



MICROS PERIPHERALS & INTERFACES – ACDA4004

Using and programming the 7-segment display
prototype circuit.

Contents

1. Introduction	3
2. Section 1	3
2.1 Overview of the Assignment Board	3
2.2 Proteus Circuit	4
2.3 Component Analysis	5
2.3.1 PIC16F84A	5
2.3.2 External Oscillator Clock	6
2.3.3 Seven-Segment Display	6
2.3.4 Resistor Network	7
2.3.5 Input Switches	8
2. Section 2	8
3.1 Hexadecimal Number Overview	8
3.2 Segment Encoding for Hexadecimal Characters	9
3. Section 3	10
4.1 Task 1	10
4.1.1 Task Overview	10
4.1.2 Program Logic	10
4.1.3 Assembly Code	11
4.1.4 Simulation Result	13
4.2 Task 2	15
4.2.1 Task Overview	15
4.2.2 Program Logic	15
4.2.3 Assembly Code	16
4.2.4 Simulation Result	17
4.3 Task 3	19
4.3.1 Task Overview	19
4.3.2 Program Logic	19
4.3.3 Assembly Code	20
4.3.4 Simulation Result	21
4.4 Task 4	23
4.4.1 Task Overview	23
4.4.2 Program Logic	23

4.4.3 Assembly Code	24
4.4.4 Simulation Result	26
References	27

1. Introduction

Microcontroller-based systems are widely used within electronics and digital applications to provide a highly flexible, programmable control over electronic systems. One such application is with the seven-segment LED display and using the display to present alphanumeric information to the user.

The following report focuses on the use and programming of the seven-segment display using the PIC16F84A microcontroller, simulated within the Proteus design software, and programmed using Assembly code. The original hardware circuit is recreated within the simulation software to analyse the roles and functions of the individual components within the system. Exploring how to represent various hexadecimal characters on the display and controlling the display through the use of input switches, including counting in sequence, displaying characters and latching behaviours.

2. Section 1

2.1 Overview of the Assignment Board

The original assignment board, recreated in figure 1, showed the microcontroller, driven by a crystal oscillator, connected to the seven-segment display with a current-limiting resistor network in series connected to the PORTB pins, with the final pin connected to the speaker output circuit, and the input switches connected to first two pins in PORTA.

The input switches were set up in a “pull up”, meaning the microcontroller would be configured to detect when the input pins are pulled low.

Pin B7 is connected to the base of a transistor acting as a switch, enabling the use of components that would typically require a higher voltage than would be possible with output of the microcontroller.

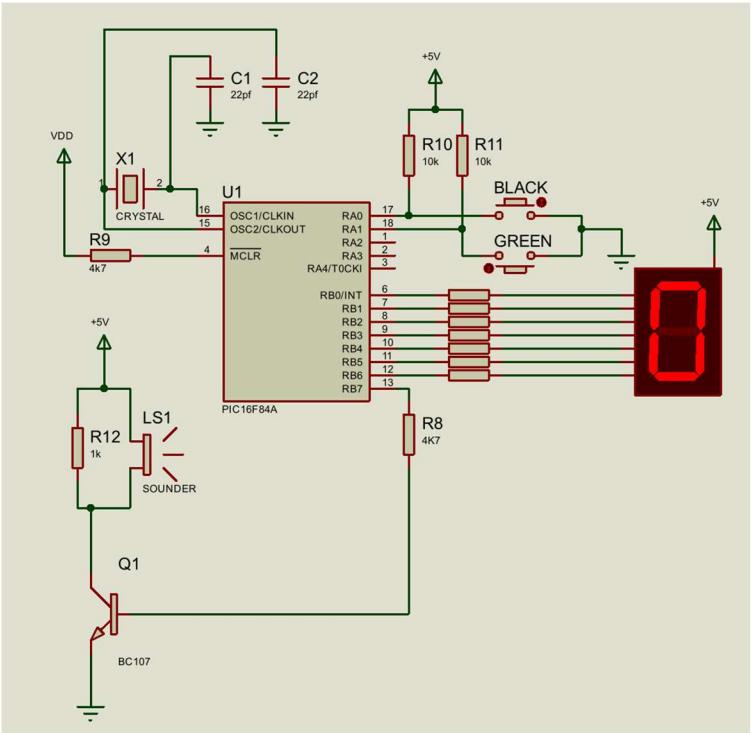


Figure 1: Recreation of the Assignment Board built within proteus.

2.2 Proteus Circuit

The circuit shown below in Figure 2, shows the annotated version of the assignment circuit, clearly showing the key areas of the circuit.

Power is supplied through the MCLR pin for the microcontroller; any additional power sources are supplied separately to the required areas, and labels are used in place of wires in places to provide clear indication of the connections in an easily digestible format.

A resistor network was used in place of the individual resistors shown in Figure 1, to better match the physical circuit and to provide easier adjustment to the values of the resistors, as since they are only required to be current limiting resistors, the current necessary would be the same across the output pins.

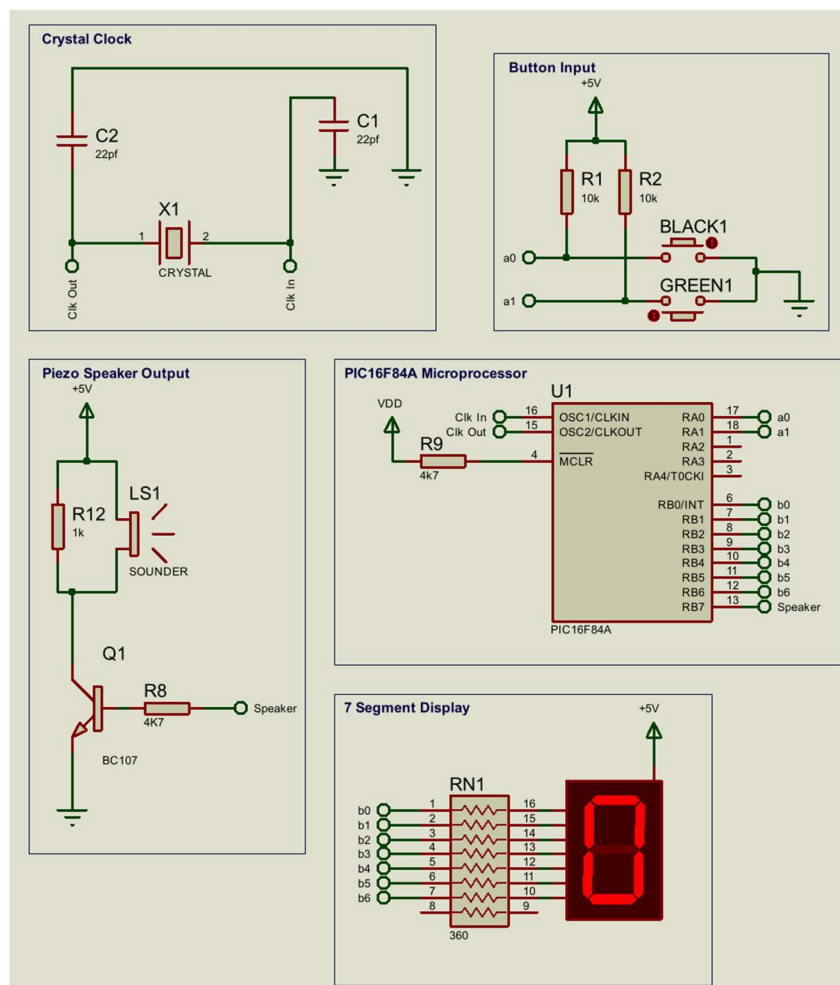


Figure 2: Annotated version of figure 1, breaking the circuit into key areas.

2.3 Component Analysis

2.3.1 PIC16F84A

The PIC16F84A is used as the central control unit for the system, executing the set program and controlling the output signals sent to the seven-segment display and the output speaker.

The microcontroller features [1]:

- 13 Programmable I/O pins
- Harvard Architecture providing separate program and data memory.
- An internal instruction cycle derived from an external clock source.
- 1000 words of program memory
 - This translates into 1024 instructions since each program memory word is the same width as each device memory instruction.
- A simple but effective instruction set, consisting of 35 overall instructions.

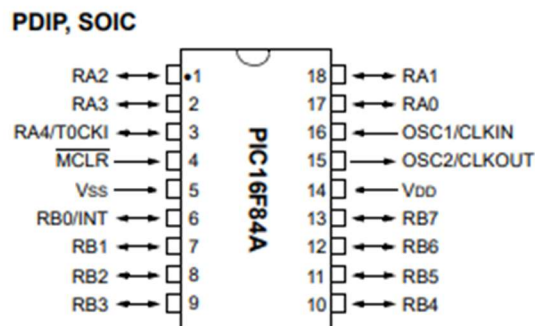


Figure 3: Pinout of the PIC16F84A

In this circuit, PORTB is configured to be the output, driving the seven-segment display and the Piezo Speaker, while PORTA is configured to handle the switch inputs. Each output corresponds to a segment on the display, allowing for individual control through software.

2.3.2 External Oscillator Clock

The clock source provides a timing reference for the microcontroller, determining how fast instructions are executed and directly affects delay routines within the program [2].

