

MPI Assignment

by Jamie Hamilton-Swingewood

Submission date: 14-Jan-2025 03:39AM (UTC+0000)

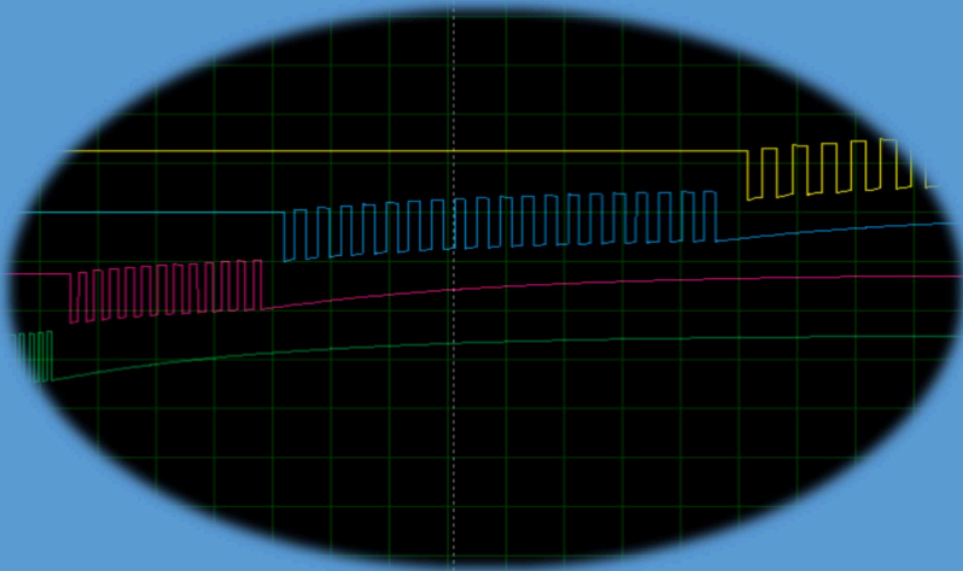
Submission ID: 248362473

File name: 51878_Jamie_Hamilton-Swingewood_MPI_Assignment_621295_1803134814.pdf (2.9M)

Word count: 4812

Character count: 25394

MICROS, PERIPHERALS AND INTERFACES



Jamie Hamilton-Swingewood

ACDA4004 University of Wales Trinity St David

Table of Contents

Introduction:	2
What is a microchip.	3
Keeping Time	5
The 7-Segment Display	6
Piezo sounder and Pulse width modulation.	8
Task 1: Display Hexadecimal Alphabet	11
Task 2: Buttons Display L and r	17
Task 3: Buttons Display all or None	20
Task 4: Buttons count up and down:	22
Task 5: Sounder	31
Task 6: PWM Signal.....	35
Bibliography	41

Introduction:

Microcontrollers have had a profound impact on the development of science, technology and engineering and their importance in the modern day cannot be overstated. Used in countless industries for almost any task that one could conceive of, electrical control units for cars, sensors, in factory equipment or even white goods and smartphones. Every modern electronic device has some form of 'microchip'.^[1] In this paper we will outline simulations and experiments using the PIC16F84A, which is 8 bit, 18 pin, reduced instruction set (RISC) microcontroller having 35 single word instructions.^[2]

Section 1:

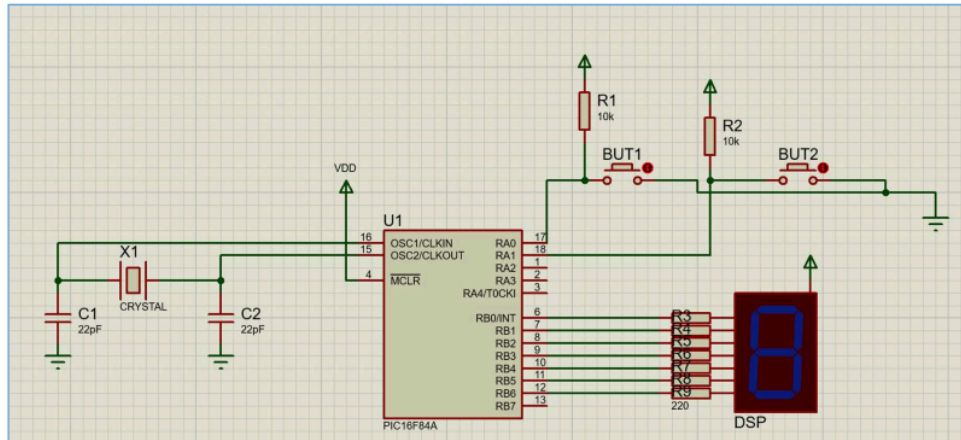


Figure 1: circuit diagram of system to be used, created using proteus, at a high level it's simply two buttons a reference clock a seven-segment display all connected to the PIC microcontroller.

Label	Component Type	Description
X1	Crystal Oscillator	This delivers a clock signal of DC - 20 MHz clock input to microcontroller.
C1 & C2	Capacitor	Both 22pF, deliver frequency stabilisation and noise reduction to the clock signal.
U1	Microcontroller	PIC16F84A, functions as the nerve centre of the circuit, processing code/instructions and logic. The RAX and RBx ports are bi-directional I/O ports. In this case input logic through RA0 and RA1 and output logic RB0-RB6. OSC1/CLKIN is the input pin for the crystal oscillator and OSC2/CLKOUT the output to the oscillator.
R1 & R2	Resistor	Both 10kΩ resistors, prevent change to logic of inputs.
BUT1 & BUT2	Button	Provide the change in input logic to get the microcontroller to do a function.
R3 – R9	Resistor	All resistors 220Ω, act as a barrier limiting the current between the outputs from RB0 – RB7 and the input pins of the 7-segment display.
DSP	Display	Each of the segment of the display are connected to the output ports RB0 to RB7. Can control output from PIC individually or as a unit via software, to only activate certain segments or display a specific character.

What is a microchip.

Before the advent of silicon integrated circuits electronics were dependent on large vacuum tubes for logic, whilst functionally the same and easy to handle there is a fundamental limit on available computing power simply due to the size of these compared to silicon transistors.^[3]



Figure 2: Replica of the world's first semiconductor transistor, invented at bell labs by physicists William Shockley, Walter Brattain and John Bardeen 1947.[3]

Whilst mostly superseded by more advanced silicon processes the vacuum tube does live on in specific industrial or scientific applications. Featuring prominently in the music world with many musicians swearing off silicon-based amplification for valve/tube amplifiers often at a financial premium, and less functionality but alleged better sound quality than solid-state amplifiers.

A microchip or more generally an integrated circuit (IC) is an assembly of what would previously be discrete components packed into a single monolithic device.^[4] In modern ICs there are billions of transistors, diodes, resistors and capacitors connected in a bewilderingly complex web but the process is still driven by logic gates, it is the further reduction in size and thereby increase in computational density allowing more compute per square millimetre of silicon overtime.

A PIC microchip is a specific family of IC developed by the microchip technology company for use in embedded systems, with the acronym PIC being an elusive term to define. It appeared to evolve over time from peripheral interface controller, programmable interface controller to programmable intelligent computer.^[5] The PIC acts as an interface between incoming and outgoing signals for example wanting to display characters on an LED display as in figure 1, the display cannot process the code required on its own and so the microchip or microprocessor process is the instructions that are given outputting the defined signal which can then be understood by the display. This holds true even for modern displays although more complex than a humble 7 segment LED the principle is the same, a modern display still requires an input signal often in the form of display port or HDMI and the input signal needs to be processed by a microchip of some description to be displayed an example in figure 3 below.

