

Caitlin Lonsdale - ACDA4004 - Assignment

by Caitlin Lonsdale

Submission date: 04-Jan-2025 01:17PM (UTC+0000)

Submission ID: 247118514

File name: 51881_Caitlin_Lonsdale_Caitlin_Lonsdale_-_ACDA4004_-_Assignment_621295_892824861.docx
(3.64M)

Word count: 9306

Character count: 46087

***Using and Programming the 7
Segment Display Prototyping
Circuit***

ACDA4004

Caitlin Lonsdale

2306892

Word Count: 3230

Contents

Abstract

Introduction

Section 1: Circuit Simulation

Section 2: Character Display

Section 3: Assembly Programming

- Task 1
- Task 2
- Task 3
- Task 4
- Task 5
- Task 6

Conclusion

References

Appendix

Abstract

This report details the simulation and exploration of a circuit centered around a PIC16F84A microcontroller. The individual components of this circuit; their functions and implementations, are evaluated in section one. Following on in section two, the report specifically explores the use of a 7-segment LED display to display a variety of characters, for example hexadecimal counting, in combination with the microcontroller. The third section explores six example tasks that can be programmed on the microcontroller, using assembly language, and interact with the embedded circuit. One of the key examples is the generation of a varying PWM (pulse width modulation) signal in software.

Introduction

PIC (programmable integrated circuit) microcontrollers are popularly used in a variety of electronic circuits: from simplistic driving of LEDs (light emitting diodes) through to automotive and industrial applications [1]. The PIC16F84A is an example of one of these, and belongs to the 8-bit microcontroller family produced by Microchip Technology. It can be used as part of a circuit, as is explored in Section 1 and onwards. For example, push buttons can be used as inputs for the PIC, and a 7 segment display and sounder used as outputs. These functions are defined in Assembly software, compatible with the PIC hardware, and can then be demonstrated in a theoretical simulation.

Section 1: Circuit Simulation

Below is shown a circuit diagram of the PIC16F84A assignment board [2], followed by the produced Proteus simulation of the same circuit. The constituent components and design and then discussed.

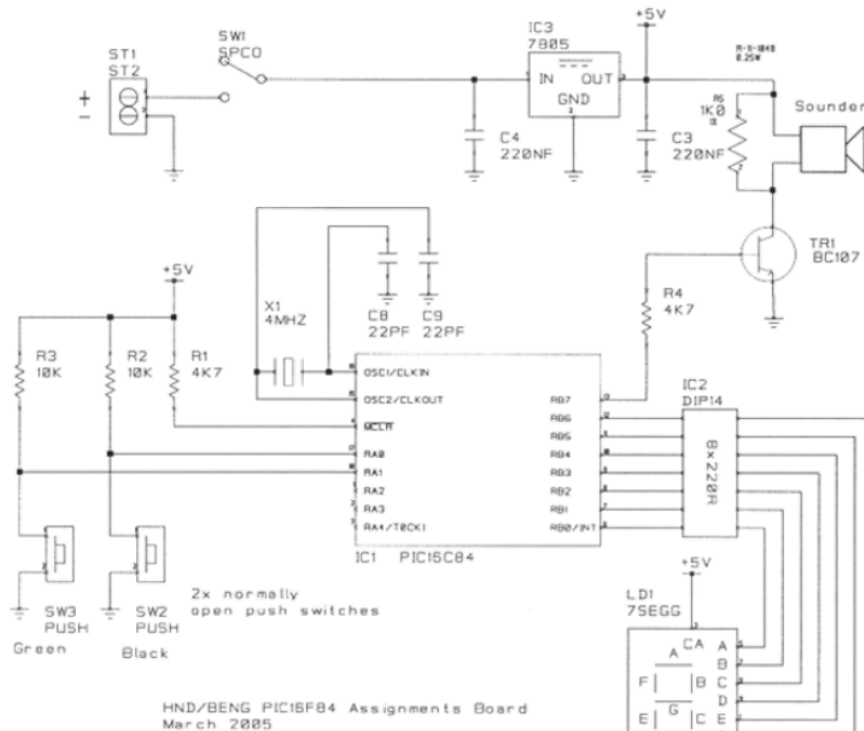


Figure 1 - Assignment Board Circuit Diagram

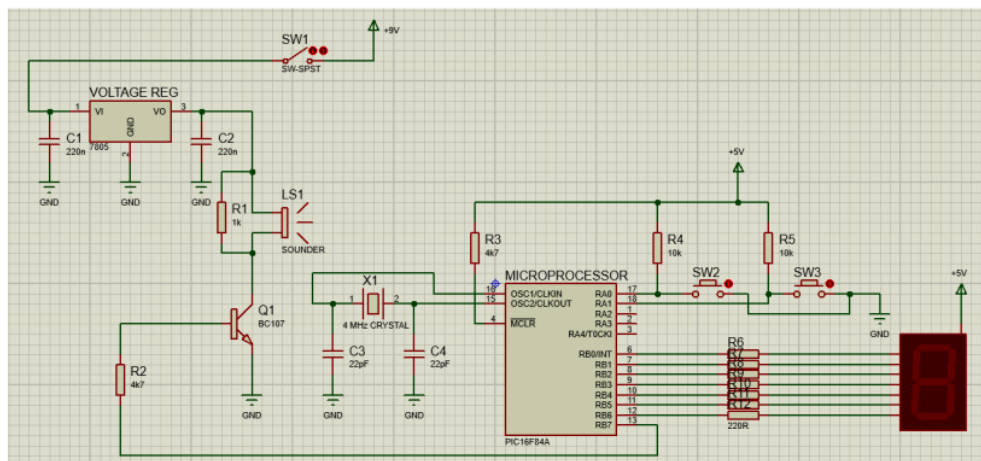


Figure 2 - Proteus Circuit Simulation

PIC16F84A Microcontroller

The circuit is based around the PIC16F84A microcontroller – an 8 bit PIC (programmable integrated circuit) produced by Microchip [1]. It is programmable, with 35 instructions, and has a 200ns execution rate. The PIC has 1.75kB program memory, 68B RAM (random access memory), and 64B EEPROM data (electrically erasable programmable read-only memory). It has 18 pins, 13 of which are I/O (input or output) and configurable in software by the user – labelled as port A and port B. There are also the Vss and Vdd pins, which give the ground reference and positive supply (2-6v) for the logic and I/O pins. Whilst these are fundamental for the PIC operation in practice, they are not present in the circuit diagram or simulation. Furthermore, there are the oscillator input and output ports, which direct the program timing.

There are various connections of its ports discussed below, in relation to the other circuit components. In addition, it must be noted that the inverting MCLR pin (master clear) has a 4.7kΩ pull-up resistor (R1) to +5v. Since the pin resets the device when low; the resistor ensures there is no reset as it sets the input on the pin to +5v (high). Pins RA2, RA3 and RA4 are unused in this circuit.

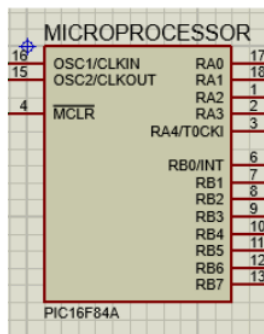


Figure 3 - Simulation Microprocessor

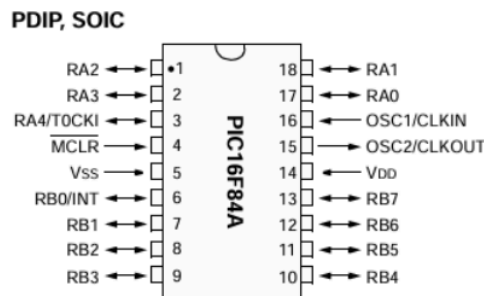


Figure 4 - Microprocessor Pin Diagram

Voltage Supply and Switch

Starting at the top left of both the diagram and simulated circuit, we have a voltage supply and switch to control current flow (shown below). The power is a +9v DC supply, and the switch is a basic contact design – if closed then current flows, if open then no current passes through.

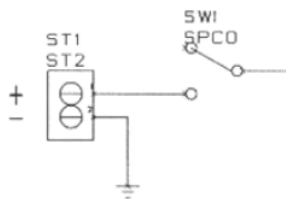


Figure 5 - Supply and Switch Diagram

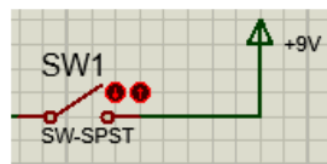


Figure 6 - Simulation Supply and Switch

7805 Voltage Regulator

The 7805 (part of the 78xx series) voltage regulator is featured, with a 220nF capacitor (C4 and C3 on the diagram) either side. It has 3 terminals; an input, and output, and a ground connection. The last 2 digits: 05, denote the output voltage of +5v [3]. The capacitors are referred to as 'bypass capacitors', and provide stability on the output and input. A voltage regulator, such as the one below, is used in a circuit to ensure a compatible voltage is supplied to the microcontroller and other components – in this case at +5v DC. A component image is also given [4].

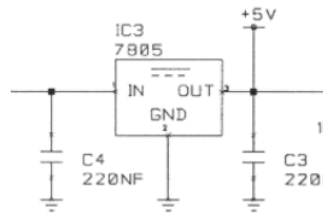


Figure 7 - 7805 Diagram

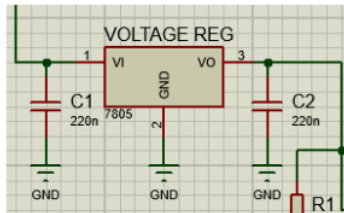


Figure 8 - Simulation 7805



Figure 9 - 7805 Component

Piezo Sounder

Following the 7805, there is a 'Sounder' on the diagram – implemented in the circuit as a piezo buzzer. A piezo buzzer is a low cost, audio-emitting device which uses the vibration of a piezoelectric crystal – under deformation by an applied voltage – to produce sound [5]. In the case of a DC circuit as in our example, the buzzer requires an on/off cycle mechanism. This is provided the connected transistor (see BC107 below) and the 1kΩ resistor in parallel. The specifications, whilst not given in the circuit diagram, can be derived from the other components: +5v DC voltage, externally driven. The example component given below has a 91dB sound level [6].

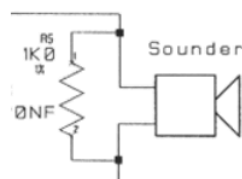


Figure 10 - Sounder Diagram

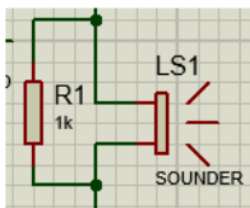


Figure 11 - Simulation Piezo Buzzer



Figure 12 - Piezo Buzzer Component

BC107 NPN Transistor

As mentioned with regards to the piezo buzzer, the BC107 transistor is used as an on/off switch for the buzzer's cycle mechanism. The BC107 is an NPN bipolar junction transistor (BJT), with 3 pins [7] [8]. The top pin, drawing from the buzzer, is the collector pin. The side pin, which takes its input from pin RB7 of the microcontroller, is the base. Lastly, the bottom pin is the emitter and hence is connected to ground. Since RB7 is an output pin on the microcontroller, the microcontroller can be programmed so to produce an on/off pulsed output of +5v (or logic 1 - see Section 3) - setting the base pin on/off. When power is supplied to the base pin, current flows from the collector to the emitter, causing the piezo buzzer to sound. Alternatively, when there is no power to the base pin, no current flows from the collector – resulting in no sound from the buzzer.

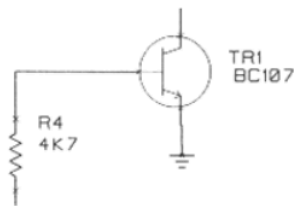


Figure 33 - BC107 Diagram

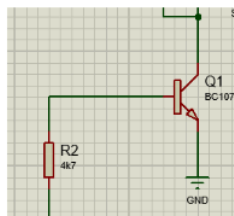


Figure 14 - Simulation BC107



Figure 15 - BC107 Component

Quartz Crystal Oscillator

Moving on to the quartz crystal oscillator; connected to the OSC1/CLKIN and OSC2/CLKOUT pins of the microcontroller, which sets the microprocessor clock (see PIC16F84A above). The example in the circuit, and shown below, has an oscillating frequency of 4 MHz [9]. Quartz crystal oscillators are considered stable, and also operate by the piezoelectric effect – generating a voltage at a designated frequency by the effect of an applied electric field [10]. The 22pF capacitors connected to ground on either side of the oscillator (C9 and C8 in the diagram) cause the input and output to be in phase by phase shifting, which is necessary for sustained clock operation [11].

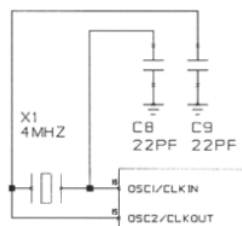


Figure 16 - Oscillator Diagram

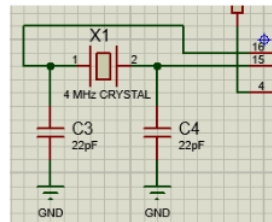


Figure 17 - Simulation Oscillator



Figure 18 - Oscillator Component

Push Buttons

Also, within the circuit are two push buttons; SW2 button inputs to pin RA0, and SW3 button inputs to pin RA1 of the microcontroller. From this, the RA0 and RA1 can be set as input pins on the microcontroller – meaning that a button output (see Section 3) can be used as a stimulus for the microcontroller – such as a program interrupt [2]. Both push buttons have 10k Ω pull-up resistors (R3 and R2) to +5v, and are also connected to ground. The pullup resistor on a button means that when it's open, the +5v rail supplies power to the microcontroller pin. On the other hand, when the button is pressed/closed the +5v passes through the button to ground (following the path of least resistance). As a result, no power is supplied to the microcontroller pin in this circumstance [12].

