CPU Fan Controller

Using PWM from a PIC18F452 Microcontroller

Summer 2014 (July 21)

Peter Minzicu and Francisco Romero

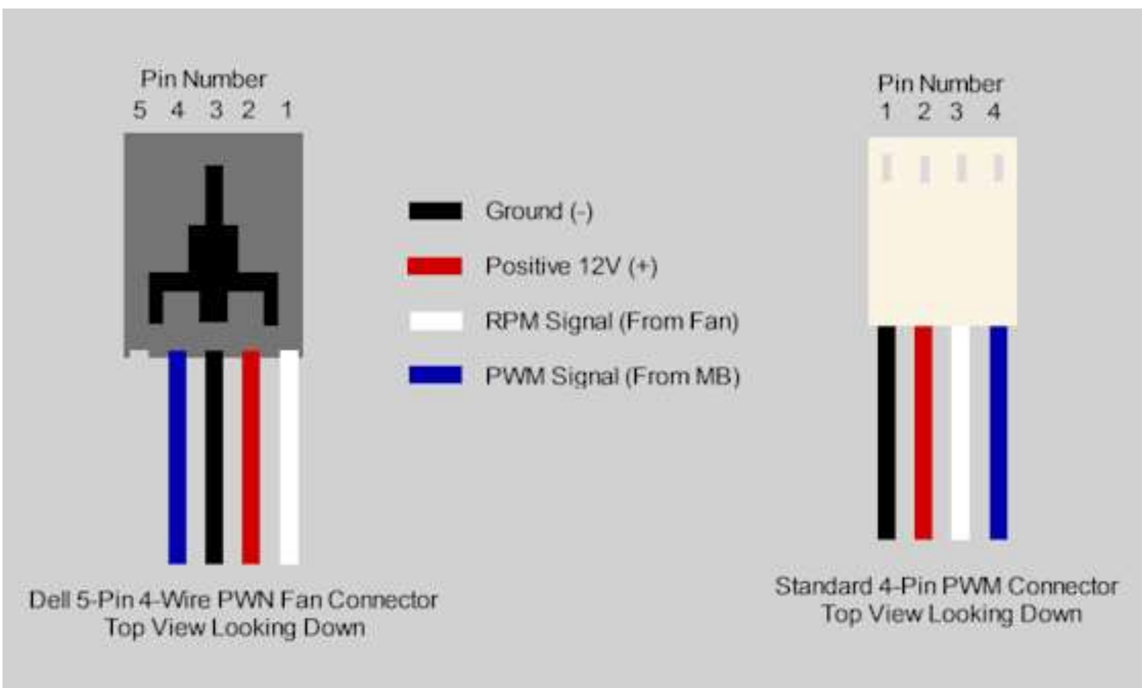
EE 425 Computer Engineering Lab

Introduction:

The purpose of the final project was to demonstrate our knowledge in programming the Pic controller. Throughout the semester, we were given specific labs to practice our programming skills on the PIC board. Instructions were given on how to program the PIC board so that certain modules can work. The different modules that were interfaced with the PIC board were a Potentiometer, a stepper motor, and a LCD Display. The PIC controller was also programmed to utilize built in functionality. Some of the built in components were an ADC and DAC converter, interrupt and an SPI Bus. For the final project, we were to implement one or more of these functionalities for a different use. Our decision was to add the use of a pushbutton and parts of the ADC code to control the speed of a computer fan. By modifying a delay counter, we are able to make a pin on the PIC board pulse at a specific rate. When the pulses reach a certain value, it can control a computer fan through Pulse Width Modulation (PWM).

Process:

In order to properly control the speed of the CPU fan, it is important to know how PWM works. PWM works by pulsing a certain voltage at a particular frequency. An example would be toggling a switch 10 times a second. The device where the voltage is being sent to would receive 10 pulses of voltage. This can also be expressed in Hertz. The device receives a PWM signal of 10Hz. PWM is used frequently in the industry to control the speeds of electric motors. PWM can be used for both AC and DC motors. Commonly, the original way to control the speed of a motor was to change the voltage to it. If a motor ran at full speed with 6 Volts, to run at half speed, 3 V was supplied. However, this isn't the most efficient way of controlling a motor. Voltage itself may not always be perfectly stable. 3 Volts can be 3.1V or 2.9V at different times. For an application that requires precise control of speed such as a drill, these variations are unacceptable. In addition, if one wanted to lower voltage, a resistor could be used. However much of the supply power will be wasted as heat in the resistor when going to the motor. This is not efficient. With PWM, voltages are more stable and this the motor will work more precisely. Voltage pulses are easily manipulated than voltage sources. In the CPU fan, integrated circuits read in a PWM signal then alter the speed of the fan. The CPU fan is rated at 12 Volts and 1.3 Amps. The 12V is always constant so computer manufacturers don't need to make a variable voltage supply. Here we will see a Pinout diagram for the CPU fan. With this we can see the wiring of the motor and how it handles its control sources.



