

Micros Peripherals and Interfaces assignment 1

by Joshua Port

Submission date: 28-Nov-2022 09:05AM (UTC+0000)

Submission ID: 190274518

File name: 32728_Joshua_Port_Micros_Peripherals_and_Interfaces_assignment_1_466098_1870855693.docx (1.73M)

Word count: 5700

Character count: 25675

Joshua J Port 2114468
BEng Embedded Electronic Systems
University of Wales Trinity St David's
Micros Peripherals and Interfaces
(ACDA4004)

Table of Contents

Section 1: 4

 Summary.....4

Section 2: 9

Section 3: 13

 Section 3.1: 13

 Section 3.2: 14

 Section 3.3: 16

References: 40

Figure 1- Full Count Circuit Design	4
Figure 2 - PIC Circuit Component	5
Figure 3 - Seven Segment Display	5
Figure 4 - Binary data sheet for the 7-segment display	6
Figure 5 – Buttons	6
Figure 6 - Crystal Oscillator	7
Figure 7 - Speaker	7
Figure 8 - MCLR resistor	8
Figure 9 - Resistor protecting the Speaker	8
Figure 10 - Binary data sheet for the 7-segment display	9
Figure 11 - LED binary numbers	10
Figure 12 - Get_count loop	10
Figure 13 - Get_count repeat	11
Figure 14 - one_sec_pause function	11
Figure 15 - Pause function	11
Figure 16 - loadN function	11
Figure 17 - Decrement N function	12
Figure 18 - Get count flow chart	13
Figure 19 - Switch B and G flow chart	14
Figure 20 - Event flag black switch	15

Section 1:

Summary

The count circuit is used to count from 0 to 255 and display the correct number using the LED's of a seven-segment display.

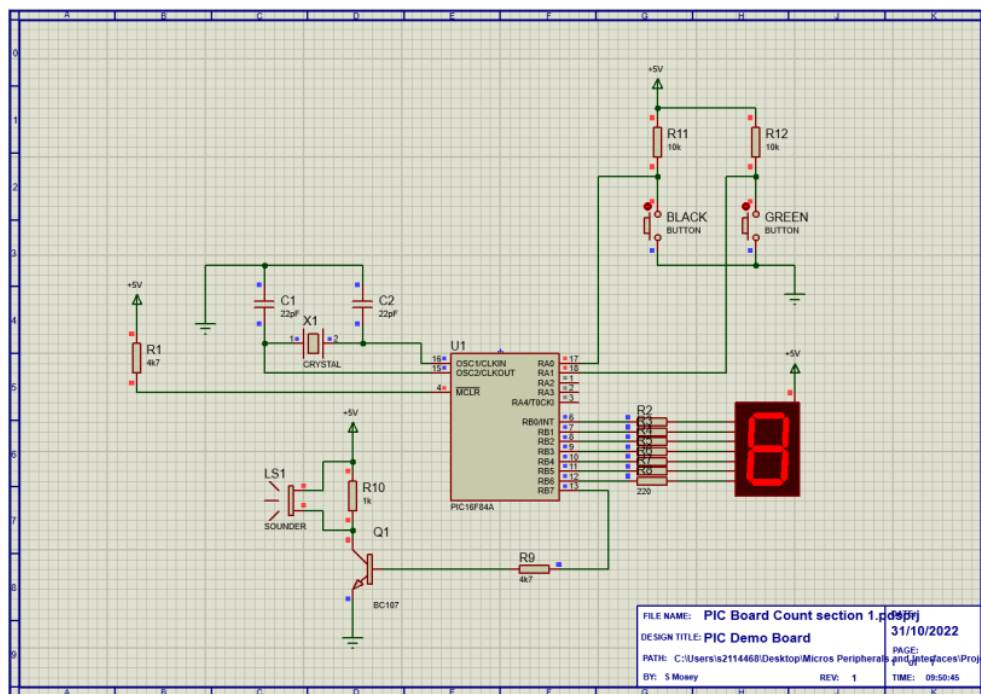


Figure 1- Full Count Circuit Design

Figure one shows the count circuit I was using to count from 0 – 255 in binary format. In this figure the seven-segment display shows all 7 input ports of the seven-segment display as low meaning all 7 LED's will be on. When the pulse from the programmable integrated chip is high, the port it is connected to on the display will not be lit. The seven-segment display currently shows the number 0 in binary form.

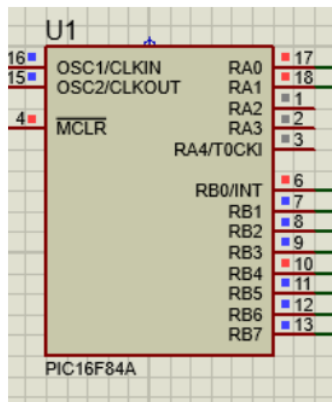


Figure 2 - PIC Circuit Component

1. RB0 – Input pin for bit 0. This pin connects to segment A on the 7-segment display.
2. RB1 – Input pin for bit 1. This pin connects to segment B on the 7-segment display.
3. RB2 – Input pin for bit 2. This pin connects to segment C on the 7-segment display.
4. RB3 – Input pin for bit 3. This pin connects to segment D on the 7-segment display.
5. RB4 – Input pin for bit 4. This pin connects to segment E on the 7-segment display.
6. RB5 – Input pin for bit 5. This pin connects to segment F on the 7-segment display.
7. RB6 – Input pin for bit 6. This pin connects to segment G on the 7-segment display.

All the input pins use Transistor – Transistor Logic (TTL). [1]

Use of 1st Person Voice

The PIC16F84A is responsible for running the seven-segment display. I am able to program the chip to undertake any task that I can set on the display. I do this by using the ports above. The purpose of having a PIC is to separate the data from the I/O pins so we can run separate operations effectively and change the operations required to complete specific tasks. [2]

The PIC16F84a also comes with an internal oscillator.

