Lab 02

LED Blinking

EECE 4038C Embedded Systems Design

Instructor: Professor Jon Lockhart

Fall 2023

A report By Mabon Ninan(M13883690)

Abstract:

The purpose of this lab is to demonstrate our capabilities in utilizing multiple switches and LED bulbs of the PIC Microcontroller using the MPLAB X IDE. While the previous lab was based on a simple experiment with involved only a single LED and switch pair, this lab challenges us to use 2 pairs of LED and Switches with non-depended (in this case a sort of congruent) functionality. The complexity in this lab stems from using multiple switches, and thus multiple registers to carry specific tasks. While lab 1 only required the use of one register to store the input from a single switch, this lab adds complexity in that we are required to use two switch's and concurrently two registers.

Methodology:

Pre-lab Preparation:

- I ensured that I had completed all the required tasks for lab 01 including familiarizing myself with the MPLAB X IDE.
- In preparation for the lab, I wrote an assembly code to implement the required objectives on the demo board:

  The objective of this experiment is to control the blinking behavior of on-board LEDs (DS1 and DS4) using two push buttons (SW1 and SW2) on the hardware platform. Specifically, the LEDs will be controlled as follows:

  > DS1 LED will blink when SW1 is pressed and stop blinking when SW1 is pressed again.
  >
  > DS4 LED will blink when SW2 is pressed and stop blinking when SW2 is pressed again.

Lab Experiments:

Step 1: Setup

1. Ensure the hardware platform is powered and functional.
2. Connect DS1 and DS4 LEDs to appropriate GPIO pins on the hardware platform.

Step 2: Initial State

1. Turn on the hardware platform.
2. Verify that DS1 LED is initially ON and producing a solid light, while DS4 LED is initially OFF.

Step 3: DS1 LED Blinking Control

1. Connect SW1 .

2. Write code to check the state of SW1:
    - If SW1 is pressed and DS1 LED is solid, start blinking DS1 LED at the rate of the on-board clock.
    - If SW1 is pressed and DS1 LED is blinking, stop blinking DS1 LED and make it solid again.
3. Upload the code to the hardware platform.
4. Verify that pressing SW1 toggles DS1 LED between solid and blinking states.

Step 4: DS4 LED Blinking Control

1. Connect SW2 .
2. Write code to check the state of SW2:
    - If SW2 is pressed and DS4 LED is OFF, start blinking DS4 LED at the rate of the on-board clock.
    - If SW2 is pressed and DS4 LED is blinking, stop blinking DS4 LED and turn it OFF.
3. Upload the code to the hardware platform.
4. Verify that pressing SW2 toggles DS4 LED between OFF and blinking states.

Step 5: Repeating Control

1. Modify the code for SW1 and SW2 to make the LED control repeatable.
    - For SW1: If DS1 LED is solid, pressing SW1 will start blinking; pressing it again will make it solid again.
    - For SW2: If DS4 LED is OFF, pressing SW2 will start blinking; pressing it again will turn it OFF.
2. Upload the modified code to the PIC hardware platform.
3. Verify that pressing SW1 repeatedly toggles DS1 LED between solid and blinking states, and pressing SW2 repeatedly toggles DS4 LED between OFF and blinking states.

