

Final

by Bradley Gillies

Submission date: 10-Dec-2024 02:15PM (UTC+0000)

Submission ID: 245401077

File name: 51298_Bradley_Gillies_Final_621295_474688316.docx (5.04M)

Word count: 7351

Character count: 34014

ACCA4004: Using and Programming the 7 Segment Display

Bradley Gillies

Word Count: 1885 (-5.75%)

—
Embedded Electronic Systems
Degree Apprenticeship

—
Micros Peripherals and Interfaces

—
2310950

Table of Contents

Contents

Table of Contents.....	2
Table of Figures	3
Table of Tables.....	4
Proteus Simulation	5
7-Segment Display – Hexadecimal Values	6
PIC Assembly Programs	9
Displaying Hexadecimal Values 0-F	9
Two Buttons – ‘L’ and ‘r’	13
Latching Buttons.....	15
Button Controlled Counting	17
Piezo Buzzer	20
Piezo Buzzer Pulse Width Modulation (PWM)	22
Appendix A: Displaying Hexadecimal Values Raw Assembly	25
Appendix B: Two Buttons Raw Assembly.....	28
Appendix C: Latching Buttons Raw Assembly	30
Appendix D: Button Controlled Counting Raw Assembly.....	32
Appendix E: Piezo Buzzer Raw Assembly.....	36
Appendix F: Piezo Buzzer PWM Raw Assembly.....	38
References	40

Table of Figures

Figure 1: PIC16F84A Proteus Circuit	5
Figure 2: 7-Segment Display Connected to PIC Microcontroller.	6
Figure 3: Displaying Hexadecimal Values Flowchart.....	9
Figure 4: 'main' Subroutine to Display Hexadecimal Values.	10
Figure 5: 'init_ram' Subroutine to Store Hexadecimal Values.....	11
Figure 6: 'delay' Subroutine to Keep the Processor Busy for ~200ms.....	12
Figure 7: Equates and Pointers.	12
Figure 8: Flow Chart Diagram of 'L' and 'r' Button Program.	13
Figure 9: Main Loop / Function for Button Polling and Outputting to Display.....	14
Figure 10: Equates and Memory Pointers.	14
Figure 11: Flow Chart Diagram of Latching Buttons Program.	15
Figure 12: Latching Buttons Assembly Program Main Loop.....	16
Figure 13: Equates and Memory Pointers Used in Latching Buttons Program.	16
Figure 14: Flow Chart Diagram of Button Controlled Diagram Program.	17
Figure 15: 'main' Subroutine in Button Controlled Counting Program.	18
Figure 16: 'delay' Subroutine in Button Controlled Counting Program.....	18
Figure 17: 'init_ram' Subroutine in Button Controlled Counting Program.	19
Figure 18: 'count_up' Subroutine in Button Controlled Counting Program.....	19
Figure 19: 'count_down' Subroutine in Button Controlled Counting Program.....	19
Figure 20: Equates and Memory Pointers Used in Button Controlled Counting Program.....	20
Figure 21: Flow Chart Diagram of Piezo Buzzer Program.....	20
Figure 22: 'main' Loop / Function in Piezo Buzzer Program.	21
Figure 23: 'delay' Subroutine in Piezo Buzzer Program.	21
Figure 24: Equates and Memory Pointers Used in Piezo Buzzer Program.	22
Figure 25: Flow Chart Diagram of Piezo Buzzer PWM Program.	22
Figure 26: 'main' Subroutine in Piezo Buzzer PWM Program.	23
Figure 27: 'handler' Interrupt Subroutine in Piezo Buzzer PWM Program.	23
Figure 28: Initial Jumps and Equates and Memory Locations for Piezo Buzzer PWM Program.	24

Table of Tables

Table 1: Segments to Light Based on Hexadecimal Character.	7
Table 2: 7-Bit Binary Representation of LEDs.	7
Table 3: Final Binary Representation of Hexadecimal Values.	8
Table 4: Hexadecimal Values to be Output by the PIC.	8
Table 5: Binary Calculation Process for Letters 'L' and 'r'.	13
Table 6: Hexadecimal Output Values for PIC for Numbers 5-9 (Inclusive).	17

Proteus Simulation

Within lectures, a simple circuit has been introduced and this assignment piece is based on that circuit. The circuit can be seen in Figure 1 below.

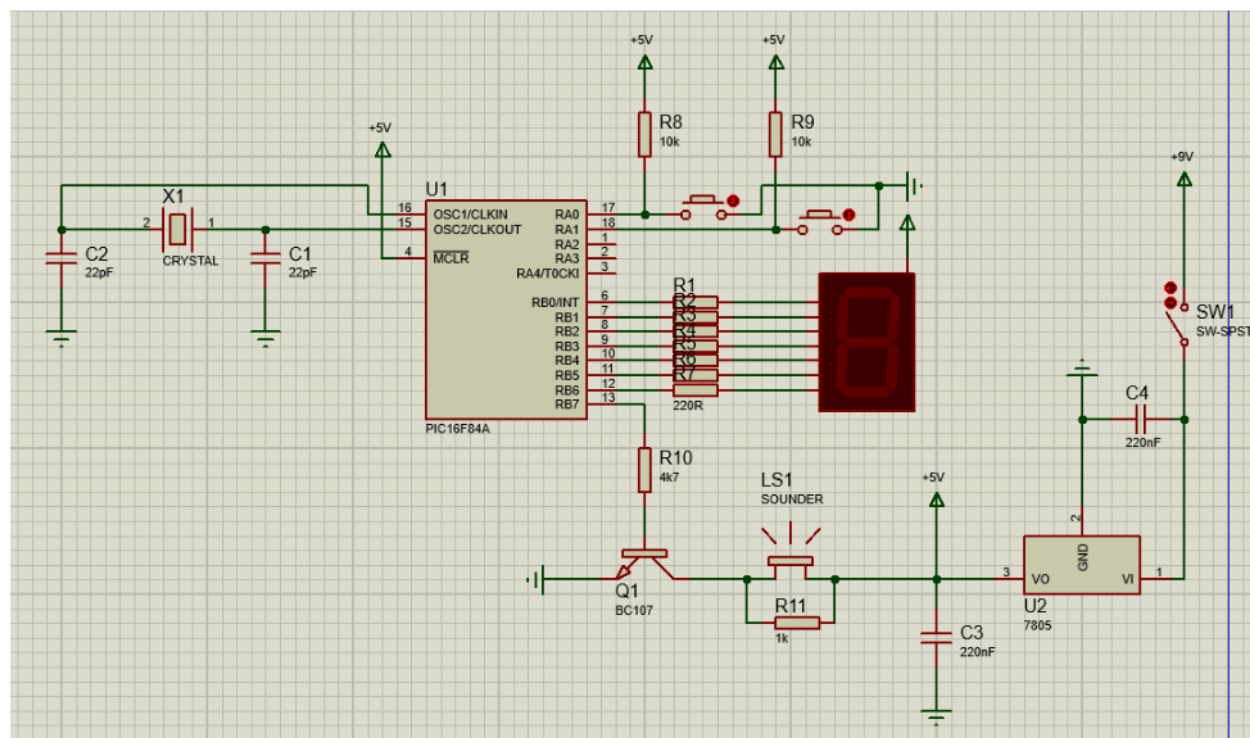


Figure 1: PIC16F84A Proteus Circuit

This circuit includes a variety of components which all serve a different purpose in the overall circuit.

PIC16F84A: An 8-bit microcontroller that was first made in 1993. This microcontroller comes in a variety of packages but the F variant is a plastic dual inline package (PDIP) and also features electrically erasable programmable read-only memory (EEPROM) [1]. This microcontroller will be responsible for controlling and using the other components within the circuit.

Crystal: A crystal is responsible for supplying clock signals to the microcontroller. All instructions that are executed by the microcontroller are in synchronisation with this clock signal [2].

Buttons: Buttons allow input from the user of the system as they connect two parts of the circuit together allowing for current and voltage to flow through. This typically results in either a high or low signal being produced that can be monitored to a microcontroller and actions can then be performed.

Seven segment common anode display: A light emitting diode (LED) display that has seven segments (LEDs) that can be used to represent decimal and hexadecimal values. A common anode display is more popular as many logic circuits can sink more current than they can source [3].

BC107: This is a low power bipolar negative-positive-negative (NPN) transistor. In this circuit, the purpose of the NPN transistor is to enable switching due to the transistor being connected to ground and an output pin from the microcontroller [4].

Sounder: This is representing a piezo buzzer which is an electronic device that can produce a tone, alarm or sound. This is being used in the circuit to enable pulse width modulation (PWM) to be performed by the microcontroller [5].

7805: An integrated circuit (IC) that is designed to maintain a consistent output voltage despite potential fluctuations in the input voltage and load. In particular, a 7805-voltage regulator is designed to ensure a stable +5V output [6]. In the physical version of this circuit, the power supply would be a 9V battery which would have fluctuations in what it output, so the voltage regulator is used to protect the circuit from any potential damage.

Switch: A switch has been used to turn the entire circuit on and off. This serves a greater purpose in the real world when compared to Proteus simulation of the circuit.

7-Segment Display – Hexadecimal Values

A seven-segment display was used to display the hexadecimal values 0 to F. As shown in Figure 2, the display is directly connected to the PIC microcontroller with 220-ohm resistors. The resistors are used to current limit what the LEDs can draw, so they do not become damaged in the real-world.

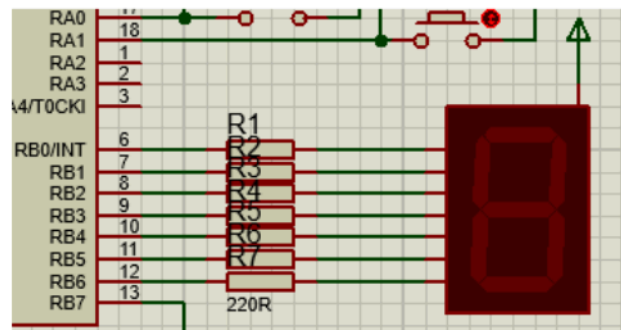


Figure 2: 7-Segment Display Connected to PIC Microcontroller.

