



L4 – Micros Peripherals and Interfaces (ACDA4004)

Using and Programming the 7-Segment Display Prototype Circuit

David Mathew Lewis

2219439

Electrical and Electronic Engineering

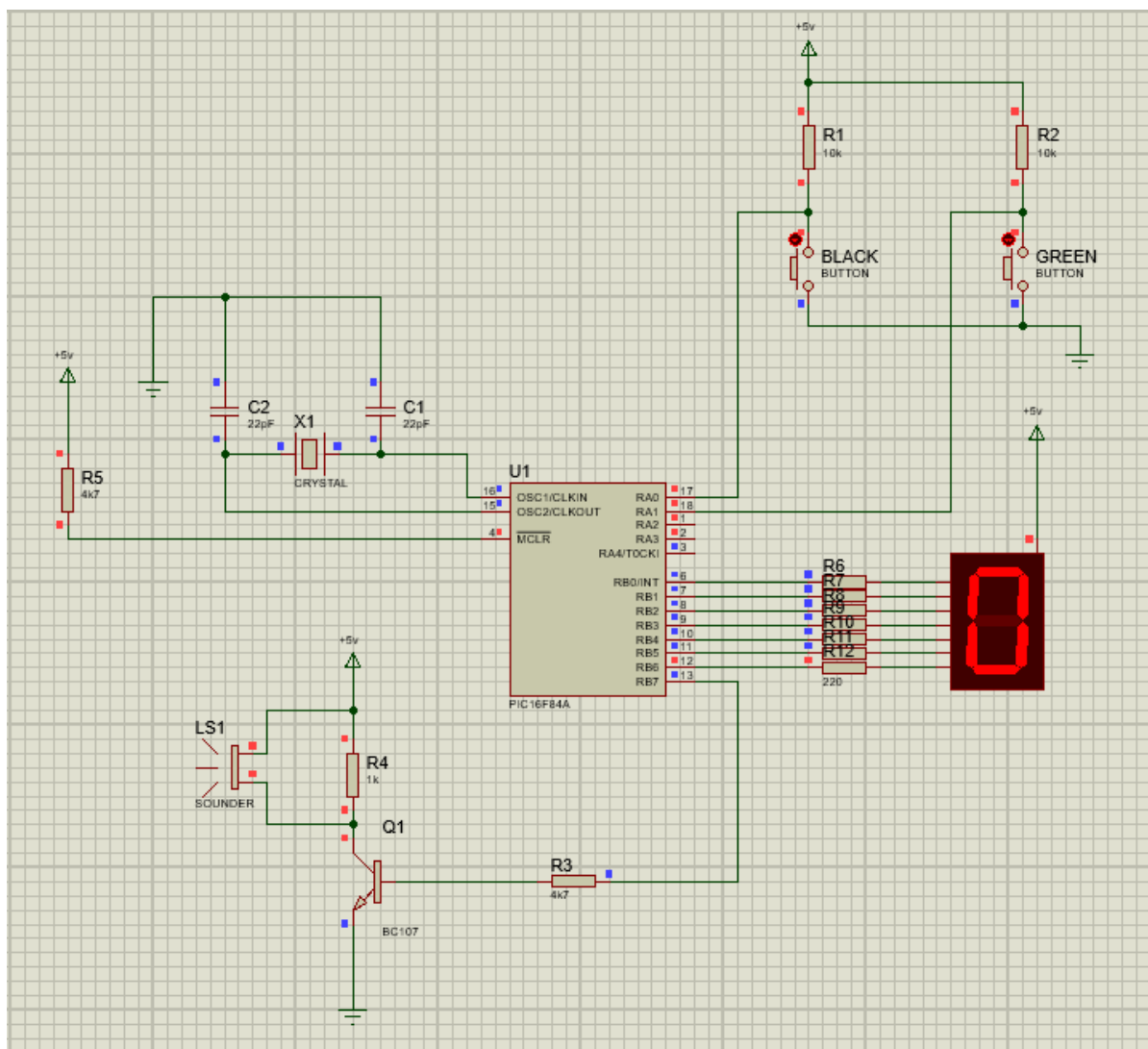
Using and Programming the 7-Segment Display Prototype CircuitDavid Mathew Lewis2219439Electrical and Electronic Engineering	1
Introduction	3
Section 1	3
Proteus Simulation.....	3
General Overview of Circuit Components	3
Crystal Oscillator	3
PIC16F84A Chip.....	4
7-Segment Display	5
Buttons.....	6
Sonder	7
Section 2	7
Segment Mapping.....	8
Look-up Table	8
Flowchart	9
Microcontroller Code.....	10
Section 3	10
Modify “Count.asm” so that the seven segment display counts from 0 to F in hexadecimal numbers in 0.5 second increments.	10
Flowchart	10
The Code	11
Write a Program so that when you press one of the buttons, it displays “1” on the 7-segment- display, then when you press the other button, it displays “2”	16
Flowchart	16
The Code	17
Modify your program further, so one button causes all of the LED segments to light, and the other switch turns them off. (the switch activations should be 'latching' i.e. you don't have to hold the button down for the on/off effect to remain).....	20
Flowchart	20
The Code	21
Combine everything together so you have a program that starts by displaying '0' but you can count up if you press one button, and count down if you press the other.	23
Flowchart	23
The Code	24
Conclusion.....	27
References	27

Introduction

In this project, we explore programming for the PIC16F84 microcontroller using assembly language. The objective is to create a program that interfaces with external hardware, specifically a seven-segment display and push buttons, to implement basic counting functionality. The program allows users to increment or decrement a count displayed on the seven-segment display using two push buttons. We will delve into the intricacies of debouncing buttons, managing count values, and controlling the display output.

