

RH_MPI's_Assignment

by Rhys Higgins

Submission date: 25-Nov-2022 12:18PM (UTC+0000)

Submission ID: 190078946

File name: 32751_Rhys_Higgins_RH_MPIs_Assignment_466098_506734020.docx (1.7M)

Word count: 6694

Character count: 33706

NOVEMBER 25, 2022

7-SEGMENT DISPLAY PROTOTYPE CIRCUIT

MICROS PERIPHERALS AND INTERFACES

RHYS HIGGINS (2110763)
UNIVERSITY OF WALES TRINITY ST DAVID
Word Count: 2429

Contents

Section 1: 2

Section 2: 5

Section 3 Code Flowcharts:..... 10

Section 3: Code 15

 Task 1: 15

 Task 2: 20

 Task 3: 25

 Task 4: 30

 Task 5: 36

Abbreviations: 40

References:..... 41

Section 1:

Shown in Figure 1.0 is the circuit schematic for the 7-Segment display circuit, which has been produced using the Proteus Design software package.

The circuit is based around a PIC16F84A, an 18 Pin PDIP package 8-bit PIC16-family RISC (Reduced Instruction Set Computer) microcontroller based on the Harvard Architecture, with Flash Memory (denoted by the 'F' following PIC16) [4]. As the PIC16F84A is based on the Harvard architecture, there are two entirely separate memory regions for both instructions and data, accessed through entirely separate buses to the CPU core [15] – illustrated by the block diagram in Figure 1.1. The main benefit of the Harvard architecture over the von Neumann architecture being that both instructions and data can be accessed by the CU (Control Unit) simultaneously in a single instruction cycle [13], as there is not the bottleneck of using a shared bus to access each. Due to being a RISC design, the PIC16F84A has only a small set of instructions, a total of 35 [6], which (barring a few) can be executed within a single clock cycle (or $1\mu\text{S}$) – making it easier to program as the instructions are less numerous, less-complex, and only perform a single function at a time [14].

The PIC16F84A has 13 pins of I/O (Input/Output), which can be assigned in either direction on an individual pin-by-pin basis [6]. As shown in Figure 1.0, a total of 10 of the 13 I/O pins are used, with 2 pins (RA0 and RA1) on Port A used as digital inputs, and 8 pins (RB0 to RB7) on Port B used as digital outputs.

Regarding the Port B outputs, pin RB0 to RB6 are sinking current, where a load is connected to an I/O pin and uses the pin as a path to ground [11] - through the internal (within the PIC package) N-Type FET transistor. In the case of the circuit, this is as these pins are connected to the 7-segment display LED control lines, via the resistors R2 to R8 – providing a path to ground. However, RB7 is sourcing current, where current is provided from the PIC to a load [11] from an internal P-type transistor - to the base of the NPN transistor Q1, via the resistor R9. For the PIC16F84A, it can source and sink a maximum of 25mA of current per pin [6].

Alongside the PIC16F84A, there are several other components found within the circuit.

On the upper right-hand-side of Figure 1.0, there are two buttons (acting as SPST switches), which are connected to Port A (RA0 and RA1) of the PIC, labelled as "BLACK" and "GREEN". Both RA0 and RA1 are held at +5V, when both switches are open (not pressed), being maintained at supply voltage through two $10\text{k}\Omega$ pull-up resistors, R11 and R12. Should the BLACK button be pressed, RA0 will be pulled down to 0V, as there is now a path from the +5V supply, through the resistor, to ground – the same going for the GREEN button and RA1. Also, R11 and R12 also prevent the +5V supply from being directly shorted to ground should either button be pressed, instead these resistors limit the current passing from the +5V rail to 0V. The calculation used for this can be seen in Figure 1.3, typical values for LED I (current) would be $\sim 0.020\text{A}$ [20] and the typical voltage drop for a RED LED being $\sim 2.0\text{V}$ [21].

The upper left-hand-side of Figure 1.0 consists of four passive components, two 22pF non-polarised capacitors, one crystal oscillator, and a $4\text{k}\Omega$ resistor. The piezo-electric (where the oscillator changes shape when a voltage is applied) crystal oscillator is used to provide an accurate reference frequency for the PIC, connected across the **CLKIN** and **CLKOUT** pins of the PIC. This type of oscillator is used due to its high frequency stability. [7] The $4\text{k}\Omega$ (R1) resistor is used to pull **NOT MCLR** (Not Master Clear) to a logic HIGH (+5V), which is used to prevent the microcontroller from resetting to the reset

vector address (0x0000) [8]. Finally, the two 22pF capacitors connected to the pins of the crystal ground would usually be matched to the Load Capacitance of the crystal to maintain the accuracy of its oscillations [9], in this case they have not been - if this is desired, the equation shown in Figure 1.2 could be used; where C_{stray} is the stray capacitance in the circuit, usually 2-5pF, and C_1 and C_2 are the two load capacitors [9].

The lower left-hand-side of Figure 1.0 consists of another two passive components alongside two active components: two resistors (one 1K Ω and another 4k7 Ω), a buzzer, and a BC107 NPN BJT transistor (Q1). The 4k7 Ω resistor (R9) is connected between Pin_B7 (13) of the PIC and the base of the BC107 transistor (Q1). The purpose of this is to enable Q1 to be turned on and therefore controlled by the PIC through Pin_B7, the resistor R9 on Pin_B7 also serves as a current-limiting device to the base of the BJT – to prevent it from being damaged through excessive current draw, but also allowing the base current to be sufficiently high to enable the transistor to be saturated fully whilst in its ON state [10]. Therefore, when Q1 has been turned on, it allows current to flow across the transistor (collector to the emitter) connecting the buzzer to ground (by allowing current flow) and activating it. Another useful feature of resistor R9 is it prevents the PIC from sourcing too much current from the internal P-type transistor in the PIC, which would otherwise damage the PIC.

Finally, the lower right-hand-side of Figure consists of a common-anode 7-segment display, along with seven 220 Ω (R2 to R8) resistors, each connected from the Port B (RB0 to RB6) outputs of the PIC to the 7-segment display inputs. Again, like before, the purpose of the seven 220 Ω series-connected resistors is to limit the current to the LEDs of the 7-segment display, to prevent the LEDs from burning out and failing prematurely [12], whilst allowing enough current to pass for the LEDs to illuminate at peak intensity.