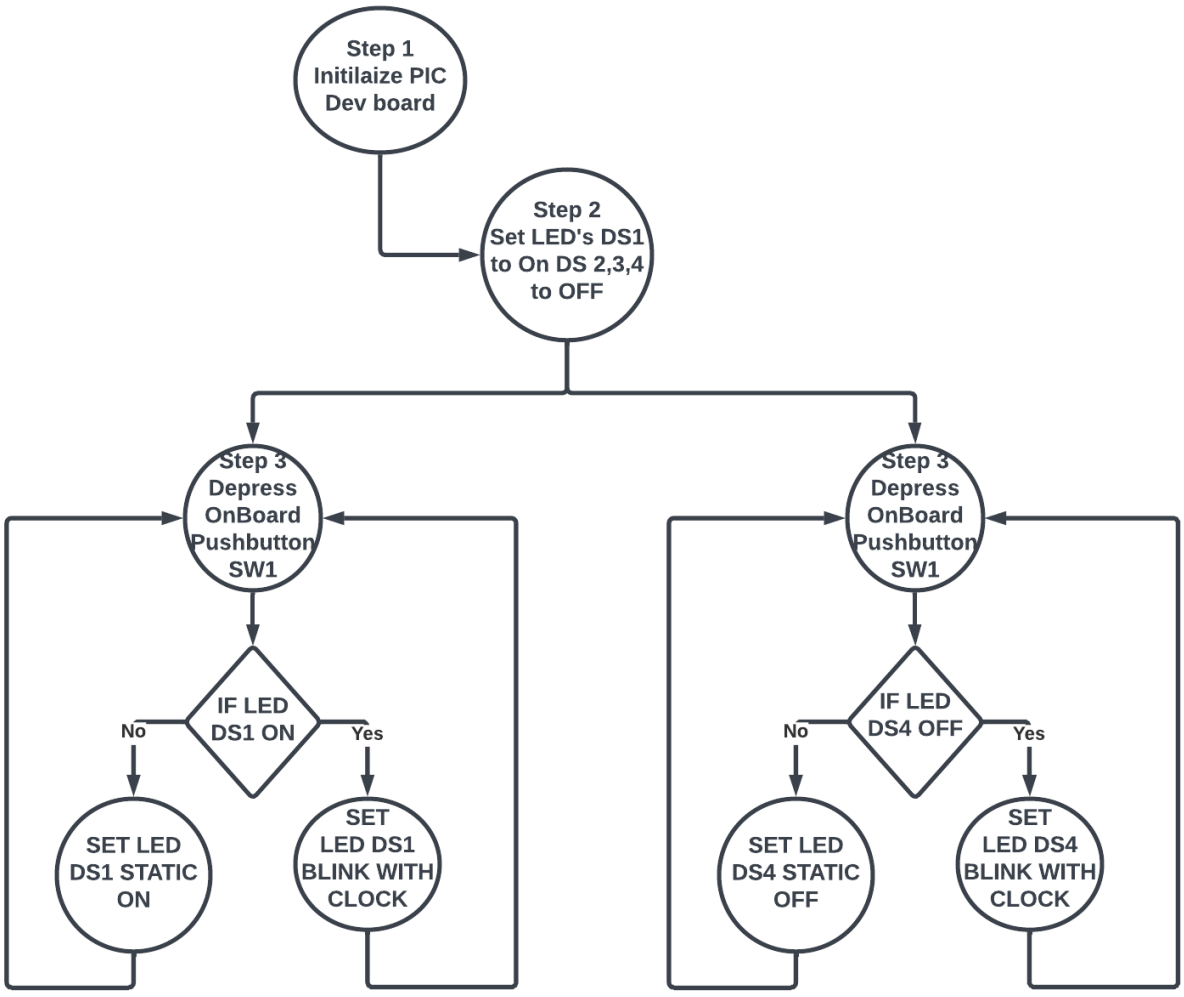
Lab Documentation:

- During the experiment, I took clear photographs and captured screenshots at key stages of the experiment, including the initial state (ie DS1 Solid On, DS4 Led Off), DS1 Blinking after depressing SW1 (while DS4 is Off). Upon depressing SW2 we should observe that DS4 starts blinking (in this state we have both DS1 and DS4 Blinking). Now depressing SW2, DS4 turns off while DS1 continues to blink. Finally, when you depress SW 1 DS1 returns to a solid state.

