

2322601 - Saffina H - Using and Programming the 7-Segment Display

by Saffina H

Submission date: 13-Jan-2025 09:34PM (UTC+0000)

Submission ID: 248334058

File name: 52210_Saffina_H_2322601_-_Saffina_H_-_Using_and_Programming_the_7-Segment_Display_621295_947060850.docx (1.93M)

Word count: 5422

Character count: 28272

Using and Programming the 7-Segment Display Prototype Circuit

Saffina H – 2322601

Micros, Peripherals and Interfaces – ACDA4004

Table of Contents

Table of Figures	3
1. Introduction	4
2. Count.asm Circuit.....	4
2.1. Proteus circuit	4
2.2. Component and Circuit Explanations.....	5
2.2.1. Specific elements	5
2.2.2. Resistors	6
2.2.3. Capacitors.....	7
3. Displaying Hexadecimal Characters	7
4. Modifying the Code	15
4.1. Incrementing Hexadecimal Characters	15
4.2. L and R Buttons	17
4.3. Latching LED segments.....	20
4.4. Counting up and down.....	23
4.5. Sounder	26
5. Conclusion.....	28
6. References.....	29
7. Appendices.....	30
Appendix A – Code for incrementing hexadecimal characters	30
Appendix B – Code for displaying L and R buttons	33
Appendix C – Code for latching the LED display on/off	36
Appendix D – Code to Increment/Decrement the Display	38
Appendix E – Code to Operate Sounder When Buttons are Pressed	42

Table of Figures

Figure 1: PIC16F84 assignment circuit diagram	4
Figure 2: Proteus simulation of count.asm circuit	5
Figure 3: Screenshot of assembly code to test input of LED display	8
Figure 4: Table of results from testing the 7 segment display inputs	10
Figure 5: Diagram to map each segment of the 7 segment display	10
Figure 6: Table showing results of mapping each characters binary code	13
Figure 7: Table with results of binary to hexadecimal conversions	14
Figure 8: Flow chart to increment and display hexadecimal characters	15
Figure 9: Code to show the key functions of the incrementing and displaying hexadecimal task	16
Figure 10: The lookup table created for the hexadecimal characters on the 7 segment display .	16
Figure 11: Screenshot of Proteus simulating the incrementing hexadecimal code	17
Figure 12: Flow chart to display L and R when buttons are pressed	18
Figure 13: Code for the L and R buttons	18
Figure 14: Screenshot of simulation displaying 'L' when the left button is pressed.....	19
Figure 15: Screenshot of simulation displaying 'r' when the right button is pressed.....	19
Figure 16: Flow chart to create latching buttons to turn LEDs on/off	20
Figure 17: Code of the main function to check status of the 7-segment display and buttons.....	21
Figure 18: Screenshot of the 7-segment display being turned off	22
Figure 19: Screenshot of the 7-segment display being turned on.....	22
Figure 20: Flow chart for incrementing or decrementing the 7 segment display	23
Figure 21: Code showing the main function of the counting program.....	24
Figure 22: Code showing subroutines of the counting program, run when a button is pressed .	24
Figure 23: Code for the incrementing and decrementing subroutines for the counting program	25
Figure 24: Screenshot showing the working counting circuit.....	25
Figure 25: Flow chart to work sounder when a button is pressed	26
Figure 26: Code for the main function of the sounder program	27
Figure 27: Code for the two key subroutines to turn the sounder on and off	27
Figure 28: Screenshot of the operating sounder program and circuit	28

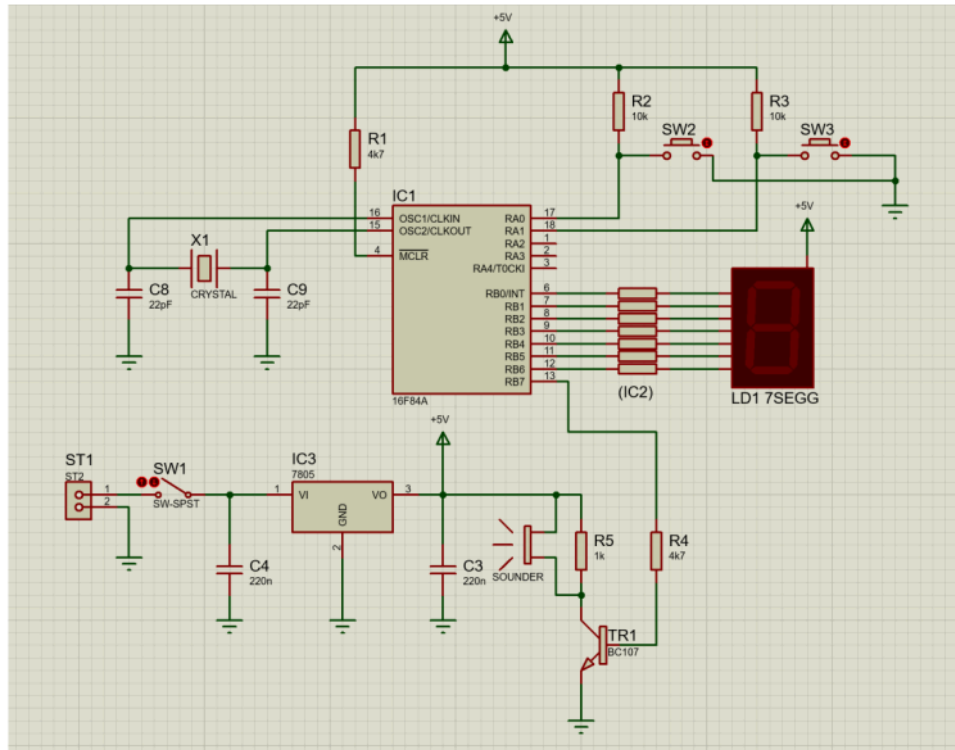


Figure 2: Proteus simulation of count.asm circuit

In later simulations, the connector ST1 is replaced with a +10V power supply, as in a simulated circuit, a power supply is more easily simulated than a terminal for a battery, while producing the same results.

2.2. Component and Circuit Explanations

2.2.1. Specific elements

IC1 – IC1 is a PIC14F84A, which has a total of 13 I/O pins and a RISC CPU [1]. It is the central chip to the circuit, and is used to drive the LED display LD1, while receiving inputs from pin A0 and A1, and external clock pulses from the crystal oscillator.

IC3 – IC3 is a 7805 voltage regulator. It receives an input from the power supply of around +10V, and outputs a voltage of +5V, which acts as an input voltage to operate the rest of the PIC circuit [2].