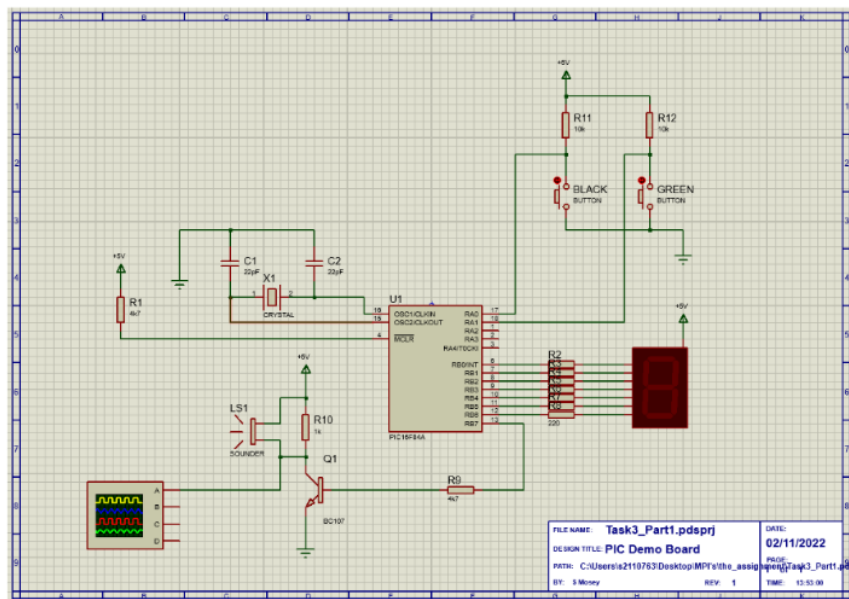


Figure 1.0: 7-Segment Board in Proteus Schematic Window

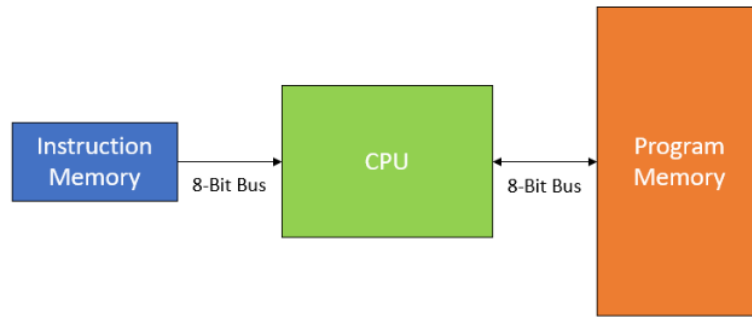


Figure 1.1: PIC16F84A Harvard Memory Layout

$$CL = \left(\frac{C1 * C2}{C1 + C2} \right) + Cstray$$

Figure 1.2: Equation for the calculation of Load Capacitance

$$\frac{V_{cc} - V_{Led}}{I}$$

Figure 1.3: Equation for the calculation of resistor value

Section 2:

An important feature of the simulated prototype circuit is the common-anode 7-segment display. This display comprises of 7 separate LEDs, labelled 'A' through to 'G' – shown in Figure 2.0. To illuminate each segment of the display, the appropriate lines of the microcontroller **Port B** outputs (Pins B0 – Pin B6) need to be pulled to logic LOW. The lines need to be pulled LOW as a common-anode display has been used for the circuit, where a logic HIGH (+5V) to the display input illuminates a segment ('A' to 'G') and a logic LOW (+0V) does the reverse.

To determine which segments of the display are illuminated, for characters A to G, a literal value needs to be calculated in binary, to determine the outputs of Port B. The literal value for the characters is calculated by converting the first binary nibble and second binary nibble into a hexadecimal number (0 to F). An example of this conversion is the first character 0; where the first nibble is **0100**, corresponding to the last three LED segments (G to E), meaning only segment G is pulled LOW, and the second binary nibble **0000** corresponding to the first four LED segments (A to D) are all HIGH. When converted to hexadecimal, this gives the value 0x40. Both calculations can be seen in the table Figure 2.1.

To output the characters 0 through to F in sequence, the LED hexadecimal values need to be defined and stored as a literal value in code, which, as discussed before, reflect the required outputs of each pin on **Port B** (of the PIC MCU). This can be achieved through providing an easily recognisable name for each literal value, i.e., LED_0 for 0, LED_1 for 1 and so on (until char F), which are then stored in the registers of the PIC. A start address (to mark the start of the character data), indexing variable (to provide the number of characters to be loaded), and intermediate variable (for calculating each character address) should all be defined in the memory map of the PIC. For the start address, this should be the register address of where the first character (LED_0 – 0x40) is stored – in the case of my code I used register 0x10, with all characters at an offset of +1 for each character from this address. Doing this at addresses offset by 1 allows for each file to be iterated through one at a time. The indexing variable is defined as equalling 0xF, to represent the 16 characters needed and the intermediate variable defined as equalling 0xC. The definitions for each the LED values can be seen in Figure 2.2.

Once the literal hexadecimal values have been declared in code (assembly), **Port B** needs to be configured as an output, this is done at the very start of the code and executed only once. To start, the **BCF** (Bit Clear File) instruction is used to clear the bank selection bits in the status register (at address 0x03), specifically Bits 5 (RP0) and 6 (RP1). The **BSF** (Bit Set File) instruction is then used on the status register (where Bit 5 is now '1') to select Bank 1, where the **TRIS** registers are located [16]. Next, the **MOVLW** instruction (Move Literal to Working) is used to move the value 0x00 to the W (working) register. This value is then moved to the file trisb (Port Direction B), using the **MOVWF** (Move Working to File) instruction. The implication of this is that all I/O (input/output) on Port B are configured as outputs – due to using the value 0x00, or 00000000 in binary, where 0 on a PIC is an output for the corresponding pin and 1 is an input [19]. To explain this further, if the value 0x03 (binary 00000011) was loaded into trisb, Bits 0 and 1 (Pins RB0 and RB1) would be configured as inputs and Bits 2 to 7 (RB2 to RB7) would be configured as outputs – the LSB (Least Significant Bit) of the binary value being RB0 and the MSB (Most Significant Bit) being RB7 [16]. After this, the **BCF** instruction needs to be used, to clear Bit 5 of the status register (where Bit 5 is now '0') to reselect Bank 0 and enable the use of **Port B**. The above configuration can be seen in Figure 2.3. Finally, the

CLRF (Clear F) [1] is used to clear the contents of the **Port B** register, the effect being all lines on **Port B** (RB0 to RB7) are cleared to a logic LOW, illuminating all segments on the LCD [17].

