

OA_Micros_assignment

by Oliver Atkinson

Submission date: 13-Jan-2025 07:19PM (UTC+0000)

Submission ID: 248319507

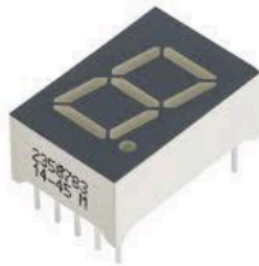
File name: 52084_Oliver_Atkinson_OA_Micros_assignment_621295_1592952110.docx (4.18M)

Word count: 4782

Character count: 23871

Using and programming the 7 segment display prototype circuit.

Oliver Atkinson



ACDA4004

Gareth Jones

31/12/2024

Cover Image from [RS Components](#)

Contents

Figure Table	3
Section 1 – The circuit	4
Overall	4
The Breakdown	5
Chip based components	5
Discrete Components	6
Section 2 – Hex to Bin	8
Display Inputs	8
Section 3 – The programs	9
Initializing the PIC	9
Task 1	10
Flowchart	10
The assembly language	11
Task 2	14
Flowchart	14
Assembly	15
Task 3	16
Flowchart	16
Assembly	17
Task 4	18
Flowchart	18
Assembly	19
Task 5	21
Flowchart	21
Assembly	22
References	23
Appendix (full code)	24
Task 1	24
Task 2	27
Task 3	28
Task 4	29
Task 5	32

Figure Table

FIGURE 1 - THE ASSIGNMENT CIRCUIT.....	4
FIGURE 2 - CIRCUIT DIAGRAM	5
FIGURE 3 - BLOCK DIAGRAM OF PIC16F84A	5
FIGURE 4 - IRL CHIP DIAGRAM.....	5
FIGURE 5 - INPUTS TO SEVEN SEGMENT DISPLAY	6
FIGURE 6 - CHIP OUTPUTS FOR 7 SEGMENT DISPLAY	6
FIGURE 7 - PIN OUT FOR 7805 DIAGRAM	7
FIGURE 8 - PIC OUTPUTS TO 7 SEG DISPLAY	8
FIGURE 9 - NODE LABELS FOR 7 SEG DISPLAY	8
FIGURE 10 - INITIALIZING CODE.....	9
FIGURE 11 - FLOWCHART PART 1.....	10
FIGURE 12 - FLOWCHART PART 2.....	10
FIGURE 13 - RESET FUNCTION.....	11
FIGURE 14 - TASK 1 MAIN LOOP	11
FIGURE 15 - GET_VAL SUBROUTINE.....	12
FIGURE 16 - TIME DELAY FUNCTION.....	13
FIGURE 17 - TASK 2 FLOWCHART	14
FIGURE 18 - TASK 2 MAIN LOOP	15
FIGURE 19 - TASK 3 FLOWCHART	16
FIGURE 20 - TASK 3 MAIN LOOP	17
FIGURE 21 - TASK 4 FLOWCHART PART 1	18
FIGURE 22 - TASK 4 FLOWCHART PART 2	18
FIGURE 23 - TASK 4 INCREMENT FUNCTION	19
FIGURE 24 - TASK 4 DECREMENT FUNCTION	19
FIGURE 25 - TASK 4 MAIN LOOP	20
FIGURE 26 - TASK 4 GET_VAL FUNCTION	20
FIGURE 27 - TASK 5 FLOWCHART	21
FIGURE 28 - TASK 5 - MAIN LOOP	22

The Breakdown

A more in-depth look into the simulated circuit used in this assignment

Chip based components

PIC16F84A

The PIC16F84A is an 8 bit microcontroller / enhanced flash EEPROM that is capable of reading and writing to RAM or EEPROM memory.

This PIC micro controller only has 35 single word instructions to learn in order to program. [1]

Specs:

- 20MHz clock speed
 - 200ns instruction cycle
 - 68 bytes of DATA RAM
 - 64 bytes of DATA EEPROM
 - 8 bit wide DATA bytes
 - Four interrupt sources
- [1]

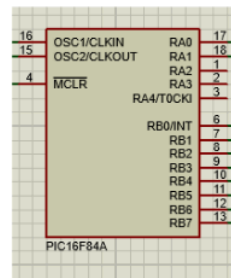


Figure 2 - circuit diagram

PDIP, SOIC

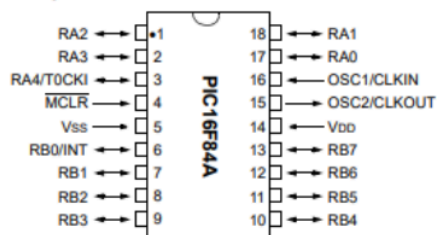


Figure 4 - irl chip diagram

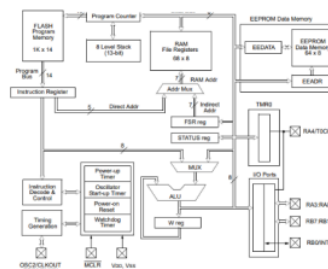


Figure 3 - block diagram of pic16f84a

Function:

In this circuit this chip is being used as the metaphorical brain of the system providing memory locations and digital computing systems reliant on binary inputs and outputs.

Within this system, the microprocessor operates with a clock frequency of 4MHz.

Crystal Oscillator

The primary use case for a crystal oscillator is to provide a stable clock input for the rest of the circuit. When a voltage is applied to a crystal oscillator the crystal oscillates at a specific frequency which is usually incredibly stable and can operate continuously without external interference. [2]

Discrete Components

7 segment display

The seven segment display in this system has a common anode which means the inputs have to be pulled low in order to display a letter/ symbol.

The 'bars' on the display correlate to specific output pins on the chip, shown below

7 seg display node	Pin on the chip
A	B0
B	B1
C	B2
D	B3
E	B4
F	B5
G	B6

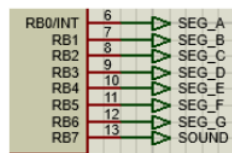


Figure 6 - chip outputs for 7 segment display

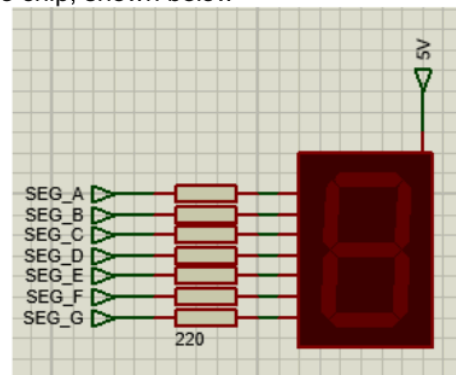


Figure 5 - inputs to seven segment display

Buzzer and transistor

The function of the piezo buzzer in this system is to signify a button being pressed through the production of noise. When voltage passes through the buzzer it produces a buzz.

