

James T Micros assignment

by James T

Submission date: 13-Jan-2025 04:13PM (UTC+0000)

Submission ID: 248291068

File name: 52211_James_T_James_T_Micros_assignment_621295_818922590.docx (814.42K)

Word count: 6160

Character count: 30686

Micros, Peripherals and interfaces

FOR ELECTRONICS

James T | Embedded Electronics | 05/01/2025

Contents

1. Introduction
2. Circuit Function
 - 2.1. Linear Regulator Input
 - 2.2. PIC16X84
 - 2.3. Timing Circuit
 - 2.4. Button inputs
 - 2.5. 7-segment display output
 - 2.6. Sounder output
3. Hexadecimal Characters
 - 3.1. PortB and display segments correlation
 - 3.2. Hexadecimal Codes 0-F
4. Assembly Language Programs
 - 4.1. Program 1
 - 4.2. Program 2
 - 4.3. Program 3
 - 4.4. Program 4
 - 4.5. Program 5
5. Conclusion
6. Bibliography
7. Appendices
 - 7.1. Count from 0-F on 7 segment display code (1)
 - 7.2. Buttons displaying L and R code (2)
 - 7.3. Buttons turning all LED's on and off code (3)
 - 7.4. Counter starting at 5 using buttons code (4)
 - 7.5. Piezo Sounder Code (5)

This report will cover the PIC16 circuit, each component's purpose within the circuit and how the PIC controller communicates with its inputs and outputs. It also covers how to work out Hexadecimal values from binary codes, along with how to get the binary codes for each pattern. Finally, it will show the code for different tasks that the PIC16F84A must complete and their matching flowcharts describing the logic.

The overall circuit that is used in this report is shown below in figure 1. It can be split into six different sections: the Linear Voltage regulator input, the PIC16X84, A timing circuit, the button inputs and the outputs



PAGE 2

Linear Regulator input

The first section of the circuit shown in figure 2 below is the Linear voltage regulator. The original circuit uses the 7805 Linear voltage regulator.

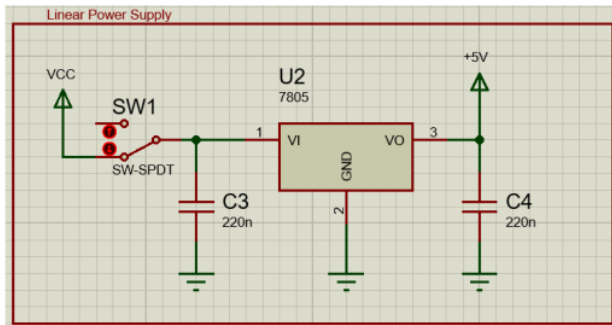


Figure 2 – Linear voltage regulator input

The original circuit from figure 1, [1]uses a single pole change over switch, like SPDT switch however with a stable off. In the figure above, an SPDT (single pole double throw) switch was used in its place. The switch is there to turn the circuit on and off.

[2]The switch then goes to the 7805 linear regulator, with an input and output capacitor. [3]The input capacitor is needed if the regulator is an appreciable distance away from a power supply filter. It acts as a filter to help reduce noise and ripple from the input power source. This capacitor's value depends on the regulator's input impedance and maximum input ripple voltage. [2]The output capacitor isn't needed but can help stability as well as improving the regulators transient response – [4]which is how fast it returns to steady-state conditions after a load change.

The 7805 Linear regulator provides a stable +5V supply to the rest of the circuit. [5]It is a three terminal IC with a fixed 5V output. It can deliver up to 1.5A, and has protection features such as: internal current limiting and thermal shutdown.

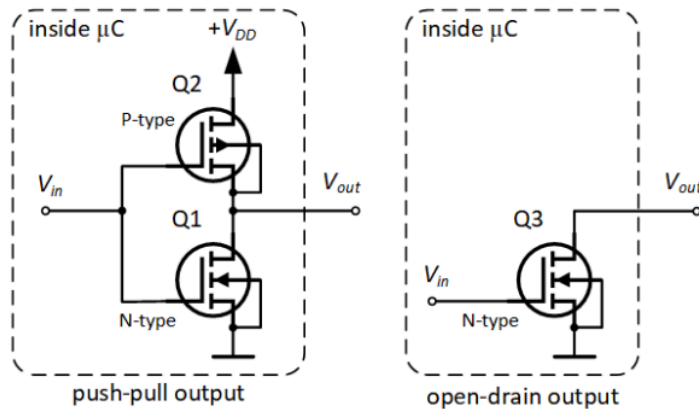
PIC16X84

[6]Within the PIC16X84 family, there are the PIC16F84 and PIC16C84. The original circuit uses the 16C84 and the proteus recreation uses the 16F84. The 16F84 is an enhanced version of the 16C84. This chip is a mid-range 8-bit microcontroller which can be interfaced with different inputs and outputs.

The PIC16X84 family architecture can be seen in figure 3 below.

[10]TTL is a logic family made up of BJT transistors meanwhile CMOS uses complementary and symmetrical p-type and n-type MOSFETs. [8]All I/O pins that are configured as inputs interface through the TTL input buffer (pins RA0-3 and RB0-7) or through the Schmitt trigger input buffer (pin RA4).

Since the input pins are TTL buffered, interfacing a TTL device or gate to the PIC can be done directly to the pin if it has one of the transistors pulling the output high. If the device has an open-collector/drain output – this will require an external pull-up resistor. An example of the differences can be seen below.

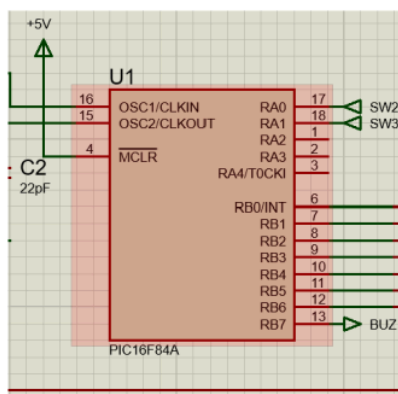


[u]Figure 4 – Open-collector/drain vs push-pull

[8]For interfacing output devices with the PIC, Pins RA0 to RA3 are full CMOS output drivers and RA4 is an open-drain output. RB0-RB7 are TTL buffered output drivers. The outputs have a maximum current source of 20mA per pin, 50mA for the entire PORTA and 100mA for PORTB. This is useful for powering the LEDs straight from the pins.

The PIC from the circuit can be seen in figure 6.

Figure 6 – PIC16F84



[7]The PIC microcontroller above shows the I/O pins mentioned previously with inputs and output devices connected, as well as the +5V to power the microcontroller

on $\overline{MCLR}$, which is a master clear (reset) input/programming voltage input. This pin is an active low, meaning that grounding this pin will restart the program stored in the EEPROM. It needs to be held high during normal program execution.