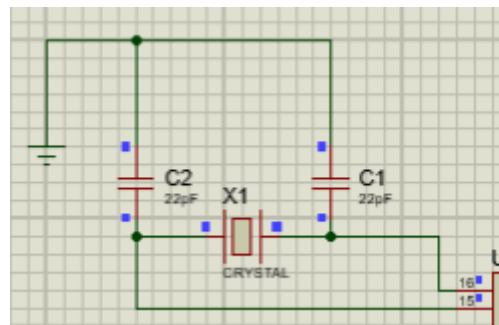
Section 1

Proteus Simulation



General Overview of Circuit Components

Crystal Oscillator



A crystal oscillator is an electronic oscillator circuit that uses the mechanical resonance of a vibrating crystal of piezoelectric material to create an electrical signal with a precise frequency. Here's how it works:

Crystal Resonance: The heart of a crystal oscillator is a small piece of quartz crystal. When an electric field is applied across the crystal, it mechanically vibrates at a specific frequency determined by its size and shape. This property is known as piezoelectricity.

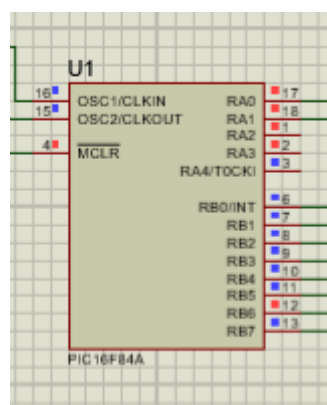
Feedback Loop: The crystal is connected to an electronic circuit that provides feedback. The feedback loop compares the frequency of the crystal's vibrations to the desired output frequency and adjusts the voltage or current to keep the crystal oscillating at the desired frequency.

Stability and Precision: Crystals have very stable resonant frequencies, meaning they oscillate at almost exactly-the-same frequency over time and under varying conditions such as temperature changes. This stability and precision make crystal oscillators ideal for applications where accurate timing is crucial, such as in clocks, radios, computers, and communication systems.

Output Signal: The output of a crystal oscillator is a clean and stable sine wave with a frequency determined by the crystal's resonant frequency. This signal can then be used as a reference for timing various operations in electronic devices.

Overall, crystal oscillators provide highly accurate and stable timing signals, making them essential components in a wide range of electronic devices and systems. [1]

PIC16F84A Chip



The PIC16F84A is a popular microcontroller chip developed by Microchip Technology. It belongs to the PIC (Peripheral Interface Controller) family of microcontrollers.

To function, the PIC16F84A chip requires several key components and resources:

Power Supply: The chip needs a stable power supply within its specified voltage range. Typically, it operates at voltages ranging from 2.0V to 5.5V, depending on the specific model and configuration. This power can be provided by batteries, a voltage regulator, or an external power supply.

Clock Source: The microcontroller requires a clock source to synchronize its internal operations. This clock determines the speed at which the CPU executes instructions and the timing of various peripheral operations.

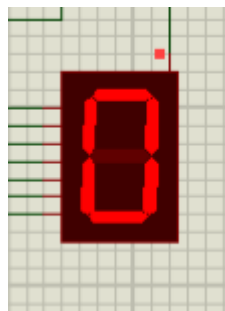
Program Memory: The chip needs program memory to store the user's code, which it executes to perform specific tasks. The PIC16F84A has internal flash memory where the user's program is stored.

Data Memory: The microcontroller requires data memory to store variables, intermediate results, and other temporary data during program execution. The PIC16F84A has internal RAM (Random Access Memory) for this purpose.

Input/Output (I/O) Pins: The chip needs input/output pins to interface with external devices, sensors, actuators, and other peripherals. These pins allow the microcontroller to send and receive digital signals, control external components, and communicate with other devices.

Supporting Components: Depending on the specific application and configuration, the PIC16F84A may require additional supporting components such as capacitors, resistors, voltage regulators, and decoupling capacitors to ensure proper operation and stability. [2]

7-Segment Display



A 7-segment display is a type of electronic display device commonly used to represent numerals. It consists of seven individually illuminated segments arranged in a rectangular pattern, as well as an additional eighth segment known as the decimal point (or dot). Each segment can be individually controlled to display different numbers and some letters.

Operating a 7-segment display using a microcontroller involves the following steps:

Connections: Connect the 7-segment display to the microcontroller. Typically, each segment of the display is connected to a GPIO (General Purpose Input/Output) pin on the microcontroller, and a common cathode or common anode pin is connected to a power supply (GND for common cathode or VCC for common anode).

Code Development: Write the code to control the display. This involves defining which segments correspond to each digit (0-9) and possibly other characters (such as letters). You'll also need to write functions to convert numerical values into the appropriate segments to display them.

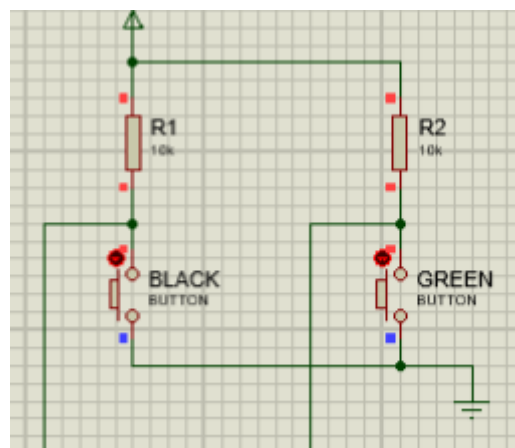
Segment Control: In your code, set the appropriate GPIO pins to control which segments should be illuminated for each digit. This is typically done by setting the pins to HIGH (for common cathode displays) or LOW (for common anode displays) to turn on the corresponding segments.

Digit Selection: If using multiplexing, use additional GPIO pins to control which digit is currently being displayed. Activate one digit at a time while simultaneously controlling the segments to display the desired number on that digit.

Compile and Upload: Compile your code and upload it to the microcontroller. Use a suitable programming tool or interface to transfer the compiled code to the microcontroller's memory.

Testing and Debugging: Test your code to ensure that the 7-segment display behaves as expected. Debug any issues by checking your connections, code logic, and timing.[3]

Buttons



Buttons play a crucial role in microcontroller circuits, offering a means for users to interact with the system. Here's how buttons affect a microcontroller circuit and how they are commonly used:

Input Devices: Buttons are primarily used as input devices in microcontroller circuits. When a button is pressed or released, it changes its electrical state, which the microcontroller can detect and respond to accordingly.