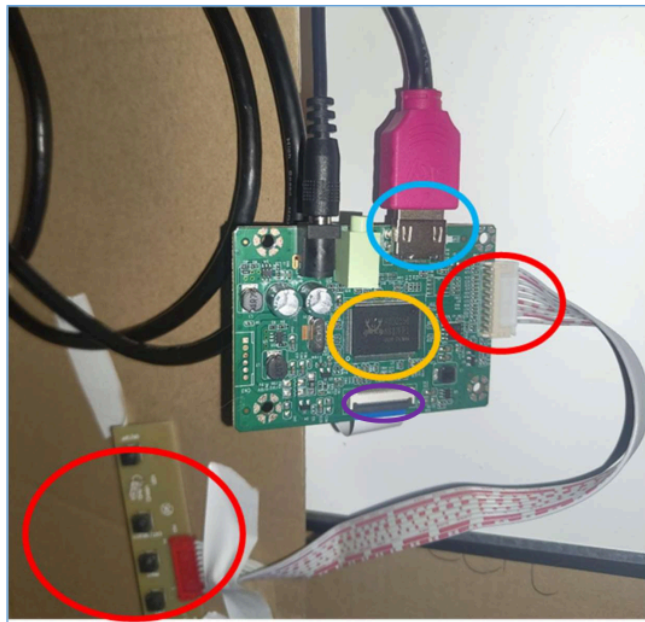


Figure 3: An example from the wild, this is the circuit board and microchip (yellow circle), for a NT156FHM-N41 salvaged laptop LCD. Just like the PIC16F84A and the circuit in figure 1 there are various I/O ports with the logic handled by the central processor. In blue is the input signal for HDMI, in purple is the output to the LCD display being an (eDP) embedded display port cable, in red are the buttons for power and the like analogous to the RA0 and RA1 pins in the PIC.

Keeping Time

Processing hundreds, thousands or even billions of instructions a second, with modern processors well into the GHz^[6], comes with chronological challenges. Requiring the precise and accurate keeping of time or at least accurate within the inertial reference frame of the microchip as moving at relativistic speeds subject to time dilation will play havoc with communications when humanity eventually reaches the stars.^[7] Relativity aside the challenge of having a sensible accurate and reliable reference for microchips comes in the form of piezo electric crystal oscillators. The piezoelectric effect being discovered by the physicists Jacques and Pierre Curie in 1880, is broadly that when a piezo electric material is subject to mechanical stimuli, stress, pressure, compression, or similar there is build-up of an electrical charge and potential difference within the material. It is not one singular field but a build-up of charge because of the displacement of ions through the lattice structure of suitable crystals with the individual unit adding up to form a macroscopic voltage across the crystal lattice.

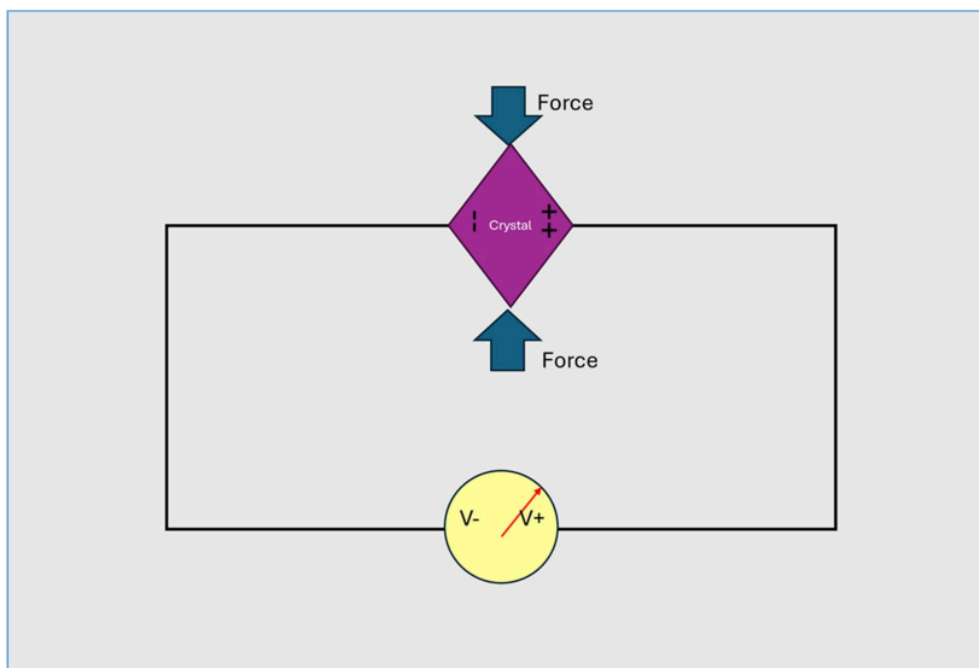


Figure 4: A simplified diagram of the piezo electric effect, the crystal turning mechanical energy exerted on the crystal into electrical energy

Likewise, it follows that one can apply an electric field across said piezo electric material it will deform in a regular and repeatable manner as the individual unit cells that make up the macroscopic crystal are displaced by the movement of ions because of the electric field.^[8,9] leveraging this picking a suitable material applying an alternating voltage across it and counting how many times it moves within a given time frame allows the microchip to accurately keep time. Quartz crystals are typically used for this purpose and microchip define the practical frequency range of fundamental mode crystals to be 600kHz to 30MHz.^[10]

The seven segment display is a series of light emitting diodes arranged in a figure of eight allowing for the display of characters by choosing to selectively activate certain parts of the LED array. These are controlled by the RB0-RB7 ports coming from the PIC16F84A. When current flows and a particular voltage is reached the LED will be activated. Seen below in figure 5 and used within the circuit in Figure 1 is a blue common anode display.

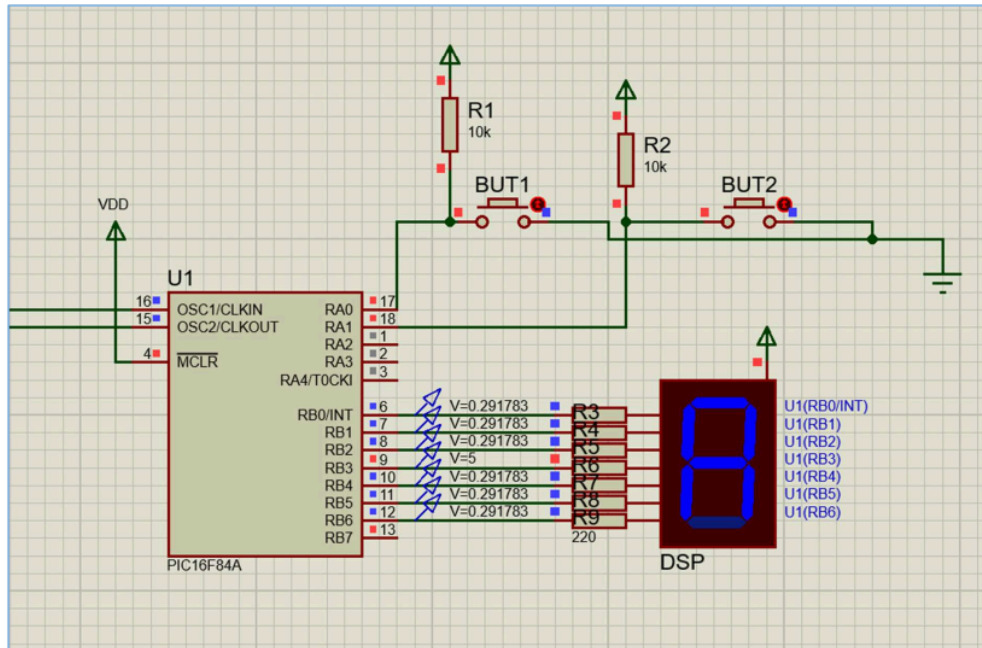


Figure 5: zooming of the circuit showing the display along with the operating voltage of each pin when A is displayed.

From the voltages probed the activated part of the display are responding to an input voltage of approximately 0.3V. Whilst this would seem counter intuitive it is because it is a common anode display, the power is coming in on the pin on top of the display going to the LED segments, the 0.3V is the voltage of the PIC output drawing current to activate the display. This means that the HIGH level logic at 5 Volts deactivates a segment of the display and low level logic voltage activates it. The common anode means that all the seven segments LEDs share is the power terminal on top of the display and the individual cathodes are connected to the resistors.^[11] With these seven segments each of which having two distinct states it can be worked out how many individual characters can be displayed.

$$\begin{aligned} \text{Number of distinct Characters} &= \text{Number of segments}^{\text{number of states}} \\ \text{Characters} &= 7^2 = 128 \{1\} \end{aligned}$$

Thus there are a total of 128 discrete characters that can be displayed.