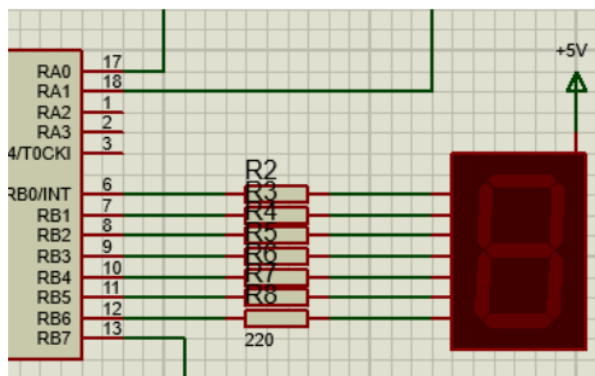
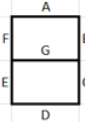


Figure 3 - Seven Segment Display

Shown on the right-hand side of figure 3 is the seven-segment display. The seven-segment display is made up of seven sections each containing one led. Using the PIC, I can program the component to

count when another component is activated, auto count or down, set specific patterns or in this case count in binary and hexadecimal. Each pin is connected to a separate pin on the PIC and there is a resistor between this connection and that of the display. The reason for having the resistor there is to limit the power able to go to the LEDs to stop them from blowing up or being damaged. The size of the resistance required is 220 ohms and so R2-R8 will all be 220-ohm resistors.

Hex	Buzz	G	F	E	D	C	B	A	Char
0x40	0	1	0	0	0	0	0	0	0
0x79	0	1	1	1	1	0	0	1	1
0x24	0	0	1	0	0	1	0	0	2
0x30	0	0	1	1	0	0	0	0	3
0x19	0	0	0	1	1	0	0	1	4
0x12	0	0	0	1	0	0	1	0	5
0x02	0	0	0	0	0	0	1	0	6
0x78	0	1	1	1	1	0	0	0	7
0x00	0	0	0	0	0	0	0	0	8
0x10	0	0	0	1	0	0	0	0	9
0x08	0	0	0	0	1	0	0	0	A
0x03	0	0	0	0	0	0	1	1	B
0x46	0	1	0	0	0	1	1	0	C
0x21	0	0	1	0	0	0	0	1	D
0x06	0	0	0	0	0	1	1	0	E
0x0E	0	0	0	0	1	1	1	0	F



0 is on
1 is off

Figure 4 - Binary data sheet for the 7-segment display

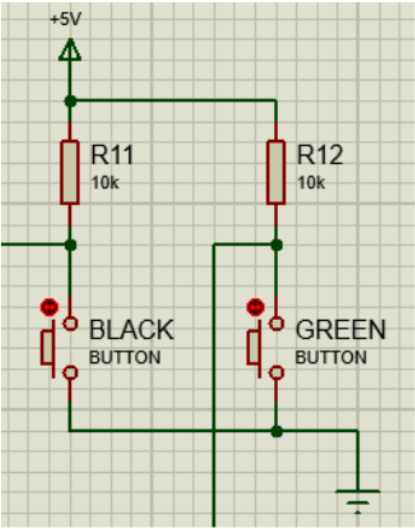


Figure 5 – Buttons

Both Resistors R11 and R12 are used as pull up resistors for this circuit. When the buttons are not pressed pin A0 and A1 are both High. However, because the pin connection is in between the resistor and the button, when a button is pressed the pin will go low as the current will be grounded. A0 will be low when the black button is pressed and A1 will be low when the green button is pressed. Because we know the pin can be high or low due to a button press, we can then

use the chip to assign a function to the button like turning on the sounder or resetting the seven-segment display.

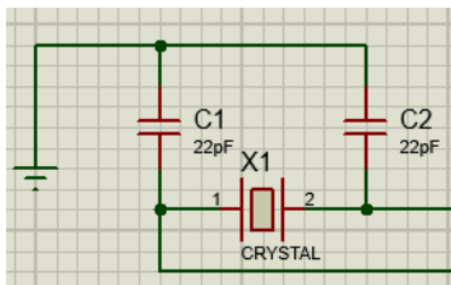


Figure 6 - Crystal Oscillator

1

Shown above in figure 5 (labelled X1) is a **crystal oscillator**. Although the programmable integrated chip has an internal oscillator, the accuracy is quite low. By using the crystal oscillator, I can have a stable output for an extended period. As well as this, the accuracy of the outputted oscillations of energy is increased [3]. The crystal oscillator gets an electrical stimulus as an input which it then the quartz crystal starts oscillating which outputs at a calibrated resonant frequency. This frequency is shown as a square wave which is then used as an input for the PIC. This square signal wave is used as a clock cycle. The more cycles you have the more information can be processed in a shorted space of time.

The two capacitors (C1 and C2) are put in place around the crystal oscillator **to smooth the signal**. What this means is, all the unwanted noise / signals are then not let into the PIC and the high accuracy of the oscillator can be maintained.

2

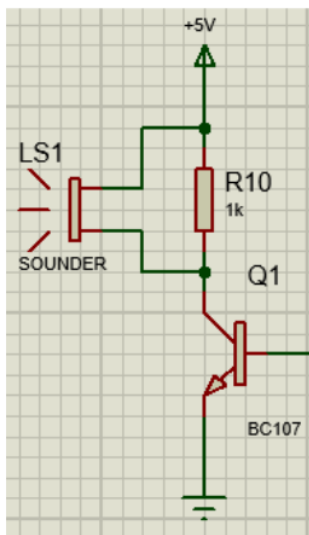


Figure 7 - Speaker

The transistor withing figure 7 and this circuit is used as a sort of on / off switch for the sounder. When there is enough current going to the transistor it allows the sounder to buzz and when there is not enough current the current gets grounded. The sounder is just used to make a sound withing

3

this circuit and the resistor R10 is simply used as a over current protector for the buzzer so it does not blow up.

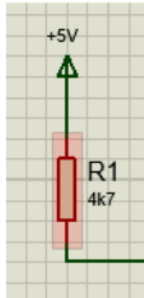


Figure 8 - MCLR resistor

R1 resistor is placed before the +5V for the purpose of a pull up resistor. The pull up resistor stops the chip from constantly resetting. Usually within a circuit there would be a button and a ground at this point as well so you can manually reset the chip when you need it. In this circuit there is no need to reset the chip as it has been set with all the parameters needed to effectively run its function.



Counselling

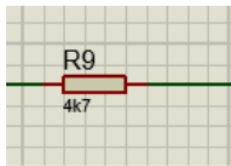


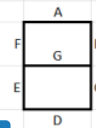
Figure 9 - Resistor protecting the Speaker

Resistor R9 is used as an over current protection resistor for the transistor. Without this resistor there would be no power to change the transistor on or off as it is a current activated component, and the transistor may blow because of an over current surge.

