

Micros Assignment

by Emma Moseley

Submission date: 13-Jan-2025 08:50AM (UTC+0000)

Submission ID: 248195257

File name: 49299_Emma_Moseley_Micros_Assignment_621295_1835636405.docx (414.63K)

Word count: 5846

Character count: 29958



JANUARY 10, 2025

ASSEMBLY CODE ASSIGNMET

MICROS, PERIPHERALS AND INTERFACES

EMMA MOSELEY (2317703)
BEng Embedded Electronics Systems Engineering

Contents

1	Circuit Design and Component Analysis	4
1.1	PIC16F84A (UI)	4
1.2	Oscillator (X1) and Capacitors (C1, C2)	5
1.3	Seven-Segment LED Display, Resistors (IC2)	6
1.4	Buttons (BUT0, BUT1)	7
1.5	Voltage Regulator (U2) and capacitors (C3, C4).....	7
1.6	Transistor (Q1)	8
1.7	Piezo Sounder (LS1), Switch (SW1)	8
2	Hexadecimal Display	9
3	Programming Logic and Implementation	10
3.1	Counting from 0 to F in Hexadecimal	10
3.1.1	Pseudocode.....	10
3.1.2	Assembly Code	11
3.2	Displaying "L" and "r" Based on Button Presses	13
3.2.1	Pseudocode.....	13
3.2.2	Assembly Code	13
3.3	Toggling All LED Segments On/Off	15
3.3.1	Pseudocode.....	15
3.3.2	Assembly Code	15
3.4	Incrementing and Decrementing a Displayed Number	17
3.4.1	Pseudocode.....	17
3.4.2	Assembly Code	17
3.5	Producing a Tone with the Piezo Sounder	20
3.5.1	Pseudocode.....	20
3.6	Generating a Dynamic PWM Signal	20
3.6.1	Pseudocode.....	20
4	References	22

Table of Figures

Figure 1: Proteus schematic for "Count.asm" program. 4

Figure 2: PIC16F84A microcontroller diagram showing the different pins [1]. 4

Figure 3: Oscillator Circuit [2]..... 5

Figure 4: Common Anode Configuration [3]. 6

Figure 5: Resistor Network for Common Anode [3]..... 6

Figure 6: Button connection in circuit with debounce diagram [4]. 7

Figure 7: Voltage Regulator Diagram [6]. 7

Figure 8: Bi-Polar Junction (BJT) Transistor [8]..... 8

Figure 9: Piezoelectric Buzzer Circuit Diagram [9]. 8

Figure 10: 7-Segment LED display [10]..... 9

Table of Tables

Table 1: Bitwise calculations for the 7-Segment LED display 10

1 Circuit Design and Component Analysis

A Proteus simulation was developed using a PIC16F84A microcontroller to control a seven-segment LED display, and it was saved as "Count.asm". When the simulation starts, various segments of the LED display light up, with the pattern changing every second. This functionality was built in assembly language with the goal of ensuring that basic output was seen on the display.

Although the seven-segment LED is the only output that this program demonstrates, the schematic also allows for user input through buttons and switches and output through a sounder in addition to the seven-segment LED. This section provides a detailed analysis of the circuit components, explaining their individual functions and their roles within the overall design.

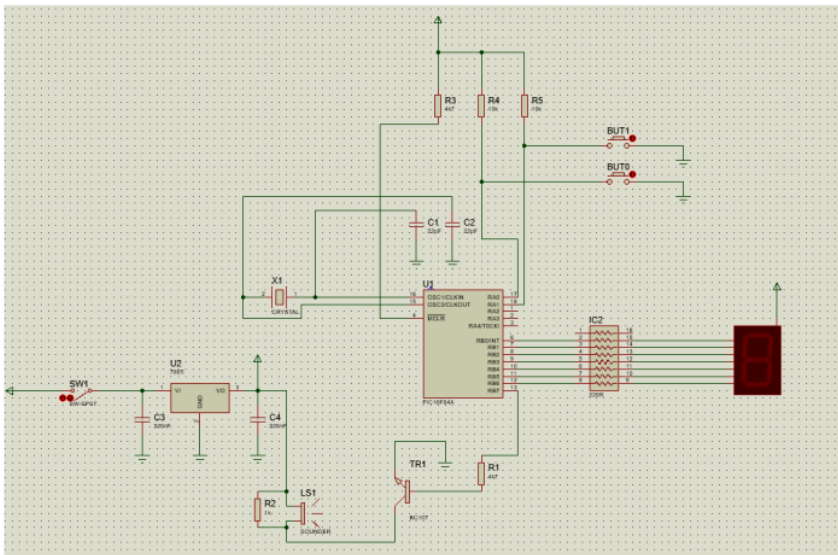


Figure 1: Proteus schematic for "Count.asm" program.

1.1 PIC16F84A (UI)

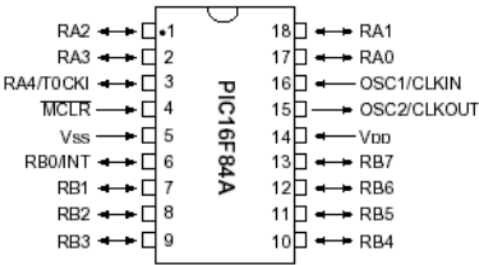


Figure 2: PIC16F84A microcontroller diagram showing the different pins [1].

The circuit's main control unit is an 8-bit microcontroller called the PIC16F84A. The microcontroller in this simulation controls a seven-segment display for visual output and a piezo sounder for audio feedback, with user input provided via buttons and a switch.

As seen in Figure 2, the PIC16F84A microcontroller has a wide variety of pins for different uses. It has 13 digital I/O pins (RA0–RA4 and RB0–RB7) that can be configured as inputs or outputs. For example, RA0 and RA1 control button inputs, while RB0–RB7 powers the display. An external crystal oscillator (OSC1/OSC2) is used by the microcontroller to produce clock signals, and a reset pin (MCLR) enables program restarts. The VDD and VSS pins provide a constant power supply. The microcontroller has 64 bytes of EEPROM for non-volatile storage, 68 bytes of RAM for temporary data storage, and 1,000 words of program memory for instructions [1].