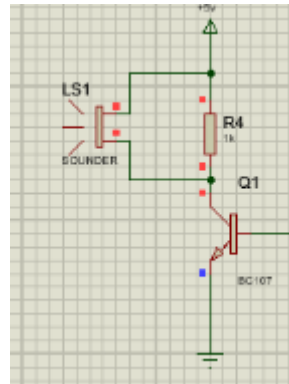
Digital Inputs: Most buttons are connected to digital input pins on the microcontroller. When the button is pressed, it typically connects the digital input pin to either VCC (for a pull-up configuration) or GND (for a pull-down configuration), changing the logic level detected by the microcontroller.

Pull-Up and Pull-Down Resistors: To ensure reliable operation, buttons are often used with pull-up or pull-down resistors. A pull-up resistor connects the input pin to VCC, while a pull-down resistor connects it to GND. These resistors ensure that the input pin has a defined logic level when the button is not pressed, preventing it from floating.

Debouncing: Mechanical buttons can produce bouncing effects when pressed, causing multiple rapid transitions between pressed and released states. To mitigate this issue, software debouncing techniques are often employed. This involves implementing a delay or a state machine in software to filter out these rapid transitions and provide a stable input signal to the microcontroller.

User Interface: Buttons provide a simple and intuitive means for users to interact with microcontroller-based systems. They can be used for various functions such as menu navigation, parameter adjustment, mode selection, or triggering actions. [4]

Sounder



A sounder, also known as a buzzer or piezo buzzer, is a common component used in microcontroller circuits to produce audible alerts or tones. Here's how a sounder is typically used in a microcontroller circuit:

Output Device: A sounder serves as an output device in the circuit. It converts electrical signals generated by the microcontroller into audible sound waves.

Audible Alerts: One of the primary uses of a sounder is to provide audible alerts or notifications to users. For example, in an alarm system, a sounder can emit a loud tone to alert users of an emergency or security breach.

Feedback Indicator: Sounders can also be used as feedback indicators to confirm user inputs or system actions. For instance, in a keypad interface, a sounder can emit a short beep each time a key is pressed, providing feedback to the user that the input has been registered.

Warning Signals: In industrial or automotive applications, sounders are often used to provide warning signals for various conditions such as low battery, high temperature, or equipment malfunction.

Tone Generation: Sounders can produce different tones or frequencies depending on the input signal from the microcontroller. By varying the frequency and duration of the signals sent to the sounder, different audible effects can be achieved, such as melodies, beeps, or sirens.

Simple Implementation: Sounders are relatively simple to integrate into microcontroller circuits. They typically require only a single digital output pin to control, making them suitable for a wide range of applications.

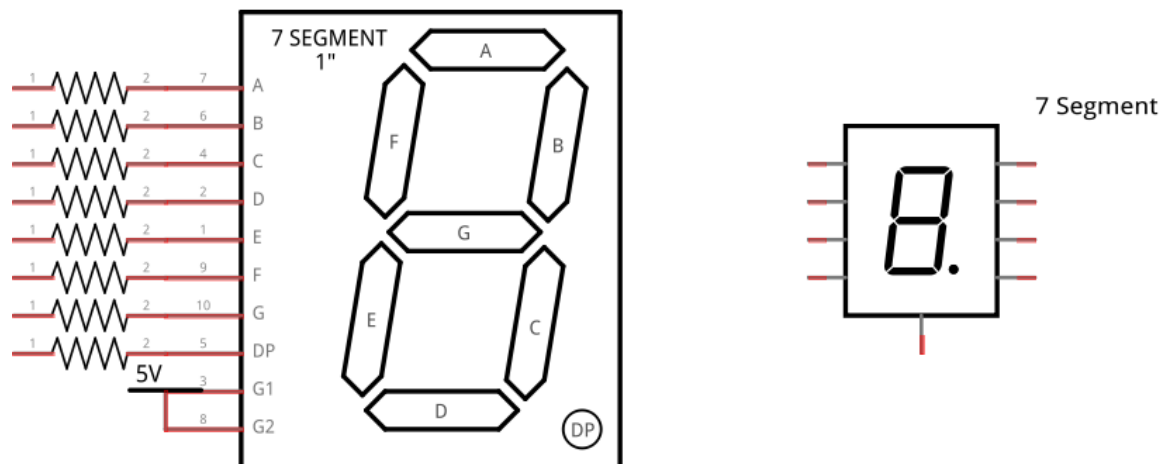
Low Cost and Compact Size: Sounders are inexpensive and compact, making them suitable for use in cost-sensitive and space-constrained designs. [5]

Section 2

The main aim of the circuit is to display from 0-9 and A-F on the 7-segment display. To achieve this a number of steps need to be followed:

Segment Mapping

Each segment will correspond to a specific bit in a binary number which will represent one segment of the 7-segment display:



[6]

Binary Values	8	4	2	1	8	4	2	1	Added Value of 0+1
Character	Souder	G	F	E	D	C	B	A	Hex
0	0	1	0	0	0	0	0	0	0X40
1	0	1	1	1	1	0	0	1	0X79
2	0	0	1	0	0	1	0	0	0X24
3	0	0	1	1	0	0	0	0	0X30
4	0	0	0	1	0	0	1	1	0X19
5	0	0	0	1	0	0	1	0	0X12
6	0	0	0	0	0	0	1	0	0X02
7	0	1	1	1	1	0	0	0	0X78
8	0	0	0	0	0	0	0	0	0X00
9	0	0	0	1	1	0	0	0	0X18
A	0	0	0	0	1	0	0	0	0X08
B	0	0	0	0	0	0	1	1	0X03
C	0	1	0	0	0	1	1	0	0X46
D	0	0	1	0	0	0	0	1	0X21
E	0	0	0	0	0	1	1	0	0X06
F	0	0	0	0	1	1	1	0	0X14