Timing circuit

The timing circuit used for clocking the PIC can be seen below.

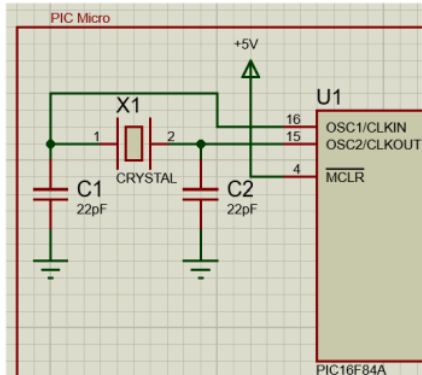


Figure 7 – timing circuit on the CLKIN/CLKOUT pins

The timing circuit is made up of a crystal oscillator and capacitors. [13] The capacitors determine the value of the crystal's load capacitance. One side of the oscillator goes to OSC1/CLKIN and the other side to OSC2/CLKOUT, which stands for oscillator 1/2 and Clock In/Out. These pins can also be seen in the PIC16X84 architecture diagram from figure 3. [14] This clock source will drive all the peripherals and instructions.

Button inputs

Next section are the button inputs on pins RA0 and RA1, which can be seen below in figure 8.

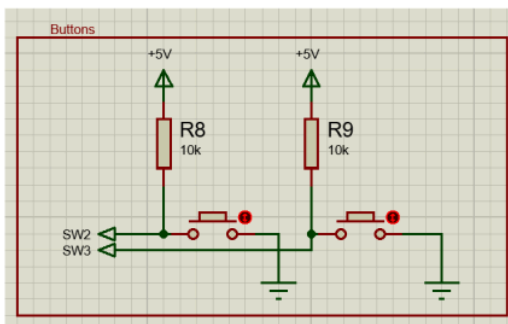


Figure 8 – Buttons circuit going to RA0 (SW2) and RA1 (SW3)

This is a simple circuit containing push buttons that go to ground and have pull up resistors to ensure that when the button is released, the input pin is high. When the button is pressed, it shorts the pin to ground, setting the pin low. Code will read it's been set to low and follow up with a set of instruction

7-segment display output

This section covers the 7 segment LED display and resistor network shown below.

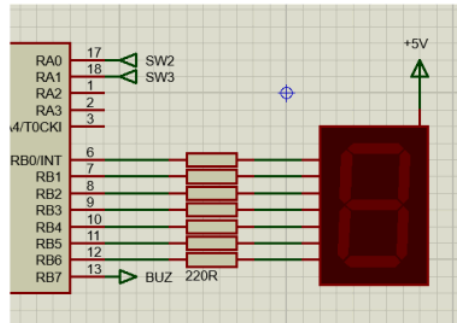
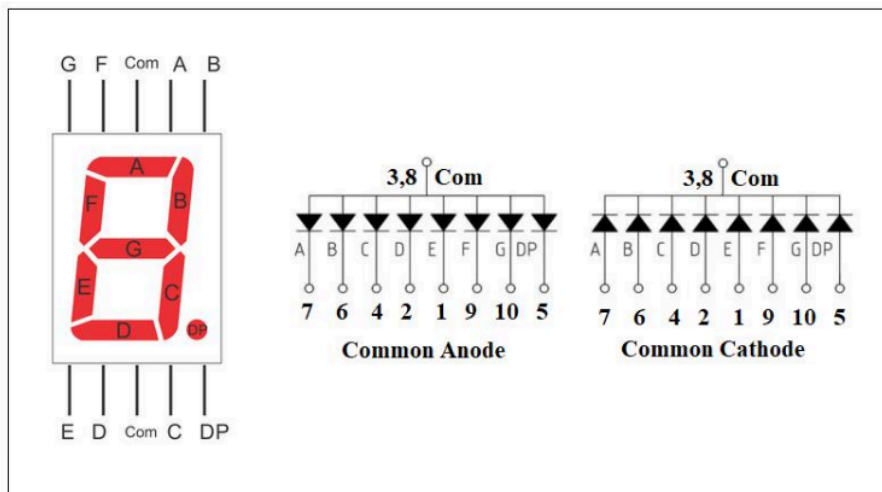


Figure 9 – 7-segment display output

PORTB of the PIC is set to outputs in this case, with RB7 as an output to the sounder. The rest of PORTB (RB0-6) output to the 7-segment display. In this circuit, there is a set of 220Ω resistors that function as current limiting resistors for the LEDs in the display. They drop a voltage across them and protect against voltage increases, which can be calculated using $R = \frac{V}{I}$ where V is voltage to be dropped ($V_{max} - V_{f\ of\ LED}$), I is the current the LED needs and R is the resistor value required. In this case, $V_{max} = 5V$ and [15]the forward voltage V_f of the LEDs can range but it is safe to assume it is around 2V. The current of the LED's are usually around 10-20mA, therefore a value resistor between 150Ω and 300Ω would work. It is important to note that the value should ideally not be less than this to protect the LED, but great can cause the LEDs to get dimmer since less voltage is across the LED.

The 7-segment display is an array of LEDs that have a common anode for each LED, with the anode going to the pins. This means that to activate each segment of the LED, the Pin corresponding to the segment must be set to low in the code and the pin must



nominally be logic level 1, otherwise the display will always be on. Figure 10 shows an example of common anode vs common cathode displays

[16]Figure 10 – 7-segment display example of common anode and cathode types

Sounder output

The sounder output makes use of a low side switch, which switches a high current device to ground using a low current – in this case the PIC controller's pin RB7 is the low current device switching the buzzer which requires more current than the PIC pin can supply. The circuit for this can be seen in figure 11.

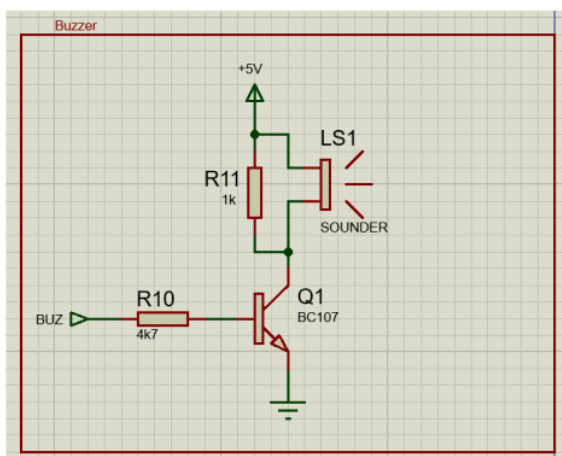


Figure 11 – sounder circuit

The BJT NPN transistor is used for the low side switch, which switches the sounder to low (ground). The resistor R10 is a current limiting resistor, which the BJT's base requires. This prevents too much current causing it to overheat. The pin RB7 switches between logic 0 and 1, with the length of the pause changing levels, changing the sound/pitch of the sounder

The second part of this circuit is the sounder with a parallel resistor. [17]The resistor is in place to return the sounder to an unenergized state when the BJT is switched off by charging and discharging the sounder. The sounder is always at high, then when the BJT switches, the lower pin is ground since the BJT allows the current to pass and the sounder makes a sound.