Additionally, the display is connected to +5V due to sharing a common anode across all of the LEDs. Pins RB0 through RB6 are used to control the display from the PIC microcontroller. Due to the presence of a common anode, all of the values output from the PIC to light up the segments need to be inverted.

Shown in Table 1 are the mappings of each hexadecimal value (0 – F) and the corresponding segments that need to be lit to display the character.

Hexadecimal Value	Segment						
	A	B	C	D	E	F	G
0	✓	✓	✓	✓	✓	✓	
1		✓	✓				
2	✓	✓		✓	✓		✓
3	✓	✓	✓	✓			✓
4		✓	✓			✓	✓
5	✓		✓	✓		✓	✓
6	✓		✓	✓	✓	✓	✓
7	✓	✓	✓				
8	✓	✓	✓	✓	✓	✓	✓
9	✓	✓	✓			✓	✓
A	✓	✓	✓		✓	✓	✓
B			✓	✓	✓	✓	✓
C	✓			✓	✓	✓	
D		✓	✓	✓	✓		✓
E	✓			✓	✓	✓	✓
F	✓				✓	✓	✓

Table 1: Segments to Light Based on Hexadecimal Character.

This table has then been converted into a 7-bit number with segment A being the least significant bit (LSB) and segment F being the most significant bit (MSB). This conversion can be seen in Table 2.

Hexadecimal Value	7-Bit Representation
0	0111111
1	0000110
2	1011011
3	1001111
4	1100110
5	1101101
6	1111101
7	0000111
8	1111111
9	1100111
A	1110111
B	1111100
C	0111001
D	1011110
E	1111001
F	1110001

Table 2: 7-Bit Binary Representation of LEDs.

However, as previously stated, these 7-bit numbers need to be flipped due to the presence of a common anode within the display. As a result, the final 7-bit representation of the hexadecimal values is shown in Table 3.

Hexadecimal Value	7-Bit Representation	Final Binary Representation
0	0111111	1000000
1	0000110	1111001
2	1011011	0100100
3	1001111	0110000
4	1100110	0011001
5	1101101	0010010
6	1111101	0000010
7	0000111	1111000
8	1111111	0000000
9	1100111	0011000
A	1110111	0001000
B	1111100	0000011
C	0111001	1000110
D	1011110	0100001
E	1111001	0000110
F	1110001	0001110

Table 3: Final Binary Representation of Hexadecimal Values.

For ease of use within the PIC assembly programming language, these final binary numbers have been converted into the hexadecimal equivalents. Shown in Table 4 are these final values.

Hexadecimal Value	7-Bit Representation	Final Binary Representation	Hexadecimal to Output
0	0111111	1000000	40
1	0000110	1111001	79
2	1011011	0100100	24
3	1001111	0110000	30
4	1100110	0011001	19
5	1101101	0010010	12
6	1111101	0000010	02
7	0000111	1111000	78
8	1111111	0000000	00
9	1100111	0011000	18
A	1110111	0001000	08
B	1111100	0000011	03
C	0111001	1000110	46
D	1011110	0100001	21
E	1111001	0000110	06
F	1110001	0001110	0E

Table 4: Hexadecimal Values to be Output by the PIC.

PIC Assembly Programs

This section of the report will cover the six assembly programs set out in the assignment brief. Each will have a pseudocode representation as well as annotated subroutines from the assembly file.

All of the raw assembly code can be found to the appendix of this report.

Displaying Hexadecimal Values 0-F

The purpose of this application is to program the PIC microcontroller to output appropriate values to ports RBo through RB6 to visually display the hexadecimal values 0-F on the 7-segment display. Previously outlined in this report are the calculations and the final values in which the PIC microcontroller needs to output to complete this task.

Shown in Figure 3 is a high-level flowchart diagram of the program flow, to help aid the assembly programming process.

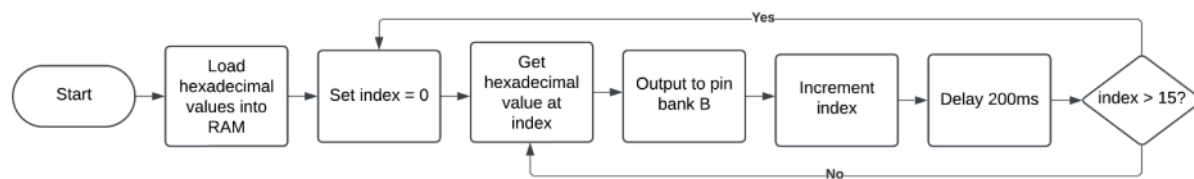


Figure 3: Displaying Hexadecimal Values Flowchart.

In the PIC assembly language, the program was split into three subroutines:

main: The entry point of the program and where the main program logic is called and handled.

init_ram: Initialises random access memory (RAM) addresses with the hexadecimal values that the PIC microcontroller needs to output.

delay: A simple loop that keeps the processor busy for approximately 200ms.

Shown in Figure 4, Figure 5 and, Figure 6 are the main, init_ram and delay subroutines along with annotations explaining the instructions being executed.

```

21; Subroutine:    main
22; Description:   The main function for the program. Starts by initialising the device registers
23;               as well as enabling port B and setting directionality of the pins. The RAM block
24;               that is utilised is then initialised with values for the LCD display. Then the
25;               main loop begins which iterates over the RAM memory block, displaying each LCD
26;               value for ~200ms and resetting once it has reached the end of the list of values.
27; Register Usage:
28;   w    -> Working register, temporary storage for values and calculations
29;   status -> For accessing status of the ALU, RESET and, the bank select bit for data memory
30;   trisB  -> For setting the directionality of pins on Port B
31;   portB  -> For outputting data on pins within Port B
32;   FSR    -> For indirect data memory addressing
33;   INDF   -> Uses the FSR to access data at memory locations (not a physical register)
34
35 main
36     BCF    status,RP0    ; Clear RP0 bit of status register
37     BCF    status,RP1    ; Clear RP1 bit of status register
38     BSF    status,RP0    ; Set RP0 bit of status register
39     MOVLW  0x00          ; Move value 0 to w
40     MOVWF  trisB         ; Set portB as all output pins
41     BCF    status,RP0    ; Clear RP0 bit of status register
42     CLRF   portB         ; Clear current pin values on portB
43
44     CALL   init_ram      ; Call init_ram to generate data
45
46 reset    MOVLW  hex_values ; Move start address of RAM into w
47           MOVWF  FSR       ; Move address in w to FSR
48 loop     MOVF   INDF, 0     ; Move the value at FSR (which is in INDF) to w
49           MOVWF  portB     ; Move value in w to port b -> display on LCD screen
50           CALL   delay     ; Delay the program for ~200ms
51           INCF   FSR       ; Increment the pointer to the next piece of memory in RAM
52           MOVF   FSR, 0    ; Load the address in FSR in w
53           XORLW  hex_end   ; XOR the address in w with the end of data in RAM
54           BTFSS  status,Z  ; If the address are equal, Z bit of status register is high (skip next instruction, replace with NOP)
55           GOTO   loop      ; Jump back to 'loop' Label to display the next number on the LCD display
56           GOTO   reset     ; Jump back to 'reset' Label to start counting up again
57           RETURN          ; Return from subroutine

```

Figure 4: 'main' Subroutine to Display Hexadecimal Values.

```

78 ; Subroutine:      init_ram
79 ; Description:     Initialises the top 16 bytes of RAM to contain
80 ;                  all of the hexadecimal values that need to be
81 ;                  sent on portB to correctly light different
82 ;                  segments of the LCD display.
83 ; Register Usage:
84 ;     w    -> Working register, stores the literal hexadecimal
85 ;              value that is being stored in RAM.
86 ;     hex_values -> Contains the address of the beginning of RAM.
87
88 init_ram
89     MOVLW    0x40      ; Hex value for: 0
90     MOVWF    hex_values ; Store value at: 0x0C
91     MOVLW    0x79      ; Hex value for: 1
92     MOVWF    hex_values+0x01 ; Store value at: 0x0D
93     MOVLW    0x24      ; Hex value for: 2
94     MOVWF    hex_values+0x02 ; Store value at: 0x0E
95     MOVLW    0x30      ; Hex value for: 3
96     MOVWF    hex_values+0x03 ; Store value at: 0x0F
97     MOVLW    0x19      ; Hex value for: 4
98     MOVWF    hex_values+0x04 ; Store value at: 0x10
99     MOVLW    0x12      ; Hex value for: 5
100    MOVWF    hex_values+0x05 ; Store value at: 0x11
101    MOVLW    0x02      ; Hex value for: 6
102    MOVWF    hex_values+0x06 ; Store value at: 0x12
103    MOVLW    0x78      ; Hex value for: 7
104    MOVWF    hex_values+0x07 ; Store value at: 0x13
105    MOVLW    0x00      ; Hex value for: 8
106    MOVWF    hex_values+0x08 ; Store value at: 0x14
107    MOVLW    0x10      ; Hex value for: 9
108    MOVWF    hex_values+0x09 ; Store value at: 0x15
109    MOVLW    0x08      ; Hex value for: A
110    MOVWF    hex_values+0x0A ; Store value at: 0x16
111    MOVLW    0x03      ; Hex value for: b
112    MOVWF    hex_values+0x0B ; Store value at: 0x17
113    MOVLW    0x46      ; Hex value for: C
114    MOVWF    hex_values+0x0C ; Store value at: 0x18
115    MOVLW    0x21      ; Hex value for: d
116    MOVWF    hex_values+0x0D ; Store value at: 0x19
117    MOVLW    0x06      ; Hex value for: E
118    MOVWF    hex_values+0x0E ; Store value at: 0x1A
119    MOVLW    0x0E      ; Hex value for: F
120    MOVWF    hex_values+0x0F ; Store value at: 0x1B
121    RETURN           ; Return from subroutine
122

```

Figure 5: 'init_ram' Subroutine to Store Hexadecimal Values.

```

59; Subroutine: delay
60; Description: Keeps the processor busy for ~200ms
61; Register Usage:
62;     w    -> Working register, temporary storage of
63;             counters.
64;     outer -> Outer loop counter - stored in RAM
65;     inner  -> Inner loop counter - stored in RAM
66
67delay
68    MOVLW    0xFF    ; w = 255
69    MOVWF    outer    ; outer = w -> outer loop counter
70loop_o    MOVLW    0xFF    ; w = 255
71    MOVWF    inner    ; inner = w -> inner loop counter
72loop_i    DECFSZ    inner, f    ; inner--
73    GOTO     loop_i    ; if inner != 0 -> Loop inner
74    DECFSZ    outer, f    ; outer--
75    GOTO     loop_o    ; if outer != 0 -> Loop outer
76    RETURN           ; Return from subroutine
77

```

Figure 6: 'delay' Subroutine to Keep the Processor Busy for ~200ms.