In this circuit, the crystal oscillator is connected to the PIC's clock input pins. The oscillator defines the instruction cycle time, which is essential when implementing precise delays and timings such as when configuring the program for PWM control.

An accurate clock ensures consistent and predictable system behaviour, both within the proteus simulation and in hardware applications.

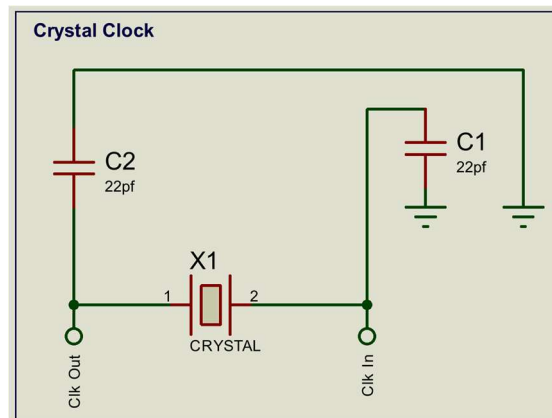


Figure 4: External Crystal Oscillator, providing a stable clock signal.

2.3.3 Seven-Segment Display

The seven-segment display is used to visually represent limited alphanumeric characters. It consists of seven individual LEDs arranged in a pattern with each LED segment labelled A through G.

By selectively illuminating the segments, characters and digits can be displayed. Within this circuit, the display is controlled through the output pins on the microcontroller, with each segment connected to a corresponding output pin.

The display used in this circuit uses a common anode, this determines whether high or low logic is required to illuminate each segment, in this case, low logic is required. This means that when a pin is driven low, the segment will illuminate.

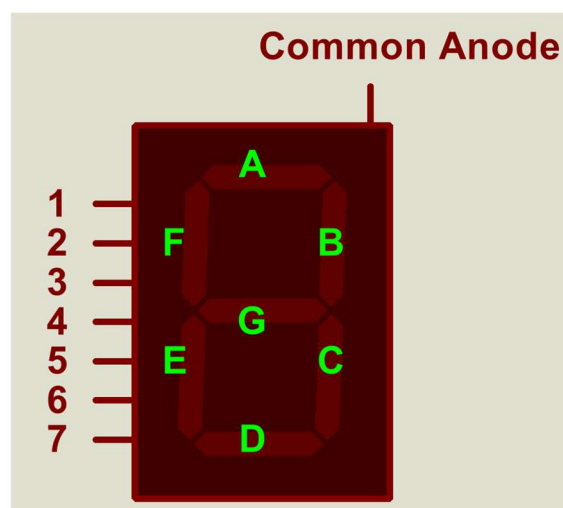


Figure 5: Segment Labelling of the Seven-Segment Display.

2.3.4 Resistor Network

Each segment of the seven-segment display is connected in series to a pin of a resistor network, which acts as a network of current-limiting resistors, protecting the LEDs from excessive current, which could cause damage in a practical application.

To calculate the resistance required for each pin, the following equation is required [3]:

$$R = \frac{V_s - V_f}{I_f} \quad (1)$$

Where:

- R represents resistance in Ohms
- V_s is supply voltage in volts.
- V_f is the LEDs forward voltage drop in Volts.
- I_f is the forward current through the LED in Amps

In this case, the resulting resistance required is 350 Ohms, as the supply voltage is 5V, the LED forward voltage is 1.5V and the current required to turn on the LED is 10mA, resulting in 350 Ohms. This is a standard value in the E96 and E192 series of resistors, however in the E24 series 360 is the nearest preferred value, such 360 Ohms is used within the circuit.

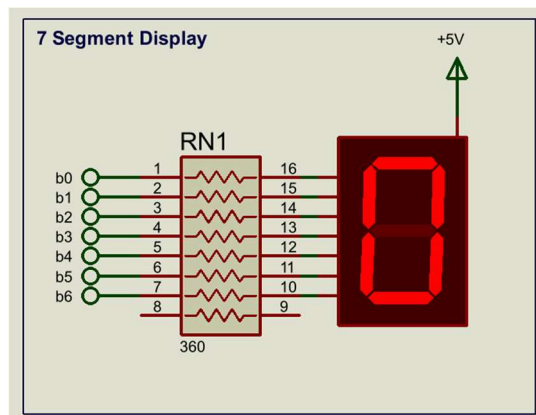


Figure 6: The seven segment display, showing the resistor network in series.

2.3.5 Input Switches

The push buttons provide user input to the circuit, allowing for the inputs to be programmed, enabling functions such as incrementing counts, toggling outputs, and changing display modes. The switches in this circuit are configured for a logic low input.

In this circuit, the switches are configured with pull-up resistors, meaning the input pins read a logic high when the switches are not pressed, and is then pulled low when the switch is pressed.

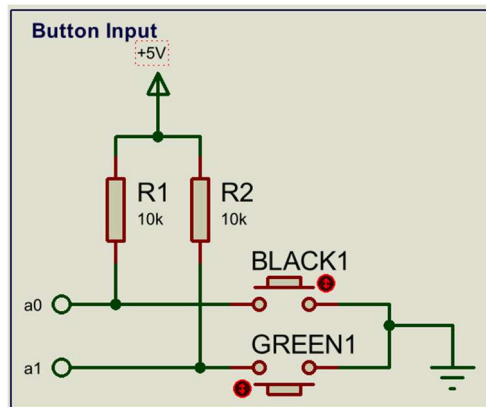


Figure 7: The input switches, labelled in line with the physical circuit.

2. Section 2

3.1 Hexadecimal Number Overview

The hexadecimal number system is commonly used in digital electronics and provides a shorthand method of representing binary values, able to show 16 digits, ranging from 0 to 15.

Values 0-9 can be represented as a standard digit, as they are single characters, however numbers with multiple characters, 10 and above, are instead represented as letters A-F, where:

- A = 10
- B = 11
- C = 12
- D = 13
- E = 14
- F = 15

Each hexadecimal digit can be represented using 4 bits, or a nibble, meaning that a full hexadecimal value can be equated to one byte of data.

3.2 Segment Encoding for Hexadecimal Characters

A seven-segment display consists of seven individual segments, labelled A – G, and by illuminating specific combinations of these segments, different characters can be displayed. Characters B and D must be displayed in lower case, as on a seven-segment display they look like 8 and 0, respectively.

Each of these segments correlate to each of the pins, so by telling the microcontroller the order of which to enable the pins, a number can be displayed on the display, this output can be defined through a hexadecimal number when using an 8-pin output.

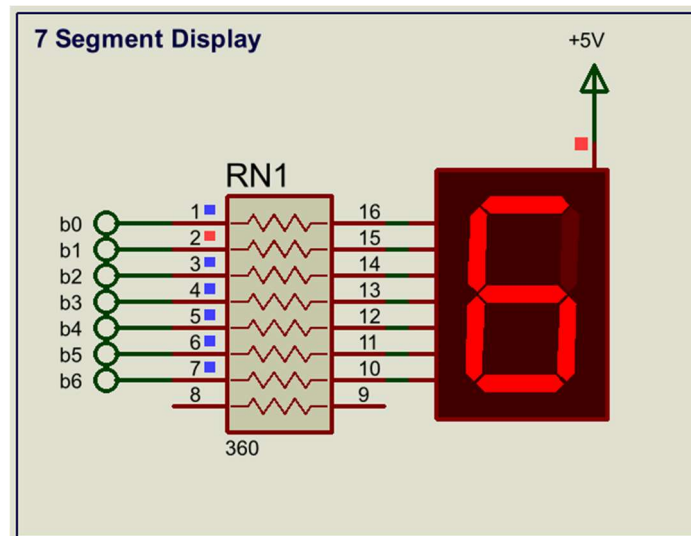


Figure 9: A seven-segment display displaying the number 6.

	SECOND DIGIT				FIRST DIGIT				
	01	02	04	08	01	02	04	08	
	a	b	c	d	e	f	g		HEX
0	0	0	0	0	0	0	1	0	0x40
1	1	0	0	1	1	1	1	0	0x79
2	0	0	1	0	0	1	0	0	0x24
3	0	0	0	0	0	1	1	0	0x30
4	1	0	0	1	1	0	0	0	0x19
5	0	1	0	0	1	0	0	0	0x12
6	0	1	0	0	0	0	0	0	0x02
7	0	0	0	1	1	1	1	0	0x78
8	0	0	0	0	0	0	0	0	0x00
9	0	0	0	1	1	0	0	0	0x18
A	0	0	0	1	0	0	0	0	0x08
B	1	1	0	0	0	0	0	0	0x03
C	0	1	1	0	0	0	1	0	0x46
D	1	0	0	0	0	1	0	0	0x21
E	0	1	1	0	0	0	0	0	0x06
F	0	1	1	1	0	0	0	0	0x0E

Table 1: A table showing the hexadecimal representation of each desired character, and how they correlate to each output pin.

Table 1 shows the binary and hexadecimal representation of each character, which when processed by the microcontroller can be applied to the output pins of a port to display certain characters. This is assuming the display used has a common anode, and such any outputs will be inverted. For example,

0 = 00000010 or 0x40, which when applied to the seven-segment display will hold all pins low except for the pin correlating to segment G, which will light all segments except from G, displaying 0.

