Introduction

Liquid Crystal Displays (LCDs) are used/seen everyday. It may be your alarm clock in the morning while waking up, your laptop computer while browsing the internet, your microwave while warming up your cup-of-noodles, or your digital watch while waiting for your class to end. LCDs are becoming popular because they offer more advantages than other display technologies. For example, they take up less space, weigh a lot less, and consume less power than a standard CRT (Cathode Ray Tube) monitor. Consuming less power may be very important for your microprocessor project if you will be running your board and LCD from a battery.

In this memorandum, we will use a parallel character based LCD purchased from www.CrystalFontz.com, one of the largest suppliers of LCDs. Our LCD is HD44780 compatible (the HD44780 is the controller chip made by Hitachi that controls the LCD). This model of the chip is industry standard for most parallel character LCDs. This project has been tested on both the CFAH2002A-NYA-JP and CFAH1602J-YYB-JP models provided by CrystalFontz. Most of the character LCDs supplied by CrystalFontz are HD44780 compatible and therefore the majority of their LCDs will work with our code provided in this memorandum, however the timing specifications vary model to model. Below is a picture of the LCD we used for this memorandum:

Figure 1. The LCD used in our memorandum.

Physical Interface

The LCD communicates with the PIC via a 4-bit or 8-bit parallel connection. To implement the LCD in the 4-bit mode, you will save yourself 4 output pins, however, you will need to write code that sends a nyble (4-bits) at a time. Thus, when you run the LCD in 4-bit mode you will have to write two lines of code to send one 8-bit instruction, whereas in 8-bit mode you would only need to write one line to send all 8 bits. Our LCD will be run in 8-bit mode because it is easier to code. The LCD itself has 16 pins, 8 are used for sending data (in 4-bit mode you only use 4), 2 for power, 1 ground, 2 to indicate the mode, 1 for clock/enable signal and 2 for powering the backlight (which is present on the CFAH1602J-YYB-JP model). The mapping of the pins is found on the data sheet. We have also attached a diagram (Figure 5 on page 7), in the schematic section of this document, depicting the setup of our LCD for each pin that we used. **We strongly recommend that you refer to this diagram as you read the rest of the memorandum.**

Setting up the LCD with your MicroP's Board

First solder wires/header-pins to the 16 pinouts on the LCD. Second, connect these wires/header-pins in a manner that will make it easy for you to debug later on (i.e. use color coded wires to differentiate between data-bit pins versus the enable pin). Figure 2 provided below gives the MicroP's Board to LCD connections that we used in our implementation:

MicroP's Board	LCD Pin-out
PORTD(7)	DB7 - Pin 14
PORTD(6)	DB6 - Pin 13
PORTD(5)	DB5 - Pin 12
PORTD(4)	DB4 - Pin 11
PORTD(3)	DB3 - Pin 10
PORTD(2)	DB2 - Pin 9
PORTD(1)	DB1 - Pin 8
PORTD(0)	DB0 - Pin 7

MicroP's Board	LCD Pin-out
PORTC(6)	E - Pin 6
PORTC(5)	RW - Pin 5
PORTC(4)	RS - Pin 4

Note: Remember to refer to the schematic at the end of this diagram to get a better visual of the setup.

Figure 2. MicroP's Board to LCD connections used.

We connect to PORTD for our data since PORTD is connected to the 8 led strip on the MicroPs board and this allows for easy debugging. We also used PORTC to connect the operation signals E (enable), RS (Register Select) and RW (Read/Write). To help debug the Enable Pin and RS Pin, you may wish to attach an LED in series with their specific ports.

Remember to connect power and ground to the LCD. The operating voltage for the LCD is 4.7 Volts; try to provide this voltage as close as possible to avoid frying or under powering the LCD. Remember that you have two power pins that need to be supplied, one power pin will power the board (Pin 2) and the other power pin (Pin 3) is the operating voltage for the LCD. The operating voltage for the LCD will need to be connected through a potentiometer. Varying this potentiometer controls the contrast of the LCD. You can obtain a potentiometer from the engineering stockroom.

