

ACDA4004_ Rodina_Macleod_Micros_Assign ment

by Rodina MacLeod

Submission date: 14-Jan-2025 01:40PM (UTC+0000)

Submission ID: 248415253

File name:

51414_Rodina_MacLeod_ACDA4004__Rodina_Macleod_Micros_Assignment_621295_1647080517.docx (2.28M)

Word count: 2581

Character count: 13033

Programme:	BEng Embedded Electronics Systems
Module:	Micros Peripherals and Interfaces
Module Code:	ACDA4004
Assignment Title:	Using and Programming the 7-segment display Prototype Circuit
Student Name:	Rodina Macleod
Student ID:	2318127

Table of Contents

Section 1	3
Abstract.....	3
Circuit Simulation	3
Components used in Circuit	5
Changes between Assignment Board & Circuit	8
Section 2	9
Bitwise Calculations	9
Section 3	12
Task 1	12
Flowchart: Task 1	12
Annotated Assembly: Task 1	13
Task 2	15
Flowchart: Task 2	16
Annotated Assembly: Task 2	16
Task 3	18
Flowchart: Task 3	19
Annotated Assembly: Task 3	19
Task 4	21
Flowchart: Task 4	21
Annotated Assembly: Task 4	22
Task 5	24
Flowchart: Task 5	24
Annotated Assembly: Task 5	25
Conclusion	27
References	27

Section 1

Abstract

This report demonstrates the use and programming of a 7-segment display prototype circuit simulated in Proteus with an understanding of microcontroller features, interfacing techniques and low-level programming skills. The circuit is based upon “count.asm”, which was provided during the course of this module and the PIC Microcontroller used throughout this report is the PIC16F84A.

Circuit Simulation

The initial task was to review the PIC16F84A Data Sheet for the 8-bit PIC (Programmable Integrated Circuit) Microcontroller [1] to understand both the architecture and the features of the PIC16F84A (Fig. 1.0) before simulating the circuit.

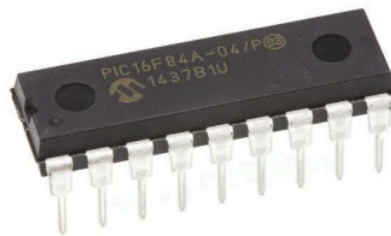


Fig. 1.0 Microchip PIC16F84A 8-bit PIC Microcontroller [2].

As relayed in the Data Sheet, the PIC16F84A has 35 instructions which classifies it as a RISC (Reduced Instruction Set Computer) processor [3]; it has 1024 words of program memory, 68 bytes of Data RAM (Random Access Memory) and 64 bytes of Data EEPROM (Electrically Erasable Programmable Read-Only Memory) [1]. The data memory of the PIC16F84A is sectioned into parts; the General Purpose Registers (GPRs) and the Special Function Registers (SFRs). SFRs are considered to be the gateway to interactions between peripherals and the CPU [4].

In order to complete the task required in Section 1, the circuit diagram for the 7-segment display prototype HND/ BENG PIC16F84 Assignment Board (Fig 1.1) had to be replicated and studied in Proteus (Fig 1.2). The program, “Count”, presented characters 0-9 incrementally on the 7-segment display.

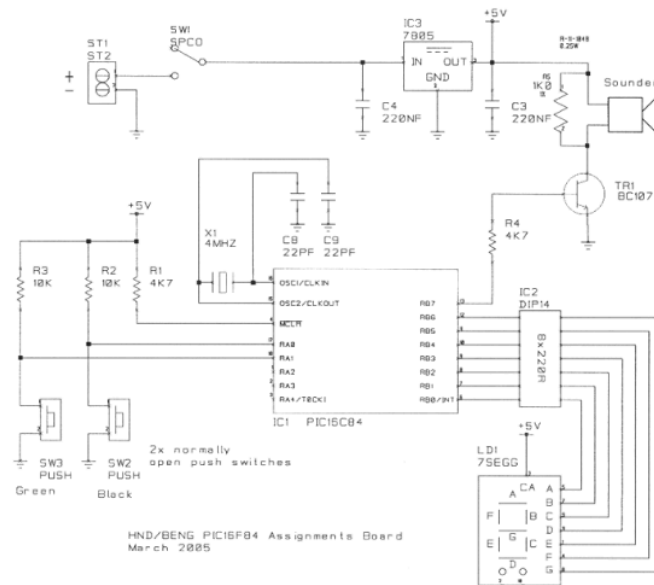


Fig 1.1. Circuit diagram for 7-segment display prototype HND/ BENG PIC16F84 Assignment Board

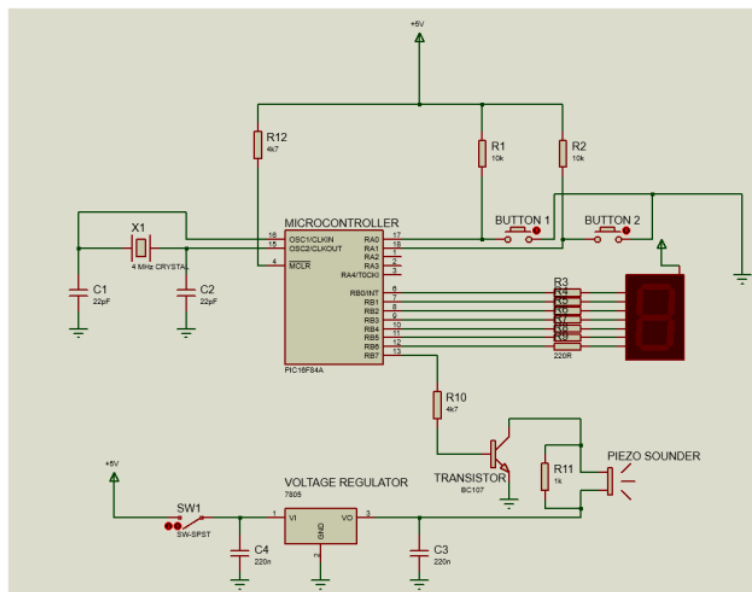
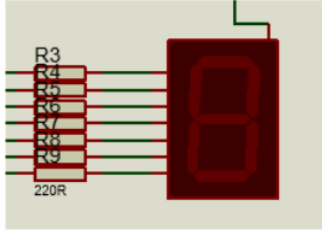
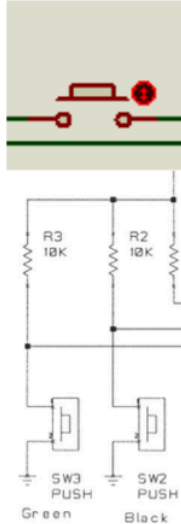
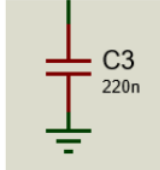
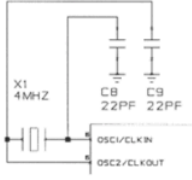
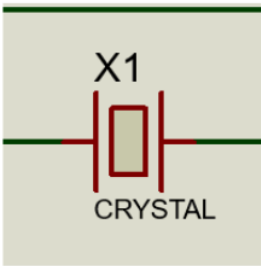
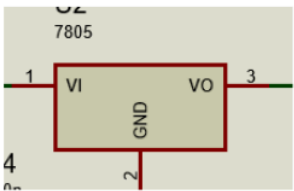
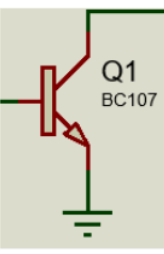
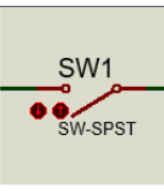
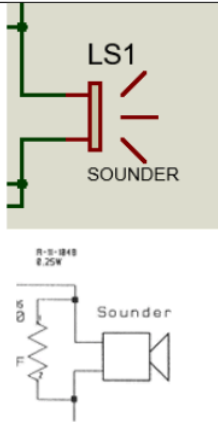
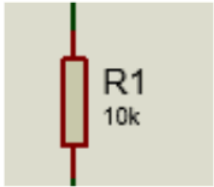
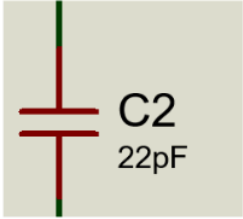
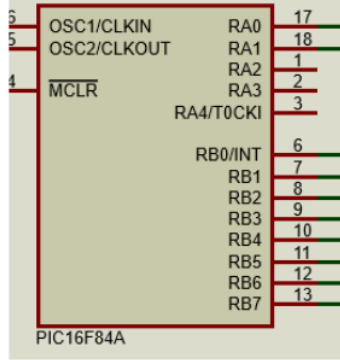