3. Section 3

4.1 Task 1

4.1.1 Task Overview

This task requires the modification of an existing assembly code to increment the display from 0 to F with a delay of 0.2 seconds, which then resets upon exceeding F, repeating indefinitely.

4.1.2 Program Logic

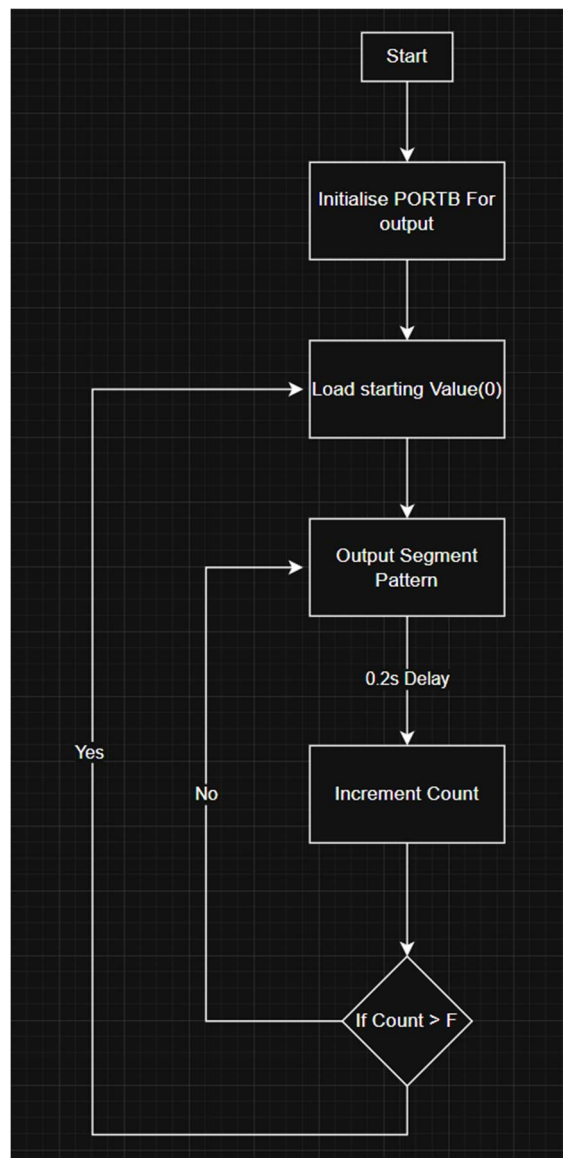


Figure 10: A flowchart outlining the logic of the intended program.

The program will initialise PORTB for output, then load the initial starting value of 0 or 0x40, which will be output to the seven-segment display, where a 0.2s delay will be in place before incrementing count and then outputting this to the display after checking if count is greater than the value of F, if it is not then the program will loop, if it is, then count will be reset to 0, and the program will begin anew.

4.1.3 Assembly Code

```
1 ; FILE: count.asm
2 ; AUTHOR: JMY
3 ; DATE: 24th March 2005
4 ; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
5 ; segment display in binary.
6 ; The pause subroutine takes apx. 0.2 sec.
7 ; Pause is called 5 times
8
9 list p=16F84 ; select processor type
10 radix hex ; define radix as hexadecimal
11
12 ;-----
13 ; cpu equates (memory map)
14 ;-----
15 portB equ 0x06 ; port B address
16 trisb equ 0x06 ; trisB address in register bank 1
17
18 Status equ 0x03 ; status register
19 RP0 equ 5 ; bit 5 of status register
20 RP1 equ 6 ; bit 6 of status register
21
22 INDF equ 0x00 ; indirect file register
23 FSR equ 0x04 ; file select register
24 PCL equ 0x02 ; program counter low byte
25
26 count equ 0x0c ; counter 0 to 255
27 nCount equ 0x0d ; pause routine inner loop counter
28 mCount equ 0x0e ; pause routine outer loop counter
29 tablestart equ 0x10 ; constant location of lookup table
30
31 ;-----
32 ;-----
33
34 org 0x000 ; Marks the beginning of the program
35
36 start
37 bcf Status,RP0 ; clear bank selection bits in the status reg
38 bcf Status,RP1
39
40 bsf Status,RP0 ; select bank 1
41 movlw 0x00 ; Load w with 0x00
42 movwf trisb ; load it into trisb to make portb all outputs
43 bcf Status,RP0 ; reselect bank 0
44
45 clrf portB ; clear all lines low
46 clrf count ; initialize counter to 0
47 goto get_count ; moves to the main loop
48
49 ;-----
50 ;-----
51 ; Main program loop
52 ;-----
53
54 get_count
55
56 movf count,w ; get count value
57 call sevensegtable ; converts the value of count to a hexadecimal value
58 movwf portB ; put contents of lookup table in portB
59
60 incf count,f ; increments the count
61 movlw 0x0F ; keeps the count in the 0-15 range
62 andwf count,f
63
64 call longpause ; 200ms delay
65 goto get_count ; repeat forever
66
67 ;-----
68 ; Seven-segment display lookup table
69 ; w must contain a value between 0 and 15 before calling
70 ; addwf pcl,f performs a calculated jump, to land on the specific instruction in the lookup table
71 ;-----
72
73 sevensegtable
```

Figure 11: Task 1 screenshot 1 of 2, showing the modified and commented code for the task.

```

72 sevensegtable
73     addwf    PCL,f    ; adds the value of the working to file to perform a calculated jump and land on the correct instruction
74     retlw    0x40     ; 0
75     retlw    0x79     ; 1
76     retlw    0x24     ; 2
77     retlw    0x30     ; 3
78     retlw    0x19     ; 4
79     retlw    0x12     ; 5
80     retlw    0x02     ; 6
81     retlw    0x78     ; 7
82     retlw    0x00     ; 8
83     retlw    0x18     ; 9
84     retlw    0x08     ; A
85     retlw    0x03     ; B
86     retlw    0x46     ; C
87     retlw    0x21     ; D
88     retlw    0x06     ; E
89     retlw    0x0E     ; F
90
91 ;-----
92 ; Short delay subroutine
93 ; Creates a delay using nested decrement loops
94 ;-----
95
96 pause
97     movlw    0xff      ; set w = 255 decimal
98     movwf    mCount    ; mCount = w: outer loop counter
99 loadN
100    movlw    0xff      ; set w = 255 decimal
101    movwf    nCount    ; nCount = w: inner loop counter
102 decN
103    decfsz   nCount, f  ; nCount--
104    goto     decN      ; if nCount != 0 then repeat nCount--
105
106    decfsz   mCount, f  ; else decrement Count
107    goto     loadN     ; if mCount != 0 then
108                    ; reload nCount to 255 and decrement
109    return
110
111 ;-----
112 ; Long delay subroutine
113 ; Calls pause multiple times to extend delay
114 ;-----
115
116 longpause
117     call    pause
118     call    pause
119     call    pause
120     call    pause
121     call    pause
122     return
123
124     end        ; End of Program
125
126
127

```

Figure 12: Task 1 screenshot 2 of 2, showing the modified and commented code for the task.

At the start of the program, PORTB and TRISB are set to the same address, then the direction of portB is set using “movlw 0x00” and “movwf trisb” which sets all of the PORTB pins to output, or 0 on the tris register.

Then using a lookup table labelled “sevensegtable”, with the hexadecimal values of 0-15, the lookup table uses the “movf count,w” and “get_count” functions to load the current value of “count” into the working register, then calls the lookup table, and inside the table adds the current value of count(W) to the program counter, causing the program to “jump” forward by W(value of count, currently stored in the working register) number of entries. This value is then stored in the working(W) register.

“Movwf portB” is then used to move the value of W to portB, which displays the stored value on the seven-segment display. The value of count is then incremented by one, the program experiences a 200ms delay, and begins the loop again, until reaching 15, where count wraps around to 0.

4.1.4 Simulation Result

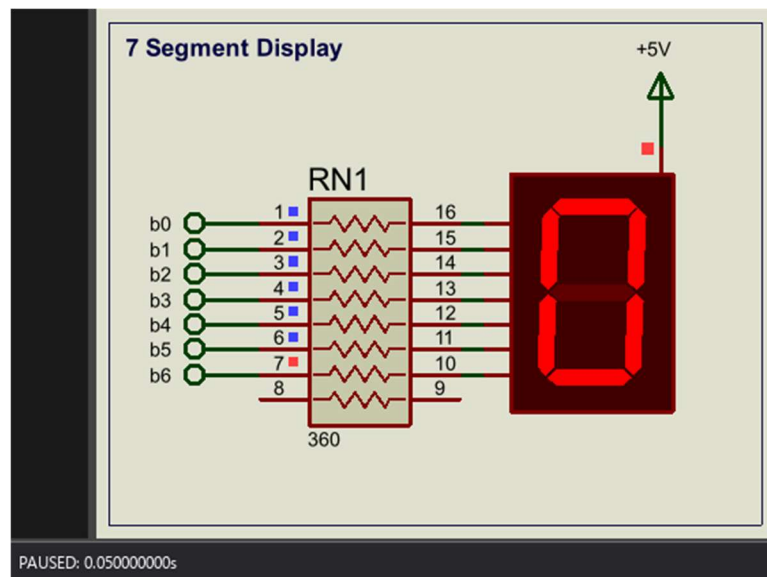


Figure 13: Start of simulation.