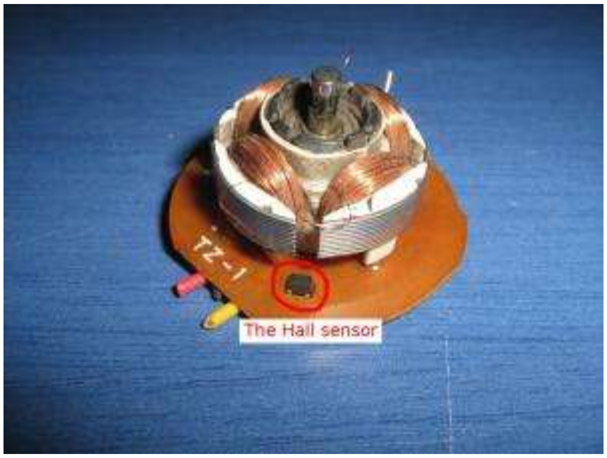
Power

There are 5 pins on the Fan connector. Pin 5 is not used. Pins 2 and 3 are for DC power. This fan requires at all times 12V with a current draw of 1.3 Amps. Originally, the power supply in the lab was going to be used for the fan. However, that power supply can only supply up to 0.5 amps at 12Volts. This caused erratic operation with the PWM control. The internal speed control circuits in the fan vary current to physically change motor speed. When different currents were pulled from the power supply, the voltage increased up to 15V and that is out of the operating range for the CPU fan. Therefore, a Computer ATX power supply was used for the fan. This power supply can supply up to 10Amps of current at 12 Volts if needed. Since the fan only draws 1.3Amps, the power supply is more than enough.

Sense

One other pin on the fan is a blue wire known as sense. This will output the speed of the fan. This is accomplished through Hall Effect sensors embedded in the CPU fan. Hall Effect sensors are sensors that can detect a magnetic field. Since motors are magnetic, these sensors can "detect" when a magnet moves over it. Thus when the fan spins, the sensor can tell if the motor turned. If a magnet passes over a Hall Effect sensor, a current is produced. There are 2 Hall Effect sensors on the CPU fan. Therefore when the fan blades make one revolution, there are 2 current pulses from the blue wire.

The currents from these sensors are so small that even the oscilloscope couldn't pick them up. Therefore, a small circuit has to be constructed. A voltage source with the same ground as the fan is sent through a 10K Ohm resistor to the blue wire. This Voltage source can be between 1V-5V. With the voltage energizing the sensor, we are able to obtain a proper reading from the fan. Shown below is a picture of a Hall Effect sensor next to a disassembled fan.



PWM control

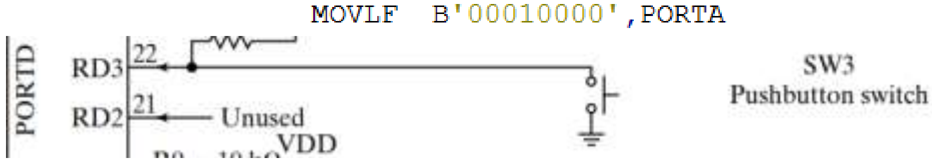
This is a white wire on the CPU fan and is the most important part of the project. With this wire we are able to control the CPU fan with PWM. This wire can control the fan in 3 ways. If this White wire is grounded to the fan ground, the internal fan circuitry commands the fan to spin at its lowest speed. Reading the speed with the oscilloscope, this was 600RPM. If this white wire is left bare and isn't receiving a signal, then the control circuitry allows the fan to run at full speed. This is because the internal circuitry assumes no PWM control is desired, so the fan runs unmetered. With the RPM reading on the oscilloscope, the max RPM of the fan was 3600 RPM. This made the fan very loud and moved a lot of air. Since having the fan operating at full speed or very low speed isn't desirable, we use PWM on the White wire to command the fan to spin at an acceptable speed. Experimenting with the signal generator in the lab, it was discovered that a PWM signal of 4 KHz with a VPP of 0.5V made the fan run at 50% speed. This made the fan move a good amount of air but it was not as noisy. Since we wanted the fan to operate at this speed, we needed to find a way to make the PIC controller pulse this frequency.

PIC Configuration

Our goal with the PIC board is to control the fan speed by using a pushbutton. Pushing the button once will make the fan run at full speed. Pushing it again will make it run at low speed. To do this we modified a delay code originally created for an Analog to Digital converter. This allows us to generate a PWM signal optimal for 50% fan speed. To do this, we need to initialize certain registers of the PIC controller and instruct them to create the PWM signal.

Register Initialization

```
MOVLF  B'11100001',TRISA
MOVLF  B'11011100',TRISB
MOVLF  B'11000000',TRISC
MOVLF  B'00001111',TRISD
MOVLF  B'00000100',TRISE
MOVLF  B'10001000',T0CON
MOVLF  B'00010000',PORTA
```



TRISD

We used PORTD RD3 as an input to read when the SW3 button was pushed. By setting bit 3 to zero we initialized it as an input.