Alongside this there is some initial setup of equates and pointers to starting address of memory being used, so that later in the assembly program, these values and locations can be referred to using a meaning mnemonic. These are shown in Figure 7.

```

1
2    LIST    p=16F84    ; Switch on output to the .LST file
3    RADIX    hex    ; What base is being used to represent numbers
4
5#include p16f84a.inc    ; Find include file for the microprocessor and include it here
6
7    UDATA_SHR    ; Declares the beginning of a section of shared uninitialised data
8hex_values    RES    0x10    ; Pointer to start of a memory block that has been reserved (size of 16 bytes)
9hex_end    ; Pointer to memory address directly after data being used
10
11portB    EQU    0x06    ; Pointer to memory address that is written to port B
12trisB    EQU    0x06    ; Pointer to memory address of data direction register for port B
13status    EQU    0x03    ; Pointer to memory address where status register is stored
14RP0    EQU    5    ; Bit number of RP0 in status register
15RP1    EQU    6    ; Bit number of RP1 in status register
16inner    EQU    0x1C    ; RAM memory address of data
17outer    EQU    0x1D    ; RAM memory address of data
18
19    ORG    0x000    ; Start the program at 0x000

```

Figure 7: Equates and Pointers.

With the use of PIC microcontroller and the assembly code above, the circuit correctly loops through hexadecimal values 0-f and resets back to 0 once the loop has completed.

Two Buttons – ‘L’ and ‘r’

The purpose of this application is to display ‘L’ and ‘r’ on the 7-segment display when the buttons on RA0 and RA1 are pressed respectively.

Shown in Figure 8, is a high-level flow chart diagram of the program flow, to help aid the assembly programming process.

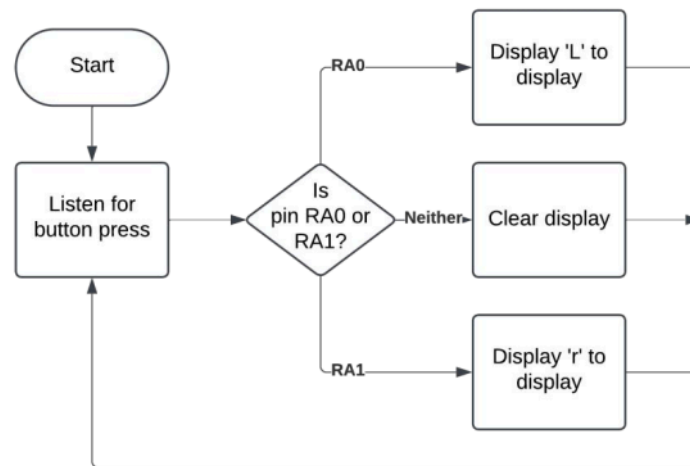


Figure 8: Flow Chart Diagram of 'L' and 'r' Button Program.

Similarly, to the hexadecimal counter program, the letters ‘L’ and ‘r’ needed to be translated into a hexadecimal value that is to be output onto pins RB0 through RB6. Shown in Table 5, is the breakdown of this process.

Value to Display	Regions to Light	7-bit Representation	Flipped Binary	Hexadecimal to Output
L	D, E, F	0111000	1000111	47
r	A, E, F	0110001	1001110	4E

Table 5: Binary Calculation Process for Letters 'L' and 'r'.

In the PIC assembly language, the program is relatively simple and as such, only the main loop / function is utilised to perform the process. Shown in Figure 9, is the main loop with appropriate comments detailing what is happening on each line.

```

19 ; Subroutine:      main
20 ; Description:     The main function for the program. Starts by initialising the device registers
21 ;                 as well as enabling port A & B and setting directionality of the pins. The main
22 ;                 loop then iterates over the two pins polling to see if the button has been activated.
23 ;                 If it has, then it modifies the w register to output the correct character and then
24 ;                 outputs this to port B (and the display) and loops back to the beginning.
25 ; Register Usage:
26 ;     w    -> Working register, temporary storage for values and calculations
27 ;     status -> For accessing status of the ALU, RESET and, the bank select bit for data memory
28 ;     trisB -> For setting the directionality of pins on Port B
29 ;     portB -> For outputting data on pins within Port B
30 ;     trisA -> For setting the directionality of pins on Port A
31 ;     portA -> For outputting data on pins within Port A
32
33 main
34     BCF     status,RP0    ; Clear RP0 bit of status register
35     BCF     status,RP1    ; Clear RP1 bit of status register
36     BSF     status,RP0    ; Set RP0 bit of status register
37     MOVLW   0x00          ; Move value 0 to w
38     MOVWF   trisB         ; Set portB as all output pins
39
40     MOVLW   0x03          ; Move value 3 to w
41     MOVWF   trisA         ; Set portA pin 0 and 1 to input pins
42     BCF     status,RP0    ; Clear RP0 bit of status register
43     CLRF    portA         ; Clear current pin values on portA
44     CLRF    portB         ; Clear current pin values on portB
45
46 loop    MOVLW   0xFF      ; Move value 255 to w
47         BTFSS   portA,0    ; Check if PIN A0 is high if it is, NOP next instruction, if Low execute next instruction
48         ANDLW   letter_L   ; AND 0xFF and 0x47 and store result in w (0x47)
49         BTFSS   portA,1    ; Check if PIN A1 is high if it is, NOP next instruction, if Low execute next instruction
50         ANDLW   letter_r   ; AND 0xFF and 0x4E and store result in w (0x4E)
51         MOVWF   portB      ; Output w to port b to display 'L' or 'r'
52         GOTO    loop       ; Loop back to the beginning of the logic

```

Figure 9: Main Loop / Function for Button Polling and Outputting to Display.

Shown in Figure 10, are the equates and memory pointers that were used within the PIC assembly program.

```

1
2     LIST      p=16F84      ; Switch on output to the .LST file
3     RADIX     hex          ; What base is being used to represent numbers
4
5 #include p16f84a.inc      ; Find include file for the microprocessor and include it here
6
7 portB        EQU         0x06      ; Pointer to memory address this is written to port B
8 trisA        EQU         0x05      ; Pointer to memory address of data direction register for port B
9 portA        EQU         0x05      ; Pointer to memory address that is written to port A
10 trisB        EQU         0x06      ; Pointer to memory address of data direction register for port A
11 status       EQU         0x03      ; Pointer to memory address where status register is stored
12 RP0          EQU         5         ; Bit number of RP0 in status register
13 RP1          EQU         6         ; Bit number of RP1 in status register
14 letter_L     EQU         0x47      ; Hexadecimal value to output to display 'L'
15 letter_r     EQU         0x4E      ; Hexadecimal value to output to display 'r'
16
17     ORG       0x000           ; Start the program at 0x000

```

Figure 10: Equates and Memory Pointers.

With the use of the PIC microcontroller and assembly program outlined above, the circuit correctly polls the buttons on the circuit and outputs 'L' and 'r' to the 7-segment display.

Latching Buttons

The purpose of this application is to create software latches for the two buttons present on the circuit. When one button is pressed, all the LED segments should be lit and stay lit until the other button is pressed.

Shown in Figure 11, is a high-level flowchart diagram of the program flow, to help aid the assembly programming process.

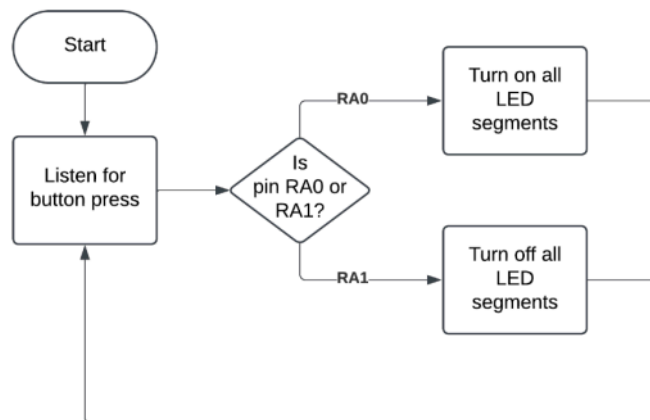


Figure 11: Flow Chart Diagram of Latching Buttons Program.

To enable the PIC microcontroller to turn on all the LED segments a value of 0x00 needs to be output to port B and to turn off all the LED segments a value of 0xFF needs to be output to port B.