The NPN transistor in the circuit functions as a switch, providing enough voltage to drive the buzzer.

Voltage regulator

This provides the chip with a stable voltage connection and keeps the voltage within the accepted ranges for the chip.

Specs:

- min input – 7V
- max input – 35V
- min output – 4.8V
- max output – 5.2V
- Current rating – 1A

[3]

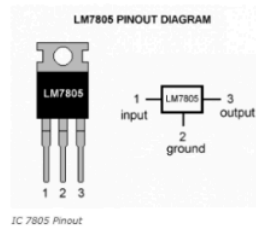


Figure 7 - pin out for 7805 diagram

Section 2 – Hex to Bin

This section covers the methods used to execute hex values on a 7 segment display and the bitwise/ binary calculations that took place.

Display Inputs

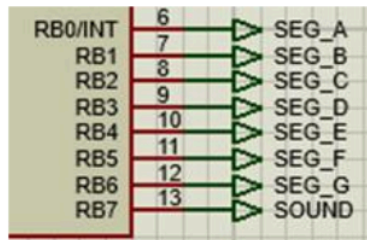


Figure 8 - PIC outputs to 7 seg display

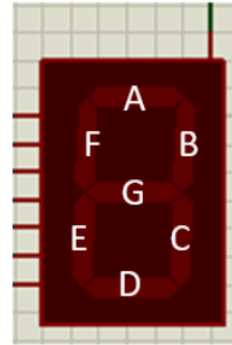


Figure 9 - node labels for 7 seg display

In order to have certain numbers appear on the display certain segments have to be powered and some have to be un-powered. For example, an '8' on the display would have all segments powered but an 'A' would have segments A, B, C, E, G, F powered. Shown below is a table with the hex value and segments requiring power:

Segments requiring power										
Hex Value	A	B	C	D	E	F	G	Binary value	Inverted	Hex
0								x0111111	1000000	40
1								x0000110	1111001	79
2								x1101101	100100	24
3								x1111001	110000	30
4								x0110011	11001	19
5								x1011011	10010	12
6								x1011111	10	02
7								x1110000	1111000	78
8								x1111111	0	00
9								x1110011	10000	10
a								x1110111	1000	08
b								x0011111	11	03
c								x1001110	1000110	46
d								x0111101	100001	21
e								x1001111	110	06
f								x1000111	1110	0E

The hex values are calculated from using the inverted binary value. This is because the 7 seg display is common anode meaning the outputs from the PIC have to be low in order to have a voltage drop across the LEDs. The 8 bit binary numbers are split into nibbles to produce the 2 digit hex values that will be input across port 'b'.

Section 3 – The programs

This section will showcase and explain the code written to perform the tasks in the assignment. Its task will have a flow chart with pseudo code as well as the annotated screenshots of the assembly code.

Initializing the PIC

The section of code shown in figure 12 is the initializing code that sets up the data banks in the microcontroller and prepares the PIC for all inputs and outputs used in the program. This is common between all tasks.

BCF STATUS, RP0 - Accesses bank 0 which on the PIC16F84A is where the PORT A and PORT B memory addresses are located, along with all general purposes (literal, working, file) [3]

CLRF PORTB – clears all values on port b

BSF STATUS, RP0 – accesses bank 1 which houses the TRISA and TRISB. These addresses tell the ports whether they are inputs or outputs.

MOVLW 0x00 – this moves the hex value 0x00 onto the working register

MOVWF TRISB – this moves the hex value from the working register onto the TRISB address which assigns PORTB as outputs.

BCF STATUS, RP0 – moves back into bank 0 in order to complete the rest of the programs

```
15 prog      code
16 start:
17     BCF     STATUS, RP0      ;accesses bank 0
18     CLRF    PORTB           ;clears all assigned values off of PORT B
19
20     BSF     STATUS, RP0      ;accesses bank 1
21     MOVLW   0x00             ;moves the value 0x00 into the working register
22     MOVWF   TRISB            ;moves this value onto TRISB - making all bits on port b outputs
23
24     BCF     STATUS, RP0      ;accesses bank 0
25
```

Figure 10 - initializing code

Task 1

This task involved counting from 0 to f in hex and displaying each value on the 7 seg display. Using a lookup table to find each value and then place it into the correct register.

Flowchart

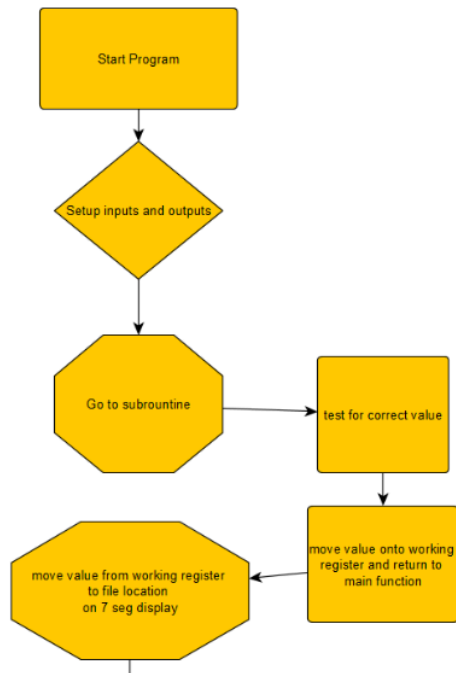


Figure 11 - flowchart part 1

The second part of the flowchart (right) shows how the system iterates through the values in hex and then resets when it hits 15.

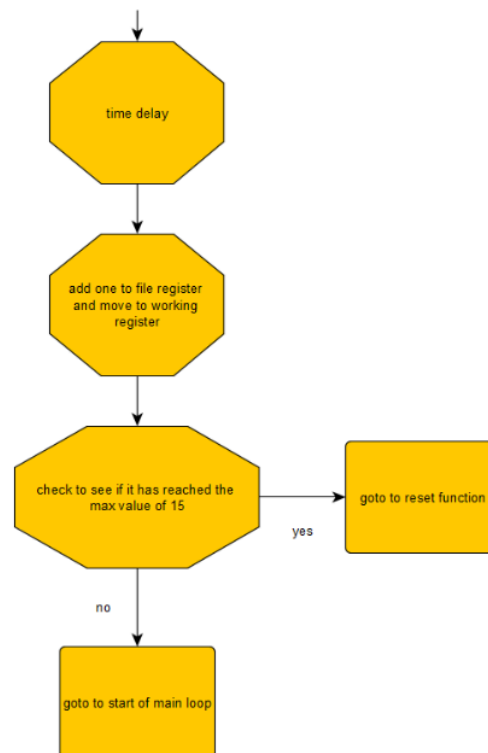


Figure 12 - flowchart part 2

The assembly language

Reset function

```
32 reset:
33     movlw 0x00      ;moves the hex value 0x00 onto the working register
34     movwf count     ;moves the contents of the working register onto the count file location in order to make the program display 0
35
```