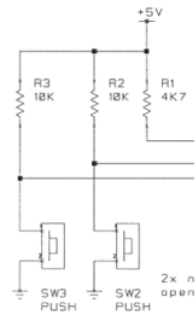


Figure 19 - Buttons Diagram

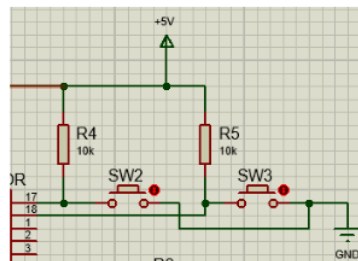


Figure 20 - Simulation Buttons

DIP14 Resistor Array

The microcontroller output pins RB0-RB6 are connected to a DIP14 resistor array, as shown below [13]. The array is composed of 6 parallel 220 Ω resistors, which is used to regulate the output signals from the microprocessor, before they input to the display (see 7 Segment Display below).

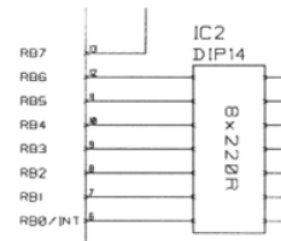


Figure 21 - Resistor Array Diagram

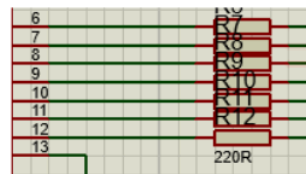


Figure 22 - Simulation Resistors

Common Anode 7-Segment Display

A 7-segment display, with a common anode configuration, is used in the circuit [14]. It receives inputs from the pins RB0-RB6, via the DIP14 resistor array, and a power supply of +5v. Each input pin corresponds to one of the 7 display segments, and is controlled on an on/off power basis [15]. These display segments are illuminated by LEDs, and are arranged to form various decimal and hexadecimal digits. Since the example used has a common anode display, all seven segments have their anodes connected together to the +5v (logic 1). As a result, segment LEDs are illuminated when a low signal is applied to their corresponding pins (logic 0) – which can be achieved by setting the connected microcontroller pin output to zero. As a result, different logic sequences can be applied to the microcontroller pins in order to display different characters on the 7-segment display.

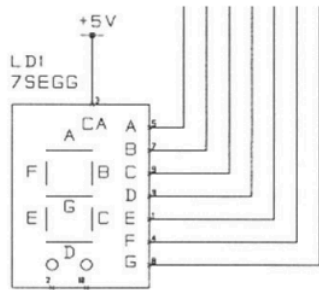


Figure 23 - Display Diagram

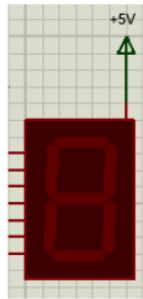


Figure 24 - Simulation Display

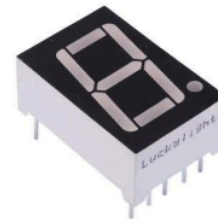


Figure 25 - Display Component

Section 2: Character Display

The 7-segment display has 7 input pins a-g which are controlled by the microcontroller pins RB0 to RB6. This relationship means a binary value can be written across port B of the microcontroller in order to illuminate the display. By evaluating which display input pins correspond to each LED segment, we can determine binary sequences in order to display each of the hexadecimal values 0-F on the 7-segment display (see table below). Pin a is shown by the least significant bit (microprocessor pin RB0), through to pin g as the 6th indexed bit (pin RB6). Since pin RB7 (most significant bit), the eighth pin on port B of the microcontroller, is unused for this application it will be set low.

It must be noted that since the 7 segment display used has a common anode configuration, segments must have a logic 0 to illuminate (see Section 1). As a result, if a segment is required its value in the sequence is zero. If a segment is off then it's represented by a binary value 1 [15].

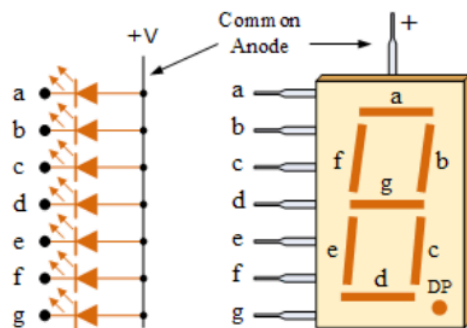


Figure 26 - 7 Segment Display Diagram

Hexadecimal Value	Display Pins	Binary Sequence
0	A B C D E F	0100 0000
1	B C	0111 1001
2	A B D E G	0010 0100
3	A B C D G	0011 0000
4	B C F G	0001 1001
5	A C D F G	0001 0010
6	A C D E F G	0000 0010
7	A B C	0111 1000
8	A B C D E F G	0000 0000
9	A B C D F G	0001 0000
A	A B C E F G	0000 1000
b	C D E F G	0000 0011
C	A D E F	0100 0110
d	B C D E G	0010 0001
E	A D E F G	0000 0110
F	A E F G	0000 1110

Section 3: Assembly Programming

Task 1

Firstly, the PIC is programmed “so that the 7-segment display counts from 0 to F in hexadecimal numbers in 0.2 second increments”. These characters can be displayed but outputting a bit sequence each to the 7-segment display, via port B of the microcontroller, as set out in section 2. The flowchart below demonstrates how indexing can be used to increment through the characters, and reset at the maximum.

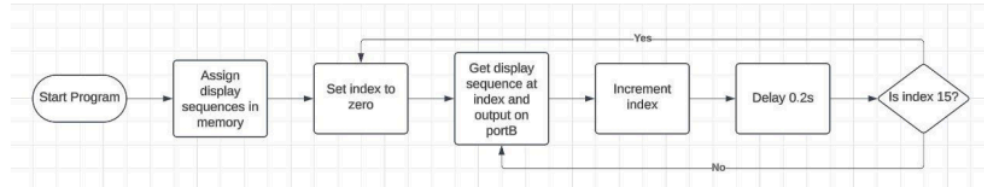


Figure 27 - Task 1 flowchart

Using assembly code, the microprocessor type and supporting file must first be referenced, then memory allocated – this is the initialisation process. Task 1 includes memory allocation for ‘hex_vals’ and ‘hex_end’ which are pointers to a block of RAM (random access memory) reserved for indexing the binary sequences required. The addresses ‘inner’ and ‘outer’ are selected to be used for the delay subroutine.

```
main.asm
1 ; FILE: Task 1.asm
2 ; AUTHOR: Caitlin Lonsdale
3 ; DATE: 02/01/2025
4
5
6 LIST p=16F84 ; Select microprocessor type
7 RADIX hex ; Define hexadecimal as the number base
8
9 #include p16f84.inc ; Find and include the microprocessor file
10
11 ;
12
13 UDATA_SHR ; Declare the start of a section of shared uninitialised data
14 hex_vals RES 0x10 ; Pointer to reserved memory block (16B)
15 hex_end ; Pointer to memory address directly after data being used
16
17 portB EQU 0x06 ; Pointer to portB memory address
18 trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
19 status EQU 0x03 ; Pointer to status register memory address (0-7)
20 RP0 EQU 5 ; Bit number (5) of RP0 in the status register
21 RP1 EQU 6 ; Bit number (6) of RP1 in the status register
22
23 inner EQU 0x1C ; RAM memory address of data
24 outer EQU 0x1D ; RAM memory address of data
25
26 ORG 0x00 ; Start the program at 0x00
27
```

Figure 28 - Task 1 assembly code: initialisation

The main subroutine starts by initialising the status register bits and setting all the pins on port B as outputs. Next the reserved RAM block is initialised with the binary sequences of the display characters, by calling the display subroutine. The main loop iterates over so the display counts up in hexadecimal and restarts. These counts are at 0.2 second increments by use of the delay subroutine.

```

28; -----
29
30; Subroutine: main
31; Description: This is the main function for the program. It starts by initialising the status register bits
32; and setting portB pins as outputs. Then the RAM block is initialised with the display values,
33; after calling the subroutine display. The main loop iterates over so the display counts up in
34; hexadecimal in 0.2 second increments.
35; Register Usage:
36;   w -> 'Working register' stores values temporarily
37;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
38;   trisB -> Sets the data direction (input/output) of portB
39;   portB -> Accesses data pins of portB
40;   FSR -> Addresses indirect data memory (data pointer)
41;   INDF -> Accesses data at FSR address (non-physical register)
42
43 main
44     BCF     status,RP0      ; Clear RP0 bit in the status register
45     BCF     status,RP1      ; Clear RP1 bit in the status register
46     BSF     status,RP0      ; Set RP0 bit in the status register
47     MOVLW   0x00            ; Move value 0 to the W register
48     MOVWF   trisB           ; Move value 0 to trisB to set portB as all output pins
49     BCF     status,RP0      ; Bit Clear File - Clear RP0 bit in the status register
50     CLRF    portB           ; Clear current pin values on portB
51
52     CALL    display         ; Call subroutine display
53
54 reset    MOVLW   hex_vals    ; Move hex_vals to the W register
55          MOVWF   FSR         ; Move hex_vals to FSR register
56
57 loop     MOVF    INDF,0      ; Move FSR value (in INDF) to the W register
58          MOVWF   portB       ; Move FSR value to portB to output to the display
59          CALL    delay        ; Call subroutine delay
60          INCF    FSR          ; Increment the FSR pointer
61          MOVF    FSR,0       ; Move FSR value to the W register
62          XORLW   hex_end      ; XOR the W register (from FSR) and hex_end (from RAM) values to test if equal
63          BTFSZ   status,Z     ; If address value are equal skip next instruction (Z bit=0 if equal)
64          GOTO    loop        ; Return to loop to display the next number
65          GOTO    reset       ; Return to reset to start hexadecimal count
66          RETURN              ; Return from subroutine main

```

Figure 29 – Task 1 assembly code: main subroutine

The delay subroutine, as shown below, is called in the main subroutine.

```

67; -----
68
69; Subroutine: delay
70; Description: This is the delay subroutine, called in the main function for the program. It forms a counting loop
71; to occupy the microprocessor for 0.2 seconds as required.
72; Register Usage:
73;   w -> 'Working register' stores values temporarily
74;   outer -> Contains the outer loop counter
75;   inner -> Contains the inner loop counter
76
77
78 delay
79     MOVLW   0xFF            ; Move value 255 to the W register
80     MOVWF   outer           ; Outer loop counter - Set outer value as 255
81
82     MOVLW   0xFF            ; Move value 255 to the W register
83     MOVWF   inner           ; Inner loop counter - Set inner value as 255
84
85     innerL  DECFSZ  inner,F    ; Decrement inner by 1, skip next instruction if 0
86             GOTO    innerL     ; Return to innerL to repeat count
87     outerL  DECFSZ  outer,F    ; Decrement outer by 1, skip next instruction if 0
88             GOTO    outerL     ; Return to outerL to repeat count
89     RETURN              ; Return from subroutine delay
90

```

Figure 30 – Task 1 assembly code: delay subroutine (1)

```

91; -----
92
93; Subroutine: display
94; Description: This is the display subroutine, called in the main function for the program. It stores all the
95; binary sequences for the hexadecimal display in RAM so they can be sequentially outputted on portB.
96; Register Usage:
97;   w -> 'Working register' stores values temporarily
98;   hex_vals -> Contains the addresses of the hexadecimal display sequences
99

```

Figure 31 – Task 1 assembly code: delay subroutine (2)

The delay subroutine, as shown below, is called in the main subroutine.

```

100 display
101     MOVLW    b'01000000'    ; Move binary sequence for 0 to the W register
102     MOVWF    hex_vals      ; Move binary sequence for 0 to hex_vals (0x0C)
103     MOVLW    b'01111001'    ; Move binary sequence for 1 to the W register
104     MOVWF    hex_vals+0x01   ; Move binary sequence for 1 to hex_vals (0x0D)
105     MOVLW    b'00100100'    ; Move binary sequence for 2 to the W register
106     MOVWF    hex_vals+0x02   ; Move binary sequence for 2 to hex_vals (0x0E)
107     MOVLW    b'00110000'    ; Move binary sequence for 3 to the W register
108     MOVWF    hex_vals+0x03   ; Move binary sequence for 3 to hex_vals (0x0F)
109     MOVLW    b'00011001'    ; Move binary sequence for 4 to the W register
110     MOVWF    hex_vals+0x04   ; Move binary sequence for 4 to hex_vals (0x10)
111     MOVLW    b'00010010'    ; Move binary sequence for 5 to the W register
112     MOVWF    hex_vals+0x05   ; Move binary sequence for 5 to hex_vals (0x11)
113     MOVLW    b'00000010'    ; Move binary sequence for 6 to the W register
114     MOVWF    hex_vals+0x06   ; Move binary sequence for 6 to hex_vals (0x12)
115     MOVLW    b'01111000'    ; Move binary sequence for 7 to the W register
116     MOVWF    hex_vals+0x07   ; Move binary sequence for 7 to hex_vals (0x13)
117     MOVLW    b'00000000'    ; Move binary sequence for 8 to the W register
118     MOVWF    hex_vals+0x08   ; Move binary sequence for 8 to hex_vals (0x14)
119     MOVLW    b'00010000'    ; Move binary sequence for 9 to the W register
120     MOVWF    hex_vals+0x09   ; Move binary sequence for 9 to hex_vals (0x15)
121     MOVLW    b'00001000'    ; Move binary sequence for A to the W register
122     MOVWF    hex_vals+0x0A   ; Move binary sequence for A to hex_vals (0x16)
123     MOVLW    b'00000011'    ; Move binary sequence for b to the W register
124     MOVWF    hex_vals+0x0B   ; Move binary sequence for b to hex_vals (0x17)
125     MOVLW    b'01000110'    ; Move binary sequence for C to the W register
126     MOVWF    hex_vals+0x0C   ; Move binary sequence for C to hex_vals (0x18)
127     MOVLW    b'00100001'    ; Move binary sequence for d to the W register
128     MOVWF    hex_vals+0x0D   ; Move binary sequence for d to hex_vals (0x19)
129     MOVLW    b'00000110'    ; Move binary sequence for E to the W register
130     MOVWF    hex_vals+0x0E   ; Move binary sequence for E to hex_vals (0x1A)
131     MOVLW    b'00001110'    ; Move binary sequence for F to the W register
132     MOVWF    hex_vals+0x0F   ; Move binary sequence for F to hex_vals (0x1B)
133     RETURN                                ; Return from subroutine display
134
135     END                                ; End program

```

Figure 32 – Task 1 assembly code: display subroutine

Task 2

Secondly, the PIC is programmed “so that when you press one of the two buttons, it displays L on the 7-segment LED display, then when you press the other one, it displays r”. These characters can be display by the same process as task 1, where a corresponding binary sequence (shown in the table below) is outputted from the pins on port B.