LD1 – LD1 is the 7-segment display. This receives 7 digital inputs from the output pins B0 to B6 of IC1, with each input corresponding to an LED segment on the display [3]. This allows the processes of the chip to more easily be visualized, as they can be formatted in a user-friendly output.

Sounder – The sounder is a method of output for IC1, the main PIC. This allows the programmer to provide an audio output to the user. The sounder will create noise when the output pin B7 is oscillating/turning on and off repeatedly to create a PWM input to the sounder [4].

TR1 – This is the transistor which controls the sounder. It is a BJT transistor and will switch on when IC1's output pin B7 is high, causing current to flow from the base to the emitter, which then causes current to flow from the collector to the emitter [5], when turned on and off repeatedly, the sounder will create noise.

SW1 – This switch is used as the overall power switch, to turn the full circuit on/off. When it is closed, the terminal to the power supply is connected.

SW2/SW3 – Switches/buttons two and three are both inputs to IC1, connected to pin A0 and A1 respectively. When pressed, they will set these input pins to low, as they create a path to ground for current to flow. When released, the input pins will be set to high.

X1 – This is the crystal, which provides an external clock input to IC1, connected to both CLKIN (clock in) and CLKOUT (clock out) pins. The crystal resonates at a fixed frequency, which provides a reliable and consistent clock cycle to the PIC. This crystal resonates at 4MHz.

ST1/2 – ST1 and ST2 represent a connector which would be connected to the power supply. This provides the voltage input to IC3, which subsequently provides the +5V used by the PIC and peripherals.

2.2.2. Resistors

R1 – As the MCLR (master clear) pin to IC1 is active-low, to prevent resetting the chip, R1 acts as a pull-up resistor, to hold this input pins high, as the chip will not need to be re-programmed when in operation [1].

R2/R3 – Both of these resistors, connected to SW2 and SW3 respectively, act as pull-up resistors. This sets the input pins A0 and A1 high when the switch is off. This also limits the current through the switch when on, preventing a short circuit through the switch [6].

R4 – Resistor 4 is connected to the base of TR1 and limits the current going to the base. This allows the transistor to operate at its required current and means it will not blow when the output pin B7 of IC1 is high and sourcing current [5].

R5 – R5 is in parallel to the piezoelectric buzzer, this allows the buzzer to charge/discharge as it has capacitive elements, and to reset the voltage. It will also control the power being provided to it, as the resistor will dissipate some of this power [7].

Resistor array (IC2) – IC2 is an array of 7 resistors, each connected to one of the LED inputs to LD1. As the output voltage of IC1 will be set at 5V, the resistor acts as a current limiting to allow the LED to operate in its optimal current rating range [6].

2.2.3. Capacitors

C3 – C3 is used as the output capacitor for the voltage regulator IC3. While it does help a little with stability, its main purpose is to help the transient response of the regulator [2].

C4 – Capacitor C4 is used as a filter for the input voltage to the voltage regulator IC3. It is used to provide a steadier input and reduce ripples. It helps with the stability of the voltage regulator and its' transient response [2].

C8/C9 – These capacitors act as pull-down capacitors on either side of the crystal oscillator. They are used to improve stability for the crystal oscillator and their values are recommended to be between 15-33pF by the PIC's datasheet [1].

3. Displaying Hexadecimal Characters

To identify the outputs needed to display each hexadecimal character, Each bit of the 7-segment display was tested individually, using the code shown below in Figure 3.

```

movlw b'1000000'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
movlw b'0100000'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause
movlw b'0010000'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause
movlw b'0001000'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause
movlw b'0000100'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause
movlw b'0000010'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause
movlw b'0000001'
movwf portB           ; move W to port B to update the display
call pause            ; delay 200ms by subroutine
call pause

```

Figure 3: Screenshot of assembly code to test input of LED display

The results of this test are summarized in Figure 4 below, along with the LED segment they map to.

Binary code	Seven-segment display output	Mapping the bit to display
1000000		1 st bit = segment G

0100000	RA0 #17 RA1 #18 RA2 #1 RA3 #2 RA4/T0CKI #3 RB0/INT #6 RB1 #7 RB2 #8 RB3 #9 RB4 #10 RB5 #11 RB6 #12 RB7 #13 R9 220R 220R 220R 220R 220R 220R 220R	2nd bit = segment F
0010000	RA0 #17 RA1 #18 RA2 #1 RA3 #2 RA4/T0CKI #3 RB0/INT #6 RB1 #7 RB2 #8 RB3 #9 RB4 #10 RB5 #11 RB6 #12 RB7 #13 R9 220R 220R 220R 220R 220R 220R 220R	3rd bit = segment E
0001000	RA0 #17 RA1 #18 RA2 #1 RA3 #2 RA4/T0CKI #3 RB0/INT #6 RB1 #7 RB2 #8 RB3 #9 RB4 #10 RB5 #11 RB6 #12 RB7 #13 R9 220R 220R 220R 220R 220R 220R 220R	4th bit = segment D
0000100	RA0 #17 RA1 #18 RA2 #1 RA3 #2 RA4/T0CKI #3 RB0/INT #6 RB1 #7 RB2 #8 RB3 #9 RB4 #10 RB5 #11 RB6 #12 RB7 #13 R9 220R 220R 220R 220R 220R 220R 220R	5th bit = segment C

0000010		6th bit = segment B
0000001		7th bit = segment A

Figure 4: Table of results from testing the 7 segment display inputs

This showed that the 7-segment display is active low, meaning that when one of the output pins B0-B6 is set low, the corresponding LED will light. This made it easier to map each segment to its bit in the binary output. For this, Figure 5 below was used.

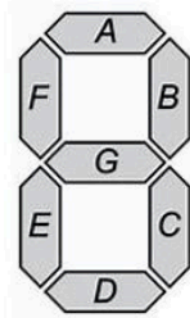
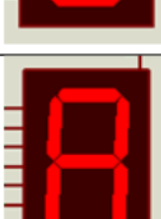


Figure 5: Diagram to map each segment of the 7 segment display

The bits which needed to be low for each hexadecimal character was then identified, and the overall binary output values were calculated in Figure 6 below.