T0CON

-This register enables timer0 and the internal clock (MOVLF B'10001000',T0CON). We set the prescaler off and set the counter to a 16 bit counter (MOVLF B’10001000, T0CON).

Mainline

```
Mainline
        rcall  Initial
L1
        btfss PORTD, RD3
        rcall ChangeSpeed

        btg   PORTC,RC2
        btfsc PORTC,RC4
        rcall  LoopTimeL

        btfss PORTC,RC4
        rcall LoopTime

    bra L1
PL1
```

Our main line procedure uses pin RD3 to determine if we have pressed the button. We used btfss PORTD, RD3 to only change the speed if RD3 is set to 1. If it is clear it will continue rotating its normal speed. When the bit is clear we will go into a function called ChangeSpeed that will change the speed (frequency) of the fan.

```
ChangeSpeed
    bsf PORTA, RA1
    bsf PORTA, RA2
    bsf PORTA, RA3

    btg PORTC,RC4
    bnz  ByteDisplayH
    rcall ByteDisplayM

return
```

When RD3 is set we turned on the 3 RA1, RA2, and RA3 to tell us that we entered this function. We toggle bit RC4 because as we can see in our main line function we used the commands bstfc and bstfss to decide which looptime to call based on the value of RC4 that will be toggled every time we enter our function. We branch ByteDisplayH to write a message to the LCD to show us we changed the speed or we call ByteDisplayM to the array "Manual" to the LCD showing we are normal speed.

```
BignumL   equ     65536-300+12+2
LoopTimeL
```

```
Bignum   equ     65536-625+12+2
LoopTime
```

We used Looptime to give us a frequency of 8.3 KHz. We obtained this result using the following formula

- $\frac{\frac{1}{8.33*10^3}}{0.4*10^{-6}} = \mathbf{625}$, this tells us that we should use this number in our formula for bignum.
- $\frac{\frac{1}{4*10^3}}{0.4*10^{-6}} = \mathbf{300}$, For BignumL we used a frequency of 4 KHz which caused our fan to spin at 50% its full potential.

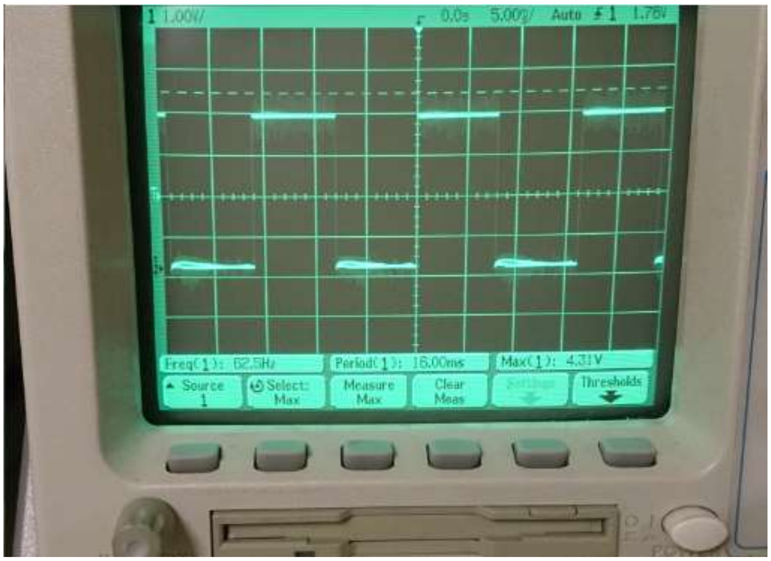
Our main line procedure kept switching between these two routines changing the pulses going into the fan controller.