1.2 Quartz Crystal Oscillator (X1) and Capacitors (C1, C2)

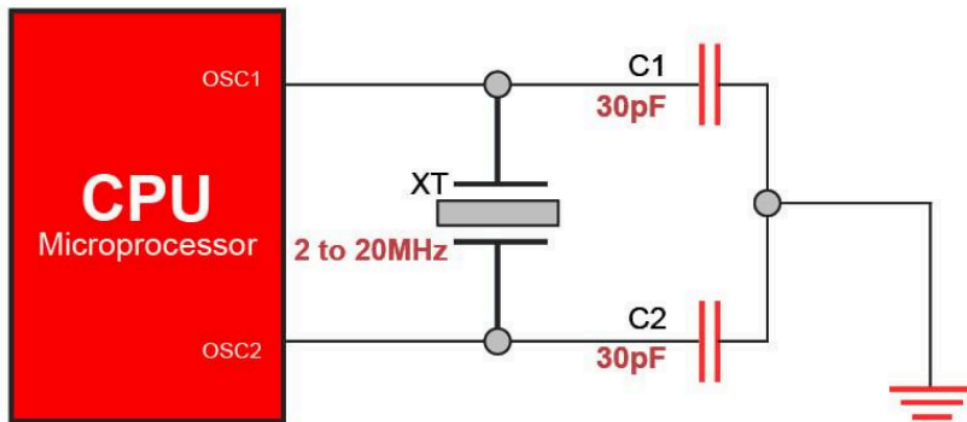


Figure 3: Oscillator Circuit [2]

A quartz crystal (X1/XT) and two capacitors (C1 and C2) make up the oscillator circuit, which provides a clock signal for the microcontroller (as shown in Figure 3). The quartz crystal oscillator generates a stable square wave signal, controlling the microcontroller's instruction execution timing [2]. Capacitors C1 and C2 (22 pF in this circuit) filter noise and stabilise the clock signal, ensuring consistent performance [1]. By using an external oscillator, timing errors are minimised compared to the internal oscillator, making it ideal for time-sensitive tasks requiring precision and reliability [2].

1.3 Seven-Segment LED Display, Resistors (IC2)

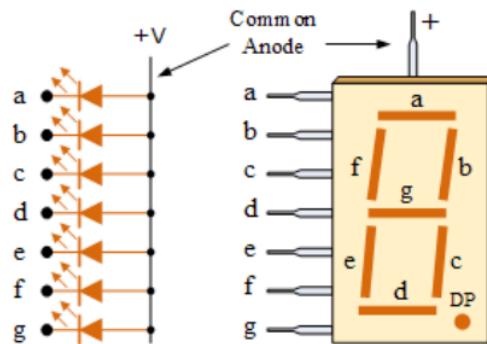


Figure 4: Common Anode Configuration [3].

This circuit's seven-segment LED is a common anode type, which means that each LED segment's anode shares a common connection to the power source (as shown in Figure 4) [3]. The cathode of each segment is linked to a different PIC microcontroller pin. By applying a logic level of 0 to the appropriate pins, the microcontroller determines which segments light up, enabling the display to show characters or numbers as required. Logic level 1 on the other hand turns off segments.

An 8-block resistor (IC2), which combines eight separate resistors into one unit, connects each segment of the seven-segment display to a PIC microcontroller pin (see Figure 5). By limiting the current to each LED segment, this component maintains consistent brightness while safeguarding the microcontroller and the LEDs [3].

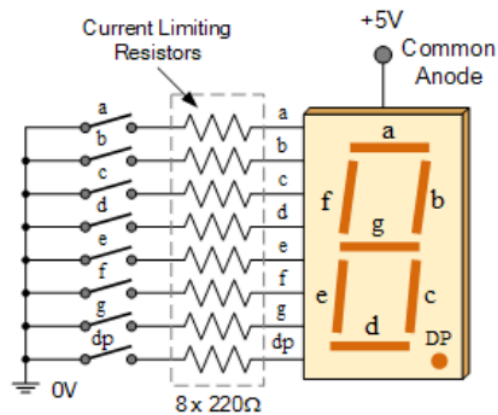


Figure 5: Resistor Network for Common Anode [3].

1.4 Buttons (BUT0, BUT1)

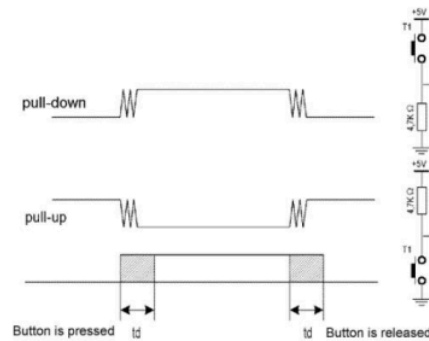


Figure 6: Button connection in circuit with debounce diagram [4].

The circuit's push buttons deliver signals to the microcontroller, with BUT0 connected to RA0 and BUT1 to RA1 on the PIC16F84A. When pressed, the buttons complete the circuit and enable the microcontroller to monitor and respond to signal changes [4]. The circuit's buttons allow interaction with the seven-segment LED display. For reliable signal detection, software debouncing reduces the impact of rapid or noisy button presses, preventing false triggers [5]. Figure 6 shows the debounce signal received when a button is pressed.

1.5 Voltage Regulator (U2) and capacitors (C3, C4)

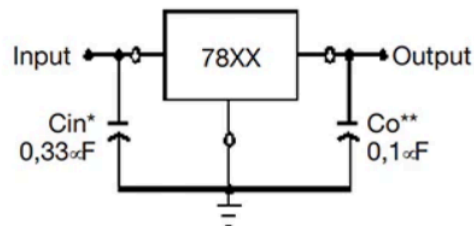


Figure 7: Voltage Regulator Diagram [6].