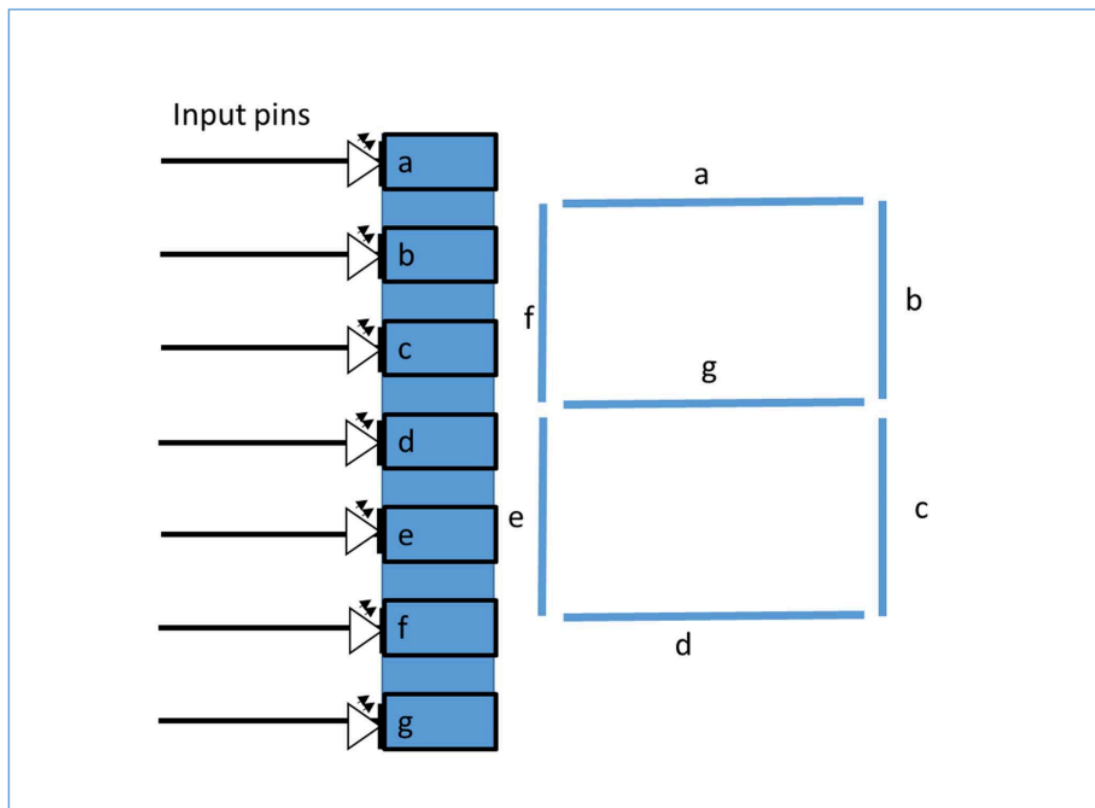


Figure 6: Cartoon of LED display locations.

For purposes of explanation the logic for figure 6 above shall be defined as, on = 1 and, off = 0. For a zero (0) to be displayed, all the input pins besides g will be on so the 7-digit output will be, 1111110. By using hex values instead the microcontroller outputs the desired logic without having to individually workout the on/off logic for each Character to be displayed.

Hex Value	Character	Hex Value	Character
0xC0	0	0x80	8
0xF9	1	0x90	9
0xA4	2	0x88	A
0xB0	3	0x83	b
0x99	4	0xC6	C
0x92	5	0xA1	d
0x82	6	0x86	E
0xF8	7	0x8E	F

Piezo sounder and Pulse width modulation.

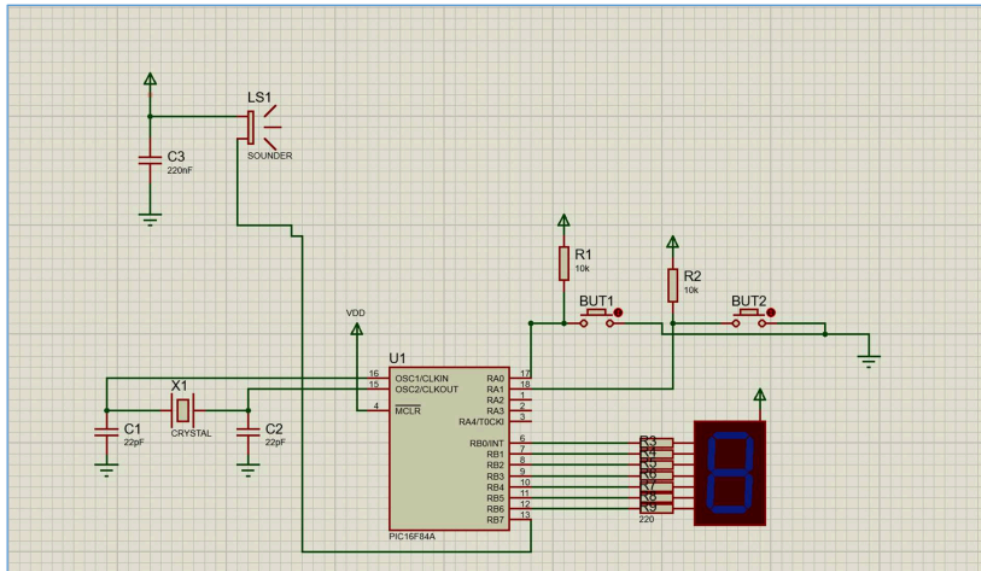


Figure 7: updated circuit diagram with the piezo sounder on LS1.

A piezo sounder as the name suggests makes use of the previously discussed piezo electric effect to drive a device that turns a digital signal into sound waves. Commonly these can be most easily found today being used in in-ear 'ear-bud/airpod' type headphones and mobile phone speakers. Whereas traditional audio equipment and headphones use a coil of copper to physically drive a diaphragm and it is the diaphragm itself that generates the sound where is for a piezo electric sounder and the electric signal will move the piezo electric material making up the diaphragm and the driver removing the need for a coil.

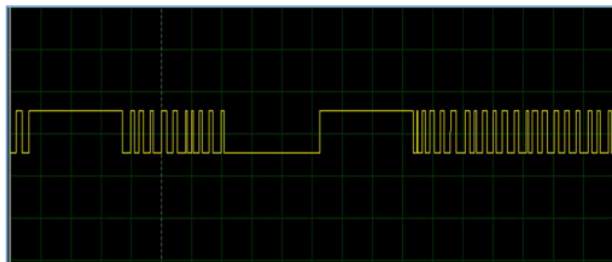


Figure 8: PWM signal sent through piezo sounder, with varying bit pattern.

To drive the piezo electric sounder and output a variety of sounds that the human ear can hear one must use the principle of pulse width modulation (PWM). A simple driving signal can be made by choosing a single pin in figure 7 (Q5) RB7. This is used as an output to the piezo sounder and by pulsing on off signals at different frequencies thus generating sound. For higher frequency notes, higher frequency pulses are required. Adding in an element

of randomness to the output pulses can achieve some very interesting sonic characteristics and has been used by artists, engineers and music producers to this effect for the best part of 80 years. A good example of a simple PWM signal common to electronic music is the bassline from Vanished by Crystal Castles, there is a fizzling sound on each bass note as the signal is squeezed and stretched.

The sounder can output specific notes, given the frequencies of musical notes are well documented, the values have been taken from Auditory Neuroscience below.^[12]

Note	Frequency (Hz)
C ₄	261.63
C [#] ₄ /D ^b ₄	277.18
D ₄	293.66
D [#] ₄ /E ^b ₄	311.13
E ₄	329.63
F ₄	349.23
F [#] ₄ /G ^b ₄	369.99
G ₄	392
G [#] ₄ /A ^b ₄	415.3
A ₄	440
A [#] ₄ /B ^b ₄	466.16
B ₄	493.88
C ₅	523.25

Figure 9: Notes of fundamental frequencies of the 4th to the 5th octave.

