

2317629

by Rowan Patey

Submission date: 14-Jan-2025 03:00PM (UTC+0000)

Submission ID: 248428095

File name: 52175_Rowan_Patey_2317629_621295_498991063.docx (943.29K)

Word count: 4470

Character count: 23695

UWTSD

MPI Assignment

[Document subtitle]

Rowan Patey (2317629)
1-13-2025

Table of Contents

1. Hardware	2
1.1 Circuit	2
1.2 Power circuit and sounder	2
1.3 Buttons and MCLR	5
1.4 Clock circuit	7
1.5 7-segment display	8
1.6 16F84A microcontroller	8
2. 7 segment display calculations	9
2.1 0	9
2.2 1	9
2.3 2	10
2.4 3	10
2.5 4	10
2.6 5	10
2.7 6	10
2.8 7	11
2.9 8	11
2.10 9	11
2.11 A	11
2.12 B (b)	12
2.13 C	12
2.14 D (d)	12
2.15 E	12
2.16 F	12
3. Assembly programming	13
3.1 0-F display cycle	13
3.1.1 Assembly code	13
3.1.2 Pseudocode	14
3.2 L and r buttons	16
3.2.1 Assembly code	16
3.2.2 Pseudocode	17
3.3 On/Off buttons	18
3.3.1 Assembly code	18
3.3.2 Pseudocode	19
References	26

1. Hardware

1.1 Circuit

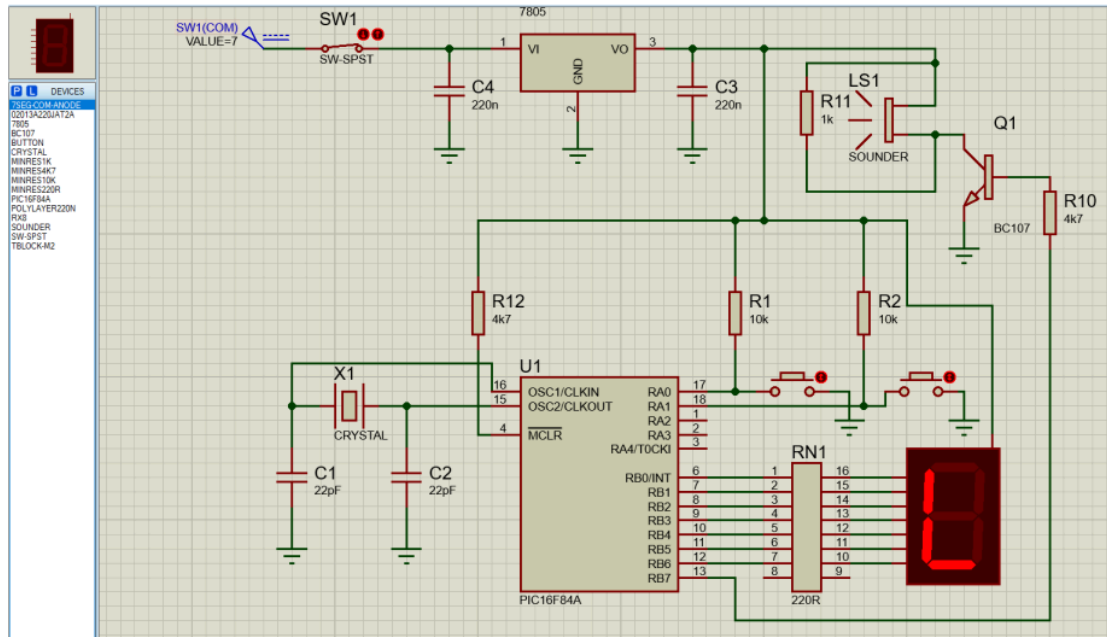


Figure 1. The full outlined circuit of “HND/BENG PIC16F84 Assignment Board” has been recreated in Proteus 8, using the PIC16F84 as the microcontroller for the circuit that can be programmed. Pin B7’s output highlighted in figure 2.

The whole circuit is controlled by the PIC16F84A microcontroller.

1.2 Power circuit and sounder

Figure 2 shows the power circuit for the assignment board. The power line has a +7V D.C. current input going through an SPST switch, SW1. Proteus will power the circuit without adding a power source, however this more accurately represents the real circuit layout of the assignment board. SPST stands for single-pole, single-throw and is the least complex configuration for a switch where only a single input and output are being connected [1]. This switch acts as a basic ON-OFF switch, where current can flow if the poles are connected and

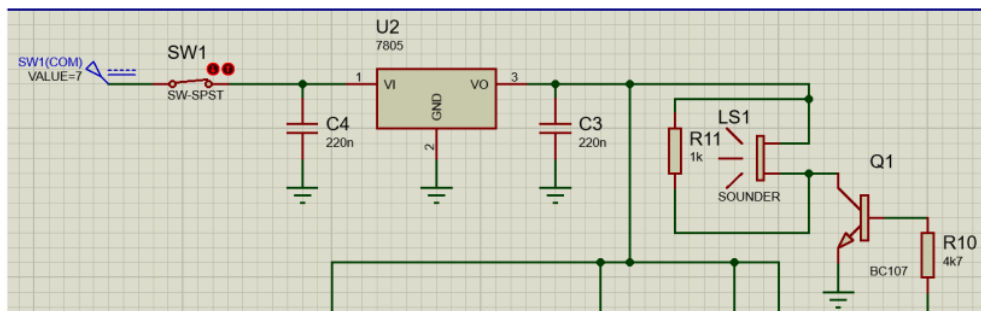


Figure 2. Power circuit for “HND/BENG PIC16F84 Assignment Board” with pin B7 output circuit included

cannot if the switch isn't making contact i.e. in the OFF position. It is perfectly suited to this application where only a single power circuit is being controlled by the switch, compared to say an SPDT switch that could switch between two potential outputs and two potential circuits to power [1]. The switch has a modelled 'closed' resistance of 0.01Ω and a modelled 'open' resistance of $100M\Omega$, meaning current will flow when closed with minimal resistance while practically not flowing (beyond negligible) when the switch is open. The example assignment board uses an SPDT switch, but for the purposes of this recreation an SPST switch is more representative given the SPDT switch of the example board only has a single output connected [2].

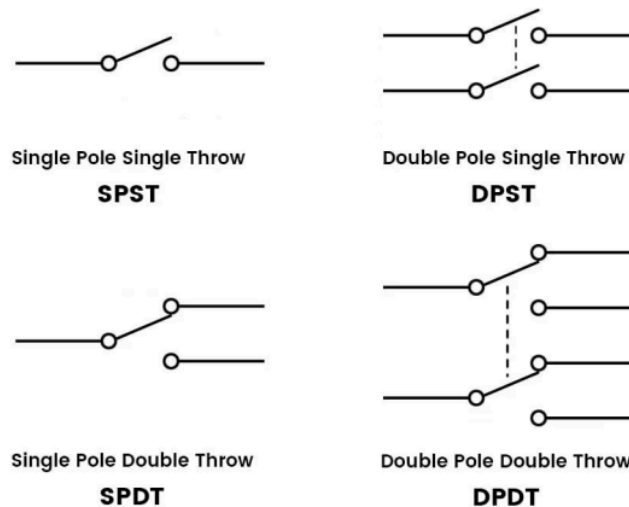


Figure 3. Different switch configurations, with SPST shown left. Switches differ by the number of inputs and outputs they have. [2]

U2 is a 7805 positive linear voltage regulator IC, part of the 78XX series of regulators [3]. Microcontroller's typically have small tolerances for voltage input, therefore it is important to regulate the power being input to the circuit to ensure the desired +5V is fed into the PIC16F84A. Regulator's themselves have a drop-out voltage, the 7805 in particular having a drop-out of 2V, therefore 7-25V is needed as an input to produce a stable regulated +5V output [3]. This dumped voltage is dissipated as waste thermal energy, therefore the IC has a metal heat sink for aiding in the removal of this heat, see figure 3 for the heatsink. Capacitors are needed on either side of the regulator to produce a fixed-voltage regulator effect, per the specifications document [3]. Therefore two 'generic primitive' non-polarised capacitors have been placed at the input (V_i) and output (V_o) pins, both with a value of $220nF$. On paper, the 7805 regulator uses an NPN Darlington power transistor, meaning it can be robust enough not to require surrounding capacitors compared to newer LDO (low drop-out) regulators that depend on output capacitors of match ESR (equivalent series resistance) [4]. The purpose of these capacitors is primarily smoothing, where the capacitors will charge and discharge slowly to smooth out any fluctuations in power flowing around the regulator [5]. For example, the switch before the input will experience 'switch bounce', a phenomenon where in the switch physically bounces against the contact due to the change

in momentum of the metal plate meeting the contact over typically $\sim 1\text{ms}$ [6]. Without the input capacitor, about these rapid spikes and troughs of voltage would be input before the voltage settles to $+7\text{V}$. The input capacitor slowly charges and discharges during this short period, creating a smoother and more definitive 'switch on' of input voltage to aid the regulator in maintaining a clean $+5\text{V}$ regulated output.