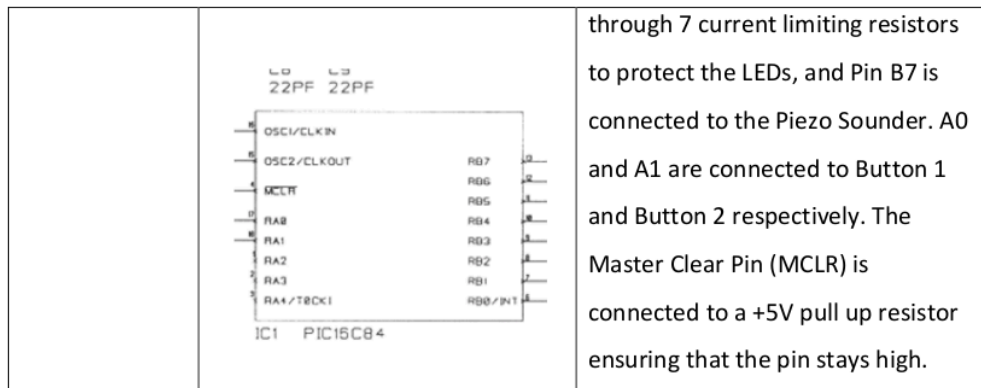
Fig 1.2. Circuit diagram of HND/ BENG PIC16F84 Assignment Board replicated in Proteus

Components used in Circuit

Name of Component	Proteus Component	Function/ Purpose within Circuit
7-Segment Display Common Anode		This 7-Segment display uses the common anode configuration. It consists of 7 LEDs. Binary/ Bitwise calculations were made to show the required characters on the 7-Segment Display, which is protected by 7 220Ω parallel resistors [5].
Button – SPST Push Button		As displayed in the Assignment Board (Fig. 1.2.) there are two push buttons in the circuit to allow user input. In this circuit, Button 1 inputs to pin RA0 and Button 2 inputs to pin RA1. The push buttons are connected to 2 pull up resistors (+5V) respectively. Depending on the requirements, the push buttons can trigger the 7-segment display to present specific Hex characters or sound the Piezo buzzer.
Capacitor – 220N Polyester Layer Capacitor		The 220N Polyester Layer Capacitors are used in line with the 7805 Voltage Regulator which stabilises it and provides a temporary current.
Crystal Oscillator – Quartz Crystal		A 4MHz quartz crystal oscillator was used in the circuit because microcontrollers depend on oscillators for precise timing in order to execute instructions [6]. The

		internal clock is driven by the stable frequency of the crystal oscillator [7]. The crystal oscillator is connected to pins OSC/CLKIN (pin 16) and OSC/CLKOUT (pin 15) on the microcontroller. There are 2 capacitors connected to the quartz oscillator to help its resonant frequency.
7805 (5V) Fixed Positive Power Supply Regulator		A 7805 Voltage Regulator with 3 terminals (I/O and Ground) is used within this circuit to maintain a constant voltage (+5V) despite any changes in load or input voltages. They are also beneficial in protecting the circuit from short circuiting or overheating [7, p. 35]
Transistor – Silicon NPN Low Power Bipolar Transistor		The NPN (Negative-Positive-Negative) Transistor used in this circuit acts as an amplifier to the signal and is controlled by the current [8]. The Piezo sounder requires a higher current, especially at louder volumes.
Switch – Interactive SPST Switch		The switch is included in this circuit in order to turn off power to the board when open and inversely, allowing the current to flow if closed.

Sounder – Piezo Sounder Model, Digital		A Piezo buzzer is a simple component, that can be procured in a variety of sizes and frequencies [9], used within a circuit to produce sound. In this circuit it is connected to Port B, Pin B7, protected by a 1K resistor in parallel. The buzzer within this circuit will be programmed in an ON/ OFF cycle.
Resistor – 10K		The 10K resistors act as “pull up resistors” in this circuit. It is connected between the power and the push buttons, therefore when the button is not pressed, the resistor pulls the pin to a logic 1 state.
Capacitor – CAP CERM 22pF		The ceramic capacitor is used in this circuit due to the presence of the crystal oscillator. 22pF is a typical value for capacitors used with crystal oscillators [10].
PIC16F84A Microcontroller		This is the PIC16F84A microcontroller used within the assignment. It is the fulcrum of the circuit – controlling and processing the peripherals within the circuit. It has 18 pins; 13 of these pins are configurable by the user to be I/O (Input/ Output) which are Port A and Port B. Port B (B0 – B6) is connected to the 7-segment display



Changes between Assignment Board & Circuit

One component to note was the Screw Terminal (Fig 1.3.) connected to the switch in the assignment board, to connect a power supply to the circuit. As this was not required in the Proteus simulated circuit, a +5V DC was connected instead. When the switch is closed the current does not pass through and when the switch is open, the current flows.