LCD Onboard RAM

There are two different types of RAM on the LCD: Display Data RAM (DDRAM) and Character Generator RAM (CGRAM). The DDRAM's extended capacity is 80 x 8 bits or 80 characters. The DDRAM stores the character at each position on the LCD. CGRAM is used to store custom character patterns created by the LCD user. The capacity of the CGRAM was not stated in the datasheet. For more on custom characters please refer to the LCD data sheet, it will not be covered in this memorandum.

Operation Modes

The CFAH2002A LCD can be set to four different modes of operation in order to do different things with the LCD: instruction mode, busy mode, write mode, and read mode.

Instruction Mode: This mode is set when the user needs to instruct the LCD to perform an action. Such actions include: moving the cursor, changing/setting display parameters, and setting the RAM address counter.

Busy Mode: Busy mode is set when the user wants to inquire as to whether or not the LCD is busy. This is indicated by pin DB7. This mode is not used in this guide because we were able to tell if the LCD was busy or not through the busy light on the MPLAB2IDE connector.

Write Mode: Write mode is set when the user wishes to write a value to RAM.

Read Mode: Read mode is set in order to read from the RAM.

Note: The address at which the data is read/written in Read/Write mode is set in the instruction mode.

Initializing LCD

The first step in programming the LCD is initializing it to your desired settings. These settings are described in the LCD datasheets. For your convenience we will step you through the general concept behind initializing the LCD. In the following steps you will be sending instructions to the LCD. Sending an instruction is setting what logic level (high or low) the 11 bits (DB0-DB7, RS, R/W, E) should be. **Note:** To figure out which bits in figure 2 need to be set high or low for different instructions/functions, refer to Appendix B while reading the descriptions of steps 3-7 in Figure 3.

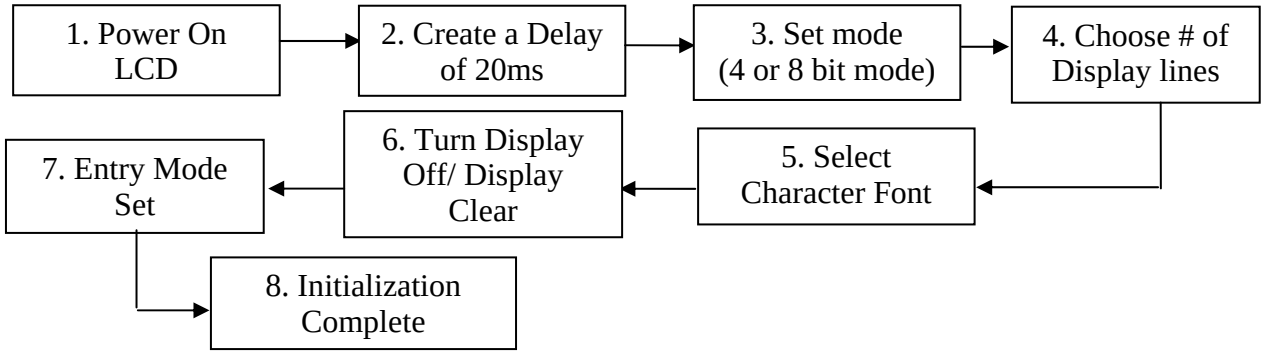


Figure 3. General Block Diagram of initializing the LCD.

1. Connect the LCD to a 4.7 Volt Power Supply (Benchtop Supply or Batteries).

2. Program the PIC to provide the delay specified in your LCD datasheet. In order to do this you will take advantage of the TMROL command and T0CON register on the PIC. You will need to write a loop that executes for the amount of time you need to delay. You will configure the timer on the PIC by setting the bits of the T0CON register. To compute the number of iterations for your loop, you will need to check what speed your board is running at (i.e. 20Mhz) and the prescale value (1:256) you set your timer for in the T0CON register. For example in our case to get a delay of 20ms we needed: {20ms/[1/(20Mhz)]}/(256) = 1562.5

iterations (0x61B). The T0CON register is a readable and writeable register that controls all aspects of Timer 0 and pre-scale conditions. If you are not already familiar with the T0CON register please refer to the PIC datasheet and/or Appendix A of this document for more information.

