CALIFORNIA STATE POLYTECHNIC UNIVERSITY, POMONA
COLLEGE OF ENGINEERING

ECE 3301L Spring 2025 Session 1 Microcontroller Lab **Felix Pinai**

LAB3
Introduction to Assembly language

In this lab, we are going to write in Assembly language instead of C language.

We are going to use the same hardware connection as in Lab #2 with the exception that the connection to PORTE bit 2 (RE2) is connected to the Logic Analyzer 7

PART 1)

As you are familiar by now with the use of MPLAB X, you will need to do the same to compile an assembly program as follows:

1) Go to the Projects box.
2) Select the project Lab3p1
3) Right click and scroll down to 'Copy' and click on it
4) A box will appear with the name of the original project. Change the name of the Project to be 'lab3p1' to create part 1) of lab3
5) The project location should be with the new directory lab3\Part1
6) Select the button 'Copy' to create the new project
7) Once the new project is created, go to that project in the 'Projects' area and right click on that new project and scroll down to 'Set as Main Project' and click on that. After that, the project name should be in bold
8) Go back and select that project again and right click on it
9) Scroll all the way down to 'Properties'
10) A new window will pop up. On the right side under the 'Compiler Toolchain', instead of selecting the XC8 compiler, you will need to select the 'mpasm' option. select a version of the assembler under 'mpasm' and hit 'OK'
11) We are not going to use the C source code but instead the Assembly source code. Now you are at the step to add the new file, do File>New File. A new window will appear. Select 'Assembler' then 'AssemblyFile.asm' and hit Next. You will need to enter the new file name. In this case, I would call it 'Lab3p1'. Hit Finish.

The next phase is to create the assembly file. Copy the following text and paste into the file.

```
; THIS FIRST ASSEMBLY LANGUAGE PROGRAM WILL FLASH LEDS
; CONNECTED TO THE PINS 0 THROUGH 3 OF PORT B

#include<P18F4620.inc>

        config          OSC = INTIO67
        config          WDT = OFF
        config          LVP = OFF
        config          BOREN = OFF


; Constant declarations

Delay1      equ         0xFF
Delay2      equ         0XFF

Counter_L   equ         0x20
Counter_H   equ         0x21

        ORG         0x0000

 ; CODE STARTS FROM THE NEXT LINE


START:

        MOVLW       0x0F                ;
        MOVWF       ADCON1              ;

        MOVLW       0x00                ;
        MOVWF       TRISB               ;
        MOVWF       TRISE               ;


MAIN_LOOP:

        MOVLW       0x05                ;
        MOVWF       PORTB               ;
        MOVWF       PORTE               ;
                                        ;
        CALL        DELAY_ONE_SEC       ;


        MOVLW       0x0A                ;
        MOVWF       PORTB               ;
        MOVWF       PORTE               ;

        CALL        DELAY_ONE_SEC       ;
```

```
        GOTO        MAIN_LOOP       ;
; the routine below is a subroutine and it is called 'DELAY_ONE_SEC'.

DELAY_ONE_SEC:
        MOVLW       Delay1          ;
        MOVWF       Counter_H       ;

LOOP_OUTER:
        NOP                         ;
        MOVLW       Delay2          ;
        MOVWF       Counter_L       ;

LOOP_INNER:
        NOP                         ;
        DECF        Counter_L,F     ;
        BNZ         LOOP_INNER      ;

        DECF        Counter_H,F     ;
        BNZ         LOOP_OUTER      ;
        RETURN

; end of subroutine 'DELAY_ONE_SEC'

        END
```

Here are some definitions of the assembly instructions used above:

```
MOVLW     literal          ; move 'literal' value into register W
MOVWF     location         ; move content of W into memory 'location'
CALL      subroutine       ; call the 'subroutine'
NOP                        ; No operation – do nothing
GOTO      location         ; jump to the new 'location'
DECF      location, F      ; decrement the content of 'location' by 1
                           ; and store the new value into 'location'
BNZ       location         ; branch if the result is not zero
RETURN                     ; return from subroutine
```

Run and execute the program. Notice the 4 LEDs on PORT B blinking.

Start the Logic Analyzer and pay attention **to channel 7 whereas it is connected to Port E bit 2**. That signal will alternately change. Measure the time duration of the half period. Next, we want to make that time to be about 1 second with **5 milliseconds** tolerance (+/- 5 milliseconds). Go back to the assembly program and look for the variables Delay1 or Delay2. **Modify either or both of those variables in order to make the time to be 1 second with the required tolerance. Capture the waveform for the report.**

PART 2)

The next project is to implement the assembly code that is equivalent to part 1) of lab #2. In short, we are to read the three switches connected to PORT A and display them to the LEDs connected to PORTB.

C Code:

```
ADCON1 = 0x0F;
TRISA = 0xFF;
TRISB = 0x00;

while (1)
{
        In = PORTA;
        In = In & 0x0F;
        PORTB = In;
}
```

Compile and run the following program (make sure that this is in a new folder called lab3p2):

```
; THIS SECOND ASSEMBLY LANGUAGE PROGRAM WILL READ THE VALUES OF
; ALL THE BITS 0-2 OF PORT A AND OUTPUT THEM
; TO THE PINS 0 THROUGH 2 OF PORT B

#include <P18F4620.inc>

        config      OSC = INTIO67
        config      WDT = OFF
        config      LVP = OFF
        config      BOREN = OFF


        ORG         0x0000

START:

        MOVLW       0x0F            ;
        MOVWF       ADCON1          ;

        MOVLW       0xFF            ;
        MOVWF       TRISA           ;
        MOVLW       0x00            ;
        MOVWF       TRISB           ;
```

```
MAIN_LOOP:                          ;
        MOVF        PORTA, W        ;
        ANDLW       0x0F            ;
        MOVWF       PORTB           ;

        GOTO        MAIN_LOOP       ;
        END
```

After you have compiled and downloaded the program into the board, change one switch at a time and check that the corresponding LED does change according to the logic state of the switch.