Project Requirements:

Microchip PIC DM164130-3 28-Pin LIN Demo Board, MPLAB X IDE, USB cable, Windows Laptop

Design Diagram:



Code:

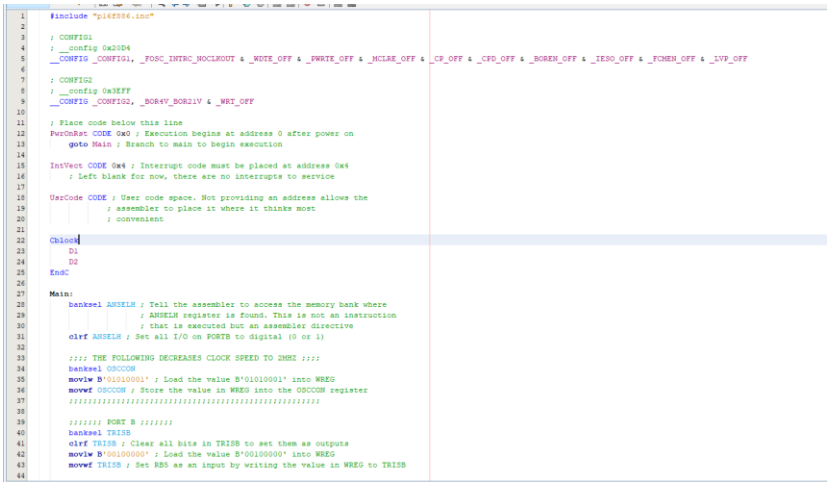
```
#include "p16f886.inc"

; CONFIG1

; __config 0x20D4

__CONFIG _CONFIG1, _FOSC_INTRC_NOCLKOUT & _WDTE_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _CPD_OFF & _BOREN_OFF & _IESO_OFF & _FCMEN_OFF & _LVP_OFF

; CONFIG2

; __config 0x3EFF

__CONFIG _CONFIG2, _BOR4V_BOR21V & _WRT_OFF

; Place code below this line

PwrOnRst CODE 0x0 ; Execution begins at address 0 after power on

   goto Main ; Branch to main to begin execution

IntVect CODE 0x4 ; Interrupt code must be placed at address 0x4

   ; Left blank for now, there are no interrupts to service

UsrCode CODE ; User code space. Not providing an address allows the

         ; assembler to place it where it thinks most

         ; convenient

Cblock

   D1

   D2

EndC


Main:

   banksel ANSELH ; Tell the assembler to access the memory bank where

            ; ANSELH register is found. This is not an instruction

            ; that is executed but an assembler directive

   clrf ANSELH ; Set all I/O on PORTB to digital (0 or 1)


   ;;;; THE FOLLOWING DECREASES CLOCK SPEED TO 2MHZ ;;;;

   banksel OSCCON

   movlw B'01010001' ; Load the value B'01010001' into WREG

   movwf OSCCON ; Store the value in WREG into the OSCCON register

   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
    ;;;;;;; PORT B ;;;;;;;
    banksel TRISB
    clrf TRISB ; Clear all bits in TRISB to set them as outputs
    movlw B'00100000' ; Load the value B'00100000' into WREG
    movwf TRISB ; Set RB5 as an input by writing the value in WREG to TRISB

    ;;;;;;; PORT E ;;;;;;;
    banksel TRISE
    clrf TRISE ; Clear all bits in TRISE to set them as outputs
    movlw B'00001000' ; Load the value B'00001000' into WREG
    movwf TRISE ; Set RE3 as an input by writing the value in WREG to TRISE
    ;;;;;;;;;;;;;;;;;;;;;;

    banksel PORTB
    clrf PORTB ; Clear all bits in PORTB to initialize it
    goto main_loop

main_loop:
    bsf PORTB, 0 ; Set bit 0 of PORTB
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto DS1_func ; Jump to DS1_func if bit 3 is clear
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto DS4_func ; Jump to DS4_func if bit 5 is clear
    goto main_loop ; Continue looping if neither condition is met

DS1_func:
    bsf PORTB, 0 ; Set bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 0 ; Clear bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
```

```
    goto both_func ; Jump to both_func if bit 5 is clear
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto main_loop ; Jump back to main_loop if bit 3 is clear
    goto DS1_func ; Continue DS1_func if neither condition is met

DS4_func:
    bsf PORTB, 3 ; Set bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto both_func ; Jump to both_func if bit 3 is clear
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto main_loop ; Jump back to main_loop if bit 5 is clear
    goto DS4_func ; Continue DS4_func if neither condition is met

both_func:
    bsf PORTB, 3 ; Set bit 3 of PORTB
    bsf PORTB, 0 ; Set bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    bcf PORTB, 0 ; Clear bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    ;;;; NEED TO CHECK FOR BOTH BUTTON ;;;;;
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto DS1_func ; Jump to DS1_func if bit 5 is clear
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto DS4_func ; Jump to DS4_func if bit 3 is clear
    goto both_func ; Continue both_func if neither condition is met

NO_OP:
    nop ; No operation (delay)
    decfsz D1, 1 ; Decrement D1 and skip next instruction if result is zero
```

```
    goto $-1 ; Jump to the previous NOP instruction if D1 is not zero
    nop ; No operation (delay)
    nop ; No operation (delay)
    decfsz D2, 1 ; Decrement D2 and skip next instruction if result is zero
    goto NO_OP ; Jump to the NO_OP subroutine if D2 is not zero
    return ; Return from subroutine
; Needed for compilers to indicate the end of the program code
; It's a directive so needs to be inline with the code
end
```