In the PIC assembly language, the program is relatively simple and as such only the main loop / function is utilised to perform the process. Shown in Figure 12, is the main loop with appropriate comments detailing what is happening on each line.

```

19; Subroutine:      main
20; Description:     The main function for the program. Starts by initialising the device registers
21;                 as well as enabling port A & B and setting directionality of the pins. The main
22;                 loop then iterates over the two pins polling to see if the button has been activated.
23;                 If it has, then it modifies the w register to output the correct light sequence and then
24;                 outputs this to port B (and the display) and loops back to the beginning.
25; Register Usage:
26;   w   -> Working register, temporary storage for values and calculations
27;   status -> For accessing status of the ALU, RESET and, the bank select bit for data memory
28;   trisB -> For setting the directionality of pins on Port B
29;   portB -> For outputting data on pins within Port B
30;   trisA -> For setting the directionality of pins on Port A
31;   portA -> For outputting data on pins within Port A
32
33main
34      BCF      status,RP0      ; Clear RP0 bit of status register
35      BCF      status,RP1      ; Clear RP1 bit of status register
36      BSF      status,RP0      ; Set RP0 bit of status register
37      MOVLW    0x00            ; Move value 0 to w
38      MOVWF    trisB           ; Set portB as all output pins
39
40      MOVLW    0x03            ; Move value 3 to w
41      MOVWF    trisA           ; Set portA pin 0 and 1 to input pins
42      BCF      status,RP0      ; Clear RP0 bit of status register
43      CLRF     portA           ; Clear current pin values on portA
44      MOVLW    all_off         ; Move value to turn all LEDs off to w
45      MOVWF    portB           ; Output the value in w to port B
46
47loop   BTFSF    portA,0         ; Check if PIN A0 is high if it is, NOP next instruction, if Low execute next instruction
48      MOVLW    all_on          ; Move Literal 0x00 to register w
49      BTFSF    portA,1         ; Check if PIN A1 is high if it is, NOP next instruction, if Low execute next instruction
50      MOVLW    all_off         ; Mover Literal 0xFF to register w
51      MOVWF    portB           ; Output w to port b to display all or none of the LED segments
52      GOTO     loop            ; Loop back to the beginning of the logic

```

Figure 12: Latching Buttons Assembly Program Main Loop.

Similarly, to the previous assembly programs outlined, shown in Figure 13, are the equates and memory pointers that were used within the program.

```

1
2      LIST      p=16F84        ; Switch on output to the .LST file
3      RADIX     hex           ; What base is being used to represent numbers
4
5#include p16f84a.inc           ; Find include file for the microprocessor and include it here
6
7portB      EQU      0x06       ; Pointer to memory address this is written to port B
8trisA      EQU      0x05       ; Pointer to memory address of data direction register for port B
9portA      EQU      0x05       ; Pointer to memory address that is written to port A
0trisB      EQU      0x06       ; Pointer to memory address of data direction register for port A
1status     EQU      0x03       ; Pointer to memory address where status register is stored
2RP0        EQU      5          ; Bit number of RP0 in status register
3RP1        EQU      6          ; Bit number of RP1 in status register
4all_on     EQU      0x00       ; Hexadecimal value to output to turn on all LED segments
5all_off    EQU      0xFF       ; Hexadecimal value to output to turn off all LED segments
6
7      ORG       0x000          ; Start the program at 0x000

```

Figure 13: Equates and Memory Pointers Used in Latching Buttons Program.

With the use of the PIC microcontroller and the assembly program outlined above, the circuit correctly polls and latches the buttons on the circuit and turns on and off all the LED segments appropriately.

Button Controlled Counting

The purpose of this application is to enable the circuit to count from 5 to 9 when one button is pressed, and backwards when the other button is pressed.

Shown in Figure 14, is a high-level flow chart diagram of the program flow, to help aid the assembly programming process.

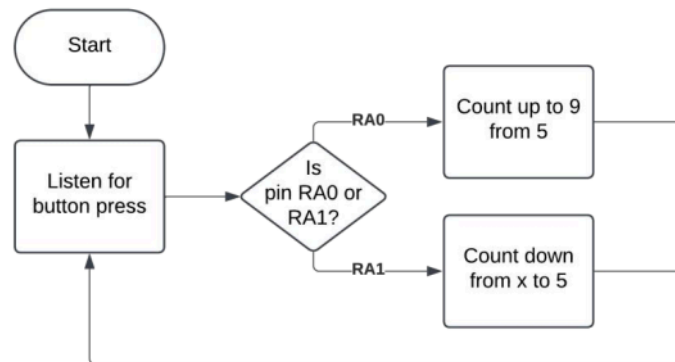


Figure 14: Flow Chart Diagram of Button Controlled Diagram Program.

To enable the PIC microcontroller to output the numbers 5 to 9 inclusive on the 7-segment display, the appropriate hex output values need be used. These can be taken from the first PIC program in this report but are also shown in Table 6.

Hexadecimal Value	7-Bit Representation	Final Binary Representation	Hexadecimal to Output
5	1101101	0010010	12
6	1111101	0000010	02
7	0000111	1111000	78
8	1111111	0000000	00
9	1100111	0011000	18

Table 6: Hexadecimal Output Values for PIC for Numbers 5-9 (Inclusive).

In the PIC assembly language, the program was split into five subroutines:

main: The entry point of the program and where the main program logic is called and handled.

delay: A simple loop that keeps the processor busy for approximately 200ms.

init_ram: Initialises random access memory (RAM) addresses with the hexadecimal values that the PIC microcontroller needs to output.

count_up: Loads the values from RAM in a positive order and outputs the value to port B to display the correct number.

count_down: Loads the values from RAM in a negative order and outputs the value to port B to display the correct number.

Shown in Figure 15, Figure 16, Figure 17, Figure 18 and, Figure 19 are the main, init_ram and delay subroutines along with annotations explaining the instructions being executed.

```

23; Subroutine:    main
24; Description:   The main function for the program. Starts by initialising the device registers
25;               as well as enabling port B & A and setting directionality of the pins. The RAM block
26;               that is utilised is then initialised with values for the LCD display. Then the
27;               main loop begins which polls the two buttons on the circuit and calls appropriate
28;               subroutines dependant on which button is high.
29; Register Usage:
30;   w    -> Working register, temporary storage for values and calculations
31;   trisB -> For setting the directionality of pins on Port B
32;   portB -> For outputting data on pins within Port B
33
34main
35    BCF    status,RP0    ; Clear RP0 bit of status register
36    BCF    status,RP1    ; Clear RP1 bit of status register
37    BSF    status,RP0    ; Set RP0 bit of status register
38    MOVLW  0x00          ; Move value 0 to w
39    MOVWF  trisB         ; Set portB as all output pins
40
41    MOVLW  0x03          ; Move value 3 to w
42    MOVWF  trisA         ; Set portA pin 0 and 1 to input pins
43    BCF    status,RP0    ; Clear RP0 bit of status register
44    CLRF   portA         ; Clear current pin values on portA
45    CLRF   portB         ; Clear current pin values on portB
46
47    MOVLW  0xFF          ; Move the Literal 0xFF into w
48    MOVWF  portB         ; Output w to port B
49
50    CALL   init_ram      ; Call init_ram to generate data
51
52loop   BTFSS   portA,0    ; Check if PIN A0 is high if it is, NOP next instruction, if Low execute next instruction
53       CALL   count_up   ; Call (branch) to the 'count_up' subroutine
54       BTFSS   portA,1    ; Check if PIN A1 is high if it is, NOP next instruction, if Low execute next instruction
55       CALL   count_down ; Call (branch) to the 'count_down' subroutine
56       GOTO   loop       ; Loop back to the beginning of the logic
57       RETURN            ; Return from subroutine
58

```

Figure 15: 'main' Subroutine in Button Controlled Counting Program.

```

105; Subroutine:    delay
106; Description:   Keeps the processor busy for ~200ms
107; Register Usage:
108;   w    -> Working register, temporary storage of
109;           counters.
110;   outer -> Outer loop counter - stored in RAM
111;   inner -> Inner loop counter - stored in RAM
112
113delay
114    MOVLW  0xFF          ; w = 255
115    MOVWF  outer         ; outer = w -> outer loop counter
116loop_o  MOVLW  0xFF          ; w = 255
117       MOVWF  inner      ; inner = w -> inner loop counter
118loop_i  DECFSZ  inner, f    ; inner--
119       GOTO   loop_i      ; if inner != 0 -> Loop inner
120       DECFSZ  outer, f    ; outer--
121       GOTO   loop_o      ; if outer != 0 -> Loop outer
122       RETURN            ; Return from subroutine

```

Figure 16: 'delay' Subroutine in Button Controlled Counting Program.

```

124; Subroutine:    init_ram
125; Description:   Initialises the top 5 bytes of RAM to contain
126;               all of the hexadecimal values that need to be
127;               sent on portB to correctly light different
128;               segments of the LCD display.
129; Register Usage:
130;     w  -> Working register, stores the literal hexadecimal
131;           value that is being stored in RAM.
132;     hex_values -> Contains the address of the beginning of RAM.
133
134init_ram
135    MOVLW    0x12      ; Hex value for: 5
136    MOVWF    hex_values ; Store value at: 0x11
137    MOVLW    0x02      ; Hex value for: 6
138    MOVWF    hex_values+0x01 ; Store value at: 0x12
139    MOVLW    0x78      ; Hex value for: 7
140    MOVWF    hex_values+0x02 ; Store value at: 0x13
141    MOVLW    0x00      ; Hex value for: 8
142    MOVWF    hex_values+0x03 ; Store value at: 0x14
143    MOVLW    0x10      ; Hex value for: 9
144    MOVWF    hex_values+0x04 ; Store value at: 0x15
145    RETURN           ; Return from subroutine

```

Figure 17: 'init_ram' Subroutine in Button Controlled Counting Program.

```

59; Subroutine:    count_up
60; Description:   Iterates over the initialised RAM memory address containing hex values to output
61;               which display values 5-9 inclusive. This subroutine counts through these values
62;               in the positive direction.
63; Register Usage:
64;     w  -> Working register, temporary storage for values and calculations
65;     status -> For accessing status of the ALU, RESET and, the bank select bit for data memory
66;     FSR  -> For indirect data memory addressing
67;     INDF -> Uses the FSR to access data at memory locations (not a physical register)
68
69count_up
70    MOVLW    hex_values ; Move start address of RAM into w
71    MOVWF    FSR        ; Move address in w to FSR
72up_loop    MOVF    INDF,0 ; Move the value at FSR (which is in INDF) to w
73           MOVWF    portB ; Move value in w to port b -> display on LCD screen
74           CALL    delay ; Delay the program for ~200ms
75           INCF    FSR ; Increment the pointer to the next piece of memory in RAM
76           MOVF    FSR,0 ; Load the address in FSR in w
77           XORLW    hex_end ; XOR the address in w with the end of data in RAM
78           BTFSS    status,Z ; If the address are equal, Z bit of status register is high (skip next instruction, replace with NOP)
79           GOTO     up_loop ; Jump back to 'up_loop' Label to display the next number on the LCD display
80           RETURN      ; Return from subroutine

```

Figure 18: 'count_up' Subroutine in Button Controlled Counting Program.

```

82; Subroutine:    count_down
83; Description:   Iterates over the initialised RAM memory address containing hex values to output
84;               which display values 9-5 inclusive. This subroutine counts through these values
85;               in the negative direction.
86; Register Usage:
87;     w  -> Working register, temporary storage for values and calculations
88;     status -> For accessing status of the ALU, RESET and, the bank select bit for data memory
89;     FSR  -> For indirect data memory addressing
90;     INDF -> Uses the FSR to access data at memory locations (not a physical register)
91
92count_down
93    MOVLW    hex_values+0x04 ; Move end address of RAM into w
94    MOVWF    FSR            ; Move address in w to FSR
95down_loop  MOVF    INDF,0 ; Move the value at FSR (which is in INDF) to w
96           MOVWF    portB ; Move the value in w to port b -> display on LCD screen
97           CALL    delay ; Delay the program for ~200ms
98           DECF    FSR ; Decrement the pointer to the next piece of memory in RAM
99           MOVF    FSR,0 ; Load the address in FSR in w
100           XORLW    hex_values-0x01 ; XOR the address in w with the address before the beginning of RAM
101           BTFSS    status,Z ; If the address are equal, Z bit of status register is high (skip next instruction, replace with NOP)
102           GOTO     down_loop ; Jump back to 'down_loop' Label to display the next number on the LCD display
103           RETURN      ; Return from subroutine

```

Figure 19: 'count_down' Subroutine in Button Controlled Counting Program.