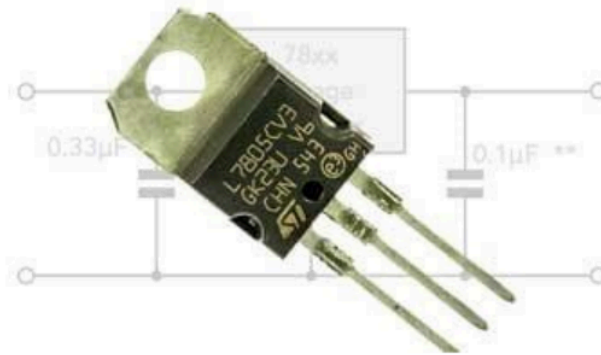


Figure 4. 3-pin 7805 voltage regulator with an inbuilt heat sink for dissipating waste energy [5]

The power circuit then connects to the $+5\text{V}$ rail to supply power to the rest of the circuit.

On the right of figure 2, the spare register port B pin of the PIC16F84A microcontroller, B7, outputs to the transistor, Q1, and sounder, LS1, first through a 4700Ω primitive resistor (R10). Pin B7 is an I/O pin of port B of the PIC16F84A with no other functionality. The I/O pins have the ability to either be inputs or outputs for the controller but crucially only input/output digital signals of 0V to 5V . They cannot be inputting and outputting at the same time, therefore B7 will be set within the TRISB data direction register to be an output pin for outputting a digital signal to the sounder [7]. The BC107 transistor and preliminary 4700Ω resistor behave like an amplification system for the sounder. The resistor acts to limit the current of the incoming $+5\text{V}$ digital signal from B7 to provide a suitable base current to drive the BC107, while being substantial enough to ensure the pin is otherwise not driving the base when not intended to be set. BC107 is an NPN transistor that is used as a 'low noise general purpose audio amplifier' [8]. As a BJT it is current driven, the BC107 will 'switch' when sufficient current is supplied to the base. Far greater current can flow from the collector to the emitter, driving the sounder by creating a complete circuit to ground through the sounder down to the emitter [9].

The circuit uses a piezoelectric sounder, LS1: Sounders such as this typically consist of a metal backing plate with a layer of piezoelectric element, usually ceramic, on top with a plastic resonator chamber around it [10]. relying on the piezoelectric effect to function. When a piezoelectric element is physically deformed, it generates a small A.C current [11]. Placing an alternating voltage across the same piezoelectric element causes it to flex and vibrate against the metal plate, creating a buzz at the frequency of the input signal [11]. The whole circuit requires a D.C. voltage input from the power rail, as such it is assumed that the piezo sounder module within Proteus is an IC that can alternate an input signal, given the

simulated sounder produces a clean single sound when testing with this configuration. The D.C. power rail is feeding into the sounder directly but a tone is still produced when the BC107 has a saturated base, contrary to the theory outlined for piezo sounders requiring an A.C. voltage to vibrate. D.C. can even depolarize the ceramic element of the sounder in a real circuit [10]. Often piezo sounders need to be driven by flipping the polarity of the two pins, however the sounder in this simulated configuration appears to be sufficiently powered by setting one of the sounder's pin's low while connecting the other to the power rail, contrary to sources [10].

1.3 Buttons and MCLR

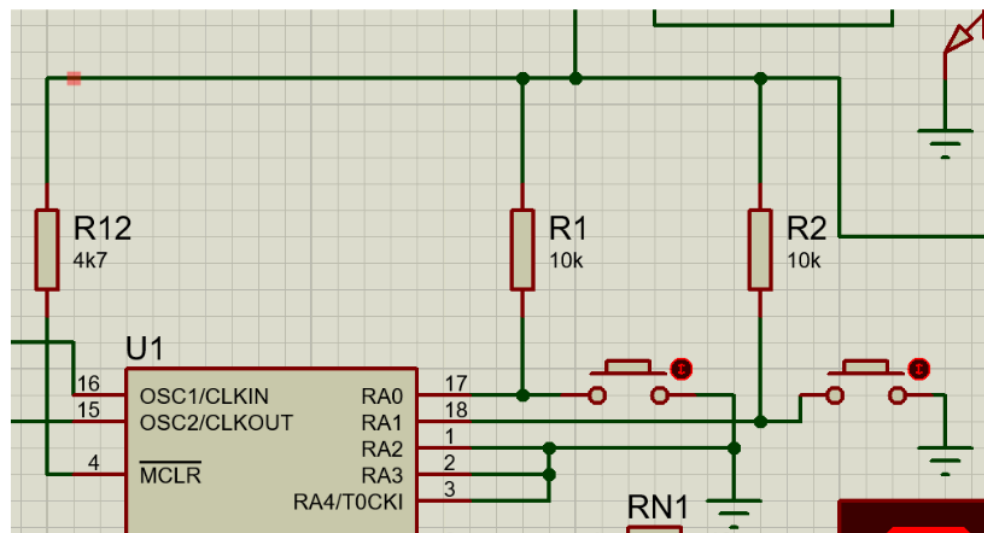


Figure 5. Interfacing circuit, fed from the power rail above, with two 'push to break' buttons connected to pins A0 and A1 of port A, with a connection line to the master clear pin MCLR.

Feeding in from the power rail above, described in 1.2, register port A of the PIC16F84A is shown. Two pushbuttons have been connected to pins A0 and A1 of port A. To have this configuration, like pin B7 of port B both of these pins must be set using the internal register for port A [7]. In this scenario, the two pins will function as input pins as these will serve as the controls of the system. However, this configuration utilises 'push-to-break' buttons, seen though the pull up resistors and grounding connection on each side of the buttons [12]. Pressing the buttons completes the circuit to ground from the power rail through the pull up resistors, R1 and R2. Pins A0 and A1 will be held logical high when the buttons are not depressed and brought to logical low when the buttons are depressed in turn [12]. The 10k Ω pull-up resistors cause little voltage drop across their pins, meaning almost all of the +5V input signal from the power rail is received at the pins; when the buttons are pressed, the high resistance ensures that the current flowing through to ground from the power rail is small enough to not short the whole circuit [12]. If the pull-up resistor values are too great, a variety of issues can arise: Of particular note here, the resistors form R-C circuits with the output capacitor of the regulator, meaning the time constant, τ , will increase

proportionally with the magnitude of the resistance [13]. This will directly slow signal transition speeds where the output capacitor will discharge slowly upon a button press, causing noticeable delay to the pin state transition. The capacitor recharging slowly after a button press drains the stored charge would exacerbate switch bounce where multiple inputs may be registered from the low to high transition taking much longer. Larger resistors make the pins more susceptible to electrical interference due to the increased impedance at the pin. Alongside this, internal impedance at the pin and the external resistor form a voltage divider, dividing the input signal too heavily to read as a logical high input [14].