Fig 1.3. Screw Terminal in circuit and image of Screw Terminal

Section 2

Bitwise Calculations

In order to proceed with this section of the assignment, an initial understanding of the binary /bitwise calculations were drawn out by hand, as seen in Fig. 2.0, Fig. 2.2.

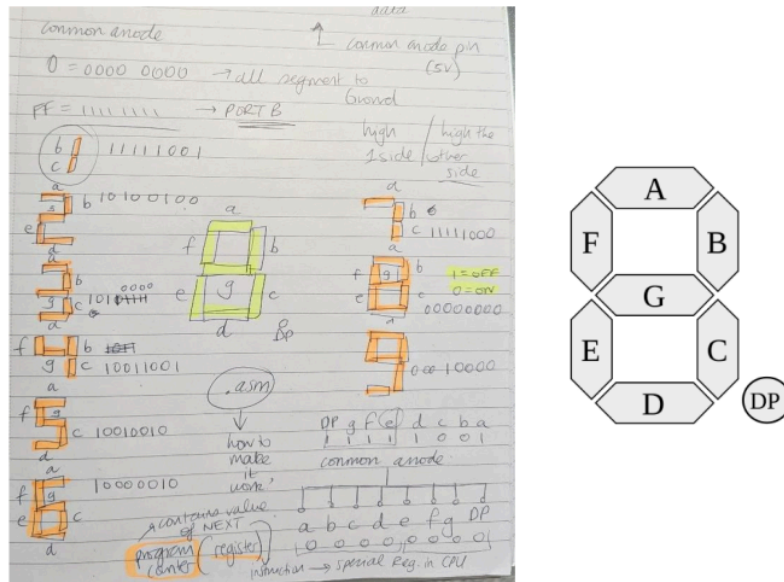


Fig. 2.0 Binary/ Bitwise calculations of the 7-Segment Display Common Anode

Each segment of the 7-Segment display represents an LED that can light up and display a character, depending on which pins were coded to switch on. This 7-Segment Display demonstrates a common anode configuration (Fig 2.1.) – the positive supply voltage is connected to the LEDs, which controls their state [11]. The anode configuration means that a LOW signal (logic 0) causes the segments to switch ON and a HIGH signal (logic 1) will cause the segment to switch OFF.

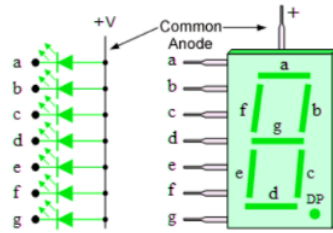


Fig 2.1. Common anode configuration of 7-Segment Display [11]

Further calculations were completed to show the hexadecimal characters 0 – F on the 7-Segment Display before implementation in the program (Fig. 2.1). Hex values were calculated by inputting the binary sequence into the Programmer setting of the Calculator and viewing the Hex output. The Hex values were required for the assembly code.

The characters 0 – F were calculated (Table 2.) using the common anode configuration of:

- 1 = **OFF**
- 0 = **ON**

Character	DP	g	f	e	d	c	b	a	Hex
0	1	1	0	0	0	0	0	0	C0
1	1	1	1	1	1	0	0	1	F9
2	1	0	1	0	0	1	0	0	A4
3	1	0	1	1	0	0	0	0	B0
4	1	0	0	1	1	0	0	1	99
5	1	0	0	1	0	0	1	0	92
6	1	0	0	0	0	0	1	0	82
7	1	1	1	1	1	0	0	0	F8
8	1	0	0	0	0	0	0	0	80
9	1	0	0	1	0	0	0	0	90
A	1	0	0	0	1	0	0	0	88
b	1	0	0	0	0	0	1	1	83
C	1	1	0	0	0	1	1	0	C6
d	1	0	1	0	0	0	0	1	A1
E	1	0	0	0	0	1	1	0	86
F	1	0	0	0	1	1	1	0	8E

Table 2. Table displaying the binary representation and Hex value for each character 0 – F.

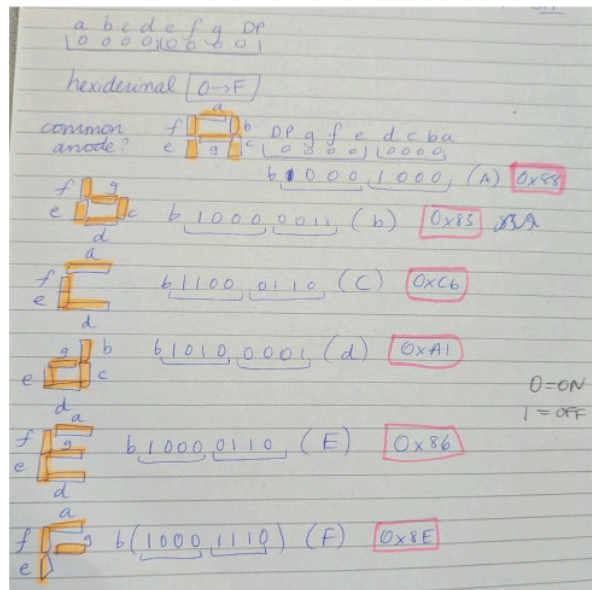


Fig. 2.2. Binary/Bitwise calculations for the Hex characters A - F

Section 3

Task 1

The first modification task to the “count.asm” circuit provided in the module was to ensure that:

- the 7-segment display counts from 0 – F in hexadecimal numbers
- in 0.2 second increments.

As seen in Table. 2, the hex values within the common anode configuration of the 7-Segment Display ranged from **0xC0** to **0x8E**, and these were added to the subroutine in the assembly code. When the program reaches 0x8E, it loops back to the beginning on repeat.

A substantial amount of time was spent reviewing the processes regarding the selection of Port B and how to define RP0 and RP1 in the Status Register. Although it was recommended to use RETLW to create a list within the code, a *hex_display* subroutine was created instead with a repeated structure from hexadecimal 0 – F, adding 1 to count each time (Fig. 1).

```
hex_display
    movlw 0xC0      ; Load w with 0xC0 - segment 0
    movwf portB     ; move W to port B to update the display
    addlw 1          ; add +1 to W
```

Fig. 1. Beginning of subroutine to count from 0 – F in increments of 0.2 seconds.

