

Chris B assignment

by Christopher B

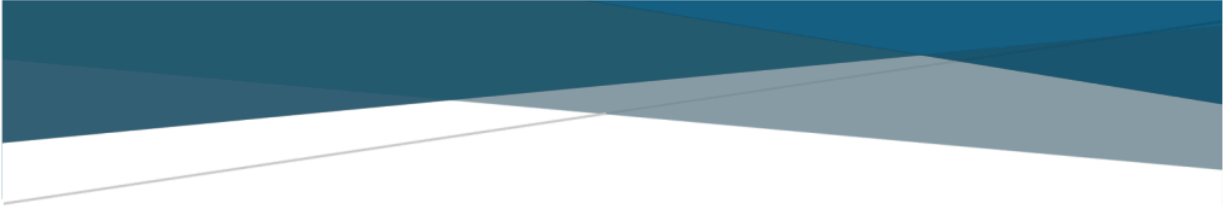
Submission date: 13-Jan-2025 04:22PM (UTC+0000)

Submission ID: 248293324

File name: 52212_Christopher_B_Chris_B_assignment_621295_1473004682.docx (614.05K)

Word count: 4162

Character count: 19660



ACDA4004 – MICROS, PERIPHERALS AND INTERFACES

Abstract

Assignment covering the function of the given circuit and its replication, displaying hexadecimal characters, and a variety of programming task.

Christopher B (2322626)

Contents

Table of figures	1
Introduction:	2
Section 1:	2
The given circuit:	2
The recreated circuit:	2
Section 2:	4
Conversion table:	5
Section 3:	7
Program 1:	7
Program 2:	11
Program 3:	13
Program 4:	16
Program 5:	20
Conclusion:	23
Bibliography	24

Table of figures

Figure 1: Original circuit to be reproduced.	2
Figure 2: Reproduced circuit in proteus	3
Figure 3: Shows which pins control each segment	4
Figure 4: This table shows the character values converted to binary and hex, with the buzzer being off.	7
Figure 5: flow chart for program 1	10
Figure 6: flow chart for program 2	13
Figure 7: flow chart for program 3	16
Figure 8: flow chart for program 4	19
Figure 9: flow chart for program 5	22

Introduction:

This report will cover 3 main sections. The first section will cover a given circuit, with a recreation of it and an explanation of each part of the circuit. The second section covers displaying 0-9, a-f on the seven segment display, using the pic microcontroller. It also covers converting from binary to hexadecimal. The third segment contains code for completing the given tasks, including a flow chart for each program.

Section 1:

The given circuit:

The figure(1) below shows the original circuit that was to be produced in proteus.

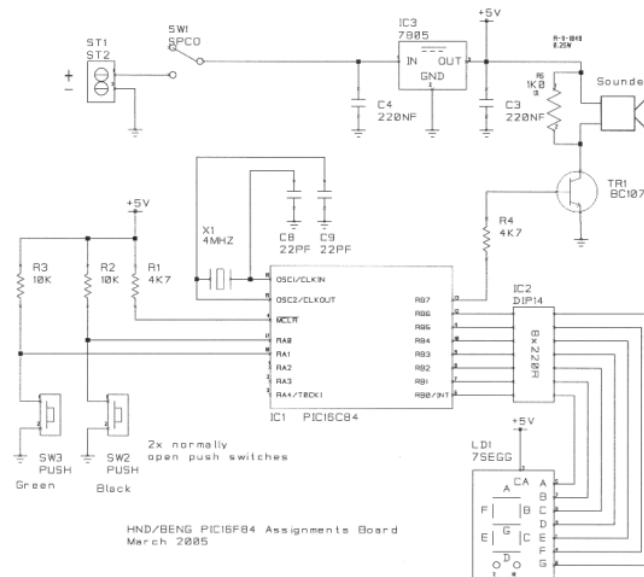


Figure 1: Original circuit to be reproduced.

