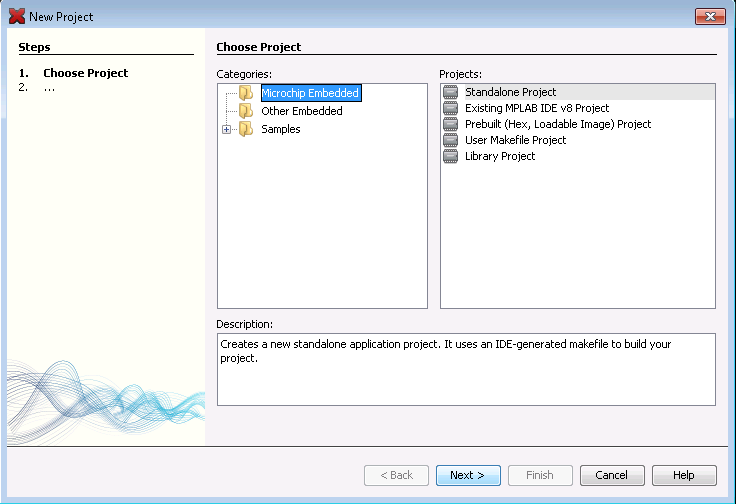
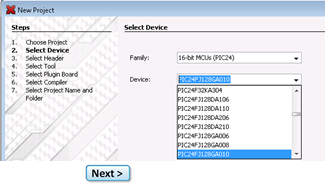
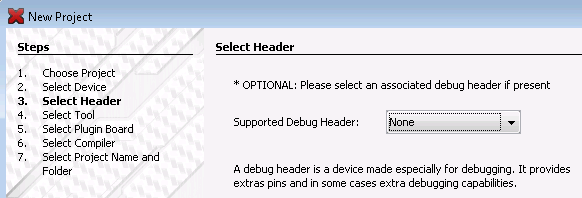
**PART A: Using MPLAB X IDE**

1. **Project Creation**
   1. Open MPLAB® X from your Desktop
   2. Close any open projects in MPLAB® X by right clicking on the Project name and selecting “Close”**.**
   3. Click the “New Project” icon to start the project creation process
   4. Select “Microchip Embedded” then “Standalone Project**”**. 
   5. Click Next
2. **Select the Processor**
   1. Select “Mid-Range 8-bit MCUs (PIC12/16/MCP)” from the ‘Family’ pull down menu, then select “PIC16F917” from the ‘Device’ menu (About 2/3 down the menu). 
   2. Click Next
3. **Header Selection**
   1. No header is needed; select “None” Click Next
4. **Select Hardware** 
   1. Select “ICD 3” under Hardware tools when asked to select a tool. (You will use other debugging hardware options in future labs)
   2. Click Next
5. **Select the Compiler** 
   1. Open the “mpasm” plus box, if not already open, and select: mpasm (v5.57) [C:\Program Files (x86)\Microchip\MPLABX\mpasmx]
   2. Click Next
6. **Select Project Name and Folder**
   1. Project Name: Lastname\_Lab1LED
   2. Location: Will be your C:\ drive. **Note:** Use the same computer every time! Be aware that MPLABX automatically saves the project and reopens the last project on Start-up. Make sure that if you are starting a new lab, you must create a new project or you will alter the previous lab and your old work will be lost.
   3. Click Finish
7. **Add Source File to the project**
   1. Right click “Source Files” in the project window and select “Add Existing Item…”
   2. This is where you will add the Main code file (.asm) for each lab that can be retrieved from Canvas.
8. **Add Header File**
   1. Right click Header Files” and select “Add Existing Item…”
   2. This is where you will add any *include* (.inc) files for each lab that can be retrieved from Canvas.
9. **Opening the Editor**
   1. Double click the Lab file to open the editor. Your code will appear in the right window. This is where you can modify your program. The *include* files can be accessed the same way.
10. **Connect the ICD 3 to the computer using the USB**
11. **Connect your PICDEM Mechatronics Board to the ICD 3**
12. **Make the appropriate pin connections using the jumper wires for the lab (see Part B)**
13. **Building the Project**

**(There are several options for building the project)**

* 1. Click the “Clean and Build Project” icon to build the Project 

This tests the functionality of the program. Then you must send the data to the chip by clicking the “Make and Program Device” icon.