Flowchart: Task 1

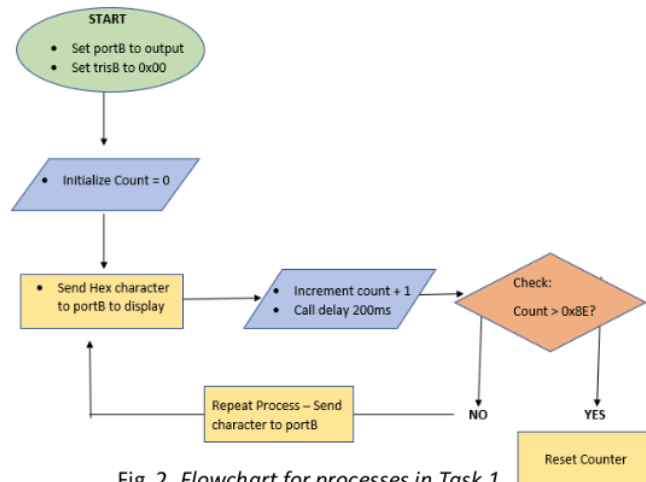


Fig. 2. Flowchart for processes in Task 1

Annotated Assembly: Task 1

See Fig. 3,4,5,6,7. in reference to the annotated modified Task 1.

```
main.asm
1 ;=====
2 ; Main.asm file generated by New Project wizard
3 ;
4 ; Created:   Wed Oct 30 2024
5 ; Author:   Rodina Macleod
6 ; Processor: PIC16F84A
7 ; Compiler: MPASM (Proteus)
8 ; Description: Modify "Count.asm" so that the seven segment display
9 ;              counts from 0 to F in hexadecimal numbers in 0.2
10 ;              second increments.
11 ;=====
12 ;
13 ;=====
14 ; DEFINITIONS
15 ;=====
16
17 #include p16f84a.inc          ; Include register definition file
18
19
20 portB    equ 0x06             ;port B address
21 trisB    equ 0x06             ;trisB address in register bank 1
22
23 status   equ 0x03             ;status register
24 RP0      equ 5                ;bit 5 of status register
25 RP1      equ 6                ;bit 6 of status register
26
27 ;=====
28 ; VARIABLES
29 ;=====
30
31 count     equ 0x0c             ; counter 0 to 255
32 nCount    equ 0x0d             ; pause routine inner loop counter
33 mCount    equ 0x0e             ; pause routine outer loop counter
34
```

Fig. 3. Defining addresses for port B to begin count program

```
35 ;=====
36 ; RESET and INTERRUPT VECTORS
37 ;=====
38
39 ; Reset Vector
40 RST      code 0x00
41          goto Start           ; initialisation
42
43 ;=====
44 ; CODE SEGMENT
45 ;=====
46
47 PGM      code
48 Start
49          bcf    status,RP0     ; clear bank selection bits in the status reg
50          bcf    status,RP1     ; clear bank selection bits in the status reg
51
52          bsf    status,RP0     ; select bank 1
53
54          movlw  0x00           ; Load w with 0x00
55          movwf  trisB          ; load it into trisb to make portb all outputs
56
57          bcf    status,RP0     ; reselect bank 0
58
59          clrf   portB          ; clear all lines low
60          clrf   count          ; initialize counter to 0
61
62          hex_display
63
```

Fig. 4. Selecting pins for input and output

```

61
62 hex_display
63
64     movlw 0xC0      ; load w with 0xC0 - segment 0
65     movwf portB     ; move W to port B to update the display
66     addlw 1         ; add +1 to W
67
68     call pause      ; delay 200ms by subroutine
69
70     movlw 0xF9      ; load w with 0xF9 - segment 1
71     movwf portB     ; move W to port B to update the display
72     addlw 1         ; add +1 to W
73
74     call pause      ; delay 200ms by subroutine
75
76     movlw 0xA4      ; load w with 0xA4 - segment 2
77     movwf portB     ; move W to port B to update the display
78     addlw 1         ; add +1 to W
79
80     call pause      ; delay 200ms by subroutine
81
82     movlw 0xB0      ; load w with 0xB0 - segment 3
83     movwf portB     ; move W to port B to update the display
84     addlw 1         ; add +1 to W
85
86     call pause      ; delay 200ms by subroutine
87
88     movlw 0x99      ; load w with 0x99 - segment 4
89     movwf portB     ; move W to port B to update the display
90     addlw 1         ; add +1 to W
91
92     call pause      ; delay 200ms by subroutine
93

```

Fig. 5. Beginning of hex_display subroutine to count in 200ms increments

```

139
140     call pause      ; delay 200ms by subroutine
141
142     movlw 0xA1      ; load w with 0xA1 - segment d
143     movwf portB     ; move W to port B to update the display
144     addlw 1         ; add +1 to W
145
146     call pause      ; delay 200ms by subroutine
147
148     movlw 0x86      ; load w with 0x86 - segment E
149     movwf portB     ; move W to port B to update the display
150     addlw 1         ; add +1 to W
151
152     call pause      ; delay 200ms by subroutine
153
154     movlw 0x8E      ; load w with 0x8E - segment F
155     movwf portB     ; move W to port B to update the display
156     addlw 1         ; add +1 to W
157
158     call pause      ; delay 200ms by subroutine
159
160
161     movwf portB     ; move W to port B to update the display
162     addlw 1         ; add +1 to W
163     movwf count     ; save W in count
164
165     call pause      ; delay 200ms by subroutine
166
167
168
169     goto hex_display ; repeat forever
170

```

Fig. 6. End of hex_display subroutine to count in 200ms increments showing loop

```

166
167
168
169     goto hex_display      ; repeat forever
170
171
172 ;-----
173
174 ;
175 ;description: two loops using "mCount" and "nCount" register files
176 ;              "mCount" is the outer loop counter (255 down to zero)
177 ;              "nCount" is the inner loop counter (255 down to zero)
178 ;
179 ;-----
180 pause
181     movlw    0xff          ; set w = 255 decimal
182     movwf    mCount        ; mCount = w: outer loop counter
183 loadN
184     movlw    0xff          ; set w = 255 decimal
185     movwf    nCount        ; nCount = w: inner loop counter
186 decN
187     decfsz   nCount, f      ; nCount--
188     goto     decN          ; if nCount != 0 then repeat nCount--
189
190     decfsz   mCount, f      ; else decrement Count
191     goto     loadN         ; if mCount != 0 then
192                          ; reload nCount to 255 and decrement
193     return                          ; else exit subroutine
194 ;=====
195     END
196

```

Fig. 7. End of Task 1 Count program – with pause subroutine.

Task 2

The second task required implementing Port A, tris A and Port B, and the task was to write a program that:

- When pressing one of the two buttons, it displays "L" on the 7 segment LED display
- Then when pressing the other one, it displays "r".