The recreated circuit:

The circuit can be reproduced in proteus, and is shown in figure 2 below. All parts of the circuit have been reproduced, however, IC3, C3, C4, and SW1 are not used in the proteus simulation. The reason this was not needed was because it is part of a voltage regulation circuit, used to produce a clean 5V DC output. The IC3 7805 chip is used to regulate the supply voltage [1] to 5V, C3 is used to filter out ripples or instability in the input waveform, and C4 is used to filter out noise and ripples produced by the voltage regulator. The input

capacitor is also used to protect the 7805 device from transient voltages [1]. This was not necessary to reproduce in proteus, as a 5V supply can be selected and used in proteus without any regulation.

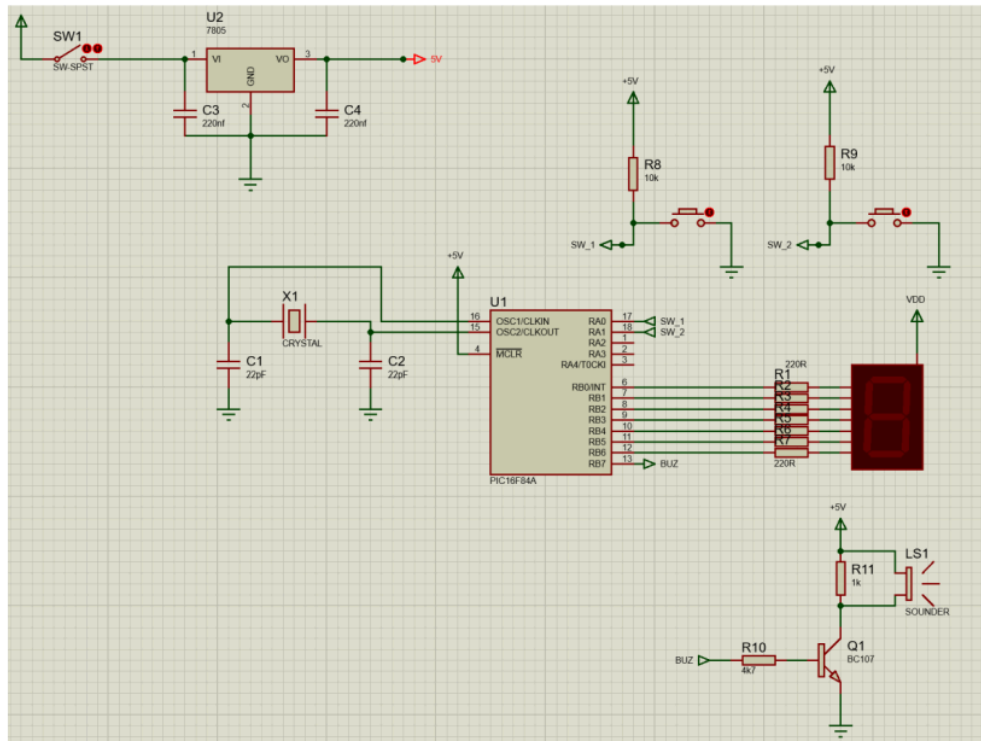


Figure 2: Reproduced circuit in proteus

The reproduced circuit has all the same sections and functions as the original one.

There are two buttons connecting to RA0, and RA1. When pressed, the pins are pulled low to ground, when not pressed, the pin is high. They are connected to the pic using ports, this is purely to make the circuit diagram easier to read.

The PIC microcontroller on proteus can have an internal oscillator, or external oscillator, and in the original it is external, so that has been recreated. The oscillator consist of 2 capacitors, and a crystal oscillator, at 4MHz. Crystal oscillators vibrate at a resonant frequency, depending on its cut. The capacitors are needed to start and sustain a stable oscillation, as well as filter out unwanted oscillations [2] [3]. The output pins are split from porta0-4, and portb0-7. PortA is a 5 bit bi-directional port set controlled by the register TRISA, the bits in this register can be set high or low to set the pin to input or output. PortB is similar, it is an 8 bit bi-directional port set, controlled by the TRISB register [4]. The PIC16F8 It is programmed with assembly language, which can be done from inside of proteus.