The voltage regulator maintains a consistent output voltage despite variations in input voltage or load conditions [6]. In this circuit, a 7805 voltage regulator is used to regulate the input voltage, to the 5V required for the PIC16F84 microcontroller to function properly.

To further stabilise the power supply, capacitors C3 and C4 are used in the circuit. C3 (220nF) is set on the input side of the regulator to reduce noise and moderate voltage variations. C4 (220nF) is set on the output side to further stabilise the voltage. These capacitors help filter out fluctuations from the regulator, resulting in a stable and reliable power supply [6]. Figure 7 shows how the voltage regulator and capacitors connect in the circuit.

1.6 Transistor (Q1)

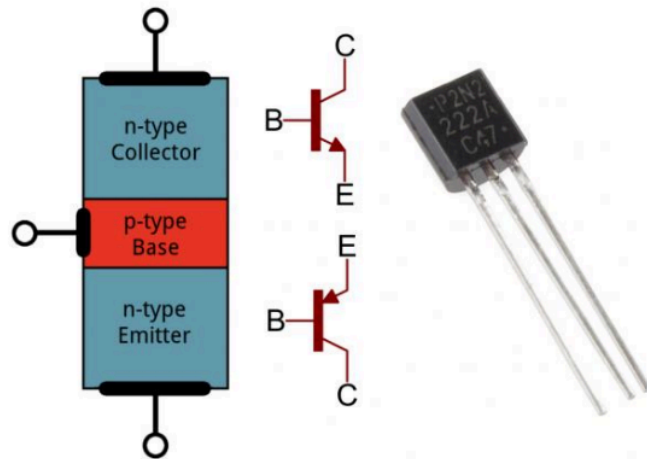


Figure 8: Bi-Polar Junction (BJT) Transistor [8].

The circuit's transistor (Q1) works as a switch to handle the piezo sounder's current requirements, which the PIC16F84 microcontroller cannot directly provide [7]. The transistor enables the microcontroller to regulate the sounder without overloading its output pins. This circuit uses an NPN transistor [8]. The microcontroller sends a signal to the transistor's base, directing current flow from the collector (connected to the sounder) to the emitter (connected to ground), activating the sounder while protecting the microcontroller.

1.7 Piezo Sounder (LS1), Switch (SW1)

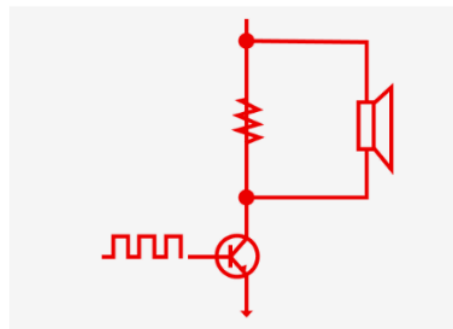


Figure 9: Piezoelectric Buzzer Circuit Diagram [9].

The piezo sounder (LS1) provides audio output in the circuit and is controlled by the PIC16F84 microcontroller via RB7. A resistor links RB7 to the base of an NPN transistor, which controls the current flowing to the sounder. The sounder is connected to the voltage regulator (U2) via a parallel resistor and grounded by the transistor. When the microcontroller signals the transistor's base,

current flows through the sounder, resulting in sound [9]. To prevent damage, the parallel resistor restricts the current. A user-controlled switch (SW1) turns on or off the sounder which provides audible output.

2 Hexadecimal Display

To display hexadecimal characters (0 through F) on a seven-segment display, the binary patterns for each character must be mapped to the display's relevant segments. A seven-segment display is made up of seven segments (as shown in Figure 3) that can be separately controlled to produce numbers and letters. In a common anode setup, each segment's anode is connected to a common positive voltage, while the microcontroller controls the cathodes [1]. To activate a segment, apply a logical low (0) to its cathode, and use a logical high (1) to deactivate it.

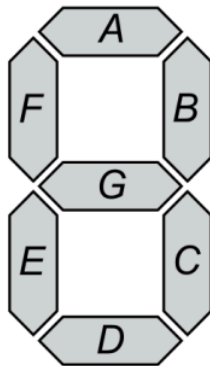


Figure 10: 7-Segment LED display [10].

The segments are controlled by the PIC16F84A microcontroller, which outputs binary values. Each hexadecimal digit (0-F) is converted to a 7-bit binary pattern, with each bit representing a segment (A-G). Table 1 shows the segment configuration for each hexadecimal character. Each row shows the configuration for each hexadecimal character, with the segments labelled as A through G. The bitwise calculation column illustrates how each bit in the 7-bit binary relates to a segment, the decimal column shows the binary pattern in decimal form, and the last column presents the hexadecimal value required for assembly code implementation.

To do this on the microcontroller, a loop cycles through hexadecimal values (0 to F), loading each into the working register and sends the binary pattern to the 7-segment display. Delays between changes ensure each character is displayed in sequence.