All of the unused register port A pins are connected to ground given they are not in use and need to complete a circuit to ground to ensure they do not introduce interference into the microcontroller.

The power rail is also connected to the MCLR pin: MCLR stands for 'master clear' and acts as a reset pin [7]. The bar over the top implies that you the pin must be brought low, an active low reset, therefore in this configuration the MCLR pin will be held high until power from the power rail is disrupted. This is an appropriate function for cases where the circuit loses power, where the MCLR reset will reset all of the internal registers and reset the PIC controller to a known state, where the drop of power may have caused corruptions or overwrites in the memory registers. The 4700Ω resistor behaves similarly to the button pull-up resistors, ensuring that little voltage drop occurs over the component to have the full +5V input from the power rail, whilst heavily limiting current in the scenario that the pin is dropped low [12]. This in turn will protect the microcontroller. All three of the pull-up resistors provide protection against noise interference with the pin there are respectively protecting [12].

1.4 Clock circuit

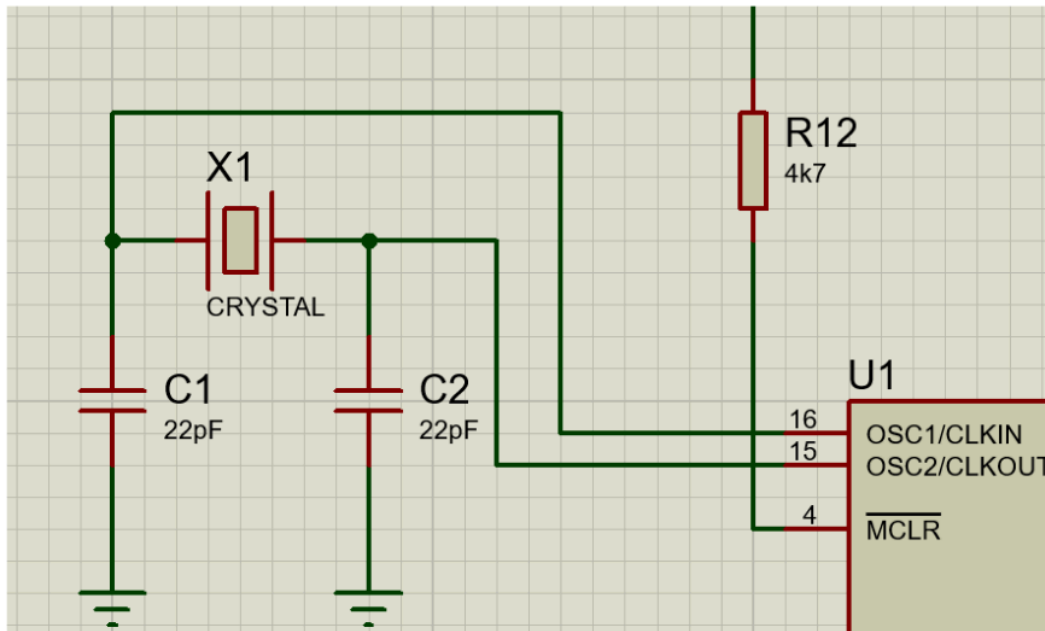


Figure 6. Clock circuit for the PIC16F84A simulated circuit, with a crystal oscillator for robustness and accuracy.

The PIC16F84A microcontroller is clocked to 4MHz in the simulation. To achieve this an external oscillator circuit is incorporated. A crystal oscillator, X1, is connected to the CLKIN and CLKOUT pins of the PIC controller [7]. The PIC controller specification outlines that a parallel cut crystal is required for this function and connection style. Once the crystal oscillator unit is turned on, through the same piezoelectric effect (described in 1.2), it will produce a consistent +5V square wave clock signal [15] that feeds into OSC1 [7]. Between OSC1 and 2 internally there is an inverter amplifier that ensures any minor losses when going through the clock circuit will be regained, as well as maintaining the fundamental frequency required [7].

An RC oscillator circuit could have been used for this clock circuit. This can be achieved by feeding an RC component into OSC1, which internally has a Schmitt inverter for this purpose [7]. However, the accuracy and stability of this type of circuit is worse than a comparative crystal oscillator circuit. This is caused by the low tolerances of resistance and capacitance typically found on resistors and capacitors, meaning frequency can't be guaranteed. Temperature increases can also cause drifting in the clock frequency produced by an RC circuit.

1.5 7-segment display

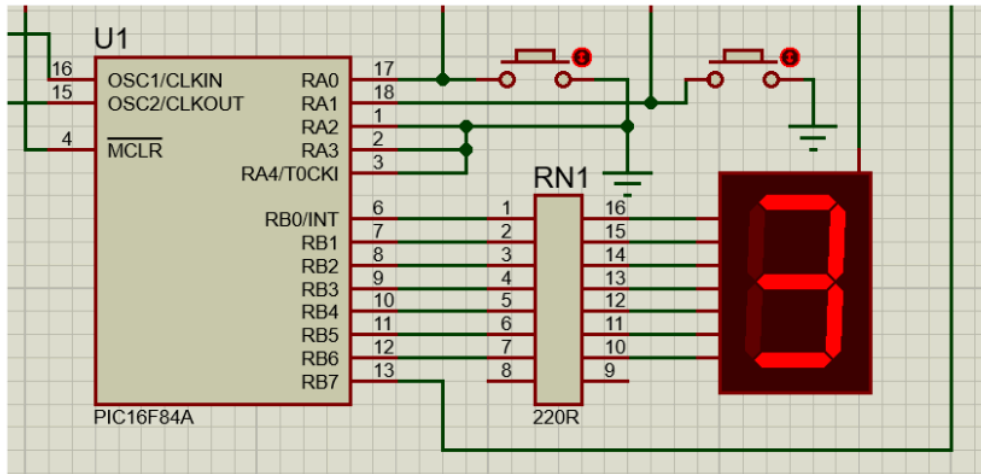


Figure 7.7 segment display connected to port B of the PIC16F84A with a series resistor network to adjust for the voltage drop of the display LEDs.

A 7-segment display inhabits 7 of the 8 register port B output pins. Port B is a data register that is 8 bits, a byte, wide, each bit corresponding to one of the I/O pins [7]. Port B can be written to, hence changing the output on each output line. All of the B pins here must be set to output pins by setting the TRISB register, another byte wide register that determines the I/O setting of each I/O pin in port B [7]. RN1 is a dual inline resistor network that simply functions to apply the determined 220Ω resistance to each output line. The resistance is based upon the voltage drop of the LEDs in the display that each line will drive. This particular display is a common anode display, indicating that the individual cathode outputs are fed by a shared anode terminal [16]. This common anode is connected to the +5V power rail. Therefore, this display requires lines to be pulled low to turn on relevant LEDs, where they are otherwise held high; 0V are required on the I/O pins because they are sinking the current/voltage from the LED display. Despite sinking current, the pins are still output pins given they are controlling the flow of current through the display's LEDs by transitioning between states. As alluded to, each segment is labelled with a letter and is directly fed by an I/O pin, therefore the binary value written to port B will determine the shape created on the display. Figure 8 in 2. shows the most common arrangement, where a is the LSb, driven by pin B0, with the other letters increasing in binary order magnitude up the output pins until reach g, the MSb, driven by pin B6. This display does not have an eighth bit, sometimes seen as a decimal point, therefore only the 7 pins are required to drive it.

1.6 16F84A microcontroller

Important features of the microcontroller relating to the functioning of this circuit have largely been covered in previous sections so to avoid redundancy, only features not mentioned prior will be outlined here.

The 16F84A is a RISC architecture computer IC that functions on a greatly reduced instruction set of just 35 instructions [17]. It carries a working (W register) that can be used to store values for ALU operations, a 13 bit program counter for storing instruction addresses, an 8 level stack for storing return addresses and POP'ing them back out as well as a status register [18]. All of these memory features will be useful when programming the circuit.