Given the above notes and frequencies this can be played by translating that to an output pin by activating the sounder every half period of the desired note. This results in moving the crystal membrane at the frequency of the note and so a soundwave of that note will propagate from the speaker. A half period is required as a full note oscillation is a full sinewave up and down as we only have 0 to 1 and no minus one the period is halved but the frequency of the note is maintained.

Section 2: Displaying the Hexabet

Hexadecimal 7-Segment

Before initialising anything, it is appropriate to reset the system using the reset vector RST following this, the bank selection bit in the status registry RPO RP1 will also need to be cleared. The next step is to initialise and set the appropriate data direction for the I/O ports, as the output ports to the 7-segment display is connected through the RBx ports we load the W register with zeros then load this into the Trisb register. For TRISB value of 0 tell the PIC that it will be outputting data and a value of 1 that it will be inputting data. If the buttons were to be used to tell the microcontroller when to start, set these as inputs and so TRISA is set to 1 in the same manner.

To stop all the hexadecimal characters trying to be displayed too quickly a delay subroutine needs to be defined. 1st we need to work out the instruction clock speed or the CPU cycle speed which is the frequency of the incoming clock signal divided by 4, as one instruction is run per 4 oscillator cycles in this case 5MHz.

$$\text{Processor Instruction Speed} = \frac{\text{Oscillator Frequency}}{4} = \frac{20\text{MHz}}{4} = 5\text{MHz} \{2.0\}$$

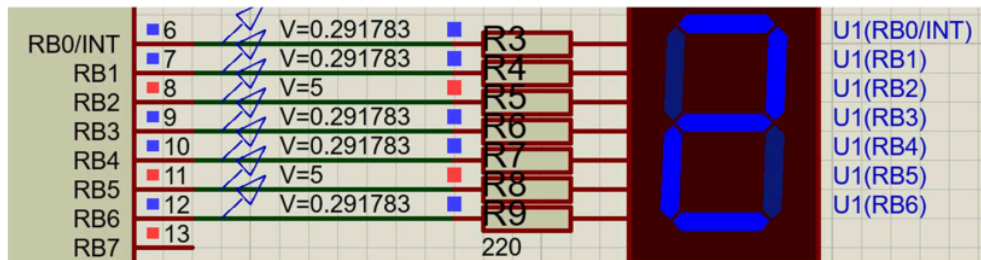
$$\text{Time Period} = \frac{1}{5\text{MHz}} = 0.2\mu\text{s} \{2.1\}$$

So every instruction cycle takes 0.2us to complete providing the given instruction only requires 1 cycle. This means that a delay can be programmed by making it count down several instruction cycles.

Next the most straightforward way to get the pic to display the hexadecimal alphabet is to create a main loop in which the hex value of the desired character is moved to the working w register and then to port B. For example, the hex value of the number 2 is A4, thereby with the two lines of code:

```
movlw 0xA4 ; A4 = 2
movwf portB
```

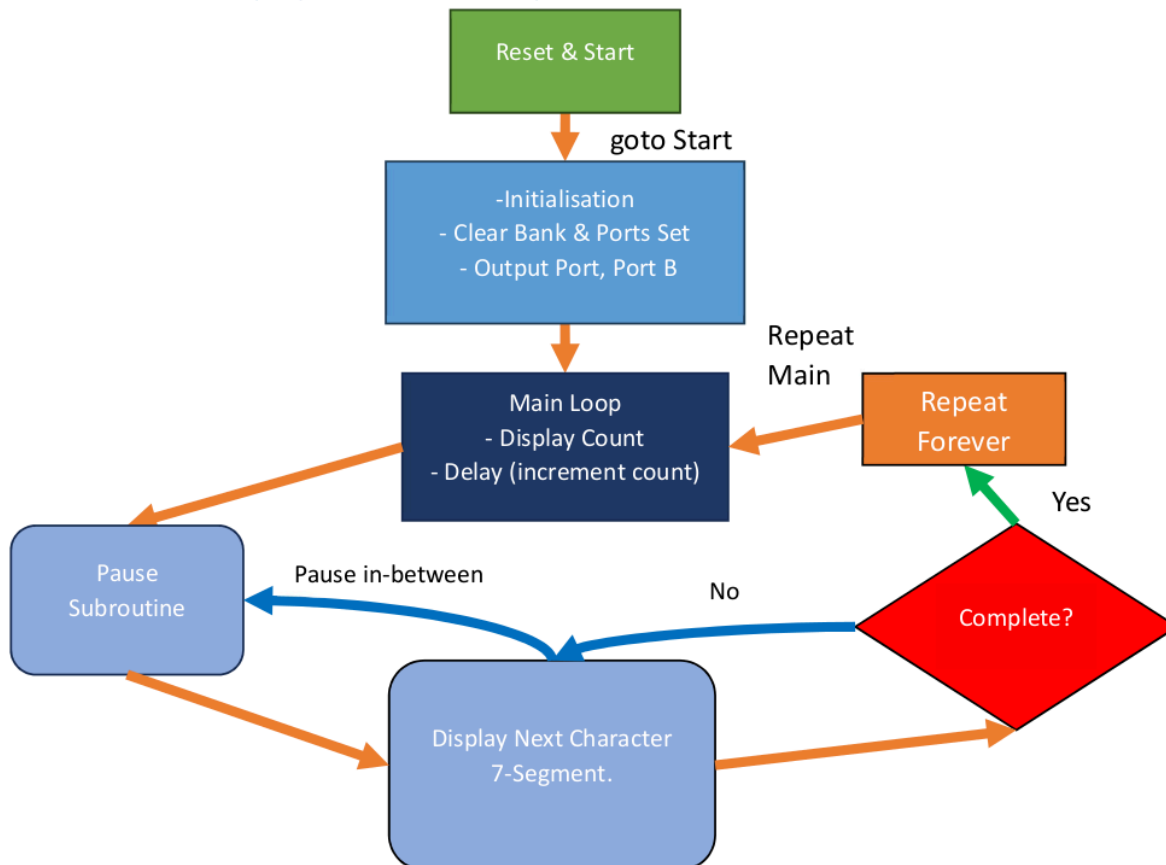
The PIC will then send the required low/high voltage logic to the appropriate segments:



Thus, displaying the character 2, in order to cycle through the hexadecimal alphabet that needs to be the aforementioned call pause after the given code above to stop all characters trying to be displayed at once. Then simply write the next character in the same manner for 3 this would be B0. This process is repeated for all the required characters in this case the number one through to letter F.

Section 3: Flow Charts

Task 1: Display Hexadecimal Alphabet



```

;=====
; Main.asm file generated by New Project wizard
;
; Created: Mon Feb 12 2018
; Edited: October 2024 by Jamie H-S
; Processor: PIC16F84A
; Compiler: MPASM (Proteus)
;=====

;=====
; DEFINITIONS
;=====

#include p16f84a.inc      ; Include register definition file

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

;=====
; VARIABLES
;=====

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

```

```

        ; Reset Vector
RST    code 0x0
        goto Start

;=====
; CODE SEGMENT
;=====

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 ;; in trisb 0 is
output 1 is input
        bcf      status,RP0      ; reselect bank 0

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

get_count
        movf     count, w        ; move count to W register
        movwf    portB          ; move W to port B to update the display
        addlw    1              ; add +1 to W
        movwf    count          ; save W in count

        call     pause          ; delay 200ms by subroutine

```

```
call    pause
call    pause
call    pause
call    pause           ; five pause executions equals ~ 1 second
```