Look-up Table

A look-up table needs to be created that maps each hexadecimal digit (0-F) to it's corresponding binary value for the segments. Here's what it looks like:

```
movlw 0x40 ; char 0 on 7seg display
movwf 0x10 ; store in address 0x10 - start of look-up table

movlw 0x79 ; char 1
movwf 0x11

movlw 0x24 ; char 2
movwf 0x12

movlw 0x30 ; char 3
movwf 0x13

movlw 0x19 ; char 4
movwf 0x14

movlw 0x12 ; char 5
movwf 0x15

movlw 0x02 ; char 6
movwf 0x16

movlw 0x78 ; char 7
movwf 0x17

movlw 0x00 ; char 8
movwf 0x18

movlw 0x18 ; char 9
movwf 0x19

movlw 0x08 ; char A
movwf 0x1A

movlw 0x03 ; char B
movwf 0x1B
movlw 0x46 ; char C
movwf 0x1C
movlw 0x21 ; char D
movwf 0x1D
movlw 0x06 ; char E
movwf 0x1E
movlw 0x0E ; char F
movwf 0x1F
```

Flowchart

A flowchart or pseudo-code will be written prior to programming, this could be for several reasons:

Planning and Organization: Flowcharts and pseudocode help in organizing thoughts and planning the overall structure of the program. They allow the programmer to visualize the logical flow of the program before diving into the details of coding.

Clarifying Logic: They serve as a tool for clarifying the logic of the program. By breaking down the problem into smaller steps and representing them visually or in plain language, it becomes easier to understand the sequence of operations and decision points.

Identifying Errors and Edge Cases: Flowcharts and pseudocode allow the programmer to identify potential errors and edge cases in the program logic before writing any code. This can save time and effort during the debugging phase.

Collaboration and Communication: They facilitate communication and collaboration among team members. Flowcharts and pseudocode provide a common language for discussing the program's logic and functionality, even if team members have different programming backgrounds.

Ease of Modification: It is often easier to modify and refine the program logic in a flowchart or pseudocode form rather than in actual code. Once the high-level logic is finalized and validated, it can be translated into assembly language or any other programming language.

Documentation: Flowcharts and pseudocode serve as valuable documentation for the program. They provide insights into the program's design and logic, which can be useful for future reference or for explaining the program to others.

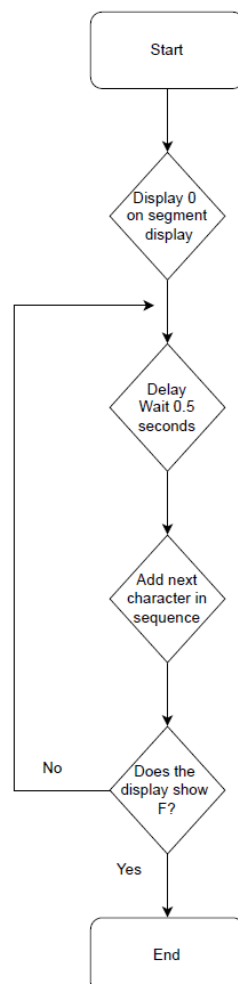
Microcontroller Code

Write code in the microcontroller to display characters 0-F and display them on the 7-segment display using the look-up table. The corresponding binary value to the pins connected to the segments of the display.

Section 3

Modify "Count.asm" so that the seven segment display counts from 0 to F in hexadecimal numbers in 0.5 second increments.

Flowchart



The Code

```

1
2 list p=16F84          ; select processor type
3
4 radix hex             ; define radix as hexadecimal
5
6 ;   #include "p16f84.inc"
7
8 ;-----
9
10 ;   cpu equates (memory map)
11
12 portB equ 0x06        ;port B address
13 trisB equ 0x06        ;trisB address in register bank 1
14 status equ 0x03       ;status register
15 RP0 equ 5             ;bit 5 of status register
16 RP1 equ 6             ;bit 6 of status register
17
18 count equ 0x0c        ; counter 0 to 255
19 nCount equ 0x0d       ; pause routine inner loop counter
20 mCount equ 0x0e       ; pause routine outer loop counter
21
22 ;-----
23
24 org 0x000
25
26 start
27 bcf status,RP0        ; clear bank selection bits in the status reg
28 bcf status,RP1
29 bsf status,RP0        ; select bank 1
30
31 movlw 0x00            ; load w with 0x00
32 movwf trisB           ; load it into trisb to make portb all outputs
33
34 bcf status,RP0        ; reselect bank 0
35 clrf portB            ; clear all lines low
36 clrf count            ; initialize counter to 0
37
38 get_count              ;movf count, w ; move count to W
39
40 movlw 0x40            ;char 0 for display
41 movwf portB           ; move W to port B to update the display
42
43 addlw 1               ; add +1 to W
44
45 movwf count           ; save W in count
46
47 call pause            ; delay 200ms by subroutine