Results

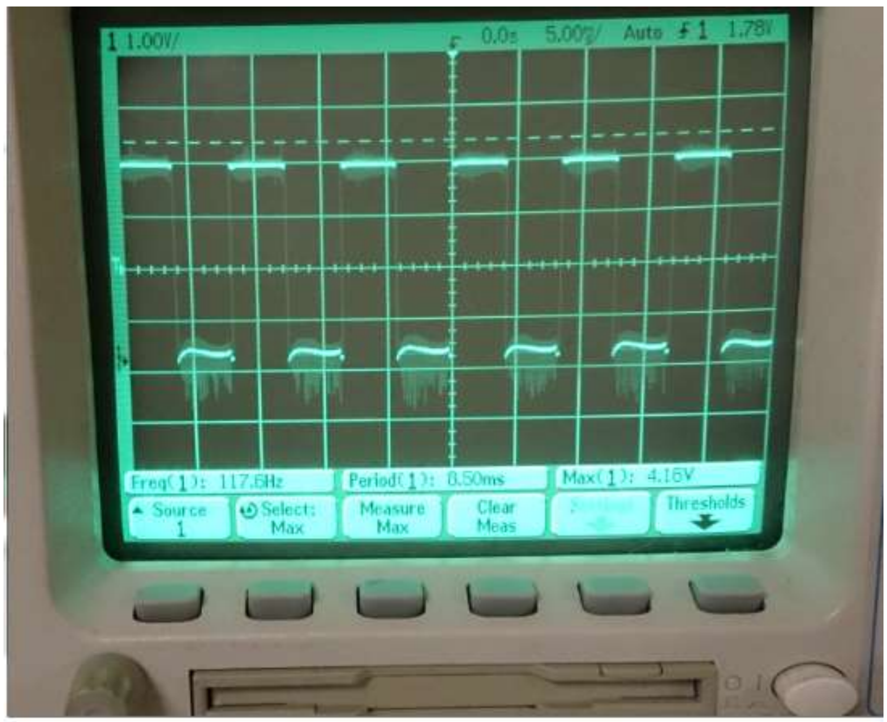
Once we compiled and transferred the code, we were able to generate a PWM signal from the pin RC2 on the PIC board. To control the fan, we connect the pin RC2 to the white control wire of the fan. We apply 12V from a power supply to the fan and finally, connect the sense pin to the oscilloscope so we can see the fan RPM. It is important to note how we calculate rotations per minute for the fan. Since each revolution is 2 pulses, we divide the frequency displayed by the oscilloscope by 2. Then since the frequency is shown per second, we multiply the result by 60. This gives us a RPM value. Also available to the class is a video showing the operation of the fan with the PIC board.

(XXX(Hz)/2)*60 = XXX RPM

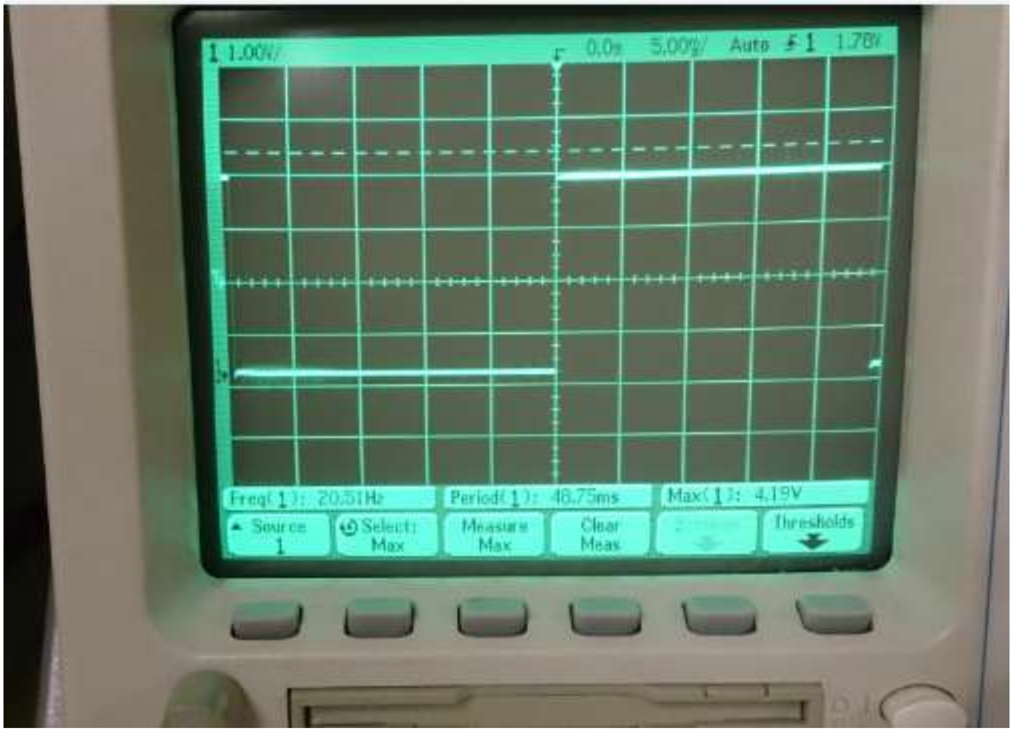
When we first execute the code we made on the PIC board, the pin RC2 will pulse at 4 KHz, this makes the fan spin at 60HZ. In RPM, this is 1800RPM. When the pushbutton isn't pushed, the fan will constantly stay at this speed. Shown here is the result on the oscilloscope:



When we push the button on the PIC board the fan jumps to its highest speed. The period shortens on the oscilloscope as the fun produces faster pulses. We read 120Hz from the oscilloscope so that means the fan is running at 3600RPM



When we push the button on the PIC board again, the fan slows to its lowest speed. The period increases on the oscilloscope as the fun produces smaller pulses. We read 20Hz from the oscilloscope so that means the fan is running at 600RPM