The sounder circuit has been connected directly to a 5V supply, removing the need for the regulation circuit, as previously discussed. The sounder is operated by the PIC output pin connected to the base of a transistor, that, when in conduction and fully saturated, can

provide a current path to ground, turning on the sounder. The resistor R11, is used to discharge the capacitance of the sounder, when the transistor is not in conduction.

The original circuit has a package called IC2 DIP14. This is a dual in line resistor package, giving seven resistors in a small, contained package. This has been replaced by single resistors, but would function exactly the same. The seven segment display is an array of led segments, and is active low. This means that when the pic outputs low, the led segment is on, and when the pic outputs high, it turns off. The pic outputs can be selected as such so that the led segment displays numbers and letters. The value of the resistors is calculated using $R=V/I$, with LED's typically drawing 10-25mA, using 220Ohm resistors means that the leds draw 22.7mA ($5/220 = 0.02272$).

Section 2:

To display hexadecimal characters 0-F on the seven segment display, it must first be determined which outputs correspond to which segments on the display. The 8 RB0-6 outputs control the 7 segment display, with RB7 connected to the sounder. The seven segment display segments are active low, so to control them the pin must be off. To find which pin controls which segment, 8 bits can be sent to the RB0-7 pins, shifting 1 each time, for example, one time send 0000 0001, the next send 0000 0010, thus giving a mapping of each pin to each segment. To test this, the value must be moved into the working register (movlw), and then moved into f, location portb (movwf portb) [5].

From testing this, the segments are controlled as follows:

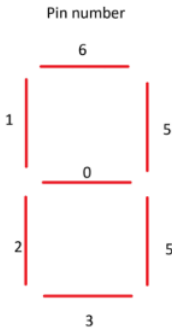
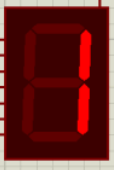


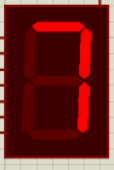
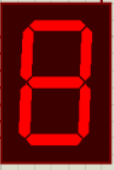
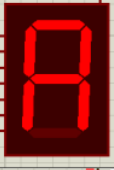
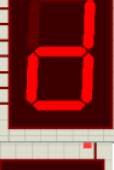
Figure 3: Shows which pins control each segment

The next step is to work out what segments make a number or letter, for example, for 0, all segments bar the middle must be on, so all pins bar the middle must be low. This means the 8 bit code for this would be 0000 0001

The rest can be seen in the table below.

Conversion table:

Character	Binary	Hex	Output
0	0100 0000	40	
1	0111 1001	79	
2	0010 0100	24	
3	0011 0000	30	
4	0001 1001	19	
5	0001 0010	12	
6	0000 0010	02	

7	0111 1000	78	
8	0000 0000	00	
9	0011 1000	18	
A	0000 1000	08	
B	0000 0011	03	
C	0100 0110	46	
D	0010 0001	21	
E	0000 0110	06	

F	0000 1110	0E	
---	-----------	----	---

Figure 4: This table shows the character values converted to binary and hex, with the buzzer being off.

As is shown in the table above, the binary values have been converted to Hexadecimal to use in the assembly code. To convert the binary values to hexadecimal, each nibble was converted, as one nibble (4 bits) can represent 16 values (8+4+2+1, 0 to 15), and so can hexadecimal (0-9, and a-f for 10-15). This means that to convert the 8 bit binary values, it must be split into nibbles and converted. For example, F character is displayed with 0000 1110, the first nibble, 0000, is equal to 0, and the second nibble, 1110, is equal to 14, and the 14th value in hex is E, this means that 0000 1110 = 0E. This principle was used to calculate all the values in the table.

Section 3:

Before writing each section of code, it was first planned out in a flow chart.