```

```
48 call pause
49 call pause           ; five pause executions equals ~ 1 second
50
51
52
53 movlw 0x79           ;char 1 for display
54 movwf portB          ; move W to port B to update the display
55 addlw 1              ; add +1 to W
56 movwf count          ; save W in count
57
58 call pause           ; delay 200ms by subroutine
59 call pause
60 call pause           ; five pause executions equals ~ 1 second
61
62 movlw 0x24           ;char 2 for display
63 movwf portB          ; move W to port B to update the display
64 addlw 1              ; add +1 to W
65 movwf count          ; save W in count
66
67 call pause           ; delay 200ms by subroutine
68 call pause
69 call pause           ; five pause executions equals ~ 1 second
70
71 movlw 0x30           ;char 3 for display
72 movwf portB          ; move W to port B to update the display
73 addlw 1              ; add +1 to W
74 movwf count          ; save W in count
75
76 call pause           ; delay 200ms by subroutine
77 call pause
78 call pause           ; five pause executions equals ~ 1 second
79
80
81
82 movlw 0x19           ;char 4 for display
83 movwf portB          ; move W to port B to update the display
84 addlw 1              ; add +1 to W
85 movwf count          ; save W in count
86
87 call pause           ; delay 200ms by subroutine
88 call pause
89 call pause           ; five pause executions equals ~ 1 second
90
91 movlw 0x12           ;char 5 for display
92 movwf portB          ; move W to port B to update the display
```

```
93      addlw 1           ; add +1 to W
94      movwf count      ; save W in count
95
96      call pause        ; delay 200ms by subroutine
97      call pause
98      call pause        ; five pause executions equals ~ 1 second
99
100
101
102      movlw 0x02        ;char 6 for display
103      movwf portB       ; move W to port B to update the display
104      addlw 1           ; add +1 to W
105      movwf count       ; save W in count
106
107      call pause        ; delay 200ms by subroutine
108      call pause
109      call pause        ; five pause executions equals ~ 1 second
110
111
112
113      movlw 0x78        ;char 7 for display
114      movwf portB       ; move W to port B to update the display
115      addlw 1           ; add +1 to W
116      movwf count       ; save W in count
117
118      call pause        ; delay 200ms by subroutine
119      call pause
120      call pause        ; five pause executions equals ~ 1 second
121
122
123
124      movlw 0x00        ;char 8 for display
125      movwf portB       ; move W to port B to update the display
126      addlw 1           ; add +1 to W
127      movwf count       ; save W in count
128
129      call pause        ; delay 200ms by subroutine
130      call pause
131      call pause        ; five pause executions equals ~ 1 second
132
133      movlw 0x18        ;char 9 for display
134      movwf portB       ; move W to port B to update the display
135      addlw 1           ; add +1 to W
136      movwf count       ; save W in count
```

```
138 call pause ; delay 200ms by subroutine
139 call pause
140 call pause ; five pause executions equals ~ 1 second
141
142 movlw 0x08 ;char A for display
143 movwf portB ; move W to port B to update the display
144 addlw 1 ; add +1 to W
145 movwf count ; save W in count
146
147 call pause ; delay 200ms by subroutine
148 call pause
149 call pause ; five pause executions equals ~ 1 second
150
151 movlw 0x03 ;char b for display
152 movwf portB ; move W to port B to update the display
153 addlw 1 ; add +1 to W
154 movwf count ; save W in count
155
156 call pause ; delay 200ms by subroutine
157 call pause
158 call pause ; five pause executions equals ~ 1 second
159
160 movlw 0x46 ;char C for display
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 call pause
167 call pause ; five pause executions equals ~ 1 second
168
169 movlw 0x21 ;char d for display
170 movwf portB ; move W to port B to update the display
171 addlw 1 ; add +1 to W
172 movwf count ; save W in count
173
174 call pause ; delay 200ms by subroutine
175 call pause
176 call pause ; five pause executions equals ~ 1 second
177
178 movlw 0x06 ;char E for display
179 movwf portB ; move W to port B to update the display
180 addlw 1 ; add +1 to W
181 movwf count ; save W in count
182
```

```

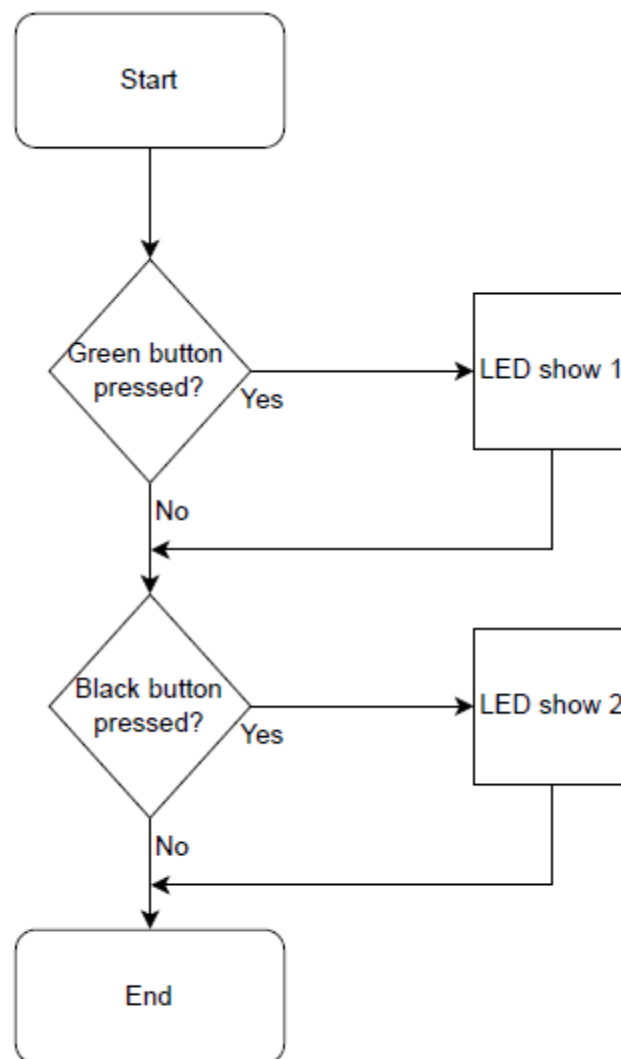
183 call pause ; delay 200ms by subroutine
184 call pause
185 call pause ; five pause executions equals ~ 1 second
186
187 movlw 0x0E ; char F for display
188 movwf portB ; move W to port B to update the display
189 addlw 1 ; add +1 to W
190 movwf count ; save W in count
191
192 call pause ; delay 200ms by subroutine
193 call pause
194 call pause ; five pause executions equals ~ 1 second
195
196 goto get_count ; repeat forever
197
198 ;-----
199
200 pause
201
202 movlw 0xff ; set w = 255 decimal
203 movwf mCount ; mCount = w: outer loop counter
204
205 loadN
206 movlw 0xff ; set w = 255 decimal
207 movwf nCount ; nCount = w: inner loop counter
208
209 decN
210 decfsz nCount, f ; nCount--
211 goto decN ; if nCount != 0 then repeat nCount--
212 decfsz mCount, f ; else decrement Count
213 goto loadN ; if mCount != 0 then
214
215 ; reload nCount to 255 and decrement
216
217 return ; else exit subroutine
218
219
220
221
222
223 end
224
225 ;-----

```

A subroutine was added into the code which pauses the count for approximately 0.5 seconds. After each segment is displayed, a pause is called.