Internally, the controller carries hardware configuration 'fuses' that can be set in registers to modify the behaviour of the controller. Crucial configuration settings for this purpose include: Disabling the watchdog time, setting the clock frequency to (4 MHz), power-up timer on, MCLR being set on as well as the OSC pin settings for managing and receiving an external clock source with the oscillator [7] [19].

2. 7 segment display calculations

This circuit will eventually output a 7 bit value to port B of the controller, setting corresponding output pins from LSb to MSb on B0 to B6 respectfully and finally outputting to a 7 segment display as detailed in 1.5. Each segment of the display will correspond to a bit of the B register output, seen in *figure 8*. Therefore the display pattern required for each value from 0 to F (hexadecimal) will need to be determined before calculating what the required input hexadecimal value is to achieve each pattern. Given the display is a **common anode** display, **setting** the corresponding bit turns **off** the desired LED segment, **clearing** the corresponding bit turns **on** the desired LED segment. A logical inversion, NOT, must be applied to the determined binary value to find the value that needs inputting to port B. The values calculated will be for 7 bit values despite port B being a byte wide, due to the used seven segment display not having an eighth input alongside port B7 being used for the sounder instead.

2.1 0

Segments required: A, B, C, D, E, F, 0
Binary output: NOT(0b0111111) = MSb 0b1000000 LSb
Hexadecimal value:
High nibble = X100
Low nibble = 0000
Hex = 0x40

2.2 1

Segments required: 0, B, C, 0, 0, 0, 0
Binary output: NOT(0b0000110) = MSb 0b1111001 LSb
Hexadecimal value:
High nibble = X111
Low nibble = 1001
Hex = 0x79

2.3 2

Segments required: A, B, 0, D, E, 0, G
Binary output: NOT(0b1011011) = MSb 0b0100100 LSB
Hexadecimal value:
High nibble = X010
Low nibble = 0100
Hex = 0x24

2.4 3

Segments required: A, B, C, D, E, F
Binary output: NOT(0b1001111) = MSb 0b0110000 LSB
Hexadecimal value:
High nibble = X011
Low nibble = 0000
Hex = 0x30

2.5 4

Segments required: 0, B, C, 0, 0, F, G
Binary output: NOT(0b1100110) = MSb 0b0011001 LSB
Hexadecimal value:
High nibble = X001
Low nibble = 1001
Hex = 0x19

2.6 5

Segments required: A, 0, C, D, 0, F, G
Binary output: NOT(0b1101101) = MSb 0b0010010 LSB
Hexadecimal value:
High nibble = X001
Low nibble = 0010
Hex = 0x12

2.7 6

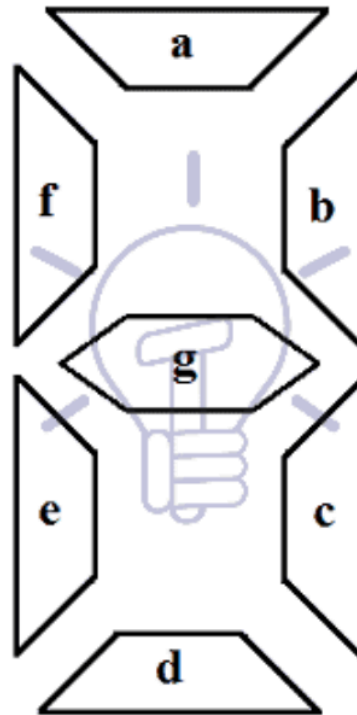


Figure 8. Common configuration for 7 segment display, a = LSB, g = MSb [16]

Segments required: A, 0, C, D, E, F, G
Binary output: NOT(0b1111101) = MSb 0b0000010 LSB
Hexadecimal value:
High nibble = X000
Low nibble = 0010
Hex = 0x02

2.8 7

Segments required: A, B, C, 0, 0, 0, 0
Binary output: NOT(0b0000111) = MSb 0b1111000 LSB
Hexadecimal value:
High nibble = X111
Low nibble = 1000
Hex = 0x78

2.9 8

Segments required: A, B, C, D, E, F, G
Binary output: NOT(0b1111111) = MSb 0b0000000 LSB
Hexadecimal value:
High nibble = X000
Low nibble = 0000
Hex = 0x00

2.10 9

Segments required: A, B, C, D, 0, F, G
Binary output: NOT(0b1101111) = MSb 0b0010000 LSB
Hexadecimal value:
High nibble = X001
Low nibble = 0000
Hex = 0x10

2.11 A

Segments required: A, B, C, 0, E, F, G
Binary output: NOT(0b1110111) = MSb 0b0001000 LSB
Hexadecimal value:
High nibble = X000
Low nibble = 1000
Hex = 0x08

2.12 B (b)

Segments required: 0, 0, C, D, E, F, G

Binary output: NOT(0b1111100) = MSb 0b0000011 LSb

Hexadecimal value:

High nibble = X000

Low nibble = 0011

Hex = 0x03

2.13 C

Segments required: A, 0, 0, D, E, F, 0

Binary output: NOT(0b0111001) = MSb 0b1000110 LSb

Hexadecimal value:

High nibble = X100

Low nibble = 0110

Hex = 0x46

2.14 D (d)

Segments required: 0, B, C, D, E, 0, G

Binary output: NOT(0b1011110) = MSb 0b0100001 LSb

Hexadecimal value:

High nibble = X010

Low nibble = 0001

Hex = 0x21

2.15 E

Segments required: A, 0, 0, D, E, F, G

Binary output: NOT(0b1111001) = MSb 0b0000110 LSb

Hexadecimal value:

High nibble = X000

Low nibble = 0110

Hex = 0x06

2.16 F

Segments required: A, 0, 0, 0, F, E, G

Binary output: NOT(0b1110001) = MSb 0b0001110 LSb

Hexadecimal value:

High nibble = X000

Low nibble = 1110
Hex = 0x0E

3. Assembly programming

3.1 Hex display program

3.1.1 Assembly code

```
main.asm
1 ;=====
2 ; DEFINITIONS - define registers
3 ;=====
4 #include p16f84a.inc           ; Including the register definition file for the PIC16F84A
5
6 portB equ 0x06                 ; Port B address
7 trisB equ 0x06                 ; trisB address in register bank 1
8
9 status equ 0x03                ; Status register
10 RP0 equ 5                     ; Bit 5 of status register, bank select register bit
11 ;=====
12 ; VARIABLES - MUST BE STORED IN RAM therefore these must point to a RAM memory address
13 ;=====
14 count equ 0x0c                 ; Counter 0 to 255 pointing to first available RAM address
15 nCount equ 0x0d                ; Pause routine inner Loop counter - used for timing loop
16 mCount equ 0x0e                ; Pause routine outer Loop counter - used for timing loop
17 ;=====
18 ; RESET and INTERRUPT VECTORS
19 ;=====
20 RST code 0x00                 ; Reset vector for when a reset pulse is recieved
21 goto Start                     ; Restarts the program at the beginning of the main program
22 ;=====
23 ; CODE SEGMENT
24 ;=====
25 PGM code
26 Start                          ; Start of program
27 ; Port setup
28 bsf status,RP0                 ; Select bank 1 and set the RP0 bit in the status register
29 movlw 0x00                     ; Load w with value 0x00
30 movwf trisB                    ; Load it into trisb to set port B as all output pins
31 bcf status,RP0                 ; Reselect bank 0, deselected bank b and reselects bank a
32 ; Initialize
33 clrf count                     ; Start counter at 0
34 movlw 0xff                     ; NOT(0b00000000) = 0b11111111 = 0xff
35 movwf portB                    ; Clear all lines low of port B to ensure no LEDs lit up
36 ; Main Loop
37 displayLoop
38 movf count,W                   ; Move the current count to w register
39 call table                     ; Call the corresponding value from the lookup table
40 movwf portB                    ; Output the value from the lookup table to the display
41 call pause                     ; Delay for 200ms
42
```