Character	Display Pins	Binary Sequence	Hexadecimal Value
L	D E F	0100 0111	0x47
r	E G	0010 1111	0x2F

The flowchart below shows the logic of the button-pressing and response. When button SW2 (on the simulation) is pressed, and gives an input to the microcontroller, the character ‘L’ is displayed. The same process takes place with SW3 and the character ‘r’. It must be noted that since the inputs from the buttons are received at pins A0 and A1, these must be set as inputs on the microcontroller (see initialisation below).

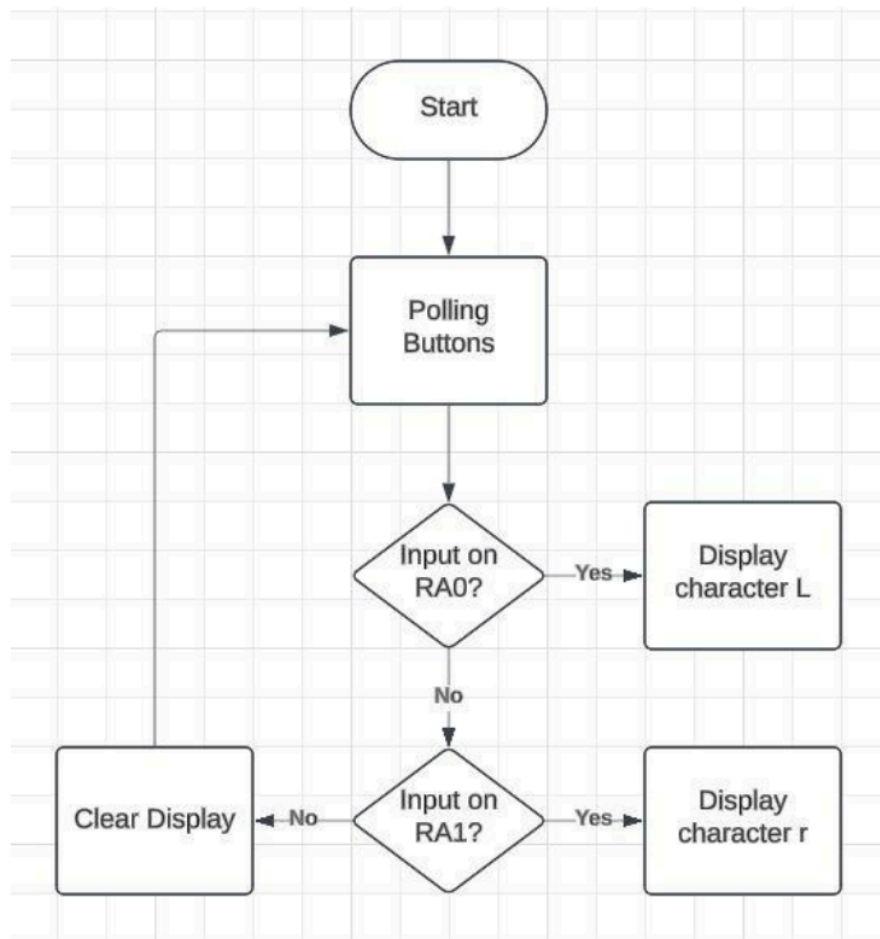


Figure 33 – Task 2 flowchart

Below we have the initialisation process for task 2 - microprocessor type and supporting file must first be referenced, then memory allocated. In this case the assigned hexadecimal values for displaying 'L' and 'r' are listed (as given in the table above).

```

main.asm
1; FILE:      Task 2.asm
2; AUTHOR:    Caitlin Lonsdale
3; DATE:      02/01/2025
4
5
6      LIST    p=16F84      ; Select microprocessor type
7      RADIX   hex          ; Define hexadecimal as the number base
8
9#include p16f84.inc        ; Find and include the microprocessor file
10
11-----
12
13portA    EQU 0x05          ; Pointer to portA memory address
14trisA    EQU 0x05          ; Pointer to portA data direction register memory address (trisA)
15portB    EQU 0x06          ; Pointer to portB memory address
16trisB    EQU 0x06          ; Pointer to portB data direction register memory address (trisB)
17status   EQU 0x03          ; Pointer to status register memory address (0-7)
18RP0      EQU 5             ; Bit number (5) of RP0 in the status register
19RP1      EQU 6             ; Bit number (6) of RP1 in the status register
20
21L         EQU 0x47          ; Hexadecimal value for display L
22r         EQU 0x2F          ; Hexadecimal value for display r
23
24      ORG 0x00             ; Start the program at 0x00
25

```

Figure 34 – Task 2 assembly code: initialisation

The main subroutine starts by initialising the status register bits – setting all the pins on port B as outputs, and first two pins of port A as inputs. The main loop polls over pins A0 and A1, to see if there is input from either being pressed. According to the brief; if SW2 is pressed the character 'L' is displayed, and if SW3 is pressed the character 'r' is displayed.

```

26; -----
27
28; Subroutine: main
29; Description: This is the main function for the program. It starts by initialising the status register bits,
30; setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
31; from the buttons to see if either have been pressed. If the left side button (SW2) is pressed,
32; then the letter L is displayed. If the right side button (SW3) is pressed, the r is displayed.
33; Register Usage:
34;   w -> 'Working register' stores values temporarily
35;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
36;   trisA -> Sets the data direction (input/output) of portA
37;   portA -> Accesses data pins of portA
38;   trisB -> Sets the data direction (input/output) of portB
39;   portB -> Accesses data pins of portB
40
41main
42      BCF    status,RP0      ; Subroutine main
43      BCF    status,RP1      ; Clear RP0 bit in the status register
44      BSF    status,RP0      ; Clear RP1 bit in the status register
45      MOVLW  0x03            ; Set RP0 bit in the status register
46      MOVWF  trisA           ; Move value b00000011 to the W register
47      MOVLW  0x00            ; Move value b00000011 to trisA to set portA pins 0 and 1 as inputs
48      MOVWF  trisB           ; Move value b00000000 to the W register
49      MOVLW  0x00            ; Move value b00000000 to trisB to set portB as all output pins
50
51      BCF    status,RP0      ; Bit Clear File - Clear RP0 bit in the status register
52      CLRF   portA           ; Clear current pin values on portA
53      CLRF   portB           ; Clear current pin values on portB
54
55loop    BTFSS  portA,0        ; Test pin A0 - if high then skip next instruction, if low then execute
56      MOVLW  L               ; Move hexadecimal value for L to the W register
57      BTFSS  portA,1        ; Test pin A1 - if high then skip next instruction, if low then execute
58      MOVLW  r               ; Move hexadecimal value for r to the W register
59      MOVWF  portB           ; Move W register value to portB to output to display
60      GOTO   loop           ; Return to loop to display the next Letter
61
62      END                   ; End program

```

Figure 35 – Task 2 assembly code: main subroutine

Task 3

Thirdly, the PIC is programmed “so one button causes all of the LED segments to light, and the other switch turns them off”. The flowchart below shows the logic of the button-pressing and response, where if button SW2 is pressed (and there is an input on pin RA0), all 7 LED segments are switched on by outputting a binary sequence to port B (as in tasks 1 and 2). If button SW3 is pressed, causing an input signal to the microcontroller, the LED segments are switched off by the same process. Below the flowchart is a table containing the corresponding binary sequences for the display.

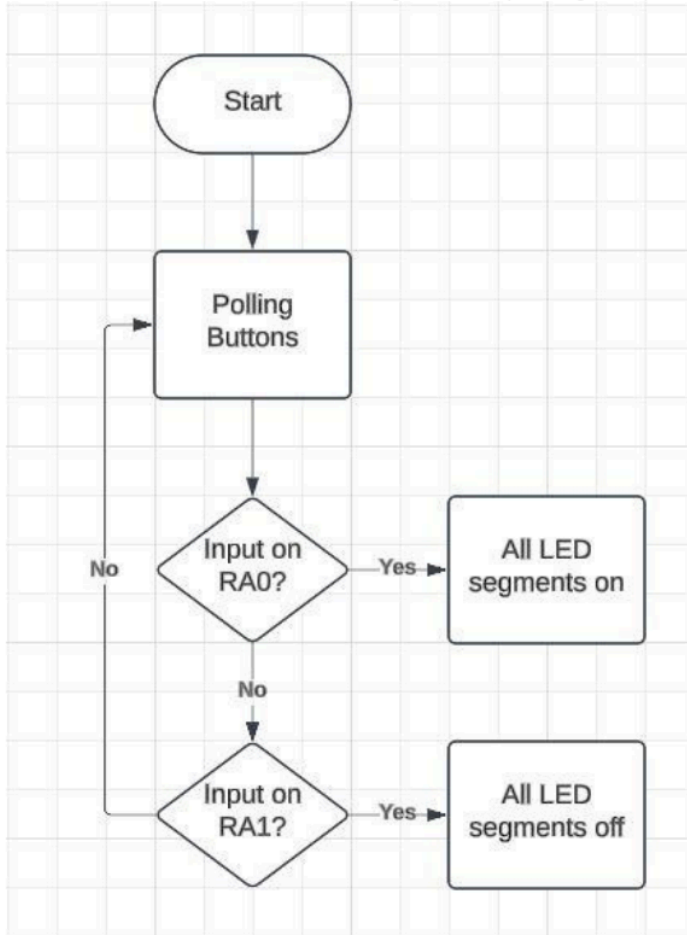


Figure 36 – Task 3 flowchart

Display	Binary Sequence	Hexadecimal Value
On	0000 0000	0x00
Off	0111 1111	0x7F

Below we have the initialisation process for task 3 - microprocessor type and supporting file must first be referenced, then memory allocated. In this case the assigned hexadecimal values for setting the display all 'on' or 'off' are listed (as given in the table above).

```

1; FILE:      Task 3.asm
2; AUTHOR:    Caitlin Lonsdale
3; DATE:      02/01/2025
4
5
6          LIST    p=16F84      ; Select microprocessor type
7          RADIX    hex          ; Define hexadecimal as the number base
8
9#include p16f84.inc              ; Find and include the microprocessor file
10
11-----
12
13portA     EQU 0x05              ; Pointer to portA memory address
14trisA     EQU 0x05              ; Pointer to portA data direction register memory address (trisA)
15portB     EQU 0x06              ; Pointer to portB memory address
16trisB     EQU 0x06              ; Pointer to portB data direction register memory address (trisB)
17status    EQU 0x03              ; Pointer to status register memory address (0-7)
18RP0       EQU 5                 ; Bit number (5) of RP0 in the status register
19RP1       EQU 6                 ; Bit number (6) of RP1 in the status register
20
21on         EQU 0x00              ; Hexadecimal value for display all on
22off        EQU 0x7F              ; Hexadecimal value for display all off
23
24          ORG 0x00              ; Start the program at 0x00
25