This involved using the same method as in Section 2 to figure out the 7 – Segment display and hexadecimal values for 'L' and 'r':

- 'L' = 0xC7
- 'r' = 0xAF

Flowchart: Task 2

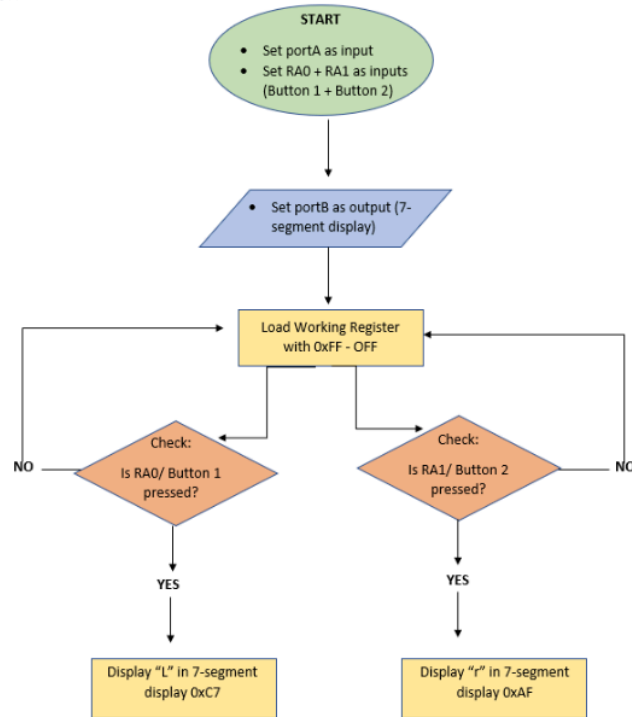


Fig. 8. Flowchart for code process in Task 2

Annotated Assembly: Task 2

```

main.asm
1 ;=====
2 ; Main.asm file generated by New Project wizard
3 ;
4 ; Created: Wed Oct 30 2024
5 ; Author: Rodina M
6 ; Processor: PIC16F84A
7 ; Compiler: MPASM (Proteus)
8 ; Description: Write a program so that when you press one of the two buttons
9 ; it displays "L" on the 7 segment LED display, then when you
10 ; press the other one, it displays "r".
11 ;=====
12
13 ;=====
14 ; DEFINITIONS
15 ;=====
16
17 #include p16f84a.inc ; Include register definition file
18
19 portA equ 0x05 ;port A address
20 portB equ 0x06 ;port B address
21
22 trisa equ 0x05 ;trisa address in register bank 1
23 trisb equ 0x06 ;trisb address in register bank 1
24
25 status equ 0x03 ;status register
26 RP0 equ 5 ;bit 5 of status register
27 RP1 equ 6 ;bit 6 of status register
28
  
```

Fig. 9. Defining addresses for port B and port A to initialise program

```

28
29 ;=====
30 ; VARIABLES
31 ;=====
32
33 count    equ    0x0c        ; counter 0 to 255
34 nCount   equ    0x0d        ; pause routine inner loop counter
35 mCount   equ    0x0e        ; pause routine outer loop counter
36
37 ;=====
38 ; RESET and INTERRUPT VECTORS
39 ;=====
40
41 ; Reset Vector
42 RST      code 0x0
43         goto Start
44
45 ;=====
46 ; CODE SEGMENT
47 ;=====
48
49 PGM      code
50 Start
51         bcf    status,RP0    ; clear bank selection bits in the status reg
52         bsf    status,RP0    ; select bank 1
53
54         movlw  0x00          ; load w with 0x00
55         movwf  trisb         ; load it into trisb to make portb all outputs
56
57         movlw  0x03          ; load w with 0x03
58         movwf  trisa         ; load it into trisa to make the two lsb's inputs
59
60         bcf    status,RP0    ; reselect bank 0
61         clrf   portB         ; clear portB - turn off all segments initially
62
63
64 Main
65
66         movlw  0xFF          ; load w with 0xff - display OFF
67         btfss  portA, 0      ; check that the button on RA0 is pressed
68         call   display_l     ; call subroutine display_l
69
70
71         btfss  portA, 1      ; check that the button on RA1 is pressed
72         call   display_r     ; call subroutine display_r
73         movwf  portB         ; send the character pattern to LED display
74
75         goto   Main         ; repeat loop
76

```

Fig. 10., Fig. 11. Main program of button input program to display specific characters

```

74      goto    Main          ; repeat loop
75
76
77 display_l
78
79      call    pause         ; call subroutine pause
80      retlw   0xC7          ; return 'l' to working register
81
82 display_r
83
84      call    pause         ; call subroutine pause
85      retlw   0xAF          ; return 'r' to working register
86
87 ;=====
88 pause
89      movlw   0x0A          ; mCount = w: outer loop counter
90      movwf   mCount
91 loadN
92      movlw   0xFF          ; set w = 255 decimal
93      movwf   nCount        ; nCount = w: inner loop counter
94 decN
95      decfsz  nCount, f      ; nCount--
96      goto    decN          ; if nCount != 0 then repeat nCount--
97
98      decfsz  mCount, f      ; else decrement Count
99      goto    loadN          ; if mCount != 0 then
100                                ; reload nCount to 255 and decrement
101      return                                ; else exit subroutine
102
103 ;=====
104      END

```

Fig. 12. Subroutines showing use of RETLW to return single value to W Register in Task 2

Task 3

The third task was not too dissimilar from Task 2, however some of the instructions and their positions in the code needed to be altered. Task 3 was a program requiring switch activation; “latching”, so that the user would not need to continually press the button for the effect to remain:

- one button causes all of the LED segments to light
- the other switch turns them off

Using the instruction RETLW returns from a subroutine and loads the W Register with the required value. In this instance, the subroutines would return with values OFF (0xFF) or ON (0x00) depending on user input.