Figure 13 - reset function

Shown in figure 13 is the first few lines of task 1. These few lines prepare the system by moving the hex value 0x00 onto the count variable. This ensures the program always starts from 0 when it begins

Main loop

```
36 ;-----
37 ;subroutine: loop
38 ;purpose: the main loop where all functions are called
39 ;registers: working, file
40 ;-----
41
42 loop:
43     call get_val      ;calls the get_val subroutine
44     movwf PORTB       ;moves the value from the working register onto portb
45     call pause        ;time delay
46     incf count, f     ;this increments the value of count in order to progress through the number sequence
47     movf count, w     ;moves the contents of count into the working register
48     xorlw 0x10        ;xors the value in count with 16 to see if it needs to be reset to zero
49     btfsc STATUS, Z   ;if the value in count is found not to be equal to 16 then the next line is skipped
50     goto reset        ;returns to the top of the code
51     goto loop         ;returns to the heading "Loop"
52
```

Figure 14 - task 1 main loop

Shown in figure 14 is the main loop of the program. This is where all subroutines are called from.

Breakdown

CALL GET_VAL – this calls the function which will find the correct hex value to output onto the 7 seg display (explain later in the report).

MOVWF PORTB – this moves the hex value from the function above onto the file location port B

CALL pause – this calls the function that produces a time delay, shown below.

INCF COUNT, F – this increases the value in the variable count by one and places it back in the f register.

MOVF count, W – this moves the variable count into the working register

XORLW 0x10 – this compares the variable count with the hex value 0x10 to check to see if it needs to be reset.

BTFSC STATUS, Z – this checks the output of the XOR instruction above, if the XOR outputs logic 0 then the next instruction is skipped.

GOTO RESET/ LOOP – this instruction makes the system jump to a specified location in the program.

GET_VAL subroutine

```
2;-----
3;;subroutine: get_val
4;purpose: acting as a lookup table for the values to display on the 7 seg display
5;registers: working and file
6;-----
7
8get_val:
9  movf    count,w      ;moves the contents of count onto the working register
10 xorlw   0x00          ;xors the contents of count with 0x00 to confirm if the value of count is 0 or not
11 btfsc   STATUS, Z     ;checks the status bit z, if it is 0 aka the number in count is not 0 then the next line is skipped
12 retlw   0x40          ;if the arithmetic check comes back as 1 then the value 0x40 is returned to the working register
13 movf    count,w      ;repeats for 1
14 xorlw   0x01
15 btfsc   STATUS, Z
16 retlw   0x79
17 movf    count,w      ;repeats for 2
18 xorlw   0x02
19 btfsc   STATUS, Z
20 retlw   0x24
21 movf    count,w      ;repeats for 3
22 xorlw   0x03
23 btfsc   STATUS, Z
24 retlw   0x30
25 movf    count,w      ;repeats for 4
26 xorlw   0x04
27 btfsc   STATUS, Z
28 retlw   0x19
29 movf    count,w      ;repeats for 5
30 xorlw   0x05
31 btfsc   STATUS, Z
32 retlw   0x12
```

Figure 15 - GET_VAL Subroutine

This subroutine is the main counter of the program. Once a value is assigned to the variable count it is XOR'd with each value 0 through f in hex. Once the output of the XOR equals 1 the correct value has been found and it returns to the main stack whilst putting the correct value into the working register to be later moved onto PORT B. (full code in the appendix)

BREAKDOWN

MOVF count, W – this moves the variable count onto the W register from the F register.

XORLW 0x00 – XORs the variable count with 0x00 (or the number between 0-f)

BTFSC STATUS, Z – checks the outputs of the XOR in order to proceed with this number or move on to the next one.

RETLW 0x40 – returns to main stack and transfers a hex value the working register

This then repeats for each number.

PAUSE Subroutine

This subroutine created a time delay by counting down from 256, 256 times. It creates a 0.2 second delay.

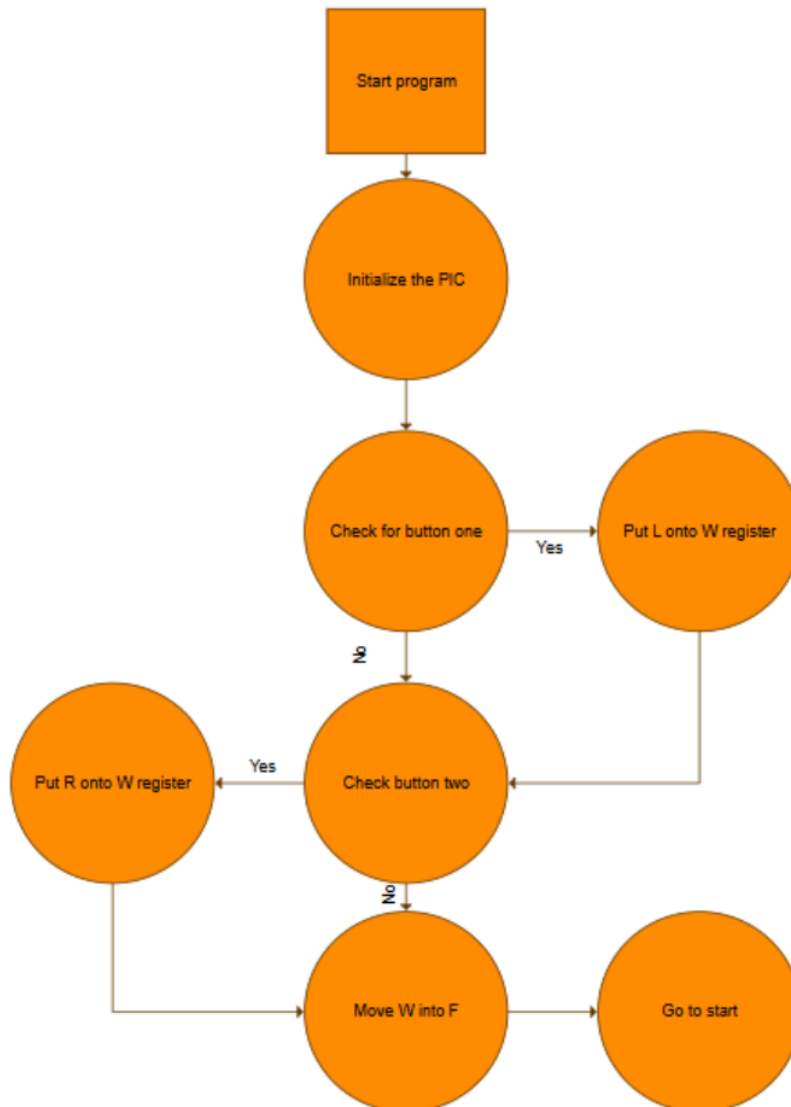
```
123 ;-----
124 ;subroutine pause: time delay
125 ;
126 ;inputs: nothing
127 ;outputs: nothing
128 ;
129 ;does: produces a 200 millisecond display
130 ;corrupts: register W
131 ;
132 ;description: two loops using "mCount" and "nCount" register files
133 ;              "mCount" is the outer loop counter (255 down to zero)
134 ;              "nCount" is the inner loop counter (255 down to zero)
135 ;
136 ;-----
137 pause
138     movlw    0xff          ; set w = 255 decimal
139     movwf    mCount        ; mCount = w: outer loop counter
140 loadN
141     movlw    0xff          ; set w = 255 decimal
142     movwf    nCount        ; nCount = w: inner loop counter
143 decN
144     decfsz   nCount, f      ; nCount--
145     goto     decN          ; if nCount != 0 then repeat nCount--
146
147     decfsz   mCount, f      ; else decrement Count
148     goto     loadN         ; if mCount != 0 then
149                             ; reload nCount to 255 and decrement
150     return                                ; else exit subroutine
151 end
```