```

Figure 37 – Task 3 assembly code: initialisation

The main subroutine starts by initialising the status register bits – setting all the pins on port B as outputs, and first two pins of port A as inputs. The main loop polls over pins A0 and A1, to see if there is input from either being pressed. According to the brief; if SW2 is pressed the display segments are all on, and if SW3 is pressed they are all off.

```

26; -----
27
28; Subroutine: main
29; Description: This is the main function for the program. It starts by initialising the status register bits,
30;              setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
31;              from the buttons to see if either have been pressed. If the left side button (SW2) is pressed,
32;              then all the display segments are on. If the right side button (SW3) is pressed, all are off.
33; Register Usage:
34;   w -> 'Working register' stores values temporarily
35;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
36;   trisa -> Sets the data direction (input/output) of portA
37;   porta -> Accesses data pins of portA
38;   trisb -> Sets the data direction (input/output) of portB
39;   portb -> Accesses data pins of portB
40
41main
42    BCF    status,RP0            ; Clear RP0 bit in the status register
43    BCF    status,RP1            ; Clear RP1 bit in the status register
44    BSF    status,RP0            ; Set RP0 bit in the status register
45    MOVLW  0x03                  ; Move value b00000011 to the W register
46    MOVWF  trisA                 ; Move value b00000011 to trisA to set portA pins 0 and 1 as inputs
47    MOVLW  0x00                  ; Move value b00000000 to the W register
48    MOVWF  trisB                 ; Move value b00000000 to trisB to set portB as all output pins
49
50    BCF    status,RP0            ; Bit Clear File - Clear RP0 bit in the status register
51    CLRF   porta                 ; Clear current pin values on portA
52    CLRF   portB                 ; Clear current pin values on portB
53
54loop   BTFSS   porta,0            ; Test pin A0 - if high then skip next instruction, if low then execute
55        MOVLW  on                 ; Move hexadecimal value for display all on to the W register
56        BTFSS   porta,1            ; Test pin A1 - if high then skip next instruction, if low then execute
57        MOVLW  off                 ; Move hexadecimal value for display all off to the W register
58        MOVWF  portB              ; Move W register value to portB to output to display
59        GOTO    loop              ; Return to loop to display again
60
61    END                          ; End program

```

Figure 38 – Task 3 assembly code: main subroutine

Task 4

Next, the PIC is programmed “so you start by displaying '5' but you can count up (to a maximum of 9) if you press one button, and count down (to a minimum of 0) if you press the other”. The flowchart below shows the logic of the button-pressing and response, where if button SW2 is pressed (and there is an input on pin RA0), the LED display number is decremented down to a minimum of zero. As in task 1, characters are displayed on the 7-segment display by outputting a binary sequence to port B (as in tasks 1 and 2). If button SW3 is pressed (and there is an input on pin RA1), the LED display is incremented up to a maximum of nine.

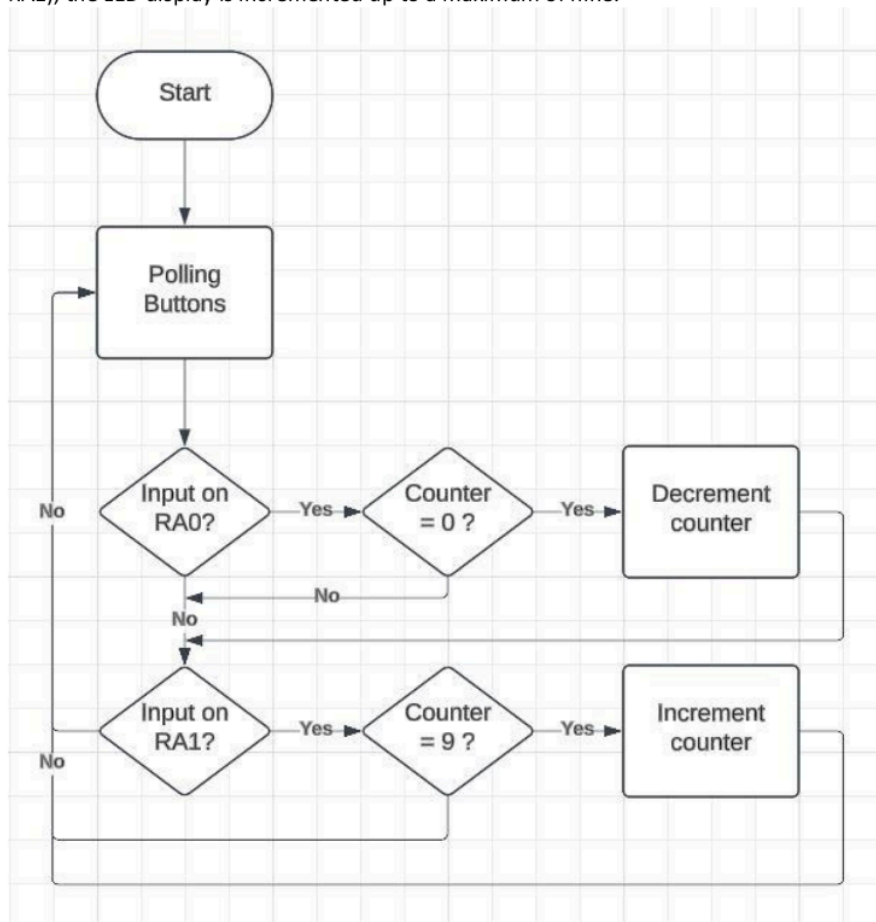


Figure 39 – Task 4 flowchart

Below we have the initialisation process - microprocessor type and supporting file must first be referenced, then memory allocated. As in task 1, memory is allocated for 'hex_vals' and 'hex_end' which are pointers to a block of RAM (random access memory) reserved for indexing the binary sequences required. The addresses 'inner' and 'outer' are selected to be used for the delay subroutine.

```

main.asm
1; FILE:      Task 4.asm
2; AUTHOR:    Caitlin Lonsdale
3; DATE:      02/01/2025
4
5
6          LIST    p=16F84      ; Select microprocessor type
7          RADIX   hex          ; Define hexadecimal as the number base
8
9#include p16f84.inc           ; Find and include the microprocessor file
10
11;
12
13          UDATA_SHR          ; Declare the start of a section of shared uninitialised data
14num_vals  RES 0x0A          ; Pointer to reserved memory block (10B)
15num_end    ; Pointer to memory address directly after data being used
16
17portA     EQU 0x05          ; Pointer to portA memory address
18trisA     EQU 0x05          ; Pointer to portA data direction register memory address (trisA)
19portB     EQU 0x06          ; Pointer to portB memory address
20trisB     EQU 0x06          ; Pointer to portB data direction register memory address (trisB)
21status    EQU 0x03          ; Pointer to status register memory address (0-7)
22RP0       EQU 5             ; Bit number (5) of RP0 in the status register
23RP1       EQU 6             ; Bit number (6) of RP1 in the status register
24
25inner     EQU 0x1C          ; RAM memory address of data
26outer    EQU 0x1D          ; RAM memory address of data
27
28          ORG 0x00          ; Start the program at 0x00
29

```

Figure 40 - Task 4 assembly code: initialisation

The main subroutine has the same initialisation as task 1 (see above). After starting the display with the number 5; the main loop polls over pins A0 and A1, to see if there is input from either being pressed. According to the brief; and if SW2 is pressed the display decrements down to 0 in integer intervals, as in the countD subroutine. Alternatively, if SW3 is pressed the display increments up to 9 in integer intervals, according to the countU subroutine. The delay subroutine is used within the two 'count' subroutines to ensure changes between the display are readable to the human eye.

```

30;
31;
32; Subroutine: main
33; Description: This is the main function for the program. It starts by initialising the status register bits,
34;              setting portA pins as inputs and setting portB pins as outputs. The number 5 is initially
35;              displayed. Then it polls over the input from the buttons to see if either have been pressed.
36;              If the left side button (SW2) is pressed, the display will decrement the number (min is 0).
37;              If the right side button (SW3) is pressed, the display will increment the number (max is 9).
38;
39; Register Usage:
40;      w -> 'Working register' stores values temporarily
41;      status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
42;      trisa -> Sets the data direction (input/output) of portA
43;      porta -> Accesses data pins of portA
44;      trisb -> Sets the data direction (input/output) of portB
45;      portb -> Accesses data pins of portB
46;      FSR -> Addresses indirect data memory (data pointer)
47;      INDF -> Accesses data at FSR address (non-physical register)
48;
49main
50      BCF    status,RP0      ; Clear RP0 bit in the status register
51      BCF    status,RP1      ; Clear RP1 bit in the status register
52      BSF    status,RP0      ; Set RP0 bit in the status register
53      MOVLW  0x03            ; Move value 00000011 to the W register
54      MOVWF  trisA           ; Move value 00000011 to trisA to set portA pins 0 and 1 as inputs
55      MOVLW  0x00            ; Move value 00000000 to the W register
56      MOVWF  trisB           ; Move value 00000000 to trisB to set portB as all output pins
57      BCF    status,RP0      ; Clear RP0 bit in the status register
58      CLRF   porta           ; Clear current pin values on portA
59      CLRF   portb           ; Clear current pin values on portB
60
61      CALL   display         ; Call subroutine display
62      MOVLW  num_vals+0x05    ; Move num_vals index 5 to the W register
63      MOVWF  FSR             ; Move num_vals index 5 to FSR register
64      MOVF   INDF,0          ; Move INDF value to the W register
65      MOVWF  portB           ; Move FSR value to portB to output to the display

```

Figure 41 – Task 4 assembly code: main subroutine (1)

```

67loop    BTFSS   portA,0      ; Test pin A0 - if high then skip next instruction, if low then execute
68          CALL   countD      ; Call subroutine countD
69          BTFSS   portA,1      ; Test pin A1 - if high then skip next instruction, if low then execute
70          CALL   countU      ; Call subroutine countU
71          GOTO    loop        ; Go to loop line

```

Figure 42 – Task 4 assembly code: main subroutine (2)

The countD subroutine, as shown below, is called in the main subroutine.

```

73; -----
74;
75; Subroutine: countD
76; Description: This is the countD subroutine, called in the main function for the program. It counts down the
77; number on the display, decreasing by 1 for each related button press, down to 0.
78; Register Usage:
79; w -> 'Working register' stores values temporarily
80; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
81; portB -> Accesses data pins of portB
82; FSR -> Addresses indirect data memory (data pointer)
83; INDF -> Accesses data at FSR address (non-physical register)
84;
85 countD      ; Subroutine countD
86      MOVF    FSR,0      ; Move FSR value to the W register
87      XORLW   0x0C      ; XOR the W register (from FSR) and 0x0C (display 0) values to test if equal
88      BTFS    status,Z  ; If address value are equal skip next instruction (Z bit=0 if equal)
89      DECF    FSR,1      ; Decrement the FSR pointer
90      MOVF    INDF,0     ; Move INDF value to the W register
91      MOVWF   portB      ; Move FSR value to portB to output to the display
92      CALL    delay      ; Call subroutine delay
93      RETURN           ; Return from subroutine countD

```

Figure 43 – Task 4 assembly code: countD subroutine

The countU subroutine, as shown below, is called in the main subroutine.

```

95; -----
96;
97; Subroutine: countU
98; Description: This is the countU subroutine, called in the main function for the program. It counts up the
99; number on the display, increasing by 1 for each related button press, up to 9.
100; Register Usage:
101; w -> 'Working register' stores values temporarily
102; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
103; portB -> Accesses data pins of portB
104; FSR -> Addresses indirect data memory (data pointer)
105; INDF -> Accesses data at FSR address (non-physical register)
106;
107 countU      ; Subroutine countU
108      MOVF    FSR,0      ; Move FSR value to the W register
109      XORLW   0x15      ; XOR the W register (from FSR) and 0x15 (display 9) values to test if equal
110      BTFS    status,Z  ; If address value are equal skip next instruction (Z bit=0 if equal)
111      INCF    FSR,1      ; Increment the FSR pointer
112      MOVF    INDF,0     ; Move INDF value to the W register
113      MOVWF   portB      ; Move FSR value to portB to output to the display
114      CALL    delay      ; Call subroutine delay
115      RETURN           ; Return from subroutine countU
116;

```

Figure 44 – Task 4 assembly code: countU subroutine

The delay subroutine, as shown below, is called in the main subroutine.

```

117; -----
118;
119; Subroutine: delay
120; Description: This is the delay subroutine, called in the main function for the program. It forms a counting loop
121; to occupy the microprocessor for 0.2 seconds as required.
122; Register Usage:
123; w -> 'Working register' stores values temporarily
124; outer -> Contains the outer loop counter
125; inner -> Contains the inner loop counter
126;
127 delay      ; Subroutine delay
128      MOVLW   0xFF      ; Move value 255 to the W register
129      MOVWF   outer      ; Outer loop counter - Set outer value as 255
130;
131 outerL      MOVLW   0xFF      ; Move value 255 to the W register
132      MOVWF   inner      ; Inner loop counter - Set inner value as 255
133;
134 innerL      DECFSZ   inner,F  ; Decrement inner by 1, skip next instruction if 0
135      GOTO     innerL        ; Return to innerL to repeat count
136      DECFSZ   outer,F  ; Decrement outer by 1, skip next instruction if 0
137      GOTO     outerL       ; Return to outerL to repeat count
138      RETURN           ; Return from subroutine delay
139;