HEXADECIMAL CHARACTERS

PortB and display segments correlation

The first stage of working out the Hexadecimal characters for 0-F on the 7-segment display, is to work out which PORTB pin corresponds to which LED segment of the display. Since the display has a common anode, the pins need to be low (0) for the LED to be on and high (1) to turn the LED off.

Since the Binary to hex conversion is:

Binary	Hex
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	A
11	B
12	C
13	D
14	E
15	F

The Hex values were calculated by separating the 8-bit binary into 2 nibbles (4-bits) and writing 8,4,2,1 above the binary. For example:

8 4 2 1 8 4 2 1

0110 0001

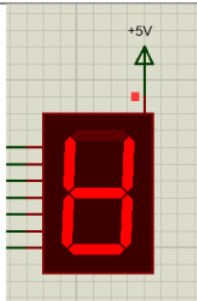
If the binary is 0, then to calculate hex it is a 0, If it is a 1, then whatever number is above the 1 is added in the calculation. For the example above, the sum would look like this:

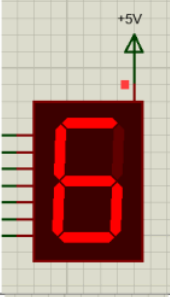
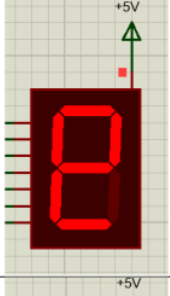
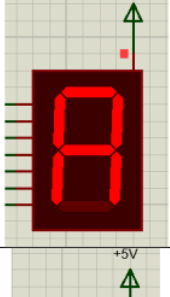
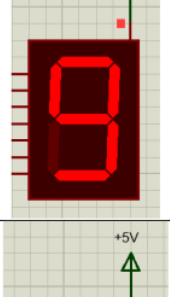
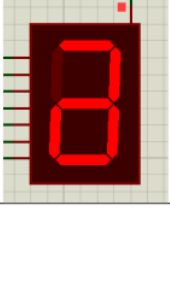
(0+4+2+0) (0+0+0+1)

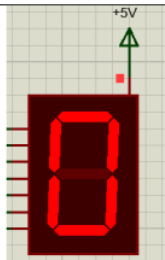
The first bracket represents the 3rd letter in the hex and the second bracket represents the 4th letter in the hex. In this case the first bracket equals 6 and the second bracket equals 1. Therefore, the hex value for this example comes out to be:

0x61

Using the method above, I ran through the following binary numbers to get the hex values to put into the PIC16:

Binary	Hex code	Picture of result
0000 0001	0x01	

0000 0010	0X02	
0000 0100	0X04	
0000 1000	0X08	
0001 0000	0X10	
0010 0000	0X20	

0100 0000	0x40	
-----------	------	--

This means that the Pins can be noted on what segment they relate too:

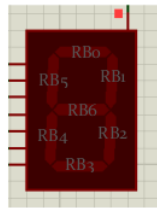
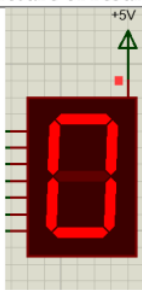
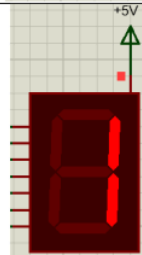
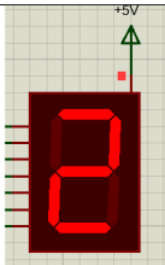
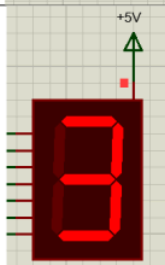
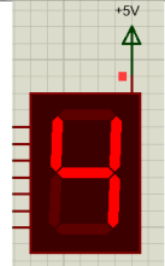
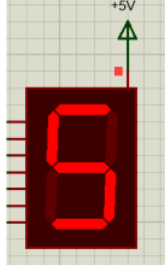
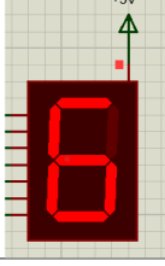


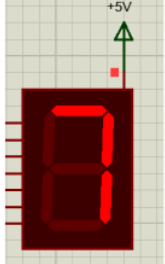
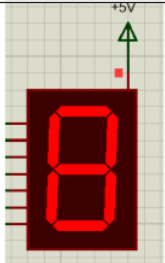
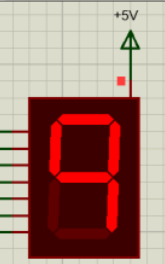
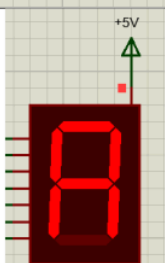
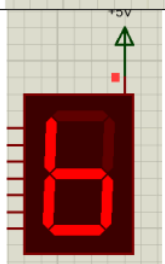
Figure 12 – Display pinout

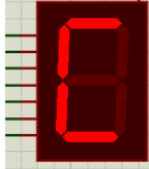
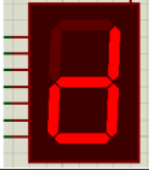
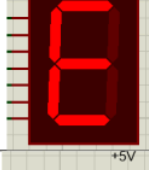
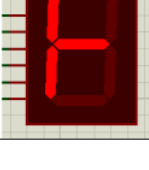
Hexadecimal codes 0-F

Now it is known what PORTB pins go to what LED in the display, it is possible to figure out what LEDs need to be on, or off, to create the patterns for 0-F. The Binary and hex codes can be seen below.

Number/letter displayed	Binary	Hex	Picture of result
0	0100 0000	0x40	
1	0111 1001	0x79	

2	0010 0100	0x24	
3	0011 0000	0x30	
4	0001 1001	0x19	
5	0001 0010	0x12	
6	0000 0010	0x02	

7	0111 1000	0x78	
8	0000 0000	0x00	
9	0001 1000	0x18	
A	0000 1000	0x08	
B	0000 0011	0x03	

C	0100 0110	0x46	
D	0010 0001	0x21	
E	0000 0110	0x06	
F	0000 1110	0x0E	

ASSEMBLY PROGRAMS

Program 1

Program 1 task is to start from 0 and count to F with 0.2 second increments. The first stage to write this code is to create a flow chart of what it does. This can be seen below.