Figure 16 - Time delay function

Task 2

Task 2 involved getting the 7 seg display to display an 'L' and 'R' on button press. This meant utilizing the BTFSC to check for each button press.

Flowchart



This flowchart shows the basic operation of task two, in which the program checks for the inputs from 2 buttons and moves the according value onto the file register (7 seg display)

If the button is pressed the hex value associated with the button is transferred onto the working register.

At the end of the cycle the value on the working register is transferred into a file location (port b) and displayed on the lcd display,

Figure 17 - task 2 flowchart

Assembly

As far as the assembly goes, the initializing code for the PIC is the same as mentioned previously. The main loop however differs from the previous code.

Main loop

```
32 loop:
33     movlw 0xFF          ;moves a standard value of 0xFF to the working register
34     btfss PORTA,0       ;tests the pin A0 to see if the bit has a signal running to it - aka button 1 has been pressed
35     movlw 0xC7          ;moves the correct hex value for an 'r' into the working register
36     btfss PORTA,1       ;tests pin A1 to see if it has been set - aka button 2 has been pressed
37     movlw 0xCE          ;moves the correct hex value for an 'l' onto the working register
38     movwf PORTB         ;moves the value stored in the working register into the file register
39
40     goto  loop          ;returns to the heading "Loop"
```

Figure 18 - task 2 main loop

MOVLW 0xff – The program starts by moving 0xff onto the working register in order to clear the 7 seg display.

BTSSS PORTA, 0 – the system then checks for button 1 being pressed. If the button is pressed then the next instruction is performed. If the button is not pressed then the next instruction is skipped.

MOVLW 0xC7 – this moves the hex value for an L onto the working register

BTSSS PORTA, 1 – checks for the other button being pressed.

MOVLW 0xCE – moves the hex value for an R onto the working register

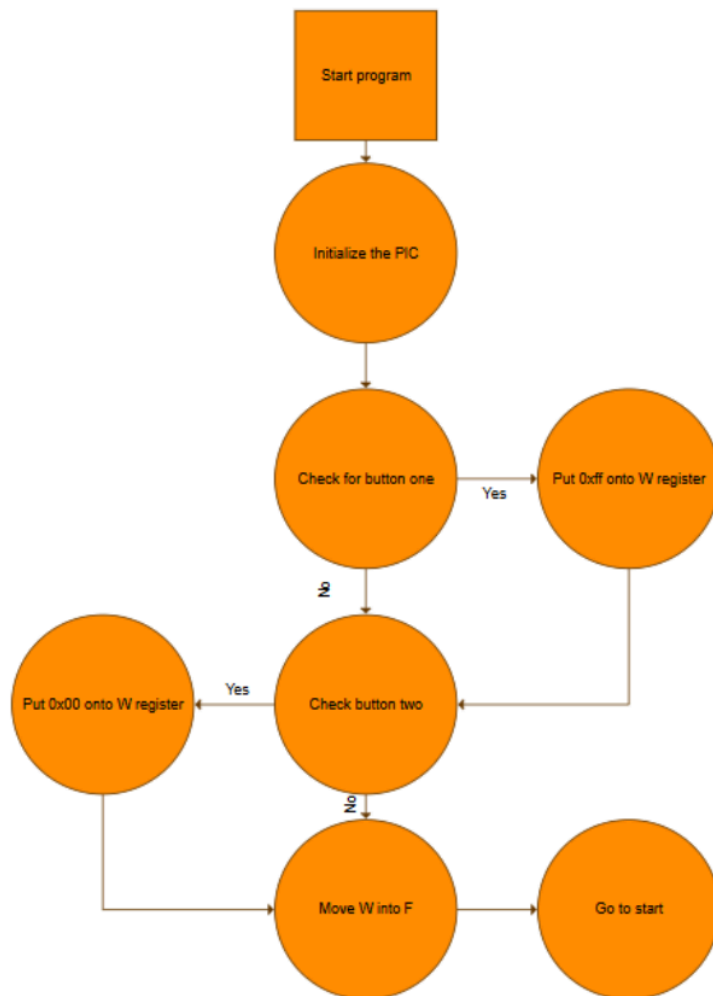
MOVWF PORTB – this moves the value in the working register onto port B (7 seg display)

This program is not latching due to the initial MOVLW 0xff. This resets the 7 seg display back to being empty after every loop, unless a button is held down.

Task 3

This task is very similar to task 2 but instead of displaying an L and an R, the program displays full nodes or no nodes.

Flowchart



This works in a very similar fashion to task 2, however this program latches the output so the 7-segment display either stays on or stays off

Figure 19 - task 3 flowchart

Assembly

This program is extremely similar to task 2 however, the hex values have changed and it no longer resets after every loop.

```
42 loop:
43     btfss PORTA, 0      ;tests for the first button being pressed, if it not pressed the next line is skipped
44     movlw 0x00          ;moves 0x00 into the working register
45     btfss PORTA, 1      ;tests for the second button being pressed, if it not pressed the next line is skipped
46     movlw 0xff          ;moves 0xff into the working register
47     movwf PORTB         ;moves the value in the working register onto portb to display on the 7 seg display
48     goto  loop         ;returns to the heading "Loop"
```

Figure 20 - task 3 main loop

This functions by checking a button input and then moving each respective hex value onto the working register. At the end of the program the value on the working register is moved onto the file location PORT B.

Task 4

Task 4 was similar to task 1 in that the get_val subroutine was involved. The program starts at 5 and with each button press the count was incremented or decremented depending on which button was pressed.