1 1.00V/
0.0s 5.00%/ Auto 1 1.78V
Freq(1): 20.51Hz
Period(1): 48.75ms
Max(1): 4.19V
Source 1
Select: Max
Measure Max
Clear Meas
Thresholds

Conclusion:

Overall what we learned in class with the PIC controller will help us greatly in our careers. The fact that it was possible to implement PWM control on the PIC board means it can be used for many applications. One application recommended by the professor was to combine this lab with a temperature sensor. Another group had a project that displayed the ambient temperature on the PICs LED screen. Based on the temperature, we can combine this lab and make the fan spin at different speeds. Depending on the ambient temperature, the fan can spin faster or slower. This can allow cooling of an important component or just for personal comfort. Another addition can be an emergency stop. When a button on the PIC controller is pushed, Fan operation goes to its slowest speed and the LCD display shows "STOP".

Appendix

Final.asm

```
;;;;;;; P2 for QwikFlash board ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Use 10 MHz crystal frequency.
; Use Timer0 for ten millisecond looptime.
; Blink "Alive" LED every two and a half seconds.
; Display PORTD as a binary number.
; Toggle C2 output every ten milliseconds for measuring looptime precisely.
;
;;;;;;; Program hierarchy ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Mainline
;   Initial
;     InitLCD
;       LoopTime
;   BlinkAlive
;   ByteDisplay (DISPLAY macro)
;     DisplayC
;       T40
;     DisplayV
;       T40
;   LoopTime
;
;;;;;;; Updated Assembler directives ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        list  P=PIC18F4520, F=INHX32, C=160, N=0, ST=OFF, MM=OFF, R=DEC, X=ON
        #include <P18F4520.inc>
        __CONFIG  _CONFIG1H, _OSC_HS_1H  ;HS oscillator
        __CONFIG  _CONFIG2L, _PWRT_ON_2L & _BOREN_ON_2L & _BORV_2_2L  ;Reset
        __CONFIG  _CONFIG2H, _WDT_OFF_2H  ;Watchdog timer disabled
        __CONFIG  _CONFIG3H, _CCP2MX_PORTC_3H  ;CCP2 to RC1 (rather than to RB3)
        __CONFIG  _CONFIG4L, _LVP_OFF_4L & _XINST_OFF_4L  ;RB5 enabled for I/O
        errorlevel -314, -315          ;Ignore lfsr messages

;;;;;;; Variables ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        cblock  0x000                ;Beginning of Access RAM
        TMR0LCOPY                    ;Copy of sixteen-bit Timer0 used by LoopTime
        TMR0HCOPY
        INTCONCOPY                   ;Copy of INTCON for LoopTime subroutine
        COUNT                        ;Counter available as local to subroutines
        ALIVECNT                     ;Counter for blinking "Alive" LED
        BYTE                         ;Eight-bit byte to be displayed
        BYTESTR:10                   ;Display string for binary version of BYTE
```

```
                CLKCYCLE
                CHOICE:0
        endc

;;;;;;; Macro definitions ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

MOVLF   macro  literal,dest
        movlw  literal
        movwf  dest
        endm

POINT   macro  stringname
        MOVLF  high stringname, TBLPTRH
        MOVLF  low stringname, TBLPTRL
        endm

DISPLAY macro  register
        movff  register,BYTE
        call  ByteDisplay
        endm

;same function as above, this one displays in the following row

DISPLAY_ macro  register
         movff  register,BYTE
         call  ByteDisplayNew
         endm
;;;;;;; Vectors ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

        org  0x0000             ;Reset vector
        nop
        goto  Mainline

        org  0x0008             ;High priority interrupt vector
        goto  $             ;Trap

        org  0x0018             ;Low priority interrupt vector
        goto  $             ;Trap

;;;;;;; Mainline program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

Mainline
        rcall  Initial            ;Initialize everything
                ;bcf PORTC, RC4
L1
                btfss PORTD, RD3
```

```
                rcall STOP

                rcall ADCON
                btg  PORTC,RC2         ;Toggle pin, to support measuring loop time
        ;rcall  BlinkAlive      ;Blink "Alive" LED
                btfsc PORTC,RC4
        rcall  LoopTimeL        ;Make looptime be ten milliseconds

                btfss PORTC,RC4