Now each character should then be loaded into the **W** register using the **MOVLW** instruction. Following this, the **MOVWF** (Move Working to File) instruction should be used, moving the character from the **W** register to the start address of the character data – stored in RAM. These instructions should then be repeated sequentially for each individual character, as shown in Figure 2.4.

After all characters have been loaded a literal value should be passed to the **W** register, using the **MOVLW** instruction again, this value being used to define the maximum count value (0x11). The **MOVWF** instruction then being used to pass this value into the indexing variable, shown in Figure 2.5. This is essential to the function of this code; without it the code would continue incrementing through memory that is out of bounds.

Essentially, a never-ending **for loop** is then used, where the **DECFSZ** instruction (Decrement register 'F' – Skip if 0 [1][2]) is used. The purpose of the **DECFSZ** instruction here is to decrement the indexing variable by 1 at each iteration prior to the code entering a **do loop** each time, where the characters are outputted to the display, skipping the **do loop GOTO** instruction (acting as a **NOP** – No Operation instruction, skipping the next instruction [2] and wasting CPU cycles) when the indexing variable reaches 0. This section of code can be seen in Figure 2.6. Once the indexing variable reaches 0, the **do loop** is skipped, due to the **DECFSZ** instruction, and an **GOTO** instruction is executed – the program then returns to the section above where the maximum count value is defined and stored into the indexing variable register.

Regarding the forementioned **do loop**, the indexing variable should be moved into the working register, using the **MOVWF** (Move File) instruction. The **SUBLW** (Subtract Literal from Working) instruction is then used to subtract 0x16 from the indexing variable, which is then stored to the intermediate variable using the **MOVWF** instruction. After this, the **ADDLW** (Add Literal to Working) instruction is used to add the start address to the intermediate variable, which is then moved to the **FSR** (File Select Register), which acts as a pointer to **INDF** (File Register 00) register [3], via the **MOVWF** instruction. The characters are then pulled from the **INDF** register into the **W** register, using the **MOVWF** instruction, where it is then copied to **Port B** using the **MOVWF** instruction. Finally, a short pause can then be called, i.e., “**CALL pause_func**” before a **GOTO** is used to return to the start of the **for loop** – repeating the execution of the code. This can be seen in Figure 2.7. Regarding **pause_func**, this pause should waste a long-enough so that the characters are clearly legible on the display. For this, an interval of ~1000ms would be suitable. This can be done through decrementing, using the **DECFSZ** instruction, an inner and outer loop counter for a total of 1000ms (5x200ms), and then using the **call** instruction on **pause_func** five times for a total of 1000ms. The way this works is the **MOVLW** and **MOVWF** instructions have been used to first declare the size of the variable and then store it. Then, the **DECFSZ** and **GOTO** instructions are used to create a loop, where the **DECFSZ** instructions takes 1μS to execute and the **GOTO** takes 2μS (1 CPU cycle takes 1μS) [1][18] – the inner loop is then executed 256 times to every 1 execution of the outer loop. Once the outer loop reaches zero, the **DECFSZ** instruction skips the next instruction [2] and the return instruction is now able to be executed, exiting the subroutine. As the outer loop is also executed 256 times, this creates the 200ms (approximate) delay required. This loop can be seen in Figure 2.8 and the calculations for the pause duration in Figure 2.9. Finally, once the subroutine has been excited, the program returns to the **for loop** and execution from this point is repeated.

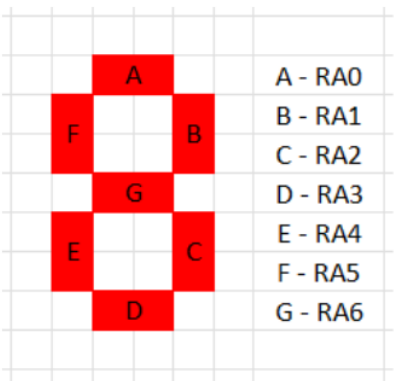


Figure 2.0: 7-segment LCD Segment Listing

	G	F	E	D	C	B	A	Hex	Char
0	1	0	0	0	0	0	0	0x40	0
1	1	1	1	1	0	0	1	0x79	1
2	0	1	0	0	1	0	0	0x24	2
3	0	0	1	1	0	0	0	0x30	3
4	0	0	0	1	1	0	1	0x19	4
5	0	0	0	1	0	0	1	0x12	5
6	0	0	0	0	0	1	0	0x02	6
7	0	1	1	1	1	0	0	0x78	7
8	0	0	0	0	0	0	0	0x00	8
9	0	0	0	1	0	0	0	0x10	9
10	0	0	0	0	1	0	0	0x08	10
11	0	0	0	0	0	1	1	0x03	11
12	0	1	0	0	0	1	1	0x46	12
13	0	0	1	0	0	0	1	0x21	13
14	0	0	0	0	0	1	1	0x06	14
15	0	0	0	0	1	1	1	0x0E	15
First Nibble					Second Nibble				

Figure 2.1 7-segment display truth table

```
LED_0 equ 0x40 ; 7 segment LED char 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 2.2 7-segment LED definitions

```

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

```

Figure 2.3 status register, trisb, and portB

```

movlw  LED_0      ; Load the char "0" into the W reg
movwf  0x10       ; move the char "0" from W to start addr of char data RAM
movlw  LED_1      ; Load the char "1" into the W reg
movwf  0x11
movlw  LED_2      ; Load the char "2" into the W reg
movwf  0x12
movlw  LED_3      ; Load the char "3" into the W reg
movwf  0x13
movlw  LED_4      ; Load the char "4" into the W reg
movwf  0x14
movlw  LED_5      ; Load the char "5" into the W reg
movwf  0x15
movlw  LED_6      ; Load the char "6" into the W reg
movwf  0x16
movlw  LED_7      ; Load the char "7" into the W reg

```

Figure 2.4 loading chars into W register.

```

get_count

    movlw  0x11      ; Load 0x11 to W reg - maximum count value
    movwf  i         ; set index variable to max count value

```

Figure 2.5 setting index value.

