```
$MODLP52
org 0000H
    ljmp    setup

; Timer/Counter 0 overflow interrupt vector
org 0x000B
    ljmp Timer0_ISR

; imports
$include(math32.inc)
$include(macros.inc)
;$include(LCD_4BIT.inc)

CLK     equ     22118400
BAUD    equ     115200
T1LOAD  equ     (0x100-CLK/(16*BAUD))

; for the alarm
T0LOAD  equ     ((65536-(CLK/4096)))

; pins for ADCs
ADC_CE      equ     P2.0
ADC_MOSI    equ     P2.1
ADC_MISO    equ     P2.2
ADC_SCLK    equ     P2.3

; pins for shift register
LED_DATA    equ     P2.4
LED_LATCH   equ     P2.5
LED_CLK     equ     P2.6
LED_CLR     equ     P2.7

DSEG at 30H
    result: ds      2
    bcd:    ds      5
    x:      ds      4
    y:      ds      4

BSEG
    mf:     dbit    1

CSEG

;---------------------------------;
; Routine to initialize the ISR   ;
; for timer 0                     ;
;---------------------------------;
timer0_init:
    ; clear bits for the timer
    mov a,      TMOD
    anl a,      #0xF0
    orl a,      #0x01
    mov TMOD,   a

    ; set reload value
    mov TH0,    #high(T0LOAD)
    mov TL0,    #low(T0LOAD)

    ; enable interrupts
    setb    ET0
    setb    TR0
    ret

```

```
;--------------------------------;
; ISR for timer 0.  Set to execute;
; every 1/4096Hz to generate a    ;
; 2048 Hz square wave at pin P3.7 ;
;--------------------------------;
timer0_ISR:
    ; operating in mode 1, reload the timer
    clr     TR0
    mov     TH0,    #high(T0LOAD)
    mov     TL0,    #low(T0LOAD)
    setb    TR0
    cpl     P3.7
    reti

; configure serial port and baudrate using timer 1
InitSerialPort:
    ; debounce reset button
    mov     R1,     #222
    mov     R0,     #166
    djnz    R0,     $
    djnz    R1,     $-4
    ; set timer
    clr     TR1
    anl     TMOD,   #0x0f
    orl     TMOD,   #0x20
    orl     PCON,   #0x80
    mov     TH1,    #T1LOAD
    mov     TL1,    #T1LOAD
    setb    TR1
    mov     SCON,   #0x52
    ret

; send character using serial port
putChar:
    jnb     TI,     putchar
    clr     TI
    mov     SBUF,   a
    ret

; send a string using serial port
putString:
    clr     a
    movc    a,  @a+dptr
    jz      putString_return
    lcall   putChar
    inc     dptr
    sjmp    putString
putString_return:
    ret

; initialize SPI
SPIinit:
    setb    ADC_MISO
    clr     ADC_SCLK
    ret

; send byte in R0, receive byte in R1
SPIcomm:
    push    ACC
    mov     R1,     #0
    mov     R2,     #8
SPIcomm_loop:
    mov     a,      R0
```

```
    rlc     a
    mov     R0,     a
    mov     ADC_MOSI,   c
    setb    ADC_SCLK
    mov     c,      ADC_MISO
    mov     a,      R1
    rlc     a
    mov     R1,     a
    clr     ADC_SCLK
    djnz    R2,     SPIcomm_loop
    pop     ACC
    ret

; main program
setup:
    mov     SP,     #7FH
    mov     PMOD,   #0

    ; initialize MCP3008
    setb    ADC_CE
    lcall   SPIInit
    lcall   InitSerialPort

    ; shift register
    clr     LED_DATA
    clr     LED_LATCH
    clr     LED_CLK
    clr     LED_CLR
    sleep(#2)
    setb    LED_CLR

    ; timer initialization
    lcall   timer0_init

    ; enable global interrupts
    setb    EA

; loops forever
loop:
    clr     ADC_CE
    ; starting bit is 1
    mov     R0,         #0x01
    lcall   SPIcomm

    ; read channel 0 & save to result, only care about lower 2 bits
    ; read xxxxxxRR xxxxxxxx
    mov     R0,         #0x80
    lcall   SPIcomm
    mov     a,          R1
    anl     a,          #0x03
    mov     result+1,   a

    ; read rest of 8-bits
    ; read xxxxxxxx RRRRRRRR
    mov     R0,         #0x55       ; doesn't matter
    lcall   SPIcomm
    mov     result,     R1
    setb    ADC_CE
    sleep(#50)

    ; convert result into BCD
    mov     x,      result
    mov     x+1,    result+1
```

```
190         mov     x+2,    #0x00
191         mov     x+3,    #0x00
192         lcall   hex2bcd
193         mov     result,     bcd
194         mov     result+1,   bcd+1
195
196         ; ignore 0 values because it's not right
197         mov     a,  result
198         jnz     loop_putBCD
199         mov     a,  result+1
200         jnz     loop_putBCD
201         ljmp    loop
202
203         ; toggle things into shift register
204
205     loop_putBCD:
206         ; print BCD for ADC value
207         putBCD(result+1)
208         putBCD(result)
209         mov     a,  #'\r'
210         lcall   putChar
211         mov     a,  #'\n'
212         lcall   putChar
213
214         ; compute Vout
215         ; x is already loaded
216         Load_y(10)
217         lcall   sub32
218         Load_y(49500)                   ; * 5000mV reference
219         lcall   mul32
220         Load_y(1023)                   ; / 1023 ratio
221         lcall   div32
222         Load_y(27300)                    ; - 2730mV voltage to convert to celcius
223         lcall   sub32
224
225         ; clear sound
226         clr     TR0
227
228         ; output to LED
229         ; < 10
230         Load_y(1000)
231         lcall   x_gteq_y
232         jb      mf,     low1
233         barLED(#1)
234         ljmp    loop_end
235     low1: ; 10 < t < 20
236         Load_y(2000)
237         lcall   x_gteq_y
238         jb      mf,     low2
239         barLED(#2)
240         ljmp    loop_end
241     low2: ; 20 < t < 25
242         Load_y(2500)
243         lcall   x_gteq_y
244         jb     mf,     low3
245         barLED(#3)
246         ljmp    loop_end
247     low3: ; 25 < t < 35
248         Load_y(3500)
249         lcall   x_gteq_y
250         jb     mf,     low4
251         barLED(#4)
252         ljmp    loop_end
```

```
low4: ; 35 < t < 45
    Load_y(4500)
    lcall   x_gteq_y
    jb      mf,     med1
    barLED(#5)
    ljmp    loop_end
med1: ; 45 < t < 60
    Load_y(6000)
    lcall   x_gteq_y
    jb      mf,     med2
    barLED(#6)
    ljmp    loop_end
med2: ; 60 < t < 70
    Load_y(7000)
    lcall   x_gteq_y
    jb      mf,     high1
    setb    TR0
    barLED(#7)
    ljmp    loop_end
high1: ; 70 < t < 80
    Load_y(8000)
    lcall   x_gteq_y
    jb      mf,     high2
    setb    TR0
    barLED(#8)
    ljmp    loop_end
high2:
    setb    TR0
    ljmp    loop_end
loop_end:
    ; print results to SPI
    lcall   hex2bcd
    putBCD(bcd+1)
    putBCD(bcd)

    ; print terminating string
    mov     a,  #'\r'
    lcall   putChar
    mov     a,  #'\n'
    lcall   putChar
    ljmp    loop
END

```