                rcall LoopTime

   bra   L1
PL1

;;;;;;; Initial subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine performs all initializations of variables and registers.

Initial
                MOVLF  B'00010001',ADCON0      ;Enable ADCON0, using AN7 for external
signal, AN4 for internal POT1
                MOVLF  B'10001101',ADCON2      ;Enable ADCON2

        MOVLF  B'11100001',TRISA      ;Set I/O for PORTA
        MOVLF  B'11011100',TRISB      ;Set I/O for PORTB
        MOVLF  B'11000000',TRISC      ;Set I/0 for PORTC
        MOVLF  B'00001111',TRISD      ;Set I/O for PORTD
        MOVLF  B'00000100',TRISE      ;Set I/O for PORTE
        MOVLF  B'10001000',T0CON      ;Set up Timer0 for a looptime of 10 ms
        MOVLF  B'00010000',PORTA      ;Turn off all four LEDs driven from PORTA
        rcall  InitLCD
                        rcall ByteDisplayH
        return

;;;;;;; InitLCD subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Initialize the Optrex 8x2 character LCD.
; First wait for 0.1 second, to get past display's power-on reset time.

InitLCD
        MOVLF  10,COUNT             ;Wait 0.1 second
        ;REPEAT_
L2
         rcall  LoopTime            ;Call LoopTime 10 times
         decf  COUNT,F
        ;UNTIL_  .Z.
```

```
	bnz		L2
RL2

	bcf  PORTE,0			;RS=0 for command
	POINT  LCDstr			;Set up table pointer to initialization string
	tblrd*				;Get first byte from string into TABLAT
	;REPEAT_
L3
	 bsf  PORTE,1			;Drive E high
	 movff  TABLAT,PORTD		;Send upper nibble
	 bcf  PORTE,1			;Drive E low so LCD will process input
	 rcall  LoopTime		;Wait ten milliseconds
	 bsf  PORTE,1			;Drive E high
	 swapf  TABLAT,W		;Swap nibbles
	 movwf  PORTD			;Send lower nibble
	 bcf  PORTE,1			;Drive E low so LCD will process input
	 rcall  LoopTime		;Wait ten milliseconds
	 tblrd+*			;Increment pointer and get next byte
	 movf  TABLAT,F			;Is it zero?
	;UNTIL_  .Z.
	bnz		L3
RL3
	return

;;;;;;; T40 subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Pause for 40 microseconds  or 40/0.4 = 100 clock cycles.
; Assumes 10/4 = 2.5 MHz internal clock rate.

T40
	movlw  100/3			;Each REPEAT loop takes 3 cycles
	movwf  COUNT
	;REPEAT_
L4
	 decf  COUNT,F
	;UNTIL_  .Z.
	bnz		L4
RL4
	return

;;;;;;;;DisplayC subroutine;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine is called with TBLPTR containing the address of a constant
; display string.  It sends the bytes of the string to the LCD.  The first
; byte sets the cursor position.  The remaining bytes are displayed, beginning
; at that position.
```

```
; This subroutine expects a normal one-byte cursor-positioning code, 0xhh, or
; an occasionally used two-byte cursor-positioning code of the form 0x00hh.

;;;beg

DisplayC
	bcf  PORTE,0			;Drive RS pin low for cursor-positioning code
	tblrd*			;Get byte from string into TABLAT
	movf  TABLAT,F			;Check for leading zero byte
	;IF_  .Z.
	bnz		L5
	 tblrd+*			;If zero, get next byte
	;ENDIF_
L5
	;REPEAT_
L6
	 bsf  PORTE,1			;Drive E pin high
	 movff  TABLAT,PORTD		;Send upper nibble
	 bcf  PORTE,1			;Drive E pin low so LCD will accept nibble
	 bsf  PORTE,1			;Drive E pin high again
	 swapf  TABLAT,W		;Swap nibbles
	 movwf  PORTD			;Write lower nibble
	 bcf  PORTE,1			;Drive E pin low so LCD will process byte
	 rcall  T40			;Wait 40 usec
	 bsf  PORTE,0			;Drive RS pin high for displayable characters
	 tblrd+*			;Increment pointer, then get next byte
	 movf  TABLAT,F			;Is it zero?
	;UNTIL_  .Z.
	bnz		L6
RL6
	return

;end

;;;;;;; DisplayV subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine is called with FSR0 containing the address of a variable
; display string.  It sends the bytes of the string to the LCD.  The first
; byte sets the cursor position.  The remaining bytes are displayed, beginning
; at that position.

DisplayV
	bcf  PORTE,0			;Drive RS pin low for cursor positioning code
	;REPEAT_
L7
	 bsf  PORTE,1			;Drive E pin high
```

```
        movff  INDF0,PORTD          ;Send upper nibble
        bcf  PORTE,1               ;Drive E pin low so LCD will accept nibble
        bsf  PORTE,1               ;Drive E pin high again
        swapf  INDF0,W              ;Swap nibbles
        movwf  PORTD                ;Write lower nibble
        bcf  PORTE,1               ;Drive E pin low so LCD will process byte