Similarly, to the previous assembly programs outlined, shown in Figure 20, are the equates and memory pointers that were used within the program.

```

2          LIST    p=16F84      ; Switch on output to the .LST file
3          RADIX   hex          ; What base is being used to represent numbers
4
5 #include p16f84a.inc          ; Find include file for the microprocessor and include it here
6
7          UDATA_SHR          ; Declares the beginning of a section of shared uninitialised data
8 hex_values RES    0x05        ; Pointer to start of a memory block that has been reserved (size of 16 bytes)
9 hex_end   ; Pointer to memory address directly after data being used
10
11 portB    EQU      0x06        ; Pointer to memory address this is written to port B
12 trisA    EQU      0x05        ; Pointer to memory address of data direction register for port B
13 portA    EQU      0x05        ; Pointer to memory address that is written to port A
14 trisB    EQU      0x06        ; Pointer to memory address of data direction register for port A
15 status   EQU      0x03        ; Pointer to memory address where status register is stored
16 RP0      EQU      5          ; Bit number of RP0 in status register
17 RP1      EQU      6          ; Bit number of RP1 in status register
18 inner    EQU      0x1C        ; RAM memory address of data
19 outer    EQU      0x1D        ; RAM memory address of data
20
21          ORG      0x000        ; Start the program at 0x000

```

Figure 20: Equates and Memory Pointers Used in Button Controlled Counting Program.

With the use of the PIC microcontroller and the assembly programme outlined above, the circuit correctly polls the buttons and counts up and down within the range of 5-9 inclusive and displays this to the 7-segment display.

Piezo Buzzer

The purpose of this application is to enable the circuit to output a sound on the buzzer. This can be achieved by switching RB7 between high and low with a set delay between the actions.

Shown in Figure 21, is a high-level flow chart diagram of the program flow, to help aid the assembly programming process.

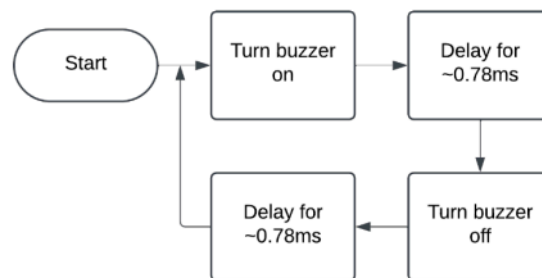


Figure 21: Flow Chart Diagram of Piezo Buzzer Program.

In the PIC assembly language, the program is relatively simple and as such only the main loop / function and delay subroutine is utilised to perform the process. Shown in Figure 22 and Figure 23, is the main loop and delay subroutine with appropriate comments detailing what is happening on each line.

```

20 ; Subroutine:    main
21 ; Description:   The main function for the program. Starts by initialising the device registers
22 ;               as well as enabling port B & A and setting directionality of the pins. It then
23 ;               simply loops and turns the buzzer on and off with a delay in between. Generating
24 ;               an audible noise from the buzzer.
25 ; Register Usage:
26 ;       w  -> Working register, temporary storage for values and calculations
27 ;       trisB -> For setting the directionality of pins on Port B
28 ;       portB -> For outputting data on pins within Port B
29
30 main
31     BCF     status,RP0    ; Clear RP0 bit of status register
32     BCF     status,RP1    ; Clear RP1 bit of status register
33     BSF     status,RP0    ; Set RP0 bit of status register
34     MOVLW   0x00          ; Move value 0 to w
35     MOVWF   trisB         ; Set portB as all output pins
36
37     MOVLW   0x03          ; Move value 3 to w
38     MOVWF   trisA         ; Set portA pin 0 and 1 to input pins
39     BCF     status,RP0    ; Clear RP0 bit of status register
40     CLRF    portA         ; Clear current pin values on portA
41     CLRF    portB         ; Clear current pin values on portB
42
43     MOVLW   0xFF          ; Move the literal 0xFF into w
44     MOVWF   portB         ; Output w to port B
45
46 loop    MOVLW   buzz_on    ; Move literal 0x80 into w
47         MOVWF   portB      ; Output w to port B (buzzer on)
48         CALL    delay      ; Call delay
49         MOVLW   buzz_off   ; Move literal 0x00 into w
50         MOVWF   portB      ; Output w to port B (buzzer off)
51         CALL    delay      ; Call delay
52         GOTO    loop       ; Jump to 'loop' label
53
54     RETURN                ; Return from subroutine

```

Figure 22: 'main' Loop / Function in Piezo Buzzer Program.

```

56 ; Subroutine:    delay
57 ; Description:   Keeps the processor busy for ~0.78ms
58 ; Register Usage:
59 ;       w  -> Working register, temporary storage of
60 ;               counters.
61 ;       delay_c -> Outer loop counter - stored in RAM
62
63 delay
64     MOVLW   0xFF          ; w = 255
65     MOVWF   delay_c       ; delay_c = w -> loop counter
66 loop_d    DECFSZ  delay_c, f ; delay_c--
67         GOTO    loop_d     ; if delay_c != 0 -> loop_d
68     RETURN                ; Return from subroutine

```

Figure 23: 'delay' Subroutine in Piezo Buzzer Program.

Similarly, to the previous assembly programs outlined, shown in Figure 24, are the equates and memory pointers that were used within the program.

```

1
2          LIST      p=16F84          ; Switch on output to the .LST file
3          RADIX     hex              ; What base is being used to represent numbers
4
5 #include p16f84a.inc              ; Find include file for the microprocessor and include it here
6
7 portB      EQU      0x06           ; Pointer to memory address this is written to port B
8 trisA      EQU      0x05           ; Pointer to memory address of data direction register for port B
9 portA      EQU      0x05           ; Pointer to memory address that is written to port A
10 trisB      EQU      0x06           ; Pointer to memory address of data direction register for port A
11 status    EQU      0x03           ; Pointer to memory address where status register is stored
12 RP0       EQU      5              ; Bit number of RP0 in status register
13 RP1       EQU      6              ; Bit number of RP1 in status register
14 delay_c    EQU      0x1D          ; RAM memory address of data
15 buzz_on    EQU      0x80          ; Value to output to port B to turn buzzer on
16 buzz_off   EQU      0x00          ; Value to output to port B to turn buzzer off
17
18          ORG      0x000           ; Start the program at 0x000

```

Figure 24: Equates and Memory Pointers Used in Piezo Buzzer Program.

With the use of the PIC microcontroller and the assembly programme outlined above, the circuit correctly outputs a single tone with the use of the piezo buzzer.

Piezo Buzzer Pulse Width Modulation (PWM)

The purpose of this application is to enable the circuit to output a fixed frequency pulse to the sounder. However, this pulse should have a constantly changing PWM waveform resulting in a varying pulse width.

Shown in Figure 25, is a high-level flow chart diagram of the program flow, to help aid the assembly programming process.

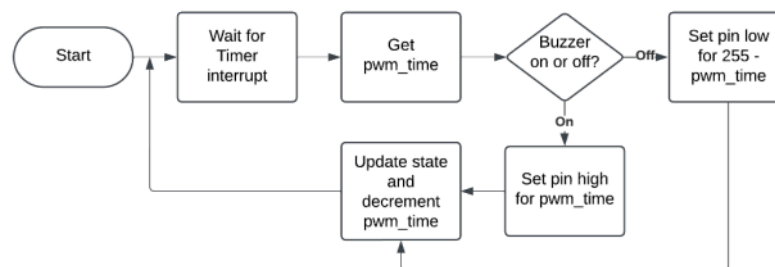


Figure 25: Flow Chart Diagram of Piezo Buzzer PWM Program.

In the PIC assembly language, the Timero interrupt was used to generate the timings for the pulse. This enabled the overall pulse width to remain the same, but the duration the buzzer is on and off for changes over time. When the 'pwm_time' variable gets to 0, the decrement causes the value to jump back to 255, resulting in the cycle restarting.

The program has been split into a main and handler function. Some logic has been added at 0x004 which tells the PIC to jump to 'handler' when an interrupt is fired and at 0x000 which tells the PIC to go to 'main' for some initial setup of registers.