Figure 9. Line 1 to 42 of the hex display program

```

main.asm
43      incf    count, F           ; Increment counter variable
44      movlw  0x10              ; Load 16 (0x10)
45      subwf  count, W           ; Mathematical subtraction of 16 in W register from counter
46      btfsc  status, Z         ; Test if zero bit in the status register is 0 indicating W and count were both 16
47      clrf   count             ; Initialize counter to 0 if 16 was reached
48      goto   displayLoop       ; Repeat the main loop
49
50      ;=====
51      ;Pause Subroutine, calling pause should implement about a ~200ms delay
52      ;Two Loops using "mCount" and "nCount" register files
53      ;"mCount" is the outer loop counter (255 down to zero)
54      ;"nCount" is the inner loop counter (255 down to zero)
55      ;=====
56      pause
57      movlw  0xff              ; Set w = 255 decimal
58      movwf  mCount            ; Set mCount = w for outer loop counter
59
60      movlw  0xff              ; Set w = 255 decimal
61      movwf  nCount            ; Set mCount = w for inner loop counter
62
63      decN   nCount, F         ; Decrement inner counter, skip if zero
64      goto   decN              ; Repeat inner loop if not zero
65      decfsz mCount, F         ; Decrement outer counter, skip if zero
66      goto   loadN             ; Repeat outer loop if not zero
67      return                    ; Else exit subroutine jumps back to the previous position in displayLoop
68
69      ;=====
70      ; Lookup table containing display patterns for 7-segment (0-F)
71      ; Program will jump to the table at the indexed value indicated by W register and then return with the value
72      ;=====
73      table
74      addwf  PCL               ; Mathematical addition of the current w register value to PCL
75      retlw  0x40              ; 0, return to break out point with this value saved in w register
76      retlw  0x79              ; 1
77      retlw  0x24              ; 2
78      retlw  0x30              ; 3
79      retlw  0x19              ; 4
80      retlw  0x12              ; 5
81      retlw  0x02              ; 6
82      retlw  0x78              ; 7
83      retlw  0x00              ; 8
84      retlw  0x10              ; 9
85      retlw  0x08              ; A
86      retlw  0x03              ; b
87      retlw  0x46              ; C
88      retlw  0x21              ; d
89      retlw  0x06              ; E
90      retlw  0x0E              ; F
91      end

```

Figure 10. Line 43 to 90 of the hex display program

3.1.2 Pseudocode

BEGIN HexDisplay Program

Variables

```

DECLARE INTEGER counter
DECLARE INTEGER InnerCount
DECLARE INTEGER OuterCount
DECLARE ARRAY DisplayPatterns[16]

```

Initialize display patterns for hex digits 0-F

```

SET DisplayPatterns[0] TO 0xC0

```

...

Initialize ports

```

SET status_RP0 TO 1 (select bank 1)
SET port B direction TO 0x00 (output)
SET status_RP0 TO 0 (reselect bank 0)

```

Clear initial values

SET counter TO 0

SET port B TO 0xFF to turn off the LEDs

Main program loop

WHILE TRUE

 READ counter value TO W register

 GET pattern FROM DisplayPatterns[W register] (index based on W register value)

 WRITE pattern TO port B

Delay subroutine (~200ms)

SET OuterCount TO 255

WHILE OuterCount > 0

 SET InnerCount TO 255

 WHILE InnerCount > 0

 SUBTRACT 1 FROM InnerCount

 ENDWHILE

 SUBTRACT 1 FROM OuterCount

ENDWHILE

Update counter

ADD 1 TO counter

IF counter = 16 THEN

 SET counter TO 0

ENDIF

ENDWHILE

END HexDisplay

3.2 L and r buttons

3.2.1 Assembly code

```
main.asm
4 ;=====
5
6 #include p16f84a.inc           ; Including the register definition file for the PIC16F84A
7
8 portA equ 0x05                 ; PortA address
9 trisa equ 0x05                 ; Trisa address in register bank 0
10
11 portB equ 0x06                 ; Port B address
12 trisb equ 0x06                 ; TrisB address in register bank 1
13
14 status equ 0x03                ; Status register
15 RP0 equ 5                     ; Bit 5 of status register, bank select register bit
16 ;=====
17 ; VARIABLES - MUST BE STORED IN RAM therefore these must point to a RAM memory address
18 ;=====
19 count equ 0x00                ; Debounce delay counter register
20
21 ;=====
22 ; RESET and INTERRUPT VECTORS
23 ;=====
24 RST code 0x00                 ; Reset vector for when a reset pulse is recieved
25 goto Start                    ; Restarts the program at the beginning of the main program
26 ;=====
27 ; CODE SEGMENT
28 ;=====
29 PGM code
30 Start                          ; Start of program
31 ; Port setup
32 bsf status,RP0                ; Select bank 1 and set the RP0 bit in the status register
33 movlw 0x00                    ; Load W with value 0x00, 0b00000000
34 movwf trisb                    ; Load it into trisB to set port B as all output pins
35
36 movlw 0x0F                    ; Load W with value 0x0F, 0b00001111
37 movwf trisa                    ; Load it into trisa to set the lower nibble of the trisa register as inputs
38 bcf status,RP0                ; Reselect bank 0, deselects bank b and reselects bank a
39
40 clrf count                     ; Clear delay counter variable to 0
41 clrf portB                     ; Clear all lines low of port B to light up the LEDs
42
43 ; Main Loop
44 buttonCheck                    ; Button check Loop starts
45 movf portA, W                  ; Moves the port A value into the W register
46
47 btfss portA, 0                 ; Bit testing pin A0 (port A 0 index), skips the next line if 1 (pressed)
48 call buttonL                   ; Calls the subroutine to display L on the display if pressed
49
50 btfss portA, 1                 ; Bit testing pin A1 (port A 1 index), skips the next line if 1 (pressed)
51 call buttonR                   ; Calls the subroutine to display r on the display if pressed
52
53 goto buttonCheck               ; Loop around again to continue checking for button interrupts
54
55 buttonL:
56 movlw 0x47                     ; NOT(0b0111000) = 0b1000111 = 0x47 moved into W register
57 movwf portB                    ; Move the hexadecimal into port B causing it to display the pattern on screen
58 call delay                     ; Calls the delay subroutine to run for debounce
59 return                          ; Exit subroutine and jump back to buttonCheck
60
61 buttonR:
62 movlw 0x2F                     ; NOT(0b1010000) = 0b0101111 = 0x2F moved into W register
63 movwf portB                    ; Move the hexadecimal into port B causing it to display the pattern on screen
64 call delay                     ; Calls the delay subroutine to run for debounce
65 return                          ; Exit subroutine and jump back to buttonCheck
66
67 ;=====
68 ; Delay Subroutine, shortened version of previous pause subroutine
69 ; For implementing software debounce
70 ;=====
71 delay
72 movlw 0xFF                     ; Set W register = 255 decimal
73 movwf count                    ; Set count = W for therefore count = 255 decimal
74
75 delayDec
76 decfsz count, F                ; Decrement the delay counter variable, skip if it has reached zero
77 goto delayDec                  ; Repeat previous decrement if counter variable is not zero
78 return                          ; Else exit subroutine jumps back to wherever it was called from
79
80 end
```

Figure 11. Full L/r button press program.