```

Figure 45 – Task 4 assembly code: delay subroutine

The display subroutine, as shown below, is called in the main subroutine.

```

140; -----
141
142; Subroutine: display
143; Description: This is the display subroutine, called in the main function for the program. It stores all the
144;               binary sequences for the 0-9 display in RAM so they can be outputted on portB.
145; Register Usage:
146;   w -> 'Working register' stores values temporarily
147;   hex_vals -> Contains the addresses of the hexadecimal display sequences
148
149 display
150     MOVLW    b'01000000'    ; Move binary sequence for 0 to the W register
151     MOVWF    num_vals       ; Move binary sequence for 0 to num_vals (0x0C)
152     MOVLW    b'01111001'    ; Move binary sequence for 1 to the W register
153     MOVWF    num_vals+0x01   ; Move binary sequence for 1 to num_vals (0x0D)
154     MOVLW    b'00100100'    ; Move binary sequence for 2 to the W register
155     MOVWF    num_vals+0x02   ; Move binary sequence for 2 to num_vals (0x0E)
156     MOVLW    b'00110000'    ; Move binary sequence for 3 to the W register
157     MOVWF    num_vals+0x03   ; Move binary sequence for 3 to num_vals (0x0F)
158     MOVLW    b'00011001'    ; Move binary sequence for 4 to the W register
159     MOVWF    num_vals+0x04   ; Move binary sequence for 4 to num_vals (0x10)
160     MOVLW    b'00010010'    ; Move binary sequence for 5 to the W register
161     MOVWF    num_vals+0x05   ; Move binary sequence for 5 to num_vals (0x11)
162     MOVLW    b'00000010'    ; Move binary sequence for 6 to the W register
163     MOVWF    num_vals+0x06   ; Move binary sequence for 6 to num_vals (0x12)
164     MOVLW    b'01111000'    ; Move binary sequence for 7 to the W register
165     MOVWF    num_vals+0x07   ; Move binary sequence for 7 to num_vals (0x13)
166     MOVLW    b'00000000'    ; Move binary sequence for 8 to the W register
167     MOVWF    num_vals+0x08   ; Move binary sequence for 8 to num_vals (0x14)
168     MOVLW    b'00010000'    ; Move binary sequence for 9 to the W register
169     MOVWF    num_vals+0x09   ; Move binary sequence for 9 to num_vals (0x15)
170     RETURN                  ; Return from subroutine display
171
172     END                    ; End program

```

Figure 46 – Task 4 assembly code: display subroutine

Task 5

Penultimately, the PIC is programmed “so that the piezo sounder produces a tone while one of the buttons is pressed”. The flowchart below shows the logic of the button-pressing and response, where if button SW2 is pressed (and there is an input on pin RA0), the piezo buzzer sounds. This is because the microcontroller will output a signal (high or logic 1) on pin 7 of port B (as discussed in section 1). When the button is not pressed, the microcontroller output on RB7 is low (or logic 0) and the piezo buzzer is off. Below the flowchart is a table containing these corresponding binary sequences.

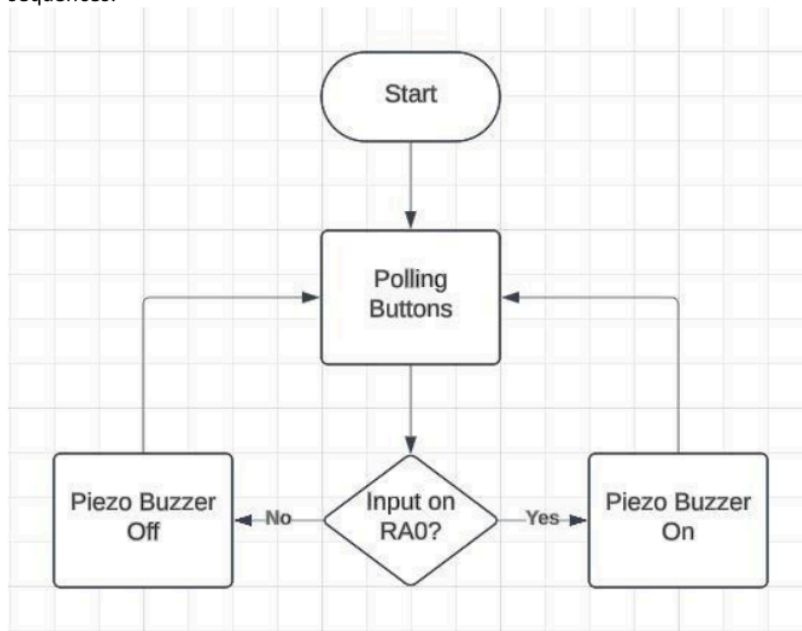


Figure 47 – Task 5 flowchart

Piezo Buzzer	Binary Sequence	Hexadecimal Value
On	1111 1111	0xFF
Off	0111 1111	0x7F

Below is also a capture of the required update to the Proteus simulation circuit – a digital oscilloscope is connected to the output of the sounder to observe the program. It is to be noted that this update is also implemented in task 6.

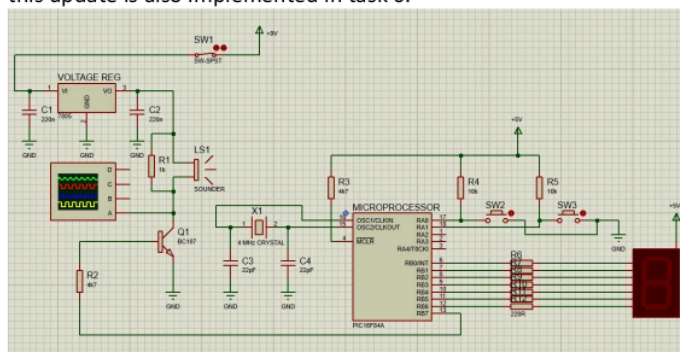


Figure 48 – Updated Proteus simulation circuit

Below we have the initialisation process - microprocessor type and supporting file must first be referenced, then memory allocated. In this case the assigned hexadecimal values for setting the piezo buzzer 'on' or 'off' are listed (as given in the table above).

```

1; FILE:      Task 5.asm
2; AUTHOR:    Caitlin Lonsdale
3; DATE:      03/01/2025
4
5
6          LIST    p=16F84      ; Select microprocessor type
7          RADIX   hex          ; Define hexadecimal as the number base
8
9#include p16f84.inc           ; Find and include the microprocessor file
10
11;
12-----
13portA     EQU 0x05            ; Pointer to portA memory address
14trisA     EQU 0x05            ; Pointer to portA data direction register memory address (trisA)
15portB     EQU 0x06            ; Pointer to portB memory address
16trisB     EQU 0x06            ; Pointer to portB data direction register memory address (trisB)
17status    EQU 0x03            ; Pointer to status register memory address (0-7)
18RP0       EQU 5               ; Bit number (5) of RP0 in the status register
19RP1       EQU 6               ; Bit number (6) of RP1 in the status register
20
21on         EQU 0xFF            ; Hexadecimal value for piezo buzzer on
22off        EQU 0x7F            ; Hexadecimal value for piezo buzzer off
23
24          ORG 0x00             ; Start the program at 0x00
25

```

Figure 49 – Task 5 assembly code: initialisation

The main subroutine has the same initialisation as previous tasks, setting port A as input and port B as outputs. Following this, it polls over the input from the button, and will set the piezo buzzer on if a signal is received. The alternate state is that the piezo buzzer is off (no output from PIC).

```

26; -----
27
28; Subroutine: main
29; Description: This is the main function for the program. It starts by initialising the status register bits,
30; setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
31; from the button (SW2) to see if it's been pressed. While pressed, the piezo buzzer will sound.
32; Register Usage:
33; w -> 'Working register' stores values temporarily
34; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
35; trisA -> Sets the data direction (input/output) of portA
36; portA -> Accesses data pins of portA
37; trisB -> Sets the data direction (input/output) of portB
38; portB -> Accesses data pins of portB
39
40main      ; Subroutine main
41          BCF    status,RP0    ; Clear RP0 bit in the status register
42          BCF    status,RP1    ; Clear RP1 bit in the status register
43          BSF    status,RP0    ; Set RP0 bit in the status register
44          MOVLW  0x01          ; Move value b00000001 to the W register
45          MOVWF  trisA         ; Move value b00000001 to trisA to set portA pins 0 as an input
46          MOVLW  0x00          ; Move value b00000000 to the W register
47          MOVWF  trisB         ; Move value b00000000 to trisB to set portB as all output pins
48
49          BCF    status,RP0    ; Clear RP0 bit in the status register
50          CLRF   portA         ; Clear current pin values on portA
51          CLRF   portB         ; Clear current pin values on portB
52
53loop      BTFSS   portA,0       ; Test pin A0 - if high then skip next instruction, if low then execute
54          MOVLW  on            ; Move hexadecimal value for piezo buzzer on to the W register
55          BTFSC   portA,0       ; Test pin A0 - if low then skip next instruction, if high then execute
56          MOVLW  off          ; Move hexadecimal value for piezo buzzer off to the W register
57          MOVWF  portB         ; Move W register value to portB to output to buzzer
58          GOTO   loop          ; Return to loop
59
60          END                  ; End program

```

Figure 50 - Task 5 assembly code: main subroutine

The capture from oscilloscope below demonstrates the activity of the circuit. At the point of the dashed vertical line we see a rising edge on the signal from the sounder. Since this capture was taken at the point of the button being pressed, it demonstrates how the microcontroller then sets the sounder on in response, as the program specifies.

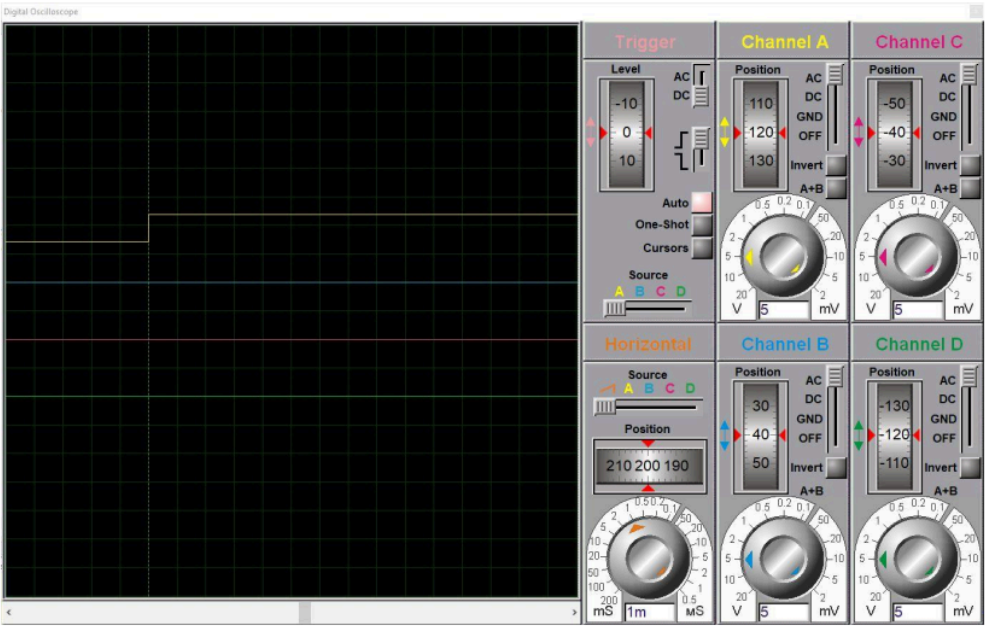


Figure 51 – Task 5 oscilloscope reading

Task 6

Lastly, the PIC is programmed “so a fixed frequency, constantly changing PWM waveform pulse train to drives the speaker on the proteus simulation”. PWM (pulse width modulation) is a waveform technique in which a square ‘pulsed’ wave is transmitted at a given frequency [16]. The waveform has a ‘duty cycle’ which determines what percentage of the time period the signal is ‘high’. In this case, ‘high’ would denote the +5v output from a microcontroller pin to the sounder.

Below is a flowchart which outlines the logic of a PWM signal output from the microcontroller, in which the duty cycle starts at 100%, decreasing down to 10% in 10% intervals – then restarts. The table includes the binary and hexadecimal reference values for these duty cycles.

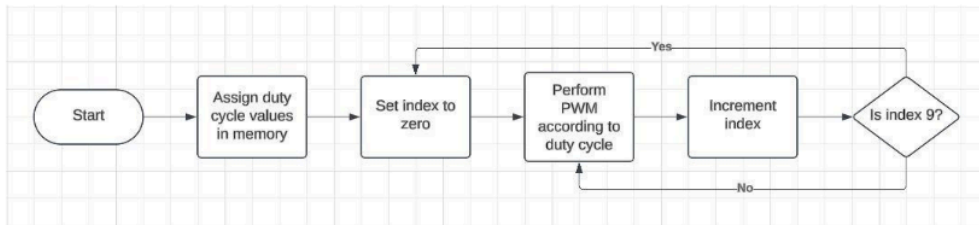


Figure 52 – Task 6 flowchart

Duty Cycle (%)	Binary Value	Hexadecimal Value
100	255	0xFF
90	229	0xE5
80	204	0xCC
70	178	0xB2
60	153	0x99
50	127	0x7F
40	101	0x65
30	76	0x4C
20	32	0x32
10	19	0x19

```

main.asm
1; FILE: Task 6.asm
2; AUTHOR: Caitlin Lonsdale
3; DATE: 03/01/2025
4
5
6 LIST p=16F84 ; Select microprocessor type
7 RADIX hex ; Define hexadecimal as the number base
8
9 #include p16f84.inc ; Find and include the microprocessor file
10
11 ;
12
13 UDATA_SHR ; Declare the start of a section of shared uninitialised data
14 cyc_vals RES 0x0A ; Pointer to reserved memory block (10B)
15 cyc_end ; Pointer to memory address directly after data being used
16
17 portB EQU 0x06 ; Pointer to portB memory address
18 trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
19 status EQU 0x03 ; Pointer to status register memory address (0-7)
20 RP0 EQU 5 ; Bit number (5) of RP0 in the status register
21 RP1 EQU 6 ; Bit number (6) of RP1 in the status register
22
23 on EQU 0xFF ; Hexadecimal value for piezo buzzer on
24 off EQU 0x7F ; Hexadecimal value for piezo buzzer off
25 duty EQU 0x1C ; Length of duty cycle
26 temp EQU 0x1D ; Length of temp duty cycle
27
28 ORG 0x00 ; Start the program at 0x00
29
  
```

Above we have the initialisation process - microprocessor type and supporting file must first be referenced, then memory allocated. As in task 5, we reference 'on' and 'off' as the sequences across port B to set the piezo buzzer on/off at the appropriate times. We also allocate memory for 'duty' and 'temp', which are used to form a varying duty cycles (see cycles subroutine below).