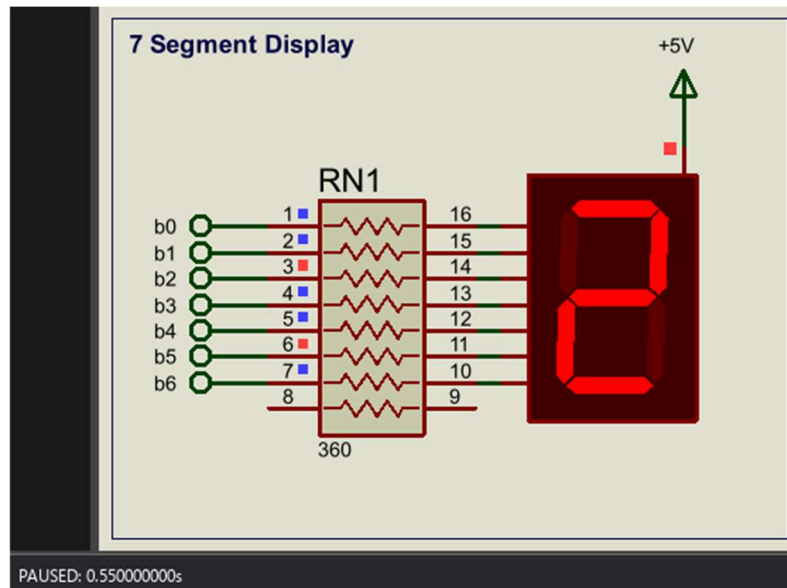


Figure 14: Simulation at 550ms.

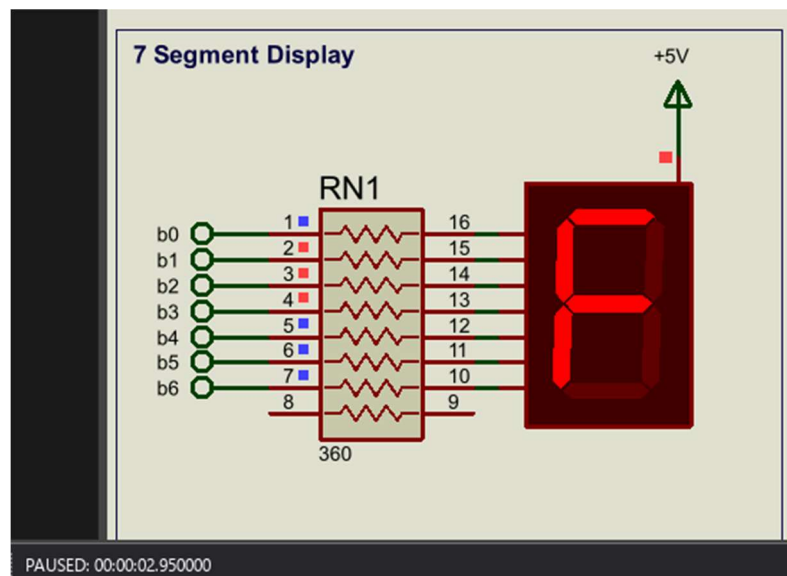
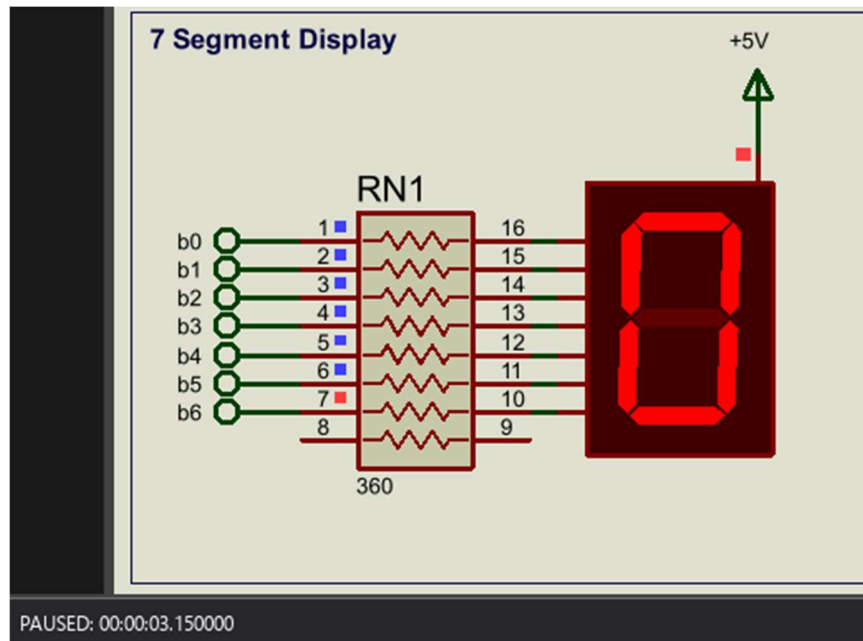


Figure 16: Simulation at 2.95s.



PAUSED: 00:00:03.150000

Figure 17: Simulation at 3.15s showing successful wrap.

4.2 Task 2

4.2.1 Task Overview

Task 2 calls for a program wherein the seven-segment display shows either an L or an R depending on which input is pressed.

4.2.2 Program Logic

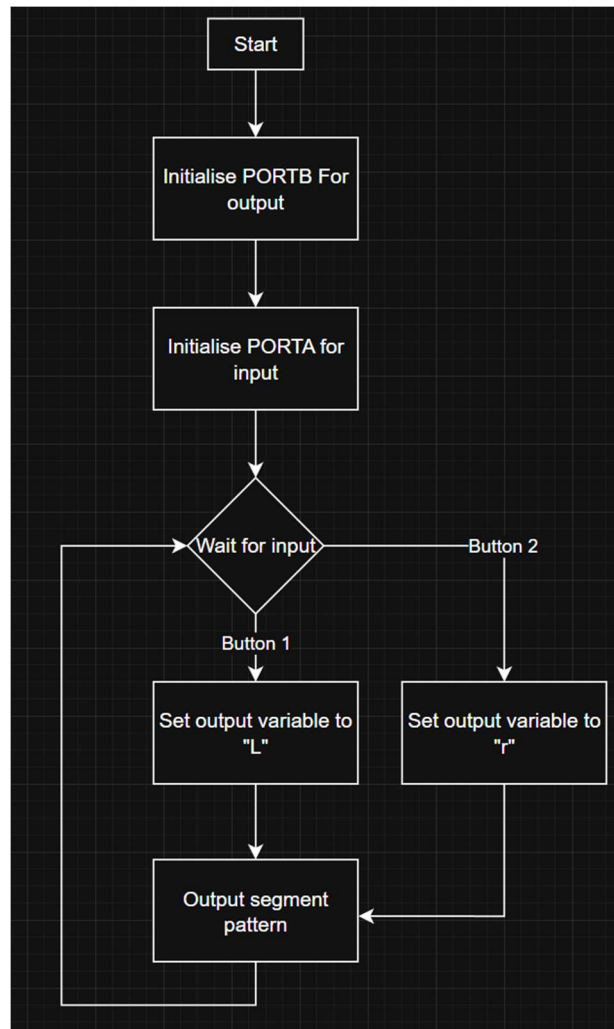


Figure 18: A flow diagram outlining the logic of the logic of the program.

The program will initialise ports B and A for output and input respectively, where it will then wait for input from portA, and upon receiving an input, display the corresponding output on the display, then return to awaiting input.