Hexadecimal Character	ON segments	OFF segments	Binary output values	Sum	Display output
0	A B C D E F	G	1000000	1000000	
1	B C	A D E F G	0000001 0001000 0010000 0100000 1000000	1111001	
2	A B D E G	C F	0000100 0100000	0100100	
3	A B C D G	E F	0010000 0100000	0110000	
4	B C F G	A D E	0000001 0001000 0010000	0011001	

5	A C D F G	B E	0000010 0010000	0010010	
6	A C D E F G	B	0000010	0000010	
7	A B C F	D E G	0001000 0010000 1000000	1011000	
8	A B C D E F G		None	0000000	
9	A B C D F G	E	0010000	0010000	
A	A B C E F G	D	0001000	0001000	

B	C D E F G	A B	0000001 0000010	0000011	
C	A D E F	B C G	0000010 0000100 1000000	1000110	
D	B C D E G	A F	0000001 0100000	0100001	
E	A D E F G	B C	0000010 0000100	0000110	
F	A E F G	B C D	0000010 0000100 0001000	0001110	

Figure 6: Table showing results of mapping each characters binary code

To test each of these codes, each binary output number was converted to hexadecimal using the method below:

- Separate the binary number into two nibbles
- Find the denary value of each of these nibbles individually
- Convert each denary value to the relevant hexadecimal character

→ Combine both hexadecimal characters together to obtain the final value [8]

An example calculation for the character '7' is shown below:

8-bit binary code = 01011000

Bit values = 8 4 2 1 8 4 2 1

Nibbles = 0 1 0 1 1 0 0 0

Denary values = 4+1 = 5 8 = 8

Hexadecimal value = 58

Equation 1: Converting the binary code to hexadecimal

The rest of the characters' hexadecimal values were calculated using the same method, and the results of each calculation are shown in Figure 7 below.

Hexadecimal Value	Binary output code	Hexadecimal conversion
0	01000000	40
1	01111001	79
2	00100100	24
3	00110000	30
4	00011001	19
5	00010010	12
6	00000010	02
7	01011000	58
8	00000000	00
9	00010000	10
A	00001000	08
B	00000011	03
C	01000110	46
D	00100001	21
E	00000110	06
F	00001110	0E

Figure 7: Table with results of binary to hexadecimal conversions

4. Modifying the Code

4.1. Incrementing Hexadecimal Characters

The flow chart for hexadecimal character display was made, shown below in Figure 8, it was designed before the program to plan the basic logical operation of the code.

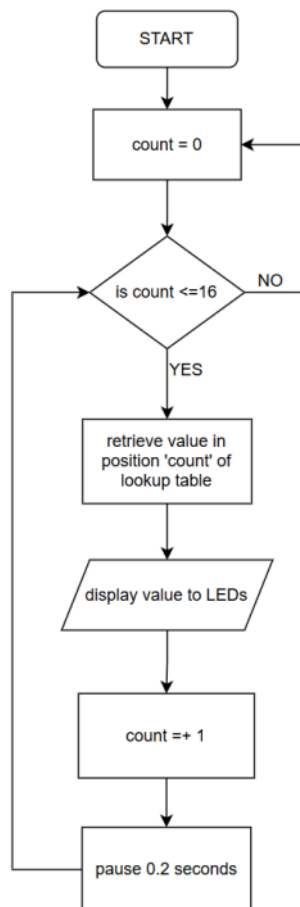


Figure 8: Flow chart to increment and display hexadecimal characters

Using the initial Proteus simulation from section 2.1, the original conut.asm program and along the hexadecimal output codes calculated previously, the following code shown in Figures 9 and 10 was written to display each hexadecimal character.

```

59 get_count
60
61     movf    count, w        ; move count to W w= workign register
62
63
64     btfss   count, 4        ;testing if msb of count is >16 (if the 16 bit is high)
65     call    increment_count ;if it isnt then increment count
66
67
68     call    pause           ;delay 200ms by subroutine
69
70     movlw   0x00            ;resets count to 0 if it > 16
71     movwf   count
72
73     goto    get_count       ; repeat forever
74
75
76
77 increment_count
78     movf    count, w        ;moves count to w
79     call    number_table    ;retrieves display code for character from lookup table
80     movwf   PORTB          ;updates display
81     call    pause
82     incf    count           ;increments count by one
83
84     goto    get_count       ;repeats forever
85

```

Figure 9: Code to show the key functions of the incrementing and displaying hexadecimal task

```

89 number_table
90
91     addwf   PCL
92
93     retlw   0x40            ;0
94     retlw   0x79            ;1
95     retlw   0x24            ;2
96     retlw   0x30            ;3
97     retlw   0x19            ;4
98     retlw   0x12            ;5
99     retlw   0x02            ;6
100    retlw   0x58            ;7
101    retlw   0x00            ;8
102    retlw   0x10            ;9
103    retlw   0x08            ;A
104    retlw   0x03            ;B
105    retlw   0x46            ;C
106    retlw   0x21            ;D
107    retlw   0x06            ;E
108    retlw   0x0e            ;F

```

Figure 10: The lookup table created for the hexadecimal characters on the 7 segment display

A lookup table was used to cleanly loop in the code, which will retrieve the hexadecimal output value for a certain character. The value is then outputted to port B, which sets pins B0 to B7 high/low, depending on each pin's corresponding bit in the binary value being outputted. The

pause subroutine is then called twice to create a delay of 0.2s between each character being outputted.

The full code can be found in Appendix A, and an image of the circuit operating in the simulation is also shown below in Figure 11

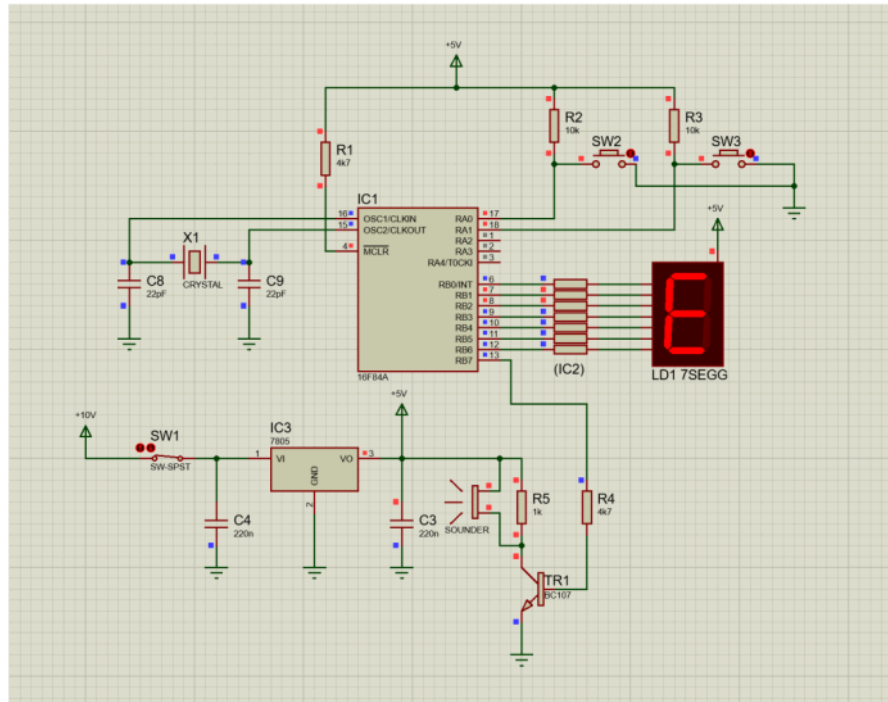


Figure 11: Screenshot of Proteus simulating the incrementing hexadecimal code

4.2. L and R Buttons

The flow chart for the L and R buttons was made, shown below in figure ***, it was designed before the program to plan the basic logical operation of the program.