Inserted bellow is a screen shot of the code:



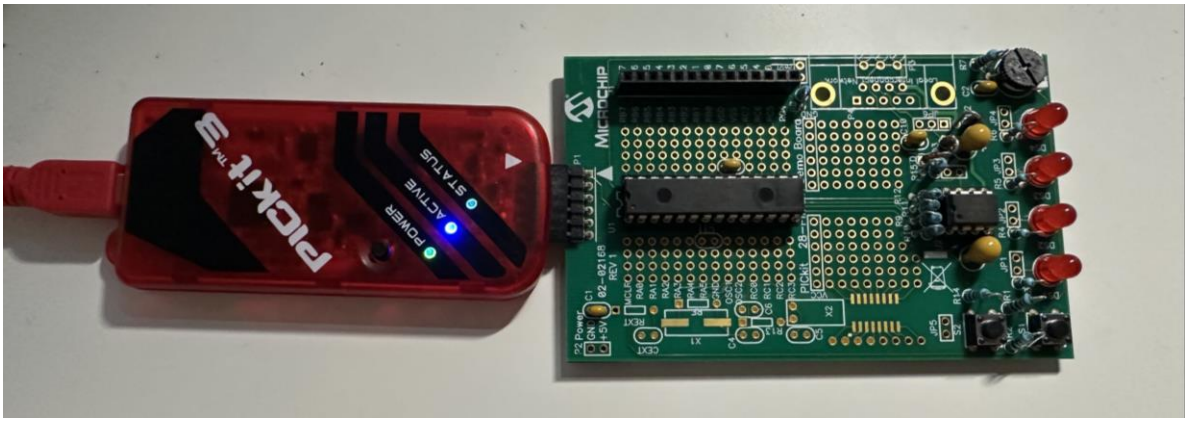
```
#include "p16f886.inc"

; CONFIG1
; __config 0x2GD4
__CONFIG _CONFIG1, _FOSC_INTRC_NOCLKOUT & _WDTE_OFF & _PWRTE_OFF & _MCLRE_OFF & _CP_OFF & _CPD_OFF & _BOREN_OFF & _IESO_OFF & _FCMEN_OFF & _LVP_OFF

; CONFIG2
; __config 0x3EFF
__CONFIG _CONFIG2, _BOR4V_BOR21V & _WRT_OFF

; Place code below this line
PwrOnRst CODE 0x0 ; Execution begins at address 0 after power on
    goto Main ; Branch to main to begin execution

IntVect CODE 0x4 ; Interrupt code must be placed at address 0x4
    ; Left blank for now, there are no interrupts to service

UsrCode CODE ; User code space. Not providing an address allows the
             ; assembler to place it where it thinks most
             ; convenient

Chleck
    D1
    D2
EndC

Main:
    banksel ANSELH ; Tell the assembler to access the memory bank where
                   ; ANSELH register is found. This is not an instruction
                   ; that is executed but an assembler directive
    clrf ANSELH ; Set all I/O on PORTB to digital (0 or 1)

    ;;;; THE FOLLOWING DECREASES CLOCK SPEED TO 2MHZ ;;;;
    banksel OSCCON
    movlw B'01010001' ; Load the value B'01010001' into WREG
    movwf OSCCON ; Store the value in WREG into the OSCCON register
    ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

    ;;;;;;; PORT B ;;;;;;;
    banksel TRISB
    clrf TRISB ; Clear all bits in TRISB to set them as outputs
    movlw B'00100000' ; Load the value B'00100000' into WREG
    movwf TRISB ; Set RB5 as an input by writing the value in WREG to TRISB
```

```
    ;;;;;;; PORT E ;;;;;;;
    banksel TRISE
    clrf TRISE ; Clear all bits in TRISE to set them as outputs
    movlw B'00001000' ; Load the value B'00001000' into WREG
    movwf TRISE ; Set RE3 as an input by writing the value in WREG to TRISE
    ;;;;;;;;;;;;;;;;;;;;;;;

    banksel PORTB
    clrf PORTB ; Clear all bits in PORTB to initialize it
    goto main_loop

main_loop:
    bsf PORTB, 0 ; Set bit 0 of PORTB
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto DS1_func ; Jump to DS1_func if bit 3 is clear
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto DS4_func ; Jump to DS4_func if bit 5 is clear
    goto main_loop ; Continue looping if neither condition is met

DS1_func:
    bsf PORTB, 0 ; Set bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 0 ; Clear bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto both_func ; Jump to both_func if bit 5 is clear
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto main_loop ; Jump back to main_loop if bit 3 is clear
    goto DS1_func ; Continue DS1_func if neither condition is met

DS4_func:
    bsf PORTB, 3 ; Set bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto both_func ; Jump to both_func if bit 3 is clear
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto main_loop ; Jump back to main_loop if bit 5 is clear
    goto DS4_func ; Continue DS4_func if neither condition is met
```