Flowchart: Task 3

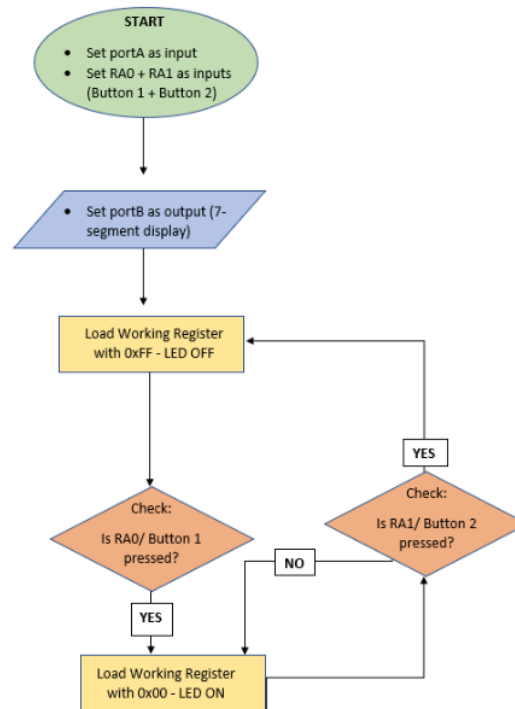


Fig. 13. Flowchart for Task 3 showing user input with 2 buttons

Annotated Assembly: Task 3

```

main.asm
1  ;=====
2  ; Main.asm file generated by New Project wizard
3  ;
4  ; Created:   Wed Oct 30 2024
5  ; Author:   Rodina M
6  ; Processor: PIC16F84A
7  ; Compiler: MPASM (Proteus)
8  ; Description: Write another program, so one button causes all of the LED
9  ;               segments to light, and the other switch turns them off.
10 ;               (the switch activations should be 'latching' i.e. you don't
11 ;               have to hold the button down for the on/off effect to remain)
12 ;=====
13 ;=====
14 ; DEFINITIONS
15 ;=====
16 ;=====
17 #include p16f84a.inc           ; Include register definition file
18 ;=====
19 portA equ 0x05                 ; port A address
20 portB equ 0x06                 ; port B address
21 ;=====
22 trisa equ 0x05                 ; trisa address in register bank 1
23 trisb equ 0x06                 ; trisb address in register bank 1
24 ;=====
25 status equ 0x03                ; status register
26 RP0 equ 5                      ; bit 5 of status register
27 RP1 equ 6                      ; bit 6 of status register
28 ;=====
29 ;=====
30 ; VARIABLES
31 ;=====
32 ;=====
  
```

```

32
33 count equ 0x0c ; counter 0 to 255
34 nCount equ 0x0d ; pause routine inner loop counter
35 mCount equ 0x0e ; pause routine outer loop counter
36
37 ;=====
38 ; RESET and INTERRUPT VECTORS
39 ;=====
40
41 ; Reset Vector
42 RST code 0x0
43 goto Start
44
45 ;=====
46 ; CODE SEGMENT
47 ;=====
48
49 PGM code
50 Start
51 bcf status,RP0 ; clear bank selection bits in the status reg
52 bcf status,RP1
53 bsf status,RP0 ; select bank 1
54
55 movlw 0x00 ; Load w with 0x00
56 movwf trisb ; Load it into trisb to make portb all outputs
57
58 movlw 0x03 ; Load w with 0x03
59 movwf trisa ; Load it into trisa to make the two lsb's inputs
60
61 bcf status,RP0 ; reselect bank 0
62 clrf portB ; clear portB - turn off all segments initially
63 movlw 0xff ; Set all pins HIGH (off) - Latching
64
65 Main
66
67
68 bcf status,RP0 ; reselect bank 0
69 clrf portB ; clear portB - turn off all segments initially
70 movlw 0xff ; Set all pins HIGH (off) - Latching
71
72 Main
73
74 btfss portA, 0 ; check that the button on RA0 is pressed
75 call display_ON ; call subroutine display_ON
76
77 btfss portA, 1 ; check that the button on RA1 is pressed
78 call display_OFF ; call subroutine display_OFF
79 movwf portB ; load it into portB
80
81 goto Main ; repeat loop
82
83 display_ON
84 call pause ; call pause subroutine
85 retlw 0x00 ; return 0x00 - all pins ON to w reg
86
87 display_OFF
88 call pause ; call pause subroutine
89 retlw 0xFF ; return 0xFF - all pins OFF to w reg
90
91 ;=====
92 pause
93 movlw 0x0A ; set w = 10 in decimal
94 movwf mCount ; mCount = w: outer loop counter
95
96 loadN
97 movlw 0xff ; set w = 255 decimal
98 movwf nCount ; nCount = w: inner loop counter
99
100

```

Fig. 14., Fig. 15., Fig. 16. Annotated Assembly listing for Task 3

Task 4

Task 4 was an amalgamation of the 3 previous tasks:

- Combine all of the techniques together, by writing a new program
- Start by displaying '5' but count up (to a maximum of 9) if one button is pressed
- Count down (to a minimum of 0) if you press the other.

Due to the fact that the recommended '*RETLW*' instruction to list the different Hexadecimal values was not undertaken in Task 1, this task was challenging and the output is not up to standard. Although the button presses do trigger the correct process, it is unable to function as planned as it is amateur in attempt. Further study in this area is recommended going forward.