Table 1: Bitwise calculations for the 7-Segment LED display

Display	7-segment Binary							Bitwise Calculations	Decimal Value	Hexadecimal Value
	A	B	C	D	E	F	G			
0	0	0	0	0	0	0	1	$0+0+0+0+0+0+64$	64	0x40
1	1	0	0	1	1	1	1	$1+0+0+8+16+32+64$	121	0x79
2	0	0	1	0	0	1	0	$0+0+4+0+0+32+0$	36	0x24
3	0	0	0	0	1	1	0	$0+0+0+0+16+32+0$	48	0x30
4	1	0	0	1	1	0	0	$1+0+0+8+16+0+0$	25	0x19
5	0	1	0	0	1	0	0	$0+2+0+0+16+0+0$	18	0x12
6	0	1	0	0	0	0	0	$0+2+0+0+0+0+0$	2	0x02
7	0	0	0	1	1	1	1	$0+0+0+8+16+32+64$	120	0x78
8	0	0	0	0	0	0	0	$0+0+0+0+0+0+0$	0	0x00
9	0	0	0	0	1	0	0	$0+0+0+0+16+0+0$	16	0x10
A	0	0	0	1	0	0	0	$0+0+0+8+0+0+0$	8	0x08
b	1	1	0	0	0	0	0	$1+2+0+0+0+0+0$	3	0x03
C	0	1	1	0	0	0	1	$0+2+4+0+0+0+64$	70	0x46
d	1	0	0	0	0	1	0	$1+0+0+0+0+32+0$	33	0x21
E	0	1	1	0	0	0	0	$0+2+4+0+0+0+0$	6	0x06
F	0	1	1	1	0	0	0	$0+2+4+8+0+0+0$	14	0x0E

3 Programming Logic and Implementation

This section demonstrates how to use the PIC16F84A microcontroller to enhance the count.asm program's capability. Starting with the Proteus simulation, the various tasks focused on a different objective such as basic counting or dynamic audio signal generation.

3.1 Counting from 0 to F in Hexadecimal

This work entails changing the "Count.asm" software such that the seven-segment display cycles through hexadecimal numerals (0-F) in 0.2-second increments. This necessitates adding a delay between modifications and ensuring that the output matches the hexadecimal encoding for each number. The pseudocode provides the logic for counting and timing, while the assembly code turns that logic into actual program steps.

3.1.1 Pseudocode

```

Initialise the microcontroller and set up the ports for the seven-segment display
Load the delay subroutine for 0.2-second timing

Start main loop:
  For each number from 0 to F in hexadecimal:
    Convert the number to the seven-segment display bit-wise calculation
    Output the encoded value to the seven-segment display
    Call the delay subroutine to wait for 0.2 seconds

  Repeat loop indefinitely

```

3.1.2 Assembly Code

```
1. ;=====
2. ; DEFINITIONS
3. ;=====
4. #include p16f84a.inc           ; Include register definition file
5.
6. portA equ 0x05                ; Define Port A address
7. trisa equ 0x05                ; Define TRISA address in register bank 0
8. portB equ 0x06                ; Define Port B address
9. trisb equ 0x06                ; Define TRISB address in register bank 1
10.
11. status equ 0x03              ; Define Status register
12. RP0 equ 5                    ; Bit 5 of Status register for bank selection
13. RP1 equ 6                    ; Bit 6 of Status register for bank selection
14.
15. ;=====
16. ; VARIABLES
17. ;=====
18. count equ 0x0c               ; Counter for 0-255
19. nCount equ 0x0d              ; Pause routine inner loop counter
20. mCount equ 0x0e              ; Pause routine outer loop counter
21.
22. ;=====
23. ; RESET and INTERRUPT VECTORS
24. ;=====
25. RST code 0x0
26. goto Start                   ; Go to Start on reset
27.
28. ;=====
29. ; CODE SEGMENT
30. ;=====
31. PGM code
32. Start
33. bcf status,RP0               ; Clear bank selection bits in the status reg
34. bcf status,RP1               ; Clear bank selection bits in the status reg
35.
36. bsf status,RP0               ; Select bank 1
37. movlw 0x00                   ; Set W (working register) to 0x00
38. movwf trisb                  ; Set Port B as output
39. bcf status,RP0               ; Return to bank 0
40.
41. clrf portB                   ; Clear Port B
42. clrf count                   ; Initialise counter
43.
44. get_count
45. movf count,w                 ; Load count into W
46. call hex_chars               ; Call hex_chars to display current value
47. goto get_count               ; Repeat loop
48.
49. ;-----
50. ; Subroutine: pause
51. ; Purpose: Creates a delay using nested loops to achieve 0.2s delay
52. ;-----
53. pause
54. movlw 0xff                   ; Set W to 255 decimal
55. movwf mCount                 ; mCount = w: Outer loop counter
56.
57. loadN
58. movlw 0xff                   ; Set W to 255 decimal
59. movwf nCount                 ; nCount = w: Inner loop counter
60.
61. decN
62. decfsz nCount, f              ; Decrement nCount
63. goto decN                    ; Repeat if nCount not zero
64.
65. decfsz mCount, f              ; Decrement mCount
66. goto loadN                   ; Repeat if mCount not zero
67.
```

```

68.     return                                ; Exit delay subroutine
69.
70.
71. ;-----
72. ; Subroutine: hex_chars
73. ; Purpose: Displays hexadecimal digits on 7-segment display from 0-F
74. ;-----
75. hex_chars
76.     movlw    0x40                        ; Load hex code for '0' in W
77.     movwf    portB                      ; Output to 7-segment display
78.     call     pause                      ; 0.2s delay
79.
80.     movlw    0x79                        ; Load hex code for '1'
81.     movwf    portB                      ; Output to 7-segment display
82.     call     pause                      ; 0.2s delay
83.
84.     movlw    0x24                        ; Load hex code for '2'
85.     movwf    portB                      ; Output to 7-segment display
86.     call     pause                      ; 0.2s delay
87.
88.     movlw    0x30                        ; Load hex code for '3'
89.     movwf    portB                      ; Output to 7-segment display
90.     call     pause                      ; 0.2s delay
91.
92.     movlw    0x19                        ; Load hex code for '4'
93.     movwf    portB                      ; Output to 7-segment display
94.     call     pause                      ; 0.2s delay
95.
96.     movlw    0x12                        ; Load hex code for '5'
97.     movwf    portB                      ; Output to 7-segment display
98.     call     pause                      ; 0.2s delay
99.
100.    movlw    0x02                        ; Load hex code for '6'
101.    movwf    portB                      ; Output to 7-segment display
102.    call     pause                      ; 0.2s delay
103.
104.    movlw    0x78                        ; Load hex code for '7'
105.    movwf    portB                      ; Output to 7-segment display
106.    call     pause                      ; 0.2s delay
107.
108.    movlw    0x00                        ; Load hex code for '8'
109.    movwf    portB                      ; Output to 7-segment display
110.    call     pause                      ; 0.2s delay
111.
112.    movlw    0x10                        ; Load hex code for '9'
113.    movwf    portB                      ; Output to 7-segment display
114.    call     pause                      ; 0.2s delay
115.
116.    movlw    0x08                        ; Load hex code for 'A'
117.    movwf    portB                      ; Output to 7-segment display
118.    call     pause                      ; 0.2s delay
119.
120.    movlw    0x03                        ; Load hex code for 'B'
121.    movwf    portB                      ; Output to 7-segment display
122.    call     pause                      ; 0.2s delay
123.
124.    movlw    0x46                        ; Load hex code for 'C'
125.    movwf    portB                      ; Output to 7-segment display
126.    call     pause                      ; 0.2s delay
127.
128.    movlw    0x21                        ; Load hex code for 'D'
129.    movwf    portB                      ; Output to 7-segment display
130.    call     pause                      ; 0.2s delay
131.
132.    movlw    0x06                        ; Load hex code for 'E'
133.    movwf    portB                      ; Output to 7-segment display
134.    call     pause                      ; 0.2s delay
135.
136.    movlw    0x0E                        ; Load hex code for 'F'