Program 1:

"Modify "Count.asm" so that the seven segment display counts from 0 to F in hexadecimal numbers in 0.2 second increments"

```
; FILE:          count.asm
; AUTHOR:        Chris
; 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
trisb      equ 0x06      ;trisB address in register bank 1
PCL        equ 0x02

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        0x000

start
```

```

        bcf          status,RP0      ; clear bank selection bits in the status
reg      bcf          status,RP1

        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            ; load w with 0x00
        addwf        count, w        ; move count to W
        call         seg_table        ; call the look up table
        movwf        portB           ; move W to port B to update the display
        incf         count, f        ; increment the count f

        movf         count, w        ; move count to register w
        sublw        0x10            ; subtract 0x10(16) from w register
        btfsc        status, z       ; if the last instruction yeilds 0, do not skip
next     clrf         count           ; if not skipped, clear count to 0, resetting the
lookup   table position

        call         pause           ;pause for 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)
;
;-----

seg_table
        addwf        PCL, f          ;add value of pcl to f
        retlw        0xc0            ;0 return to address in stack (return from subroutine)
and load literal value into w register
        retlw        0x79            ;1 load segments for 1
        retlw        0x24            ;2 load segments for 2
        retlw        0x30            ;3 load segments for 3
        retlw        0x19            ;4 load segments for 4
        retlw        0x12            ;5 load segments for 5
        retlw        0x02            ;6 load segments for 6
        retlw        0x78            ;7 load segments for 7
        retlw        0x00            ;8 load segments for 8
        retlw        0x18            ;9 load segments for 9
        retlw        0x08            ;A load segments for A
        retlw        0x03            ;b load segments for b
        retlw        0x46            ;C load segments for C

```

```

    retlw 0x21    ;d load segments for d
    retlw 0x06    ;e load segments for e
    retlw 0x0e    ;F load segments for F

;-----

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

```

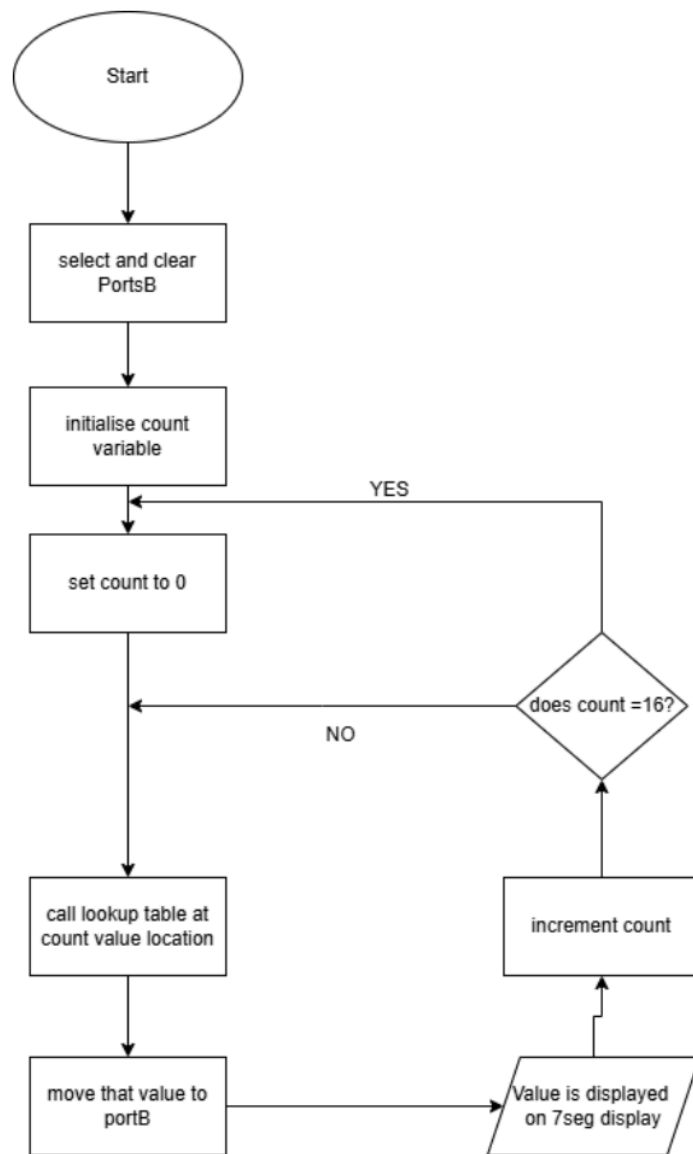


Figure 5: flow chart for program 1

Program 2:

"Write a program so that when you press one of the two buttons, it displays "L" on the 7 segment LED display, then when you press the other one, it displays "r"."

```
; FILE:          count.asm
; AUTHOR:        Chris

; DESCRIPTION:    Display L when l button is pressed, r when the other is
;
;
;
; list p=16F84      ; select processor type
; radix hex         ; define radix as hexadecimal
; #include "pl6f84.inc"
;-----
; cpu equates (memory map)
portA      equ 0x05
trisa      equ 0x05
portB      equ 0x06      ;port B address
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
;-----

RST code 0x0      ;
    goto start

PGM code
start
    bcf      status,RP0      ; clear bank selection bits in the status
reg
    bcf      status,RP1
    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 0x00
    movwf    trisa      ; load it into trisa to make porta the two
lsb inputs

    bcf      status,RP0      ; reselect bank 0

    clrf     portB      ; clear all lines low
    comf     portB,1
    clrf     count      ; initialize counter to 0
```

```

seg_l
    btfsc    portA,0      ; check if button on a0 is pressed
    goto     seg_r        ; go to seg_r if button is not pressed
    movf     count,w      ; move count to w
    movlw    0x47         ; move 0x47, or L into the w reg
    movwf    portB        ; move w register into portb, the output pins
    movwf    count        ; move count into f reg
    call     pause        ; wait 200ms
    call     pause
    call     pause
    call     pause
    movf     count,w      ;sets 7seg to 0 after 1 second
    movlw    0x7f         ; move w into f and to portb
    movwf    portB
    movwf    count
    call     pause

seg_r
    btfsc    portA,1      ; check if button on a1 is pressed
    goto     seg_l        ; go to seg_L if button is not pressed
    movf     count,w      ; move count to w
    movlw    0x2f         ; move 0x2f, or L into the w reg
    movwf    portB        ; move w register into portb, the output pins
    movwf    count        ; move count into f reg
    call     pause        ; wait 200ms
    call     pause
    call     pause
    call     pause
    movf     count,w      ; sets 7seg to 0 after 1 second
    movlw    0x7f         ; move w into f and to portb
    movwf    portB
    movwf    count
    call     pause
    goto     seg_l        ; go back to seg_l

;-----
;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

```

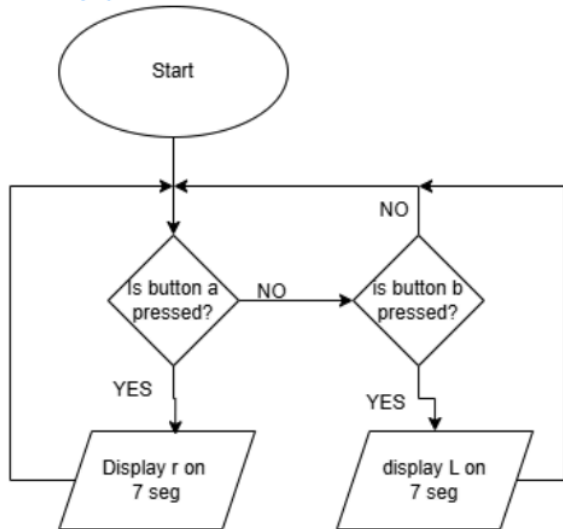


Figure 6: flow chart for program 2

Program 3:

"Write another program, 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)"

```

; FILE:          count.asm
; AUTHOR:        Chris
; DESCRIPTION:    Latching buttons turning 7seg on or off
;
;
;
        list p=16F84          ; select processor type
        radix hex              ; define radix as hexadecimal
;        #include "p16f84.inc"
;-----
;        cpu equates (memory map)
portA    equ 0x05
trisa    equ 0x05