4.2.3 Assembly Code

```
1      list    p=16F84          ; select processor type
2      radix   hex              ; define radix as hexadecimal
3
4      ;-----
5      ; Register and I/O definitions
6      ;-----
7
8      portA   equ    0x05      ; port A address
9      portB   equ    0x06      ; port B address
10
11     trisa    equ    0x05      ; TRISA address in register bank 1
12     trisb    equ    0x06      ; TRISB address in register bank 1
13
14     Status   equ    0x03      ; status register
15     RP0      equ    5         ; defines RP0 to bit 5 of status register
16     RP1      equ    6         ; defines RP1 to bit 6 of status register
17
18     seg_L     equ    0x47      ; Segment Pattern to show L - stored as seg_L
19     seg_r     equ    0x2F      ; Segment Pattern to show r - stored as seg_r
20
21     org       0x000          ; program starts and resets at 0x000
22
23     ;-----
24     ; Initialisation
25     ;-----
26
27     start
28         bsf      Status,RP0    ; switch to Bank 1 to configure I/O direction
29
30         movlw    b'00000011'   ; set RA0 and RA1 as inputs stores as working file
31         movwf    trisa         ; write working to TRISA
32
33         movlw    0x00          ; sets PORTB to be all outputs stores as working file
34         movwf    trisb         ; write working to TRISB
35
36         bcf      Status,RP0    ; back to Bank 0
37
38         movlw    0xFF          ; turns off all segments initially, stores value as working file
39         movwf    portB         ; writes working to PORTB
40
41     ;-----
42     ; Main program Loop
43     ; Continuously polls the two buttons waiting for an input
44     ;-----
45
46     main_loop
47         btfss    portA,0       ; checks for input from pin A0,
48                                 ; if no input, skips next instruction
49         call     show_L        ; calls show_L if pinA0 input is true
50
51         btfss    portA,1       ; checks for input from pin A1
52                                 ; if no input, skips next instruction
53         call     show_r        ; calls show_r if pinA1 input is true
54
55         goto     main_loop     ; Loops function
56
57     ;-----
58     ; Waits for input A0 to be released, then displays 'L'
59     ;-----
60
61     show_L
62     wait_L_release
63         btfss    portA,0       ; wait until RA0 returns to high(button released)
64         goto     wait_L_release
65
66         movlw    seg_L         ; Loads segment pattern for L
67         movwf    portB         ; outputs pattern to port B
68         return                ; return to main Loop
69
70     ;-----
71     ; Waits for input A1 to be released, then displays 'r'
72     ;-----
73
74     show_r
75     wait_r_release
76         btfss    portA,1       ; wait until RA1 returns to high(button released)
77         goto     wait_r_release
78
79         movlw    seg_r         ; Loads segment pattern for L
80         movwf    portB         ; outputs pattern to port B
81         return                ; return to main Loop
82
83     end
```

Figure 19: Task 2 Assembly code.

The program begins by defining ports A and B to and then defining the segment hex values under an easily referenceable definition allowing for future changes if necessary. Then the input and output pins are initialised, setting pins A0 and A1 as input pins, and all pins on port B to the output.

The main loop works by continuously polling the input pins for a change in status, and depending on the input, runs the correlating subroutine.

The show_L and show_r subroutines are identical in how they work, with the only difference being the input pin and the file to be moved to working, where they wait for the input to be released, then brings the value of seg_L or seg_r and moves it to the working register to be displayed in portB, before returning to the main loop

4.2.4 Simulation Result

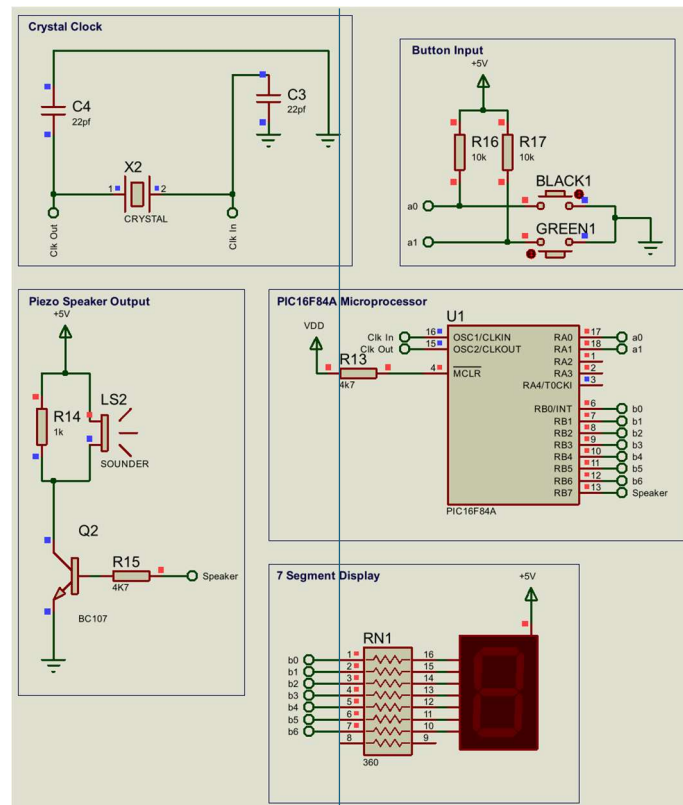


Figure 20: simulation with no inputs.

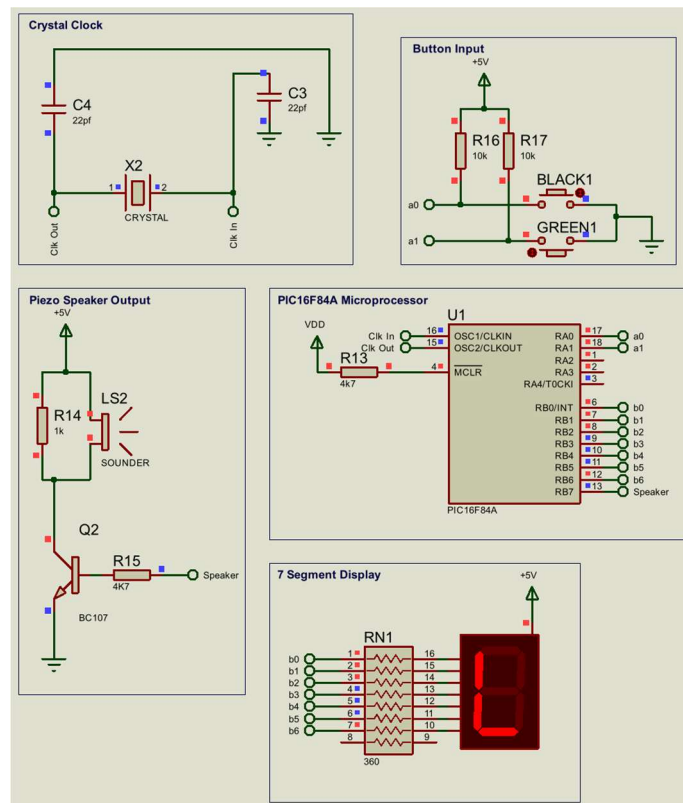


Figure 21: Simulation after input A0 has been pushed.

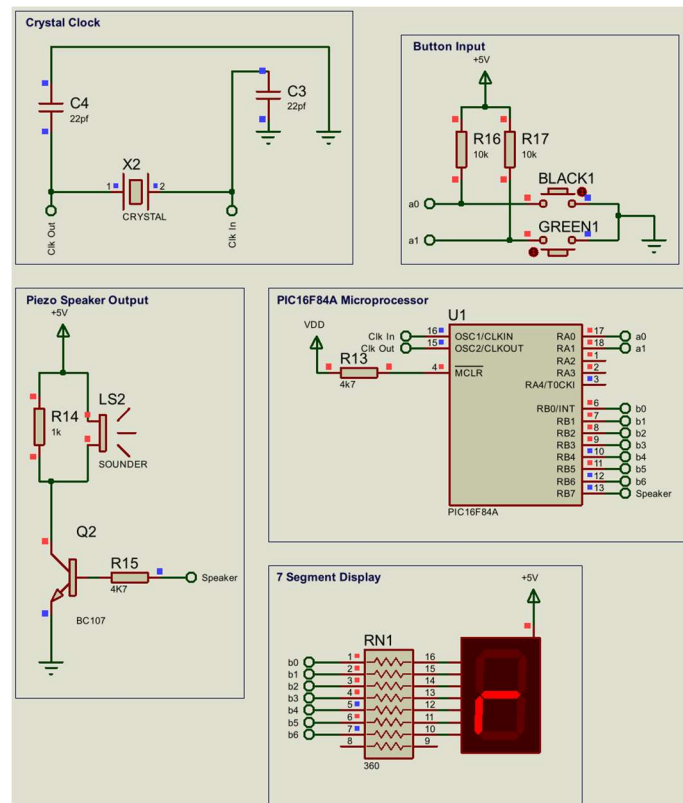


Figure 22: Simulation after input A1 has been pushed.

4.3 Task 3

4.3.1 Task Overview

This task calls for a program wherein a latching effect is put in place, so that when one button is pressed all pins are set high, and when another is pressed all pins are set low.