**OR**

* 1. Build a Debug version of the Project by selecting the “Debug Project” which does everything. This will also verify that the operation is working on the hardware.

1. **View the Disassembly window**

This will bring up a window for viewing the disassembled code for each instruction.

* 1. Select *Window>Output>Disassembly Listing File* to open the window. This window is available only after the project has been built.

**PART B**

**Objectives**

1. Use the PIC16F917 to read a tactile switch input.

2. Implement switch debouncing.

3. Toggle a LED when a switch is pushed.

**Connections:**

Found on Pg. 25 of PICDEM Mechatronics User Guide

1. RA0 to a SW2
2. RD7 to a D0 (left most LED)

**Once you have built the project and is working on your board, you must be signed off by the instructor or TA to receive credit for the lab. Do not disconnect your jumpers as we will be building upon this project in LAB 2.**

**The following is a breakdown of the code with comments to the right of each instruction. It will be your responsibility to duplicate this for future labs or exams.**

**The Include File:**

#include <LightLED.inc>

This file includes variable definitions and pin assignments.

#include <p16f917.inc>

\_\_CONFIG \_CP\_OFF & \_CPD\_OFF & \_BOD\_OFF & \_PWRTE\_ON & \_WDT\_OFF & \_INTRC\_OSC\_NOCLKOUT & \_MCLRE\_ON & \_FCMEN\_OFF & \_IESO\_OFF

errorlevel -302 ; supress "register not in bank0, check page bits" message

Pin Declarations:

We must define our pins so that we can reference SW2 and LED and the program will know where to retrieve data.

#define SW2 PORTA,0 ; Tells the program that the switch is connected to bit 0 on

; PORTA

#define LED PORTD,7 ; the LED is located at bit 7 of PORTD. RD7 is connected to D0,

; the leftmost LED

Variable Declarations:

We won’t be using variables in this lab but are necessary for certain operations. They must be declared before the Main program code. The variables are stored in the General Purpose registers of each bank.

Option 1:

cblock 0x20 ; 20h is the location is the start of our General Purpose Registers

; for Bank0.

Temp ; make the name of the variable relevant to its purpose in

var1 ; complex programs.

endc

Option 2:

org 0x20 ; Creates an origin statement to start listing the variables at the

temp ; address of the General purpose Register

var1

Option 3:

bank0 udata ;got to Bank 0 and the data space associated with this bank

var1 res 1 ;reserve 1 byte for “var1”

**Start the Program:**

org 0x00 ; Origin statement that tells the program to start at 0

goto Initialize

org 0x05 ; Initialize starts at 5

**Initialize comparators, internal oscillator, I/O pins, analog pins, variables**:

Initialize

bsf STATUS,RP0 ;sets the RP0 bit of the status register. Sets up bank 1 of

;the Special Function Registers.

The data memory is partitioned into multiple banks which contain the General Purpose Registers (GPRs) and the Special Function Registers (SFRs). Bits RP0 and RP1 are bank select bits. Pg.26 DS

The STATUS register contains:

• the arithmetic status of the ALU

• the Reset status

• the bank select bits for data memory (SRAM)

The STATUS register can be the destination for any instruction, like any other register. Pg.34 DS

Configure I/O pins for project

Each port has a corresponding TRIS (Tri-state Enable) register for configuring the port either as an input or as an output. Setting a TRIS register bit puts the corresponding output driver in high impedance mode and the pin can be used as a digital or analog input. Clearing the bit configures the pin as an output.

bcf TRISD,7 ; Bit clear f. Clears pin 7 of TRISD, making RD7 an output. We

; already defined RD7 as the LED. Functions of RD7 are on

; Pg. 74 DS

bsf TRISA,0 ; Bit set f. Sets pin 0 of TRISA, making RA0 is an input.RA0 is the

; location of our switch. A description of the functions of the

; RA0 pin are on Pg.47 DS

Set internal oscillator frequency

movlw b'01110000' ; move literal to the working register. We can configure the bits

; for desired frequency. bits 6-4, 111=8Mhz

movwf OSCCON ;The data from the working register is written to the OSCCON

;register. Oscillator Control register selection options are found ; ; on Pg. 90 DS.