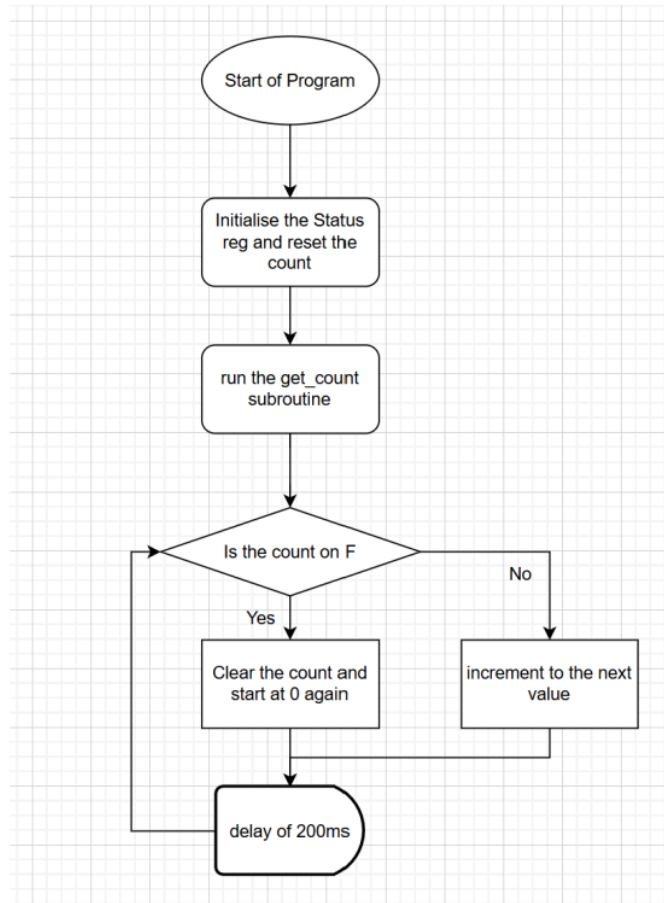


Figure 13 – Flowchart for counting 0-F

The code written using this flowchart can be seen in appendix 1.

Program 2

Program 2 uses buttons to display an L or R temporarily. Before writing this code, a flow chart was created to get describe the operation plan, which can be seen below

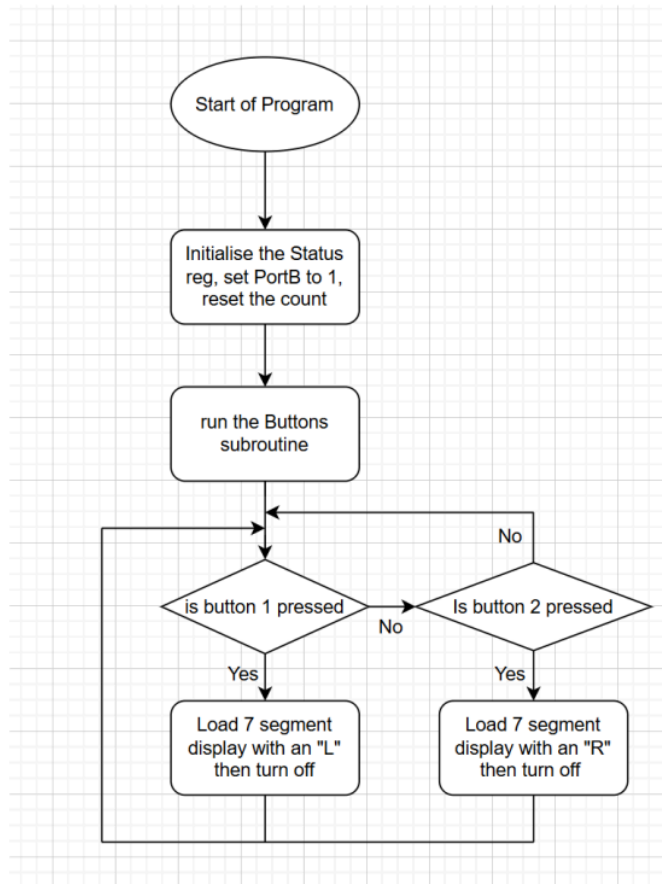


Figure 14 – Flowchart for displaying L and R

The code written for this can be seen in appendix 2.

Program 3

Program 3 uses a button to turn all LEDs of the display on and one to turn them all off. A flowchart was written for this to describe its logic and operation, which can be seen below

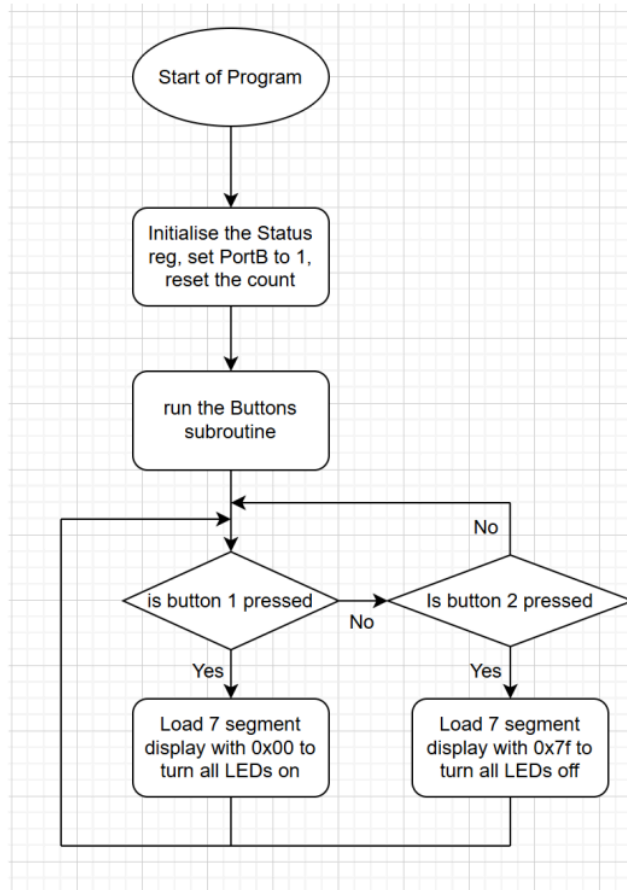


Figure 15 – Flowchart for turning all LEDs on and off on the display

The code for program 3 can be seen in appendix 3.