Below we have the main subroutine, as seen in all the previous tasks. To start, it sets the buzzer on 'high' with the first duty cycle - counted by decrementing the reference value down to zero, once per instruction cycle. Once zero is reached, the buzzer (output on RB7) is set off for the 'low' portion of the waveform. It must be noted that the length of the low portion is calculated by subtracting the duty cycle length from 255 (since there is an 8-bit resolution) – as can be seen in the subroutine pwm. For each cycle the duty cycle is decreased, according to the subroutine cycles. When the smallest duty cycle is reached the program restarts.

Figure 53 – Task 6 assembly code: initialisation

```

30 ; -----
31 ;
32 ; Subroutine: main
33 ; Description: This is the main function for the program. It starts by initialising the status register bits,
34 ; and setting portB as all output pins. It sets the buzzer off (via the portB output) to start.
35 ; It produces a fixed frequency pulse with a decreasing duty cycle (PWM - pulse width modulation)
36 ; which loops repeatedly, and can be heard on the buzzer.
37 ; Register Usage:
38 ; W -> 'Working register' stores values temporarily
39 ; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
40 ; trisB -> Sets the data direction (input/output) of portB
41 ; portB -> Accesses data pins of portB
42 ; FSR -> Addresses indirect data memory (data pointer)
43 ; INDF -> Accesses data at FSR address (non-physical register)
44 ;
45 main                                ; Subroutine main
46     BCF    status,RP0                ; Clear RP0 bit in the status register
47     BCF    status,RP1                ; Clear RP1 bit in the status register
48     BSF    status,RP0                ; Set RP0 bit in the status register
49     MOVLW  0x00                      ; Move value 0 to the W register
50     MOVWF  trisB                     ; Move value 0 to trisB to set portB as all output pins
51     BCF    status,RP0                ; Clear RP0 bit in the status register
52     CLRF   portB                     ; Clear current pin values on portB
53 ;
54     CALL   cycles                    ; Call subroutine cycles
55     MOVLW  cyc_vals                  ; Move num_vals index 0 to the W register
56     MOVWF  FSR                      ; Move num_vals index 0 to FSR register
57     MOVF   INDF,0                   ; Move INDF value to the W register
58     MOVWF  duty                     ; Move INDF value to duty
59 ;
60 reset    MOVLW  off                   ; Move buzzer off value to the W register
61           MOVWF  portB                ; Move buzzer off value to portB
62           MOVLW  cyc_vals              ; Move cyc_vals to the W register
63           MOVWF  FSR                  ; Move cyc_vals to FSR register
64 ;

```

Figure 54 – Task 6 assembly code: main subroutine (1)

```

65 loop     MOVF   INDF,0                ; Move INDF value to the W register
66           MOVWF  duty                 ; Move INDF value to duty
67           CALL   pwm                  ; Call subroutine pwm
68           INCF   FSR                  ; Increment the FSR pointer
69           MOVF   FSR,0                ; Move FSR value to the W register
70           XORLW  cyc_end              ; XOR the W register (from FSR) and cyc_end (from RAM) values to test if equal
71           BTFSS  status,Z             ; If address value are equal skip next instruction (Z bit=0 if equal)
72           GOTO   loop                 ; Return to loop to display the next number
73           GOTO   reset                ; Return to reset to start cycle count
74           RETURN                      ; Return from subroutine main
75 ;

```

Figure 55 – Task 6 assembly code: main subroutine (2)

The pwm subroutine, as shown below, is called in the main subroutine.

```

76 ; -----
77 ;
78 ; Subroutine: pwm
79 ; Description: This is the pwm subroutine, called in the main function for the program. It sets the output on
80 ; portB (pin 7) to high/Low according to the duty cycle set by the subroutine cycles.
81 ; Register Usage:
82 ; w -> 'Working register' stores values temporarily
83 ; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
84 ; trisB -> Sets the data direction (input/output) of portB
85 ; portB -> Accesses data pins of portB
86
87 pwm      MOVF    duty,0      ; Move duty value to the W register
88          MOVWF   temp        ; Move duty value to temp
89          MOVLW   on          ; Move buzzer on value to the W register
90          MOVWF   portB       ; Move buzzer on value to portB
91
92 pwmHi    DECFSZ  temp        ; Decrement temp value by 1, if zero the skip next instruction
93          GOTO    pwmHi       ; Return to pwmHi
94          MOVLW   0xFF        ; Move value 255 to the W register
95          MOVWF   temp        ; Move value 255 to temp
96          MOVF    duty,0      ; Move duty value to the W register
97          SUBWF   temp,1      ; Subtract duty value from temp, store the result in temp
98          MOVLW   off         ; Move buzzer off value to the W register
99          MOVWF   portB       ; Move buzzer off value to portB
100
101 pwmLo    DECFSZ  temp        ; Decrement temp value by 1, if zero the skip next instruction
102          GOTO    pwmLo       ; Return to pwmLo
103          RETURN              ; Return from subroutine pwm
104

```

Figure 56 – Task 6 assembly code: pwm subroutine

The cycles subroutine, as shown below, is called in the main subroutine.

```

105 ; -----
106 ;
107 ; Subroutine: cycles
108 ; Description: This is the cycles subroutine, called in the main function for the program. It sets the
109 ; hexadecimal values for the duty cycles 100-10% (10% decrements). It stores all of these
110 ; in RAM so they can be accessed and used for the values duty and temp.
111 ; Register Usage:
112 ; w -> 'Working register' stores values temporarily
113 ; cyc_vals -> Contains the addresses of the duty cycle values
114
115 cycles   ; Subroutine cycles
116          MOVLW   0x0F        ; Move 100% duty cycle (255) to the W register
117          MOVWF   cyc_vals     ; Move 100% duty cycle (255) to cyc_vals index 0
118          MOVLW   0xE5        ; Move 90% duty cycle (229) to the W register
119          MOVWF   cyc_vals+0x01 ; Move 90% duty cycle (229) to cyc_vals index 1
120          MOVLW   0xCC        ; Move 80% duty cycle (204) to the W register
121          MOVWF   cyc_vals+0x02 ; Move 80% duty cycle (204) to cyc_vals index 2
122          MOVLW   0xB2        ; Move 70% duty cycle (178) to the W register
123          MOVWF   cyc_vals+0x03 ; Move 70% duty cycle (178) to cyc_vals index 3
124          MOVLW   0x99        ; Move 60% duty cycle (153) to the W register
125          MOVWF   cyc_vals+0x04 ; Move 60% duty cycle (153) to cyc_vals index 4
126          MOVLW   0x7F        ; Move 50% duty cycle (127) to the W register
127          MOVWF   cyc_vals+0x05 ; Move 50% duty cycle (127) to cyc_vals index 5
128          MOVLW   0x65        ; Move 40% duty cycle (101) to the W register
129          MOVWF   cyc_vals+0x06 ; Move 40% duty cycle (101) to cyc_vals index 6
130          MOVLW   0x4C        ; Move 30% duty cycle (76) to the W register
131          MOVWF   cyc_vals+0x07 ; Move 30% duty cycle (76) to cyc_vals index 7
132          MOVLW   0x32        ; Move 20% duty cycle (50) to the W register
133          MOVWF   cyc_vals+0x08 ; Move 20% duty cycle (50) to cyc_vals index 8
134          MOVLW   0x19        ; Move 10% duty cycle (25) to the W register
135          MOVWF   cyc_vals+0x09 ; Move 10% duty cycle (25) to cyc_vals index 9
136          RETURN              ; Return from subroutine cycles
137
138          END                  ; End program

```

Figure 57 – Task 6 assembly code: cycles subroutine

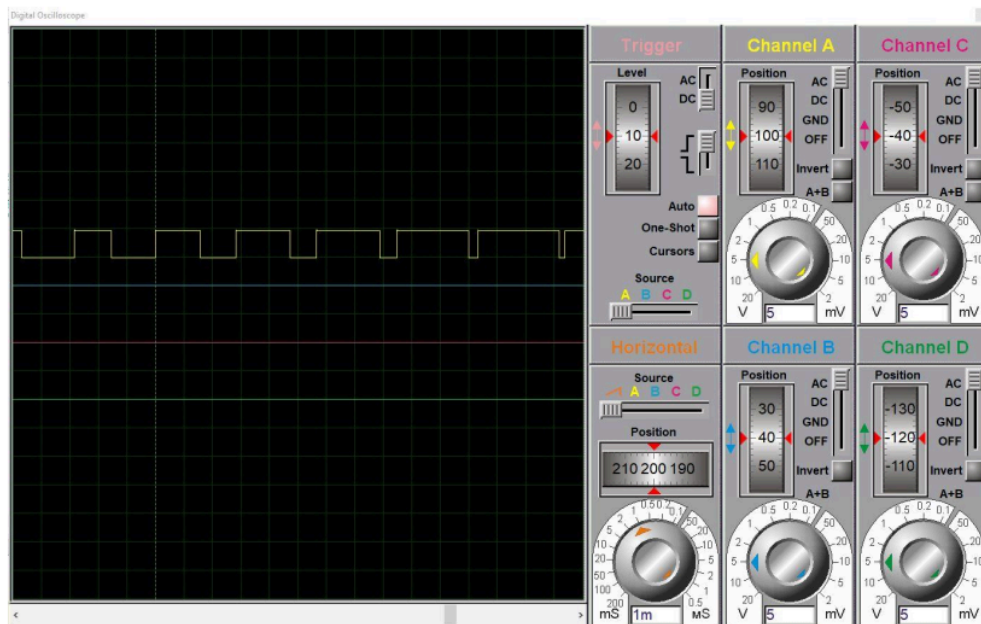


Figure 58 – Task 6 oscilloscope reading

The capture from oscilloscope above demonstrates the activity of the circuit. We see the square waveform that is characteristic to the PWM technique. We can also observe that the width of the 'high' portion changes over time, demonstrating how the duty cycle changes.

Conclusion

In conclusion, the PICF84A has been shown to be applicable for embedded circuit control. Section 1 explored how it can be incorporated into a circuit design, and is integrable with a variety of circuit components, such as driving the 7-segment display and a piezo buzzer, whilst receiving input from buttons. Section 2 demonstrated in more detail how the PIC could be used with the 7-segment display, including displaying a variety of hexadecimal characters by outputting signals from the pins on the PIC. Lastly, section 3 implemented the theory to produce assembly language applicable to the PIC in simulation software. There were a variety of tasks completed to exhibit the capabilities of the microcontroller, from a simple counting on the display, through to a varying PWM waveform. All these tasks were achieved in an efficient and conventional manner.

References

- [1] Microchip, "PIC16F84A," 2024. [Online]. Available: <https://www.microchip.com/en-us/product/pic16f84a>. [Accessed 30 12 2024].
- [2] UWTSD, *Micros Peripherals and Interfaces EES*, 2024.
- [3] W. H. Paul Horowitz, in *The Art of Electronics Third Edition*, New York, Cambridge University Press, 2015, pp. 601-602.
- [4] RS Components, "Texas Instruments LM7805CT/NOPB," 2024. [Online]. Available: Texas Instruments LM7805CT/NOPB. [Accessed 30 12 2024].
- [5] RS Components, "How do Piezo Buzzers Work?," 29 01 2024. [Online]. Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/how-do-piezo-buzzers-work?srltid=AfmBOopR7E12_EXorqH9gwauTeps1D_sK8epX13qAQmF3rSivYFa9E1D. [Accessed 30 12 2024].
- [6] RS Components, "Same Sky 91dB Through Hole External Buzzer," 2024. [Online]. Available: <https://uk.rs-online.com/web/p/piezo-buzzers/2596247?gb=s>. [Accessed 30 12 2024].
- [7] ST Microelectronics, "BC107 Datasheet," 2006.
- [8] DigiKey, "BC107," 2024. [Online]. Available: <https://www.digikey.co.uk/en/products/detail/central-semiconductor-corp/BC107-PBFREE/4806926>. [Accessed 30 12 2024].
- [9] RS Components, "Euroquartz 4MHz Crystal," 2024. [Online]. Available: <https://uk.rs-online.com/web/p/crystal-units/6938774?gb=s>. [Accessed 30 12 2024].
- [10] W. H. Paul Horowitz, in *The Art of Electronics Third Edition*, New York, Cambridge University Press, 2016, p. 443.
- [11] Suntsu Electronics, "Crystal Load Capacitance," 2024. [Online]. Available: <https://suntsu.com/suntsu-application-notes/crystal-load-capacitance/>. [Accessed 12 30 2024].
- [12] Sparkfun, "Pull-up Resistors," 2024. [Online]. Available: <https://learn.sparkfun.com/tutorials/pull-up-resistors/all>. [Accessed 12 30 2024].
- [13] Mouser Electronics, "DIP-14 Resistor Networks & Arrays Datasheets," 2024. [Online]. Available: <https://eu.mouser.com/c/ds/passive-components/resistors/resistor-networks-arrays/?package%20%2F%20case=DIP-14&termination%20style=DIP>. [Accessed 30 12 2024].
- [14] Switch Electronics, "Seven Segment LED Display Common Anode," 2024. [Online]. Available: <https://www.switchelectronics.co.uk/products/blue-0-56-seven-segment-led-display-common-anode?srltid=AfmBOopse5HHnelvNV3d5Go6QdXVZfYuZ0ecrHpFgbOQh9ZyHBL4nNAf>. [Accessed 30 12 2024].
- [15] Electronics Tutorials, "7 Segment Display," 2024. [Online]. Available: <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>. [Accessed 30 12 2024].
- [16] CircuitBread, "What is a PWM Signal?," 14 2 2022. [Online]. Available: <https://www.circuitbread.com/ee-faq/what-is-a-pwm-signal>. [Accessed 04 01 2025].