Section 2:

Hex	Buzz	G	F	E	D	C	B	A	Char
0x40	0	1	0	0	0	0	0	0	0
0x79	0	1	1	1	1	0	0	1	1
0x24	0	0	1	0	0	1	0	0	2
0x30	0	0	1	1	0	0	0	0	3
0x19	0	0	0	1	1	0	0	1	4
0x12	0	0	0	1	0	0	1	0	5
0x02	0	0	0	0	0	0	1	0	6
0x78	0	1	1	1	1	0	0	0	7
0x00	0	0	0	0	0	0	0	0	8
0x10	0	0	0	1	0	0	0	0	9
0x08	0	0	0	0	1	0	0	0	A
0x03	0	0	0	0	0	0	1	1	B
0x46	0	1	0	0	0	1	1	0	C
0x21	0	0	1	0	0	0	0	1	D
0x06	0	0	0	0	0	1	1	0	E
0x0E	0	0	0	0	1	1	1	0	F

6



0 is on
1 is off

Figure 10 - Binary data sheet for the 7-segment display

As shown once again these are my binary calculations for displaying hexadecimal on the seven-segment display. Each binary number is 8 characters long and then split into 2 bites for calculation. The binary is set from right to left doubling each time. Character one would be number 1, character 2 would be number 2, character 3 would be number 4 and character 4 would be number 8. If we take this into consideration above, I have the binary number 01000000. Split into bites that is 0100 and 0000. The one is positioned in the third character slot which equals 4 and the second binary number has no ones so is just 0. In total the hex format because of this is 40.

Before trying to set the LEDs of the seven-segment display to count from 0 to F in hex, I needed to assign each LED a binary number to show its hexadecimal character as shown in figure 10 in the hex column.

```

portB equ 0x0655      ;port B address
trisB equ 0x06        ;trisB address in register bank 1

Status equ 0x03       ;status register
RP0 equ 5              ;bit 5 of status register
RP1 equ 6              ;bit 6 of status register

nCount equ 0x0d        ; pause routine inner loop counter
mCount equ 0x0e        ; pause routine outer loop counter

LED_0 equ 0x40         ; 7 Segment LED Character definitions
LED_1 equ 0x79
LED_2 equ 0x24
LED_3 equ 0x30
LED_4 equ 0x19
LED_5 equ 0x12
LED_6 equ 0x02
LED_7 equ 0x78
LED_8 equ 0x00
LED_9 equ 0x10
LED_A equ 0x08
LED_B equ 0x03
LED_C equ 0x46
LED_D equ 0x21
LED_E equ 0x06
LED_F equ 0x0E

```

Figure 11 - LED binary numbers

Using port b gives me 8 pins for the seven-segment display to use. One as a power and the other 7 for the individual LEDs. As shown in figure 11 above, I have set each number to its binary value. This means when I run the LED_0 for example I can light up all of the LEDs within the seven- segment display to show the character 0. I have done this 15 times placing relevant binary value with each character.

I then start the main program. The first thing to do is to load the working register with the input of 0x00. This turns off all the LEDs ready for a new input. Port B (which is our 8 pins for the seven-segment display) is then cleared using the clrf function which will clear all the lines within Port B and set all the pins to low. As well as this the nCount (which is the variable the get_count loop is stored in) will be cleared to start at zero.

```

get_count

    movlw LED_0
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_1
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_2
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_3
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_4
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_5
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

    movlw LED_6
    movwf portB      ; move W to port B to update the display
    call one_sec_pause

```

Figure 12 - Get_count loop

The function above is written in the main start loop. The get_count function is initialised. First, we move the value of LED_0 stated previously in the code to the working register. The working register

now only contains the value of LED_0 in decimal converted to the binary form showing zero on the LEDs of the seven-segment display. Next, I move the working register containing this value to portB which is the port containing 8 pins used to display the value on the display. The value is then displayed on the seven-segment display. For the final part, I then add a one second pause before moving to the next value. This is repeated with all 15 characters from 0 to F in hex. Once it reaches the end the goto function shown below returns to the start of the get-count function making an infinite loop.

```
goto    get_count    ; repeat forever
```

Figure 13 - Get_count repeat

one_sec_pause

```
call    pause        ; delay 200ms by subroutine
call    pause
call    pause
call    pause
call    pause        ; five pause executions equals ~ 1 second
return
```

Figure 14 - one_sec_pause function

The function one_sec_pause is used as a 1 second delay for the count code above. Each pause variable called is worth 200ms. By using the call function and variable 5 times I am able to call a 1000 millisecond / 1 second pause. By using pauses I am able to see the changes made on the 7 segment-display. Without them the change would be too fast, and I would be unable to see the LEDs change.

```
pause
    movlw    0xff        ; set w = 255 decimal
    movwf    mCount      ; mCount = w: outer loop counter
```

Figure 15 - Pause function

By making a pause variable I am able to set a specific time in binary form that the LED switches between. By using the movlw function I can place the literal value of 255 in decimal straight into the working register. I then use the movwf function to move the working register into the file within the ram. This means that it can be run. The file that has been assigned in ram has been called mCount. When the function “pause” is called within the main code or the count code, a set time is used as a delay. This delay can be found in the f file.

```
loadN
    movlw    0xff        ; set w = 255 decimal
    movwf    nCount      ; nCount = w: inner loop counter
```

Figure 16 - loadN function

Like the pause variable function movlw is used placing 255 in decimal into the working register ready to be used. The function of movwf then moves this value into the ram file f and runs the inner count loop activating the working register to count in decimal.



```
decN
    decfsz    nCount, f      ; nCount--
    goto      decN          ; if nCount != 0 then repeat nCount--

    decfsz    mCount, f      ; else decrement Count
    goto      loadN          ; if mCount != 0 then
                                ; reload nCount to 255 and decrement
    return                                ; else exit subroutine
```

Figure 17 - Decrement N function

DECFSZ is used to decrement f or skip if the value of f is 0. This means that the loop of nCount will constantly run through until the program is stopped. When the n count does not equal 0 the loop continues creating the count infinite. mCount is not used in the count code.