```

```

portB      equ 0x06      ;port B address
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

;-----

RST code 0x0      ;
    goto start

PGM code
start
    bcf     status,RP0    ; clear bank selection bits in the status
reg
    bcf     status,RP1
    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 0x00
    movwf   trisa         ; load it into trisa to make porta the two
lsb inputs
    bcf     status,RP0    ; reselect bank 0

    clrf    portB         ; clear all lines low
    comf    portB,1
    clrf    count        ; initialize counter to 0

seg_on
    btfsc   portA,0       ; check if button on a0 is pressed
    goto    seg_off      ; go to seg_off if button is not pressed
    movf    count,w       ; move count to f
    movlw   0x00          ; move 0x00 to w reg
    movwf   portB         ; move 0x00 to portB, turning on the leds
    movwf   count        ; return count to f reg
    call    pause         ; wait 200ms

seg_off
    btfsc   portA,1       ; check if button on a1 is pressed
    goto    seg_on       ; go to seg_on if not pressed
    movf    count,w       ; move count to f
    movlw   0x7f          ; move 0x7f to w
    movwf   portB         ; move 0x7f to portb, turning off the leds
    movwf   count        ; return count to f reg
    call    pause         ; wait 200ms

```

```

        goto    seg_on        ; go back to seg_on

;-----
;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

```

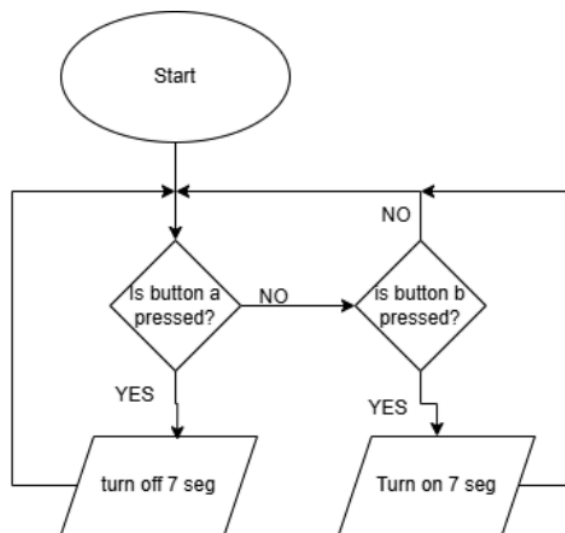


Figure 7: flow chart for program 3

Program 4:

"Then combine all of these techniques together, by writing a new program where you start by displaying '5' but you can count up (to a maximum of 9) if you press one button, and count down (to a minimum of 0) if you press the other."

```

; FILE:          count.asm
; AUTHOR:        Chris
; DESCRIPTION:    Starts by displaying 5, then count up to 9 with one
button
;                and down to 0 with the other
;
;
;
list p=16F84      ; select processor type
radix hex         ; define radix as hexadecimal
; #include "p16f84.inc"
;-----
; cpu equates (memory map)
portA equ 0x05
trisa equ 0x05
portB equ 0x06      ;port B address
trisb equ 0x06      ;trisB address in register bank 1
PCL equ 0x02
z equ 2
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

```

```

;-----
RST code 0x0      ;
    goto start

PGM code
start
    bcf      status,RP0    ; clear bank selection bits in the status
reg
    bcf      status,RP1

    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 porta the two
lsb inputs
    bcf      status,RP0    ; reselect bank 0

    clrf     portB         ; clear all lines low
    movlw    0x05
    movwf    count

get_count
    movf     count,w       ; move count to w
    call     seg_table     ; go to seg_table
    movwf    portB         ; move w to portb

    call     pause         ; wait 200ms
    btfsc    portA,0       ; check if button on a0 is pressed
    goto     decrement    ; if it is, go to decrement
    call     pause         ; wait 200ms
    goto     increment     ; then go to increment

increment

    movf     count,w       ; move count to w
    sublw    0x09          ; subtract 0x09 with w register
    btfsc    status,z      ; test the status of the z bit
    goto     get_count     ; go to get_count if z is 1
    incf     count,f       ; increment count in f by 1
    call     pause         ; wait 200ms
    goto     get_count     ; go to get_count

decrement

    btfsc    portA,1       ; check if button on a1 is on
    goto     get_count     ; if not go to get_count
    call     pause         ; wait 200ms
    movf     count,w       ; move count to w