3. Set the bit in the "Function Set" instruction corresponding to the data length interface that you wish to write your code (in our case it was 4bits or 8bits, we chose 4bits).

4. Set the bit in the "Function Set" instruction corresponding to the number of display lines you desire (in our case it was either 1 or 2, we chose 2).

5. Set the bit in the "Function Set" instruction corresponding to the display font type you desire (in our case it was either 5x11 pixels or 5x8 pixels, we chose 5x11). Now that you have set all the important bits in the "Function Set" instruction, you must send this instruction to the LCD.

6. Send the "Display Off Control" instruction from the PIC to the LCD. This instruction will also encode whether you wish to have a cursor on your LCD and whether you want it to be blinking or not. Then you will need to send the Clear Display instruction to the LCD. This instruction's function is self-explanatory.

7. Send the "Entry Mode Set" instruction to the LCD. This instruction assigns the cursor moving direction (right or left) and enables the shift of the entire display.

8. After going through these steps initialization is complete. You are now ready to align the cursor on the LCD where you want it to be.

Aligning the Cursor on the LCD

After initializing the LCD, you will want to place the cursor at specific locations to display characters. You must again refer to the datasheet and look for the DDRAM address mapping for each segment of the display, in our case we have 32 segments for our 16x2 Character display. For example, below is the mapping that was provided in our datasheet:

Display Position DDRAM address

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

2-Line by 16-Character Display

In your code you first need to tell your LCD which character position you want to display at by setting the DDRAM address equal to one of the above addresses. Once the position is set the user simply sends over the bits representing the character (explained in the next section). To set the DDRAM address you must set the RS bit high every time before you send a DDRAM instruction. The format of the instruction bits is '1XXXXXXX' (DB7 to DB0, which are the data inputs to the LCD as mentioned in Figure2) where the X region

is the corresponding DDRAM address. For example the hexadecimal instruction ‘C0’ would tell the LCD to move to DDRAM position `40`.

Displaying Characters on the LCD

After figuring out how to align your cursor you are now ready to display characters onto the LCD screen. Your LCD comes with a CGROM and CGRAM on it. The CGROM (Character Generator ROM) stores the 5x8 or 5x10 dot character patterns from 8-bit character codes. These character codes for each specific letter/symbol you wish to display will also be located in the datasheet. The CGRAM (Character Generator RAM) is provided so you can generate as well as rewrite new characters by programming it yourself. To display the selected characters from either the CGROM or CGRAM onto the LCD, you must store the specific character code for that character into the DDRAM location in the same manner we described how to set the position of the cursor. The only difference is that when sending a character command you set the RS bit low whereas when sending a cursor position you set the RS bit high.

Specifications

PIC18CXX2 Data Sheet
http://ww1.microchip.com/downloads/en/DeviceDoc/39026c.pdf

CFAH1602J-YYB-JP Data Sheet
http://www.crystalfontz.com/products/1602j/CFAH1602JYYBJP.pdf

CFAH2002A-NYA-JP Data Sheet
http://www.crystalfontz.com/products/2002a/CFAH2002ANYAJP.pdf