Section 3:

Section 3.1:

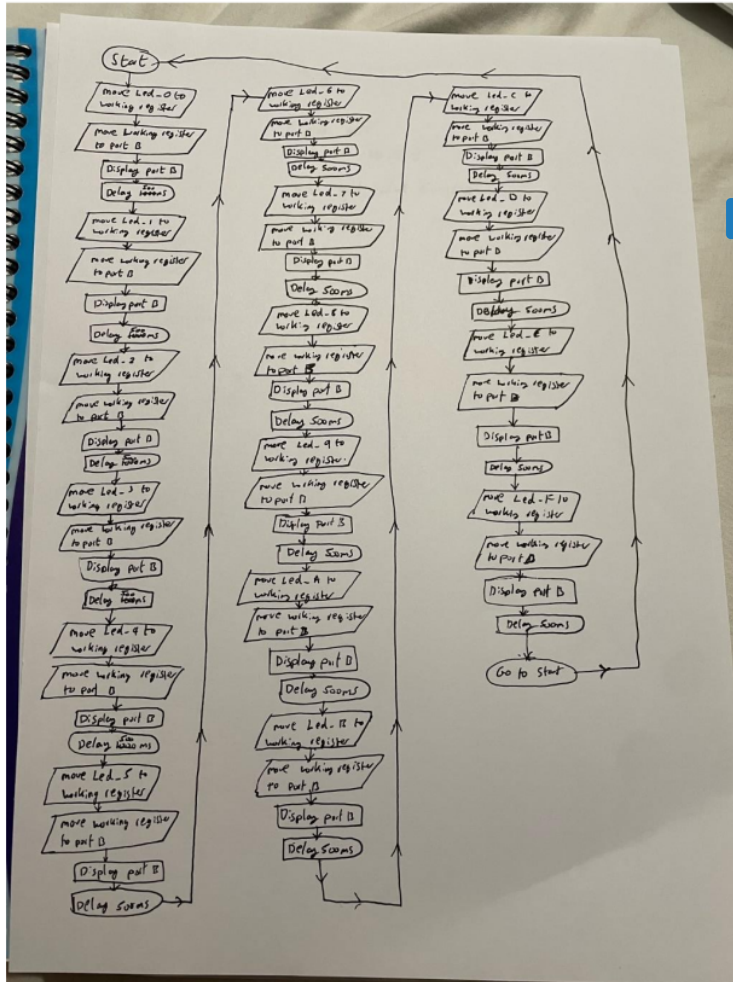


Figure 18 - Get count flow chart

The get_count loop is primarily made up of 4 functions. Movlw, Movwf, call and goto. Using the flow chart in figure 18 I am able to load and display all 15 characters in hex. The movlw function moves the literal value of the led into the working register. The movwf moves the value of the led to Port B so it can be displayed. I used the call function to call a delay of 500 ms so the LEDs can be seen and not change too fast.

Section 3.2:

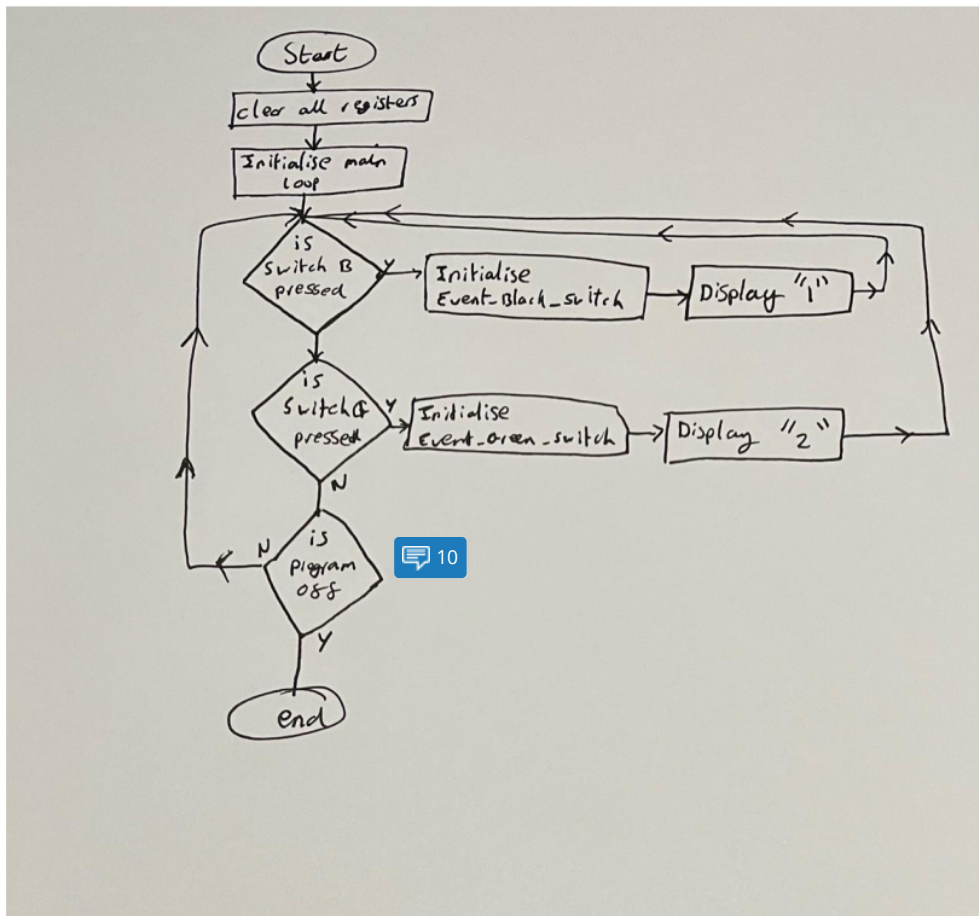


Figure 19 - Switch B and G flow chart

To start this flow chart, I need to clear all the registers so there is no data left in them before I start the program. The loop is initialised and checks for the black button to be pressed down. If the button is pressed the program continues and displays a 1 on the seven-segment display. If the button is not pressed, then the program moves on to check if the green button has been pressed. If the green button has been pressed the display will show 2 however if the button is not pressed the loop continues until one is pressed. The final stage is seeing if the program has finished. If it has then the loop ends and the event is finished.