Write a Program so that when you press one of the buttons, it displays “1” on the 7-segment-display, then when you press the other button, it displays “2”.

Flowchart



The Code

```

8
9      list    p=16F84          ; select processor type
10     radix   hex              ; define radix as hexadecimal
11     ;      #include "p16f84.inc"
12     ;-----
13
14     indr     equ 0x00
15     FSR      equ 0x04
16
17     portA    equ 0x05          ;port A address
18     trisa    equ 0x05          ;trisa in adress register bank 1
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     count    equ 0x0c          ; counter 0 to 255
28     nCount   equ 0x0d          ; pause routine inner loop counter
29     mCount   equ 0x0e          ; pause routine outer loop counter
30
31     ;-----
32
33     org      0x000
34
35     start
36         bcf    status,RP0      ; clear bank selection bits in the status reg
37         bcf    status,RP1
38
39         bsf    status,RP0      ; select bank 1
40         ; Set portA as inputs
41
42         movlw  0x03            ; load w with 0x03 - 2 LSBs set
43         movwf  trisa           ;set 2MSB = port A as inputs (switches)
44
45         movlw  0x00            ; load w with 0x00
46         movwf  trisb           ; load it into trisb to make portb all outputs

```

```

48      bcf      status,RP0      ; reselect bank 0
49
50      clrf     portB           ; clear all lines low
51      clrf     count           ; initialize counter to 0
52
53      movlw    0x79
54      movwf    0x10
55
56      movlw    0x24
57      movwf    0x11
58  main_loop
59
60      btfss    portA,0         ; test black button
61      call     black_button_pressed
62
63      btfss    portA,1         ; test green button
64      call     green_button_pressed
65
66      movf     count,w         ; get count value
67      addlw    0x10            ; add offset for start of look-up table
68      movwf    FSR             ; set address for look-up table
69      movf     indr,w          ; get value from look-up table
70      movwf    portB           ; set LED display
71
72      goto     main_loop       ; repeat forever
73
74  black_button_pressed
75      call     pause           ; wait 20ms for switch to debounce
76      btfss    portA,0
77      goto     black_button_pressed
78      incf     count,f         ; increment count
79      btfsc    count,4         ; test to see if we need to reset count
80      clrf     count           ; reset count
81      return;
82
83
84  green_button_pressed
85      call     pause           ; wait 20ms for switch to debounce
86      btfss    portA,1
87      goto     green_button_pressed
88      decf     count,f         ; decrement count
89      btfsc    count,4         ; test to see if we've gone negative
90      call     preset_count    ;
91      return

```

```

92
93     preset_count
94     movlw    0x0F
95     movwf    count
96     return
97
98 pause
99     movlw    0xff          ; set w = 255 decimal
100    movwf    mCount        ; mCount = w: outer loop counter
101 loadN
102    movlw    0xff          ; set w = 255 decimal
103    movwf    nCount        ; nCount = w: inner loop counter
104 decN
105    decfsz   nCount, f      ; nCount--
106    goto     decN          ; if nCount != 0 then repeat nCount--
107
108    decfsz   mCount, f      ; else decrement Count
109    goto     loadN         ; if mCount != 0 then
110                                ; reload nCount to 255 and decrement
111    return                ; else exit subroutine
112
113
114     end
115

```

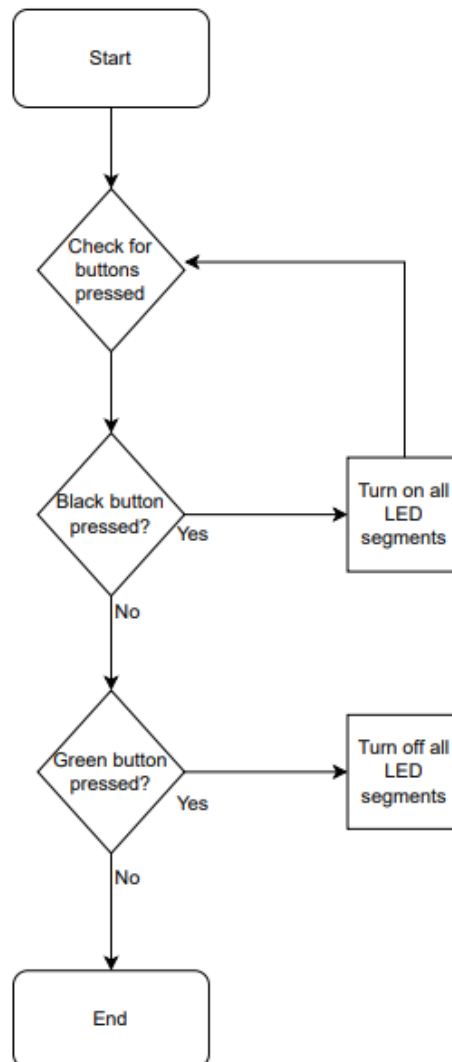
To add two buttons that allow increasing and decreasing the count in the program, we need to implement button detection routines and handle count incrementing and decrementing accordingly.

- Add button detection subroutines for detecting button presses.
- Call these subroutines in the main loop to check for button presses.
- Implement count incrementing and decrementing logic based on button presses.

Black_button_pressed and green_button_pressed represents the buttons connected to the inputs. Inside the main_loop, we check for button presses using btfss instructions. If a button is pressed, we call the appropriate subroutine (incf or decf). The incf subroutine increases the count by 1 and decf decreases the count by 1.