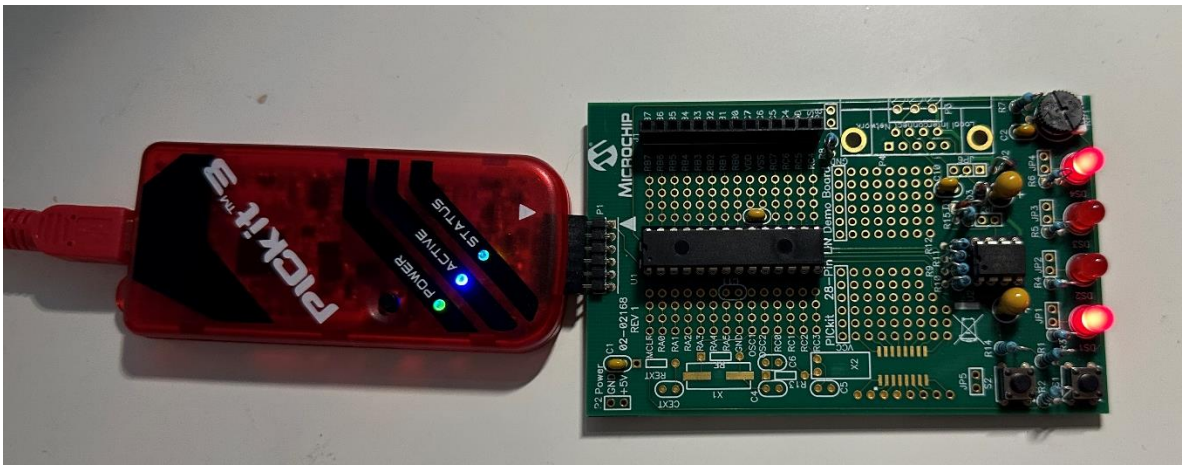
```

both_func:
    bsf PORTB, 3 ; Set bit 3 of PORTB
    bsf PORTB, 0 ; Set bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    bcf PORTB, 3 ; Clear bit 3 of PORTB
    bcf PORTB, 0 ; Clear bit 0 of PORTB
    call NO_OP ; Call the NO_OP subroutine
    ;;;; NEED TO CHECK FOR BOTH BUTTON ;;;;;
    btfss PORTB, 5 ; Test if bit 5 of PORTB is clear (skip if set)
    goto DS1_func ; Jump to DS1_func if bit 5 is clear
    btfss PORTE, 3 ; Test if bit 3 of PORTE is clear (skip if set)
    goto DS4_func ; Jump to DS4_func if bit 3 is clear
    goto both_func ; Continue both_func if neither condition is met

NO_OP:
    nop ; No operation (delay)
    decfsz D1, 1 ; Decrement D1 and skip next instruction if result is zero
    goto $-1 ; Jump to the previous NOP instruction if D1 is not zero

    nop ; No operation (delay)
    nop ; No operation (delay)
    decfsz D2, 1 ; Decrement D2 and skip next instruction if result is zero
    goto NO_OP ; Jump to the NO_OP subroutine if D2 is not zero
    return ; Return from subroutine

; Needed for compilers to indicate the end of the program code
; It's a directive so needs to be inline with the code
end