Flowchart

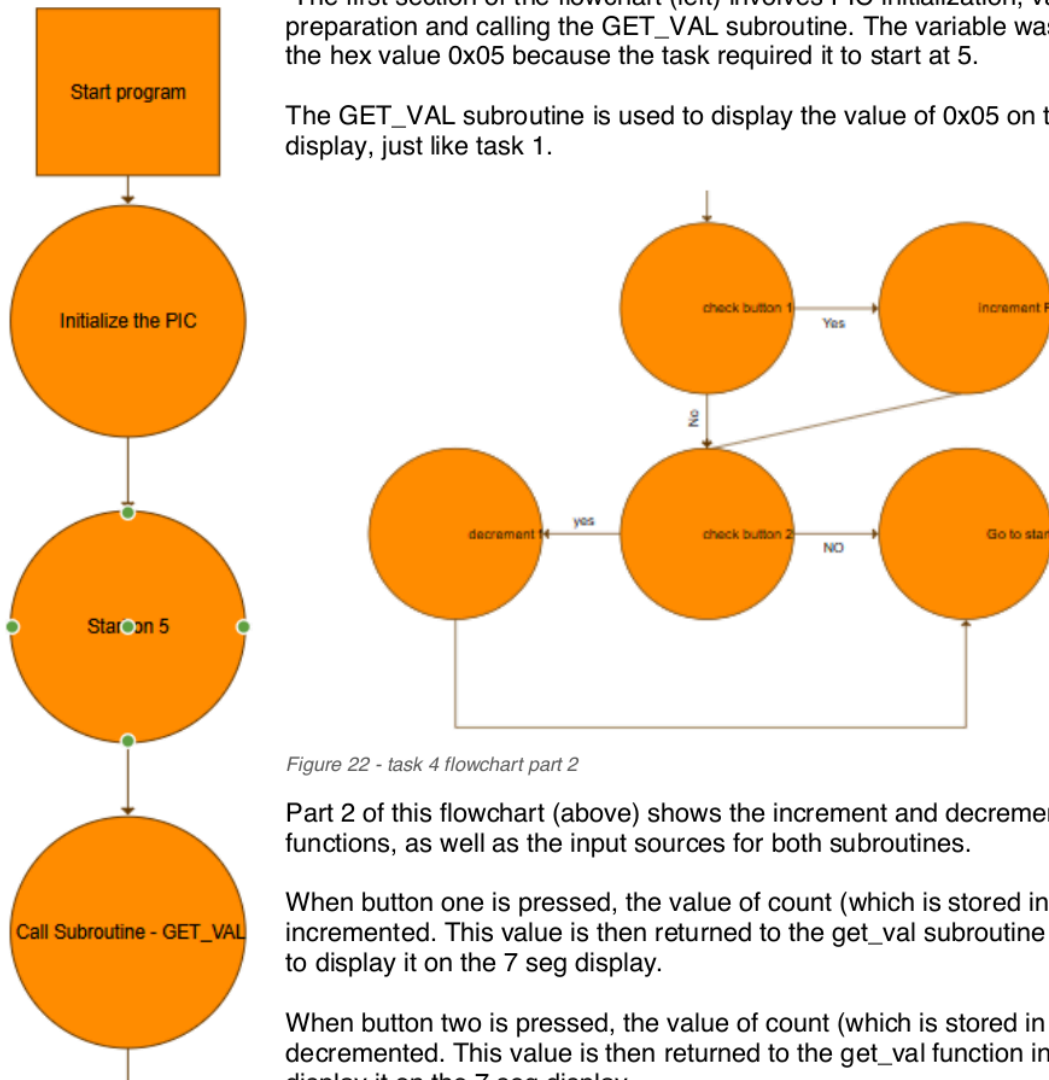


Figure 22 - task 4 flowchart part 2

Part 2 of this flowchart (above) shows the increment and decrement functions, as well as the input sources for both subroutines.

When button one is pressed, the value of count (which is stored in f) is incremented. This value is then returned to the get_val subroutine in order to display it on the 7 seg display.

When button two is pressed, the value of count (which is stored in f) is decremented. This value is then returned to the get_val function in order to display it on the 7 seg display.

Figure 21 - task 4 flowchart part 1

Assembly

As with the previous tasks, the initializing code is the same, and this utilizes the get_val subroutine however it only goes to 9 instead of f. The biggest difference between this and the first task is the increment and decrement functions within the main loop.

Increment

```
105 inc:
106     movf    count, w        ;moves the contents of count onto the working register
107     xorlw   0x09            ;xors the contents of count with 9 to check if it has reached the max value
108     btfss   STATUS, Z        ;if count has reached 9 the next is skipped
109     incf    count, f         ;increments count by 1 and returns it to the file register
110     call    pause           ;time delay
111     return                   ;returns to where it was called
```

Figure 23 - task 4 increment function

MOVF count, w – moves count onto the working register

XORLW 0x09 – checking that count is less than 9

BTFSS STATUS, Z – checks the output of the XOR operation, if one then the next line is skipped.

INCF count, f – the value in count is incremented by one and placed back in f.

CALL PAUSE – time delay.

RETURN – go to the location where the subroutine was called from.

Once the value hits 9 it will not allow the incrementing of count until it has been decremented.

Decrement

```
119 dec:
120     movf    count, w        ;same as inc function
121     xorlw   0x00            ;xors the contents of the working register with 0 to see if it has reached its minimum
122     btfss   STATUS, Z        ;if the contents of the w register is 0 the next line is skipped
123     decf    count, f         ;decrements the count by one and returns it to the file register
124     call    pause           ;time delay
125     return                   ;returns to main loop
```

Figure 24 - task 4 decrement function

This logically functions the same as the increment function however it checks for count being equal to 0. If the value is found to be high than 0 then the count can be decremented by 1. If the value is 0 then the count will no longer decrement until it is incremented.

Main Loop

The main loop is largely similar to task one, however it includes the inc and dec functions, shown below.

```
44 loop:
45     call    get_val    ;calls get_val
46     movwf   PORTB      ;moves the contents of the working register onto port b
47     btfss   PORTA, 0    ;testing if pin A0 is set or not
48     call    inc         ;if pin A0 is set then it calls the subroutine inc
49     btfss   PORTA, 1    ;testing if pin A1 is set or not
50     call    dec         ;if pin A1 is set the subroutine dec is called
51     call    pause       ;time delay
52     goto    loop        ;returns to the section header
```

Figure 25 - task 4 main loop

It puts the count onto the 7-segment display then checks for each button being pressed. When button 1 and 2 are pressed the program will increment and decrement count respectively, then moves this new value onto the 7-segment display.