        rcall  T40                ;Wait 40 usec
        bsf  PORTE,0               ;Drive RS pin high for displayable characters
        movf  PREINC0,W             ;Increment pointer, then get next byte
      ;UNTIL_  .Z.                ;Is it zero?
      bnz          L7
RL7
      return

;;;;;;;;;;;;;Added Subroutine;;;;;;;;;;;;;;;;;;;;



ADCON

         rcall T40
         MOVLF B'00000010',ADCON1 ;analog input
         bsf ADCON0,1

L1000
         btfsc ADCON0,1
bra L1000

         MOVLF B'00001110',ADCON1 ;digital output

         MOVLF ADRESL, CLKCYCLE

         ;DISPLAY_ ADRESH
         ;DISPLAY ADRESL

return



;;;;;;; BlinkAlive subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine briefly blinks the LED next to the PIC every two-and-a-half
; seconds.

BlinkAlive
    ;  bsf  PORTA,RA4              ;Turn off LED
```

```
        ;           bsf  PORTC, RC4
        decf  ALIVECNT,F            ;Decrement loop counter and return if not zero
        ;IF_  .Z.
        bnz         L8
          MOVLF  250,ALIVECNT        ;Reinitialize BLNKCNT
          bcf  PORTA,RA4            ;Turn on LED for ten milliseconds every 2.5 sec
                    bcf PORTC, RC4
L8
        return

;;;;;;; LoopTime subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine waits for Timer0 to complete its ten millisecond count
; sequence. It does so by waiting for sixteen-bit Timer0 to roll over. To obtain
; a period of precisely 10000/0.4 = 25000 clock periods, it needs to remove
; 65536-25000 or 40536 counts from the sixteen-bit count sequence.  The
; algorithm below first copies Timer0 to RAM, adds "Bignum" to the copy ,and
; then writes the result back to Timer0. It actually needs to add somewhat more
; counts to Timer0 than 40536.  The extra number of 12+2 counts added into
; "Bignum" makes the precise correction.


Bignum  equ     (100); 500
LoopTime
;           rcall ByteDisplayM
        ;REPEAT_
L9
    ; btg PORTA, RA4
  ;UNTIL_  INTCON,TMR0IF == 1    ;Wait until ten milliseconds are up
        btfss INTCON,TMR0IF

        bra         L9

RL9
        movff  INTCON,INTCONCOPY      ;Disable all interrupts to CPU
        bcf  INTCON,GIEH
        movff  TMR0L,TMR0LCOPY        ;Read 16-bit counter at this moment
        movff  TMR0H,TMR0HCOPY
        movlw  low  Bignum
        addwf  TMR0LCOPY,F
        movlw  high  Bignum
        addwfc  TMR0HCOPY,F
        movff  TMR0HCOPY,TMR0H
        movff  TMR0LCOPY,TMR0L        ;Write 16-bit counter at this moment
        movf  INTCONCOPY,W           ;Restore GIEH interrupt enable bit
        andlw  B'10000000'
```

```
	iorwf  INTCON,F
	bcf  INTCON,TMR0IF		;Clear Timer0 flag
;		bcf PORTA, RA1
		bsf PORTA, RA1
		bcf PORTA, RA2
;		bcf PORTA, RA2
;		bcf PORTA, RA3
		bsf PORTA, RA4

	return

;;;;;;; LoopTime subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; This subroutine waits for Timer0 to complete its ten millisecond count
; sequence. It does so by waiting for sixteen-bit Timer0 to roll over. To obtain
; a period of precisely 10000/0.4 = 25000 clock periods, it needs to remove
; 65536-25000 or 40536 counts from the sixteen-bit count sequence.  The
; algorithm below first copies Timer0 to RAM, adds "Bignum" to the copy ,and
; then writes the result back to Timer0. It actually needs to add somewhat more
; counts to Timer0 than 40536.  The extra number of 12+2 counts added into
; "Bignum" makes the precise correction.


BignumL  equu	65536-300+12+2

LoopTimeL
	;REPEAT_
L91
	;UNTIL_  INTCON,TMR0IF == 1	;Wait until ten milliseconds are up

	btfss INTCON,TMR0IF

	bra		L91
RL91
	movff  INTCON,INTCONCOPY	;Disable all interrupts to CPU
	bcf  INTCON,GIEH
	movff  TMR0L,TMR0LCOPY		;Read 16-bit counter at this moment
	movff  TMR0H,TMR0HCOPY
	movlw  low  BignumL

	addwf  TMR0LCOPY,F
	movlw  high  BignumL
	addwfc  TMR0HCOPY,F
	movff  TMR0HCOPY,TMR0H
	movff  TMR0LCOPY,TMR0L		;Write 16-bit counter at this moment
	movf  INTCONCOPY,W		;Restore GIEH interrupt enable bit
```

```
    andlw  B'10000000'
    iorwf  INTCON,F
    bcf  INTCON,TMR0IF          ;Clear Timer0 flag
            bcf PORTA, RA1
            bsf PORTA, RA2
      ;     bcf PORTA, RA3
      ;     bcf PORTA, RA4

    return


STOP
      ;btfss PORTD, RD3


      bsf PORTA, RA1
      bsf PORTA, RA2
      bsf PORTA, RA3