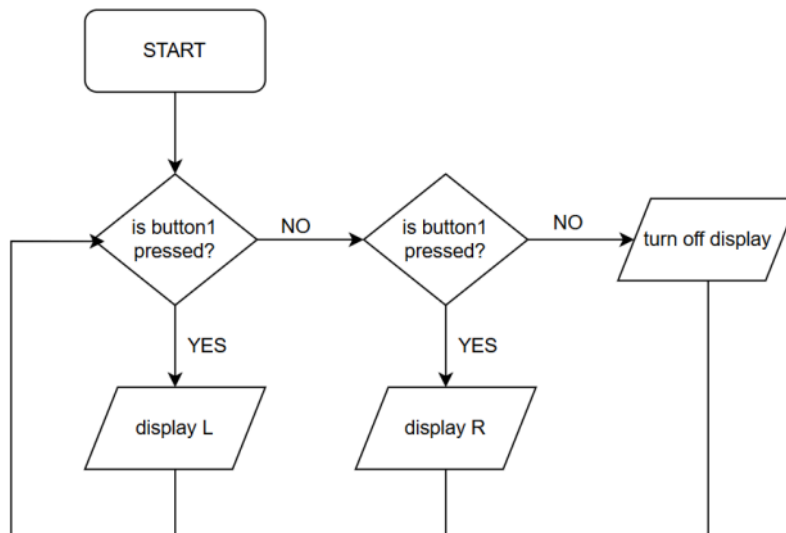


Figure 12: Flow chart to display L and R when buttons are pressed

The main program was written, shown below in Figure 13, the full program can be found in Appendix B.

```

67 button_check
68
69         ; Set RA0 as input
70         bsf TRISA, 0      ; Set RA0 (bit 0) as an input
71         bsf TRISA, 1      ; Set RA1 (bit 1) as an input
72
73
74         ; Check if pin RA0 is high
75         btfss PORTA, 0    ; Bit Test f, Skip if Set (Check if RA0 is high/first button pressed)
76         goto pin0_low    ; If RA0 is Low, jump to pin_low
77         goto pin0_high
78
79
80 pin0_low: ;will run if pin RA0 is Low
81         movlw 0x47        ;L
82         movwf portB       ;move W to port B to update the display
83         goto button_check ;Loops back infinitely
84
85
86 pin0_high: ;will skip to this line if PIN RA0 is high
87         btfss PORTA, 1    ; Bit Test f, Skip if Set (Check if RA1 is high/second button pressed)
88         goto pin1_low    ; If RA1 is Low, jump to pin_low subrouting
89         goto pin1_high   ; skips to this line if RA1 is high, jump to pin1_high subroutine
90
91
92 pin1_high: ;turns display off if neither button is pressed
93         movlw 0xff        ; move W to port B to update the display
94         movwf portB
95         goto button_check
96
97 pin1_low: ;r
98         movlw 0x2f        ; move W to port B to update the display
99         movwf portB

```

Figure 13: Code for the L and R buttons

This code infinitely checks the status of each button, calling for the display to show L, R or nothing depending on which buttons are pressed. The hexadecimal codes for L and R were calculated in the same way as the hexadecimal characters were in section 3. The full code can be found in Appendix B, and screenshots of the working simulation are shown below in Figures 14 and 15.

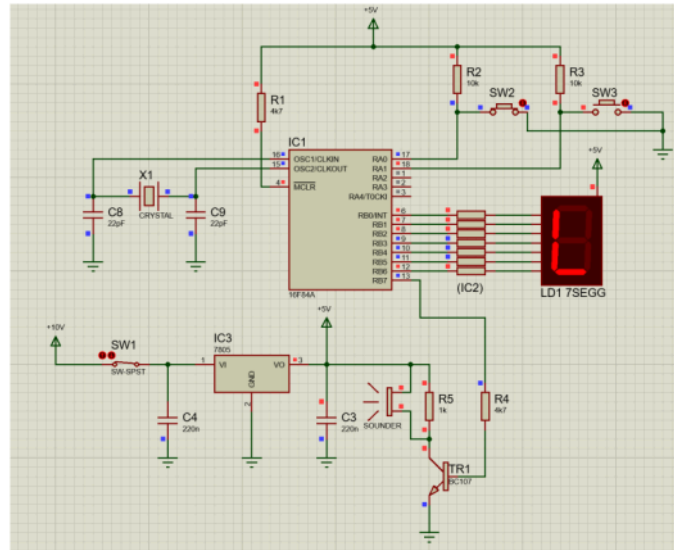


Figure 14: Screenshot of simulation displaying 'L' when the left button is pressed

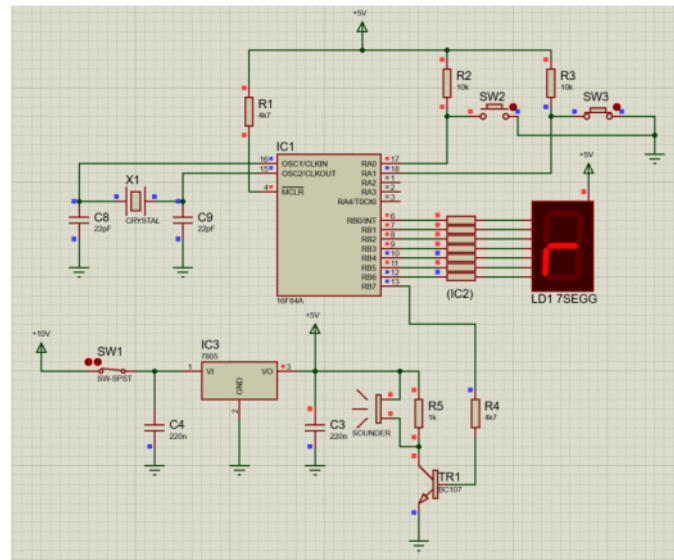


Figure 15: Screenshot of simulation displaying 'r' when the right button is pressed

4.3. Latching LED segments

The flow chart for the latching LEDs was made, shown below in Figure 16 it was designed before the program to plan the basic logical operation of the program.