Modify your program further, so one button causes all of the LED segments to light, and the other switch turns them off. (the switch activations should be 'latching' i.e. you don't have to hold the button down for the on/off effect to remain)

Flowchart



The Code

```

 9      list    p=16F84          ; select processor type
10      radix   hex              ; define radix as hexadecimal
11      ;      #include "p16f84.inc"
12      ;-----
13      ;      cpu equates (memory map)
14
15      indr     equ 0x00
16      FSR      equ 0x04
17
18      portA    equ 0x05          ;port A address
19      trisa     equ 0x05          ;trisa in address register bank 1
20
21      portB    equ 0x06          ;port B address
22      trisb     equ 0x06          ;trisB address in register bank 1
23
24      status   equ 0x03          ;status register
25      RP0      equ 5             ;bit 5 of status register
26      RP1      equ 6             ;bit 6 of status register
27
28      count    equ 0x0c          ; counter 0 to 255
29      nCount   equ 0x0d          ; pause routine inner loop counter
30      mCount   equ 0x0e          ; pause routine outer loop counter
31
32      ;-----
33
34      org      0x000
35
36      start
37          bcf    status,RP0      ; clear bank selection bits in the status reg
38          bcf    status,RP1
39
40          bsf    status,RP0      ; select bank 1
41          ; Set portA as inputs
42
43          movlw  0x03            ; Load w with 0x03 - 2 LSBs set
44          movwf  trisa           ;set 2MSB = port A as inputs (switches)
45
46          movlw  0x00            ; Load w with 0x00
47          movwf  trisb           ; Load it into trisb to make portb all outputs
48
49          bcf    status,RP0      ; reselect bank 0
50

```

```

51      clrf    portB           ; clear all lines low
52      clrf    count          ; initialize counter to 0
53
54      movlw   0x00
55      movwf   0x10
56
57      movlw   0xff
58      movwf   0x11
59  main_loop
60
61      btfss   portA,0         ; test black button
62      call    black_button_pressed
63
64      btfss   portA,1         ; test green button
65      call    green_button_pressed
66
67      movf    count,w         ; get count value
68      addlw   0x10            ; add offset for start of look-up table
69      movwf   FSR             ; set address for look-up table
70      movf    indr,w          ; get value from look-up table
71      movwf   portB           ; set LED display
72
73      goto    main_loop       ; repeat forever
74
75  black_button_pressed
76  call    pause               ; wait 20ms for switch to debounce
77  btfss   portA,0
78  goto    black_button_pressed
79  incf     count,f            ; increment count
80  btfsc    count,4            ; test to see if we need to reset count
81  clrf     count              ; reset count
82  return;
83
84
85  green_button_pressed
86  call    pause               ; wait 20ms for switch to debounce
87  btfss   portA,1
88  goto    green_button_pressed
89  decf     count,f            ; decrement count
90  btfsc    count,4            ; test to see if we've gone negative
91  call    preset_count        ;
92  return
93
94  preset_count
95  movlw   0x0F
96  movwf   count
97  return

```

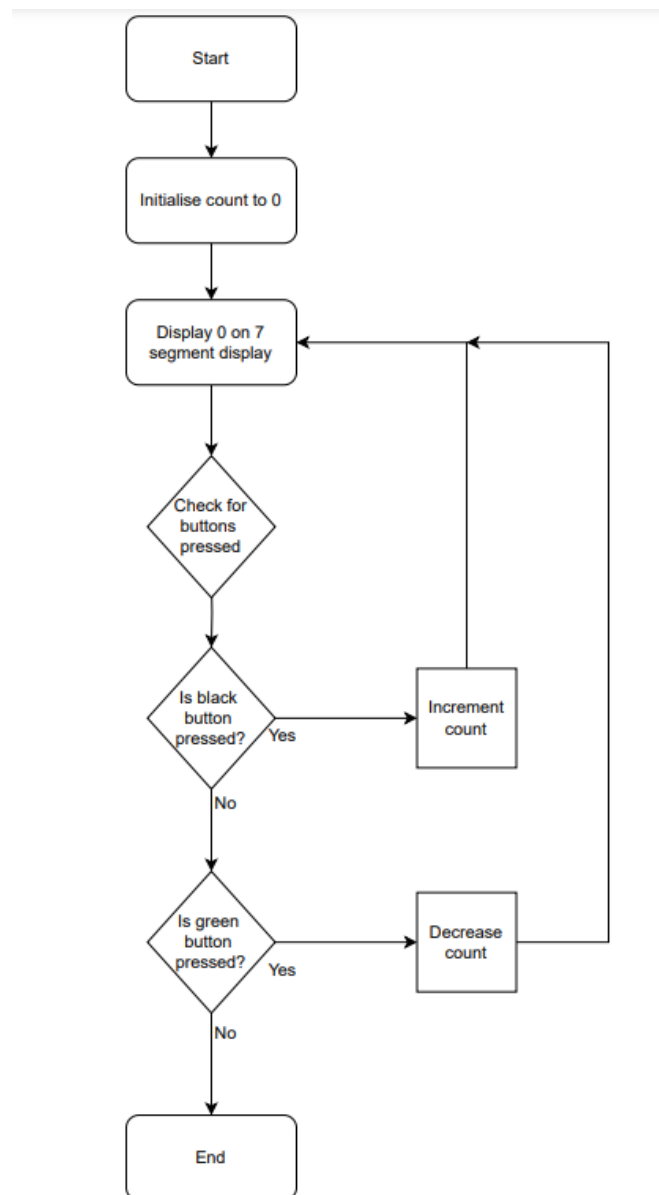
```

99  ; -----
100  pause
101      movlw   0xff            ; set w = 255 decimal
102      movwf   mCount          ; mCount = w: outer loop counter
103  loadN
104      movlw   0xff            ; set w = 255 decimal
105      movwf   nCount          ; nCount = w: inner loop counter
106  decN
107      decfsz  nCount, f        ; nCount--
108      goto    decN            ; if nCount != 0 then repeat nCount--
109
110      decfsz  mCount, f        ; else decrement Count
111      goto    loadN           ; if mCount != 0 then
112                               ; reload nCount to 255 and decrement
113  return                      ; else exit subroutine
114
115
116  end

```

Combine everything together so you have a program that starts by displaying '0' but you can count up if you press one button, and count down if you press the other.