Appendix

Task 1 Assembly Code

```
; FILE: Task 1.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 02/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

hex_vals UDATA_SHR
hex_end RES 0x10

portB EQU 0x06
trisB EQU 0x06
status EQU 0x03
RP0 EQU 5
RP1 EQU 6

inner EQU 0x1C
outer EQU 0x1D

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits
; and setting portB pins as outputs. Then the RAM block is initialised with the display values,
; after calling the subroutine display. The main loop iterates over so the display counts up in
; hexadecimal in 0.2 second increments.
; Register Usage:
```

```

;      w -> 'Working register' stores values temporarily
;      status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
;      trisB -> Sets the data direction (input/output) of portB
;      portB -> Accesses data pins of portB
;      FSR -> Addresses indirect data memory (data pointer)
;      INDF -> Accesses data at FSR address (non-physical register)

main
    BCF      status,RP0      ; Subroutine main
    BCF      status,RP1      ; Clear RP0 bit in the status register
    BSF      status,RP0      ; Clear RP1 bit in the status register
    MOVLW    0x00            ; Set RP0 bit in the status register
    MOVWF    trisB           ; Move value 0 to the W register
    BCF      status,RP0      ; Move value 0 to trisB to set portB as all output pins
    CLRF     portB           ; Bit Clear File - Clear RP0 bit in the status register
                                ; Clear current pin values on portB
    CALL     display         ; Call subroutine display

reset
    MOVLM    hex_vals       ; Move hex vals to the W register
    MOVWF    FSR             ; Move hex_vals to FSR register

loop
    MOVF     INDF,0          ; Move FSR value (in INDF) to the W register
    MOVWF    portB           ; Move FSR value to portB to output to the display
    CALL     delay           ; Call subroutine delay
    INCF     FSR              ; Increment the FSR pointer
    MOVF     FSR,0           ; Move FSR value to the W register
    XORLW    hex_end         ; XOR the W register (from FSR) and hex end (from RAM) values to test if equal
    BTFSS    status,Z        ; If address value are equal skip next instruction (Z bit=0 if equal)
    GOTO     loop            ; Return to loop to display the next number
    GOTO     reset           ; Return to reset to start hexadecimal count
    RETURN                  ; Return from subroutine main
; -----

; Subroutine: delay
; Description: This is the delay subroutine, called in the main function for the program. It forms a counting loop
;              to occupy the microprocessor for 0.2 seconds as required.
; Register Usage:
;      w -> 'Working register' stores values temporarily

```

```

;      outer -> Contains the outer loop counter
;      inner -> Contains the inner loop counter

delay      ; Subroutine delay
            ; Move value 255 to the W register
            MOV LW 0xFF
            MOV WF  outer

outerL      ; Move value 255 to the W register
            MOV LW 0xFF
            MOV WF  inner

innerL      ; Decrement inner by 1, skip next instruction if 0
            DECFSZ inner,f
            GOTO  innerL
            ; Return to innerL to repeat count
            DECFSZ outer,f
            GOTO  outerL
            ; Decrement outer by 1, skip next instruction if 0
            ; Return to outerL to repeat count
            RETURN
            ; Return from subroutine delay
; -----

; Subroutine: display
; Description: This is the display subroutine, called in the main function for the program. It stores all the
;              binary sequences for the hexadecimal display in RAM so they can be sequentially outputted on portB.
; Register Usage:
;   w -> 'Working register' stores values temporarily
;   hex_vals -> Contains the addresses of the hexadecimal display sequences

display
            MOV LW b'01000000'
            MOV WF hex_vals
            MOV LW b'01111001'
            MOV WF hex_vals+0x01
            MOV LW b'00100100'
            MOV WF hex_vals+0x02
            MOV LW b'00110000'
            MOV WF hex_vals+0x03
            MOV LW b'00111001'
            MOV WF hex_vals+0x04
            MOV LW b'00100101'
            MOV WF hex_vals+0x05
            MOV LW b'00000010'
            ; Move binary sequence for 0 to the W register
            ; Move binary sequence for 0 to hex_vals (0x0C)
            ; Move binary sequence for 1 to the W register
            ; Move binary sequence for 1 to hex_vals (0x0D)
            ; Move binary sequence for 2 to the W register
            ; Move binary sequence for 2 to hex_vals (0x0E)
            ; Move binary sequence for 3 to the W register
            ; Move binary sequence for 3 to hex_vals (0x0F)
            ; Move binary sequence for 4 to the W register
            ; Move binary sequence for 4 to hex_vals (0x10)
            ; Move binary sequence for 5 to the W register
            ; Move binary sequence for 5 to hex_vals (0x11)
            ; Move binary sequence for 6 to the W register

```

```

MOVWF    hex_vals+0x06
MOVLW    b'01111000',
MOVWF    hex_vals+0x07
MOVLW    b'00000000',
MOVWF    hex_vals+0x08
MOVLW    b'00010000',
MOVWF    hex_vals+0x09
MOVLW    b'00001000',
MOVWF    hex_vals+0x0A
MOVLW    b'00000011',
MOVWF    hex_vals+0x0B
MOVLW    b'01000110',
MOVWF    hex_vals+0x0C
MOVLW    b'00100001',
MOVWF    hex_vals+0x0D
MOVLW    b'00000110',
MOVWF    hex_vals+0x0E
MOVLW    b'00001110',
MOVWF    hex_vals+0x0F
RETURN
END
; Move binary sequence for 6 to hex_vals (0x12)
; Move binary sequence for 7 to the_W register
; Move binary sequence for 7 to hex_vals (0x13)
; Move binary sequence for 8 to the_W register
; Move binary sequence for 8 to hex_vals (0x14)
; Move binary sequence for 9 to the_W register
; Move binary sequence for 9 to hex_vals (0x15)
; Move binary sequence for A to the_W register
; Move binary sequence for A to hex_vals (0x16)
; Move binary sequence for b to the_W register
; Move binary sequence for b to hex_vals (0x17)
; Move binary sequence for C to the_W register
; Move binary sequence for C to hex_vals (0x18)
; Move binary sequence for d to the_W register
; Move binary sequence for d to hex_vals (0x19)
; Move binary sequence for E to the_W register
; Move binary sequence for E to hex_vals (0x1A)
; Move binary sequence for F to the_W register
; Move binary sequence for F to hex_vals (0x1B)
; Return from subroutine display
; End program

```

Task 2 Assembly Code

```
; FILE: Task 2.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 02/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

portA EQU 0x05 ; Pointer to portA memory address
trisA EQU 0x05 ; Pointer to portA data direction register memory address (trisA)
portB EQU 0x06 ; Pointer to portB memory address
trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
status EQU 0x03 ; Pointer to status register memory address (0-7)
RP0 EQU 5 ; Bit number (5) of RP0 in the status register
RP1 EQU 6 ; Bit number (6) of RP1 in the status register

L EQU 0x47 ; Hexadecimal value for display L
r EQU 0x2F ; Hexadecimal value for display r

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits,
; setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
; from the buttons to see if either have been pressed. If the left side button (SW2) is pressed,
; then the letter L is displayed. If the right side button (SW3) is pressed, the r is displayed.
; Register Usage:
; w -> 'Working register' stores values temporarily
; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
; trisa -> Sets the data direction (input/output) of portA
; portA -> Accesses data pins of portA
```

```

;      trisB -> Sets the data direction (input/output) of portB
;      portB -> Accesses data pins of portB

main
;      Subroutine main
;      Clear RP0 bit in the status register
;      Clear RP1 bit in the status register
;      Set RP0 bit in the status register
;      Move value b000000011 to the W register
;      Move value b000000011 to trisa to set portA pins 0 and 1 as inputs
;      Move value b000000000 to the W register
;      Move value b000000000 to trisB to set portB as all output pins
;      Bit Clear File - Clear RP0 bit in the status register
;      Clear current pin values on portA
;      Clear current pin values on portB

loop
    BTFSS    status,RP0
    MOVILW   status,RP1
    BCF      status,RP0
    BSF      status,RP0
    MOVILW   0x03    trisa
    MOVWF    trisa
    MOVILW   0x00    trisB
    MOVWF    trisB

    BCF      status,RP0
    CLRF     portA
    CLRF     portB

    BTFSS    portA,0
    MOVILW   L
    BTFSS    portA,1
    MOVILW   r
    MOVWF    portB
    GOTO     loop

END
; End program

```

Task 3 Assembly Code

```
; FILE: Task 3.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 02/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

portA EQU 0x05 ; Pointer to portA memory address
trisA EQU 0x05 ; Pointer to portA data direction register memory address (trisA)
portB EQU 0x06 ; Pointer to portB memory address
trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
status EQU 0x03 ; Pointer to status register memory address (0-7)
RP0 EQU 5 ; Bit number (5) of RP0 in the status register
RP1 EQU 6 ; Bit number (6) of RP1 in the status register

on EQU 0x00 ; Hexadecimal value for display all on
off EQU 0x7F ; Hexadecimal value for display all off

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits,
; setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
; from the buttons to see if either have been pressed. If the left side button (SW2) is pressed,
; then all the display segments are on. If the right side button (SW3) is pressed, all are off.
; Register Usage:
; w -> 'Working register' stores values temporarily
; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
; trisA -> Sets the data direction (input/output) of portA
; portA -> Accesses data pins of portA
```

```

; trisB -> Sets the data direction (input/output) of portB
; portB -> Accesses data pins of portB

main
    BCF      status,RP0
    BCF      status,RP1
    BSF      status,RP0
    MOVLW    0x03
    MOVWF    trisa
    MOVLW    0x00
    MOVWF    trisB

    BCF      status,RP0
    CLRF     portA
    CLRF     portB

loop
    BTFSS    portA,0
    MOVLW    on
    BTFSS    portA,1
    MOVLW    off
    MOVWF    portB
    GOTO     loop

    END
; Subroutine main
; Clear RP0 bit in the status register
; Clear RP1 bit in the status register
; Set RP0 bit in the status register
; Move value b00000011 to the W register
; Move value b00000011 to trisa to set portA pins 0 and 1 as inputs
; Move value b00000000 to the W register
; Move value b00000000 to trisB to set portB as all output pins
; Bit Clear File - Clear RP0 bit in the status register
; Clear current pin values on portA
; Clear current pin values on portB
; Test pin A0 - if high then skip next instruction, if low then execute
; Move hexadecimal value for display all on to the W register
; Test pin A1 - if high then skip next instruction, if low then execute
; Move hexadecimal value for display all off to the W register
; Move W register value to portB to output to display
; Return to loop to display again
; End program