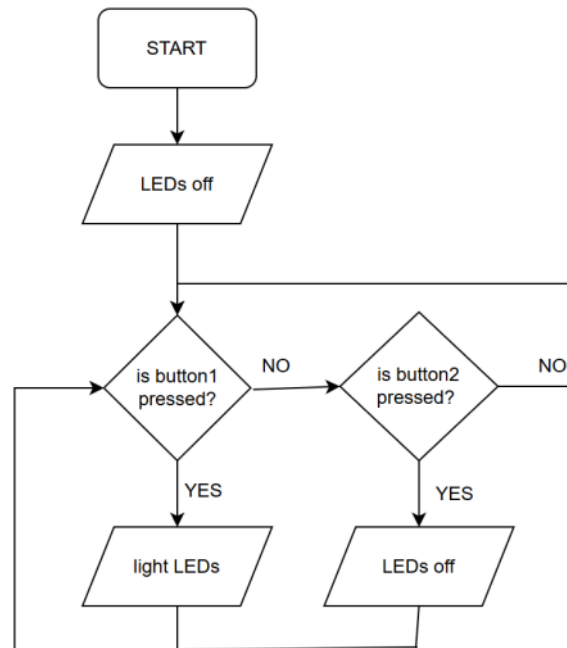


Figure 16: Flow chart to create latching buttons to turn LEDs on/off

The code for this program was then written, the assembly code for the key function is shown below in Figure 17.

```

68 button_check
69
70     bsf TRISA, 0      ; Set RA0 (bit 0) as an input
71     bsf TRISA, 1      ; Set RA1 (bit 1) as an input
72
73     btfss PORTB, 0     ;checks if the LED display is already on
74     goto buttonA       ;runs if the display is on
75     goto continue      ;jumps to next section
76
77     buttonA:           ;will run if the LED display is currently off
78     btfss PORTA, 0     ;checks if button one is pressed to turn display off
79     goto turn_off      ;turns off LEDs
80     goto button_check  ;repeats main function to continue checking
81
82     turn_off:
83     bsf PORTB, 0       ;sets all bits in portB high (turns LEDs off)
84     bsf PORTB, 1
85     bsf PORTB, 2
86     bsf PORTB, 3
87     bsf PORTB, 4
88     bsf PORTB, 5
89     bsf PORTB, 6
90     goto button_check  ;repeats main function
91
92     continue:          ;will run if the LED display is currently off
93     btfss PORTA, 1     ;checks if button two is pressed
94     goto turn_on       ;turns on LEDs
95     goto button_check  ;repeats main function
96
97     turn_on:
98     clrf PORTB         ;clears all pins on port B
99     goto button_check

```

Figure 17: Code of the main function to check status of the 7-segment display and buttons

The program will first check if the LED is on or off, if it's on, it will check if button one has been pressed and turn off if needed. When it is off, it will check button 2 and turn on when it is pressed. The full code can be found in Appendix C, and screenshots of the buttons latching and turning the display on and off can be seen below in Figures 18 and 19.

Figure 18: Screenshot of the 7-segment display being turned off

Figure 19: Screenshot of the 7-segment display being turned on

4.4. Counting up and down

The flow chart for the incrementing/decrementing project was made, shown below in Figure 20, it was designed before the program to plan the basic logical operation of the program.

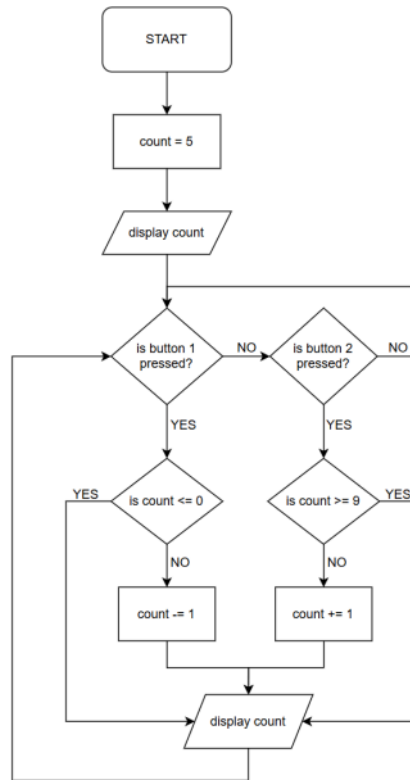


Figure 20: Flow chart for incrementing or decrementing the 7 segment display

The key bits of code and functions called can be found below in Figures 21 and 22. The lookup table created in section 4.1 was also used.

```

45 start
46     bcf     status,RP0      ; clear bank selection bits in the status reg
47     bcf     status,RP1
48
49     bsf     status,RP0      ; select bank 1
50     movlw   0x00            ; load w with 0x00
51     movwf   trisb           ; load it into trisb to make portb all outputs
52     bcf     status,RP0      ; reselect bank 0
53
54     clrf    portB           ; clear all lines low
55
56     movlw   0x05            ; initially sets count to 5
57     movwf   count           ; retrieves hex code to display 5 on the display
58     call    number_table    ; initially displays count
59     movwf   PORTB
60
61
62
63 button_check                    ;infinitely checks if buttons are pressed
64
65     bsf     TRISA, 0        ; Set RA0 (bit 0) as an input
66     bsf     TRISA, 1        ; Set RA1 (bit 1) as an input
67
68
69     btfss   PORTA, 0        ;checks if button 1 has been pressed
70     call    button_one
71
72     btfss   PORTA, 1        ;checks if button 2 has been pressed
73     call    button_two
74
75     goto    button_check    ; repeat forever

```

Figure 21: Code showing the main function of the counting program

```

80 button_one                    ;code to run when button 1 is pressed
81     decfsz  count           ;decrements count and tests if its zero
82     call    count_dec       ;update count and display
83
84     movlw   0x00            ;sets count to 0
85     movwf   count           ;finds hex code for 0 in Lookup table
86     call    number_table    ;outputs to display
87     movwf   PORTB
88
89     goto    button_check    ;returns to main program
90
91
92
93
94 button_two                    ;code to run when button 2 is pressed
95     btfss   count, 3        ;tests if count >= 8 by testing the '8's bit of the register
96     call    count_inc       ;update count and display
97
98     movlw   0x09            ;sets count to 9
99     movwf   count           ;returns hex code for 9 from Lookup table
100    call    number_table    ;sends hex code to display
101    movwf   PORTB
102
103    goto    button_check    ;returns to main program