Program 4

Program 4 starts at 5 on the display and uses buttons to count up and down between 0 and 9. A flowchart was drawn to show its logic and operation and can be seen in figure 17

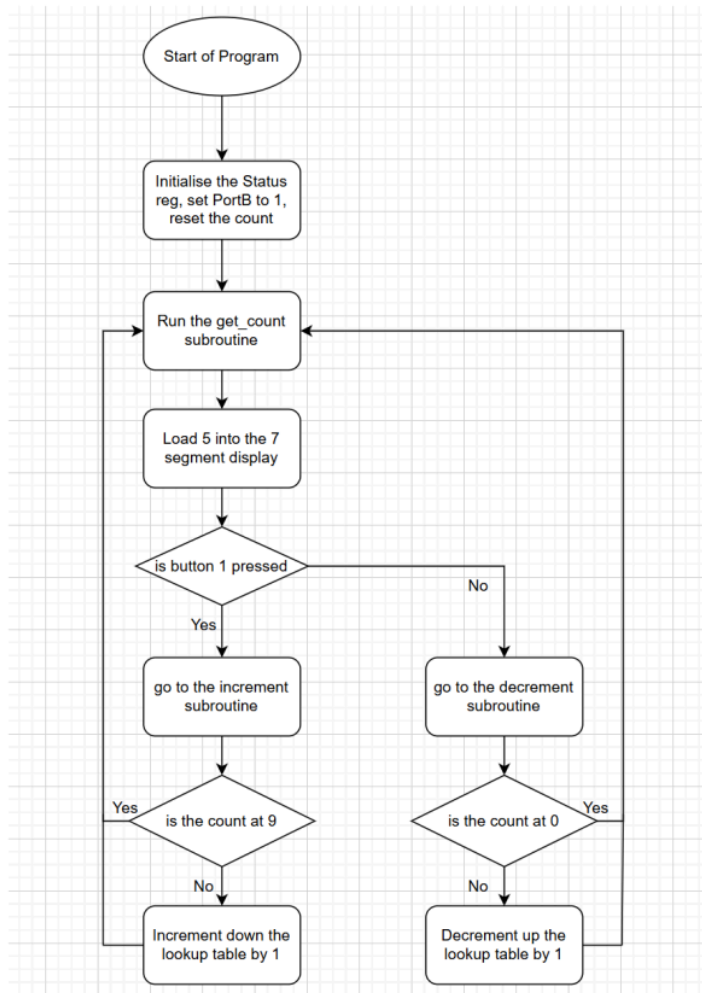


Figure 16 – Flowchart for counting between 0 and 9 starting at 5

The code to run this program can be seen in appendix 4.

Program 5

Program 5 uses a button to trigger a tone from the sounder. A flowchart was written to describe its operation and can be seen below. Changing the tone of the sounder is done by adjusting the nCount and mCount values in the Pause subroutine from the code in appendix 5.

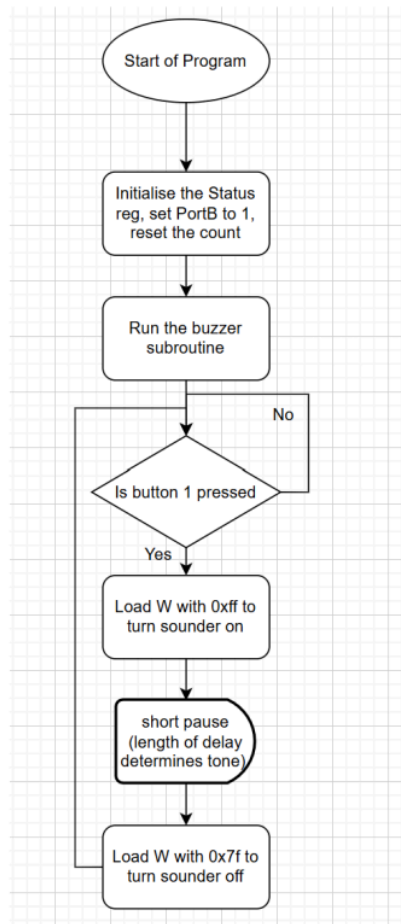


Figure 17 – Flowchart to activate the sounder

The code of this flowchart can be found in appendix 5.

CONCLUSION

To conclude, the circuit for this microcontroller is a simple circuit with a linear regulator for a constant +5V supply, a timing circuit for timing the PIC, buttons for inputs as well as a 7-segment display and a low side switch for the sounder as outputs. The Hex values were calculated from working out the binary codes for each pattern and the Assembly codes were written along with flowcharts for each task to describe its operation.

Bibliography

[histo.cat, "SPST - SPDT - SPCO - SPTT - DPST - DPDT - DPCO," histo.cat, [Online].
1 Available: <https://www.histo.cat/e/SPST-SPDT-SPCO-SPTT-DPST-DPDT-DPCO>.
] [Accessed 10 January 2025].

[Onsemi, "Voltage regulators - positive 1A MC7800, MC7800A, MC7800AE,
2 NCV7800," 2019. [Online]. Available:
] <https://www.farnell.com/datasheets/3743668.pdf>. [Accessed 10 January 2025].

[Microchip, "Input and Output Capacitors for LDOs," Microchip, 7 February 2024.
3 [Online]. Available: [https://microchip.my.site.com/s/article/Input-and-Output-](https://microchip.my.site.com/s/article/Input-and-Output-Capacitors-for-LDOs)
] [Capacitors-for-LDOs](https://microchip.my.site.com/s/article/Input-and-Output-Capacitors-for-LDOs). [Accessed 10 January 2025].

[Texas Instruments, "Linear and Switching Voltage Regulator Fundamental Part 1,"
4 2011. [Online]. Available: <https://www.ti.com/lit/an/snva558/snva558.pdf>. [Accessed
] 10 January 2025].

[R. Teja, "Understanding 7805 IC Voltage Regulator," ElectronicsHub, 25 March
5 2024. [Online]. Available: [https://www.electronicshub.org/understanding-7805-ic-](https://www.electronicshub.org/understanding-7805-ic-voltage-regulator/)
] [voltage-regulator/](https://www.electronicshub.org/understanding-7805-ic-voltage-regulator/). [Accessed 10 January 2025].

[J. Mulindi, "Key Features of PIC16X84 Family of Microcontrollers," 19 February 2024.
6 [Online]. Available: [https://microsystems.electricalandcontrol.com/key-features-of-](https://microsystems.electricalandcontrol.com/key-features-of-pic16x84-family-of-microcontrollers/)
] [pic16x84-family-of-microcontrollers/](https://microsystems.electricalandcontrol.com/key-features-of-pic16x84-family-of-microcontrollers/). [Accessed 11 January 2025].

[J. Mulindi, "PIC16F84 Microcontroller Features," 1 July 2023. [Online]. Available:
7 <https://www.electricalandcontrol.com/microcontroller-pic16f84/>. [Accessed 11
] January 2025].

[J. Mulindi, "How to Interface PIC Microcontroller to Input & Output Devices," 11
8 January 2024. [Online]. Available:
] [https://microsystems.electricalandcontrol.com/how-to-interface-pic-](https://microsystems.electricalandcontrol.com/how-to-interface-pic-microcontroller-to-input-output-devices/)
] [microcontroller-to-input-output-devices/](https://microsystems.electricalandcontrol.com/how-to-interface-pic-microcontroller-to-input-output-devices/). [Accessed 11 January 2025].