```

Results:

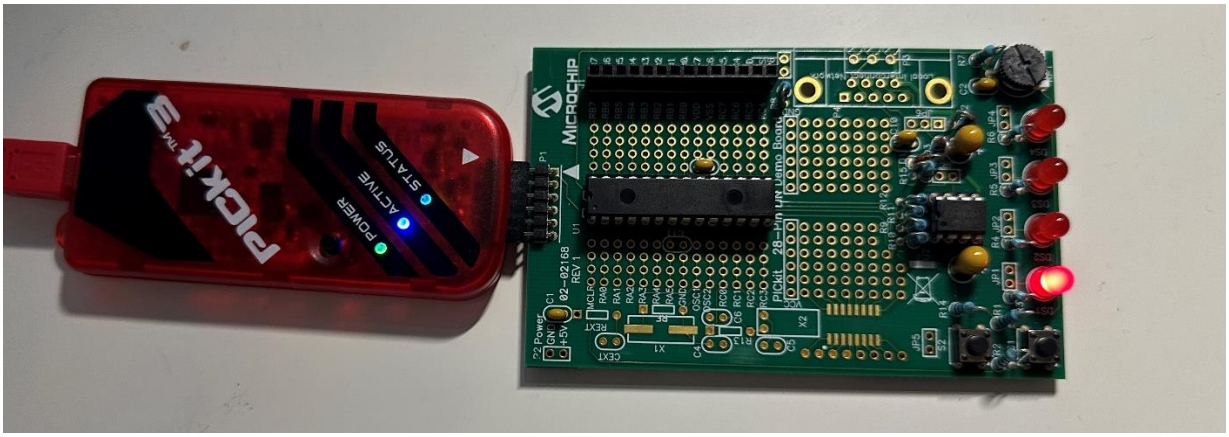
1 PIC Initialized



2 SW1 Depressed (DS1 Blinking)



3 SW2 Depressed (Both DS1 and DS4 are Blinking)



4 SW1 and SW2 are Depressed (DS1 Set Solid, DS4 Set OFF)

Conclusion:

In conclusion, this lab helped us to build on the foundation laid out in the first lab of this course. The new concepts explored here included the utilizing of the clock, the use of both switches and 2 LED's. Here we demonstrate that we are able to write instruction code in such a format that allows for independent operation of the two LEDs in question and their corresponding unique input from the two switches. Through this hands-on experience, we have successfully demonstrated our ability to create code that orchestrates the unique behavior of each LED in response to the distinct input signals from the two switches. This accomplishment marks a significant milestone in our journey of mastering microcontroller programming, equipping us with the skills and confidence to tackle more intricate projects in the future.