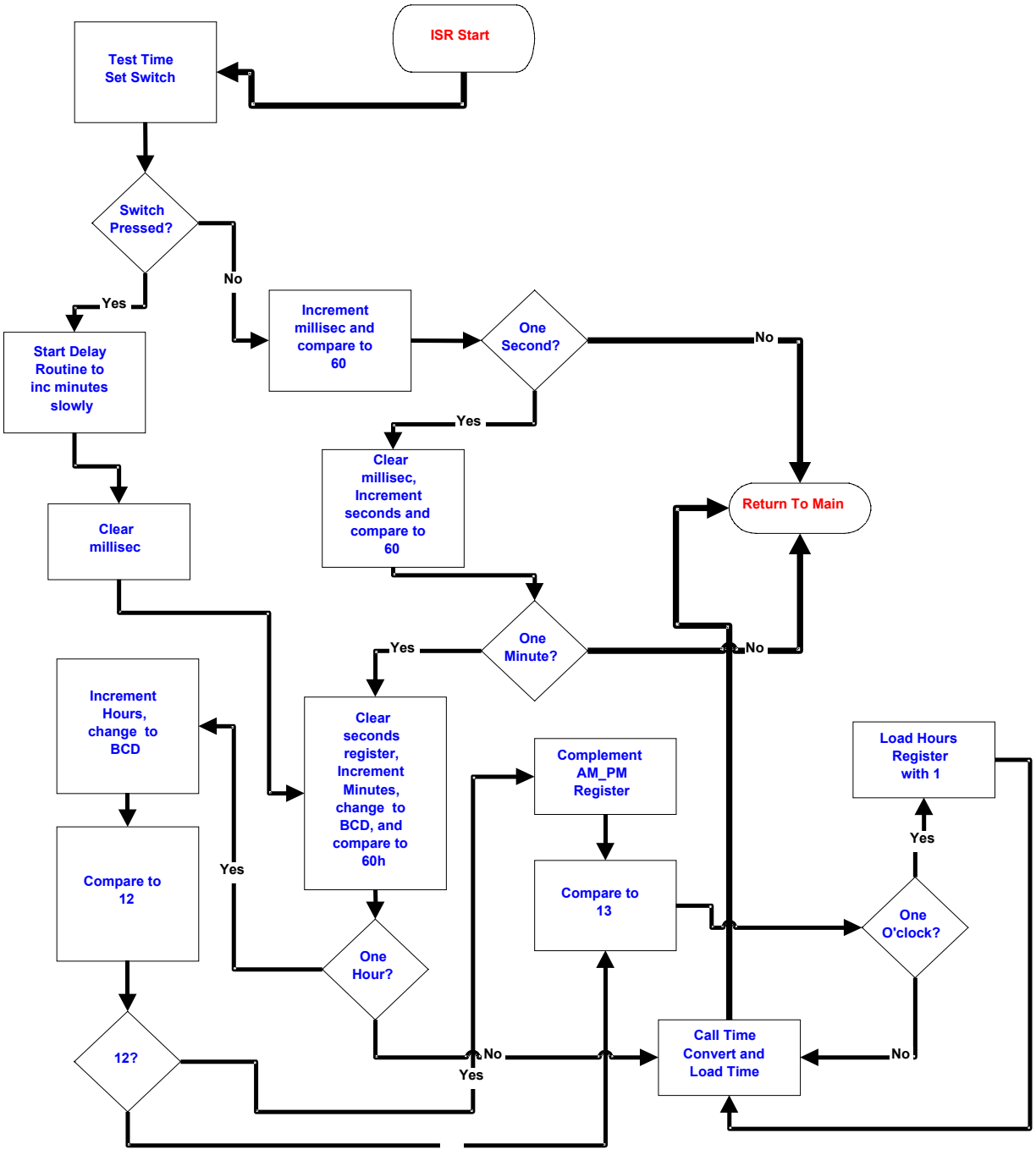


Figure 5. 60-Hz Interrupt Main Routine

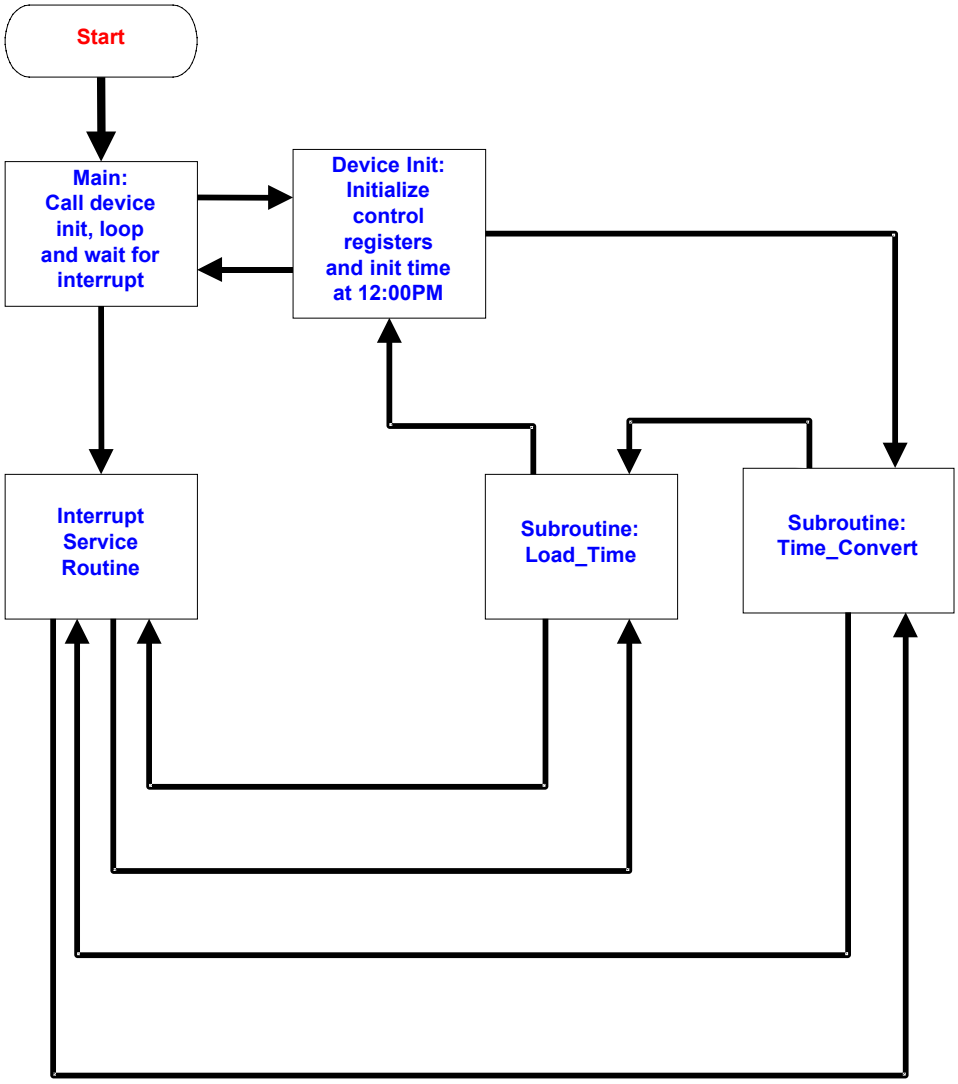