4.3.2 Program Logic

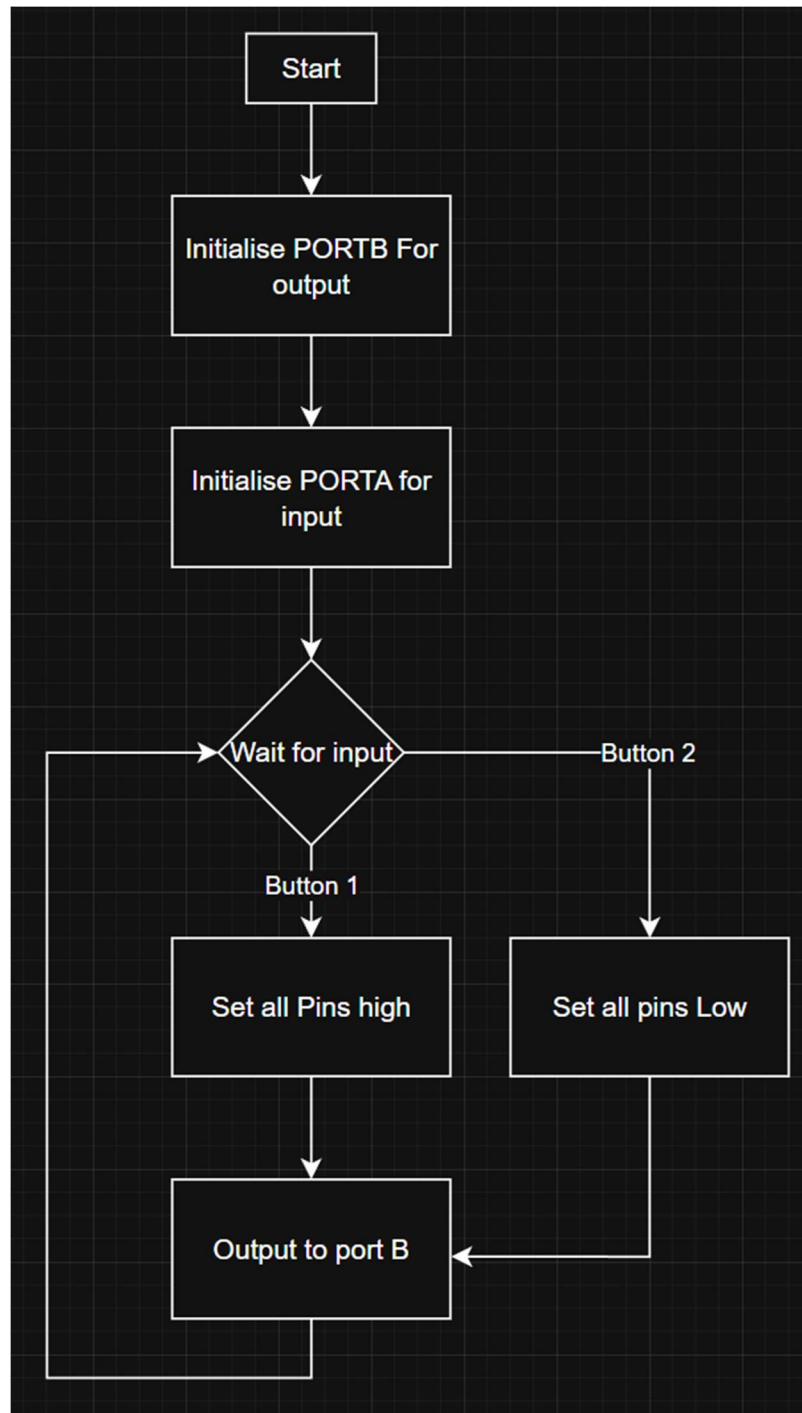


Figure 23: a flowchart of the program logic.

The program will start by initialising ports A and B for input and output respectively, where it will then poll for inputs, setting the pins high when one is pushed, and low when the other is pushed.

4.3.3 Assembly Code

```
1      list    p=16F84          ; select processor type
2      radix   hex              ; define radix as hexadecimal
3
4      -----
5      ; Register and I/O definitions
6      -----
7
8      portA   equ    0x05      ; port A address
9      portB   equ    0x06      ; port B address
10
11     trisa    equ    0x05      ; TRISA address in register bank 1
12     trisb    equ    0x06      ; TRISB address in register bank 1
13
14     Status   equ    0x03      ; status register
15     RP0      equ    5         ; defines RP0 to bit 5 of status register
16     RP1      equ    6         ; defines RP1 to bit 6 of status register
17
18     on       equ    0x00      ; hex value for turning all segments on
19     off      equ    0xFF      ; hex value for turning all segments off
20
21     org      0x000           ; program starts and resets at 0x000
22
23
24     ; Initialisation
25     -----
26
27 start
28     bsf      Status,RP0      ; switch to Bank 1 to configure I/O direction
29     movlw    b'0000011'      ; set RA0 and RA1 as inputs stores as working file
30     movwf    trisa           ; write working to TRISA
31
32     movlw    0x00            ; sets PORTB to be all outputs stores as working file
33     movwf    trisb           ; write working to TRISB
34
35     bcf      Status,RP0      ; back to Bank 0
36
37     movlw    0xFF            ; turns off all segments initially, stores value as working file
38     movwf    portB           ; writes working to PORTB
39
40
41     ;-----
42     ; Main program loop
43     ; Continuously polls the two buttons waiting for an input
44     ;-----
45
46 main_loop
47     btfss    portA,0         ; checks for input from pin A0,
48     ; if no input, skips next instruction
49     call     show_on          ; calls show_on if pinA0 input is true
50
51     btfss    portA,1         ; checks for input from pin A1
52     ; if no input, skips next instruction
53     call     show_off         ; calls show_off if pinA1 input is true
54
55     goto     main_loop        ; Loops function
56
57 ;-----
58 ; Waits for input A0 to be released, then turns all segments on
59 ;-----
60
61 show_on
62 wait_on_release
63     btfss    portA,0         ; wait until RA0 returns to high(button released)
64     goto     wait_on_release
65
66     movlw    on              ; Loads 'on' bringing all pins low
67     movwf    portB           ; outputs pattern to port B
68     return                    ; return to main loop
69
70 ;-----
71 ; Waits for input A1 to be released, then turns all segments off
72 ;-----
73
74 show_off
75 wait_off_release
76     btfss    portA,1         ; wait until RA1 returns to high(button released)
77     goto     wait_off_release
78
79     movlw    off             ; Loads off bringing all pins high
80     movwf    portB           ; outputs pattern to port B
81     return                    ; return to main loop
82
83 end
```

Figure 24: Task 3 assembly code.

The program begins by defining ports A and B to and then defining the on/off values under an easily referenceable definition allowing for future changes if necessary. Then the input and output pins are initialised, setting pins A0 and A1 as input pins, and all pins on port B to the output.

The main loop works by continuously polling the input pins for a change in status, and depending on the input, runs the correlating subroutine.

The show_on and show_off subroutines are identical in how they work, with the only difference being the input pin and the file to be moved to working, where they wait for the input to be released, then brings the value of 'on' or 'off' and moves it to the working register to be displayed in portB, before returning to the main loop.

4.3.4 Simulation Result

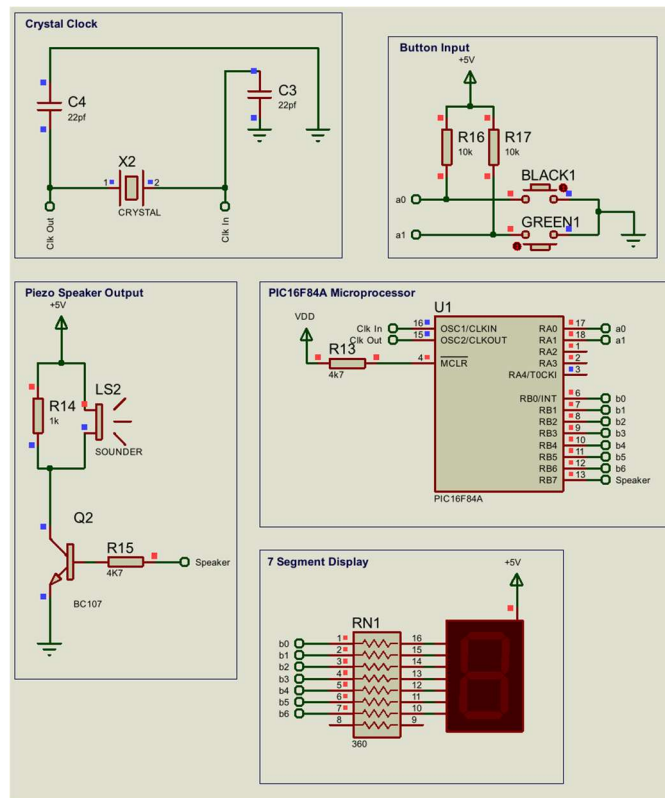


Figure 25: Simulation start.

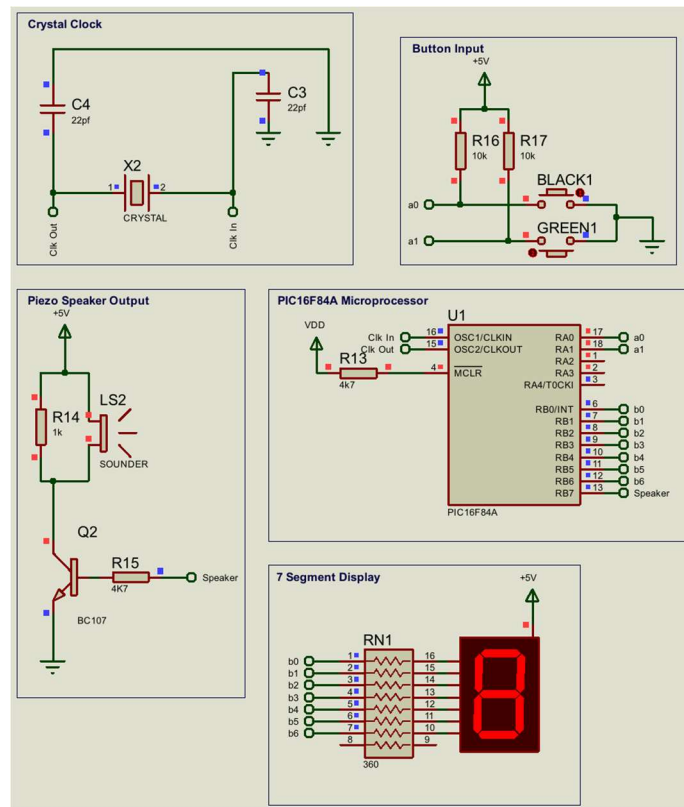


Figure 26: Simulation after input 1.