Shown in Figure 26, Figure 27 and, Figure 28 is the main and handler function, as well as the initial jumps.

```

19 main
20     BSF     INTCON, T0IE    ; Enable the Timer0 interrupt
21     BSF     INTCON, GIE    ; Enable global interrupts
22
23     BANKSEL TRISB          ; Select bank for TRISB
24     MOVLW   0x00           ; Move Literal 0x00 to w
25     MOVWF   TRISB          ; Set all ports on B to be outputs
26
27     BANKSEL OPTION_REG     ; Select bank for OPTION_REG
28     MOVF    OPTION_REG,w   ; Move OPTION_REG into w
29     ANDLW   0xC0           ; Mask off bits 7 and 6
30     IORLW   0x04           ; Set prescaler to 4 (1:32)
31     MOVWF   OPTION_REG     ; Move w to OPTION_REG
32
33     BANKSEL TMR0           ; Select bank for TMR0
34     MOVLW   0xFF           ; Move Literal 0xFF into w
35     MOVWF   TMR0           ; Move w to TMR0
36
37     BANKSEL TMR0           ; Select bank for TMR0
38     MOVLW   0xFF           ; Move Literal 0xFF into w
39     MOVWF   pwm_time       ; Move w to pwm_time
40
41 loop    GOTO    loop        ; Infinite loop

```

Figure 26: 'main' Subroutine in Piezo Buzzer PWM Program.

```

43 handler
44     BTFSS   INTCON, T0IF    ; Test if interrupt is from Timer0
45     RETFIE  ; If it isn't, return from interrupt
46     BCF     INTCON, T0IF    ; Clear the interrupt flag for Timer0
47     BANKSEL TMR0           ; Select bank for TMR0
48     MOVF    pwm_time,w     ; Move the value at pwm_time into w
49     BTFSS   buzz_state,0    ; Test the LSB of buzz_state: 0 -> execute next instruction. 1 -> NOP next instruction
50     SUBLW   0xFF           ; Subtract pwm_time from 255 (255 - pwm_time)
51     MOVWF   TMR0           ; Move value in w to Timer0
52     DECF    pwm_time,1      ; Decrement pwm_time and store back in pwm_time
53
54     MOVLW   buzz_on        ; Move Literal buzz_on into w
55     BTFSS   buzz_state,0    ; Test the LSB of buzz_state: 0 -> execute next instruction. 1 -> NOP next instruction
56     MOVLW   buzz_off       ; Move Literal buzz_off into w
57     BANKSEL PORTB          ; Select bank for PORTB
58     MOVWF   PORTB          ; Move w to PORTB
59
60     MOVLW   0x01           ; Move Literal 0x01 into w
61     XORWF   buzz_state,1    ; XOR w with buzz_state, store result back into buzz_state
62     RETFIE  ; Return from interrupt

```

Figure 27: 'handler' Interrupt Subroutine in Piezo Buzzer PWM Program.

```

1
2          LIST      p=16F84      ; Switch on output to the .LST file
3          RADIX     hex          ; What base is being used to represent numbers
4
5 #include p16f84a.inc          ; Find include file for the microprocessor and include it here
6
7 pwm_time      EQU    0x0C      ; RAM memory address of data
8 delay_c       EQU    0x0D      ; RAM memory address of data
9 buzz_on       EQU    0x80      ; Value to output to port B to turn buzzer on
10 buzz_off      EQU    0x00      ; Value to output to port B to turn buzzer off
11 buzz_state    EQU    0x0E      ; RAM memory address of data
12
13 reset         CODE    0x000     ; Start the program at 0x000
14             GOTO     main      ; Jump to 'main' Label
15
16 interrupt     CODE    0x004     ; Jump to 'handler' Label
17             GOTO     handler

```

Figure 28: Initial Jumps and Equates and Memory Locations for Piezo Buzzer PWM Program.

With the use of the PIC microcontroller and assembly program outlined above, the circuit correctly outputs a pulse to the piezo buzzer with a constantly changing pulse width.

Appendix A: Displaying Hexadecimal Values Raw Assembly

```
LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

          UDATA_SHR    ; Declares the beginning of a section of shared
uninitialised data
hex_values RES      0x10 ; Pointer to start of a memory block that has been
reserved (size of 16 bytes)
hex_end   ; Pointer to memory address directly after data being
used

portB     EQU      0x06 ; Pointer to memory address that is written to port B
trisB     EQU      0x06 ; Pointer to memory address of data direction
register for port B
status    EQU      0x03 ; Pointer to memory address where status register is
stored
RP0       EQU      5    ; Bit number of RP0 in status register
RP1       EQU      6    ; Bit number of RP1 in status register
inner     EQU      0x1C ; RAM memory address of data
outer     EQU      0x1D ; RAM memory address of data

          ORG      0x000 ; Start the program at 0x000

; Subroutine:      main
; Description:     The main function for the program. Starts by initialising the device
registers
;                  as well as enabling port B and setting directionality of the pins.
The RAM block
;                  that is utilised is then initialised with values for the LCD display.
Then the
;                  main loop begins which iterates over the RAM memory block, displaying
each LCD
;                  value for ~200ms and resetting once it has reached the end of the
list of values.
; Register Usage:
;      w      -> Working register, temporary storage for values and calculations
;      status -> For accessing status of the ALU, RESET and, the bank select bit for
data memory
;      trisB   -> For setting the directionality of pins on Port B
;      portB   -> For outputting data on pins within Port B
;      FSR     -> For indirect data memory addressing
;      INDF    -> Uses the FSR to access data at memory locations (not a physical
register)

main
          BCF      status,RP0 ; Clear RP0 bit of status register
          BCF      status,RP1 ; Clear RP1 bit of status register
```

```

        BSF      status,RP0      ; Set RP0 bit of status register
        MOVLW    0x00            ; Move value 0 to w
        MOVWF    trisB           ; Set portB as all output pins
        BCF      status,RP0      ; Clear RP0 bit of status register
        CLRF     portB           ; Clear current pin values on portB

        CALL     init_ram        ; Call init_ram to generate data

reset    MOVLW    hex_values      ; Move start address of RAM into w
        MOVWF    FSR             ; Move address in w to FSR
loop     MOVF     INDF, 0         ; Move the value at FSR (which is in INDF) to w
        MOVWF    portB          ; Move value in w to port b -> display on LCD screen
        CALL     delay           ; Delay the program for ~200ms
        INCF     FSR             ; Increment the pointer to the next piece of memory

in RAM   MOVF     FSR, 0          ; Load the address in FSR in w
        XORLW    hex_end         ; XOR the address in w with the end of data in RAM
        BTFSS    status,Z        ; If the address are equal, Z bit of status register
is high (skip next instruction, replace with NOP)
        GOTO     loop           ; Jump back to 'loop' label to display the next
number on the LCD display
        GOTO     reset          ; Jump back to 'reset' label to start counting up
again
        RETURN                  ; Return from subroutine

; Subroutine:      delay
; Description:     Keeps the processor busy for ~200ms
; Register Usage:
;     w    -> Working register, temporary storage of
;             counters.
;     outer -> Outer loop counter - stored in RAM
;     inner -> Inner loop counter - stored in RAM

delay
        MOVLW    0xFF           ; w = 255
        MOVWF    outer          ; outer = w -> outer loop counter
loop_o   MOVLW    0xFF           ; w = 255
        MOVWF    inner          ; inner = w -> inner loop counter
loop_i   DECFSZ   inner, f        ; inner--
        GOTO     loop_i          ; if inner != 0 -> loop inner
        DECFSZ   outer, f        ; outer--
        GOTO     loop_o          ; if outer != 0 -> loop outer
        RETURN                  ; Return from subroutine

; Subroutine:      init_ram
; Description:     Initialises the top 16 bytes of RAM to contain
;                 all of the hexadecimal values that need to be
;                 sent on portB to correctly light different
;                 segments of the LCD display.
; Register Usage:
;     w    -> Working register, stores the literal hexadecimal
;                 value that is being stored in RAM.
;     hex_values -> Contains the address of the beginning of RAM.

init_ram
        MOVLW    0x40           ; Hex value for: 0
        MOVWF    hex_values     ; Store value at: 0x0C
        MOVLW    0x79           ; Hex value for: 1

```

```
MOVWF    hex_values+0x01 ; Store value at: 0x0D
MOVLW    0x24             ; Hex value for: 2
MOVWF    hex_values+0x02 ; Store value at: 0x0E
MOVLW    0x30             ; Hex value for: 3
MOVWF    hex_values+0x03 ; Store value at: 0x0F
MOVLW    0x19             ; Hex value for: 4
MOVWF    hex_values+0x04 ; Store value at: 0x10
MOVLW    0x12             ; Hex value for: 5
MOVWF    hex_values+0x05 ; Store value at: 0x11
MOVLW    0x02             ; Hex value for: 6
MOVWF    hex_values+0x06 ; Store value at: 0x12
MOVLW    0x78             ; Hex value for: 7
MOVWF    hex_values+0x07 ; Store value at: 0x13
MOVLW    0x00             ; Hex value for: 8
MOVWF    hex_values+0x08 ; Store value at: 0x14
MOVLW    0x10             ; Hex value for: 9
MOVWF    hex_values+0x09 ; Store value at: 0x15
MOVLW    0x08             ; Hex value for: A
MOVWF    hex_values+0x0A ; Store value at: 0x16
MOVLW    0x03             ; Hex value for: b
MOVWF    hex_values+0x0B ; Store value at: 0x17
MOVLW    0x46             ; Hex value for: C
MOVWF    hex_values+0x0C ; Store value at: 0x18
MOVLW    0x21             ; Hex value for: d
MOVWF    hex_values+0x0D ; Store value at: 0x19
MOVLW    0x06             ; Hex value for: E
MOVWF    hex_values+0x0E ; Store value at: 0x1A
MOVLW    0x0E             ; Hex value for: F
MOVWF    hex_values+0x0F ; Store value at: 0x1B
RETURN                                ; Return from subroutine

END
```