```

```

    btfsc    status,z      ; check z bit status
    goto     get_count     ; if 1, go to get_count

    decf     count,f       ; decrememnt count by 1
    call     pause         ; wait 200ms
    goto     get_count     ; go to get_count

;-----
;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)
;
;-----
seg_table
    addwf    PCL, f        ;add value of pcl to f
    retlw    0xc0          ;0 return to address in stack (return from subroutine)
and load literal value into w register
    retlw    0x79          ;1 load segments for 1
    retlw    0x24          ;2 load segments for 2
    retlw    0x30          ;3 load segments for 3
    retlw    0x19          ;4 load segments for 4
    retlw    0x12          ;5 load segments for 5
    retlw    0x02          ;6 load segments for 6
    retlw    0x78          ;7 load segments for 7
    retlw    0x00          ;8 load segments for 8
    retlw    0x18          ;9 load segments for 9

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

```

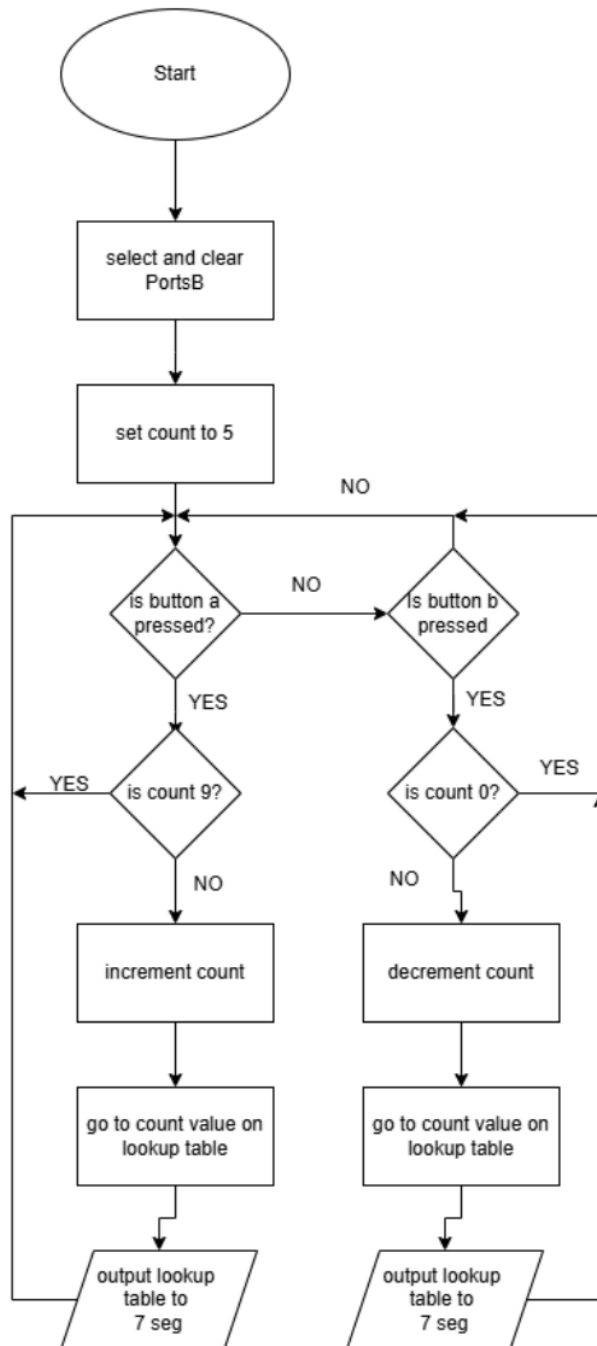


Figure 8: flow chart for program 4

Program 5:

"Write a new program so that the piezo sounder produces a tone while one of the switches/buttons is pressed. As you are using a simulation program and not the actual hardware, see if you can get Proteus to output the virtual speaker/sounder to the actual PC speaker, but if this fails, attach a virtual oscilloscope to the speaker/sounder and monitor the waveform visually."

```
; FILE:          count.asm
; AUTHOR:        Chris
; DESCRIPTION:    Produce a tone when button is pressed
;
;
;
        list p=16F84          ; select processor type
        radix hex             ; define radix as hexadecimal
;      #include "p16f84.inc"
;-----
;      cpu equates (memory map)
portA    equ 0x05
trisa    equ 0x05
portB    equ 0x06             ;port B address
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
;-----

RST code 0x0      ;
        goto start

PGM code
start
        bcf      status,RP0    ; clear bank selection bits in the status
reg      bcf      status,RP1
        bsf      status,RP0    ; select bank 1

        movlw    0x7f          ; 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
```

sound

```
    btfsc portA,1    ; if button a1 is pressed
    clrf  portB      ; clear portB
    btfsc portA,1    ; if button a1 is pressed
    goto  sound      ; go to sound
```

```
    movlw 0xff       ; move 0xff into w
    movwf portB      ; move w into portb
    call  pause      ; wait
    movlw 0x7f       ; move 0x7f into w
    movwf portB      ; move w into port b
    call  pause      ; wait
    goto  sound      ; go to sound
```

```
-----
;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 0x02       ; set w = 2 decimal
    movwf mCount     ; mCount = w: outer loop counter
loadN
    movlw 0xbb       ; set w = 187 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

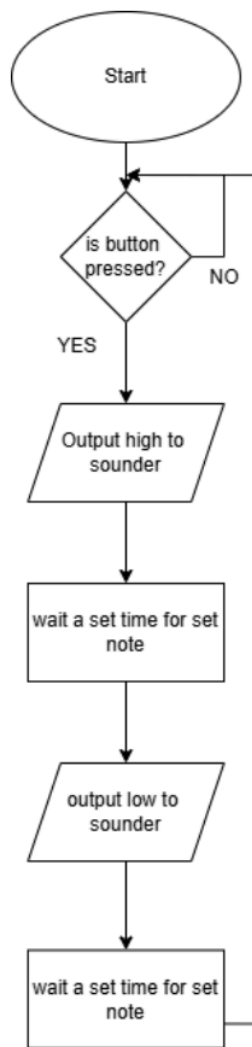


Figure 9: flow chart for program 5

Conclusion:

This report has covered the basic function of a given circuit, including component functions, and how this can be recreated in proteus. It has also covered the basics of addressing a 7 segment display from a pic micro controller, including converting from binary to hexadecimal. It also includes 5 coding tasks and a flow chart for each.

Bibliography

- [1] Texas Instruments, "LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators," [Online]. Available: <https://www.ti.com/lit/ds/symlink/lm7800.pdf>. [Accessed 13 01 2025].
- [2] G. C. -. P. P. Engineer, "Creating Reliable Crystal Oscillators Circuits for Your MCU Design," Renesas, 30 01 2023. [Online]. Available: <https://www.renesas.com/en/blogs/creating-reliable-crystal-oscillators-circuits-your-mcu-design>. [Accessed 11 01 2025].
- [3] Microchip Technology inc, "Crystal Oscillator Basics and Crystal Selection for rPIC and PICmicro Devices," [Online]. Available: <https://ww1.microchip.com/downloads/en/appnotes/00826a.pdf>. [Accessed 11 01 2025].
- [4] Microchip technology, "18-pin Enhanced FLASH/EEPROM 8-Bit Microcontroller," [Online]. Available: <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/35007C.pdf>. [Accessed 13 01 2025].
- [5] Microchip technology, "Section 29. Instruction Set," [Online]. Available: <https://ww1.microchip.com/downloads/en/DeviceDoc/31029a.pdf>. [Accessed 11 01 2025].

Chris B assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

68/100

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

GRADING FORM: GJ_2019

CHRISTOPHER B

340

ENGAGEMENT WITH LITERATURE



Some citations used well. but the number is too limited - need to use more reference sources, and cover in depth. even more of the principles of the components.

58

KNOWLEDGE AND UNDERSTANDING



What is written is fine, but adding more details and referring to more research text would have helped get a higher mark here.

67

COGNITIVE AND INTELLECTUAL

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

70

PRACTICAL APPLICATION

 Circuit well implemented. 5 out 6 tasks done.

75

TRANSFERABLE SKILLS

 Well written and presented, but more actual info,
diagrams etc on things like the component
explanations would have been good.

70