```

Figure 22: Code showing key subroutines of the counting program, run when either button is pressed

```

108 count_inc                                ;code to increment and display count
109     incf    count                        ;count+=1
110     movf    count, w                     ;moves count to w
111     call    number_table                 ;returns hex code in position of count from lookup table
112     movwf   PORTB                        ;sends hex code to display
113
114     call    pause                         ;pause for 200ms
115
116     goto    button_check                 ;returns to main program
117
118
119
120
121 count_dec                                ;subroutine to decrement and display count
122     movf    count, w                     ;moves value of count to w
123     call    number_table                 ;returns hex code in position of count from lookup table
124     movwf   PORTB                        ;outputs that hex number
125
126     call    pause                         ;pauses for 200ms
127
128     goto    button_check                 ;returns to main program

```

Figure 23: Code for the incrementing and decrementing subroutines for the counting program

The program works by first looping until one of the buttons are pressed. When they are, it then checks the value of 'count' if it is already at nine, it will continue to be set at nine if it tries to increment, similarly it will remain at 0 if it tries to decrement further. In any other situation, it will increment/decrement as needed and update the display. The full code can be found in Appendix D, and a screenshot of the operating circuit is shown in Figure 24 below.

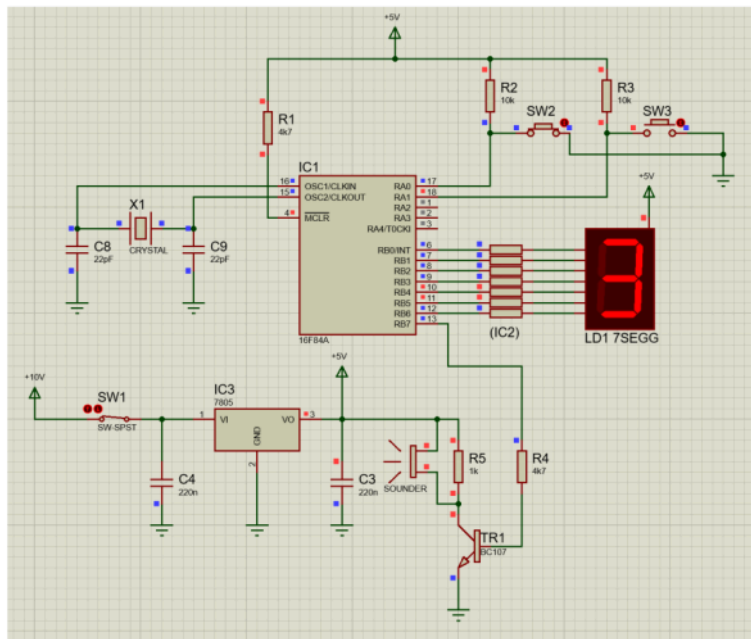


Figure 24: Screenshot showing the working counting circuit

4.5. Sounder

As an extension task, a flow chart for sounder to buzz was made, shown below in Figure 25 it was designed before the program to plan the basic logical operation of the program.

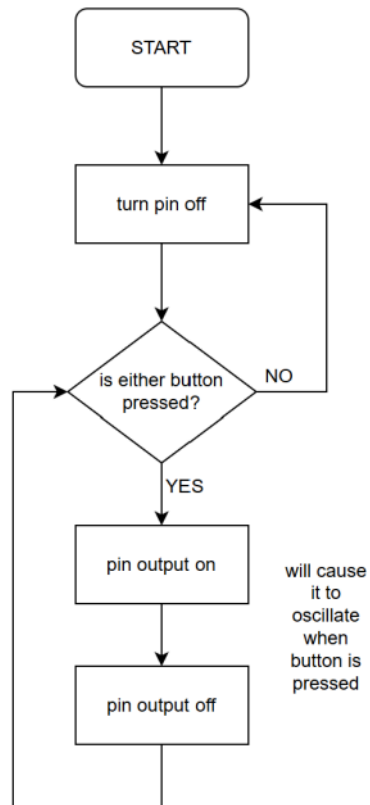


Figure 25: Flow chart to work sounder when a button is pressed

The program was implemented using the key functions shown below in Figures 26 and 27.

```

59 button_check                                ;infinitely checks if buttons are pressed
60
61     bsf    TRISA, 0                        ; Set RA0 (bit 0) as an input
62     bsf    TRISA, 1                        ; Set RA1 (bit 1) as an input
63
64
65     btfss  PORTA, 0                        ;checks if button 1 has been pressed
66     call   turn_on                        ;turns sounder on
67
68     btfss  PORTA, 1                        ;checks if button 2 has been pressed
69     call   turn_on                        ;turns sounder on
70
71
72     goto   button_check                    ; repeat forever

```

Figure 26: Code for the main function of the sounder program

```

77 turn_on
78     bsf    PORTB, 7                        ;sets pin B7 high
79
80     btfss  PORTA, 0                        ;checks if button one has been released
81     call   turn_off                       ;turns off sonuder
82
83     btfss  PORTA, 1                        ;checks if button two has been released
84     call   turn_off                       ;turns off sounder
85
86     bcf    PORTB, 7                        ;sets pin B7 Low
87     nop                                     ;creates even pwm cycle
88
89     goto   turn_on                        ;repeats the loop while buttons being held
90
91
92
93 turn_off
94     bcf    PORTB, 7                        ;turns off pin B7
95     goto   button_check                   ;returns to main program

```

Figure 27: Code for the two key subroutines to turn the sounder on and off

The buttons were checked in a very similar way to the previous 3 tasks, when either one was pressed, the program then jumps to the subroutine to turn the sounder on by creating a PWM output. It will then check when the buttons have been released and turn the sounder off. The full code can be found in Appendix E, and a screenshot of the sounder operating is shown below in Figure 28.

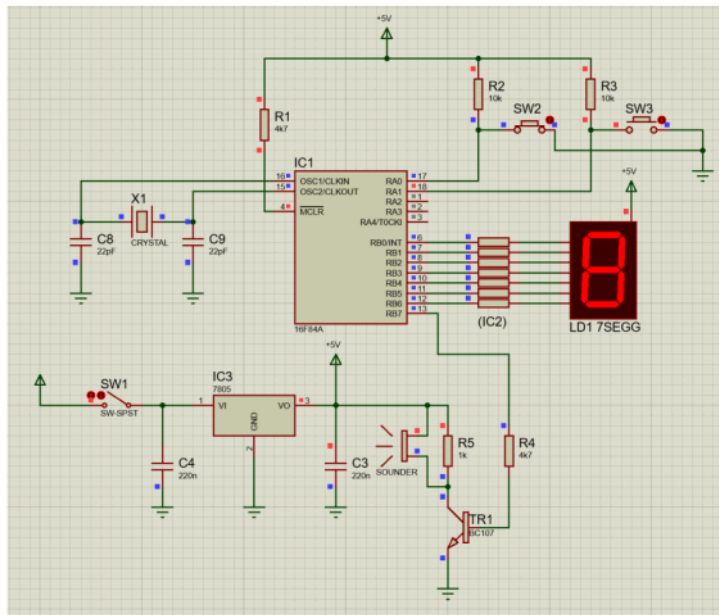


Figure 28: Screenshot of the operating sounder program and circuit

5. Conclusion

To summarize, this report has covered the function and operation of the assignment circuit, looking at each chip's uses and all the components' purposes in the circuit. It has also explained how to display hexadecimal characters to a 7-segment display, and the method used to test and calculate each output needed. The report concludes by going through 5 of the set tasks, their operation and snapshot of the code and its working circuit.