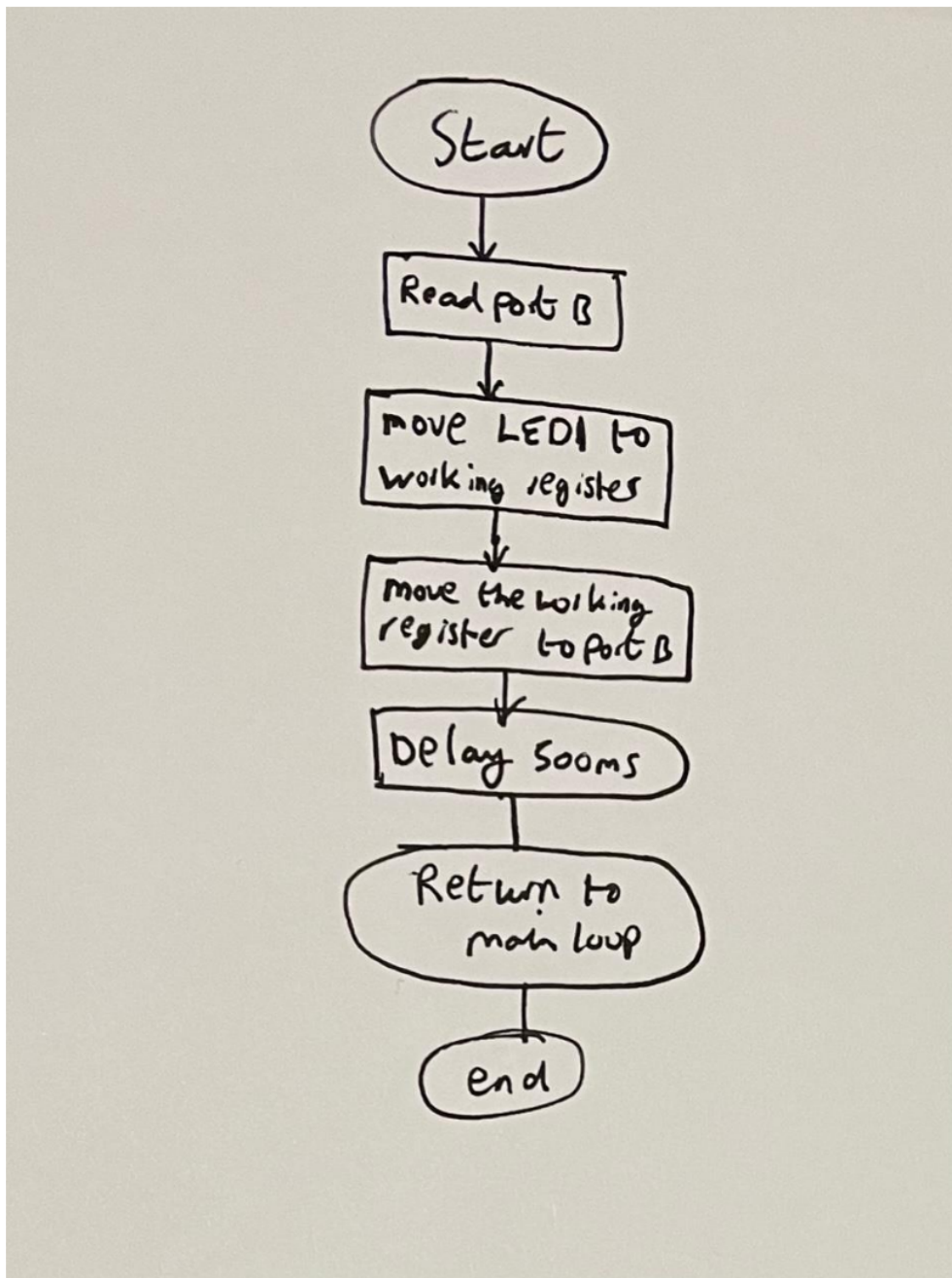


Figure 20 - Event flag black switch

Inside of the event flags lie the led value in port B. When the button is pressed in the main loop the event flag loop runs until there is a change being a second button is pressed or the program is finished.

Section 3.3:

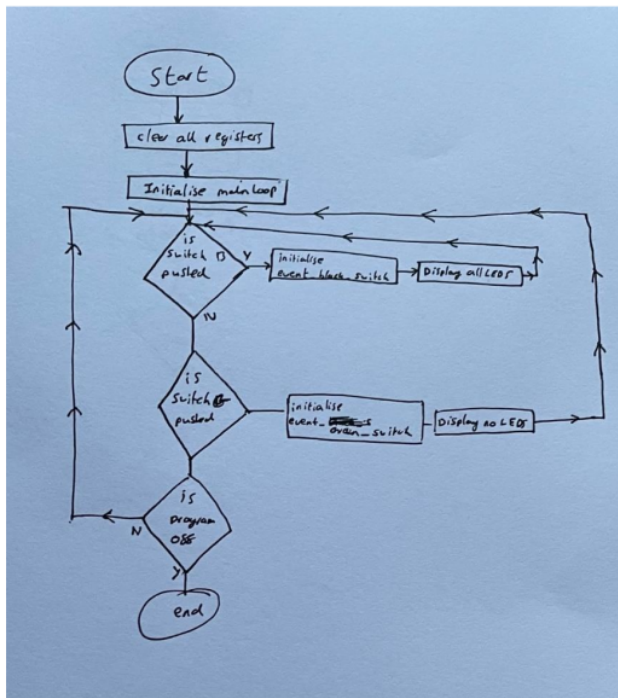


Figure 21 - Switch LEDs on/off

Codes for tasks 1-3.3:

Section 1 code:

```
; FILE:          count.asm
; AUTHOR:        Joshua J Port
; DATE:          25/11/2022
; DESCRIPTION:   Write a code to count from 0 to 255 in binary using the LEDs of a 7-segment display
;               Use 5 time 200 ms for a delay called pause.

        list      p=16F84          ; select processor type
        radix     hex              ; define radix as hexadecimal
;        #include "p16f84.inc"
;-----

;        cpu equates (memory map)
portB    equ 0x06                  ;port B address
trisb    equ 0x06                  ;trisB address in register bank 1

status   equ 0x03                  ;status register
RP0       equ 5                    ;bit 5 of status register
RP1       equ 6                    ;bit 6 of status register

count    equ 0x0c                  ; counter 0 to 255
nCount    equ 0x0d                  ; pause routine inner loop counter
mCountequ 0x0e                    ; pause routine outer loop counter

;-----

        org      0x000

start
```

```

    bcf    status,RP0    ; clear bank selection bits in the status reg
    bcf    status,RP1

    bsf    status,RP0    ; select bank 1
    movlw  0x00          ; load w with 0x00
    movwf  trisb         ; load it into trisb to make portb all outputs
    bcf    status,RP0    ; reselect bank 0

    clrf   portB         ; clear all lines low
    clrf   count        ; initialize counter to 0

get_count
    movf   count, w      ; move count to W

    movwf  portB         ; move W to port B to update the display

    addlw  1             ; add +1 to W

    movwf  count         ; save W in count

    call   pause         ; delay 200ms by subroutine
    call   pause
    call   pause
    call   pause
    call   pause         ; five pause executions equals ~ 1 second

    goto   get_count     ; repeat forever

;-----
;                               "mCount" is the outer loop counter (255 down to zero)