```

loop_start                                ; for loop start
    decfsz i,f
    goto  do_loop
    goto  get_count                      ; repeat forever

```

Figure 2.6 for loop

```

do_loop

    ; j = 15 - i
    movf  i,w          ; move file 'i' into working reg
    sublw 0x10         ; 16 - i
    movwf j            ; store value into j
    ; FSR = startaddr + j
    addlw startaddr    ; add j to start addr
    movwf fsr          ; move start addr to fsr
    ; PortB = INDF
    movf  indf,w       ; pull data from RAM and store in W reg
    movwf portB        ; copy data to portB
    ; pause
    call  half_sec_pause
    goto  loop_start

```

Figure 2.7. do loop

```

;-----
pause
    movlw 0xff      ; set w = 256 decimal          1
    movwf mCount    ; mCount = w: outer loop counter 1
loadN
    movlw 0xff      ; set w = 256 decimal          1
    movwf nCount    ; nCount = w: inner loop counter 1
decN
    decfsz nCount, f ; nCount--                    1
    goto decN        ; if nCount != 0 then repeat nCount-- 2

    decfsz mCount, f ; else decrement Count        1
    goto loadN       ; if mCount != 0 then          2
                        ; reload nCount to 255 and decrement
return              ; else exit subroutine

```

Figure 2.8. pause function

$$((1\mu s + 2\mu s) * 256) * 256 = 199,680\mu s / \sim 200ms$$

Figure 2.9 Pause function working

Section 3 Code Flowcharts:

Task 1.

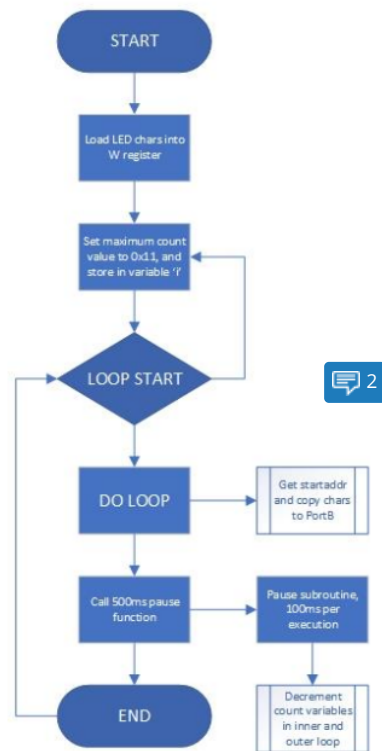


Figure 2.1 – Task 1. Code Flowchart

Task 2.

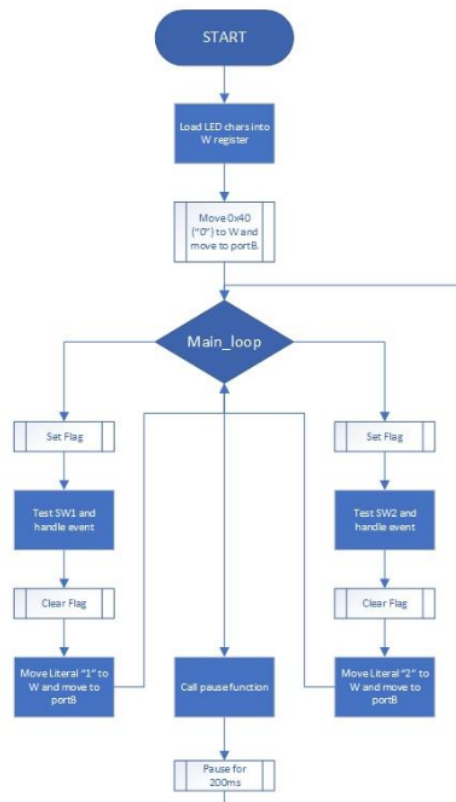


Figure 2.2 – Task 2. Code Flowchart

Task 3.

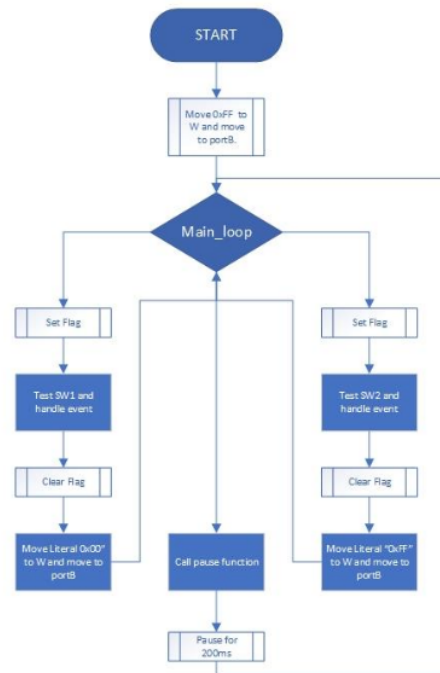


Figure 2.3 – Task 3. Code Flowchart

Task 4.

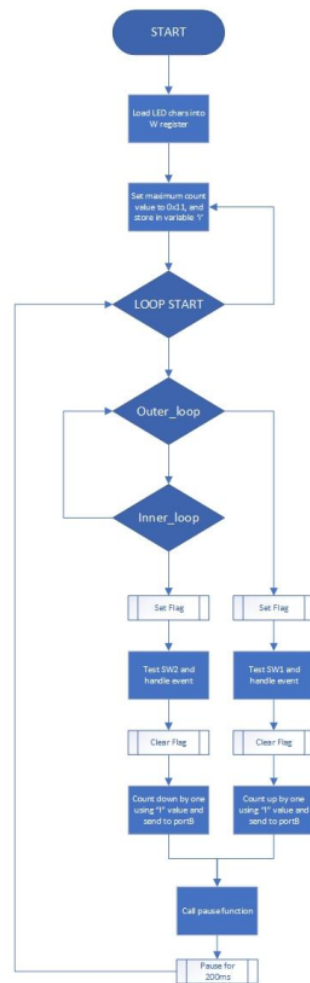


Figure 2.4 – Task 4. Code Flowchart

Task 5.

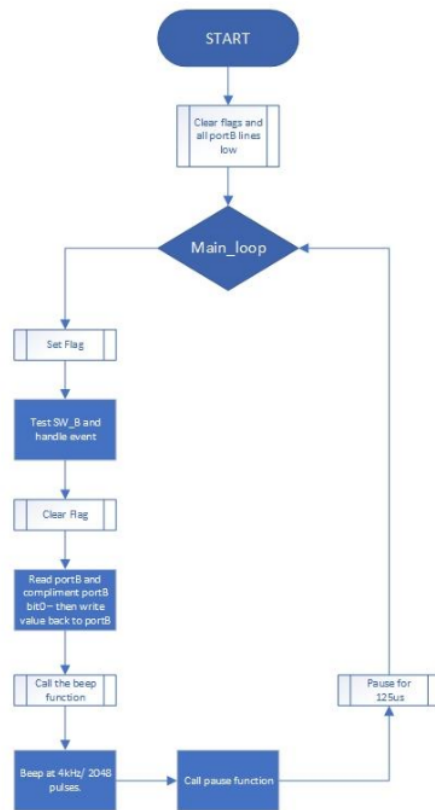


Figure 2.5 – Task 5. Code Flowchart

Section 3: Code

Task 1:

```
; FILE:      count.asm
; AUTHOR:    RPH
; DATE:      25th Nov 2022
; DESCRIPTION:      Count to 255 at 1 second intervals and display on a seven
;                  segment display in binary.
;                  The pause subroutine takes apx. 0.2 sec.