[J. Mulindi, "Multiplexers and Demultiplexers in Digital Circuits," 19 January 2023.
9 [Online]. Available: [https://www.electricalandcontrol.com/multiplexers-and-](https://www.electricalandcontrol.com/multiplexers-and-demultiplexers-in-digital-circuits/)
] [demultiplexers-in-digital-circuits/](https://www.electricalandcontrol.com/multiplexers-and-demultiplexers-in-digital-circuits/). [Accessed 11 January 2025].

[Components101, "An Overview of TTL and CMOS ICs and How to Choose Between
1 Them," Components101, 18 October 2019. [Online]. Available:

o [https://components101.com/articles/ttl-vs-cmos-ics-and-how-to-choose-between-](https://components101.com/articles/ttl-vs-cmos-ics-and-how-to-choose-between-them)
] them. [Accessed 11 January 2025].

[A. Raj, "Difference between Open Drain and Push Pull," Circuitdigest, 16 August
11 2016. [Online]. Available: [https://circuitdigest.com/forums/embedded/difference-](https://circuitdigest.com/forums/embedded/difference-between-open-drain-and-push-pull)
] between-open-drain-and-push-pull. [Accessed 11 January 2025].

[bedros, "Device Specifications and Logic Families," 11 September 2014. [Online].
1 Available: [https://www.slideserve.com/bedros/device-specifications-and-logic-](https://www.slideserve.com/bedros/device-specifications-and-logic-families)
2 families. [Accessed 11 January 2025].
]

[ElectronicsTutorials, "Quartz Crystal Oscillators," ElectronicsTutorials, [Online].
1 Available: [https://www.electronics-](https://www.electronics-tutorials.ws/oscillator/crystal.html#:~:text=These%20two%20capacitors%20determine%20the%20value%20of%20the,The%20crystal%20oscillates%20at%20its%20series%20resonance%20frequency..)
3 tutorials.ws/oscillator/crystal.html#:~:text=These%20two%20capacitors%20determi
] ne%20the%20value%20of%20the,The%20crystal%20oscillates%20at%20its%20serie
s%20resonance%20frequency.. [Accessed 11 January 2025].

[Microchip, "8-bit PIC® MCU External Crystal Oscillator Modes," Microchip, 22
1 November 2023. [Online]. Available:
4 [https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8bit-](https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8bit-pic/oscillator-options/external-crystal-oscillator-modes/#:~:text=The%20External%20Crystal%20Oscillator%20Mode%20is%20for%20external,drive%20all%20the%20peripherals%20and%20the%20pins)
] pic/oscillator-options/external-crystal-oscillator-
modes/#:~:text=The%20External%20Crystal%20Oscillator%20Mode%20is%20for%
20external,drive%20all%20the%20peripherals%20and%20the%20pins. [Accessed 11
January 2025].

[Broadcom, "8-mm (0.31-in.) Ultra Mini Seven-Segment Displays," [Online].
1 Available: [https://www.mouser.co.uk/datasheet/2/678/AV02_3584EN_2024_07_08-](https://www.mouser.co.uk/datasheet/2/678/AV02_3584EN_2024_07_08-1828161.pdf)
5 1828161.pdf. [Accessed 11 January 2025].
]

["Common Anode and Cathode 7 Segment display interfacing with Arduino," 1 June
1 2018. [Online]. Available: [https://razibshahdat.blogspot.com/2018/06/common-](https://razibshahdat.blogspot.com/2018/06/common-anode-and-cathode-7-segment.html)
6 anode-and-cathode-7-segment.html. [Accessed 11 January 2025].
]

["Piezo Buzzer Driver Schematic," 25 September 2015. [Online]. Available:
1 <https://fortunepdf.weebly.com/blog/piezo-buzzer-driver-schematics>. [Accessed 11
7 January 2025].
]

Appendices

Appendix 1 – Count from 0-F on 7 segment display code (1)

```
; FILE:          count.asm
; AUTHOR:        JMY
; DATE:          24th March 2005
; DESCRIPTION:    Count to 255 at 1 second intervals and display on a seven
;                segment display in binary.
;                The pause subroutine takes apx. 0.2 sec.
;                Pause is called 5 times

        list p=16F84          ; select processor type
        radix hex             ; define radix as hexadecimal
;
; #include "p16f84.inc"
;-----
; cpu equates (memory map)
portB    equ 0x06             ; port B address in register bank 0
trisB    equ 0x06             ; trisB address in register bank 1
PCL       equ 0x02            ; PCL address in register bank 0

status    equ 0x03            ; status register
RP0       equ 5               ; bit 5 of status register
RP1       equ 6               ; bit 6 of status register
Z         equ 2               ; bit 2 of status register

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

        org    0x00           ; indicates start of code

start
        ; start routine
        bcf     status,RP0     ; clear bank selection bit 5 in the status reg
        bcf     status,RP1     ; clear bank selection bit 6 in the status reg

        bsf     status,RP0     ; select bank 1
        movlw   0x00           ; load w with 0x00
        movwf   trisB          ; load it into trisb to make portb all outputs
        bcf     status,RP0     ; reselect bank 0

        clrf    portB          ; clear all lines low
        clrf    count          ; initialize counter to 0

get_count
        movlw   0x00           ; get_count subroutine
        addwf   count, w        ; Starting number from lookup table
        ; move count to W
        call    sevenseg_lookup ; call the look up table
        movwf   portB          ; move W to port B to update the display
        incf    count, f        ; increment the count f

        ; checks for if we are at F from the lookup table
        movf    count, w        ; move the current count into the w reg
        sublw   0x10           ; subtract 16 from the w reg to check if the count is at
F (16th Hex value in lookup table)
        btfsc   status, Z       ; if the count is at 16, 16-16=0, do clrf
        clrf    count          ; clears the count, going back to the 1st hex value in
the lookup table

        ; go to the pause if not on the 16th hex value
        call    pause          ; pause of 200ms
        goto    get_count      ; repeat forever

;-----
;subroutine pause: lets waste some time....
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;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)
;
;-----
sevenseg_lookup      ; lookup table
    addwlf    PCL, f      ; creates the offset for the lookup table to move down
values, stored in f
    RETLW    0x40 ;0 Hex value
    RETLW    0x79 ;1 Hex value
    RETLW    0x24 ;2 Hex value
    RETLW    0x30 ;3 Hex value
    RETLW    0x19 ;4 Hex value
    RETLW    0x12 ;5 Hex value
    RETLW    0x02 ;6 Hex value
    RETLW    0x78 ;7 Hex value
    RETLW    0x00 ;8 Hex value
    RETLW    0x18 ;9 Hex value
    RETLW    0x08 ;a Hex value
    RETLW    0x03 ;b Hex value
    RETLW    0x46 ;c Hex value
    RETLW    0x21 ;d Hex value
    RETLW    0x06 ;e Hex value
    RETLW    0x0e ;f Hex value

;-----

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

;-----
; Configuration! Important!
;
; at blast time, select:
;   memory unprotected
;   watchdog timer disabled
;   standard crystal (4 MHz)
;   power-up timer on