```

```

137.    movwf    portB           ; Output to 7-segment display
138.    call     pause          ; 0.2s delay
139.
140.    return                    ; Exit hex_chars subroutine
141.
142.    end
143.

```

3.2 Displaying "L" and "r" Based on Button Presses

This job seeks to show the letter "L" on the 7-segment LED display when one button is pressed, and "r" when the second button is touched. It detects button pushes and displays the corresponding output on the LED display. The pseudocode defines the logic for responding to button pushes and creating the display encoding, whilst the assembly code provides the implementation.

3.2.1 Pseudocode

```

Set up the ports for the seven-segment display as output
Set up the ports for the buttons as input

Create constant variables for "L" and "r" based on the bit-wise calculation

Start main loop:
    Check the status of Button 1:
        If Button 1 is pressed:
            Output the encoding for "L" to the seven-segment display

    Check the status of Button 2:
        If Button 2 is pressed:
            Output the encoding for "r" to the seven-segment display

Repeat loop indefinitely

```

3.2.2 Assembly Code

```

1.  ;=====
2.  ; DEFINITIONS
3.  ;=====
4.
5.  #include p16f84a.inc           ; Include register definition file
6.
7.  portA    equ 0x05              ; Define Port A address
8.  trisa    equ 0x05              ; Define TRISA address in register bank 0
9.  portB    equ 0x06              ; Define Port B address
10. trisb    equ 0x06              ; Define TRISB address in register bank 1
11.
12. status   equ 0x03              ; Status register
13. RP0      equ 5                  ; Bit 5 of status register
14. RP1      equ 6                  ; Bit 6 of status register
15.
16. ;=====
17. ; VARIABLES
18. ;=====
19. count     equ 0x0c              ; Counter 0 to 255
20. nCount    equ 0x0d              ; Pause routine inner loop counter
21. mCount    equ 0x0e              ; Pause routine outer loop counter
22. ;=====
23. ; CONSTANTS for 7-segment display
24. ;=====
25.
26. L_Display equ 0x47              ; Hex for L for 7-segment LED
27. r_Display equ 0x2F              ; Hex for r for 7-segment LED
28.
29. ;=====

```

```

30. ; RESET and INTERRUPT VECTORS
31. ;=====
32.
33. ; Reset Vector
34. RST    code 0x0
35.     goto Start                ; Go to start on reset
36.
37. ;=====
38. ; CODE SEGMENT
39. ;=====
40.
41. PGM    code
42. Start
43.     bcf    status,RP0          ; Clear bank selection bits in the status reg
44.     bcf    status,RP1
45.
46.     bsf    status,RP0          ; Select bank 1
47.     movlw  0x00                ; Load w with 0x00
48.     movwf  trisb                ; Set portB as output
49.     movlw  0xFF                ; Load w with 0xFF
50.     movwf  trisa                ; Select portA as input
51.     bcf    status,RP0          ; Reselect bank 0
52.
53.     movlw  0xFF                ; Set Port B to logic 1 to turn off all segments
54.     movwf  portB                ; Initialise the 7-segment display to "off"
55.     clrf   count                ; Initialise counter to 0
56.
57. buttons
58.     ; Check if button 1 (BUT0) is pressed (RA0)
59.     btfss  portA,0              ; Skip next instruction if RA0 is high
60.     goto   display_L            ; Go to display_L if BUT0 is pressed (RA0 is low)
61.
62.     ; Check if button 2 (BUT1) is pressed (RA1)
63.     btfss  portA,1              ; Skip next instruction if RA1 is high
64.     goto   display_r            ; Go to display_r if BUT1 is pressed (RA1 is low)
65.
66.     goto   buttons              ; Otherwise, continue checking buttons
67.
68. display_L
69.     movlw  L_Display            ; Load the 7-segment code for "L"
70.     movwf  portB                ; Output to Port B to display "L"
71.     call   pause                ; 0.2s delay
72.     goto   buttons              ; Return to checking button presses
73.
74. display_r
75.     movlw  r_Display            ; Load the 7-segment code for "r"
76.     movwf  portB                ; Output to Port B to display "r"
77.     call   pause                ; 0.2s delay
78.     goto   buttons              ; Return to checking button presses
79.
80.
81. ;=====
82. ; Pause subroutine to add delay
83. ;=====
84. pause
85.     movlw  0xFF                ; Set w = 255 (outer loop counter)
86.     movwf  mCount              ; mCount = w: outer loop counter
87. loadN
88.     movlw  0xFF                ; Set w = 255 (inner loop counter)
89.     movwf  nCount              ; nCount = w: inner loop counter
90. decN
91.     decfsz nCount, f            ; Decrement inner loop counter
92.     goto   decN                ; Repeat if nCount != 0
93.     decfsz mCount, f           ; Decrement outer loop counter
94.     goto   loadN               ; Repeat if mCount != 0
95.
96.     return                    ; Exit subroutine
97.
98. end

```

3.3 Toggling All LED Segments On/Off

This task is focused on programming a latching mechanism in which one button lights all LED segments on the display while the other turns them off. The latching effect maintains the state until the opposite button is pressed. The pseudocode provides the logic for maintaining and toggling states, while the assembly code converts it to machine-level instructions.