; 0x prefix indicates hexadecimal format and preceeds the given hex value to be displayed

```
movlw 0xC0 ; C0 = 0
```

```
movwf portB
```

```
call pause
```

```
movlw 0xF9 ; |Hex Value, F9| = |1|, is displayed on 7 segment |
```

```
movwf portB
```

```
call pause
```

```
movlw 0xA4 ; A4 = 2
```

```
movwf portB
```

```
call pause
```

```
movlw 0xB0 ; B0 = 3
```

```
movwf portB
```

```
call pause
```

```
movlw 0x99 ; 99 = 4
```

```
movwf portB
```

```
call pause
```

```
movlw 0x92 ; 92 = 5
```

```
movwf portB
```

```
call pause
```

```
movlw 0x82 ; 82 = 6
```

```
movwf portB
```

```
call pause
```

```
movlw 0xF8 ; F8 = 7
```

```
movwf portB
```

```
call pause
```

```
movlw 0x80 ; 80 = 8
```

```
movwf portB
```

```
call pause
```

```
movlw 0x90 ; 90 = 9
```

```
movwf portB
```

```
call pause
```

```
movlw 0x88 ; 88 = A
```

```
movwf portB
```

```
call pause
```

```
movlw 0x83 ; 83 = b
```

```
movwf portB
```

```
call pause
```

```
movlw 0xC6 ; C6 = C
```

```
movwf portB
```

```
call pause
```

```
movlw 0xA1 ; A1 = d
```

```
movwf portB
```

```
call pause
```

```
movlw 0x86 ; 86 = E
```

```
movwf portB
```

```
call pause
```

```
movlw 0x8E ; 8E = F
```

```
movwf portB
```

```
call pause
```

```
goto get_count ; repeat forever
```

```
;-----
```

```
;subroutine pause: lets waste some time....
```

```
;description: two loops using "mCount" and "nCount" register files
```

```
; "mCount" is the outer loop counter (255 down to zero)
```

```
; "nCount" is the inner loop counter (255 down to zero)
```

```
;;-----
```

```
pause
```

```
movlw 0xff ; set w = 255 decimal
```

```
movwf mCount ; mCount = w: outer loop counter
```

```
loadN
```

```
movlw 0xff ; set w = 255 decimal
```

```
movwf nCount ; nCount = w: inner loop counter
```

```
decN
```

```
decfsz nCount, f ; nCount--
```

```
goto decN ; if nCount != 0 then repeat nCount--
```

```
decfsz mCount, f ; else decrement Count
```

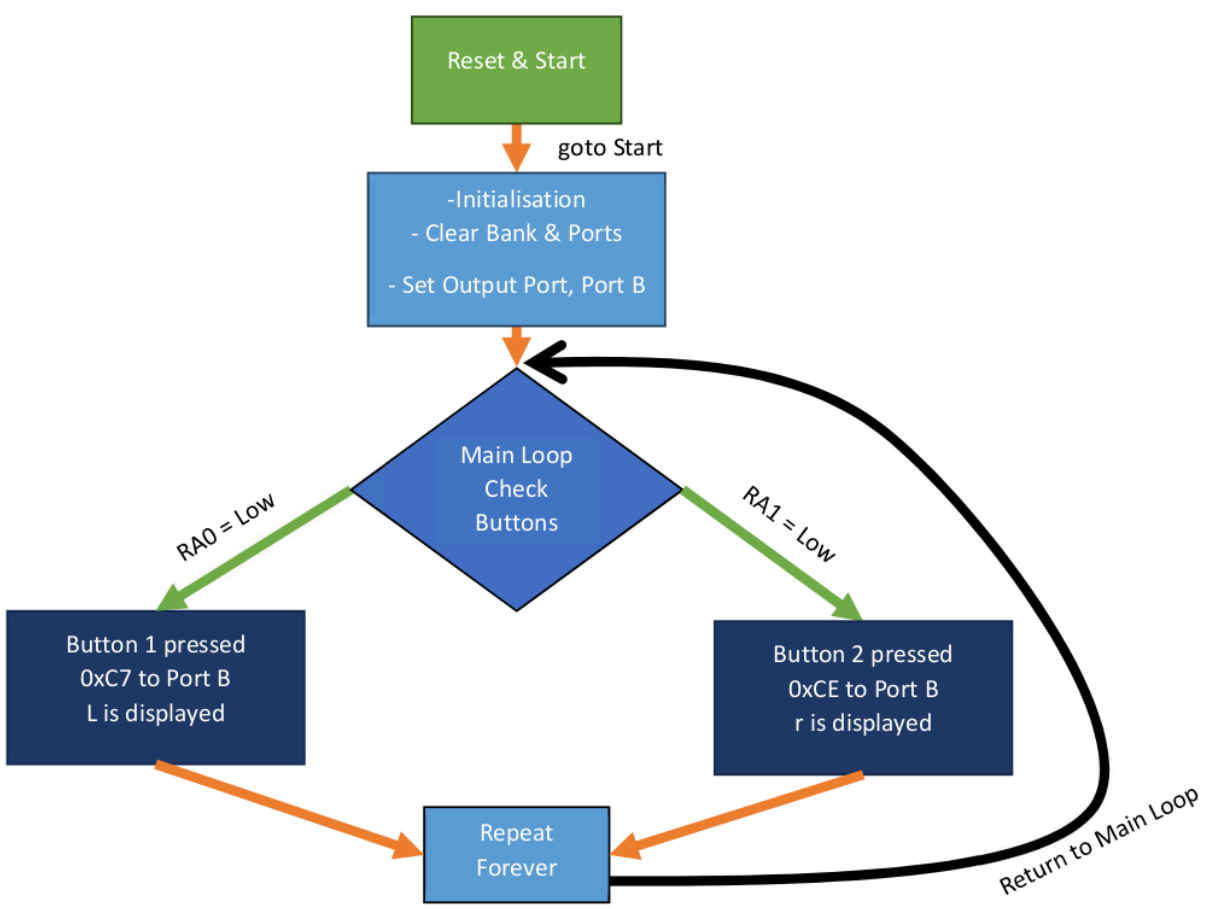
```
goto loadN ; if mCount != 0 then
```

```
; reload nCount to 255 and decrement
```

```
return ; else exit subroutine
```

END

Task 2: Buttons Display L and r



;Jamie H-S

#include p16f84a.inc

portB equ 0x06 ;port B address

trisb equ 0x06 ;trisB address in register bank 1

status equ 0x03 ;