Figure 6. 60-Hz Interrupt Load_Time Subroutine

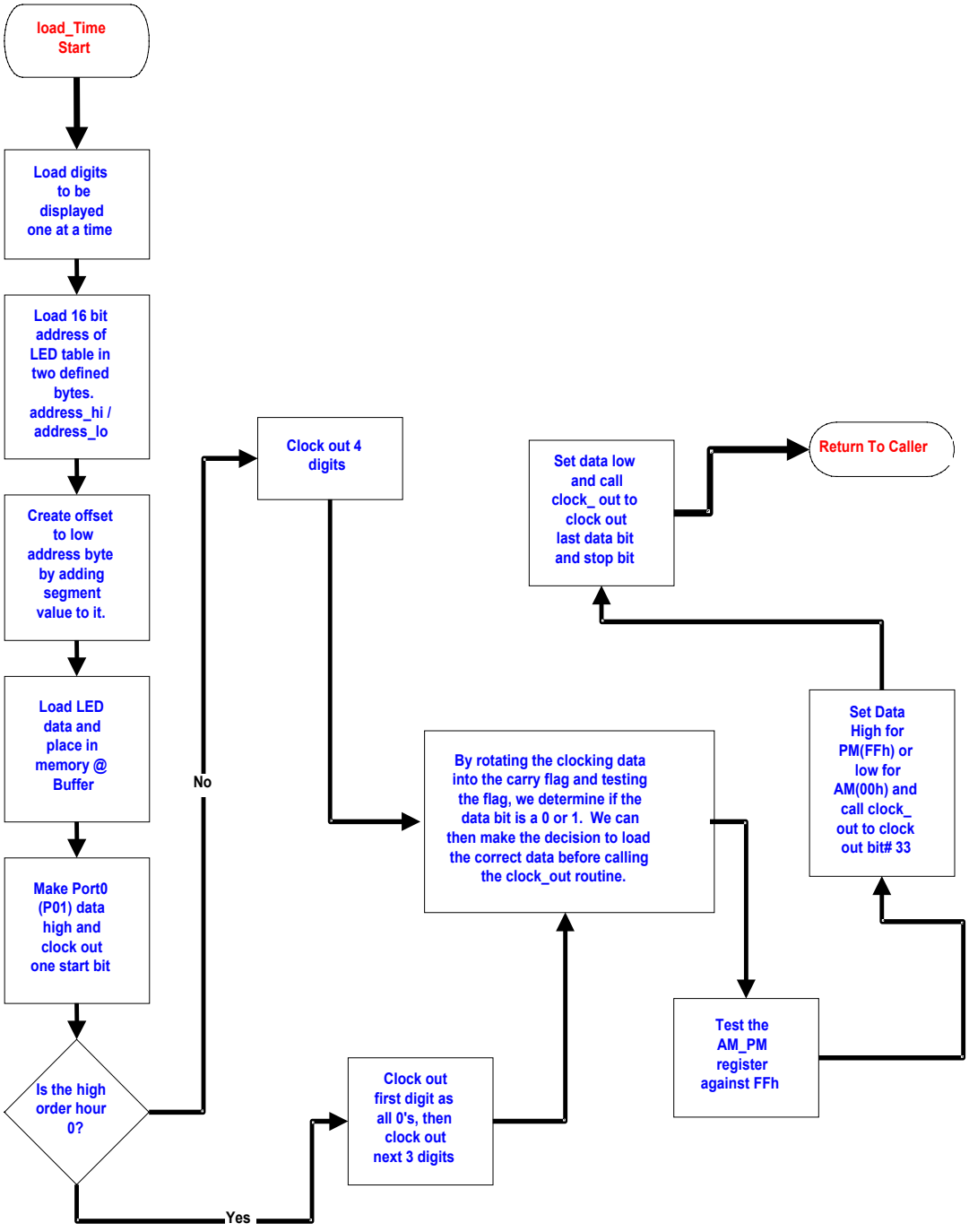


