CoE 115 Lab 0: Laboratory Introduction

January 2019

Topics/Objectives:

- Microcontroller introduction
- IDE Familiarization
- Application of assembly language
- Understand timing of microcontroller and implement in delay timers

Microcontroller: PIC24FJ64GB002

- PIC24 - Part of the 16-bit family of Micorchip's microcontrollers
- 64 - Refers to the size of program memory available

IDE Familiarization: MPLAB X

This software is available across common OS platforms – Linux, Windows, and Mac. Institute laptops currently run Windows OS, therefore this exercise will be Windows-oriented.

Instructions: Creating your Project:

1. On the Desktop, Create a Folder with the name **CoE115LabSection_LastName.**
2. Go to START Menu and search for MPLAB X.
3. Once the IDE is launched, Go to File > New Project.
4. On the pop-up window, Select Microchip Embedded under Categories and Standalone Project under Projects, then click Next>.
5. Under Families, select **16-bit MCUs (PIC24)** from the drop-down menu. Under Device, scroll and select **PIC24FJ16GB002**. An alternative would be to type **PIC24FJ64GB002** on the Device field. Click Next>.
6. On the Select Tool step, choose **Simulator** and click Next>. (*Note: Once you work with the actual hardware, you must select PICkit 3 as your programmer/debugging tool.*)
7. On the Select Compiler step, select **XC16**. (*Note: Eventually, you may opt to use C language. Also notice that the toolchain you are selecting must have a green highlight bullet to indicate that it is installed in your system, and is fully supported.*)
8. The last step will be to select a location and to name your project. Select the folder you created at the beginning as the project location, and name the project as **CoE115_LastName_Lab0**. Click on Finish.

Developing your Project:

1. On the left hand side of the window, you should see a panel showing your newly created project.
2. Select Linker Files Folder, right click on it and choose to Add Existing Item. Navigate to your computer's Microchip Program Files. Navigate to xc16>v***>Support>PIC24F>gld. Search for the device specific gld file and select it.
3. Select Source Code folder, right click on it, and choose to add a New>AssemblyFile.s. Name this file as **main.s**.
4. Type in the following code:

```
/*
    Author: ***
    Date: December 2016
    */

    .include "p24fj64gb002.inc"
    .global __reset

    .bss
    i: .space 1
    j: .space 1
    k: .space 1

    .text
    __reset:

        mov #__SP_init, W15
        mov #__SPLIM_init, W0
        mov W0, SPLIM

        clr.b i
        clr.b j
        clr.b k

        avalue=100
        mov.b #avalue, W0
        mov.b WREG, i
        inc.b i
        mov.b i,WREG
        mov.b WREG,j
        dec.b j
        mov.b i, WREG
        add.b j, WREG
        mov.b WREG,k

    done:
        goto done
    .end
```

5. Go to Run menu and click on **Build**.

Exploring the Simulator:

1. On the left edge of the coding field, you will see line numbers. Clicking on the line number of a certain instruction line will place a breakpoint there.
2. Add breakpoints at each line of the Stack Pointer initialization.
3. From the Debug menu, click on *Debug Project*.
4. From the Window menu, go to *PIC Memory Views>SFRs*. You should notice at the bottom of the screen, an array of working registers will appear. Scroll down to see the other *Special Function Registers*.
5. The simulation will begin and pause at the first breakpoint (mov #_SP_init, W15). You should observe that WREG15 (W15) now has a value.
6. A set of icons should have appeared at the top of the screen. Hover over each to see their functions (and Function Key shortcuts). In order for the simulation to advance to the next breakpoint, click on **Continue**.
7. You will now observe that the **Program Counter (PC)** increments as you advance through each line of code.
8. You may use the *Continue* command to complete the simulation.
9. Once the simulation reaches the end of the code, before the **done** loop, click on continue one more time. The simulation will now be stuck in the loop. You may use the **Pause** command to stop the looping.
10. From the Window menu, select Debugging>variables. Declared variables i, j, and k must be displayed at the bottom screen. Their stored values should now show.
11. Reset the simulation by clicking on the Reset button/command.
12. Typically, you would only want to observe a select number of registers and variables. In order to monitor a custom selection of these, you may use the **Watches** view from Window> Debugging menu. Just click on an empty line and add the name of the variable or register that you want to monitor. Do this for variables *i, j, k* and the registers *WREG0 and WREG15. Note: You must stop the simulation in order to add items to the watch. Reinitialize the debugging of the project.*
13. Rerun the simulation, observing the values of the variables, then use the **Step Into** command to simulate by instruction line. Observe the variable values as you step into each clear instruction. Observe the values stored in each variable.

Execution Timing and Instruction Cycles

1. From the Window Menu, select Debugging, then open the **Stopwatch** utility. The stopwatch window should show on the bottom panel of the screen. If needed, select to Debug the project again.
2. Notice that initially, the program will be simulated until the first breakpoint. Reset the simulation and use the **Step Into** command to execute each instruction line by line. Take note of the time elapsed for each instruction. Make a prediction of the instruction cycle frequency setting of the simulator.
3. Right-click on the main folder of the project on the left hand pane of the simulator. Go to properties, then select the Simulator Configuration. Under oscillator options, set the instruction cycle frequency to a value of 16 MHz. Click on Apply, then click on OK to close the property window.
4. Reset the simulation (Debug the project again if needed). Using the Step Into command, note the execution time for each line of instruction.

   Note: Looking at the datasheet, it is indicated that for a maximum oscillator frequency of 32MHz, the microcontroller can execute up to a maximum of 16MIPS. This translates to 2 clock cycles per instruction.

Seatwork:

Modify the sample code to implement an 8-bit timer that increments every 500 ms. Use the simulator tools you've explored to help you verify your code.

- Do not change the instruction cycle frequency (must currently be set to 16MHz from step 3 above).
- Stopwatch will be reset from the first increment to the timer (initialization time will not be counted)
- Set an arbitrary variable to serve as your timer/counter