        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

indf     equ     0x00      ; indir file reg
fsr      equ     0x04      ; indir addr reg

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

startaddr equ     0x10      ; start addr of char data
i        equ     0x0F      ; indexing variable
j        equ     0x0C      ; intermediate variable for calculating addresses

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

LED_0    equ     0x40      ; 7 segment LED char 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
```

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

```
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

    movlw  LED_0          ; load the char "0" into the W reg
    movwf  0x10           ; move the char "0" from W to start addr of char data RAM
    movlw  LED_1          ; load the char "1" into the W reg
    movwf  0x11
    movlw  LED_2          ; load the char "2" into the W reg
    movwf  0x12
    movlw  LED_3          ; load the char "3" into the W reg
    movwf  0x13
    movlw  LED_4          ; load the char "4" into the W reg
    movwf  0x14
```

```
    movlw LED_5      ; load the char "5" into the W reg
    movwf 0x15
    movlw LED_6      ; load the char "6" into the W reg
    movwf 0x16
    movlw LED_7      ; load the char "7" into the W reg
    movwf 0x17
    movlw LED_8      ; load the char "8" into the W reg
    movwf 0x18
    movlw LED_9      ; load the char "9" into the W reg
    movwf 0x19
    movlw LED_A      ; load the char "A" into the W reg
    movwf 0x1A
    movlw LED_B      ; load the char "B" into the W reg
    movwf 0x1B
    movlw LED_C      ; load the char "C" into the W reg
    movwf 0x1C
    movlw LED_D      ; load the char "D" into the W reg
    movwf 0x1D
    movlw LED_E      ; load the char "E" into the W reg
    movwf 0x1E
    movlw LED_F      ; load the char "F" into the W reg
    movwf 0x1F
```

get_count

```
    movlw 0x11      ; load 0x11 to W reg - maximum count value
    movwf i          ; set index variable to max count value
```

loop_start ; for loop start

```
    decfsz i,f
    goto do_loop
    goto get_count ; repeat forever
```

do_loop

```
    ; j = 15 - i
    movf i,w          ; move file 'i' into working reg
    sublw 0x10        ; 16 - i
```

```
        movwf j            ; store value into j
; FSR = startaddr + j
        addlw startaddr    ; add j to start addr
        movwf fsr          ; move start addr to fsr
; PortB = INDF
        movf indf,w        ; pull data from RAM and store in W reg
        movwf portB        ; copy data to portB
; pause
        call half_sec_pause

        goto loop_start

half_sec_pause

        call pause         ; delay 100ms by subroutine
        call pause
        call pause
        call pause
        call pause         ; five pause executions equals ~ half a second

;-----
;subroutine pause: waste one 500ms
;inputs: nothing
;outputs: nothing
;
;does: calls pause 5x to waste 500ms (5 x 100ms)
;corrupts: register W
;
;description: two loops using "mCount" and "nCount" register files
;
;           "mCount" is the outer loop counter (255 down to zero)
;
;           "nCount" is the inner loop counter (128 down to zero)
;
;
;           this has been modified, so that the outer loop counter
;
;           is now '0xc0', rather than '0xff', to change the count
;
;           period from 200ms to 100ms.
;-----
pause
        movlw 0xff         ; set w = 255 decimal                1
```

```
        movwf  mCount      ; mCount = w: outer loop counter      1
loadN
        movlw  0x80        ; set w = 128 decimal                1
        movwf  nCount      ; nCount = w: inner loop counter      1
decN
        decfsz nCount, f    ; nCount--                            1
        goto   decN        ; if nCount != 0 then repeat nCount--  2

        decfsz mCount, f    ; else decrement Count                1
        goto   loadN       ; if mCount != 0 then                  2
                                ; reload nCount to 255 and decrement
        return            ; else exit subroutine

        end

;-----
; Configuration! Important!
;
; at blast time, select:
;     memory unprotected
;     watchdog timer disabled
;     standard crystal (4 MHz)
;     power-up timer on
```

Task 2:

```
; FILE:      count.asm
; AUTHOR:    RPH
; DATE:      25th Nov 2022
; DESCRIPTION:  TBC

        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

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

indf     equ     0x00      ; indir file reg
fsr      equ     0x04      ; indir addr reg

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

event_flag    equ     0x11      ; event_handler start flag

temp    equ     0x1      ; temp variable

SW1     equ     0x01      ; SW1 bitmask of portA
SW2     equ     0x02      ; SW2 bitmask of portA

; 7 segment LED char definitions
LED_0   equ     0x40      ; '0'
LED_1   equ     0x79      ; '1'
```

```
LED_2 equ    0x24    ; '2'
```

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

```
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
```

```
                                ; Initialise 7 segment to 0
movlw  LED_0           ; move literal to working, LED_0 val into working
movwf  portB           ; move working to portB
```

```
main_loop
```

```
    ;SW1
```

```
    movf   portA,w
    movwf  temp        ; move working to file
    comf   temp,w      ;
    andlw  SW1          ; test SW1
    andwf  event_flag,w ; test event_flag
    movwf  temp        ; stores w reg value (SW1 status) in temp
    btfsc  temp,0       ; test bit 0 of temp (black switch), if button not pressed
```

```
skip
```

```
    call   event_SW1
```

```
    movf   portA,w      ; read portA
    movwf  temp        ;
    comf   temp,w      ;
    andlw  SW1          ; test black switch
    movwf  temp        ; move working to temp
```