```

Task 4 Assembly Code

```
; FILE: Task 4.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 02/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

num_vals
num_end

UDATA_SHR
RES 0x0A

portA EQU 0x05
trisa EQU 0x05
portB EQU 0x06
trisB EQU 0x06
status EQU 0x03
RP0 EQU 5
RP1 EQU 6

inner EQU 0x1C
outer EQU 0x1D

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits,
; setting portA pins as inputs and setting portB pins as outputs. The number 5 is initially
; displayed. Then it polls over the input from the buttons to see if either have been pressed.
; If the left side button (SW2) is pressed, the display will decrement the number (min is 0).
; If the right side button (SW3) is pressed, the display will increment the number (max is 9).
```

```

; Register Usage:
;   w -> 'Working register' stores values temporarily
;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
;   trisa -> Sets the data direction (input/output) of portA
;   porta -> Accesses data pins of portA
;   trisb -> Sets the data direction (input/output) of portB
;   portb -> Accesses data pins of portB
;   FSR -> Addresses indirect data memory (data pointer)
;   INDF -> Accesses data at FSR address (non-physical register)

main
    ; Subroutine main
    BCF    status,RP0    ; Clear RP0 bit in the status register
    BCF    status,RP1    ; Clear RP1 bit in the status register
    BSF    status,RP0    ; Set RP0 bit in the status register
    MOVLW  0x03          ; Move value b00000011 to the W register
    MOVWF  trisa         ; Move value b00000011 to trisa to set portA pins 0 and 1 as inputs
    MOVLW  0x00          ; Move value b00000000 to the W register
    MOVWF  trisb         ; Move value b00000000 to trisb to set portB as all output pins
    BCF    status,RP0    ; Clear RP0 bit in the status register
    CLRF   porta         ; Clear current pin values on portA
    CLRF   portb         ; Clear current pin values on portB

    CALL   display       ; Call subroutine display
    MOVLW  num_vals+0x05 ; Move num_vals index 5 to the W register
    MOVWF  FSR           ; Move num_vals index 5 to FSR register
    MOVF   INDF,0         ; Move INDF value to the W register
    MOVWF  portb         ; Move FSR value to portB to output to the display

loop
    BTFSS  portA,0       ; Test pin A0 - if high then skip next instruction, if low then execute
    CALL   countD        ; Call subroutine countD
    BTFSS  portA,1       ; Test pin A1 - if high then skip next instruction, if low then execute
    CALL   countU        ; Call subroutine countU
    GOTO   loop         ; Go to loop line
; -----

; Subroutine: countD
; Description: This is the countD subroutine, called in the main function for the program. It counts down the

```

```

; number on the display, decreasing by 1 for each related button press, down to 0.
; Register Usage:
;   w -> 'Working register' stores values temporarily
;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
;   portB -> Accesses data pins of portB
;   FSR -> Addresses indirect data memory (data pointer)
;   INDF -> Accesses data at FSR address (non-physical register)

countD
    MOVF    FSR,0
    XORLW   0x0C
    BTFSS   status,Z
    DECF    FSR,1
    MOVF    INDF,0
    MOVWF   portB
    CALL    delay
    RETURN

; -----
; Subroutine: countU
; Description: This is the countU subroutine, called in the main function for the program. It counts up the
; number on the display, increasing by 1 for each related button press, up to 9.
; Register Usage:
;   w -> 'Working register' stores values temporarily
;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
;   portB -> Accesses data pins of portB
;   FSR -> Addresses indirect data memory (data pointer)
;   INDF -> Accesses data at FSR address (non-physical register)

countU
    MOVF    FSR,0
    XORLW   0x15
    BTFSS   status,Z
    INCF    FSR,1
    MOVF    INDF,0
    MOVWF   portB
    CALL    delay
    RETURN

```



```

MOVWF num_vals+0x02
MOVLW b'00110000',
MOVWF num_vals+0x03
MOVLW b'00011001',
MOVWF num_vals+0x04
MOVLW b'00010010',
MOVWF num_vals+0x05
MOVLW b'00000010',
MOVWF num_vals+0x06
MOVLW b'01111000',
MOVWF num_vals+0x07
MOVLW b'00000000',
MOVWF num_vals+0x08
MOVLW b'00010000',
MOVWF num_vals+0x09
RETURN
END
; Move binary sequence for 2 to num_vals (0x0E)
; Move binary sequence for 3 to the W register
; Move binary sequence for 3 to num_vals (0x0F)
; Move binary sequence for 4 to the W register
; Move binary sequence for 4 to num_vals (0x10)
; Move binary sequence for 5 to the W register
; Move binary sequence for 5 to num_vals (0x11)
; Move binary sequence for 6 to the W register
; Move binary sequence for 6 to num_vals (0x12)
; Move binary sequence for 7 to the W register
; Move binary sequence for 7 to num_vals (0x13)
; Move binary sequence for 8 to the W register
; Move binary sequence for 8 to num_vals (0x14)
; Move binary sequence for 9 to the W register
; Move binary sequence for 9 to num_vals (0x15)
; Return from subroutine display
; End program

```

Task 5 Assembly Code

```
; FILE: Task 5.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 03/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

portA EQU 0x05 ; Pointer to portA memory address
trisA EQU 0x05 ; Pointer to portA data direction register memory address (trisA)
portB EQU 0x06 ; Pointer to portB memory address
trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
status EQU 0x03 ; Pointer to status register memory address (0-7)
RP0 EQU 5 ; Bit number (5) of RP0 in the status register
RP1 EQU 6 ; Bit number (6) of RP1 in the status register

on EQU 0xFF ; Hexadecimal value for piezo buzzer on
off EQU 0x7F ; Hexadecimal value for piezo buzzer off

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits,
; setting portA pins as inputs and setting portB pins as outputs. Then it polls over the input
; from the button (SW2) to see if it's been pressed. While pressed, the piezo buzzer will sound.
; Register Usage:
; w -> 'Working register' stores values temporarily
; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
; trisA -> Sets the data direction (input/output) of portA
; portA -> Accesses data pins of portA
; trisB -> Sets the data direction (input/output) of portB
```

```

; portB -> Accesses data pins of portB

main
; Subroutine main
; Clear RP0 bit in the status register
BCF status,RP0
; Clear RP1 bit in the status register
BCF status,RP1
; Set RP0 bit in the status register
BSF status,RP0
; Move value b000000001 to the W register
MOVLW 0x01
; Move value b000000001 to trisa to set portA pins 0 as an input
MOVWF trisa
; Move value b00000000 to the W register
MOVLW 0x00
; Move value b00000000 to trisB to set portB as all output pins
MOVWF trisB

; Clear RP0 bit in the status register
BCF status,RP0
; Clear current pin values on portA
CLRF porta
; Clear current pin values on portB
CLRF portB

loop
; Test pin A0 - if high then skip next instruction, if low then execute
BTFSF porta,0
; Move hexadecimal value for piezo buzzer on to the W register
MOVLW on
; Test pin A0 - if low then skip next instruction, if high then execute
BTFSF porta,0
; Move hexadecimal value for piezo buzzer off to the W register
MOVLW off
; Move W register value to portB to output to buzzer
MOVWF portB
; Return to loop
GOTO loop

; End program
END

```

Task 6 Assembly Code

```
; FILE: Task 6.asm
; AUTHOR: Caitlin Lonsdale
; DATE: 03/01/2025

LIST p=16F84 ; Select microprocessor type
RADIX hex ; Define hexadecimal as the number base

#include p16f84.inc ; Find and include the microprocessor file
; -----

cyc_vals UDATA_SHR
cyc_end RES 0x0A ; Declare the start of a section of shared uninitialised data
; Pointer to reserved memory block (10B)
; Pointer to memory address directly after data being used

portB EQU 0x06 ; Pointer to portB memory address
trisB EQU 0x06 ; Pointer to portB data direction register memory address (trisB)
status EQU 0x03 ; Pointer to status register memory address (0-7)
RP0 EQU 5 ; Bit number (5) of RP0 in the status register
RP1 EQU 6 ; Bit number (6) of RP1 in the status register

on EQU 0xFF ; Hexadecimal value for piezo buzzer on
off EQU 0x7F ; Hexadecimal value for piezo buzzer off
duty EQU 0x1C ; Length of duty cycle
temp EQU 0x1D ; Length of temp duty cycle

ORG 0x00 ; Start the program at 0x00
; -----

; Subroutine: main
; Description: This is the main function for the program. It starts by initialising the status register bits,
; and setting portB as all output pins. It sets the buzzer off (via the portB output) to start.
; It produces a fixed frequency pulse with a decreasing duty cycle (PWM - pulse width modulation)
; which loops repeatedly, and can be heard on the buzzer.
; Register Usage:
```

```

;
; w -> 'Working register' stores values temporarily
; status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
; trisB -> Sets the data direction (input/output) of portB
; portB -> Accesses data pins of portB
; FSR -> Addresses indirect data memory (data pointer)
; INDF -> Accesses data at FSR address (non-physical register)

main
    BCF      status,RP0
    BCF      status,RP1
    BSF      status,RP0
    MOVLW    0x00
    MOVWF    trisB
    BCF      status,RP0
    CLRF     portB

    ; Subroutine main
    ; Clear RP0 bit in the status register
    ; Clear RP1 bit in the status register
    ; Set RP0 bit in the status register
    ; Move value 0 to the W register
    ; Move value 0 to trisB to set portB as all output pins
    ; Clear RP0 bit in the status register
    ; Clear current pin values on portB

    CALL     cycles
    MOV LW   cyc_vals
    MOVWF    FSR
    MOVF     INDF,0
    MOVWF    duty

reset
    MOVLW    off
    MOVWF    portB
    MOVLW    cyc_vals
    MOVWF    FSR

loop
    MOVF     INDF,0
    MOVWF    duty
    CALL     pwm
    INCF     FSR
    MOVF     FSR,0
    XORLW    cyc_end
    BRFS     status,Z
    GOTO     loop
    GOTO     reset
    RETURN
; -----

```

```

; Subroutine: pwm
; Description: This is the pwm subroutine, called in the main function for the program. It sets the output on
; portB (pin 7) to high/low according to the duty cycle set by the subroutine cycles.
; Register Usage:
;   w -> 'Working register' stores values temporarily
;   status -> Contains the ALU arithmetic status, RESET status, and bank select bit for data memory
;   trisB -> Sets the data direction (input/output) of portB
;   portB -> Accesses data pins of portB

pwm      MOVF    duty,0      ; Move duty value to the W register
         MOVWF   temp       ; Move duty value to temp
         MOVLW   on        ; Move buzzer on value to the W register
         MOVWF   portB      ; Move buzzer on value to portB

pwmHi    DECFSZ  temp       ; Decrement temp value by 1, if zero the skip next instruction
         GOTO    pwmHi      ; Return to pwmHi
         MOVLW   0xFF      ; Move value 255 to the W register
         MOVWF   temp       ; Move value 255 to temp
         MOVF    temp,0     ; Move duty value to the W register
         SUBWF   temp,1     ; Subtract duty value from temp, store the result in temp
         MOVLW   off       ; Move buzzer off value to the W register
         MOVWF   portB     ; Move buzzer off value to portB

pwmLo    DECFSZ  temp       ; Decrement temp value by 1, if zero the skip next instruction
         GOTO    pwmLo      ; Return to pwmLo
         RETURN             ; Return from subroutine pwm
; -----

; Subroutine: cycles
; Description: This is the cycles subroutine, called in the main function for the program. It sets the
; hexadecimal values for the duty cycles 100-10% (10% decrements). It stores all of these
; in RAM so they can be accessed and used for the values duty and temp.
; Register Usage:
;   w -> 'Working register' stores values temporarily
;   cyc_vals -> Contains the addresses of the duty cycle values

cycles   ; Subroutine cycles

```

```

MOVW 0x0F      ; Move 100% duty cycle (255) to the W register
MOVWF cyc_vals ; Move 100% duty cycle (255) to cyc_vals index 0
MOVW 0xE5      ; Move 90% duty cycle (229) to the W register
MOVWF cyc_vals+0x01 ; Move 90% duty cycle (229) to cyc_vals index 1
MOVW 0xCC      ; Move 80% duty cycle (204) to the W register
MOVWF cyc_vals+0x02 ; Move 80% duty cycle (204) to cyc_vals index 2
MOVW 0xB2      ; Move 70% duty cycle (178) to the W register
MOVWF cyc_vals+0x03 ; Move 70% duty cycle (178) to cyc_vals index 3
MOVW 0x99      ; Move 60% duty cycle (153) to the W register
MOVWF cyc_vals+0x04 ; Move 60% duty cycle (153) to cyc_vals index 4
MOVW 0x7F      ; Move 50% duty cycle (127) to the W register
MOVWF cyc_vals+0x05 ; Move 50% duty cycle (127) to cyc_vals index 5
MOVW 0x65      ; Move 40% duty cycle (101) to the W register
MOVWF cyc_vals+0x06 ; Move 40% duty cycle (101) to cyc_vals index 6
MOVW 0x4C      ; Move 30% duty cycle (76) to the W register
MOVWF cyc_vals+0x07 ; Move 30% duty cycle (76) to cyc_vals index 7
MOVW 0x32      ; Move 20% duty cycle (50) to the W register
MOVWF cyc_vals+0x08 ; Move 20% duty cycle (50) to cyc_vals index 8
MOVW 0x19      ; Move 10% duty cycle (25) to the W register
MOVWF cyc_vals+0x09 ; Move 10% duty cycle (25) to cyc_vals index 10
RETURN          ; Return from subroutine cycles

END            ; End program

```

Caitlin Lonsdale - ACDA4004 - Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

75/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 42

PAGE 43

PAGE 44

PAGE 45

PAGE 46

PAGE 47

PAGE 48

PAGE 49

GRADING FORM: GJ_2019

CAITLIN LONSDALE

373

ENGAGEMENT WITH LITERATURE



Excellent range of relevant sources cited

80

KNOWLEDGE AND UNDERSTANDING



Very good level of K&U derived from the EWL. Very well explained through the use of the diagrams.

80

COGNITIVE AND INTELLECTUAL

 Some very interesting solutions to the programming tasks - shows a good level of intellect

75

PRACTICAL APPLICATION

 Circuit very well implemented. Good attempt at all the programming questions, although some flaws in the understanding - e.g. the buzzer needs to be vibrated, not just switched on and off, also - why was the PWM implemented in steps, rather than by ± 1 ??

70

TRANSFERABLE SKILLS

 Generally well laid out and presented. Could have had a more detailed TOC though.

68