Flowchart: Task 4

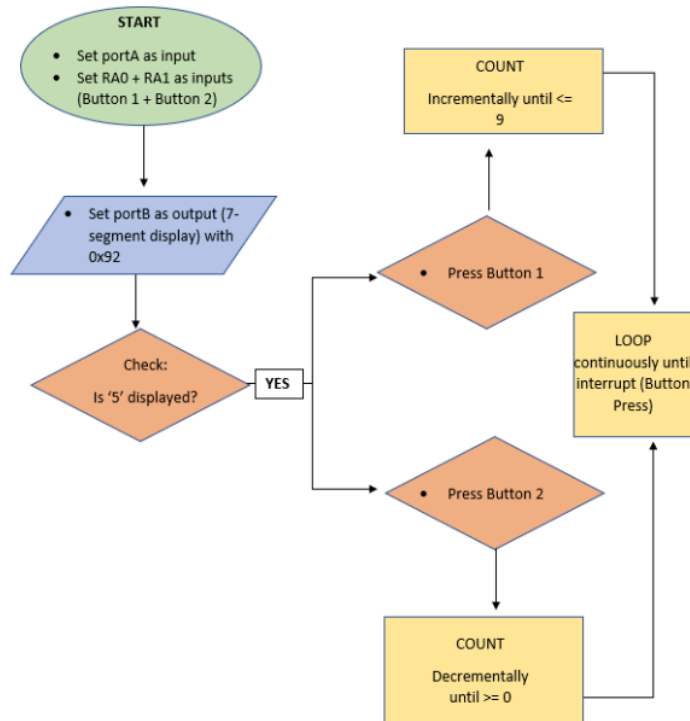


Fig. 17. Flowchart for Task 4 showing intended process if code has functioned

Annotated Assembly: Task 4

```

47 ;=====
48
49 PGM code
50 Start
51     bcf     status,RP0    ; clear bank selection bits in the status reg
52     bcf     status,RP1
53
54     bsf     status,RP0    ; select bank 1
55
56     movlw   0x00          ; Load w with 0x00
57     movwf   trisb         ; Load it into trisb to make portb all outputs
58
59     movlw   0x03          ; Load w with 0x03
60     movwf   trisa         ; Load it into trisa to make the two lsb's inputs
61
62     bcf     status,RP0    ; reselect bank 0
63
64     clrf    portB         ; clear portB - turn off all segments initially
65     clrf    portA         ; clear portA
66
67     clrf    count        ; clear count
68
69
70 Main
71     movlw   0x92          ; Set pin to display '5'
72
73     btfss   portA, 0      ; check that the button on RA0 is pressed
74     call    button_1     ; call subroutine button_1
75
76
77     btfss   portA, 1      ; check that the button on RA1 is pressed
78     call    button_2     ; call subroutine button_2
79     movwf   portB         ; Load it into portB
80
81     goto    Main         ; repeat loop
82
83 button_1
84
85
86     movlw   0x92          ; Load w with 0x92 - segment 5
87     movwf   portB         ; move W to port B to update the display
88     addlw   1             ; add +1 to W
89
90     call    pause        ; delay 200ms by subroutine
91
92
93     movlw   0x82          ; Load w with 0x82 - segment 6
94     movwf   portB         ; move W to port B to update the display
95     addlw   1             ; add +1 to W
96
97     call    pause        ; delay 200ms by subroutine
98
99     movlw   0xF8          ; Load w with 0xF8 - segment 7
100    movwf   portB         ; move W to port B to update the display
101    addlw   1             ; add +1 to W
102

```

Fig. 18., Fig. 19. Initialising Task 4 and setting up Main program; assigning ports.

```

118     movwf    portB      ; move W to port B to update the display
119     addlw    1          ; add +1 to W
120     movwf    count      ; save W in count
121
122     call     pause      ; delay 200ms by subroutine
123
124     return
125
126 button_2
127
128     movlw    0x90        ; Load w with 0x09 - segment 9
129     movwf    portB      ; move W to port B to update the display
130     addlw    1          ; add +1 to W
131
132     call     pause      ; delay 200ms by subroutine
133
134     movlw    0x80        ; Load w with 0x80 - segment 8
135     movwf    portB      ; move W to port B to update the display
136     addlw    1          ; add +1 to W
137
138     call     pause      ; delay 200ms by subroutine
139
140     movlw    0xF8        ; Load w with 0xF8 - segment 7
141     movwf    portB      ; move W to port B to update the display
142     addlw    1          ; add +1 to W
143
144     call     pause      ; delay 200ms by subroutine
145
146
147     movlw    0x82        ; Load w with 0x82 - segment 6
148     movwf    portB      ; move W to port B to update the display
149     addlw    1          ; add +1 to W
150
151
152     call     pause      ; delay 200ms by subroutine
153
154
155     movwf    portB      ; move W to port B to update the display
156     addlw    1          ; add +1 to W
157     movwf    count      ; save W in count
158
159     call     pause      ; delay 200ms by subroutine
160
161     return
162
163 ;=====
164 pause
165
166     movlw    0xFF        ; mCount = w: outer Loop counter
167     movwf    mCount
168
169     movlw    0xFF        ; set w = 255 decimal
170     movwf    nCount      ; nCount = w: inner Loop counter
171
172 decN
173     decfsz   nCount, f    ; nCount--
174     goto     decN        ; if nCount != 0 then repeat nCount--
175
176     decfsz   mCount, f    ; else decrement Count
177     goto     loadN       ; if mCount != 0 then
178                             ; reload nCount to 255 and decrement
179                             ; else exit subroutine
180     return
181
182 ;=====
183     END
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217

```

Fig. 20., Fig. 21. Annotated Assembly language listing for Task 4 – End of program

Task 5

Task 5 required the use of the Piezo buzzer for the first time and this was triggered by pressing one of the buttons (RA0):

- Write a new program so that the piezo sounder produces a tone while one of the switches/buttons is pressed
- Get Proteus to output the virtual speaker/sounder to the actual PC speaker
- If this fails, attach a virtual oscilloscope to the speaker/sounder and monitor the waveform visually.

There was a sound emitting from the laptop speaker however, as seen in Fig. 26., a square wave from the oscilloscope reading can be viewed.

Flowchart: Task 5

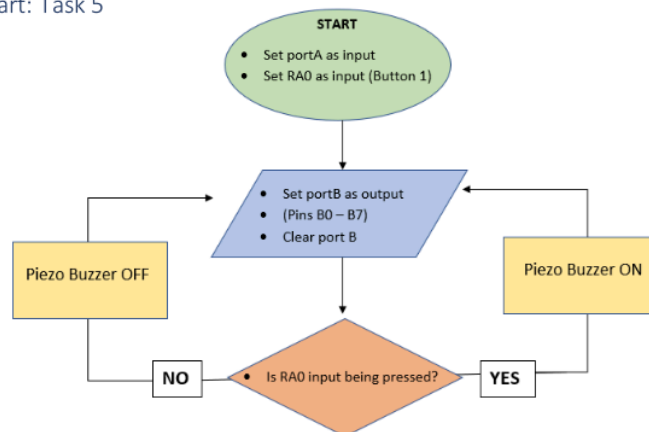


Fig. 22. Simple flowchart showing the process of Task 5 with Piezo buzzer

Annotated Assembly: Task 5

```

1  ;=====
2  ; Main.asm file generated by New Project wizard
3  ;
4  ; Created: Wed Oct 30 2024
5  ; Processor: PIC16F84A
6  ; Author: Rodina M
7  ; Compiler: MPASM (Proteus)
8  ; Description: 5.      Write a new program so that the piezo sounder
9  ;                      produces a tone while one of the switches/buttons is pressed.
10 ;                      As you are using a simulation program and not the actual hardware,
11 ;                      see if you can get Proteus to output the virtual speaker/sounder
12 ;                      to the actual PC speaker, but if this fails, attach a virtual
13 ;                      oscilloscope to the speaker/sounder and monitor the waveform visually.
14 ;=====
15
16 ;=====
17 ; DEFINITIONS
18 ;=====
19
20 #include p16f84a.inc      ; Include register definition file
21
22 portA equ 0x05            ;port A address
23 portB equ 0x06            ;port B address
24
25 trisa equ 0x05            ;trisa address in register bank 1
26 trisb equ 0x06            ;trisb address in register bank 1
27
28 status equ 0x03           ;status register
29 RP0 equ 5                 ;bit 5 of status register
30 RP1 equ 6                 ;bit 6 of status register
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52 PGM   code
53 Start
54     bcf     status,RP0     ; clear bank selection bits in the status reg
55     bcf     status,RP1     ; clear bank selection bits in the status reg
56
57     bsf     status,RP0     ; select bank 1
58
59     movlw   0x00           ; load w with 0x00
60     movwf   trisb          ; load it into trisb to make portb all outputs
61
62     movlw   0x03           ; load w with 0x03
63     movwf   trisa          ; load it into trisa to make the two lsb's inputs
64
65     bcf     status,RP0     ; reselect bank 0
66
67     clrf    portA          ; clear portA
68     clrf    portB          ; clear portB - turn off all segments initially
69     movlw   0xff           ; Set all pins HIGH (off) - Latching
70
71 Main
72
73     btfss   portA, 0       ; check that the button on RA0 is pressed, skip if set
74     call    play_sound     ; call subroutine play_sound
75     movwf   portB          ; load it into portB
76
77     goto    Main          ; repeat loop
78
79 play_sound
80
81     bcf     portB, 0xFF    ; bit clear, portB set at 255 (OFF)
82     call    pause         ; call pause subroutine
83