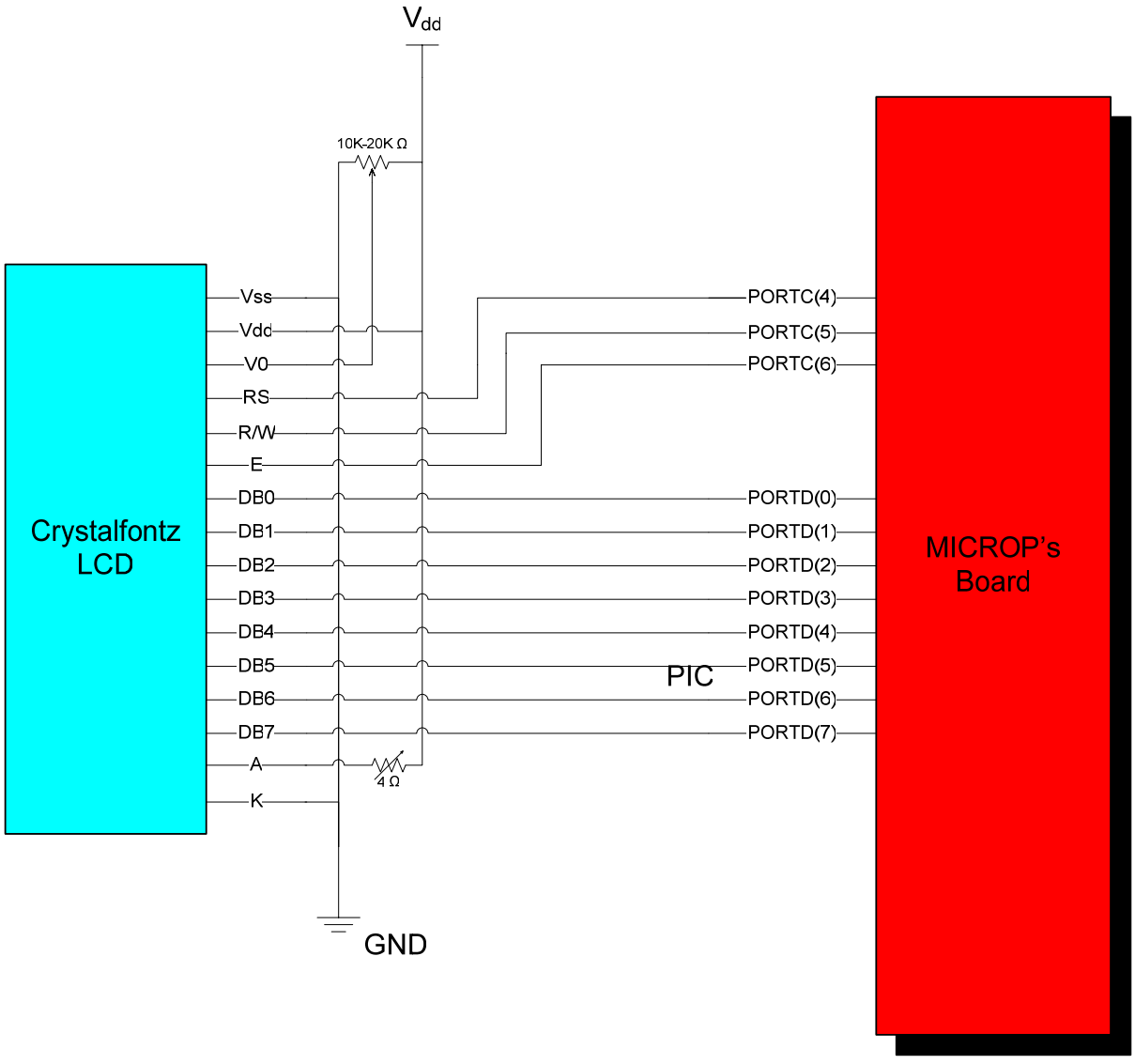
Supplier

Part	Vendor	Part #	Price
LCD	Crystalfontz	CFAH2002A-NYA-JP	$13.45
LCD	Crystalfontz	CFAH1602J-YYB-JP	$11.43

www.crystalfontz.com

As already mentioned the 2002A series all run the same commands, as do most of the other LCDs at the site. It varies only in the timing constraints

Schematic


Vdd
10K-20K Ω
Crystalfontz LCD
Vss
Vdd
V0
RS
R/W
E
DB0
DB1
DB2
DB3
DB4
DB5
DB6
DB7
A
K
4 Ω
GND
PORTC(4)
PORTC(5)
PORTC(6)
PORTD(0)
PORTD(1)
PORTD(2)
PORTD(3)
PORTD(4)
PORTD(5)
PORTD(6)
PORTD(7)
PIC
MICROP's Board