3.2.2 Pseudocode

BEGIN LRbutton program

Variables

DECLARE INTEGER counter

Initialize ports

SET status_RP0 TO 1

SET port B direction TO 0x00 (output)

SET port A direction TO 0x0F (input on the lower nibble of the register)

SET status_RP0 TO 0

Clear initial values

SET counter TO 0

SET port B TO 0x00 (turn the LEDs on)

Main program loop

WHILE TRUE

 READ port A TO W register

 IF index0 port A bit = 0 THEN

 CALL DisplayLetterL (as left button has been pressed)

 ENDIF

 IF index1 port A bit = 0 THEN

 CALL DisplayLetterR (as right button has been pressed)

 ENDIF

ENDWHILE

L button

SUBROUTINE DisplayLetterL

 SET W register TO 0x47

 WRITE W register TO port_b

 CALL Debounce

 RETURN

END SUBROUTINE

R button

SUBROUTINE DisplayLetterR

 SET W register TO 0x2F

 WRITE W register TO port B

 CALL Debounce

 RETURN

ENDSUBROUTINE

Debounce delay

SUBROUTINE Debounce

 SET counter TO 255

 WHILE counter > 0

 SUBTRACT 1 FROM counter

 ENDWHILE

 RETURN

ENDSUBROUTINE

END LRbutton program

3.3 On/Off buttons

3.3.1 Assembly code

```
main.asm
1
2 ;=====
3 ; DEFINITIONS - define registers testing
4 ;=====
5
6 #include p16f84a.inc           ; Including the register definition file for the PIC16F84A
7
8 portA equ 0x05                 ; PortA address
9 trisa equ 0x05                 ; Trisa address in register bank 0
10
11 portB equ 0x06                 ; Port B address
12 trisb equ 0x06                 ; TrisB address in register bank 1
13
14 status equ 0x03                ; Status register
15 RP0 equ 5                      ; Bit 5 of status register, bank select register bit
16 ;=====
17 ; VARIABLES - MUST BE STORED IN RAM therefore these must point to a RAM memory address
18 ;=====
19 count equ 0x0D                 ; Debounce delay counter register
20 ;=====
21 ; RESET and INTERRUPT VECTORS
22 ;=====
23
24 RST code 0x00                  ; Reset vector for when a reset pulse is recieved
25 goto Start                     ; Restarts the program at the beginning of the main program
26 ;=====
27 ; CODE SEGMENT
28 ;=====
29 PG0 code                       ; Start of program
30 Start
31 ; Port setup
32 bsf status,RP0                 ; Select bank 1 and set the RP0 bit in the status register
33 movlw 0x00                     ; Load W with value 0x00, 0b00000000
34 movwf trisb                     ; Load it into trisb to set port B as all output pins
35
36 movlw 0x0F                     ; Load W with value 0x0F, 0b00001111
37 movwf trisa                     ; Load it into trisa to set the lower nibble of the trisa register as inputs
38 bcf status,RP0                 ; Reselect bank 0, deselects bank b and reselects bank a
39
40 clrf count                     ; Clear delay counter variable to 0
41 movlw 0x79                     ; NOT(0b0000110) = 0b1111001 = 0x79 moved into W register
42 movwf portB                     ; Move the hexadecimal into port B causing it to display a '1' on screen as a start point
43
44 ; Main Loop
45 buttonCheck                     ; Button check Loop starts
46 movf portA,W                   ; Moves the port A value into the W register
47
48 btfss portA,0                   ; Bit testing pin A0 (port A 0 index), skips the next line if 1 (pressed)
49 call buttonON                   ; Calls the subroutine to turn on all of the display LEDs
50
51 btfss portA,1                   ; Bit testing pin A1 (port A 1 index), skips the next line if 1 (pressed)
52 call buttonOFF                  ; Calls the subroutine to turn off all of the display LEDs
53
54 goto buttonCheck               ; Loop around again to continue checking for button interrupts
55
56 ; Button subroutines will naturally latch given port B is being rewritten, without another rewrite occurring until another button interrupt occurs
57 buttonON:
58 movlw 0x00                     ; NOT(0b1111111) = 0b0000000 = 0x00 moved into W register
59 movwf portB                     ; Move the hexadecimal into port B causing it to turn on all of the LEDs
60 call delay                     ; Calls the delay subroutine to run for debounce
61 return                          ; Exit subroutine and jump back to buttonCheck
62
63 buttonOFF:
64 movlw 0x7F                     ; NOT(0b0000000) = 0b1111111 = 0x7F moved into W register
65 movwf portB                     ; Move the hexadecimal into port B causing it to turn off all of the LEDs
66 call delay                     ; Calls the delay subroutine to run for debounce
67 return                          ; Exit subroutine and jump back to buttonCheck
68
69 ;=====
70 ; Delay Subroutine, shortened version of previous pause subroutine
71 ; For implementing software debounce
72 ;=====
73 delay
74 movlw 0xFF                     ; Set W register = 255 decimal
75 movwf count                     ; Set count = W for therefore count = 255 decimal
76
77 delayDec
78 decfsz count,F                 ; Decrement the delay counter variable, skip if it has reached zero
79 goto delayDec                  ; Repeat previous decrement if counter variable is not zero
80 return                          ; Else exit subroutine jumps back to wherever it was called from
81 end
```

Figure 12. Full ON/OFF buttons program.

3.3.2 Pseudocode

BEGIN ONOFFButtons program

Variables

DECLARE INTEGER counter

Initialize ports

SET status_RP0 TO 1

SET port B direction TO 0x00 (output)

SET port A direction TO 0x0F (input)

SET status_RP0 TO 0

Clear initial values

SET counter TO 0

SET W register TO 0x79

WRITE W register TO port B (sets the starting display pattern as 1 for easy differentiation)

Main program loop

WHILE TRUE

 READ port A TO W register

 IF index0 port A bit = 0 THEN (checking if the left button has been pressed)

 CALL AllLEDsOn

 ENDIF

 IF index1 port A bit = 0 THEN (checking if the right button has been pressed)

 CALL AllLEDsOff

 ENDIF

ENDWHILE

Turning all LEDs on

SUBROUTINE AllLEDsOn

 SET W register TO 0x00

 WRITE W register TO port B (common anode display therefore 0x00 turns on all LEDs)

 CALL Debounce

 RETURN

ENDSUBROUTINE

Turning all LEDs off

SUBROUTINE AllLEDsOff

 SET W register TO 0x7F

 WRITE W register TO port B (common anode display therefore 0x7F turns off all LEDs)

 CALL Debounce (small delay introduced to make each button press reading discrete)

 RETURN

ENDSUBROUTINE

Contd.