Figure 7. 60-Hz Interrupt INIT, Time_Convert, and Clock_Out Subroutines

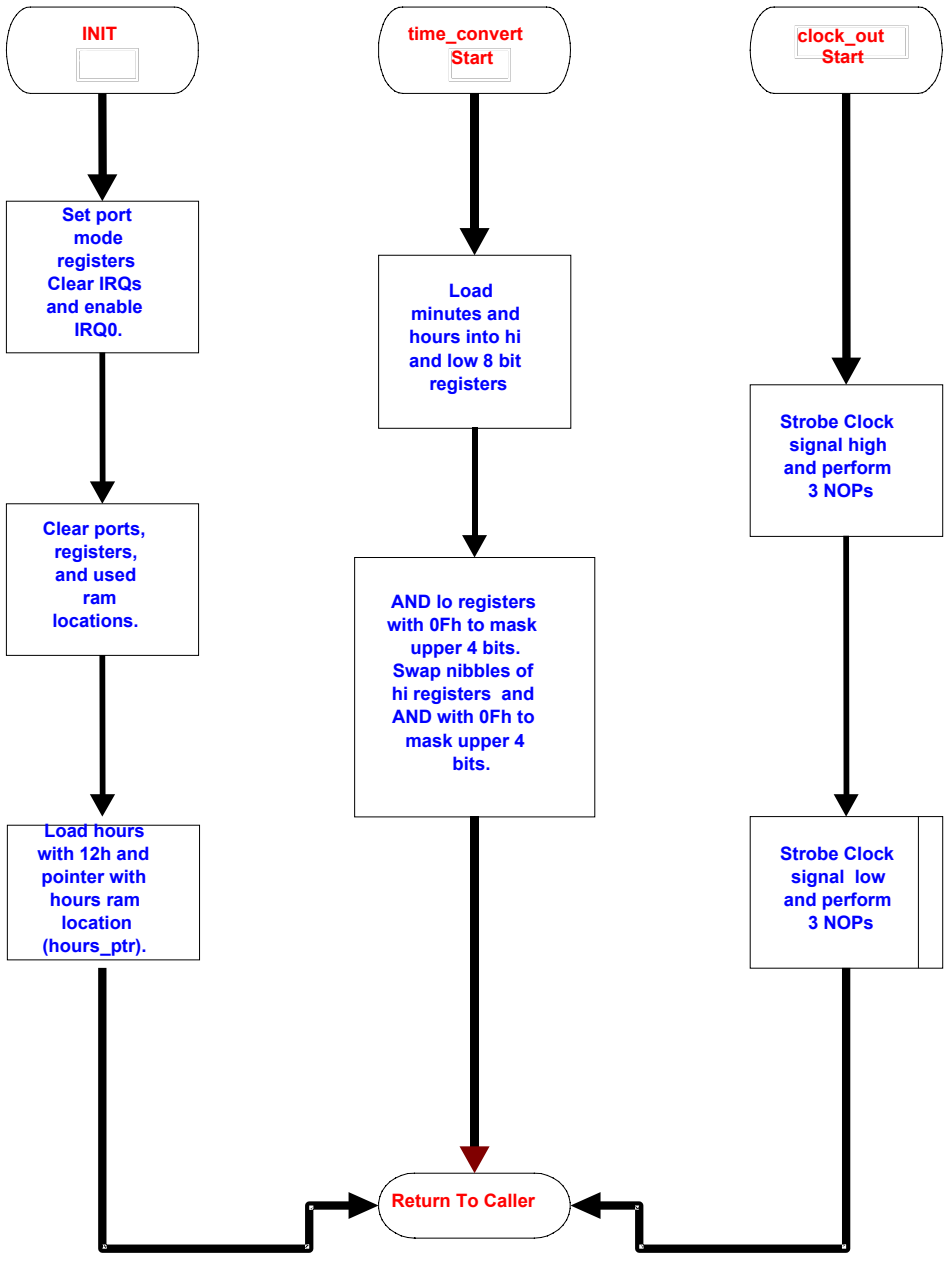