Appendix B: Two Buttons Raw Assembly

```
LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

portB      EQU      0x06      ; Pointer to memory address this is written to port B
trisA      EQU      0x05      ; Pointer to memory address of data direction
register for port B
portA      EQU      0x05      ; Pointer to memory address that is written to port A
trisB      EQU      0x06      ; Pointer to memory address of data direction
register for port A
status     EQU      0x03      ; Pointer to memory address where status register is
stored
RP0        EQU      5         ; Bit number of RP0 in status register
RP1        EQU      6         ; Bit number of RP1 in status register
letter_L   EQU      0x47      ; Hexadecimal value to output to display 'L'
letter_r   EQU      0x4E      ; Hexadecimal value to output to display 'r'

ORG        0x000          ; Start the program at 0x000

; Subroutine:      main
; Description:     The main function for the program. Starts by initialising the device
registers
;                as well as enabling port A & B and setting directionality of the
pins. The main
;                loop then iterates over the two pins polling to see if the button has
been activated.
;                If it has, then it modifies the w register to output the correct
character and then
;                outputs this to port B (and the display) and loops back to the
beginning.
; Register Usage:
;      w      -> Working register, temporary storage for values and calculations
;      status -> For accessing status of the ALU, RESET and, the bank select bit for
data memory
;      trisB  -> For setting the directionality of pins on Port B
;      portB  -> For outputting data on pins within Port B
;      trisA  -> For setting the directionality of pins on Port A
;      portA  -> For outputting data on pins within Port A

main
BCF      status,RP0      ; Clear RP0 bit of status register
BCF      status,RP1      ; Clear RP1 bit of status register
BSF      status,RP0      ; Set RP0 bit of status register
MOVLW    0x00            ; Move value 0 to w
MOVWF    trisB           ; Set portB as all output pins

MOVLW    0x03            ; Move value 3 to w
MOVWF    trisA           ; Set portA pin 0 and 1 to input pins
BCF      status,RP0      ; Clear RP0 bit of status register
CLRF     portA           ; Clear current pin values on portA
CLRF     portB           ; Clear current pin values on portB
```

```

loop      MOVLW    0xFF          ; Move value 255 to w
          BTFSS    portA,0        ; Check if PIN A0 is high if it is, NOP next
instruction, if low execute next instruction
          ANDLW    letter_L       ; AND 0xFF and 0x47 and store result in w (0x47)
          BTFSS    portA,1        ; Check if PIN A1 is high if it is, NOP next
instruction, if low execute next instruction
          ANDLW    letter_r       ; AND 0xFF and 0x4E and store result in w (0x4E)
          MOVWF    portB          ; Output w to port b to display 'L' or 'r'
          GOTO     loop           ; Loop back to the beginning of the logic

END

```

Appendix C: Latching Buttons Raw Assembly

```
LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

portB     EQU          0x06      ; Pointer to memory address this is written to port B
trisA     EQU          0x05      ; Pointer to memory address of data direction
register for port B
portA     EQU          0x05      ; Pointer to memory address that is written to port A
trisB     EQU          0x06      ; Pointer to memory address of data direction
register for port A
status    EQU          0x03      ; Pointer to memory address where status register is
stored
RP0       EQU          5         ; Bit number of RP0 in status register
RP1       EQU          6         ; Bit number of RP1 in status register
all_on    EQU          0x00      ; Hexadecimal value to output to turn on all LED
segments
all_off   EQU          0xFF      ; Hexadecimal value to output to turn off all LED
segments

ORG       0x000          ; Start the program at 0x000

; Subroutine:      main
; Description:     The main function for the program. Starts by initialising the device
registers
;                as well as enabling port A & B and setting directionality of the
pins. The main
;                loop then iterates over the two pins polling to see if the button has
been activated.
;                If it has, then it modifies the w register to output the correct
light sequence and then
;                outputs this to port B (and the display) and loops back to the
beginning.
; Register Usage:
;      w      -> Working register, temporary storage for values and calculations
;      status -> For accessing status of the ALU, RESET and, the bank select bit for
data memory
;      trisB  -> For setting the directionality of pins on Port B
;      portB  -> For outputting data on pins within Port B
;      trisA  -> For setting the directionality of pins on Port A
;      portA  -> For outputting data on pins within Port A

main
BCF      status,RP0      ; Clear RP0 bit of status register
BCF      status,RP1      ; Clear RP1 bit of status register
BSF      status,RP0      ; Set RP0 bit of status register
MOVLW    0x00            ; Move value 0 to w
MOVWF    trisB           ; Set portB as all output pins
```

```

        MOVLW    0x03            ; Move value 3 to w
        MOVWF    trisA          ; Set portA pin 0 and 1 to input pins
        BCF      status,RP0     ; Clear RP0 bit of status register
        CLRF     portA          ; Clear current pin values on portA
        MOVLW    all_off        ; Move value to turn all LEDs off to w
        MOVWF    portB          ; Output the value in w to port B

loop    BTFSS    portA,0        ; Check if PIN A0 is high if it is, NOP next
instruction, if low execute next instruction
        MOVLW    all_on         ; Move literal 0x00 to register w
        BTFSS    portA,1        ; Check if PIN A1 is high if it is, NOP next
instruction, if low execute next instruction
        MOVLW    all_off        ; Mover literal 0xFF to register w
        MOVWF    portB          ; Output w to port b to display all or none of the
LED segments
        GOTO     loop           ; Loop back to the beginning of the logic

END

```

Appendix D: Button Controlled Counting

Raw Assembly

```

LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

          UDATA_SHR    ; Declares the beginning of a section of shared
uninitialised data
hex_values RES      0x05 ; Pointer to start of a memory block that has been
reserved (size of 16 bytes)
hex_end    ; Pointer to memory address directly after data being
used

portB      EQU      0x06 ; Pointer to memory address this is written to port B
trisA      EQU      0x05 ; Pointer to memory address of data direction
register for port B
portA      EQU      0x05 ; Pointer to memory address that is written to port A
trisB      EQU      0x06 ; Pointer to memory address of data direction
register for port A
status     EQU      0x03 ; Pointer to memory address where status register is
stored
RP0        EQU      5    ; Bit number of RP0 in status register
RP1        EQU      6    ; Bit number of RP1 in status register
inner      EQU      0x1C ; RAM memory address of data
outer      EQU      0x1D ; RAM memory address of data

          ORG      0x000 ; Start the program at 0x000

; Subroutine:      main
; Description:      The main function for the program. Starts by initialising the device
registers
;                  as well as enabling port B & A and setting directionality of the
pins. The RAM block
;                  that is utilised is then initialised with values for the LCD display.
; Then the
;                  main loop begins which polls the two buttons on the circuit and calls
; appropriate
;                  subroutines dependant on which button is high.
; Register Usage:
;      w  -> Working register, temporary storage for values and calculations
;      trisB -> For setting the directionality of pins on Port B
;      portB -> For outputting data on pins within Port B

main
BCF      status,RP0    ; Clear RP0 bit of status register
BCF      status,RP1    ; Clear RP1 bit of status register
BSF      status,RP0    ; Set RP0 bit of status register
MOVLW    0x00          ; Move value 0 to w
MOVWF    trisB         ; Set portB as all output pins

```

```

        MOVLW    0x03            ; Move value 3 to w
        MOVWF    trisA          ; Set portA pin 0 and 1 to input pins
        BCF      status,RP0     ; Clear RP0 bit of status register
        CLRF     portA          ; Clear current pin values on portA
        CLRF     portB          ; Clear current pin values on portB

        MOVLW    0xFF           ; Move the literal 0xFF into w
        MOVWF    portB          ; Output w to port B

        CALL     init_ram       ; Call init_ram to generate data

loop    BTFSS    portA,0        ; Check if PIN A0 is high if it is, NOP next
instruction, if low execute next instruction
        CALL     count_up      ; Call (branch) to the 'count_up' subroutine
        BTFSS    portA,1        ; Check if PIN A1 is high if it is, NOP next
instruction, if low execute next instruction
        CALL     count_down    ; Call (branch) to the 'count_down' subroutine
        GOTO     loop          ; Loop back to the beginning of the logic
        RETURN                 ; Return from subroutine

; Subroutine: count_up
; Description: Iterates over the initialised RAM memory address containing hex
values to output
;              which display values 5-9 inclusive. This subroutine counts through
these values
;              in the positive direction.
; Register Usage:
;      w    -> Working register, temporary storage for values and calculations
;      status -> For accessing status of the ALU, RESET and, the bank select bit for
data memory
;      FSR   -> For indirect data memory addressing
;      INDF  -> Uses the FSR to access data at memory locations (not a physical
register)

count_up
        MOVLW    hex_values     ; Move start address of RAM into w
        MOVWF    FSR            ; Move address in w to FSR
up_loop MOVF     INDF,0          ; Move the value at FSR (which is in INDF) to w
        MOVWF    portB         ; Move value in w to port b -> display on LCD screen
        CALL     delay          ; Delay the program for ~200ms
        INCF     FSR            ; Increment the pointer to the next piece of memory

in RAM
        MOVF     FSR,0          ; Load the address in FSR in w
        XORLW    hex_end        ; XOR the address in w with the end of data in RAM
        BTFSS    status,Z       ; If the address are equal, Z bit of status register
is high (skip next instruction, replace with NOP)
        GOTO     up_loop        ; Jump back to 'up_loop' label to display the next
number on the LCD display
        RETURN                 ; Return from subroutine

; Subroutine: count_down
; Description: Iterates over the initialised RAM memory address containing hex
values to output
;              which display values 9-5 inclusive. This subroutine counts through
these values
;              in the negative direction.
; Register Usage:
;      w    -> Working register, temporary storage for values and calculations