```
        btfss temp,0      ; test bit 0 of temp
        bsf    event_flag,0 ; set flag to allow event_sw1
;SW2
        movf   portA,w    ;
        movwf  temp       ;
        comf   temp,w     ;
        andlw  SW2        ; test SW1
        andwf  event_flag,w ; test event_flag
        movwf  temp       ; stores w reg value (SW2 status) in temp
        btfsc  temp,1     ; test bit 1 of temp (black switch), if button not pressed
skip
        call   event_SW2

        movf   portA,w    ; read portA
        movwf  temp       ; move working to file
        comf   temp,w     ;
        andlw  SW2        ; test black switch
        movwf  temp       ; move working to file
        btfss  temp,1     ; test bit 0 of temp
        bsf    event_flag,1 ; set flag to allow event_sw2

        goto   main_loop  ; return to start of the loop

; pause
        call   short_pause

;-----
event_SW1
    ; check portB to see if LED is ON or OFF
    bcf    event_flag,0 ; clear flag to show event complete
    movf   portB,w     ; read portB
    call   LED1_ON      ; short beep
    return

;-----
event_SW2
    ; check portB to see if LED is ON or OFF
    bcf    event_flag,0 ; clear flag to show event complete
    movf   portB,w     ; read portB
```

[illegible]

```
        return                ; else exit subroutine

    end

;-----
; Configuration! Important!
;
; at blast time, select:
;     memory unprotected
;     watchdog timer disabled
;     standard crystal (4 MHz)
;     power-up timer on
```

Task 3:

```
; FILE:      count.asm
; AUTHOR:    RPH
; DATE:      25th Nov 2022
; DESCRIPTION:  TBC

        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      ; trisa address in register bank 1

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

indf     equ    0x00      ; indir file reg
fsr      equ    0x04      ; indir addr reg

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

event_flag    equ    0x11      ; event_handler start flag

temp    equ    0x1      ; temp variable

SW1     equ    0x01      ; SW1 bitmask of portA
SW2     equ    0x02      ; SW2 bitmask of portA

; 7 segment LED char definitions
LED_1   equ    0x00
```

```
LED_2    equ    0xFF
```

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

```
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
```

```
          ; Initialise 7 segment to 0
```

```
movlw    LED_2          ; move literal to working, LED_2 val into working
movwf    portB          ; move working to portB
```

```
main_loop
```

```
    ;SW1
```

```
    movf   portA,w
    movwf  temp        ; move working to file
    comf   temp,w      ;
    andlw  SW1          ; test SW1
    andwf  event_flag,w ; test event_flag
    movwf  temp        ; stores w reg value (SW1 status) in temp
    btfsc  temp,0       ; test bit 0 of temp (black switch), if button not pressed
```

```
skip
```

```
    call   event_SW1
```

```
    movf   portA,w      ; read portA
    movwf  temp        ;
    comf   temp,w      ;
    andlw  SW1          ; test black switch
```

```
        movwf temp        ; move working to temp
        btfss temp,0      ; test bit 0 of temp
        bsf    event_flag,0 ; set flag to allow event_sw1
;SW2
        movf   portA,w    ;
        movwf  temp       ;
        comf   temp,w     ;
        andlw  SW2        ; test SW1
        andwf  event_flag,w ; test event_flag
        movwf  temp       ; stores w reg value (SW2 status) in temp
        btfsc  temp,1     ; test bit 1 of temp (black switch), if button not pressed
skip
        call   event_SW2

        movf   portA,w    ; read portA
        movwf  temp       ; move working to file
        comf   temp,w     ;
        andlw  SW2        ; test black switch
        movwf  temp       ; move working to file
        btfss  temp,1     ; test bit 0 of temp
        bsf    event_flag,1 ; set flag to allow event_sw2

        goto   main_loop  ; return to start of the loop

; pause
        call   short_pause

;-----
event_SW1
    ; check portB and
        bcf    event_flag,0 ; clear flag to show event complete
        movf   portB,w     ; read portB
        call   LED_ON      ; call LED ON func
        return

;-----
event_SW2
    ; check portB and
        bcf    event_flag,0 ; clear flag to show event complete
```

```
    movf    portB,w        ; read portB
    call    LED_OFF        ; call LED OFF func
    return

;-----
LED_ON                ; illuminates all 7 segments
    movlw   LED_1
    movwf   portB
    return

;-----
LED_OFF                ; clears all 7 segments
    movlw   LED_2
    movwf   portB
    return

;-----
short_pause
    call    pause         ; delay 200ms by subroutine

;-----
;subroutine pause:
;actions: calls pause to waste 200ms
;corrupts: register W
;
;description: two loops using "mCount" and "nCount" register files
;
;              "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

;-----
; Configuration! Important!
;
; at blast time, select:
;     memory unprotected
;     watchdog timer disabled
;     standard crystal (4 MHz)
;     power-up timer on
```

Task 4:

```
; FILE:      count.asm
; AUTHOR:    RPH
; DATE:      25th November 2022
; DESCRIPTION:  TBC

        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      ; trisa address in register bank 1

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

indf     equ     0x00      ; indir file reg
fsr      equ     0x04      ; indir addr reg

status   equ     0x03      ; status register
RP0      equ     5         ; bit 5 of status register
RP1      equ     6         ; bit 6 of status register
z        equ     0x13

startaddr    equ     0x11      ; start addr of char data
i            equ     0x0F      ; indexing variable
j            equ     0x0C      ; intermediate variable for calculating addresses

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

event_flag    equ     0x10      ; event_handler start flag

temp        equ     0x1      ; temp variable
```

```
SW1    equ    0x01    ; SW1 bitmask of portA
SW2    equ    0x02    ; SW2 bitmask of portA

LED_0  equ    0x40    ; 7 segment LED char 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

;-----

        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

        movlw   0xFF          ; clear all lines low
        movwf   portB

        movlw   LED_0         ; load the char "0" into the W reg
```

```
    movwf 0x11          ; move the char "0" from W to start addr of char data RAM
    movlw LED_1          ; load the char "1" into the W reg
    movwf 0x12
    movlw LED_2          ; load the char "2" into the W reg
    movwf 0x13
    movlw LED_3          ; load the char "3" into the W reg
    movwf 0x14
    movlw LED_4          ; load the char "4" into the W reg
    movwf 0x15
    movlw LED_5          ; load the char "5" into the W reg
    movwf 0x16
    movlw LED_6          ; load the char "6" into the W reg
    movwf 0x17
    movlw LED_7          ; load the char "7" into the W reg
    movwf 0x18
    movlw LED_8          ; load the char "8" into the W reg
    movwf 0x19
    movlw LED_9          ; load the char "9" into the W reg
    movwf 0x1A
    movlw LED_A          ; load the char "A" into the W reg
    movwf 0x1B
    movlw LED_B          ; load the char "B" into the W reg
    movwf 0x1C
    movlw LED_C          ; load the char "C" into the W reg
    movwf 0x1D
    movlw LED_D          ; load the char "D" into the W reg
    movwf 0x1E
    movlw LED_E          ; load the char "E" into the W reg
    movwf 0x1F
    movlw LED_F          ; load the char "F" into the W reg
    movwf 0x20
```

get_count