```

```
; "nCount" is the inner loop counter (255 down to zero)
```

```
;-----
```

```
pause
```

```
    movlw 0xff      ; set w = 255 decimal
```

```
    movwf mCount    ; mCount = w: outer loop counter
```

```
loadN
```

```
    movlw 0xff      ; set w = 255 decimal
```

```
    movwf nCount    ; nCount = w: inner loop counter
```

```
decN
```

```
    decfsz nCount, f ; nCount--
```

```
    goto   decN      ; if nCount != 0 then repeat nCount--
```

```
    decfsz mCount, f ; else decrement Count
```

```
    goto   loadN     ; if mCount != 0 then
```

```
                    ; reload nCount to 255 and decrement
```

```
    return           ; else exit subroutine
```

```
end
```

Section 2 code:

```
; FILE:          count.asm
; AUTHOR:        Joshua J Port
; DATE:          25/11/2022
; DESCRIPTION:    Program the seven-segment display to count from 0-F in hex.
;               Move up in 1 second intervals using pause

                list    p=16F84          ; select processor type
                radix    hex              ; define radix as hexadecimal
;               #include "p16f84.inc"
;-----
;               cpu equates (memory map)
portB    equ 0x06          ;port B address
trisb    equ 0x06          ;trisB address in register bank 1

Status    equ 0x03          ;status register
RP0       equ 5             ;bit 5 of status register
RP1       equ 6             ;bit 6 of status register

nCount    equ 0x0d          ; pause routine inner loop counter
mCount    equ 0x0e          ; pause routine outer loop counter

;               ; 7 Segment LED Character definitions
LED_0     equ 0x40          ; Set 0 character on 7-segment display
LED_1     equ 0x79          ; Set 1 character on 7-segment display
LED_2     equ 0x24          ; Set 2 character on 7-segment display
LED_3     equ 0x30          ; Set 3 character on 7-segment display
LED_4     equ 0x19          ; Set 4 character on 7-segment display
LED_5     equ 0x12          ; Set 5 character on 7-segment display
```

```

LED_6 equ 0x02      ; Set 6 character on 7-segment display
LED_7 equ 0x78      ; Set 7 character on 7-segment display
LED_8 equ 0x00      ; Set 8 character on 7-segment display
LED_9 equ 0x10      ; Set 9 character on 7-segment display
LED_A equ 0x08      ; Set A character on 7-segment display
LED_B equ 0x03      ; Set B character on 7-segment display
LED_C equ 0x46      ; Set C character on 7-segment display
LED_D equ 0x21      ; Set D character on 7-segment display
LED_E equ 0x06      ; Set E character on 7-segment display
LED_F equ 0x0E      ; Set F character on 7-segment display

```

```

;-----

```

```

    org    0x000

```

```

start

```

```

    bcf    Status,RP0    ; clear bank selection bits in the status reg
    bcf    Status,RP1

    bsf    Status,RP0    ; select bank 1
    movlw  0x00          ; load w with 0x00
    movwf  trisb         ; load it into trisb to make portb all outputs
    bcf    Status,RP0    ; reselect bank 0

    clrf   portB         ; clear all lines low
    clrf   nCount        ; initialize counter to 0

```

```

get_count

```

```

    movlw  LED_0         ; move l value LED_0 to working register
    movwf  portB         ; move W to port B to update the display

```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_1          ; move I value LED_1 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_2          ; move I value LED_2 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_3          ; move I value LED_3 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_4          ; move I value LED_4 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_5          ; move I value LED_5 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_6          ; move I value LED_6 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_7          ; move I value LED_7 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_8          ; move I value LED_8 to working register
```

```
movwf portB      ; move W to port B to update the display  
call    one_sec_pause ; pause for one second
```

```
movlw LED_9      ; move I value LED_9 to working register  
movwf portB      ; move W to port B to update the display  
call    one_sec_pause ; pause for one second
```

```
movlw LED_A      ; move I value LED_A to working register  
movwf portB      ; move W to port B to update the display  
call    one_sec_pause ; pause for one second
```

```
movlw LED_B      ; move I value LED_B to working register  
movwf portB      ; move W to port B to update the display  
call
```

```
goto    get_count    ; repeat forever
```

```
-----  
;  
;          "mCount" is the outer loop counter (255 down to zero)  
;  
;          "nCount" is the inner loop counter (255 down to zero)  
-----
```

```
one_sec_pause
```

```
call    pause        ; delay 200ms by subroutine  
call    pause        ; delay 200ms by subroutine  
call    pause        ; delay 200ms by subroutine  
call    pause        ; delay 200ms by subroutine  
call    pause        ; delay 200ms by subroutine  
return  ; return to the main code
```

```
; five pause executions equals ~ 1 second
```

```
-----
```

```
pause
```

```
movlw  0xff          ; set w = 255 decimal  
movwf  mCount        ; mCount = w: outer loop counter
```

```
loadN
```

```

        movlw 0xff          ; set w = 255 decimal
        movwf nCount       ; nCount = w: inner loop counter
decN
        decfsz nCount, f    ; nCount--
        goto   decN        ; if nCount != 0 then repeat nCount--

        decfsz mCount, f    ; else decrement Count
        goto   loadN       ; if mCount != 0 then
                           ; reload nCount to 255 and decrement

        return            ; else exit subroutine

end

```

```
; FILE:          count.asm
; AUTHOR:        Joshua J Port
; DATE:          25/11/2022
; DESCRIPTION:    Modify the previous code to display the hexadecimal figures 0-F on a seven-segment
display
;
;               Use pause to have a 0.5 second pause in between number change.
```

```

list      p=16F84                ; select processor type
radix     hex                    ; define radix as hexadecimal
;
#include "p16f84.inc"
;-----
;      cpu equates (memory map)

portB     equ 0x06                ;port B address

trisB     equ 0x06                ;trisB address in register bank 1

Status     equ 0x03                ;status register

RP0        equ 5                  ;bit 5 of status register

RP1        equ 6                  ;bit 6 of status register

nCount     equ 0x0d                ; pause routine inner loop counter

mCount     equ 0x0e                ; pause routine outer loop counter

;-----
;      7 Segment LED Character definitions

LED_0      equ 0x40                ; Set 0 character on 7-segment display
LED_1      equ 0x79                ; Set 1 character on 7-segment display
LED_2      equ 0x24                ; Set 2 character on 7-segment display
LED_3      equ 0x30                ; Set 3 character on 7-segment display
LED_4      equ 0x19                ; Set 4 character on 7-segment display
LED_5      equ 0x12                ; Set 5 character on 7-segment display

```