Flowchart



The Code

```

 9      list    p=16F84          ; select processor type
10      radix   hex              ; define radix as hexadecimal
11      ;      #include "p16f84.inc"
12      ;-----
13      ;      cpu equates (memory map)
14
15      indr     equ 0x00
16      FSR      equ 0x04
17
18      portA    equ 0x05          ;port A address
19      trisa     equ 0x05          ;trisa in adress register bank 1
20
21      portB    equ 0x06          ;port B address
22      trisb     equ 0x06          ;trisB address in register bank 1
23
24      status    equ 0x03          ;status register
25      RP0       equ 5             ;bit 5 of status register
26      RP1       equ 6             ;bit 6 of status register
27
28      count     equ 0x0c          ; counter 0 to 255
29      nCount    equ 0x0d          ; pause routine inner loop counter
30      mCount    equ 0x0e          ; pause routine outer loop counter
31
32      ;-----
33
34      org      0x000
35
36      start
37          bcf      status,RP0      ; clear bank selection bits in the status reg
38          bcf      status,RP1
39
40          bsf      status,RP0      ; select bank 1
41          ; Set portA as inputs
42
43          movlw    0x03            ; load w with 0x03 - 2 LSBs set
44          movwf    trisa           ;make two LSBs of port A inputs (switches)
45
46          movlw    0x00            ; load w with 0x00
47          movwf    trisb           ; load it into trisb to make portb all outputs

```

```
49      bcf      status,RP0      ; reselect bank 0
50
51      clrf     portB           ; clear all lines low
52      clrf     count          ; initialize counter to 0
53
54      movlw    0x40            ; char 0 on 7seg display
55      movwf    0x10            ; store in address 0x10 - start of look-ip table
56
57      movlw    0x79            ; char 1
58      movwf    0x11
59
60      movlw    0x24            ; char 2
61      movwf    0x12
62
63      movlw    0x30            ; char 3
64      movwf    0x13
65
66      movlw    0x19            ; char 4
67      movwf    0x14
68
69      movlw    0x12            ; char 5
70      movwf    0x15
71
72      movlw    0x02            ; char 6
73      movwf    0x16
74
75      movlw    0x78            ; char 7
76      movwf    0x17
77
78      movlw    0x00            ; char 8
79      movwf    0x18
80
81      movlw    0x18            ; char 9
82      movwf    0x19
83
84      movlw    0x08            ; char A
85      movwf    0x1A
86
87      movlw    0x03            ; char B
88      movwf    0x1B
89      movlw    0x46            ; char C
90      movwf    0x1C
```

```
90      movwf 0x1C
91      movlw 0x21      ; char D
92      movwf 0x1D
93      movlw 0x06      ; char E
94      movwf 0x1E
95      movlw 0x0E      ; char F
96      movwf 0x1F
97
98
99 main_loop
100
101      btfss portA,0      ; test black button
102      call black_button_pressed
103
104      btfss portA,1      ; test green button
105      call green_button_pressed
106
107      movf count,w      ; get count value
108      addlw 0x10      ; add offset for start of look-up table
109      movwf FSR      ; set address for look-up table
110      movf indr,w      ; get value from look-up table
111      movwf portB      ; set LED display
112
113      goto main_loop      ; repeat forever
114
115 black_button_pressed
116      call pause      ; wait 20ms for switch to debounce
117      btfss portA,0
118      goto black_button_pressed
119      incf count,f      ; increment count
120      btfsc count,4      ; test to see if we need to reset count
121      clrf count      ; reset count
122      return;
123
124
125 green_button_pressed
126      call pause      ; wait 20ms for switch to debounce
127      btfss portA,1
128      goto green_button_pressed
129      decf count,f      ; decrement count
130      btfsc count,4      ; test to see if we've gone negative
131      call preset_count      ;
132      return
133
```

```

134      preset_count
135      movlw    0x0F
136      movwf    count
137      return
138
139      ;-----
140      pause
141      movlw    0x19      ; set w = 255 decimal
142      movwf    mCount    ; mCount = w: outer loop counter
143      loadN
144      movlw    0xff      ; set w = 255 decimal
145      movwf    nCount    ; nCount = w: inner loop counter
146      decN
147      decfsz   nCount, f  ; nCount--
148      goto     decN      ; if nCount != 0 then repeat nCount--
149
150      decfsz   mCount, f  ; else decrement Count
151      goto     loadN      ; if mCount != 0 then
152                        ; reload nCount to 255 and decrement
153      return      ; else exit subroutine
154
155
156      end
157

```

Conclusion

In conclusion, this programming endeavor has provided a hands-on experience in developing embedded systems software using assembly language for the PIC16F84 microcontroller. We successfully implemented a program that interfaces with external hardware components, including push buttons and a seven-segment display, to create a basic counting application. Throughout the project, we addressed challenges such as debouncing buttons to ensure reliable input, managing count values within specified ranges, and controlling the display output.

References

- [1] A. Schure, *Crystal Oscillators*. New York: J.F. Rider, 1955.
- [2] M. Predko, *Handbook of Microcontrollers*. New York: McGraw-Hill, 1999.
- [3] Working with seven segment displays - introduction to the basics, <https://www.jameco.com/Jameco/workshop/TechTip/working-with-seven-segment-displays.html> (accessed Mar. 26, 2024).
- [4] I/O is one of those concepts that can seem self-explanatory. But simple as they may seem, "What is I/O? understanding GPIO & input/output functions," Arrow.com, <https://www.arrow.com/en/research-and-events/articles/what-is-io-computer-and-electronic-input-output-explained> (accessed Mar. 26, 2024).
- [5] M. Bates, *Interfacing Pic Microcontrollers Embedded Design by Interactive Simulation*. Oxford: Newnes, 2006.
- [6] P. Shaw, "7 segment display," Digital Lab, <https://digitalab.org/2015/08/7-segment-display/> (accessed Mar. 26, 2024).