```
    movlw 0x11          ; load 0x11 to W reg - maximum count value
    movwf i              ; set index variable to max count value
```

main_loop

```
    ; SW1
    movf    portA,w
    movwf   temp        ; move working to file
    comf    temp,w       ;
    andlw   SW1          ; test SW1
    andwf   event_flag,w ; test event_flag
    movwf   temp        ; stores w reg value (SW1 status) in temp
    btfsc   temp,0       ; test bit 0 of temp (black switch), if button not pressed
skip
    call    event_SW1

    movf    portA,w      ; read portA
    movwf   temp        ;
    comf    temp,w       ;
    andlw   SW1          ; test black switch
    movwf   temp        ; move working to temp
    btfss   temp,0       ; test bit 0 of temp
    bsf     event_flag,0 ; set flag to allow event_sw1
; SW2
    movf    portA,w      ;
    movwf   temp        ;
    comf    temp,w       ;
    andlw   SW2          ; test SW1
    andwf   event_flag,w ; test event_flag
    movwf   temp        ; stores w reg value (SW2 status) in temp
    btfsc   temp,1       ; test bit 1 of temp (black switch), if button not pressed
skip
    call    event_SW2

    movf    portA,w      ; read portA
    movwf   temp        ; move working to file
    comf    temp,w       ;
    andlw   SW2          ; test black switch
    movwf   temp        ; move working to file
    btfss   temp,1       ; test bit 0 of temp
    bsf     event_flag,1 ; set flag to allow event_sw2

    goto    main_loop
```

```
;-----
event_SW1
    decfsz i,f
    goto   continue
    goto   get_count
continue
    ; check portB and count up
    bcf    event_flag,0 ; clear flag to show event complete
    ; j = 15 - i
    movf   i,w          ; move file 'i' into working reg
    sublw  0x10          ; 16 (F) - i
    movwf  j            ; store value into j
    ; FSR = startaddr + j
    addlw  startaddr     ; add j to start addr
    movwf  fsr           ; move start addr to fsr
    ; PortB = INDF
    movf   indf,w        ; pull data from RAM and store in W reg
    movwf  portB         ; copy data to portB

    return
;-----
event_SW2
    ; check portB and count down
    bcf    event_flag,1 ; clear flag to show event complete
    ; j = 15 - i
    movf   i,w          ; move file 'i' into working reg
    ; FSR = startaddr + j
    addlw  startaddr     ; add j to start addr
    movwf  fsr           ; move start addr to fsr
    ; PortB = INDF
    movf   indf,w        ; pull data from RAM and store in W reg
    movwf  portB         ; copy data to portB

    return
;-----
    end
;-----
```

```
; Configuration! Important!  
;  
; at blast time, select:  
;     memory unprotected  
;     watchdog timer disabled  
;     standard crystal (4 MHz)  
;     power-up timer on
```

Task 5:

```
; FILE:      count.asm
; AUTHOR:    Rhys Higgins
; DATE:      28th September 2022
; DESCRIPTION: beeps buzzer at 4khz when button is pressed
;
;
;
;
;      list    p=16F84      ; select processor type
;      radix   hex          ; define radix as hexadecimal
;      #include "p16f84.inc"
;-----
;      cpu equates (memory map)
portA equ    0x05    ; portA address
trisa equ    0x05    ; trisa address in register bank 1

portB equ    0x06    ; portB address
trisb equ    0x06    ; trisb address in register bank 1

indf equ    0x00    ; indir file reg
fsr  equ    0x04    ; indir addr reg

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

event_flag equ    0x11    ; event handler started flag

temp  equ    0x10    ; temp variable

LEDA  equ    0x00    ; LED segment of a bit of portB
```

```
LEDb    equ    0x01    ; LED segment of a bit of portB

SW_B    equ    0x01    ; black switch bitmask of portA
SW_G    equ    0x02    ; green switch bitmask of portA
;-----
        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 on portA 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
        bsf     event_flag,0    ; make sure no events registered

main_loop
        ; black switch
        movf     portA,w        ; read portA
        movwf    temp           ;
        comf     temp,w         ;
        andlw    SW_B           ; test black switch
        andwf    event_flag,w   ; test event_flag
        movwf    temp           ; stores w reg value (SW_B status) in temp
        btfsc    temp,0         ; test bit 0 of temp (black switch), if button not pressed
skip
        call     event_black_sw

        movf     portA,w        ; read portA
        movwf    temp           ;
        comf     temp,w         ;
        andlw    SW_B           ; test black switch
        movwf    temp
```

```
        btfss temp,0      ; test bit 0 of temp
        bsf     event_flag,0 ; set flag to allow event_black_sw

        goto    main_loop    ; return to start of the loop
;-----
event_black_sw
    ; check portB to see if LED is ON or OFF
    bcf     event_flag,0 ; clear flag to show event complete
    movf    portB,w      ; read portB
    xorlw   0x01         ; compliment portB bit 0
    movwf   portB        ; write value back to portB
    call    beep         ; short beep
    return
;-----
beep
                                ; 0x7FF - 2047 Pulses
    movlw   0x07          ; set w = 07 decimal
    movwf   mCount        ; mCount = w: outer loop counter
loadN_beep
    movlw   0xff          ; set w = 255 decimal
    movwf   nCount        ; nCount = w: inner loop counter
decN_beep
    bsf     portB,7       ; set sounder output high
    call    short_pause   ; wait 125us
    bcf     portB,7       ; set sounder output low
    call    short_pause   ; wait 125us

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

    decfsz  mCount, f     ; else decrement Count
    goto    loadN_beep    ; if mCount != 0 then return

    return
;-----
; PAUSE FUNCTION
; DESCRIPTION: Generates ~4kHz clk (2000 pulses == 4kHz)
;             2048 pulses
```

```
;          125us ON
;          125us OFF

short_pause
    movlw 0x29      ; set w = 41 decimal          1
    movwf oCount    ; oCount = w: inner loop counter 1 = 2uS
decN
    decfsz oCount, f ; oCount--                    1
    goto  decN      ; if oCount != 0 then repeat oCount-- 2
                                                    ;*41 = 123uS
    return          ; else exit subroutine