Figure 8. 8-MHz Crystal Main Routine

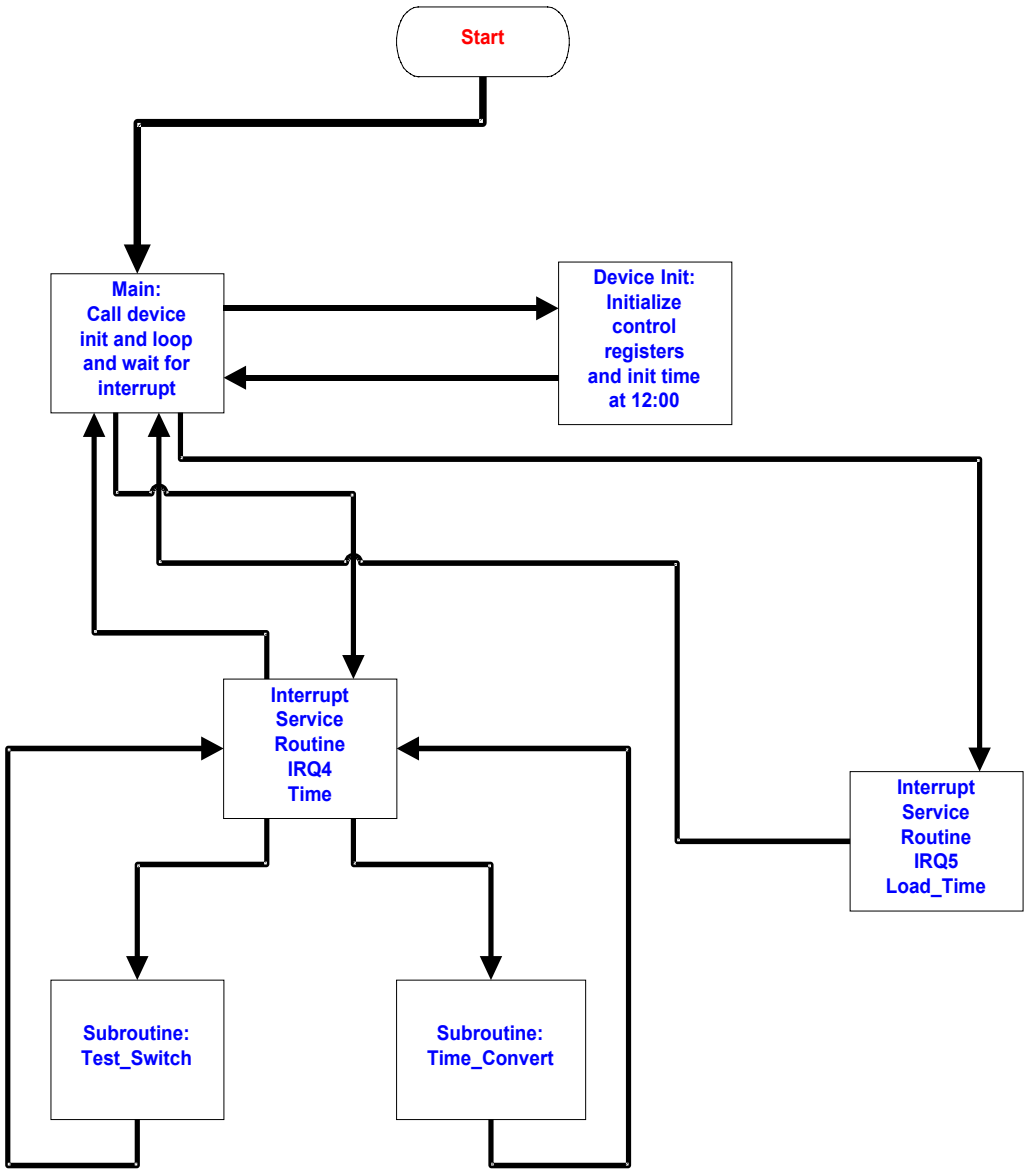
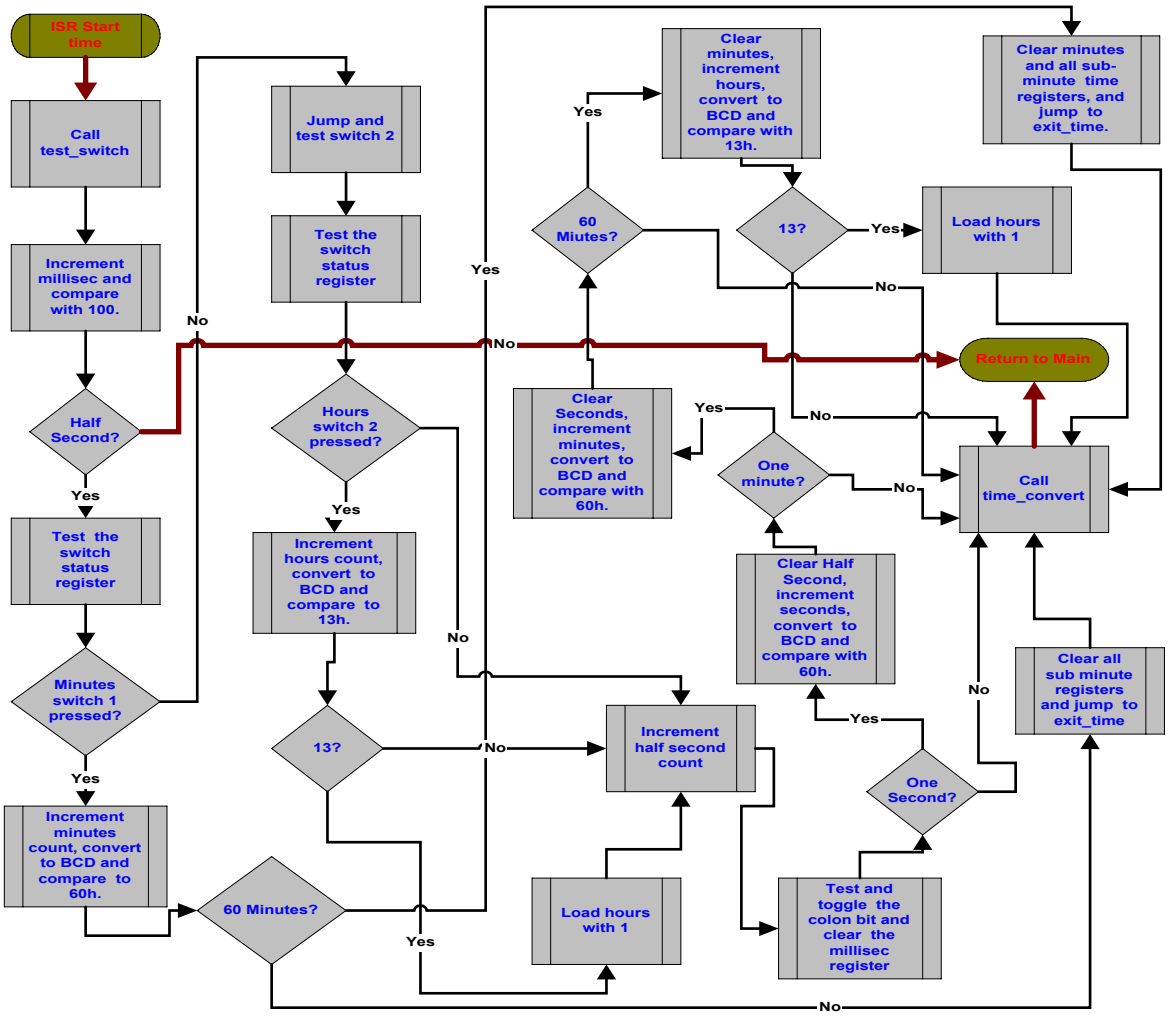