```

LED_6 equ 0x02      ; Set 6 character on 7-segment display
LED_7 equ 0x78      ; Set 7 character on 7-segment display
LED_8 equ 0x00      ; Set 8 character on 7-segment display
LED_9 equ 0x10      ; Set 9 character on 7-segment display
LED_A equ 0x08      ; Set A character on 7-segment display
LED_B equ 0x03      ; Set B character on 7-segment display
LED_C equ 0x46      ; Set C character on 7-segment display
LED_D equ 0x21      ; Set D character on 7-segment display
LED_E equ 0x06      ; Set E character on 7-segment display
LED_F equ 0x0E      ; Set F character on 7-segment display

```

```

;-----

```

```

    org    0x000

```

```

start

```

```

    bcf    Status,RP0    ; clear bank selection bits in the status reg
    bcf    Status,RP1

    bsf    Status,RP0    ; select bank 1
    movlw  0x00          ; load w with 0x00
    movwf  trisb         ; load it into trisb to make portb all outputs
    bcf    Status,RP0    ; reselect bank 0

    clrf   portB         ; clear all lines low
    clrf   nCount        ; initialize counter to 0

```

```

get_count

```

```

    movlw  LED_0         ; move l value LED_0 to working register
    movwf  portB         ; move W to port B to update the display

```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_1          ; move I value LED_1 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_2          ; move I value LED_2 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_3          ; move I value LED_3 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_4          ; move I value LED_4 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_5          ; move I value LED_5 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_6          ; move I value LED_6 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_7          ; move I value LED_7 to working register
```

```
movwf   portB          ; move W to port B to update the display
```

```
call    one_sec_pause ; pause for one second
```

```
movlw   LED_8          ; move I value LED_8 to working register
```

```
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_9      ; move I value LED_9 to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_A      ; move I value LED_A to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_B      ; move I value LED_B to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_C      ; move I value LED_C to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause
movwf nCount     ; save W in count
```

```
movlw LED_D      ; move I value LED_D to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_E      ; move I value LED_E to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
movlw LED_F      ; move I value LED_F to working register
movwf portB      ; move W to port B to update the display
call  one_sec_pause ; pause for one second
```

```
goto    get_count    ; repeat forever
```

```
;-----
```

```
one_sec_pause
```

```
call    pause        ; delay 200ms by subroutine
call    pause        ; delay 200ms by subroutine
call    pause        ; delay 200ms by subroutine
call    pause        ; delay 200ms by subroutine
call    pause        ; delay 200ms by subroutine
return   ; finish the sub-routine and return to main code
; five pause executions equals ~ 1 second
```

```
;-----
```

```
;                "mCount" is the outer loop counter (255 down to zero)
```

```
;                "nCount" is the inner loop counter (255 down to zero)
```

```
;-----
```

```
pause
```

```
movlw 0x7f ; set w = 255 decimal
movwf mCount ; mCount = w: outer loop counter
```

```
loadN
```

```
movlw 0xff ; set w = 255 decimal
movwf nCount ; nCount = w: inner loop counter
```

```
decN
```

```
decfsz nCount, f ; nCount--
goto    decN      ; if nCount != 0 then repeat nCount--
```


Section 3.2 code:

```
; FILE:          count.asm
; AUTHOR:        Joshua J Port
; DATE:         25/11/2022
; DESCRIPTION: Modify the previous codes so that when the black button is pressed, the hex
character 1 is displayed
;               on the seven-segment display and when the green button is pressed the hex character 2 is
displayed
;               on the display

list    p=16F84          ; select processor type
radix   hex              ; define radix as hexadecimal
;   #include "p16f84.inc"
;-----
;   cpu equates (memory map)
portA   equ    0x05       ; port A address
trisa   equ    0x05       ; tris A address in register bank 1

portB   equ    0x06       ; port B address
trisb   equ    0x06       ; tris B address in register bank 1

indf    equ    0x00       ; indirect file register
fsr     equ    0x04       ; indirect address register

Status  equ    0x03       ; status register
RP0     equ    5          ; bit 5 of status register
RP1     equ    6          ; bit 6 of status register

nCount  equ    0x0d       ; pause routine inner loop counter
mCount  equ    0x0e       ; pause routine outer loop counter
oCount  equ    0x0F
```

```
temp    equ    0x10        ; temp variable
```

```
LED1    equ    0x79        ; LED segment a bit of Port B
```

```
LED2    equ    0x24        ; LED segment b bit of Port B
```

```
SW_B    equ    0x01        ; Black switch bitmask for Port A
```

```
SW_G    equ    0x02        ; Green switch bitmask for Port A
```

```
;------
```

```
org      0x000
```

```
start
```

```
bcf      Status,RP0        ; clear bank selection bits in the status reg
```

```
bcf      Status,RP1
```

```
bsf      Status,RP0        ; select bank 1
```

```
movlw    0x03              ; Set switch pins as inputs
```

```
movwf    trisa
```

```
movlw    0x00              ; load w with 0x00
```

```
movwf    trisb              ; load it into trisb to make portb all outputs
```

```
bcf      Status,RP0        ; reselect bank 0
```

```
clrf     portB              ; clear all lines low
```

main_loop

```
    movf    portA,w          ; read Port A
    movwf   temp             ; move temp from working register to ram
    comf    temp,w           ; if temp is 0 complement w if temp is 1 move back to register
    andlw   SW_B             ; test black switch
    movwf   temp             ; move temp from w into ram again
    btfsc   temp,0           ; test bit 0 of temp (black switch) for a 1
    call    event_black_switch ; call sub-routine event_black_switch
```

```
    movf    portA,w          ; read Port A
    movwf   temp             ; move temp from working register to ram
    comf    temp,w           ; if temp is 0 complement w if temp is 1 move back to register
    andlw   SW_G             ; test green switch
    movwf   temp             ; move temp from w into ram again
    btfsc   temp,1           ; test bit 1 of temp
    call    event_green_switch ; call sub-routine event_green_switch
```

```
    goto    main_loop        ; go back to the main loop
```

event_black_switch

```
    movf    portB,w          ; read port B
    movlw   LED1             ; compliment port B bit 0
    movwf   portB            ; write value back to port B
    call    short_pause      ; call 0.5 second pause
    return                    ; else return to main code
```

event_green_switch

```
    movf    portB,w          ; read port B
```

```
    movlw LED2          ; compliment port B bit 0
    movwf portB         ; write value back to port B
    call  short_pause   ; call 0.5 second pause
    return              ; else return to main code
```

```
;-----
```

```
short_pause
```

```
    movlw 0x7F          ; set w = 127 decimal
    movwf oCount        ; nCount = w: inner loop counter
decN
    decfsz oCount, f     ; nCount--
    goto  decN          ; if nCount != 0 then repeat nCount--

    return              ; else exit subroutine
```

```
end
```