Set TMR0 parameters

The timer runs continuously and creates a “flag” when it terminates.

movlw b'10000110' ; move literal to working register. Configure timer controls of

; the Option register by setting the bits.

; bit 7, 1=PORTB pull-up disabled

; bit 6, 0=interrupt on falling edge

; bit 5; 0=internal instruction cycle clock

; bit 4, 0=increment on low-to-high transition

; bit 3, 0=prescaler assigned to Timer0 module

; bit 2-0, 110=prescaler rate is set to 1:128

movwf OPTION\_REG ; the prescaler assignment in the Option register is moved to the

; file register. Option register selection options are found on Pg.

; 103 DS

Turn off comparators

This is an extra step that is done because these chips are multiplexed. The pin that is used for RA0 is also used for the comparators. Turning off the comparators will prevent any interference with our program.

movlw 0x07 ; sends 07h to the comparator register

movwf CMCON0 ; configure the comparator controls by setting the bits.

; bits 2-0=111 turns off comparators. Comparator controls are

; found on Pg. 118 DS

Turn off Analog

clrf ANSEL ; clearing the ANSEL register selects a digital I/O. Pg. 45 DS

Initialize Variables

clrf var1 ; clears the value of var1

bcf STATUS,RP0 ; clears the bit and takes us back to bank 0

bcf LED ; clears the bit to LED

**Main Program:**

When SW2 is pressed, the LED is toggled, and the debounce routine for SW2 is initiated. Debouncing a switch is necessary to prevent one button press from being read as more than one button press by the microcontroller. A microcontroller can read a button so fast that contact jitter in the switch may be interpreted as more than one button press.

Main

btfsc SW2 ; Bit test f, skip (to the next line of code) if clear.

goto Main ; Loops here until switch 2 is pressed (SW2 is defined in the

; include file) until it is 0. Once switch 2 is pressed, toggle the

; LED on or off.

movlw 0x80 ; writes a 1 to bit 7 the working register.

xorwf PORTD,f ; exclusive OR the working register with PORTD. Toggles RD7

; (the LED) on and off.

The XOR instruction implements the bitwise XOR operation. The XOR operation sets the resultant bit to 1, if and only if the bits from the operands are different. If the bits from the operands are same (both 0 or both 1), the resultant bit is cleared to 0.

**Debouncing:**

Switch debouncing is done to ensure that mechanical contact chatter in the switch is not mistaken for more than one button push. Debouncing also ensures that for every one press of the button, only one function is executed. In this project that function is toggling a LED. State2 ensures that SW2 is unpressed for 16ms before returning to look for the next time SW2 is pressed. If SW2 is pressed again in this state, then return to State1.

DebounceState1

btfss SW2 ; bit test f, skip if set (Pushing the switch sends a 0)

goto DebounceState1 ; Sets up a loop to wait here until SW2 is released.

clrf TMR0 ; once the switch is released, the timer is cleared in preparation ; to count to 16ms

bcf INTCON,T0IF ; clears timer interrupt flag

DebounceState2

btfss SW2 ; bit test f, skip if set.

goto DebounceState1 ; If the switch is unpressed, it returns to the first debounce

; state.

btfss INTCON,T0IF ;bit test f, skip if set. This allows the program to continue only if

; the timer has terminated. 16ms Time = TMR0\_max \* TMR0

; prescaler \* (1/(Fosc/4)) = 256\*128\*0.5E-6 = 16.4ms

goto DebounceState2

goto Main

END ; directive 'end of program'