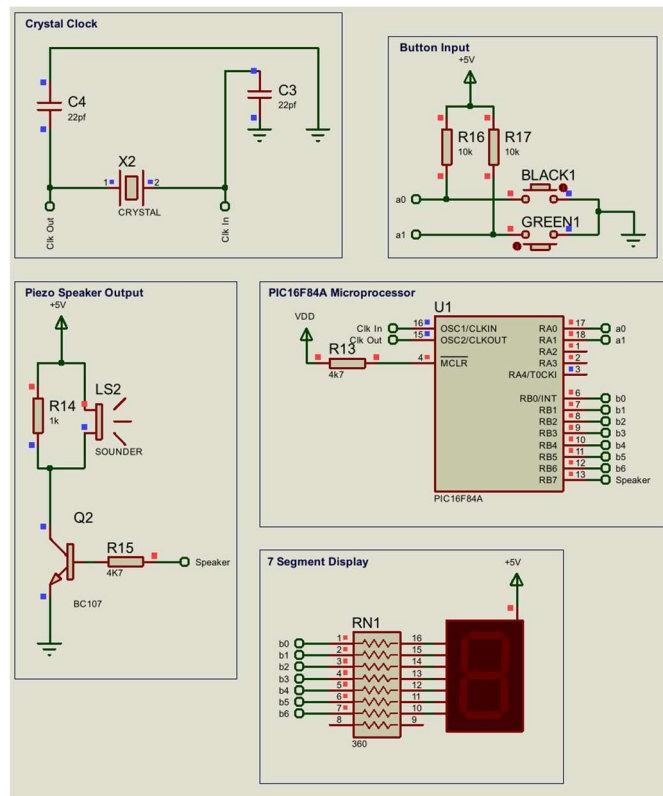


Figure 27: Simulation after input 2.

4.4 Task 4

4.4.1 Task Overview

Task 4 calls for the integration of the components from the previous task to create a counter that starts at 5, counts up to 9 and down to 0, without wrapping.

4.4.2 Program Logic

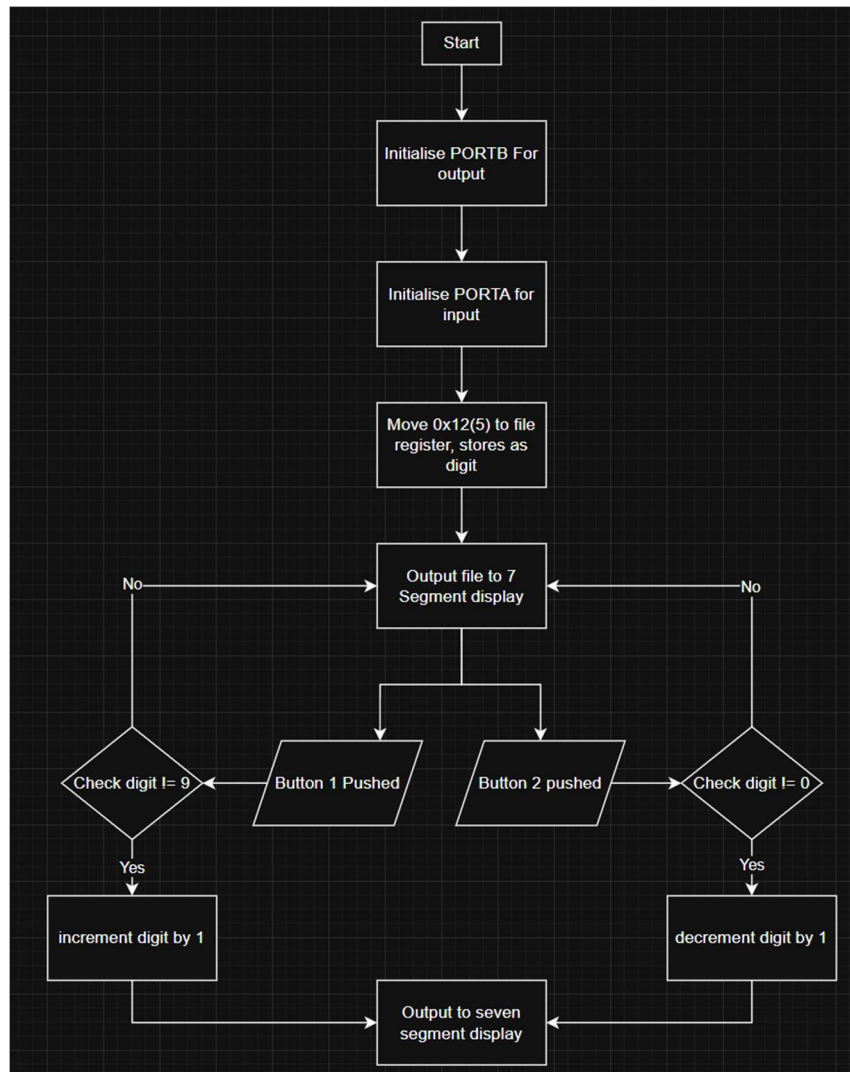


Figure 28: A flow diagram showing the logic of task 4.

The program will start by initialising ports A and B for input and output respectively, then moving the hex value for 5 to the file register, and storing this as the starting digit, where this will be output to the 7-segment display. From this the inputs will be polled to wait for button 1 or 2, if button 1 is pressed, a check is made to see if digit is not 9 yet, if it isn't 9, allows for the value of digit to be incremented and displayed on the display, if it is not, it returns to the previous step. Button 2 checks for 0 instead, if it is not 0, allows digit to be decremented, if it is returns to the previous state, and just displays the current value for digit on the display.

4.4.3 Assembly Code

```
1      list    p=16F84          ; select processor type
2      radix   hex              ; define radix as hexadecimal
3
4      ;-----
5      ; Register and I/O definitions
6      ;-----
7
8      portA   equ    0x05      ; port A address
9      portB   equ    0x06      ; port B address
10
11     trisa    equ    0x05      ; TRISA address in register bank 1
12     trisb    equ    0x06      ; TRISB address in register bank 1
13
14     Status   equ    0x03      ; status register
15     RP0      equ    5         ; defines RP0 to bit 5 of status register
16     RP1      equ    6         ; defines RP1 to bit 6 of status register
17
18
19     PCL      equ    0x02      ; program counter Low (needed for addwf PCL,f)
20
21     digit    equ    0x0C      ; variable holding the current value for 'digit'(between 0 and 9)
22
23     org      0x000           ; program starts and resets at 0x000
24
25     ;-----
26     ; Initialisation
27     ;-----
28
29     start
30         bsf    Status,RP0     ; switch to Bank 1 to configure I/O direction
31
32         movlw   b'0000011'    ; set RA0 and RA1 as inputs stores as working file
33         movwf   trisa         ; write working to TRISA
34
35         movlw   0x00          ; sets PORTB to be all outputs stores as working file
36         movwf   trisb         ; write working to TRISB
37
38         bcf     Status,RP0     ; back to Bank 0
39
40         movlw   0x05          ; moved value 5 to working
41         movwf   digit         ; stores 5 in digit
42
43         call    display       ; calls display to start the seven segment display at 5
44
45     ;-----
46     ; Main program Loop
47     ; Continuously polls the two buttons waiting for an input
48     ;-----
49
50     main_loop
51         btfss   portA,0       ; checks for input from pin A0,
52                                ; if no input, skips next instruction
53         call    count_up      ; calls count_up if pinA0 input is true
54
55         btfss   portA,1       ; checks for input from pin A1
56                                ; if no input, skips next instruction
57         call    count_down    ; calls count_down if pinA1 input is true
58
59         goto    main_loop     ; loops function
60
61     ;-----
62     ; Waits for input A0 to be released, then increments the count
63     ;-----
64
65     count_up
66     wait_up_release
67         btfss   portA,0       ; wait until RA0 returns to high(button released)
68         goto    wait_up_release
69
70         movlw   0x09          ; compares the literal value with 9
```

Figure 29: Task 4 part 1 of 2.

```

70     movlw    0x09          ; compares the literal value with 9
71     subwf    digit,w       ; Working value = digit - 9
72     btfsc    Status,2      ; if digit equals 9, do not increment
73     goto     update_display ; just re-display current value and return, preventing increment
74
75     incf     digit,f        ; digit = digit + 1
76     goto     update_display ; updates display and returns
77
78 ;-----
79 ; Waits for input A1 to be released, then decrement count
80 ;-----
81
82 count_down
83 wait_down_release
84     btfss    portA,1        ; wait until RA1 returns high (button released)
85     goto     wait_down_release
86
87     movf     digit,f        ; sets Zbit value if digit == 0
88     btfsc    Status,2      ; if digit equals 0, do not decrement
89     goto     update_display ; just re-display current value
90
91     decf     digit,f        ; digit = digit - 1
92
93 ;-----
94 ; update display
95 ;-----
96 update_display
97     call     display        ; update 7-seg output
98     return
99
100
101 ;-----
102 ; Uses the value of digit (0-9) to fetch a 7-seg pattern and outputs to PORTB
103 ;-----
104
105 display
106     movf     digit,w        ; W = digit
107     call     DigitTable     ; W = 7-seg pattern for that digit
108     movwf    portB         ; output to display
109     return
110
111 ;-----
112 ; Lookup table holding the hex values of 0 through 9
113 ;-----
114
115 DigitTable
116     addwf    PCL,f
117     retlw    0x40           ; hex value of 0
118     retlw    0x79           ; hex value of 1
119     retlw    0x24           ; hex value of 2
120     retlw    0x30           ; hex value of 3
121     retlw    0x19           ; hex value of 4
122     retlw    0x12           ; hex value of 5
123     retlw    0x02           ; hex value of 6
124     retlw    0x78           ; hex value of 7
125     retlw    0x00           ; hex value of 8
126     retlw    0x18           ; hex value of 9
127
128
129     end
130

```

Figure 30: Task 4 part 2 of 2.