Appendix A: T0CON (Timer0 configuration) Register bit descriptions

bit 7 **TMR0ON:** Timer0 On/Off Control bit
1 = Enables Timer0
0 = Stops Timer0

bit 6 **T08BIT**: Timer0 8-bit/16-bit Control bit
1 = Timer0 is configured as an 8-bit timer/counter
0 = Timer0 is configured as a 16-bit timer/counter

bit 5 **T0CS**: Timer0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: Timer0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Timer0 Prescaler Assignment bit
1 = TImer0 prescaler is NOT assigned. Timer0 clock input bypasses prescaler.
0 = Timer0 prescaler is assigned. Timer0 clock input comes from prescaler output.

bit 2-0 **T0PS2:T0PS0**: Timer0 Prescaler Select bits
111 = 1:256 prescale value
110 = 1:128 prescale value
101 = 1:64 prescale value
100 = 1:32 prescale value
011 = 1:16 prescale value
010 = 1:8 prescale value
001 = 1:4 prescale value
000 = 1:2 prescale value

Note: A prescale value scales the timer0 to count in different amount of delays. For example in our case we chose bit2-0 to be 111, setting the prescale value to be 1:256. This means after 256 clock cycles our timer0 increments by 1. Thus this is the largest delay possible by timer0 in between increments.

```
; Scrolling.asm
; Kevin Lloyd and Rajdeep Roy
; Use the 18F452 PIC microprocessor


; Overview:
; This code displays "Kevin and Raj Bring" "....-=The LCD=-...."
; on two lines and then shifts the lines back and forth.

; The following at the pin connections used in this code
;PORTD(7) - DB7 - Pin 14
;PORTD(6) - DB6 - Pin 13
;PORTD(5) - DB5 - Pin 12
;PORTD(4) - DB4 - Pin 11
;PORTD(3) - DB3 - Pin 10
;PORTD(2) - DB2 - Pin 09
;PORTD(1) - DB1 - Pin 08
;PORTD(0) - DB0 - Pin 07

;PORTC(6) - E   - Pin 6
;PORTC(4) - RS  - Pin 4

    LIST p=18F452
    include "p18f452.inc"

; allocate variables
;LEDS   EQU     0x00
L_Init          EQU     0x21
L_8bit          EQU     0x22
L_Off           EQU     0x23
L_Clear         EQU     0x24
L_EntryMode     EQU     0x25
L_On            EQU     0x26
temp            EQU     0x27

;begin main program
    org 0

start
; Initialization Stuff;;;;;;;;;;;;;;;;;;;;;;;;
    clrf TRISD
    clrf TRISC

    #define LCD_E   PORTC,0x06  ;should be 6
    #define LCD_RS  PORTC,0x04  ;should be 7
;   #define LCD_RW  PORTD,0x03

    movlw b'11000111'           ; Setup the timer to enumerate every 256 cycles
    movwf T0CON

    movlw b'00110000'           ; The standard Initilization code
    movwf L_Init

    movlw b'00111000'           ; The code indicate 8-bit operating mode
    movwf L_8bit

    movlw b'00001000'           ; Turn off the LCD
    movwf L_Off

    movlw b'00000001'           ; Clear the LCD
    movwf L_Clear

    movlw b'00000110'           ; Set the cursor increments when written to the DRAM
    movwf L_EntryMode

    movlw b'00001111'           ; Turn the LCD back on
    movwf L_On
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    call LCD_Init

    call moveRow2
```

```
    clrf TBLPTRU
    movlw 0x10                  ; Set the pointer to address 1000, address for the 1st Message
    movwf TBLPTRH               ; Sets high bits of TBLPTR
    movlw 0x00
    movwf TBLPTRL               ; Sets low bits of TBLPTR

    call DispMes

    call ShiftLeft              ; The following shifts just move the words around
    call Dlay20
    call ShiftLeft
    call Dlay20
    call ShiftLeft
    call Dlay20
    call ShiftLeft
    call Dlay20
    call ShiftLeft
    call Dlay20

    call ShiftRight
    call Dlay20
    call ShiftRight
    call Dlay20
    call ShiftRight
    call Dlay20
    call ShiftRight
    call Dlay20
    call ShiftRight
    call Dlay20

    bra done
LCD_Init
    call Dlay20

;   The below lines are needed if you want to re-initialize the LCD while on.
;
;   movff L_Init, PORTD
;   call Pulse_E
;   call Dlay5
;
;   movff L_Init, PORTD
;   call Pulse_E
;   call Dlay5
;
;   movff L_Init, PORTD
;   call Pulse_E
;   call Dlay5


    movff L_8bit, PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5