```

Appendix 2 – Buttons displaying L and R code (2)

```

; FILE:      count.asm
; AUTHOR:    JMY
; DATE:      24th March 2005
; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
;               segment display in binary.
;               The pause subroutine takes apx. 0.2 sec.
;               Pause is called 5 times

    list    p=16F84      ; select processor type
    radix    hex          ; define radix as hexadecimal
;   #include "p16f84.inc"
;-----
;   cpu equates (memory map)
portA    equ 0x05      ; port A address in register bank 0
portB    equ 0x06      ; port B address in register bank 0
trisa    equ 0x05      ; trisa address in register bank 1
trisb    equ 0x06      ; trisB address in register bank 1

status    equ 0x03      ; status register

```

```

RP0      equ 5           ; bit 5 of status register
RP1      equ 6           ; bit 6 of status register

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
;-----

;-----

org       0x00           ; indicates start of code

start
; start routine
bcf       status,RP0     ; clear bank selection bit 5 in the status reg
bcf       status,RP1     ; clear bank selection bit 6 in the status reg

bsf       status,RP0     ; select bank 1
movlw    0x00            ; load w with 0x00
movwf    trisb           ; load it into trisb to make portb all outputs
movlw    0x03            ; load w with 0x03, loading 1 on A0 and A1
movwf    trisa           ; load it into trisa to make the RA0 and RA1 inputs for the
buttons
bcf       status,RP0     ; reselect bank 0

clrf     portB           ; clear all lines low on portB
comf     portB, 1        ; inverts all lines high on portB
clrf     count          ; resets the value of count

buttons
; buttons routine
btfsc    portA, 0        ; Bit test - if the button on A0 is pressed, movf count
goto     display_r       ; skip is clear - if button on A0 is not pressed, go
straight to display_r subroutine
movf     count, w        ; if button was pressed, move the current count to the w reg
movlw    0x47            ; load w with 0x47 to create L on 7 seg display
movwf    portB           ; move W to port B to update the display
movwf    count          ; move the count back into the f reg
call     pause           ; pause of 200ms
movf     count, w        ; move the current count to the w reg
movlw    0x7f            ; load w with 0x7f to turn all LED's off
movwf    portB           ; move W to port B to update the display
movwf    count          ; move the count back into the f reg
call     pause           ; pause of 200ms

display_r
; display_r subroutine
btfsc    portA, 1        ; Bit test - if the button on A1 is pressed, movf count
goto     buttons         ; skip if clear - if the button on A0 is not pressed, loop
back to the buttons routine
movf     count, w        ; if button was pressed, move the current count to the w reg
movlw    0x2f            ; load w with 0x2f to create r on 7 seg display
movwf    portB           ; move W to port B to update the display
movwf    count          ; move the count back into the f reg
call     pause           ; pause of 200ms
movf     count, w        ; move the current count to the w reg
movlw    0x7f            ; load w with 0x7f to turn all LED's off
movwf    portB           ; move W to port B to update the display
movwf    count          ; move the count back into the f reg
call     pause           ; pause of 200ms
goto     buttons         ; loop back to buttons routine for endless loop

;-----
;subroutine pause: lets waste some time....
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;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
;-----
; Configuration! Important!
;
; at blast time, select:
;   memory unprotected
;   watchdog timer disabled
;   standard crystal (4 MHz)
;   power-up timer on
```

Appendix 3 – Buttons turning all LED's on and off code (3)

```
; FILE:      count.asm
; AUTHOR:    JMY
; DATE:      24th March 2005
; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
;              segment display in binary.
;              The pause subroutine takes apx. 0.2 sec.
;              Pause is called 5 times

    list p=16F84      ; select processor type
    radix hex         ; define radix as hexadecimal
;
; #include "p16f84.inc"
;-----
;   cpu equates (memory map)
portA    equ 0x05      ;port A address in register bank 0
portB    equ 0x06      ;port B address in register bank 0
trisa    equ 0x05      ;trisA address in register bank 1
trisb    equ 0x06      ;trisB address in register bank 1

status   equ 0x03      ;status register
RP0      equ 5         ;bit 5 of status register
RP1      equ 6         ;bit 6 of status register

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

    org     0x00        ; indicates start of code

start
    bcf     status,RP0  ; clear bank selection bit 5 in the status reg
    bcf     status,RP1  ; clear bank selection bit 6 in the status reg
```

```

    bsf      status,RP0    ; select bank 1
    movlw   0x00           ; load w with 0x00
    movwf   trisb          ; load it into trisb to make portb all outputs
    movlw   0x03           ; load w with 0x03
    movwf   trisa          ; load it into trisa to make the two lsb's inputs
    bcf     status,RP0     ; reselect bank 0

    clrf    portB          ; clear all lines low
    comf    portB, 1       ; inverts all lines high
    clrf    count         ; resets the value of count

;-----
; buttons routine
; Bit test - if the button on A0 is pressed, movf count
; skip if clear - if button on A0 is not pressed, go
; straight to display_r subroutine
; if button was pressed, move the current count to the W reg
; load w with 0x00 to turn LED's on
; move w to port B to update the display
; move the count back into f reg
; pause of 200ms
; display_r routine
; Bit test - if the button on A1 is pressed, movf count
; skip if clear - if the button on A0 is not pressed, loop
; back to the buttons routine
; if button was pressed, move the current count to the w reg
; load w with 0x7f to turn LED's off
; move W to port B to update the display
; move the count back into f reg
; pause of 200ms
; loop back to buttons routine for endless loop
;-----
;subroutine pause: lets waste some time...
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;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
; set w = 255 decimal
; mCount = w: outer loop counter
loadN
; set w = 255 decimal
; 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
; Configuration! Important!
;

```

```

; at blast time, select:
;   memory uprotected
;   watchdog timer disabled
;   standard crystal (4 MHz)
;   power-up timer on