RP0 equ 5 ;bit 5 of status register

RP1 equ 6 ;bit 6 of status register

; Reset Vector

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

movlw 0x00

movwf trisb

bcf status,RP0

MainLoop:

```

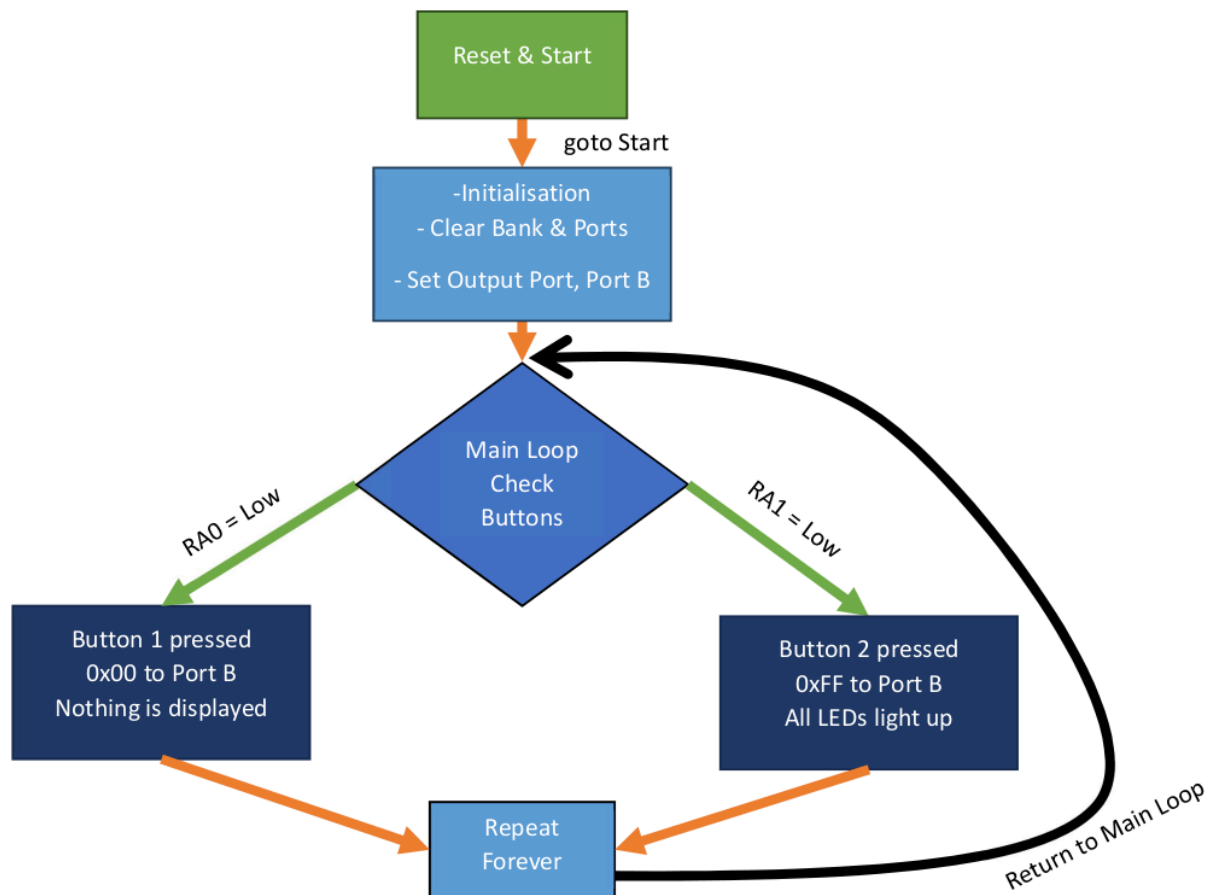
    btfss    PORTA, 0           ; is button BUT1 pressed (input = RA0)
    movlw    0xC7               ; displays L if BUT1, L for left
    btfss    PORTA, 1           ; is BUT2 pressed (RA1)
    movlw    0xCE               ; displays r if, r for right
    movwf    PORTB              ; stores either r or L in working register and remains
displayed

    goto     MainLoop           ;repeats loop

END

```

Task 3: Buttons Display all or None



CODE:

```
;Jamie H-S
#include p16f84a.inc
portB  equ 0x06
trisb  equ 0x06

status equ 0x03
```

```
RP0    equ 5
```

```
RP1    equ 6
```

```
    ; Reset Vector
```

```
RST    code 0x0
```

```
    goto Start
```

```
PGM    code
```

```
Start
```

```
    bcf        status,RP0
```

```
    bcf        status,RP1
```

```
    bsf        status,RP0
```

```
    movlw 0x00
```

```
    movwf trisb
```

```
    bcf        status,RP0
```

```
MainLoop:
```

```
    btfss    PORTA, 0          ; is button BUT1 pressed (input = RA0)
```

```
    movlw 0x00                ; displays nothing if BUT1 pressed
```

```
    btfss    PORTA, 1          ; is BUT2 pressed (RA1)
```

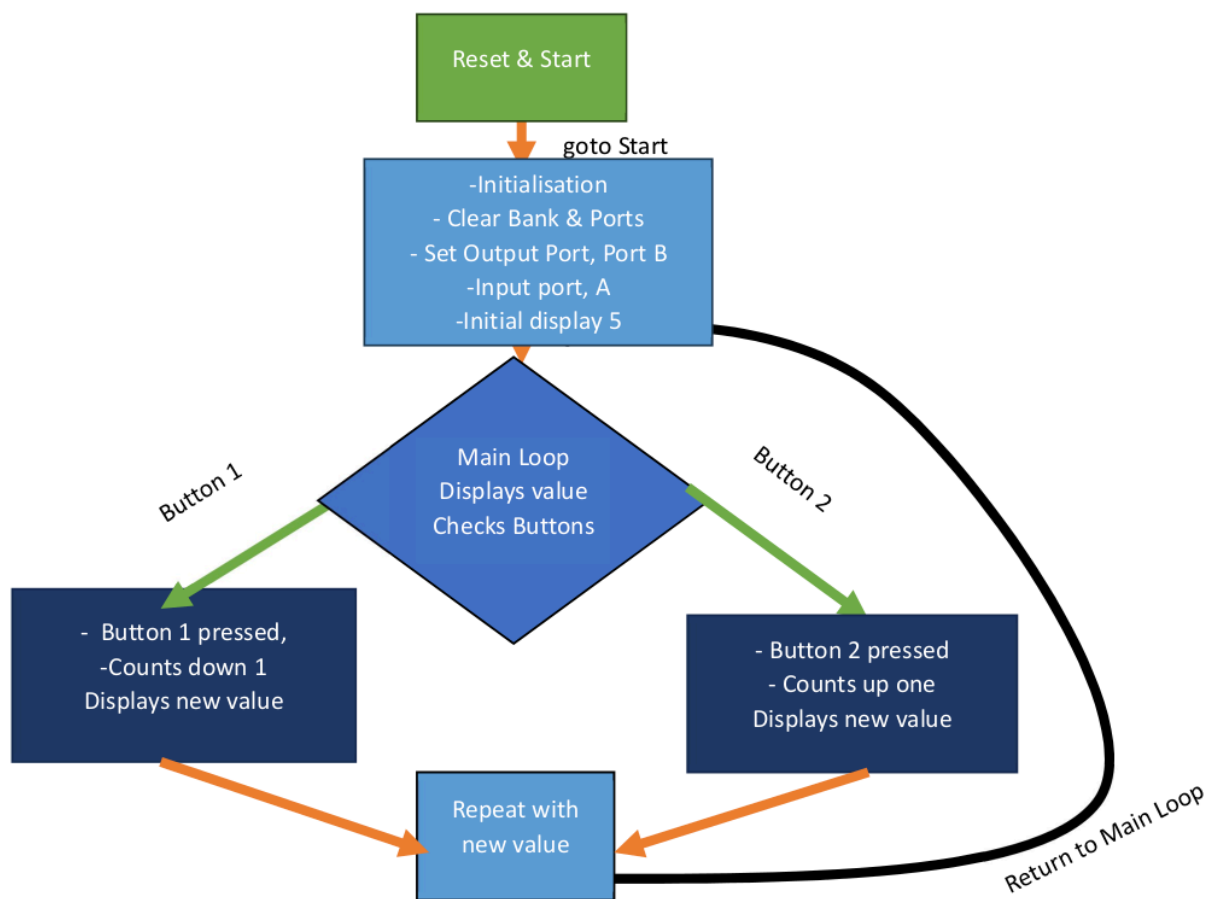
```
    movlw 0xFF                ; displays All segments
```

```
    movwf PORTB                ; stores either on or off in working register and remains  
displayed
```

goto MainLoop ;repeats loop

END

Task 4: Buttons count up and down:



Code:

; Jamie H-S

#include p16f84a.inc

porta equ 0x05

trisa equ 0x05

portb equ 0x06

trisb equ 0x06

status equ 0x03

RP0 equ 5

RP1 equ 6

count equ 0x0c ; counter 0 to 255

nCount equ 0x0d ; pause routine inner loop counter

mCountequ 0x0e ; pause routine outer loop counter

; Reset Vector

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
        movlw 0x00          ; load w with 0
        movwf trisb          ; load it into trisb to make portb all outputs ;; in trisb 0 is
output 1 is input
        bcf          status,RP0      ; reselect bank 0

        clrf portb          ; clear all lines low
        clrf count          ; initialize counter to 0
```

five: ; starts at 5 holding button 1 counts down button 2 counts up

```
        movlw 0x92 ;5
```

```
        movwf portb
```

```
        call pause
```

```
        call pause
```

```
        call pause
```

```
        btfss porta,0
```

```
        GOTO four
```

```
        btfss porta,1
```

```
        goto six
```

```
        btfsc porta,0 ; if nothing pressed stays on 5 'latching loop'
```

```
        btfsc porta,1
```

```
        GOTO five
```

zero:

movlw 0xC0 ;0

movwf portb

call pause

btfs porta,0

GOTO nine

btfs porta,1

goto one

btsc porta,0 ; if nothing pressed stays on 0 'latching loop'

btsc porta,1

GOTO zero

one:

movlw 0xF9 ; displays 5 on 7 seg

movwf portb

call pause

btfs porta,0 ; checks button 1

GOTO zero

btfs porta,1 ;checks button 2

goto two