```

Fig. 23., Fig. 24. Annotated Assembly language showing initialisation and subroutines of Task 5

```

72
73      btfss    portA, 0      ; check that the button on RA0 is pressed, skip if set
74      call     play_sound    ; call subroutine play_sound
75      movwf    portB         ; load it into portB
76
77      goto     Main          ; repeat loop
78
79 play_sound
80
81      bcf      portB, 0xFF    ; bit clear, portB set at 255 (OFF)
82      call     pause         ; call pause subroutine
83
84 ;=====
85 pause
86
87      movlw    1              ; mCount = w: outer Loop counter
88      movwf    mCount
89
90 loadN
91      movlw    0xFF           ; set w = 255 decimal
92      movwf    nCount         ; nCount = w: inner Loop counter
93
94 decN
95      decfsz   nCount, f      ; nCount--
96      goto     decN          ; if nCount != 0 then repeat nCount--
97
98      decfsz   mCount, f      ; else decrement Count
99      goto     loadN         ; if mCount != 0 then
100                                ; reload nCount to 255 and decrement
101                                ; else exit subroutine
102      return
103 ;=====
104      END

```

Fig. 25. End of subroutines and annotated assembly language for Task 5 and Piezo buzzer

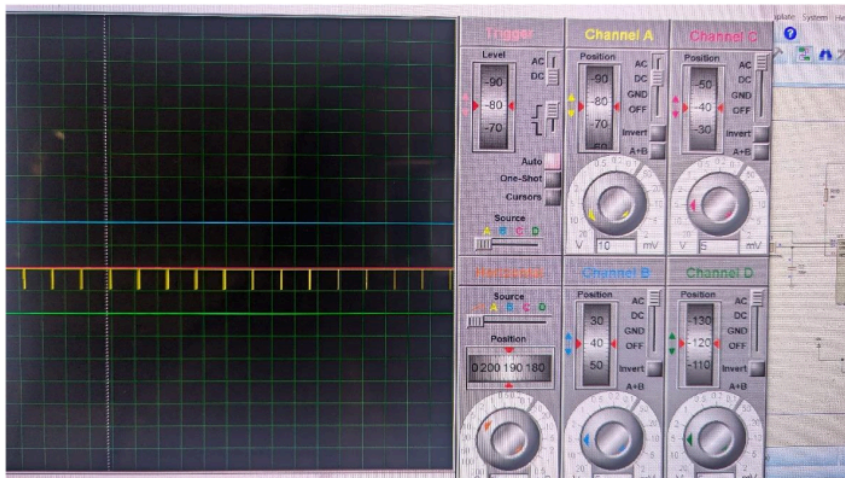


Fig. 26. Oscilloscope showing output of square wave from Task 5 Piezo buzzer

Conclusion

This report has explored and demonstrated the versatility of the PIC16F84A microcontroller. Using Assembly language and instructions, the PIC can be programmed to display characters on a 7-segment display, complete continuous loops, increment and decrement values in a loop and interact with a large variety of peripherals.

As a recommendation for the future, more efficient and established Assembly language programs could be demonstrated with less instructions and more finesse.

References

- [1] "PIC16F84A Data Sheet," Microchip Technology Inc., 2001. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>. [Accessed January 2025].
- [2] "Microchip PIC16F84A-04/P, 8bit PIC Microcontroller, PIC16F, 4MHz, 1024 x 14 words, 64 x 14 words Flash, 18-Pin PDIP," [Online]. Available: <https://uk.rs-online.com/web/p/microcontrollers/3792897> [Accessed January 2025].
- [3] M. P. Bates, Programming 8-Bit PIC Microcontrollers in C with Interactive Hardware Simulation, Newnes, 2008, p. 4.
- [4] www.sciencedirect.com, "CHAPTER 2 - Introducing the PIC mid-range family and the 16F84A," in *Designing Embedded Systems with PIC Microcontrollers*, Newnes, 2010, pp. 27-49.
- [5] "7-segment Display," [Online]. Available: <https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html> [Accessed January 2025].
- [6] "The Role of Oscillators in Microcontrollers," [Online]. Available: <https://resources.pcb.cadence.com/blog/2022-the-role-of-oscillators-in-microcontrollers>. [Accessed January 2025].
- [7] P. H. Winfield Hill, The Art of Electronics, 3 ed., Cambridge University Press, 2015, p. 443.
- [8] E. Ashley, "What is NPN Transistor? Definition, types & applications.," 28 April 2021. [Online]. Available: <https://www.rs-online.com/designspark/what-is-npn-transistor-definition-types-applications>. [Accessed January 2025].
- [9] "What is a Piezo Buzzer: Benefits, How to Use it & How it Works," 4 October 2023. [Online]. Available: https://fhdmfg.com/news/news_detail/piezo-buzzers. [Accessed January 2025].

- [10] "AVR Microcontroller Hardware Design Considerations - Recommended Capacitor Values," [Online]. Available: <https://onlinedocs.microchip.com/oxy/GUID-F626284A-58F0-4C25-A6F3-0EA5054F3E2B-en-US-6/GUID-782EECEC-624C-42AB-BB63-677EC267B920.html>. [Accessed January 2025].
- [11] CD-Team, "7 Segment Display Pinout, Codes, Working, and Interfacing," 5 June 2024. [Online]. Available: <https://www.electronicsforu.com/resources/7-segment-display-pinout-understanding>. [Accessed January 2025].

ACDA4004_ Rodina_Macleod_Micros_Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

63/100

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 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

GRADING FORM: GJ_2019

RODINA MACLEOD

317

ENGAGEMENT WITH LITERATURE



A good number of sources well cited

75

KNOWLEDGE AND UNDERSTANDING

 All components have a description of their fundamental operation. Could have included more depth of detail for higher marks **67**

COGNITIVE AND INTELLECTUAL

 Fairly straightforward implementation of the code, so as this section is for flair and innovation, this mark is fairly low. **35**

PRACTICAL APPLICATION

 Good circuit implementation and most of the tasks attempted.
Good flowcharting. **70**

TRANSFERABLE SKILLS

 Generally well laid out and presented. **70**