Get_Val

This function is explained in task 1, and function exactly the same here except for the fact it counts to 9, not f. (full code in the appendix)

```
get_val:
40     movf    count,w      ;moves the contents of count onto the working register
41     xorlw   0x00          ;xors the value in count with 0 to confirm it is 0
42     btfsc   STATUS, Z     ;if the value is not 0 then the next line is skipped
43     retlw   0x40          ;if the value in count is confirmed to be 0 then the hex value associated with the 7 seg display is return to the working register
44     movf    count,w      ;repeats for 1
45     xorlw   0x01
46     btfsc   STATUS, Z
47     retlw   0x79          ;repeats for 2
48     movf    count,w
49     xorlw   0x02
50     btfsc   STATUS, Z
51     retlw   0x24          ;repeats for 3
52     movf    count,w
53     xorlw   0x03
54     btfsc   STATUS, Z
55     retlw   0x30          ;repeats for 4
56     movf    count,w
57     xorlw   0x04
58     btfsc   STATUS, Z
59     retlw   0x19          ;repeats for 5
60     movf    count,w
61     xorlw   0x05
62     btfsc   STATUS, Z
63     retlw   0x12
```

Figure 26 - task 4 get_val function

Task 5

This task involved a piezo sounder that needed to sound on button press. This was achieved in much similar ways to tasks 2 and 3.

Flowchart

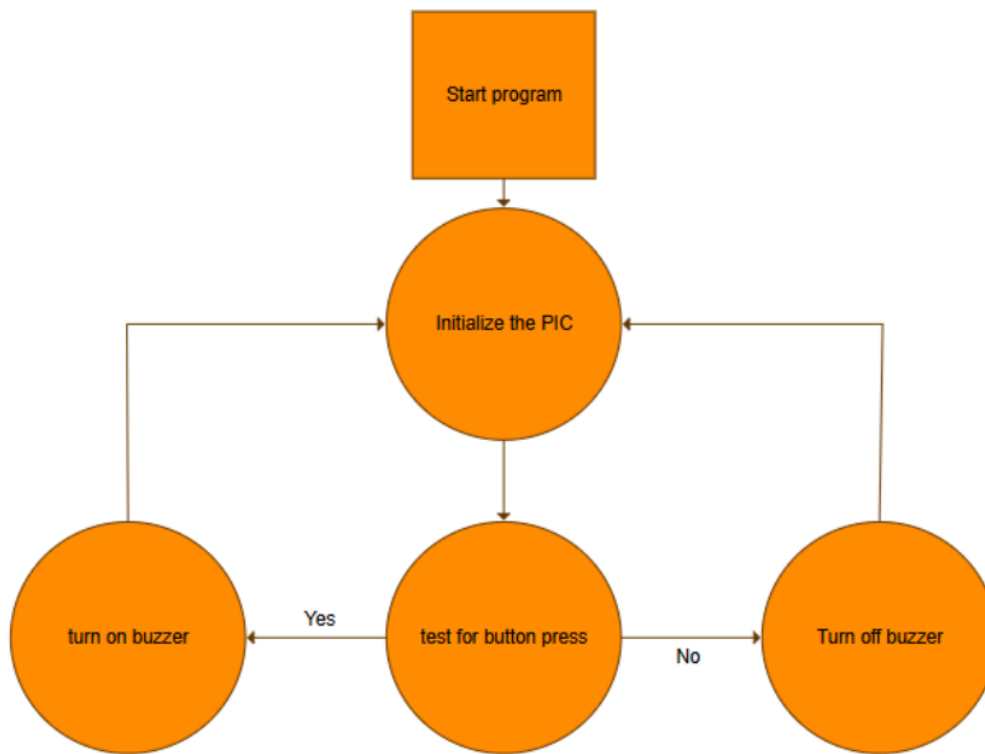


Figure 27 - Task 5 flowchart

This shows that the premise of this task is just checking the button to see if its pressed or not. This system is not latching so the buzzer only turns on while the button is held down.

Assembly

```
42 loop:
43     btfsc PORTA, 0    ;tests for the first button being pressed, if it not pressed the next line is skipped
44     movlw 0xff        ;moves 0xff into the working register this when moved onto portb will turn it off
45     btfss PORTA, 0    ;testing to see if the button is / is not pressed
46     movlw 0x7f        ;this value when moved onto the file register will turn the buzzer on
47     movwf PORTB       ;moves the value in the working register onto portb to display on the 7 seg display
48     goto  loop        ;returns to the heading "loop"
```

Figure 28 - task 5 - main loop

Shown above is the main loop from task 5.

BTFSC **PORTA, 0** – checking to see if the button is not pressed

MOVLW **0xff** – when the button is not pressed, all bits are set high so nothing is turned on.

BTFSS **PORTA, 0** – checking if the button is pressed

MOVLW **0x7f** – turn the buzzer on

MOVWF **PORTB** – moves the value onto port b

References

- [1] Microchip, "PIC16F84A Datasheet," Microchip, USA, 2001.
- [2] Electronics Tutorials, "Quartz crystal oscillators," Electronics tutorials, 31 12 24. [Online]. Available: <https://www.electronics-tutorials.ws/oscillator/crystal.html>. [Accessed 31 12 24].
- [3] Farnell, "uk.farnell.com," farnell, 13 01 2025. [Online]. Available: https://uk.farnell.com/stmicroelectronics/l7805cv/ic-v-reg-5-0v-7805-to-220-3/dp/9756078?srsId=AfmBOoq-oQA9UBZscE1DPdsQOshcE5mlAh4kEdUXnC2qjDHUJh9_ogbC. [Accessed 13 01 2025].
- [4] Microchip, "PIC16F84A Data sheet," 01 01 2001. [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/35007b.pdf>. [Accessed 10 01 2025].

Appendix (full code)

Task 1

```
#include p16f84a.inc

count equ    0x0c      ; counter 0 to 255
nCount  equ   0x0d      ; pause routine inner loop counter
mCount  equ   0x0e      ; pause routine outer loop counter

; reset and interrupt vectors

rst      code  0x0
        goto  start

; code

prog  code
start:
        BCF      STATUS, RPo      ;accesses bank 0
        CLRF     PORTB            ;clears all assigned values off of PORT B

        BSF      STATUS, RPo      ;accesses bank 1
        MOVLW    0x00             ;moves the value 0x00 into the working register
        MOVWF    TRISB            ;moves this value onto TRISB - making all bits on port b
        outputs

        BCF      STATUS, RPo      ;accesses bank 0

;-----
;subroutine: reset
;purpose: makes the program start from 0
;registers: working, file
;-----

reset:
        movlw    0x00             ;moves the hex value 0x00 onto the working register
        movwf    count           ;moves the contents of the working register onto the count file
        location in order to make the program display 0

;-----
;subroutine: loop
;purpose: the main loop where all functions are called
;registers: working, file
;-----

loop:
        call     get_val          ;calls the get_val subroutine
```

```

        movwf PORTB           ;moves the value from the working register onto portb
        call  pause           ;time delay
        incf  count, f        ;this increments the value if count in order to progress through the
number sequence
        movf  count, w        ;moves the contents of count into the working register
        xorlw 0x10            ;xors the value in count with 16 to see if it needs to be reset to zero
        btfsc STATUS, Z      ;if the value in count is found not to be equal to 16 then the next
line is skipped
        goto  reset           ;returns to the top of the code
        goto  loop            ;returns to the heading "loop"