;-----
end
```

Abbreviations:

CU	-	Control Unit
LED	-	Light Emitting Diode
MCU	-	Microcontroller
RAM	-	Random Access Memory
PIC	-	Peripheral Interface Controller
RISC	-	Reduced Instruction Set
CPU	-	Central Processing Unit
SPST	-	Single Pole, Single Throw
PDIP	-	Plastic Dual Inline Package
BJT	-	Bi-polar Junction Transistor
NPN	-	Negative Positive Negative

References:

- [1] Unknown., "Section 29. Instruction Set", Microchip.
<http://ww1.microchip.com/downloads/en/DeviceDoc/31029a.pdf> [Online] Pg. 29-3,
[Accessed 8th November 2022]
- [2] Unknown., "DECSZ", Teach Micro. <https://www.teachmemicro.com/pic-assembly-instruction-set/decfsz/> [Accessed 8th November 2022]
- [3] M. Bates, et. al., "Interfacing PIC Microcontrollers"
<https://www.sciencedirect.com/topics/engineering/file-register> [Online] 2nd Edition, Pg 91-118. Newnes, 2014. [Accessed 8th November 2022]
- [4] N. Agnihotri., "PIC Microcontroller: All You Need To Know", Engineers Garage.
<https://www.engineersgarage.com/pic-microcontroller-all-you-need-to-know/> [Accessed 8th November 2022]
- [5] Unknown., "PIC16F84A Microcontroller introduction and features", Microcontrollers Lab.
<https://microcontrollerslab.com/pic16f84a-microcontroller-introduction-and-features/>
[Accessed 9th November 2022]
- [6] Unknown., "PIC16F84A Data Sheet", Akizukidenshi.
<http://akizukidenshi.com/download/PIC16F84A.pdf> Pg. 3 [Accessed 14th November 2022]
- [7] Unknown., "Quartz Crystal Oscillator", Electronics Tutorials. <https://www.electronics-tutorials.ws/oscillator/crystal.html> [Accessed 09th November 2022]
- [8] Unknown., "MCLR (Master Clear) on PIC microcontrollers", Tech Zorro.
<https://www.techzorro.com/en/blog/mclr/#:~:text=The%20Master%20Clear%20or%20MCLR,or%20disconnecting%20the%20power%20source.> [Accessed 09th November 2022]
- [9] Unknown., "Choosing the Right Crystal and Caps for your Design", Adafruit.
<https://blog.adafruit.com/2012/01/24/choosing-the-right-crystal-and-caps-for-your-design/>
[Accessed 10th November 2022]
- [10] Unknown., "Transistor Circuits", Electronics Club.
<https://electronicsclub.info/transistorcircuits.htm#:~:text=A%20base%20resistor%20limits%20the,fully%20saturated%20when%20switched%20on.> [Accessed 10th November 2022]
- [11] Unknown., "Current Sourcing and Sinking", Starting Electronics.
<https://startingelectronics.org/articles/current-sourcing-sinking/> [Accessed 14th November 2022]
- [12] Unknown., "Resistors for LED Circuits", EE POWER. <https://eepower.com/resistor-guide/resistor-applications/resistor-for-led/#> [Online] Chapter 5, Pg 5. [Accessed 14th November 2022]
- [13] N. Kularatna., "Modern Component Families and Circuit Block Design, 2000", Science Direct.
<https://www.sciencedirect.com/topics/engineering/harvard-architecture> [Accessed 14th November 2022]

- [14] C. Chen, et. al., "RISC vs CISC", Stanford.edu.
<https://cs.stanford.edu/people/eroberts/courses/soco/projects/risc/riscisc/> [Accessed 15th November 2022]
- [15] Unknown., "What is a PIC microcontroller: The Harvard Architecture", PCB Cadence.
<https://resources.pcb.cadence.com/blog/2020-what-is-a-pic-microcontroller-the-harvard-architecture> [Accessed 18th November 2022]
- [16] Unknown., "The TRIS and PORT registers", PCB Heaven.
http://www.pcbheaven.com/picpages/The_TRIS_and_PORT_registers/ [Accessed 18th November 2022]
- [17] Unknown., "SETF and CLRF Instructions for PIC Microcontroller", May 2022.
<https://engineeringxpert.com/setf-and-clrf-instructions-for-pic-microcontroller/> [Accessed 18th November 2022]
- [18] Unknown., "Section 29. Instruction Set", Microchip.
<http://ww1.microchip.com/downloads/en/DeviceDoc/31029a.pdf> [Online] Pg. 29-2 [Accessed 25th November 2022]
- [19] Unknown., "TRISB, Microchip, <https://www.microchip.com/forums/m698147.aspx> [Accessed 25th November 2022]
- [20] Unknown., "The Light Emitting Diode", Electronics Tutorials. https://www.electronics-tutorials.ws/diode/diode_8.html#:~:text=Most%20common%20LED's%20require%20a,being%20the%20most%20common%20orange. [Accessed 25th November 2022]
- [21] Crowell G., "The Forward Voltages of Different LEDs", Circuit Bread.
<https://www.circuitbread.com/ee-faq/the-forward-voltages-of-different-leds> [Accessed 25th November 2022]

RH_MPI's_Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

72 / 100

PAGE 1

PAGE 2

PAGE 3



Comment 1

Do you mean 0.02A or 20mA.

0.020mA = 20μA - this is too small for a LED

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11



Comment 2

You've used a conditional block for LOOP START. What condition allows you to continue to DO LOOP and what condition allows you back to "set maximum count value"? There also seems no way back from the white boxes on the right

PAGE 12

PAGE 13

PAGE 14

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

PAGE 31

PAGE 32

PAGE 33

PAGE 34

PAGE 35

PAGE 36

PAGE 37

GRADING FORM: SMU GENERIC_SMOSEY

RHYS HIGGINS

71.60

ENGAGEMENT WITH LITERATURE

including reading, referencing, academic conventions and academic honesty



Excellent description of the circuit

15

KNOWLEDGE & UNDERSTANDING

Knowledge of the basic underlying concepts and principles of a subject.



Excellent description of the operation of the circuit and its components

15

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.



Very good code - excellent solutions to the problems given.

14

PRACTICAL APPLICATION SKILLS

Different approaches to solving problems in particular contexts.



Simulations should all work. Flowcharts have some odd errors though.

12.6

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.



Very well presented and well written report.

15