Figure 9. 8-MHz Crystal IRQ4 Service Routine



Prepared by | Date | Page 1 of 1

ZiLOG | Mark Thissen | 9/13/99

Process | Approved by | Date

Timekeeping using an 8MHz Crystal
IRQ5 Service Routine (load_time) | | 9/13/99

Figure 10. 8-MHz Crystal IRQ5 Service Routine

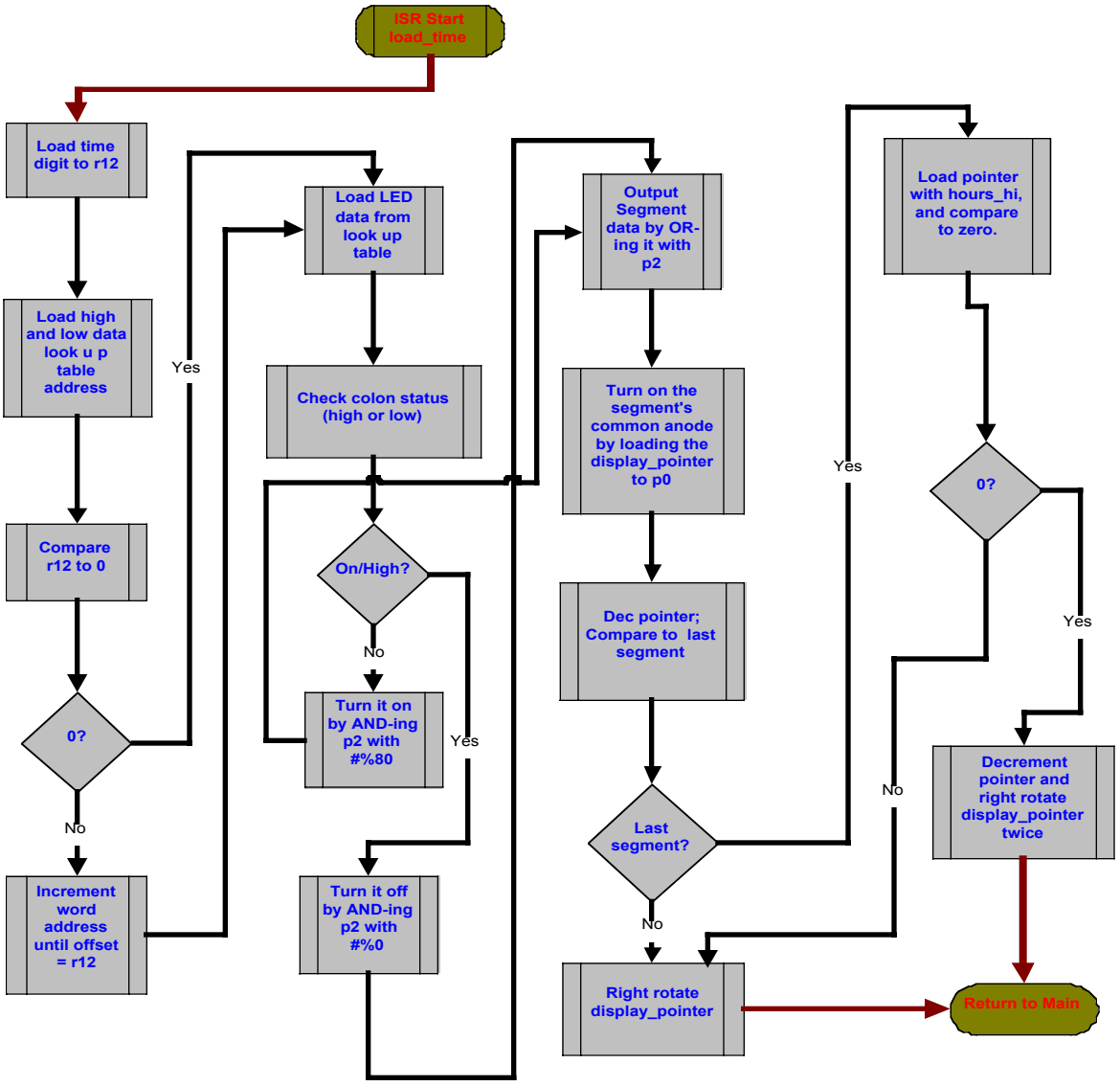


Figure 11. 8-MHz Crystal INIT and Time_Convert Subroutines

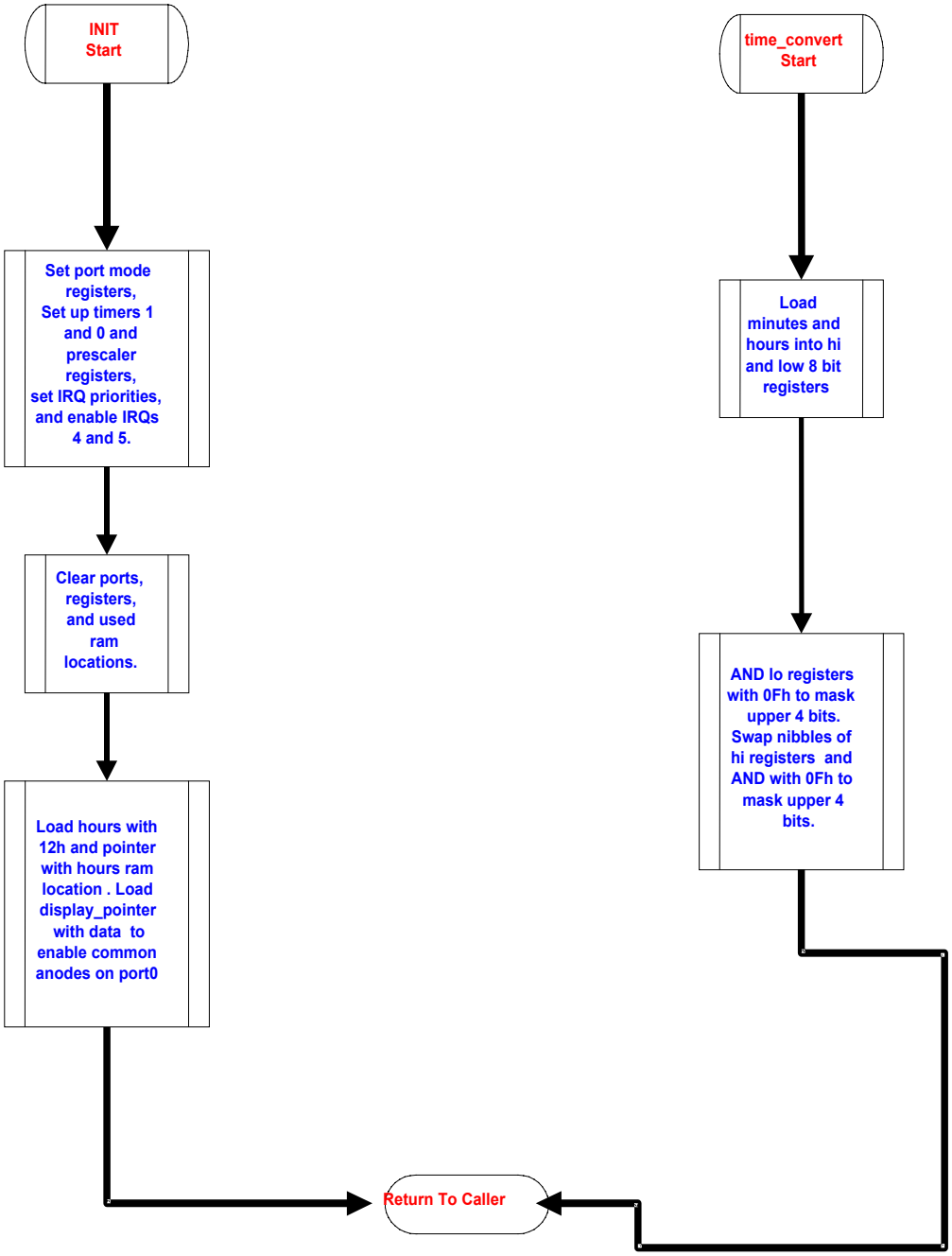
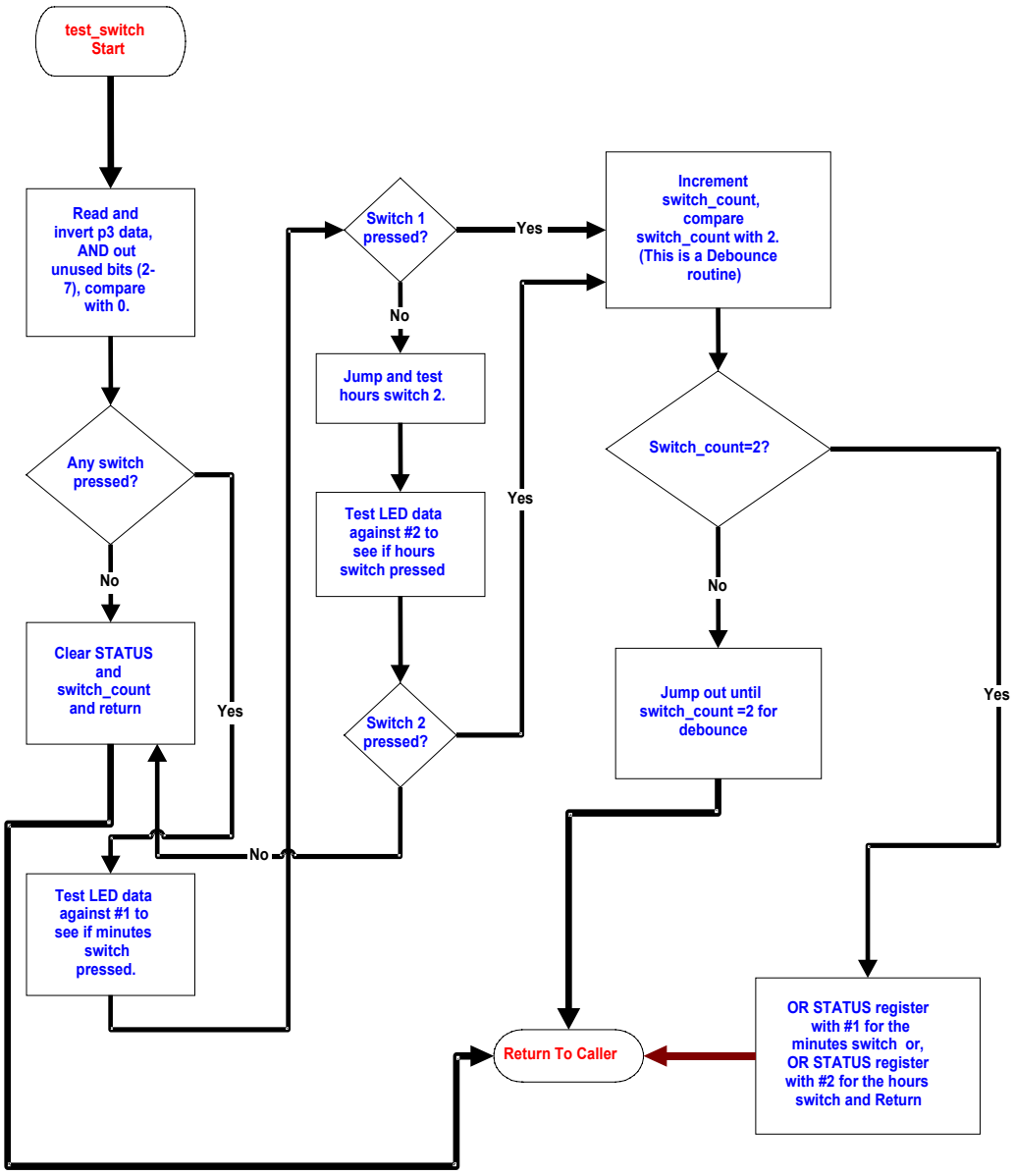


Figure 12. 8-MHz Crystal Test_Switch Subroutine



Test Procedure

Equipment Used

- IBM-compatible PC with WIN95 or better
- Z86CCP01ZEM (Z8 CCP Emulation Board)
- Z86CCP00ZAC. This accessory kit provides a power supply for the emulator, 28- and 40-pin ZIF programming sockets, all the emulation ribbon cables, and an RS-232 cable for PC communication; the 40-pin ZIF socket is required to program the Z86E40 and the 40-pin ribbon cable for emulation

General Test Setup and Execution

1. Build up the target test board using the schematic provided in the Appendix.
2. Type in and assemble the source code using ZDS 2.12 or higher. ZDS can be downloaded free of charge from: http://www.zilog.com/support/sd.html
3. For the 8-MHz clock, use either the V_{CC} from the emulator or from the target board. For the 60-Hz clock, use the V_{CC} from the target board. Simply remove the J1 jumper from the emulator board.
4. Power up the emulator (refer to the emulator User's Manual) and download the assembled code.
5. Power up the target, and plug the emulator into it using the appropriate emulation ribbon cable. Ensure that either the target or the emulator crystal is disabled.
6. Use ZDS to reset and go, and run the application.
7. When the emulation process is satisfactory, burn a blank OTP, insert it into the project, and power it up.