6. References

- [1] Microchip, *PIC16F84A Data Sheet*, 2001, pp. 1-1.
- [2] Texas instruments, *LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators*, 2016, p. 6.
- [3] Geeksforgeeks, "Seven Segment Displays," 16 08 2024. [Online]. Available: <https://www.geeksforgeeks.org/seven-segment-displays/>. [Accessed 13 01 2025].
- [4] M. Bilal, "How to Generate Sound using PWM with PIC Microcontroller- (Part 22/25)," PIC Microcontroller, 28 05 2020. [Online]. Available: <https://pic-microcontroller.com/how-to-generate-sound-using-pwm-with-pic-microcontroller-part-22-25/>. [Accessed 13 01 2025].
- [5] P. Horowitz and W. Hill, "The Art Of Electronics 3rd Edition," Cambridge University Press, 2015, pp. 71-73.
- [6] P. Horowitz and W. Hill, "The Art of Electronics 3rd Edition," Cambridge University Press, 2015, pp. 3-7.
- [7] CUI inc, *Buzzer Basics - technologies, Tones, and Drive Circuits*, 2019.

7. Appendices

Appendix A – Code for incrementing hexadecimal characters

```
;=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc                ; Include register definition file

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

status   equ 0x03                  ;status register    number is in hex
RP0 equ 5                          ;bit 5 of status register assigns memory address as
register
RP1      equ 6                    ;bit 6 of status register

;=====
; VARIABLES
;=====

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

;=====
; RESET and INTERRUPT VECTORS
;=====

        ; Reset Vector
RST      code 0x0
        goto start

;=====
; CODE SEGMENT
;=====

PGM      code

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

```

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

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

get_count

    movf     count, w      ; move count to W w= workign register

    btfss    count, 4      ;testing if msb of count is >16 (if the 16 bit is
high)
    call     increment_count ;if it isnt then increment count

    call     pause         ;delay 200ms by subroutine

    movlw    0x00          ;resets count to 0 if it > 16
    movwf    count

    goto     get_count     ; repeat forever

increment_count
    movf     count, w      ;moves count to w
    call     number_table  ;retrieves display code for character from
lookup table
    movwf    PORTB         ;updates display
    call     pause         ;
    incf     count         ;increments count by one

    goto     get_count     ;repeats forever

;=====

number_table

    addwf    PCL

    retlw    0x40          ;0
    retlw    0x79          ;1
    retlw    0x24          ;2
    retlw    0x30          ;3
    retlw    0x19          ;4
    retlw    0x12          ;5
    retlw    0x02          ;6
    retlw    0x58          ;7
    retlw    0x00          ;8
    retlw    0x10          ;9
    retlw    0x08          ;A

```

```

    retlw    0x03        ;B
    retlw    0x46        ;C
    retlw    0x21        ;D
    retlw    0x06        ;E
    retlw    0x0e        ;F

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

```

Appendix B – Code for displaying L and R buttons

```
=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include pl16f84a.inc                ; Include register definition file

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

portA    equ 0x11                    ;port A address
trisa    equ 0x11                    ;tris a address

status   equ 0x03                    ;status register    number is in hex
RP0 equ 5                            ;bit 5 of status register assigns memory address as
register
RP1      equ 6                        ;bit 6 of status register

;=====
; VARIABLES
;=====

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

;=====
; RESET and INTERRUPT VECTORS
;=====

; Reset Vector
RST      code 0x0
        goto start

;=====
; CODE SEGMENT
;=====

PGM      code

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

    bsf      status,RP0      ; select bank 1
    movlw    0xFF            ; load w with 0x03
    movwf    trisa           ; load it into trisa to set input pins
    bcf      status,RP0      ; reselect bank 0

    clrf     portB           ; clear all lines low

button_check

    ; Set RA0 as input
    bsf      TRISA, 0        ; Set RA0 (bit 0) as an input
    bsf      TRISA, 1        ; Set RA1 (bit 1) as an input

    ; Check if pin RA0 is high
    btfss    PORTA, 0        ; Bit Test f, Skip if Set (Check if RA0
is high/first button pressed)
    goto     pin0_low        ; If RA0 is low, jump to pin_low
    goto     pin0_high

    pin0_low:                ;will run if pin RA0 is low
    movlw    0x47            ;L
    movwf    portB           ;move W to port B to update the display
    goto     button_check    ;loops back infinitely

    pin0_high:               ;will skip to this line if PIN RA0 is high
    btfss    PORTA, 1        ; Bit Test f, Skip if Set (Check if RA1 is
high/second button pressed)
    goto     pinl_low        ; If RA1 is low, jump to pin_low subrouting
    goto     pinl_high       ; skips to this line if RA1 is high, jump to
pinl_high subroutine

    pinl_high:
    movlw    0xff            ;turns display off if neither button is
pressed
    movwf    portB           ; move W to port B to update the display
    goto     button_check

    pinl_low:
    movlw    0x2f            ;r
    movwf    portB           ; move W to port B to update the display

    goto     button_check

```

;

END

Appendix C – Code for latching the LED display on/off

```
=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include pl16f84a.inc                ; Include register definition file

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

portA    equ 0x11                    ;port A address
trisa    equ 0x11                    ;tris a address

status   equ    0x03                ;status register    number is in hex
RP0 equ    5                        ;bit 5 of status register assigns memory address as
register
RP1      equ 6                      ;bit 6 of status register