    movff L_Off, PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5

    movff L_Clear, PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5

    movff L_EntryMode, PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5

    movff L_On, PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
```

```
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5
    return

moveRow1
    movlw 0x80                  ; Move to the first line, first spot
    movwf PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay5
    return

moveRow2
    movlw 0xC0                  ; Move to the second line, first spot
    movwf PORTD
    bcf LCD_RS                  ; Set RS low, indicating an Instruction
    call Pulse_E                ; Pulse E to tell the LCD that a new Instruction is there
    call Dlay20
    return

DispMes
mes2
    TBLRD*+                     ; Pointer initialization
    movf    TABLAT, 0           ; move the value from the TBLPTR to the w-reg
    xorlw 0x00                  ; if the end of message (return is 0) then
    bz mes2d                    ;    branch to done
    call DispChar
    bra mes2
mes2d
    return

DispChar                        ; Send a character to the LCD (assume char in w-reg)
    movwf PORTD                 ; Send the char code to PORTD and thus to LCD pins
    bsf LCD_RS                  ; Set the RS port high to indicate incomming Data
    call Pulse_E                ; Pulse Enable so that knows to work with the data being sent
    call Dlay5                  ; Wait 5 ns so that the LCD doesn't get overloaded
    return


ShiftLeft                       ; Shift the characters displayed one character to the left
    movlw b'00011000'           ; Command to shift the display left
    movwf PORTD                 ; Send the address to PORTD
    bcf LCD_RS                  ; Set RS low to indicate that an Instruction is being sent
    call Pulse_E                ; Tell the LCD to use the data that is pressently being sent
    call Dlay5                  ; Hold up a bit
    return

ShiftRight                      ; Works same as ShiftLeft, just different op-code
    movlw b'00011100'           ; Command to shift the display right
    movwf PORTD                 ; Send the address to PORTD
    bcf LCD_RS                  ; Set RS low to indicate that an Instruction is being sent
    call Pulse_E                ; Tell the LCD to use the data that is pressently being sent
    call Dlay5                  ; Hold up a bit
    return

Pulse_E                         ; This pulses the Enable bit so that the LCD pays attention
        bsf LCD_E
        nop
        nop
        bcf LCD_E
    return


Dlay20                          ; Dealy 20 ms by delaying 5ms four times
    call Dlay5
    call Dlay5
    call Dlay5
    call Dlay5
    return

Dlay5                           ; Dealy 5 ms by dealying 1ms five times
    call Dlay1
```

```
    call Dlay1
    call Dlay1
    call Dlay1
    call Dlay1
    return

Dlay1                          ; Delay 1ms (little more than)
    movlw b'1001111'            ; The Number x 256 to count up to in order to have a 1ms delay
                                ; found by doing (delay_time/clockcycle_time)/256 prescalar
;   movlw b'0000100'            ; 10011111 is for 20 MHz, 00000100 is for 1 MHz
    clrf TMR0L                  ; Reset the Timer
Dlay1b
    cpfsgt TMR0L                ; If the timer counter is less than the w-reg, keep going
    bra Dlay1b
  return                        ; Goes here when timer is greater than w-reg, done


done
    bra done

    org 0x1000                      ;Messages to be displayed
    ; To display messages we used databanks because the TBLPTR function
    ; allows for easy enumeration through a DB, essentially reading
    ; through it. When storing characters in there it goes character by
    ; character until the 0x00 character is read and then realizes that
    ; it is the end of the sentence.
    ;
    DB  "Raj and Kevin Bring "0x00  ;0x1000
    DB  "....-=The LCD=-.... "0x00  ;0x1016


    end
```