Section 3.3 code:

```
; FILE:          count.asm
; AUTHOR:        Joshua J Port
; DATE:          25/11/2022
; DESCRIPTION:   Change the code so the black button turns all of the LEDs on
;               and change the code so the green button turns all of the LEDs off
;               Use pause intervals so the code does not corrupt.
```

```
list    p=16F84          ; select processor type
radix    hex              ; define radix as hexadecimal
;    #include "p16f84.inc"
;-----
;    cpu equates (memory map)

portA    equ    0x05      ;port A address
trisa     equ    0x05      ;tris a address within register bank 1

portB    equ    0x06      ;port B address
trisb     equ    0x06      ;tris b address within register bank 1

indf     equ    0x00      ;indirect file register
fsr       equ    0x04      ;indirect address register

Status    equ    0x03      ;status register
RP0       equ    5         ;bit 5 of status register
RP1       equ    6         ;bit 6 of status register

nCount    equ    0x0d      ; pause routine inner loop counter
mCounteq  equ    0x0e      ; pause routine outer loop counter
oCount    equ    0x0F
```

```
temp    equ    0x10        ; temp variable
```

```
LED1    equ    0x00        ; LED segment a bit of Port B
```

```
LED2    equ    0x7F        ; LED segment b bit of Port B
```

```
SW_B    equ    0x01        ; Black switch bitmask for Port A
```

```
SW_G    equ    0x02        ; Green switch bitmask for Port A
```

```
;------
```

```
org      0x000
```

```
start
```

```
bcf      Status,RP0        ; clear bank selection bits in the status reg
```

```
bcf      Status,RP1
```

```
bsf      Status,RP0        ; select bank 1
```

```
movlw    0x03              ; Set switch pins as inputs
```

```
movwf    trisa             ; move trisa from working register to the ram
```

```
movlw    0x00              ; load w with 0x00
```

```
movwf    trisb             ; load it into trisb to make portb all outputs
```

```
bcf      Status,RP0        ; reselect bank 0
```

```
clrf     portB             ; clear all lines low
```

```
main_loop
```

```
movf  portA,w          ; read Port A
movwf temp             ; move temp variable from working register to ram
comf  temp,w           ; if temp is 0 store in ram if temp is 1 store back in w
andlw SW_B             ; test black switch
movwf temp             ; move temp to ram
btfsc temp,0           ; test bit 0 of temp (black switch) for a 1
call  event_black_switch ; call subroutine event_black_switch
```

```
movf  portA,w          ; read Port A
movwf temp             ; move temp variable from working register to ram
comf  temp,w           ; if temp is 0 store in ram if temp is 1 store back in w
andlw SW_G             ; test green switch
movwf temp             ; move temp to ram
btfsc temp,1           ; test bit 0 of temp
call  event_green_switch ; call subroutine event_green_switch
```

```
goto  main_loop
```

event_black_switch

```
movf  portB,w          ; read port B
movlw LED1             ; compliment port B bit 0
movwf portB            ; write value back to port B
call  short_pause      ; delay by 200 ms
return                 ; else exit the sub routine
```

event_green_switch

```
movf  portB,w          ; read port B
movlw LED2             ; compliment port B bit 0
```

```
movwf portB      ; write value back to port B
call  short_pause ; delay by 200 ms
return           ; else exit the sub routine
```

```
;-----
;               "mCount" is the outer loop counter (255 down to zero)
;               "nCount" is the inner loop counter (255 down to zero)
;-----
```

short_pause

```
movlw 0x29      ; set w = 41 decimal
movwf oCount    ; nCount = w: inner loop counter
```

decN

```
decfsz oCount, f ; nCount--
goto   decN      ; if nCount != 0 then repeat nCount--
```

```
return          ; else exit subroutine
```

end

References:

- [1] Microchip Technology Inc., "PIC16F84A Data Sheet," Microchip Technology Inc., 2001.
- [2] Techopedia, Peripheral Interface Controller (PIC), Techopedia, 2022.
- [3] CADENCE PCB SOLUTIONS, "The Role of Oscillators in Microcontrollers," CADENCE PCB SOLUTIONS, 2022.

Micros Peripherals and Interfaces assignment 1

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

63 / 100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

QM

Use of 1st Person Voice

When writing technical reports, do not use first person voice. Instead, use the third person passive voice.

You should not use words like "I", "you", "we" etc..

PAGE 6

PAGE 7



Comment 1

the component X1 is a crystal resonator, the circuit it is in is part of a crystal oscillator



Comment 2

They actually provide a load and allow the oscillator to oscillate at a stable frequency



Comment 3

No, not quite. it allows current to the piezo buzzer which will deform while it has power to it. To get it to make a noise, we need to turn it on and off quickly to make it expand and

contract quickly creating a sound wave.

PAGE 8



Comment 4

but it is in parallel so it can't control the current flow in the buzzer.

It is there to provide a DC current path to remove charge from the buzzer the because the buzzer is a capacitive device which will hold charge.



Comment 5

Good

PAGE 9



Comment 6

Good

PAGE 10

PAGE 11



Comment 7

"pause" is a subroutine not a variable.

PAGE 12



Comment 8

register?

PAGE 13



Comment 9

This would work, but there are more elegant ways of doing this.

PAGE 14



Comment 10

Good, although you don't need the test "is program off" and can just loop back to the start without the test,

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30



Comment 11

Good you've modified the pause to give half the time.

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

PAGE 38

PAGE 39

GRADING FORM: SMU GENERIC_SMOSEY

JOSHUA PORT

63

ENGAGEMENT WITH LITERATURE*including reading, referencing, academic conventions and academic honesty*

Excellent descriptions of all the components in the PIC demo board

15

KNOWLEDGE & UNDERSTANDING*Knowledge of the basic underlying concepts and principles of a subject.*

Excellent descriptions of how the circuits on the board function and how to control them

13

COGNITIVE & INTELLECTUAL*Evaluate underlying concepts and principles of a subject and interpret qualitative and quantitative data in order to develop lines of argument and make sound judgements.*

Use of the indirect register would have gained you more marks here, while your code is functional,

10

there are more elegant methods.

PRACTICAL APPLICATION SKILLS

Different approaches to solving problems in particular contexts.



Simulations should function.
minimum number of tasks completed.

11

TRANSFERABLE SKILLS

Communicate the results of their study/work accurately and reliably, and with structured and coherent arguments; the qualities needed for employment requiring the exercise of some personal responsibility within a structured and managed environment.



generally well structured report. Flowcharts could have been neater - I recommend using an application such as yEd to create them. Try not to use 1st person voice.

14