Test Results

Test results indicate that the 8-MHz clock runs flawlessly, whether being emulated or running from an OTP. The 60-Hz clock also runs flawlessly when running from an OTP. When running under emulation, the 60-Hz clock runs twice as fast. There are two interrupts that vector to IRQ0 with the C50 ICE chip. Expect the minutes to update every 30 seconds under emulation conditions. The clock operates correctly when an OTP is burned and installed.

Appendix

Schematics

Figure 13. Timekeeping Based on a 60-Hz Power Line Interrupt

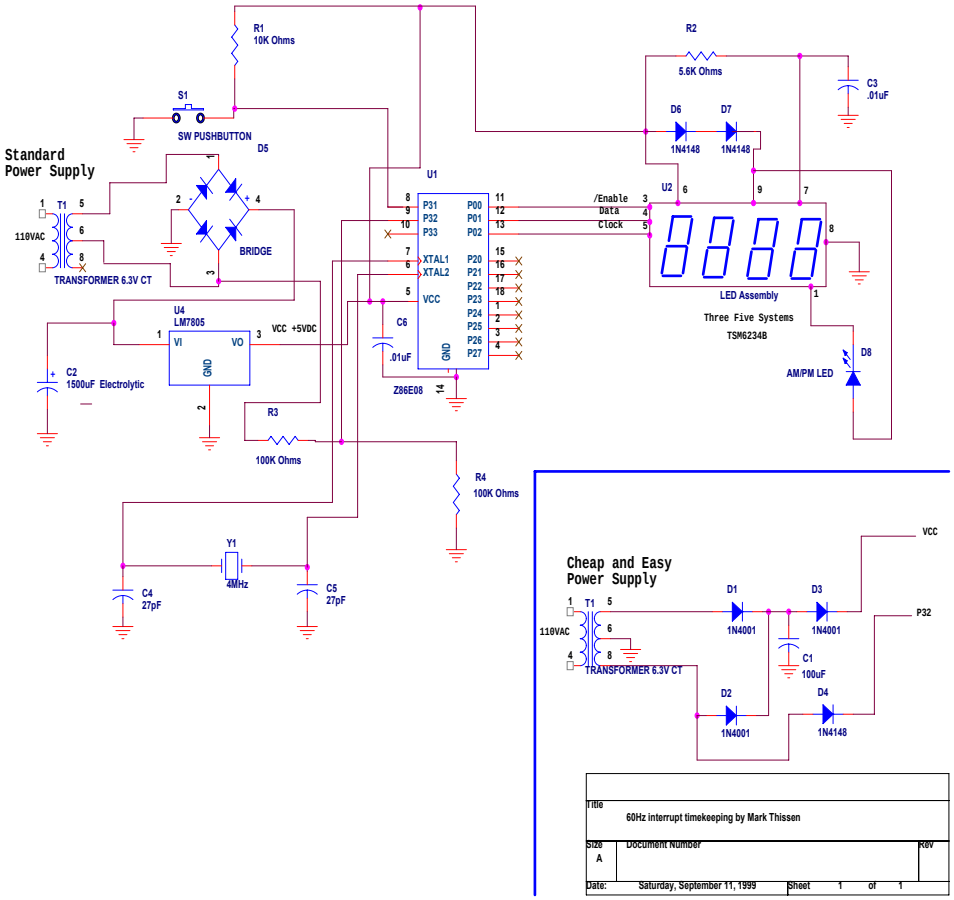


Figure 14. Timekeeping Based on the 8-MHz Crystal (1 of 2)

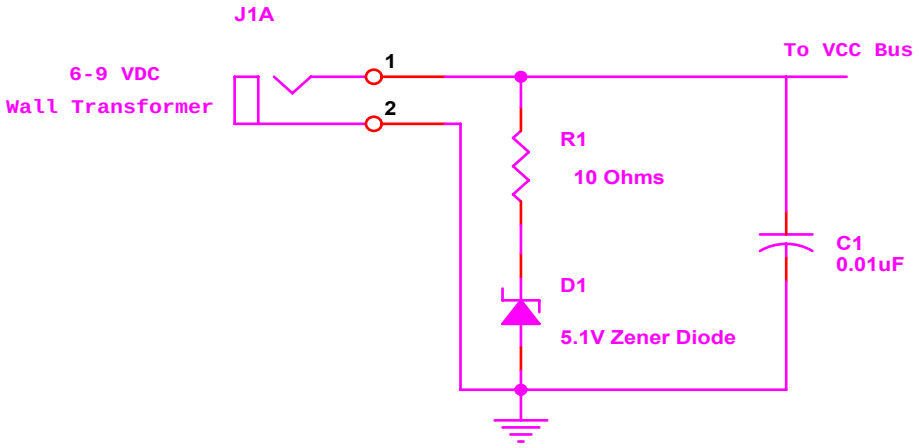


Figure 15. Timekeeping Based on the 8-MHz Crystal (2 of 2)