```

```

;-----
;;subroutine: get_val
;purpose: acting as a lookup table for the values to display on the 7 seg display
;registers: working and file
;-----

```

```

get_val:
        movf  count, w        ;moves the contents of count onto the working register
        xorlw 0x00            ;xors the contents of count with 0x00 to confirm if the value of
count is 0 or not
        btfsc STATUS, Z      ;checks the status bit z, if it is 0 aka the number in count is not 0
then the next line is skipped
        retlw 0x40            ;if the arithmetic check comes back as 1 then the value 0x40 is
returned to the working register
        movf  count, w        ;repeats for 1
        xorlw 0x01
        btfsc STATUS, Z
        retlw 0x79
        movf  count, w        ;repeats for 2
        xorlw 0x02
        btfsc STATUS, Z
        retlw 0x24
        movf  count, w        ;repeats for 3
        xorlw 0x03
        btfsc STATUS, Z
        retlw 0x30
        movf  count, w        ;repeats for 4
        xorlw 0x04
        btfsc STATUS, Z
        retlw 0x19
        movf  count, w        ;repeats for 5
        xorlw 0x05
        btfsc STATUS, Z
        retlw 0x12
        movf  count, w        ;repeats for 6
        xorlw 0x06
        btfsc STATUS, Z
        retlw 0x02
        movf  count, w        ;repeats for 7
        xorlw 0x07
        btfsc STATUS, Z

```

```

retlw 0x78
movf count,w           ;repeats for 8
xorlw 0x08
btfsc STATUS, Z
retlw 0x00
movf count,w           ;repeats for 9
xorlw 0x09
btfsc STATUS, Z
retlw 0x10
movf count,w           ;repeats for a
xorlw 0x0A
btfsc STATUS, Z
retlw 0x08
movf count,w           ;repeats for b
xorlw 0x0B
btfsc STATUS, Z
retlw 0x03
movf count,w           ;repeats for c
xorlw 0x0C
btfsc STATUS, Z
retlw 0x46
movf count,w           ;repeats for d
xorlw 0x0D
btfsc STATUS, Z
retlw 0x21
movf count,w           ;repeats for e
xorlw 0x0E
btfsc STATUS, Z
retlw 0x06

retlw 0x0E           ;at this point it is assumed that the value in count is 0x0F

```

```

;-----
;subroutine pause: time delay
;
;inputs: nothing
;outputs: nothing
;
;does: produces a 200 millisecond display
;corrupts: register W
;
;description: two loops using "mCount" and "nCount" register files
;              "mCount" is the outer loop counter (255 down to zero)
;              "nCount" is the inner loop counter (255 down to zero)
;-----
pause
    movlw 0xFF           ; set w = 255 decimal
    movwf mCount         ; mCount = w: outer loop counter
loadN
    movlw 0xFF           ; set w = 255 decimal
    movwf nCount         ; nCount = w: inner loop counter

```

```

decN
    decfsz nCount, f      ; nCount--
    goto    decN          ; if nCount != 0 then repeat nCount--

    decfsz mCount, f      ; else decrement Count
    goto    loadN         ; if mCount != 0 then
                           ; reload nCount to 255 and decrement
    return              ; else exit subroutine
end

```

Task 2

```
#include p16f84a.inc
```

```
; reset and interrupt vectors
```

```

rst      code    0x0
        goto    start

```

```
; code
```

```

prog    code
start:

```

```

    BCF      STATUS, RPO      ;accesses bank 0
    CLRF    PORTB             ;clears all assigned values off of PORT B

```

```

    BSF      STATUS, RPO      ;accesses bank 1
    MOVLW    0x00             ;moves the value 0x00 into the working register
    MOVWF    TRISB            ;moves this value onto TRISB - making all bits on port b

```

```
outputs
```

```

    BCF      STATUS, RPO      ;accesses bank 0

```

```

;-----
;subroutine: loop
;purpose: main loop that performs the functions requested in the task
;registers: working, file
;-----

```

```
loop:
```

```

    movlw    0xFF              ;moves a standard value of 0xFF to the working register
    btfss    PORTA, 0          ;tests the pin A0 to see if the bit has a signal running to it -

```

```
aka button 1 has been pressed
```

```

    movlw    0xC7              ;moves the correct hex value for an 'r' into the working register
    btfss    PORTA, 1          ;tests pin A1 to see if it has been set - aka button 2 has been

```

```
pressed
```

```

    movlw    0xCE              ;moves the correct hex value for an 'l' onto the working register

```

```

        movwfPORTB           ;moves the value stored in the working register into the file
register
        goto  loop           ;returns to the heading "loop"
end

```

Task 3

```
#include p16f84a.inc
```

```
; reset and interrupt vectors
```

```
rst      code  0x0
        goto  start
```

```
; code
```

```
prog  code
start:
```

```
        BCF      STATUS,RPo           ;accesses bank 0
        CLRF PORTB           ;clears all assigned values off of PORT B
```

```
        BSF      STATUS,RPo           ;accesses bank 1
        MOVLW    0x00           ;moves the value 0x00 into the working register
        MOVWF    TRISB           ;moves this value onto TRISB - making all bits on port b
```

```
outputs
```

```
        BCF      STATUS,RPo           ;accesses bank 0
```

```

;-----
;subroutine: reset
;purpose: makes sure the program starts with nothing displaying on the 7 seg display
;registers: working and file registers
;-----

```

```
reset:
        movlw 0xff           ;clears the lcd display so it starts in the off position
        movwfPORTB           ;moves the value onto portb
```

```

;-----
;subroutine: main loop
;purpose: this is the main function of the program and where the main processes happen
;registers: working, file
;-----

```

```

loop:      btfss  PORTA, 0          ;tests for the first button being pressed, if it not pressed the next
line is skipped
           movlw 0x00              ;moves 0x00 into the working register
           btfss  PORTA, 1          ;tests for the second button being pressed, if it not pressed the next
line is skipped
           movlw 0xff              ;moves 0xff into the working register
           movwf PORTB              ;moves the value in the working register onto portb to
display on the 7 seg display
           goto  loop              ;returns to the heading "loop"

           end

```

Task 4

```

#include p16f84a.inc

count equ 0x0c          ; counter 0 to 255
nCount equ 0x0d          ; pause routine inner loop counter
mCount equ 0x0e          ; pause routine outer loop counter

; reset and interrupt vectors

rst      code 0x0
        goto start

; code

prog code
start:
        BCF     STATUS, RPo          ;accesses bank 0
        CLRF    PORTB                ;clears all assigned values off of PORT B

        BSF     STATUS, RPo          ;accesses bank 1
        MOVLW   0x00                ;moves the value 0x00 into the working register
        MOVWF   TRISB                ;moves this value onto TRISB - making all bits on port b
outputs
        BCF     STATUS, RPo          ;accesses bank 0

;-----
;subroutine: reset
;purpose: resets the seven segment display to 5
;registers: none
;-----

reset:
        movlw 0x05                  ;moves 5 into the working register