3.3.1 Pseudocode

```

Initialise the microcontroller:
    Set up the ports for the seven-segment display as output
    Set up the ports for the buttons as input

Create a state variable:
    Set initial state to "OFF"

Start main loop:
    Check the status of Button 1:
        If Button 1 is pressed:
            Update state to "ON"

    Check the status of Button 2:
        If Button 2 is pressed:
            Update state to "OFF"

    Based on the current state:
        If state is "ON":
            Turn on all LED segments
        If state is "OFF":
            Turn off all LED segments

    Repeat loop indefinitely

```

3.3.2 Assembly Code

```

1. ;=====
2. ; DEFINITIONS
3. ;=====
4.
5. #include p16f84a.inc          ; Include register definition file
6.
7. portA    equ 0x05             ; Define Port A address
8. trisa    equ 0x05             ; Define TRISA address in register bank 0
9. portB    equ 0x06             ; Define Port B address
10. trisb    equ 0x06             ; Define TRISB address in register bank 1
11.
12. status   equ 0x03             ; Status register
13. RP0      equ 5                ; Bit 5 of status register
14. RP1      equ 6                ; Bit 6 of status register
15.
16. ;=====
17. ; VARIABLES
18. ;=====
19. count     equ 0x0c             ; Counter 0 to 255
20. nCount    equ 0x0d             ; Pause routine inner loop counter
21. mCount    equ 0x0e             ; Pause routine outer loop counter
22. ledState  equ 0x0f             ; Variable to store LED state (1=on, 0=off)
23.
24. ;=====

```

```

25. ; CONSTANTS for 7-segment display
26. ;=====
27.
28. ALL_ON    equ 0x00          ; 7-segment code for all segments ON
29. ALL_OFF   equ 0xFF          ; 7-segment code for all segments OFF
30.
31. ;=====
32. ; RESET and INTERRUPT VECTORS
33. ;=====
34.
35. ; Reset Vector
36. RST code 0x0
37.     goto Start
38.
39. ;=====
40. ; CODE SEGMENT
41. ;=====
42.
43. PGM code
44. Start
45.     bcf     status,RP0          ; clear bank selection bits in the status reg
46.     bcf     status,RP1
47.
48.
49.     bsf     status,RP0          ; select bank 1
50.     movlw   0x00                ; load w with 0x00
51.     movwf   trisb                ; set portB as output
52.     movlw   0xFF                ; load w with 0xFF
53.     movwf   trisa                ; set portA as input
54.     bcf     status,RP0          ; reselect bank 0
55.
56.
57.
58.     movlw   0xFF                ; Set Port B
59.     movwf   portB                ; Set the 7-segment display to "off"
60.     clrf    count                ; Initialise counter
61.     clrf    ledState             ; Set LED state to OFF (0)
62.
63.
64. ; Main loop to check button presses
65. buttons
66.     ; Check if button 1 (BUT0) is pressed (RA0)
67.     btfss   portA,0              ; If RA0 is high, BUT0 is not pressed
68.     goto    toggle_led_on        ; Go to toggle_led_on if BUT0 is pressed
69.
70.     ; Check if button 2 (BUT1) is pressed (RA1)
71.     btfss   portA,1              ; If RA1 is high, BUT1 is not pressed
72.     goto    toggle_led_off       ; Go to toggle_led_off if BUT1 is pressed
73.
74.     goto    buttons              ; Otherwise, continue checking buttons
75.
76. toggle_led_on
77.     ; If LED state is OFF (0), turn all LED segments ON
78.     btfsc   ledState, 0          ; Check if LED is already on (if ledState is 1)
79.     goto    buttons              ; If already on, do nothing and return to buttons check
80.
81.     movlw   ALL_ON               ; Load code for all LED segments ON
82.     movwf   portB                ; Output to Port B to turn ON all segments
83.     bsf     ledState, 0          ; Set ledState to 1 (LED ON state)
84.     call    pause                ; Call pause to debounce the button
85.     goto    buttons              ; Return to checking button presses
86.
87. toggle_led_off
88.     ; If LED state is ON (1), turn all LED segments OFF
89.     btfss   ledState, 0          ; Check if LED is already off (if ledState is 0)
90.     goto    buttons              ; If already off, do nothing and return to buttons check
91.
92.     movlw   ALL_OFF              ; Load code for all LED segments OFF
93.     movwf   portB                ; Output to Port B to turn OFF all segments
94.     bcf     ledState, 0          ; Set ledState to 0 (LED OFF state)

```

```

95.      call    pause           ; Call pause to debounce the button
96.      goto    buttons        ; Return to checking button presses
97.
98.
99. ;-----
100. ; Pause subroutine to add delay (debouncing)
101. ;-----
102. pause
103.      movlw   0xff            ; Set w = 255 (outer loop counter)
104.      movwf   mCount         ; mCount = w: outer loop counter
105.      loadN
106.      movlw   0xff            ; Set w = 255 (inner loop counter)
107.      movwf   nCount         ; nCount = w: inner loop counter
108.      decN
109.      decfsz  nCount, f       ; Decrement inner loop counter
110.      goto    decN           ; Repeat if nCount != 0
111.      decfsz  mCount, f       ; Decrement outer loop counter
112.      goto    loadN          ; Repeat if mCount != 0
113.
114.      return                  ; Exit subroutine
115.
116.      end
117.