;=====
; VARIABLES
;=====

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

;=====
; RESET and INTERRUPT VECTORS
;=====

; Reset Vector
RST      code 0x0
        goto start

;=====
; CODE SEGMENT
;=====

PGM      code

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

    bsf      status,RP0    ; select bank 1
    movlw    0xFF          ; load w with 0xFF
    movwf    trisa         ; load it into trisa to set input pins
    bcf      status,RP0    ; reselect bank 0

    clrf     portB         ; clear all lines low

button_check

    bsf      TRISA, 0      ; Set RA0 (bit 0) as an input
    bsf      TRISA, 1      ; Set RA1 (bit 1) as an input

    btfss    PORTB, 0      ;checks if the LED display is already on
    goto     buttonA       ;runs if the display is on
    goto     continue      ;jumps to next section

    buttonA:               ;will run if the LED display is currently off
    btfss    PORTA, 0      ;checks if button one is pressed to turn display
off
    goto     turn_off      ;turns off LEDs
    goto     button_check  ;repeats main function to continue checking

    turn_off:
    bsf      PORTB, 0      ;sets all bits in portB high (turns LEDs off)
    bsf      PORTB, 1
    bsf      PORTB, 2
    bsf      PORTB, 3
    bsf      PORTB, 4
    bsf      PORTB, 5
    bsf      PORTB, 6
    goto     button_check  ;repeats main function

    continue:              ;will run if the LED display is currently off
    btfss    PORTA, 1      ;checks if button two is pressed
    goto     turn_on       ;turns on LEDs
    goto     button_check  ;repeats main function

    turn_on:
    clrf     PORTB         ;clears all pins on port B
    goto     button_check

;=====

    END

```

Appendix D – Code to Increment/Decrement the Display

```
=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
=====

;=====
; DEFINITIONS
;=====

#include pl16f84a.inc                ; Include register definition file

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

status   equ 0x03                    ;status register    number is in hex
RP0      equ 5                      ;bit 5 of status register assigns memory address as
register
RP1      equ 6                      ;bit 6 of status register

;=====
; VARIABLES
;=====

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

;=====
; RESET and INTERRUPT VECTORS
;=====

        ; Reset Vector
RST      code 0x0
        goto start

;=====
; CODE SEGMENT
;=====

PGM      code

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   0x05
    movwf   count          ;initially sets count to 5
    call    number_table    ;retrieves hex code to display 5 on the display
    movwf   PORTB          ;initially displays count


button_check                ;infinitely checks if buttons are pressed

    bsf     TRISA, 0        ; Set RA0 (bit 0) as an input
    bsf     TRISA, 1        ; Set RA1 (bit 1) as an input


    btfss   PORTA, 0        ;checks if button 1 has been pressed
    call    button_one

    btfss   PORTA, 1        ;checks if button 2 has been pressed
    call    button_two

    goto    button_check    ; repeat forever


button_one                  ;code to run when button 1 is pressed
    decfsz  count          ;decrements count and tests if its zero
    call    count_dec      ;update count and display

    movlw   0x00
    movwf   count          ;sets count to 0
    call    number_table    ;finds hex code for 0 in lookup table
    movwf   PORTB          ;outputs to display

    goto    button_check    ;returns to main program


button_two                  ;code to run when button 2 is pressed
    btfss   count, 3        ;tests if count >= 8 by testing the '8's bit
of the register
    call    count_inc      ;update count and display

    movlw   0x09
    movwf   count          ;sets count to 9
    call    number_table    ;returns hex code for 9 from lookup table
    movwf   PORTB          ;sends hex code to display

    goto    button_check    ;returns to main program


count_inc                  ;code to increment and display count
    incf    count          ;count+=1

```

```

    movf    count, w          ;moves count to w
    call    number_table      ;returns hex code in position of count from
lookup_table
    movwf   PORTB             ;sends hex code to display

    call    pause             ;pause for 200ms

    goto    button_check      ;returns to main program


count_dec          ;subroutine to decrement and display count
    movf    count, w          ;moves value of count to w
    call    number_table      ;returns hex code in position of count from
lookup_table
    movwf   PORTB             ;outputs that hex number

    call    pause             ;pauses for 200ms

    goto    button_check      ;returns to main program


;=====

number_table

    addwf   PCL

    retlw   0x40              ;0
    retlw   0x79              ;1
    retlw   0x24              ;2
    retlw   0x30              ;3
    retlw   0x19              ;4
    retlw   0x12              ;5
    retlw   0x02              ;6
    retlw   0x58              ;7
    retlw   0x00              ;8
    retlw   0x10              ;9


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
```

Appendix E – Code to Operate Sounder When Buttons are Pressed

```
=====
; Main.asm file generated by New Project wizard
;
; Created:   Wed Oct 30 2024
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include pl16f84a.inc                ; Include register definition file

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

status   equ 0x03                    ;status register    number is in hex
RP0      equ 5                      ;bit 5 of status register assigns memory address as
register
RP1      equ 6                      ;bit 6 of status register

;=====
; VARIABLES
;=====

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

;=====
; RESET and INTERRUPT VECTORS
;=====

        ; Reset Vector
RST      code 0x0
        goto start

;=====
; CODE SEGMENT
;=====

PGM      code

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

button_check                ;infinitely checks if buttons are pressed

    bsf     TRISA, 0      ; Set RA0 (bit 0) as an input
    bsf     TRISA, 1      ; Set RA1 (bit 1) as an input

    btfss   PORTA, 0      ;checks if button 1 has been pressed
    call    turn_on       ;turns sounder on

    btfss   PORTA, 1      ;checks if button 2 has been pressed
    call    turn_on       ;turns sounder on

    goto    button_check  ; repeat forever

turn_on
    bsf     PORTB, 7      ;sets pin B7 high

    btfss   PORTA, 0      ;checks if button one has been released
    call    turn_off      ;turns off sonuder

    btfss   PORTA, 1      ;checks if button two has been released
    call    turn_off      ;turns off sounder

    bcf     PORTB, 7      ;sets pin B7 low
    nop                     ;creates even pwm cycle

    goto    turn_on       ;repeats the loop while buttons being held

turn_off
    bcf     PORTB, 7      ;turns off pin B7
    goto    button_check  ;returns to main program

;=====

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
```

2322601 - Saffina H - Using and Programming the 7-Segment Display

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

74/100

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

GRADING FORM: GJ_2019

SAFFINA H**368****ENGAGEMENT WITH LITERATURE**

A reasonable number of sources cited - but could have included even more to cover as many aspects of the explanation as possible (within reason of course!)

60**KNOWLEDGE AND UNDERSTANDING**

Despite the limited EWL, the actual explanations of the components and included diagrams are excellent.

80**COGNITIVE AND INTELLECTUAL**

Most areas show a good level of thought into how to solve the programming tasks. Solid, but not exceptionally innovative/creative.

68**PRACTICAL APPLICATION**



Circuit very well implemented, and 5 out of the 6 tasks well attempted with very well laid out flowcharts.

75

TRANSFERABLE SKILLS



Exceptionally well presented

85