```

movwfcount ;moves 5 onto the count register in order to start the simulation
with 5 being displayed on the 7 seg display

;subroutine: loop
;purpose: the main loop where all other functions are called
;registers: working and file
;inputs: PORTA, 0, 1
;outputs: PORTB

;

loop:
 call get_val ;calls get val
 movwfPORTB ;moves the contents of the working register onto port b
 btfss PORTA, 0 ;testing if pin A0 is set or not
 call inc ;if pin A0 is set then it calls the subroutine inc
 btfss PORTA, 1 ;testing if pin A1 is set or not
 call dec ;if pin A1 is set the subroutine dec is called
 call pause ;time delay
 goto loop ;returns to the section header

;subroutine: get_val
;purpose: acting as a lookup table for the values to display on the 7 seg display
;registers: working and file

;

get_val:
 movf count,w ;moves the contents of count onto the working register
 xorlw 0x00 ;xors the value in count with 0 to confirm it is 0
 btfsc STATUS, Z ;if the value is not 0 then the next line is skipped
 retlw 0x40 ;if the value in count is confirmed to be 0 then the hex value
associated with the 7 seg display is return to the working register
 movf count,w ;repeats for 1
 xorlw 0x01
 btfsc STATUS, Z
 retlw 0x79
 movf count,w ;repeats for 2
 xorlw 0x02
 btfsc STATUS, Z
 retlw 0x24
 movf count,w ;repeats for 3
 xorlw 0x03
 btfsc STATUS, Z
 retlw 0x30
 movf count,w ;repeats for 4
 xorlw 0x04
 btfsc STATUS, Z
 retlw 0x19
 movf count,w ;repeats for 5
 xorlw 0x05
 btfsc STATUS, Z
 retlw 0x12

```

    movf    count,w           ;repeats for 6
    xorlw   0x06
    btfsc   STATUS, Z
    retlw   0x02
    movf    count,w           ;repeats for 7
    xorlw   0x07
    btfsc   STATUS, Z
    retlw   0x78
    movf    count,w           ;repeats for 8
    xorlw   0x08
    btfsc   STATUS, Z
    retlw   0x00

    retlw   0x10               ;at this point it is assumed the number is 9 and the hex value for the
7 seg display is return

;-----
;subroutine: inc
;purpose: increments the contents of count by 1 whilst checking to see if it has reached the max value,
9
;registers: working and file
;-----

inc:
    movf    count, w           ;moves the contents of count onto the working register
    xorlw   0x09               ;xors the contents of count with 9 to check if it has reached the max
value
    btfss   STATUS, Z           ;if count has reached 9 the next is skipped
    incf    count, f           ;increments count by 1 and returns it to the file register
    call    pause               ;time delay
    return                      ;returns to where it was called

;-----
;subroutine: dec
;purpose: decrements the contents of count by 1 whilst checking to see if it has reached the min value,
0
;registers: working and file
;-----

dec:
    movf    count, w           ;same as inc function
    xorlw   0x00               ;xors the contents of the working register with 0 to see if it has
reached its minimum
    btfss   STATUS, Z           ;if the contents of the w register is 0 the next line is skipped
    decf    count, f           ;decrements the count by one and returns it to the file register
    call    pause               ;time delay
    return                      ;returns to main loop

;-----
;subroutine pause: time delay
;
;inputs: nothing
;outputs: nothing

```

```

;
;does: produces a 200 millisecond delay
;corrupts: register W
;
;description: two loops using "mCount" and "nCount" register files
;              "mCount" is the outer loop counter (255 down to zero)
;              "nCount" is the inner loop counter (255 down to zero)
;
;-----
pause
    movlw 0xff      ; set w = 255 decimal
    movwf mCount    ; mCount = w: outer loop counter
loadN
    movlw 0xff      ; set w = 255 decimal
    movwf nCount    ; nCount = w: inner loop counter
decN
    decfsz nCount, f ; nCount--
    goto   decN      ; if nCount != 0 then repeat nCount--

    decfsz mCount, f ; else decrement Count
    goto   loadN     ; if mCount != 0 then
                    ; reload nCount to 255 and decrement
    return          ; else exit subroutine
end

```

Task 5

```
#include p16f84a.inc
```

```
; reset and interrupt vectors
```

```
rst      code  0x0
        goto  start
```

```
; code
```

```
prog  code
start:
```

```

    BCF      STATUS, RPO      ;accesses bank 0
    CLRF PORTB                ;clears all assigned values off of PORT B

    BSF      STATUS, RPO      ;accesses bank 1
    MOVLW    0x00              ;moves the value 0x00 into the working register
    MOVWF    TRISB             ;moves this value onto TRISB - making all bits on port b

```

```
outputs
```

BCF STATUS,RP0 ;accesses bank 0

```
;-----  
;subroutine: reset  
;purpose: makes sure the program starts with nothing displaying on the 7 seg display  
;registers: working and file registers  
;-----
```

```
reset:  
    movlw 0xff          ;clears the lcd display so it starts in the off position  
    movwf PORTB         ;moves the value onto portb
```

```
;-----  
;subroutine: main loop  
;purpose: this is the main function of the program and where the main processes happen  
;registers: working, file  
;-----
```

```
loop:  
    btfsc PORTA, 0      ;tests for the first button being pressed, if it not pressed the next  
line is skipped  
    movlw 0xff          ;moves 0xff into the working register this when moved onto portb  
will turn it off  
    btfss PORTA, 0      ;testing to see if the button is / is not pressed  
    movlw 0x7f          ;this value when moved onto the file register will turn the buzzer on  
    movwf PORTB         ;moves the value in the working register onto portb to  
display on the 7 seg display  
    goto loop           ;returns to the heading "loop"  
  
end
```

OA_Micros_assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

65/100

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

PAGE 29

PAGE 30

PAGE 31

PAGE 32

PAGE 33

GRADING FORM: GJ_2019

OLIVER ATKINSON

324

ENGAGEMENT WITH LITERATURE

 Four reference sources is too limited considering the number of components etc involved. Need to include far more for an assignment such as this. **35**

KNOWLEDGE AND UNDERSTANDING

 Despite the limited EWL, the actual K&U demonstrated is good, with excellent use of diagrams to help in the explanations. **75**

COGNITIVE AND INTELLECTUAL

 Some interesting demonstration of creativity in solving some of the problems - but a pity this didn't extend to Tasks 5 & 6 **65**

PRACTICAL APPLICATION

 Circuit well simulated. Four of the tasks done ok, but task 5 won't work as the buzzer needs to be oscillated, not just switched on. **69**

TRANSFERABLE SKILLS

 Very well written and presented. **80**