```
    btfsc porta,0 ; if nothing pressed stays on 1 'latching loop'  
    btfsc porta,1  
    GOTO one
```

two:

```
    movlw 0xA4 ;5  
    movwf portb
```

```
    call pause
```

```
    btfss porta,0  
    GOTO one
```

```
    btfss porta,1  
    goto three
```

```
    btfsc porta,0 ; if nothing pressed stays on 2'latching loop'  
    btfsc porta,1  
    GOTO two
```

three:

```
    movlw 0xB0 ;5  
    movwf portb  
    call pause
```

```
    btfss porta,0  
    GOTO two
```

```
    btfss porta,1
```

```
    goto four
```

```
    btfsc porta,0 ; if nothing pressed stays on 3 'latching loop'
```

```
    btfsc porta,1
```

```
    GOTO three
```

```
four:
```

```
    movlw 0x99 ;5
```

```
    movwf portb
```

```
    call pause
```

```
    btfss porta,0
```

```
    GOTO three
```

```
    btfss porta,1
```

```
    goto five
```

```
    btfsc porta,0 ; if nothing pressed stays on 4 'latching loop'
```

```
    btfsc porta,1
```

```
    GOTO four
```

```
six:
```

```
    movlw 0x82 ;5
```

```
    movwf portb
```

```
    call pause
```

```
    btfss porta,0
```

GOTO five

btfs porta,1

goto seven

btfs porta,0 ; if nothing pressed stays on 6 'latching loop'

btfs porta,1

GOTO six

seven:

movlw 0xF8 ;5

movwf portb

call pause

btfs porta,0

GOTO six

btfs porta,1

goto eight

btfs porta,0 ; if nothing pressed stays on 7 'latching loop'

btfs porta,1

GOTO seven

eight:

movlw 0x80 ;5

movwf portb

call pause

```
    btfss porta,0
```

```
    GOTO seven
```

```
    btfss porta,1
```

```
    goto nine
```

```
    btfsc porta,0 ; if nothing pressed stays on 8 'latching loop'
```

```
    btfsc porta,1
```

```
    GOTO eight
```

```
nine:
```

```
    movlw 0x90 ;5
```

```
    movwf portb
```

```
    call pause
```

```
    btfss porta,0
```

```
    GOTO eight
```

```
    btfss porta,1
```

```
    goto zero
```

```
    btfsc porta,0 ; nothing pressed stays on 5 'latching loop'
```

```
    btfsc porta,1
```

```
    GOTO nine
```

```
pause
```

```
    movlw 0xff          ; set w = 255 decimal
```

```

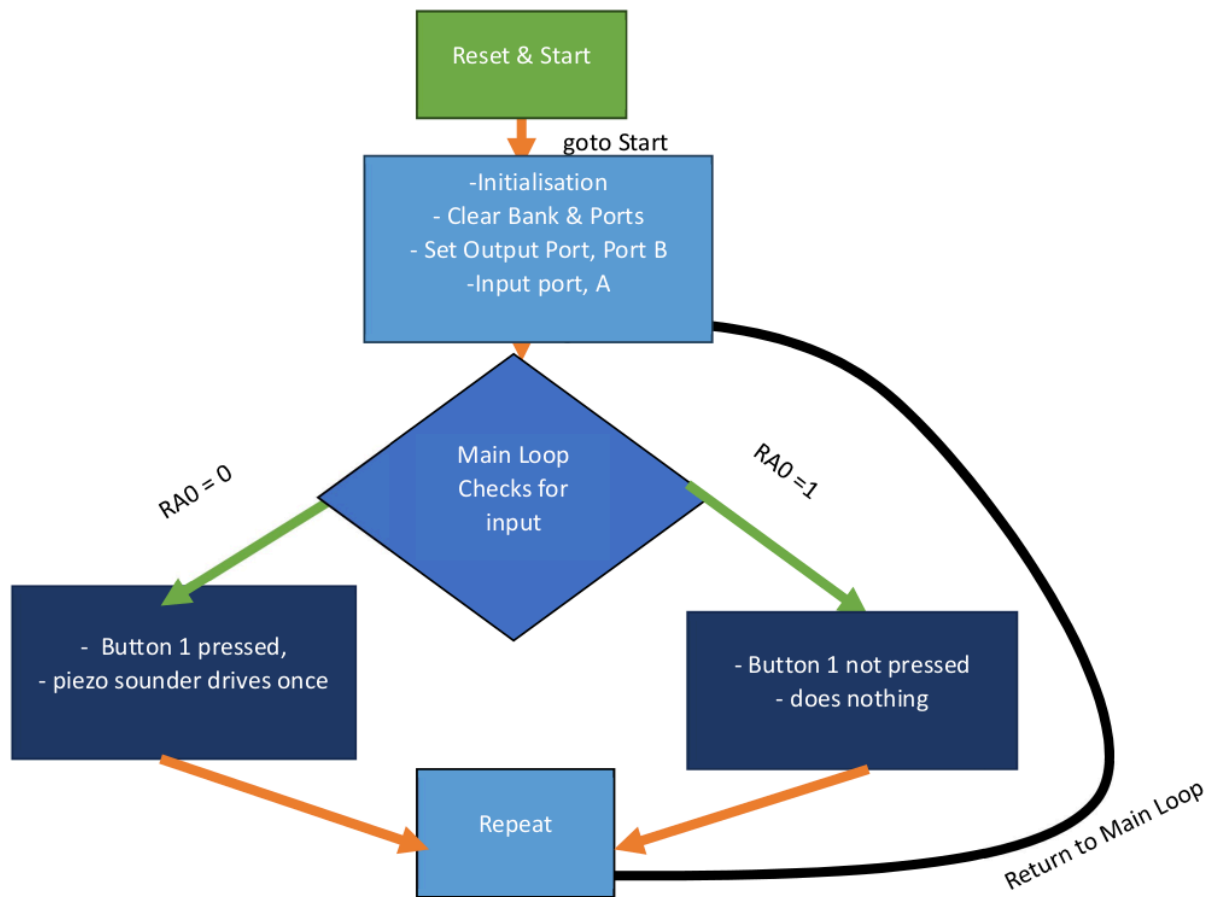
        movwf mCount      ; mCount = w: outer loop counter
loadN
        movlw 0xff        ; set w = 255 decimal
        movwf nCount      ; nCount = w: inner loop counter
decN
        decfsz nCount, f   ; nCount--
        goto    decN       ; if nCount != 0 then repeat nCount--

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

END

```

Task 5: Sounder



Code:

```
;Jamie H-S  
#include p16f84a.inc
```

```
porta equ 0x05  
trisa equ 0x05
```

```
portb equ 0x06
```

```
trisb equ 0x06
```

```
status equ 0x03
```

```
RP0 equ 5
```

```
RP1 equ 6
```

```
count equ 0x0c ; counter 0 to 255
```

```
nCount equ 0x0d ; pause routine inner loop counter
```

```
mCountequ 0x0e ; pause routine outer loop counter
```

```
; Reset Vector
```

```
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
```

```
movlw 0x00 ; load w with 0
```

```
movwf trisb ; load it into trisb to make portb all outputs ;; in trisb 0 is  
output 1 is input
```