PART 3)

Next, your team will implement part 2) of Lab #2 in assembly.

Take the code in the above Part 2) and modify it to control the RGB LED D1 connected to PORTC. Just use the c code done in Lab #2 part 2) as reference and change it into assembly based on the example code provided above. Don't forget to add the additional TRISC to control the output of PORTC.

PART 4)

We will implement now the part 3) of Lab #2. We do need to write an infinite loop with an internal loop that count from 0 to 7 and then repeat itself while outputting that count to PORTC and then call a subroutine to delay 1 second.

The following program will implement the FOR loop by using an up counter saved at the location 0x28 and it is used as an index for the color to be outputted to the PORTC. Let us call that counter as 'Color_Value' and we will load with a starting value of 0.

We will use another counter at location 0x29h and we will call that counter 'Loop_Count' and we will initialize it with the value of 08h at the start.

The counter at Color_Value will be incremented by 1 each time through the loop while the counter Loop_Count will be decremented by 1. When it reaches the value of 0, the FOR loop is completed.

The subroutine 'DELAY_ONE_SEC' is called once a color is outputted to the port for the purpose of creating a long delay to allow the color to be displayed for a good amount of time. Remember that routine 'DELAY_ONE_SEC' can be copied from this lab's part 1. Make sure to use the adjusted values for Delay1 and Delay2

```
#include <P18F4620.inc>
 config OSC = INTIO67
 config WDT = OFF
 config LVP = OFF
 config BOREN = OFF

Counter_L     equ          0x20
Counter_H     equ          0x21
Color_Value   equ          0x28
Loop_Count    equ          0x29

 ORG 0x0000



; CODE STARTS FROM THE NEXT LINE
START:
        ORG          0x0000

START:
        MOVLW        0x0F                ;
        MOVWF        ADCON1              ;

        MOVLW        0x00                ;
        MOVWF        TRISC               ;


WHILE_LOOP:                              ;
        MOVLW        0x00                ;
        MOVWF        Color_Value         ;
        MOVLW        0x08                ;
        MOVWF        Loop_Count          ;

FOR_LOOP:
        MOVF         Color_Value,W       ;
        MOVWF        PORTC               ;
        CALL         DELAY_ONE_SEC       ;

        INCF         Color_Value,F       ;
        DECF         Loop_Count,F        ;
        BNZ          FOR_LOOP            ;
                                         ;
        GOTO         WHILE_LOOP          ;

; add the code for 'DELAY_ONE_SEC' here

        END
```

Remember to add the code under 'DELAY_ONE_SEC' from part 1) of this lab to the above code (before 'END').

PART 5)

From the array(s) generated in lab #2 part 5), we will use its content to fill in two sequences of 8 values. For example for the first array of data, use an arbitrary location like 0x40. Put in the first value of the array at that location 0x40. Next, take the second value and place it at the next location 0x41. Repeat the process for the remaining values of the array. When completed, all the locations from 0x40 to 0x47 should have the values of the first array from the previous lab.

The next step is to repeat the same process for the second array and pick another starting location like 0x50.

For example, if the first two values of the first array is xx and yy. We will need to move those two values to location 0x40 and 0x41. We will simply do the following:

```
MOVLW    xx
MOVWF    0x40
MOVLW    yy
MOVWF    0x41
```

Keep doing the same process with the 6 additional values for the first array.

Next, repeat the same process for the 8 values of the second array and use the starting memory locations at 0x50. However, pay attention to the schematics and especially the connections for the two RGB LEDs D1 and D2. For D1, the three connections start from bit 0 and end at bit 2. That is the normal alignment to match with a decimal number. On the other hand, the connections for the RGB LED D2 are different. The three connections start from bit 3 and end at bit 5. If we output directly the decimal values that we have obtained from the previous lab, the values will go directly to PORT D bits 0 to 2 and not to the proper bits (bits 3 through 5). To properly output the values, we will need to shift the values with the proper number of bits. In this case, we will need to shift them by 3 bits or equivalently to multiply the values by 2^3 or 8. You will need to modify the 8 original values of the second array.

After the creation of those two arrays are completed, now we will use the indirect addressing mode (with the registers FSR0 and FSR1) to fetch the color value to be outputted to the PORT(s) associated with the LEDs D1 and D2. Use the instruction:

```
LFSR        x,address
```

Where ‘x’ can be 0,1,or 2 for the FSR0, FSR1, FSR2 respectively
And ‘address’ is the value of the address where the FSR register needs to point to.
Therefore, if we want to point FSR0 to the address 0x40, we will write:

```
LFSR        0,0x0040
```

Do the same for FSR1 to point to the address 0x50.

To fetch the data pointed by the FSR register into the W register, we will use the instruction:

```
MOVF        INDFx, W
```

Where ‘x’ is the number of the FSR register.

Once the data is loaded in the W, then we can move the content of W into the appropriate port:

```
MOVWF       PORTx
```

To increment the FSR register by 1 in order to point to the next value in the array, use the instruction:

```
INCF        FSRxL
```

Use the code from Part 4) above to add the changes to implement the same function as in lab2 part 5).

Notes: Don’t forget to program the TRISx registers used by the RGB LEDs D1 and D2.

Once the program is completely implemented, run it and verify that the color sequence is the same as in the previous lab. Use the logic analyzer to capture the waveforms for the LEDs D1 and D2 and use the capture to verify that the implementation is correct.