```

Appendix 4 – Counter starting at 5 using buttons code (4)

```

; FILE:          count.asm
; AUTHOR:        JMY
; DATE:          24th March 2005
; DESCRIPTION:    Count to 255 at 1 second intervals and display on a seven
;                segment display in binary.
;                The pause subroutine takes apx. 0.2 sec.
;                Pause is called 5 times

list    p=16F84      ; select processor type
radix    hex          ; define radix as hexadecimal
#include "p16f84.inc"
;-----
;   cpu equates (memory map)
portA    equ 0x05      ; port A address in register bank 0
portB    equ 0x06      ; port B address in register bank 0
trisa    equ 0x05      ; trisa address in register bank 1
trisb    equ 0x06      ; trisB address in register bank 1
PCL      equ 0x02      ; PCL address in register bank 0

status    equ 0x03      ; status register
RP0       equ 5         ; bit 5 of status register
RP1       equ 6         ; bit 6 of status register
Z         equ 2         ; bit 2 of status register

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

org       0x00          ; indicates start of code

start     ; start routine
bcf       status,RP0    ; clear bank selection bit 5 in the status reg
bcf       status,RP1    ; clear bank selection bit 6 in the status reg

bsf       status,RP0    ; select bank 1
movlw    0x00           ; load w with 0x00
movwf    trisb          ; load it into trisb to make portb all outputs
bcf       status,RP0    ; reselect bank 0

clrf     portB          ; clear all lines low
movlw    0x05           ; load w with 5 from the lookup table
movwf    count          ; move w to f - load it into the seven segment display

get_count ; get_count subroutine
movf     count, w       ; move the current count to the W reg
call     sevenseg_lookup ; call the look up table
movwf    portB          ; move W to port B to update the display

btfsc    portA, 0       ; is the button on RA0 pressed?
goto     decrement      ; if the button on RA0 is not pressed, goto decrement
goto     increment      ; if the button on RA0 is pressed, goto increment

increment ; increment subroutine
call     pause          ; pause to avoid the button debouncing, counting up by
just 1 when PIC is set to 4MHz
movf     count, w       ; move the current count into the W reg
sublw    0x09           ; subtract 9 from the w reg to check if the count is at 9
btfsc    status, Z       ; if the count is at 9, 9-9=0, jump to check_upper
goto     get_count       ; goto get_count if the count is at 9

incf     count, f       ; Increment the counter - move down the lookup
table
call     pause          ; pause for a while (200ms)
goto     get_count       ; goto get count

```

```

decrement          ; decrement subroutine
    btfsc    portA, 1      ; is the button on RA1 pressed?
    goto     get_count     ; if the button on RA1 is not pressed, loop back to
get_count and wait for a button press
    call     pause         ; if pressed, pause to avoid the buttons debouncing,
counting up by just 1 when PIC is set to 4MHz
    movf     count, w      ; move the current count to the w reg
    sublw    0x00         ; subtract 0 from the w reg to check if the count is
already at 0
    btfsc    status, Z     ; if the result from the sum is 0, goto get_count
    goto     get_count     ; goto get_count to wait until the display isnt at 0

    decf     count, f      ; decrement the counter - move up the lookup table
    call     pause         ; pause of 200ms
    goto     get_count     ; repeat forever - go back to get_count
subroutine

;-----
;subroutine pause: lets waste some time....
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;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)
;-----

sevenseg_lookup    ;lookup table
    addwf     PCL, f      ;creates the offset for the lookup table to move down
values, stored in f
    retlw    0x40 ;0 hex value
    retlw    0x79 ;1 hex value
    retlw    0x24 ;2 hex value
    retlw    0x30 ;3 hex value
    retlw    0x19 ;4 hex value
    retlw    0x12 ;5 hex value
    retlw    0x02 ;6 hex value
    retlw    0x78 ;7 hex value
    retlw    0x00 ;8 hex value
    retlw    0x18 ;9 hex value

;-----

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

;-----
; Configuration! Important!
;
; at blast time, select:
;     memory unprotected
;     watchdog timer disabled

```

```
; standard crystal (4 MHz)
; power-up timer on
```

Appendix 5 – Piezo Sounder Code (5)

```
; FILE: count.asm
; AUTHOR: JMY
; DATE: 24th March 2005
; DESCRIPTION: Count to 255 at 1 second intervals and display on a seven
; segment display in binary.
; The pause subroutine takes apx. 0.2 sec.
; Pause is called 5 times

list p=16F84 ; select processor type
radix hex ; define radix as hexadecimal
#include "p16f84.inc"
;-----
; cpu equates (memory map)
portA equ 0x05 ; port A address in register bank 0
portB equ 0x06 ; port B address in register bank 0
trisa equ 0x05 ; trisa address in register bank 1
trisb equ 0x06 ; trisB address in register bank 1

status equ 0x03 ; status register
RP0 equ 5 ; bit 5 of status register
RP1 equ 6 ; bit 6 of status register

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
;-----

;-----
org 0x00 ; indicates start of code

start ; start routine
bcf status,RP0 ; clear bank selection bit 5 in the status reg
bcf status,RP1 ; clear bank selection bit 6 in the status reg

bsf status,RP0 ; select bank 1
movlw 0x7f ; load w with 0x7f, turning all LED's on display off
movwf trisb ; load it into trisb to make portb all outputs
movlw 0x03 ; load w with 0x03
movwf trisa ; load it into trisa to make the two lsb's inputs
bcf status,RP0 ; reselect bank 0

clrf portB ; clear all lines low on portB
clrf count ; resets the value of count

buzzer ; buzzer routine
btfsc portA, 0 ; Bit test - if the button on A0 is pressed, Bit test portA
again goto buzzer ; skip if clear - if the button on A0 is not pressed,
loop back to buzzer routine
clrf portB ; if button was pressed, clear all lines low on portB
movf count, w ; move the current count to the w reg
movlw 0xff ; load w with 0xff
movwf portB ; move W to port B to turn buzzer on
movwf count ; move the count back into f reg
call pause ; pause of 200ms
movf count, w ; move the current count to the w reg
movlw 0x7f ; load w with 0x7f
movwf portB ; move w to port B to turn buzzer off
movwf count ; move the count back into the f reg
call pause ; pause of 200ms
goto buzzer ; loop back to buzzer routine for endless loop
```

```

;-----
;subroutine pause: lets waste some time...
;
;inputs: nothing
;outputs: nothing
;
;does: nothing but waste some time
;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)
;
;-----

; mCount of 8 and nCount of 200 is a frequency of approx 2.4kHz

;-----

pause
movlw    0x08      ; set w = 8 decimal, change this value to change sounder
tone
movwf    mCount    ; mCount = w: outer loop counter
loadN
movlw    0xc8      ; set w = 200 decimal, change this value to change sounder
tone
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 200 and decrement
return   ; else exit subroutine

end

;-----
; Configuration! Important!
;
; at blast time, select:
;   memory unprotected
;   watchdog timer disabled
;   standard crystal (4 MHz)
;   power-up timer on

```

James T Micros assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

76/100

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

GRADING FORM: GJ_2019

JAMES T

381

ENGAGEMENT WITH LITERATURE



Good range of relevant sources cited.
Note that its not standard practice to put the [1]

78

citation at the start of the sentence - see things like IEEE published papers for typical usage.

KNOWLEDGE AND UNDERSTANDING

 A very good level of K&U derived from the EWL. Excellent breakdown of all the components. **78**

COGNITIVE AND INTELLECTUAL

 All but one programming tasks attempted ok. Reasonable level of though put into the code with sensible flowcharting. **70**

PRACTICAL APPLICATION

 Circuit well implemented. 5 out 6 tasks done. **75**

TRANSFERABLE SKILLS

 Very Well written and presented. **80**