```

3.4 Incrementing and Decrementing a Displayed Number

This task uses previous techniques to create a counting mechanism controlled by two buttons. Starting at "5," one button raises the count to "9," while the other lowers it to "0." The program manages state changes and sets restrictions to prevent overflow and underflow. The pseudocode provides the logic for incrementing, decrementing, and preserving boundaries, whereas the assembly code provides the implementation steps.

3.4.1 Pseudocode

```

Set up the ports for the seven-segment display as output
Set up the ports for the buttons as input

Initialise variables:
    Set current_number to 5
    Define maximum_number as 9
    Define minimum_number as 0

Define a lookup table for seven-segment encodings (0 to 9)

Start main loop:
    Check the status of the "Increment" button:
        If the button is pressed:
            If current_number is less than maximum_number:
                Increment current_number by 1

    Check the status of the "Decrement" button:
        If the button is pressed:
            If current_number is greater than minimum_number:
                Decrement current_number by 1

    Use the lookup table to get the seven-segment encoding for current_number
    Output the encoding to the seven-segment display

    Repeat loop indefinitely

```

3.4.2 Assembly Code

```

1. ;=====

```

```

2. ; DEFINITIONS
3. ;=====
4.
5.#include p16f84a.inc                ; Include register definition file
6.
7. portA    equ 0x05                  ; Define Port A address
8. trisA    equ 0x05                  ; Define TRISA address in register bank 0
9. portB    equ 0x06                  ; Define Port B address
10. trisB    equ 0x06                  ; Define TRISB address in register bank 1
11.
12. status   equ 0x03                  ; Status register
13. RP0      equ 5                     ; Bit 5 of status register
14. RP1      equ 6                     ; Bit 6 of status register
15.
16. ;=====
17. ; VARIABLES
18. ;=====
19. count     equ 0x0c                  ; Counter 0 to 255
20. nCount    equ 0x0d                  ; Pause routine inner loop counter
21. mCount    equ 0x0e                  ; Pause routine outer loop counter
22.
23. ;=====
24. ; RESET VECTOR
25. RST       code 0x0
26.         goto Start
27. ;=====
28. ; MAIN PROGRAM
29. ;=====
30. PGM       code
31. Start
32.         bcf      status,RP0          ; clear bank selection bits in the status reg
33.
34.         bsf      status, RP0          ; Select bank 1
35.         movlw    0x00                  ; Set Port B as output
36.         movwf    trisB
37.
38.         movlw    0xFF                  ; Set Port A as input
39.         movwf    trisA
40.         bcf      status, RP0          ; Select bank 0
41.
42.         movlw    0x05                  ; Initialise count to 5
43.         movwf    count
44.
45.         call DisplayCount
46.
47. buttons
48.         ; Read the state of BUT0 (RA0) and BUT1 (RA1)
49.         btfss    portA, 0             ; Check if BUT0 is pressed (RA0 = 0)
50.         goto     BUT0_Pressed         ; Go to BUT0_Pressed if BUT0 is pressed
51.
52.         btfss    portA, 1             ; Check if BUT1 is pressed (RA1 = 0)
53.         goto     BUT1_Pressed         ; Go to BUT1_Pressed if BUT0 is pressed
54.
55.         goto     buttons              ; Otherwise, keep checking
56.
57. ; Decrement count (for BUT0)
58. BUT0_Pressed
59.         ; Check if the button was pressed (RA0 = 0)
60.         btfsc    portA, 0             ; Check if RA0 is low (pressed)
61.         goto     Decrement
62.
63.         goto     buttons
64.
65. Decrement
66.         decfsz   count, f              ; Decrement the count
67.         movf     count, w              ; Load count into W
68.         call     DisplayCount          ; Update 7-segment display with new value
69.         call     pause                 ; Delay

```

```

70.    goto    buttons        ; Return to buttons check
71.
72. ; Increment count (for BUT1)
73. BUT1_Pressed
74.    ; Check if the button was pressed (RA1 = 0)
75.    btfsc   portA, 1        ; Check if RA1 is low (pressed)
76.    goto    Increment
77.
78.    goto    buttons
79.
80. Increment
81.    incf     count, f        ; Increment the count
82.    movlw    0x09            ; Limit to 9 (hexadecimal 0x09)
83.    subwf    count, w        ; Check if count > 9
84.    btfsc    status, Z       ; If count > 9, reset to 0
85.    clrf     count
86.    movf     count, w        ; Load count into W
87.    call     DisplayCount    ; Update 7-segment display with new value
88.    call     pause           ; Delay
89.    goto    buttons        ; Return to buttons check
90.
91.
92.
93. ; Display count on 7-segment display
94. DisplayCount
95.    movf     count, w        ; Get count value
96.    call     lookup_table    ; Lookup 7-segment code
97.    movwf    portB           ; Output to Port B to display on 7-segment
98.    return
99.
100.
101. ; -----
102. ; Pause subroutine to add delay (debouncing)
103. ; -----
104. pause
105.    movlw    0xff            ; Set w = 255 (outer loop counter)
106.    movwf    mCount          ; mCount = w: outer loop counter
107. loadN
108.    movlw    0xff            ; Set w = 255 (inner loop counter)
109.    movwf    nCount          ; nCount = w: inner loop counter
110. decN
111.    decfsz   nCount, f        ; Decrement inner loop counter
112.    goto     decN            ; Repeat if nCount != 0
113.    decfsz   mCount, f        ; Decrement outer loop counter
114.    goto     loadN           ; Repeat if mCount != 0
115.
116.    return                    ; Exit subroutine
117.
118.
119.
120.
121. ; =====
122. ; LOOKUP TABLE FOR 7-SEGMENT DISPLAY (digits 0-9)
123. ; =====
124. lookup_table
125.    addwf    PCL, f
126.    retlw    0x40            ; '0' on LED display
127.    retlw    0x79            ; '1' on LED display
128.    retlw    0x24            ; '2' on LED display
129.    retlw    0x30            ; '3' on LED display
130.    retlw    0x19            ; '4' on LED display
131.    retlw    0x12            ; '5' on LED display
132.    retlw    0x02            ; '6' on LED display
133.    retlw    0x78            ; '7' on LED display
134.    retlw    0x00            ; '8' on LED display
135.    retlw    0x10            ; '9' on LED display
136.
137.    end
138.