;     rcall ByteDisplayH
      btg PORTC,RC4
      bnz  ByteDisplayH
      rcall ByteDisplayM
;     rcall LoopTimeL
;     btfss PORTD, RD3

return
;;;;;;; ByteDisplay subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;

; Display whatever is in BYTE as a binary number.
ByteDisplayM
    POINT BYTE_Manual           ;Display "BYTE="
    rcall  DisplayC
    lfsr  0,BYTESTR+8
    ;REPEAT_
L1021
     clrf  WREG
     rrcf  BYTE,F           ;Move bit into carry
     rlcf  WREG,F           ;and from there into WREG
     iorlw  0x30            ;Convert to ASCII
     movwf  POSTDEC0          ; and move to string
     movf  FSR0L,W           ;Done?
     sublw  low BYTESTR
    ;UNTIL_  .Z.
    bnz         L1021
```

```
RL1021

      lfsr  0,BYTESTR               ;Set pointer to display string
      MOVLF  0x80,BYTESTR           ;Add cursor-positioning code (top row)
      clrf  BYTESTR+9               ;and end-of-string terminator
      ;rcall  DisplayV
      return




; Display whatever is in BYTE as a binary number.
ByteDisplayH
      POINT  BYTE_High              ;Display "BYTE="
      rcall  DisplayC
      lfsr  0,BYTESTR+8
      ;REPEAT_
L102
       clrf  WREG
       rrcf  BYTE,F                ;Move bit into carry
       rlcf  WREG,F                ;and from there into WREG
       iorlw  0x30                 ;Convert to ASCII
       movwf  POSTDEC0             ; and move to string
       movf  FSR0L,W               ;Done?
       sublw  low BYTESTR
      ;UNTIL_  .Z.
      bnz         L102
RL102

      lfsr  0,BYTESTR               ;Set pointer to display string
      MOVLF  0x80,BYTESTR           ;Add cursor-positioning code (top row)
      clrf  BYTESTR+9               ;and end-of-string terminator
      ;rcall  DisplayV
      return



ByteDisplay
      ;POINT  BYTE_1                ;Display "BYTE="
      ;rcall  DisplayC
      lfsr  0,BYTESTR+8
      ;REPEAT_
L10
       clrf  WREG
       rrcf  BYTE,F                ;Move bit into carry
       rlcf  WREG,F                ;and from there into WREG
```

```
        iorlw  0x30                ;Convert to ASCII
        movwf  POSTDEC0            ; and move to string
        movf  FSR0L,W              ;Done?
        sublw  low BYTESTR
      ;UNTIL_ .Z.
      bnz           L10
RL10

      lfsr  0,BYTESTR              ;Set pointer to display string
      MOVLF  0xc0,BYTESTR          ;Add cursor-positioning code (top row)
      clrf  BYTESTR+9              ;and end-of-string terminator
      rcall  DisplayV
      return

;;;;;;; ByteDisplay subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; Display whatever is in BYTE as a binary number.

ByteDisplayNew
      ;POINT  BYTE_1               ;Display "BYTE="
      ;rcall  DisplayC
      lfsr  0,BYTESTR+2
      ;REPEAT_
L100
        clrf  WREG
        rrcf  BYTE,F               ;Move bit into carry
        rlcf  WREG,F               ;and from there into WREG
        iorlw  0x30                ;Convert to ASCII
        movwf  POSTDEC0            ; and move to string
        movf  FSR0L,W              ;Done?
        sublw  low BYTESTR
      ;UNTIL_ .Z.
      bnz           L100
RL100

      lfsr  0,BYTESTR              ;Set pointer to display string
      MOVLF  0x80,BYTESTR          ;Add cursor-positioning code (bottom row)
      clrf  BYTESTR+3              ;and end-of-string terminator
      rcall  DisplayV
      return
;;;;;;; Constant strings ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

LCDstr  db  0x33,0x32,0x28,0x01,0x0c,0x06,0x00  ;Initialization string for LCD
BYTE_1  db  "\x80BYTE=   \x00"        ;Write "BYTE=" to first line of LCD
BYTE_High  db  "\x80STOP    \x00"        ;Write "BYTE=" to first line of LCD
BYTE_Manual  db  "\x80Manual    \x00"
```

```
EMERGENCY db   "\x80STOP    \x00"
BarChars                      ;Bargraph user-defined characters
       db  0x00,0x48                ;CGRAM-positioning code
       db  0x90,0x90,0x90,0x90,0x90,0x90,0x90,0x90  ;Column 1
       db  0x98,0x98,0x98,0x98,0x98,0x98,0x98,0x98  ;Columns 1,2
       db  0x9c,0x9c,0x9c,0x9c,0x9c,0x9c,0x9c,0x9c  ;Columns 1,2,3
       db  0x9e,0x9e,0x9e,0x9e,0x9e,0x9e,0x9e,0x9e  ;Columns 1,2,3,4
       db  0x9f,0x9f,0x9f,0x9f,0x9f,0x9f,0x9f,0x9f  ;Column 1,2,3,4,5
       db  0x00                     ;End-of-string terminator

       end
```