```

```

;      status  -> For accessing status of the ALU, RESET and, the bank select bit for
data memory
;      FSR      -> For indirect data memory addressing
;      INDF     -> Uses the FSR to access data at memory locations (not a physical
register

count_down
    MOVLW    hex_values+0x04 ; Move end address of RAM into w
    MOVWF    FSR             ; Move address in w to FSR
down_loop   MOVF    INDF,0    ; Move the value at FSR (which is in INDF) to w
    MOVWF    portB          ; Move the value in w to port b -> display on LCD
screen
    CALL     delay          ; Delay the program for ~200ms
    DECF     FSR            ; Decrement the pointer to the next piece of memory
in RAM
    MOVF     FSR,0          ; Load the address in FSR in w
    XORLW    hex_values-0x01 ; XOR the address in w with the address before the
beginning of RAM
    BTFSS    status,Z       ; If the address are equal, Z bit of status register
is high (skip next instruction, replace with NOP)
    GOTO     down_loop      ; Jump back to 'down_loop' label to display the next
number on the LCD display
    RETURN                  ; Return from subroutine

; Subroutine:    delay
; Description:    Keeps the processor busy for ~200ms
; Register Usage:
;      w  -> Working register, temporary storage of
;             counters.
;      outer -> Outer loop counter - stored in RAM
;      inner  -> Inner loop counter - stored in RAM

delay
    MOVLW    0xFF          ; w = 255
    MOVWF    outer         ; outer = w -> outer loop counter
loop_o      MOVLW    0xFF          ; w = 255
    MOVWF    inner         ; inner = w -> inner loop counter
loop_i      DECFSZ    inner, f     ; inner--
    GOTO     loop_i         ; if inner != 0 -> loop inner
    DECFSZ    outer, f     ; outer--
    GOTO     loop_o         ; if outer != 0 -> loop outer
    RETURN                  ; Return from subroutine

; Subroutine:    init_ram
; Description:    Initialises the top 5 bytes of RAM to contain
;                all of the hexadecimal values that need to be
;                sent on portB to correctly light different
;                segments of the LCD display.
; Register Usage:
;      w  -> Working register, stores the literal hexadecimal
;             value that is being stored in RAM.
;      hex_values -> Contains the address of the beginning of RAM.

init_ram
    MOVLW    0x12          ; Hex value for: 5
    MOVWF    hex_values    ; Store value at: 0x11
    MOVLW    0x02          ; Hex value for: 6
    MOVWF    hex_values+0x01 ; Store value at: 0x12

```

```
MOVLW    0x78          ; Hex value for: 7
MOVWF    hex_values+0x02 ; Store value at: 0x13
MOVLW    0x00          ; Hex value for: 8
MOVWF    hex_values+0x03 ; Store value at: 0x14
MOVLW    0x10          ; Hex value for: 9
MOVWF    hex_values+0x04 ; Store value at: 0x15
RETURN                   ; Return from subroutine
```

```
END
```

Appendix E: Piezo Buzzer Raw Assembly

```
LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

portB      EQU      0x06      ; Pointer to memory address this is written to port B
trisA      EQU      0x05      ; Pointer to memory address of data direction
register for port B
portA      EQU      0x05      ; Pointer to memory address that is written to port A
trisB      EQU      0x06      ; Pointer to memory address of data direction
register for port A
status     EQU      0x03      ; Pointer to memory address where status register is
stored
RP0        EQU      5        ; Bit number of RP0 in status register
RP1        EQU      6        ; Bit number of RP1 in status register
delay_c    EQU      0x1D      ; RAM memory address of data
buzz_on    EQU      0x80      ; Value to output to port B to turn buzzer on
buzz_off   EQU      0x00      ; Value to output to port B to turn buzzer off

ORG        0x000          ; Start the program at 0x000

; Subroutine:      main
; Description:     The main function for the program. Starts by initialising the device
registers
;                  as well as enabling port B & A and setting directionality of the
pins. It then
;                  simply loops and turns the buzzer on and off with a delay in between.
Generating
;                  an audible noise from the buzzer.
; Register Usage:
;   w  -> Working register, temporary storage for values and calculations
;   trisB -> For setting the directionality of pins on Port B
;   portB -> For outputting data on pins within Port B

main
    BCF      status,RP0      ; Clear RP0 bit of status register
    BCF      status,RP1      ; Clear RP1 bit of status register
    BSF      status,RP0      ; Set RP0 bit of status register
    MOVLW    0x00            ; Move value 0 to w
    MOVWF    trisB           ; Set portB as all output pins

    MOVLW    0x03            ; Move value 3 to w
    MOVWF    trisA           ; Set portA pin 0 and 1 to input pins
    BCF      status,RP0      ; Clear RP0 bit of status register
    CLRF     portA           ; Clear current pin values on portA
    CLRF     portB           ; Clear current pin values on portB

    MOVLW    0xFF            ; Move the literal 0xFF into w
    MOVWF    portB           ; Output w to port B

loop
    MOVLW    buzz_on         ; Move literal 0x80 into w
    MOVWF    portB           ; Output w to port B (buzzer on)
```

```

        CALL    delay            ; Call delay
        MOVLW   buzz_off        ; Move literal 0x00 into w
        MOVWF   portB           ; Output w to port B (buzzer off)
        CALL    delay            ; Call delay
        GOTO    loop            ; Jump to 'loop' label

        RETURN                  ; Return from subroutine

; Subroutine:    delay
; Description:    Keeps the processor busy for ~0.78ms
; Register Usage:
;     w  -> Working register, temporary storage of
;           counters.
;     delay_c -> Outer loop counter - stored in RAM

delay
        MOVLW   0xFF            ; w = 255
        MOVWF   delay_c         ; delay_c = w -> loop counter
loop_d  DECFSZ   delay_c, f      ; delay_c--
        GOTO    loop_d          ; if delay_c != 0 -> loop_d
        RETURN                  ; Return from subroutine

END

```

Appendix F: Piezo Buzzer PWM Raw Assembly

```
LIST      p=16F84      ; Switch on output to the .LST file
RADIX     hex          ; What base is being used to represent numbers

#include p16f84a.inc    ; Find include file for the microprocessor and
include it here

pwm_time   EQU      0x0C      ; RAM memory address of data
delay_c    EQU      0x0D      ; RAM memory address of data
buzz_on     EQU      0x80      ; Value to output to port B to turn buzzer on
buzz_off    EQU      0x00      ; Value to output to port B to turn buzzer off
buzz_state  EQU      0x0E      ; RAM memory address of data

reset       CODE      0x000    ; Start the program at 0x000
           GOTO      main      ; Jump to 'main' label

interrupt    CODE      0x004    ; Jump to 'handler' label
           GOTO      handler

main

           BSF       INTCON, T0IE ; Enable the Timer0 interrupt
           BSF       INTCON, GIE  ; Enable global interrupts

           BANKSEL   TRISB        ; Select bank for TRISB
           MOVLW     0x00          ; Move literal 0x00 to w
           MOVWF     TRISB        ; Set all ports on B to be outputs

           BANKSEL   OPTION_REG    ; Select bank for OPTION_REG
           MOVF      OPTION_REG,W  ; Move OPTION_REG into w
           ANDLW     0xC0          ; Mask off bits 7 and 6
           IORLW     0x04          ; Set prescaler to 4 (1:32)
           MOVWF     OPTION_REG    ; Move w to OPTION_REG

           BANKSEL   TMR0          ; Select bank for TMR0
           MOVLW     0xFF          ; Move literal 0xFF into w
           MOVWF     TMR0          ; Move w to TMR0

           BANKSEL   TMR0          ; Select bank for TMR0
           MOVLW     0xFF          ; Move literal 0xFF into w
           MOVWF     pwm_time      ; Move w to pwm_time

loop        GOTO      loop        ; Infinite loop

handler     BTFSS     INTCON, T0IF ; Test if interrupt is from Timer0
           RETFIE     ; If it isn't, return from interrupt
           BCF       INTCON, T0IF ; Clear the interrupt flag for Timer0
           BANKSEL   TMR0          ; Select bank for TMR0
           MOVF      pwm_time,W    ; Move the value at pwm_time into w
           BTFSS     buzz_state,0  ; Test the LSB of buzz_state: 0 -> execute next
instruction. 1 -> NOP next instruction
           SUBLW     0xFF          ; Subtract pwm_time from 255 (255 - pwm_time)
```

```

        MOVWF    TMRO            ; Move value in w to Timer0
        DECF     pwm_time,1      ; Decrement pwm_time and store back in pwm_time

        MOVLW    buzz_on        ; Move literal buzz_on into w
        BTFSS    buzz_state,0    ; Test the LSB of buzz_state: 0 -> execute next
instruction. 1 -> NOP next instruction
        MOVLW    buzz_off       ; Move literal buzz_off into w
        BANKSEL  PORTB          ; Select bank for PORTB
        MOVWF    PORTB          ; Move w to PORTB

        MOVLW    0x01           ; Move literal 0x01 into w
buzz_state XORWF    buzz_state,1 ; XOR w with buzz_state, store result back into
        RETFIE                  ; Return from interrupt

END

```

References

- [1] AMD, “AMD,” 24 November 2016. [Online]. Available: https://adaptivesupport.amd.com/s/question/oD52E00006hpboUSAQ/microcontrollers-getting-started-with-pic16f84a?language=en_US. [Accessed 02 December 2024].
- [2] CADENCE PCB SOLUTIONS, “Cadence,” 2022. [Online]. Available: <https://resources.pcb.cadence.com/blog/2022-the-role-of-oscillators-in-microcontrollers>. [Accessed 02 December 2024].
- [3] Electronics Tutorials, “ElectronicsTutorials 7-segment Display,” 2022. [Online]. Available: <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>. [Accessed 02 December 2024].
- [4] multcomp, “BC107/BC108 Series,” 2006.
- [5] APC International, LTD, “Piezoelectric Buzzers,” [Online]. Available: <https://www.americanpiezo.com/products-services/buzzers/>. [Accessed 02 December 2024].
- [6] EFY Bureau, “7805 Voltage Regulator IC Pinout, Circuit, and Applications,” 11 June 2024. [Online]. Available: <https://www.electronicsforu.com/technology-trends/learn-electronics/7805-ic-voltage-regulator>. [Accessed 02 December 2024].

Final

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

86/100

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

BRADLEY GILLIES

428

ENGAGEMENT WITH LITERATURE

 Good implementation of IEEE referencing - note that for first class marks for EWL, even more sources could have been included and should cover as many relevant areas as possible - e.g. more info on the voltage reg, why the capacitors are included etc etc.

68

KNOWLEDGE AND UNDERSTANDING

 A very good level of K&U demonstrated - more EWL sources would have given an even higher mark here

75

COGNITIVE AND INTELLECTUAL

 Superb programming skills for this level of module!

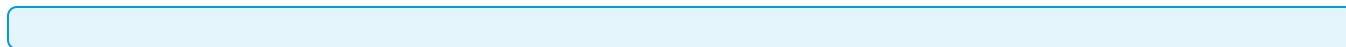
95

PRACTICAL APPLICATION

 Excellent work.

100

TRANSFERABLE SKILLS



Very well written and presented.

90