```

3.5 Producing a Tone with the Piezo Sounder

This task aims to make the piezo sounder generate a tone when a button is pressed. The circuit detects the button's state and generates a square wave with a set frequency to power the sounder. The pseudocode describes the logic behind button detection and tone creation.

3.5.1 Pseudocode

```
Configure the port connected to the piezo sounder as output
Configure the port connected to the button as input

Create constants:
  Set the tone frequency (e.g., 1 kHz)
  Calculate the delay required for the tone period (1/frequency divided by 2)

Start main loop:
  Check the status of the button:
    If the button is pressed:
      Generate the tone:
        Toggle the piezo output pin (high to low and low to high)
        Wait for the calculated delay
    Else:
      Set the piezo output pin to low (silence)

  Repeat loop indefinitely
```

3.6 Generating a Dynamic PWM Signal

This task was to create a pulse train with a dynamically changing pulse width to power the piezo sounder. This involves using a basic pulse-width modulation (PWM) approach with a continuously changing duty cycle to simulate a synthesiser-like waveform. The pseudocode describes the logic used to generate the fluctuating PWM signal.

3.6.1 Pseudocode

```
Configure the port connected to the speaker as output
Set a fixed PWM frequency (e.g., 1 kHz) for the pulse train

Create constants:
  Set initial PWM period (e.g., 1 ms for a 1 kHz frequency)
  Set initial duty cycle (e.g., 50%)
  Define PWM frequency modulation parameters (e.g., modulation range and speed)

Start main loop:
  Generate a pulse train:
    For each PWM period:
      Toggle the speaker output pin based on the current duty cycle
      Wait for the high time (duty cycle * period)
      Wait for the low time ((1 - duty cycle) * period)

  Modulate the duty cycle:
    Gradually change the duty cycle value (increase or decrease)
    Use a sine wave, sawtooth, or other modulation technique to create a changing effect
    Update the duty cycle value for the next PWM period

  Repeat loop indefinitely
```

These activities demonstrate the PIC16F84A microcontroller's versatility and ability to manage various hardware aspects. Together, they demonstrate the microcontroller's ability to manage complex operations through efficient programming, laying the framework for future study and development in embedded system design.

4 References

- [1] PIC16F84A Data Sheet. Microchip Technology Inc, 2001. Available:
<https://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>
- [2] S. Mugo, "Introduction to Quartz Crystal Oscillators," Eepower.com, Dec. 16, 2022.
<https://eepower.com/technical-articles/introduction-to-quartz-crystal-oscillators/#> (accessed Jan. 12, 2025).
- [3] Electronics Tutorials, "7-segment Display and Driving a 7-segment Display," Basic Electronics Tutorials, Feb. 16, 2018. <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html>
- [4] "FriendlyWire - Discover the joy of understanding electronics," Friendlywire.com, 2019.
<https://www.friendlywire.com/tutorials/pushbutton/> (accessed Jan. 12, 2025).
- [5] "7.3 connecting buttons to PIC microcontroller pins," Blogspot.com, 2025.
<https://newengineeringpractice.blogspot.com/2012/01/73-connecting-buttons-to-pic.html> (accessed Jan. 12, 2025).
- [6] Pi Hut, "The Pi Hut," The Pi Hut, 2015. <https://thepihut.com/blogs/raspberry-pi-tutorials/how-to-use-voltage-regulators-in-a-circuit?srsId=AfmBOooUTtlf1uVI0Blk8oZtlj8uvtlhS4SDQGKbLo8FGh38ckzOWyJS> (accessed Jan. 12, 2025).
- [7] JIMBLOM, "Transistors - learn.sparkfun.com," Sparkfun.com, 2018.
<https://learn.sparkfun.com/tutorials/transistors/all>
- [8] BC107-BC107B Data Sheet. ST, 2006. Available:
<https://www.st.com/resource/en/datasheet/cd00003228.pdf>
- [9] P. Kendall, "How Do Piezo Buzzers Work | RS," Rs-online.com, Oct. 30, 2023. https://uk.rs-online.com/web/content/discovery/ideas-and-advice/how-do-piezo-buzzers-work?srsId=AfmBOooh8CFx4uk6OTMENNpBdj0bBkoY88X_D9jrbUeqjSsi8qSqDNjy (accessed Jan. 12, 2025).
- [10] Admin, "LED 7 segment display driver circuit. Basic seven segment display system ,7 segment display driver using 7446," Electronic Circuits and Diagrams-Electronic Projects and Design, Mar. 27, 2012. <https://www.circuitstoday.com/7-segment-display-driver>

Micros Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

72/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

GRADING FORM: GJ_2019

EMMA MOSELEY

358

ENGAGEMENT WITH LITERATURE



Reasonable number of sources cited

70

KNOWLEDGE AND UNDERSTANDING



A very good explanation of the component operation - good use of diagrams

75

COGNITIVE AND INTELLECTUAL



Fairly straightforward implementation of the coding, so the marks for this section aren't as high

58

(as this was for 'flair' in solving the problems).
Good to see the array implemented in the last programming task though.

PRACTICAL APPLICATION



Circuit implemented ok, Pseudocode reasonable for last two tasks even if the code wasn't attempted.

70

TRANSFERABLE SKILLS



Very well laid out and presented

85