The program works by first initialising the registers and providing definitions for the I/O, where the I/O ports are then set to input on portA and output on portB. The main loop of the code works by polling the input ports and when they are pulled low triggering the count up or count down subroutines. For count up the value of digit is compared against the literal value of 9, where it will force the program to loop if digit == 9, preventing the program from counting up, otherwise increasing the value of digit, which pulls the hex value from the lookup table in reference to the current value. Count down does the same, however it does not need to reference against a value due to the limit being 0.

4.4.4 Simulation Result

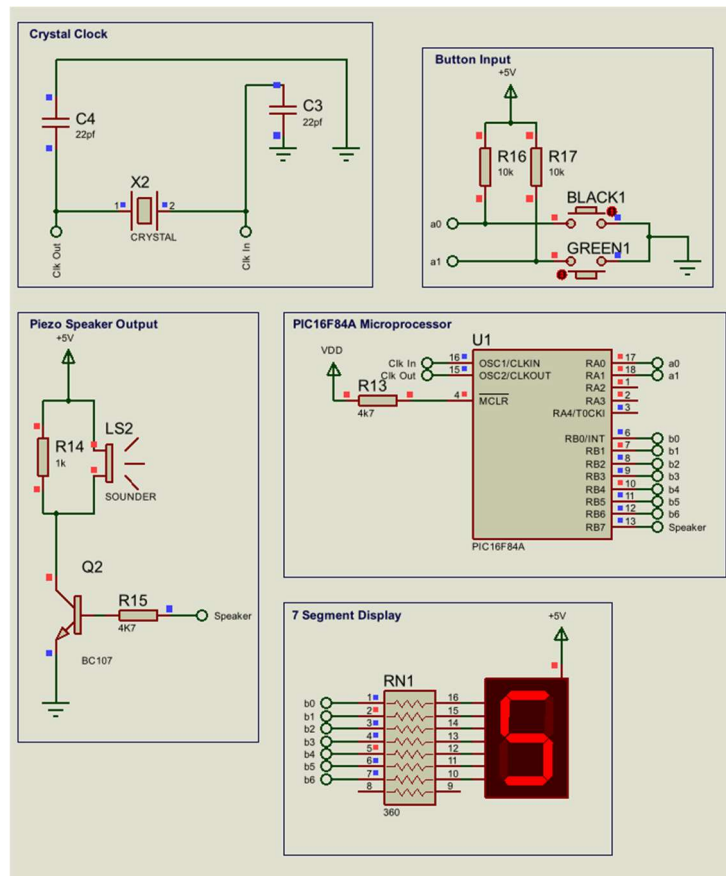


Figure 31: Simulation at the beginning .

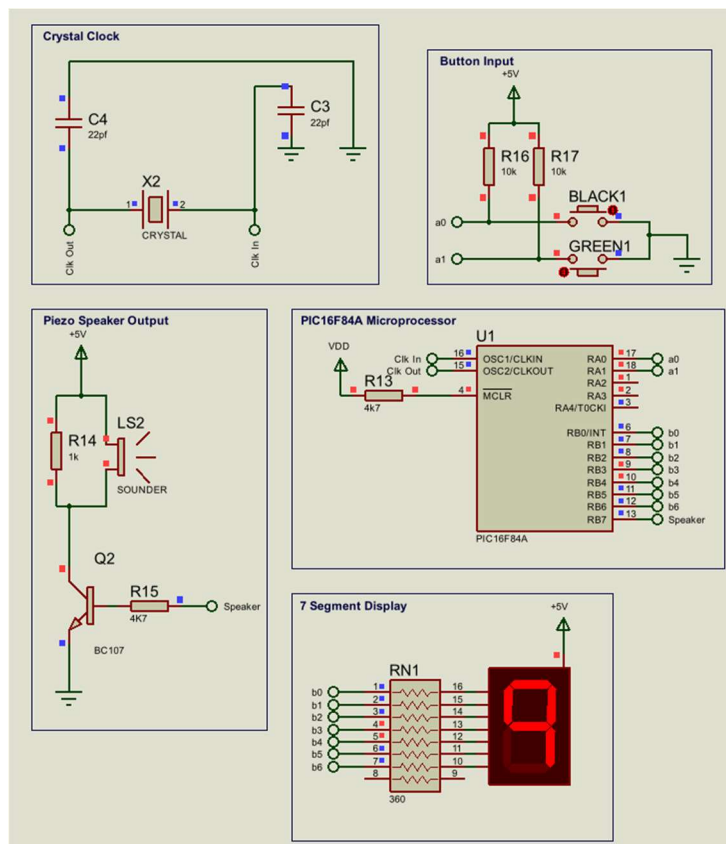


Figure 32: Simulation at the count_up limit.

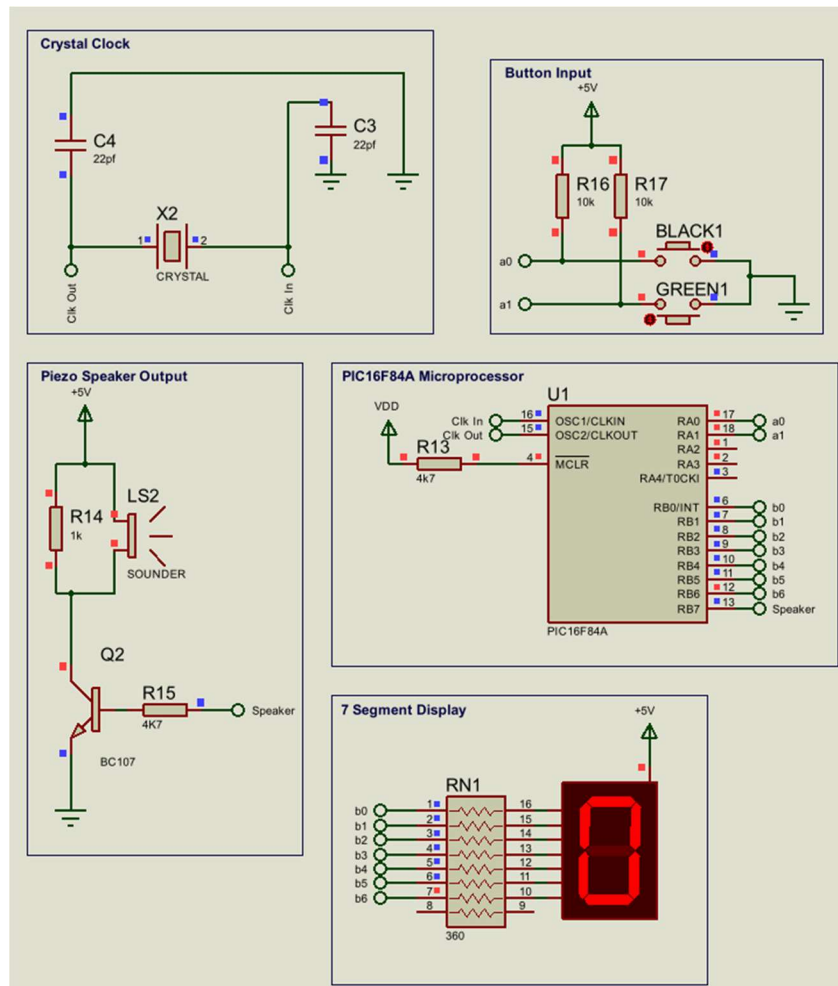


Figure 33: Simulation at count_down limit.

References

- [1] Microchip, "PIC16F84A Data Sheet," Microchip, 2001.
- [2] Electronics Tutorials, "Electronics Tutorials," Electronics Tutorials, November 2019. [Online]. Available: <https://www.electronics-tutorials.ws/oscillator/crystal.html>. [Accessed 21 12 2025].
- [3] RS, "LED Resistor Value Calculator," RS, 23 January 2024. [Online]. Available: <https://uk.rs-online.com/web/content/discovery/tools-and-calculators/led-resistor-calculator>. [Accessed 25 12 21].