Debounce delay

SUBROUTINE Debounce

SET counter TO 255

WHILE counter > 0

SUBTRACT 1 FROM counter

ENDWHILE

RETURN

ENDSUBROUTINE

END ONOFFButtons

3.4 Counter controlled by buttons

3.4.1 Assembly code

```
main.asm
1
2 ;=====
3 ; DEFINITIONS - define registers testing
4 ;=====
5
6 #include p16f84a.inc           ; Including the register definition file for the PIC16F84A
7
8 portA equ 0x05                ; PortA address
9 trisa equ 0x05                ; TrisA address in register bank 0
10
11 portB equ 0x06                ; Port B address
12 trisb equ 0x06                ; TrisB address in register bank 1
13
14 status equ 0x03               ; Status register
15 RP0 equ 5                     ; Bit 5 of status register, bank select register bit
16 ;=====
17 ; VARIABLES - MUST BE STORED IN RAM therefore these must point to a RAM memory address
18 ;=====
19 dcount equ 0x00               ; Delay counter register
20
21 ;=====
22 ; RESET and INTERRUPT VECTORS
23 ;=====
24 RST code 0x00                 ; Reset vector for when a reset pulse is recieved
25 goto Start                    ; Restarts the program at the beginning of the main program
26 ;=====
27 ; CODE SEGMENT
28 ;=====
29 PGM code
30 Start                          ; Start of program
31 ; Port setup
32 bsf status,RP0                ; Select bank 1 and set the RP0 bit in the status register
33 movlw 0x00                    ; Load W with value 0x00, 0b00000000
34 movwf trisb                    ; Load it into trisB to set port B as all output pins
35
36 movlw 0x0F                    ; Load W with value 0x0F, 0b00001111
37 movwf trisa                    ; Load it into trisa to set the lower nibble of the trisa register as inputs
38 bcf status,RP0                ; Reselect bank 0, deselects bank b and reselects bank a
39
40 clrf dcount                    ; Clear delay counter variable to 0
41
```

```

43
44 ; Main Loop
45 counterDisplay
46     movf    portA,W           ; Read in the current input button states to the W register
47     xorwf   prevNum,W         ; Logical XOR the saved previous value against the current port A value
48     andlw   0x03              ; Mask all but the two button bits through a logical AND against 0b00000011
49     btfsc   status,Z          ; Bit test the zero bit in the status register, will skip the next instruction as there was no change if Zb= 0
50     goto    displayUp         ; If Zb= 1, jumps to the display update subroutine
51
52     movf    portA,W           ; Read in the current input button states to the W register
53     movwf   prevNum           ; Store this value as the new previous value displayed
54
55     btfss   portA,0           ; Bit testing the A0 port of port A, where a button press would pull this pin low and reflect in the register as a 0.
56     call    countDec          ; Will be skipped if the button hadn't been pressed as the previous instruction skips if set, otherwise this jumps to the decrement subroutine
57
58     btfss   portA,1           ; Bit testing the A1 pin of port A, where a button press would pull this pin low and reflect in the register as a 0.
59     call    countInc          ; Will be skipped if the button hadn't been pressed as the previous instruction skips if set, otherwise this jumps to the increment subroutine
60
61 ;=====
62 ; DISPLAY UPDATE SUBROUTINE
63 ;=====
64 displayUp
65     movf    count,W           ; Write current counter value into W register
66     call    table             ; Jump to the lookup table to the index corresponding to the count value
67     movwf   portB             ; Move the value that is returned from the table into the port B register
68     call    delay             ; Jump to the debounce delay subroutine to introduce some minor delay
69     goto    counterDisplay    ; Return to the main loop
70
71 ;=====
72 ; INCREMENT SUBROUTINE
73 ;=====
74 countInc
75     movlw   0x09              ; Load maximum count value allowed for the program into the W register
76     subwf   count,W           ; Check if the counter has reached the maximum by a mathematical subtraction of the count value from the maximum allowed value
77     btfsc   status,Z          ; If counter != max value, this bit test would trigger and skip the next instruction
78     return                                ; Returns to the main loop if the count did == 9
79     incf    count,F           ; Increment the counter by +1 if the count != 9, where the tested bit would have been set to trigger the skip
80     return                                ; Return from the subroutine to the main loop
81
82 ;=====
83 ; DECREMENT SUBROUTINE
84 ;=====
85 countDec
86     movf    count,F           ; Move the count value to the f register
87     btfsc   status,Z          ; Skip the next instruction if not zero, where the zero bit in the f register is now not 0
88     return                                ; Return if at zero as no more decrementing needed
89     decf    count,F           ; Otherwise decrement the counter by -1
90     return                                ; Return from subroutine to main loop
91
92 ;=====
93 ; DELAY SUBROUTINE - For button debounce
94 ;=====
95 delay
96     movlw   0xFF              ; Load maximum delay value of 255 in W register
97     movwf   dCount            ; Then move this value into the delay counter variable
98
99     delayLoop
100    decfsz  dCount,F           ; Decrement counter, skip the delay loop if this bit = 0
101    goto    delayLoop          ; Continue delay loop if the bit was not zero
102    return                                ; Return from delay routine to the main loop
103
104 ;=====
105 ; LOOKUP TABLE for LED patterns
106 ;=====
107 table
108     addwf   PCL               ; Add offset to program counter
109     retlw   0x40               ; 0
110     retlw   0x79               ; 1
111     retlw   0x24               ; 2
112     retlw   0x30               ; 3
113     retlw   0x19               ; 4
114     retlw   0x12               ; 5
115     retlw   0x02               ; 6
116     retlw   0x78               ; 7
117     retlw   0x00               ; 8
118     retlw   0x10               ; 9
119     end

```

3.4.2 Pseudocode

BEGIN SevenSegmentCounter program

Variables

DECLARE INTEGER count

DECLARE INTEGER previousState

DECLARE INTEGER delayCounter

Initialize ports

SET status_RP0 TO 1

SET port B direction TO 0x00 (output)

SET port A direction TO 0x0F (input)

SET status_RP0 TO 0

Set initial values

SET count TO 5

SET previousState TO 5

WRITE DisplayPatterns[5] TO port B (sets initial display pattern to 5)

Main program loop

WHILE TRUE

 READ port A TO W register

 XOR W register WITH previousState

 AND W register WITH 0x03 (mask to check only button bits)

 IF result = 0 THEN

 GOTO DisplayUpdate

 ENDIF

 SET previousState TO current port A value

 IF port A bit0 = 0

 THEN (check decrement button)

 CALL DecrementCounter

 ENDIF

 IF port A bit1 = 0

 THEN (check increment button)

 CALL IncrementCounter

 ENDIF

ENDWHILE

Display update subroutine

SUBROUTINE DisplayUpdate GET pattern FROM lookup_table[count]

WRITE pattern TO port B

CALL Debounce RETURN TO main loop

ENDSUBROUTINE

Increment counter subroutine

SUBROUTINE IncrementCounter

IF count = 9 THEN

RETURN

ENDIF

ADD 1 TO count

RETURN

ENDSUBROUTINE

```
Decrement counter subroutine
SUBROUTINE DecrementCounter
IF count = 0 THEN
RETURN ENDIF
SUBTRACT 1 FROM count
RETURN
ENDSUBROUTINE

Debounce delay subroutine
SUBROUTINE Debounce
SET delayCounter TO 255
WHILE delayCounter > 0 SUBTRACT 1 FROM delayCounter
ENDWHILE RETURN ENDSUBROUTINE

END SevenSegmentCounter
```

3.5 Piezo sound on button press

3.5.1 Assembly code

```

main.asm
1
2 ;=====
3 ; DEFINITIONS - define registers testing
4 ;=====
5
6 #include p16f84a.inc           ; Including the register definition file for the PIC16F84A
7
8 portA equ 0x05                 ; PortA address
9 trisa equ 0x05                 ; Trisa address in register bank 0
10
11 portB equ 0x06                 ; Port B address
12 trisb equ 0x06                 ; TrisB address in register bank 1
13
14 status equ 0x03                ; Status register
15 RP0 equ 5                     ; Bit 5 of status register, bank select register bit
16 ;=====
17 ; VARIABLES - MUST BE STORED IN RAM therefore these must point to a RAM memory address
18 ;=====
19 dcount equ 0x0D                ; Delay counter register
20
21 ;=====
22 ; RESET and INTERRUPT VECTORS
23 ;=====
24 RST code 0x0                  ; Reset vector for when a reset pulse is recieved
25 goto Start                    ; Restarts the program at the beginning of the main program
26 ;=====
27 ; CODE SEGMENT
28 ;=====
29 PGM code
30 Start                          ; Start of program
31 ; Port setup
32 bsf status,RP0                ; Select bank 1 and set the RP0 bit in the status register
33 movlw 0x00                    ; Load W with value 0x00, 0b00000000
34 movwf trisb                    ; Load it into trisB to set port B as all output pins
35
36 movlw 0x0F                    ; Load W with value 0x0F, 0b00001111
37 movwf trisa                    ; Load it into trisa to set the lower nibble of the trisa register as inputs
38 bcf status,RP0                ; Reselect bank 0, deselects bank 1 and reselects bank 0
39
40 clrf dcount                    ; Clear delay counter variable to 0
41
42 ; Main Loop
43 buttonCheck                    ; Button check Loop starts
44 movf portA, W                  ; Moves the port A value into the W register
45
46 btfsf portA, 0                 ; Bit testing pin A0 (port A 0 index), skips the next line if 1 (pressed)
47 call playTone                  ; Calls the subroutine to drive the sounder.
48
49 goto buttonCheck               ; Loop around again to continue checking for button interrupts, thereby repeating the tone if held
50
51 playTone
52 bsf portB, 7                   ; Set bit 7 (0indexed) of port B high, driving the sounder
53 call delay                     ; Delay of user determined time (by editing the value saved to count variable in delay subroutine)
54 bcf portB, 7                   ; Turn speaker off (corrected from RB0)
55 call delay                     ; Wait user determined time again
56 return                          ; Exit subroutine and jump back to buttonCheck
57 ;=====
58 ; Delay Subroutine, shortened version of previous pause subroutine
59 ; For implementing basic 50% duty cycle on the output signal
60 ;=====
61 delay
62 movlw 0x66                     ; Set W register to given value, changing this value changes the tone played
63 movwf dcount                    ; Set count = W
64
65 delayDec
66 decfsz dcount, F                ; Decrement the delay counter variable, skip if it has reached zero
67 goto delayDec                  ; Repeat previous decrement if counter variable is not zero
68 return                          ; Else exit subroutine jumps back playTone
69 end

```

3.5.2 Pseudocode

```

BEGIN SoundOnPress
DECLARE INTEGER delay_counter

```

```

Initialize ports
SET status_RP0 TO 1
SET port_b_direction TO 0x00
SET port_a_direction TO 0x0F
SET status_RP0 TO 0

// Clear initial values
SET delay_counter TO 0

// Main program loop
WHILE TRUE
    READ port_a TO working_register

    IF port_a_bit0 = 0 THEN
        CALL GenerateTone
    ENDIF
ENDWHILE

// Subroutine for generating square wave tone
SUBROUTINE GenerateTone
    SET port_b_bit7 TO 1
    CALL ToneDelay
    SET port_b_bit7 TO 0
    CALL ToneDelay
    RETURN
ENDSUBROUTINE

// Tone timing delay subroutine
SUBROUTINE ToneDelay
    SET delay_counter TO 0x66
    WHILE delay_counter > 0
        SUBTRACT 1 FROM delay_counter
    ENDWHILE
    RETURN
ENDSUBROUTINE

END SoundOnPress

```

1. Write a piece of code that sends a fixed frequency pulse train to drive the speaker on the demo board/proteus simulation. This pulse train should not have a fixed pulse width however - it should comprise of a constantly changing PWM waveform such as one would find on a music synthesiser with such a feature. See the following to illustrate the concept: PWM should be software based by bit bashing and then maybe hardware based- can replace it with the larger PIC suggested to be able to get the module needed.

References

- [1] Ariat Technology, "Understanding Switch Poles and Throws: An Analysis of Common Mechanical Switches," 12 01 2025. [Online]. Available: <https://www.ariat-tech.com/blog/understanding-switch-poles-and-throws-an-analysis-of-common-mechanical-switches.html>.
- [2] Setick, "Types of Light Switches," 13 01 2025. [Online]. Available: <https://www.setick.com/types-of-light-switches/>.
- [3] Texas Instruments, "µA7800 SERIES POSITIVE-VOLTAGE REGULATORS," 2004. [Online]. Available: <https://cdn.sparkfun.com/assets/1/7/7/3/2/LM7805.pdf>. [Accessed 12 01 2025].
- [4] Texas Instruments, "Engineers note: Capacitors are key to voltage regulator design," 2011. [Online]. Available: <https://www.ti.com/lit/wp/snoa842/snoa842.pdf?ts=1736721034909>. [Accessed 12 01 2025].
- [5] I. Poole, "7800 Series Voltage Regulators: 7805, 7812, 7824, etc," ElectronicsNotes, N/A. [Online]. Available: https://www.electronics-notes.com/articles/analogue_circuits/power-supply-electronics/7805-7812-voltage-regulator.php. [Accessed 12 01 2025].
- [6] W. H. Paul Horowitz, The Art of Electronics 3rd Edition, New York: Cambridge University Press, 2015.
- [7] Microchip Technology Inc., "PIC16F84A 18-pin Enhanced FLASH/EEPROM 8-Bit Microcontroller," 2001. [Online]. Available: <https://www.mouser.co.uk/datasheet/2/268/35007C-3442610.pdf>. [Accessed 12 01 2025].
- [8] Multicomp PRO, "NPN Silicon Planar Transistors BC107 / BC108 Series," 20 07 01. [Online]. Available: <https://www.farnell.com/datasheets/3094291.pdf>. [Accessed 12 01 2025].
- [9] WatElectronics, "BC107 Transistor : PinOut, Specifications, Circuit, Working, Datasheet, Equivalents & Its Applications," WatElectronics.com, 30 09 2024. [Online]. Available: <https://www.watelectronics.com/bc107-transistor/>. [Accessed 12 01 2025].
- [10] IBEX Technology Development Specialists, "Piezo Sounders," IBEX Technology Development Specialists, N/A. [Online]. Available: <https://ibex.tech/resources/geek-area/electronics/sounders/piezo-sounders>. [Accessed 12 01 2025].
- [11] RS Online, "How Do Piezo Buzzers Work?," RS Online, 30 10 2023. [Online]. Available: <https://uk.rs-online.com/web/content/discovery/ideas-and-advice/how-do-piezo-buzzers-work>. [Accessed 12 01 2025].
- [12] Electronics Tutorials, "Pull-up Resistors," Electronics Tutorials, N/A. [Online]. Available: <https://www.electronics-tutorials.ws/logic/pull-up-resistor.html>. [Accessed 12 01 2025].
- [13] EE Power, "Pull-up and Pull-down Resistors," EE Power, N/A. [Online]. Available: <https://eepower.com/resistor-guide/resistor-applications/pull-up-resistor-pull-down-resistor/#>. [Accessed 12 01 2025].
- [14] BITSMAshed, "Pull-up Resistors," SparkFun, N/A. [Online]. Available: <https://learn.sparkfun.com/tutorials/pull-up-resistors/all>. [Accessed 12 01 2025].
- [15] ECS INC International, "Crystal Oscillator Design Application Note," ECS INC International, N/A. [Online]. Available: <https://ecstxtal.com/crystal-oscillator-design-application-note/>. [Accessed 12 01 2025].
- [16] Electrical Technology, "BCD to 7-Segment Display Decoder – Construction, Circuit & Operation," Electrical Technology, N/A. [Online]. Available: <https://www.electricaltechnology.org/2018/05/bcd-to-7-segment-display-decoder.html>. [Accessed 12 01 2025].
- [17] Microchip, "Section 29. Instruction Set: PICmicro MID-RANGE MCU FAMILY Instruction Set," [Online]. Available:

https://moodle.uwtsd.ac.uk/pluginfile.php/983052/mod_resource/content/0/PIC%20instruction%20set.pdf.
[Accessed 12 01 2025].

- [18] WatElectronics, "PIC16F84A Microcontroller : PinOut, Specifications, Interfacing, Datasheet & Its Applications," WatElectronics.com, 18 06 2024. [Online]. Available: <https://www.watelectronics.com/pic16f84a-microcontroller/>. [Accessed 12 01 2025].
- [19] Melanie, "Thread: Presetting Configuration Fuses (PIC Defines) into your Program," Crownhill Associates, 12 07 2004. [Online]. Available: [https://www.picbasic.co.uk/forum/showthread.php/543-Presetting-Configuration-Fuses-\(PIC-Defines\)-into-your-Program](https://www.picbasic.co.uk/forum/showthread.php/543-Presetting-Configuration-Fuses-(PIC-Defines)-into-your-Program). [Accessed 12 01 2025].

FINAL GRADE

GENERAL COMMENTS

80/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

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

GRADING FORM: GJ_2019

ROWAN PATEY

402

ENGAGEMENT WITH LITERATURE



Excellent range of relevant sources cited.

87

KNOWLEDGE AND UNDERSTANDING

 Very detailed and accurate description of component operation **85**

COGNITIVE AND INTELLECTUAL

 An excellent level of thought demonstrated in terms of appropriate methods to efficiently solve the tasks. **77**
Pity the PWM task wasn't done.

PRACTICAL APPLICATION

 Good circuit and 5 out of the 6 tasks attempted, with detailed pseudocode provided. **75**

TRANSFERABLE SKILLS

 Well written and decently presented. **78**