```
bcf status,RP0 ; reselect bank 0
```

```
clrf portb ; clear all lines low
```

```
clrf count ; initialize counter to 0
```

mainloop:

movlw 0xFF

movwf portb

btfss porta,0 ; skips if RA0 = 1

GOTO sounder

btfsc porta,1 ; if nothing pressed stays on 5 'latching loop'

btfsc porta,0

goto mainloop

sounder:

movlw 0x7F

movwf portb

btfss porta,0

goto sounder

btfsc porta,0

goto mainloop

pause

movlw 0xff ; set w = 255 decimal

movwf mCount ; mCount = w: outer loop counter

loadN

movlw 0xff ; set w = 255 decimal

movwf nCount ; nCount = w: inner loop counter

decN

decfsz nCount, f ; nCount--

goto decN ; if nCount != 0 then repeat nCount--

decfsz mCount, f ; else decrement Count

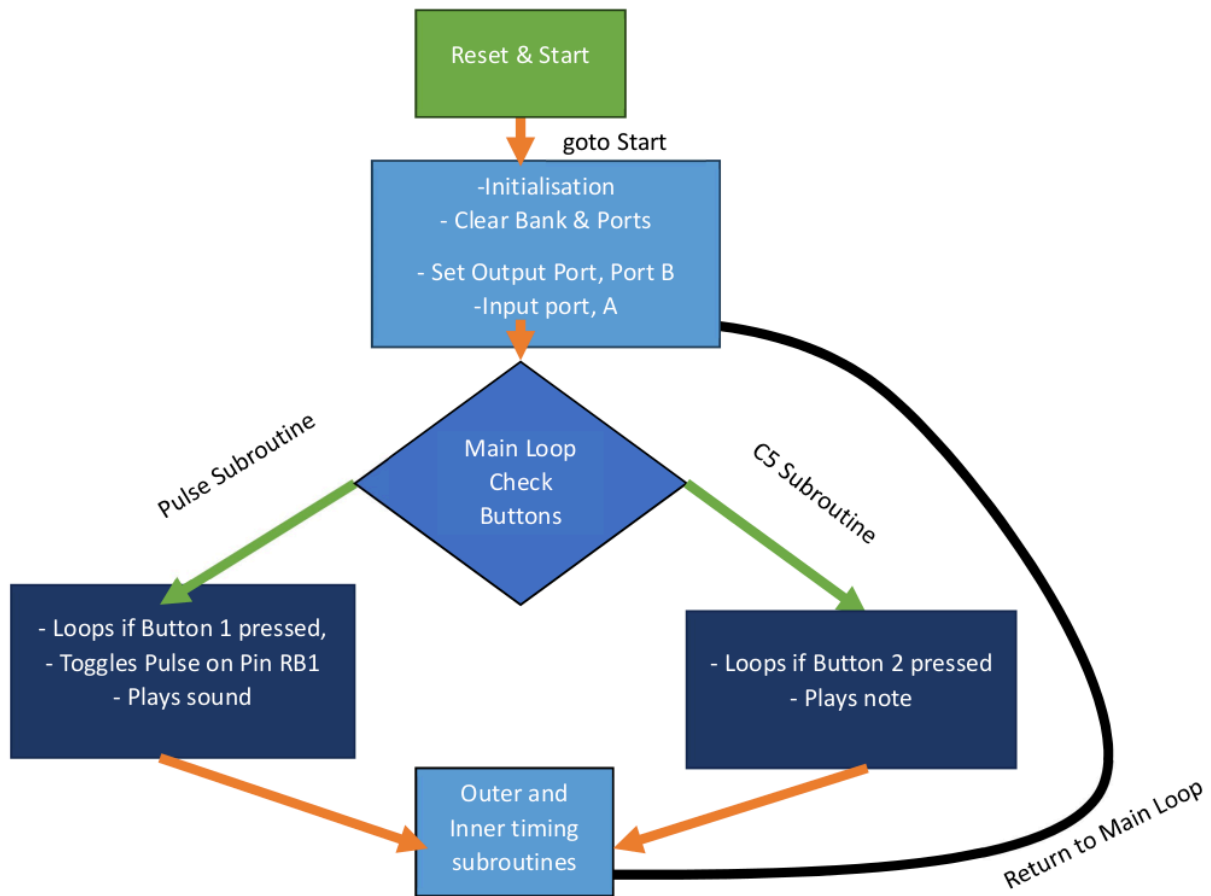
goto loadN ; if mCount != 0 then

; reload nCount to 255 and decrement

return ; else exit subroutine

END

Task 6: PWM Signal



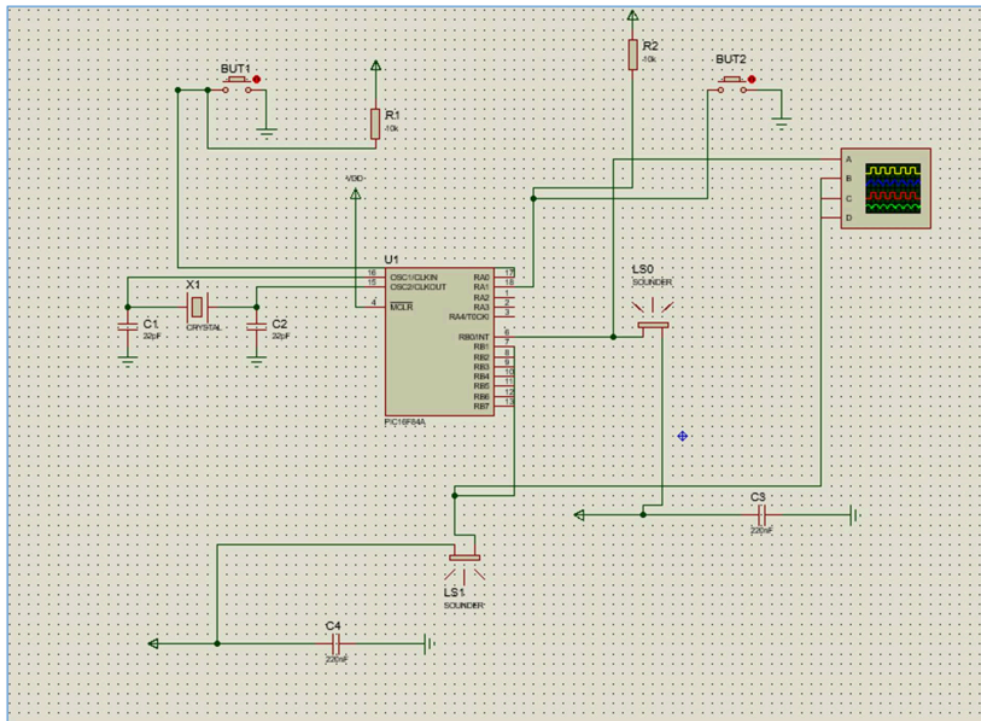


Figure 10: Updated Circuit

Code:

; Jamie H-S

```
#include p16f84a.inc
```

```
porta    equ    0x05
```

```
trisa    equ    0x05
```

```
portb    equ    0x06
```

```
trisb    equ    0x06
```

```
status   equ    0x03
```

```
RP0      equ    5
```

```
RP1      equ    6
```

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

```
;reset vector
```

```
org    0x0000
goto    Start
```

```
Start:
```

```
bcf    status, RP0    ; Bank 0
bcf    status, RP1

bsf    status, RP0    ; Switch to Bank 1
movlw  0x00    ; all PORTB pins as outputs
movwf  trisb
bcf    status, RP0    ; back to Bank 0

clrf   portb    ; clear all lines low
clrf   count    ; initialize counter to 0
```

```
mainloop:
```

```
; PWM attempt
btfss   porta,1    ; bit test f skip if set port RA1
call    C5    ; RA1 = 0 button 2 pressed C5 note plays
```

```
btfss    porta, 0
        call    Pulse ; press button 1 for some sonic modulation
```

```
        goto    mainloop    ; loop forever
```

C5:

```
btfss    porta,1    ; bit test f skip if set port RA1
        goto    C5_loop    ; if RA1=0 which is on then button is pressed so note can be played
        return    ; if RA1=1 exits PWM routine
```

C5_loop:

; is RB1 high or

```
bsf      portb,0
```

```
call     C5halfPeriodDelay
```

; is RB1 low

```
bcf      portb,0
```

```
call     C5halfPeriodDelay
```

```
btfss    porta,1 ; checks RA0
```

```
goto     C5_loop
```

```
return
```

C5halfPeriodDelay: ; $(1/523) * 0.5 = T1/2$ this gives a $T1/2$ of around 960us roughly corresponding to a
;C5 note,

;I hope... sounds a little high think I broke it

```
    movlw 0xF0 ;240
    movwf nCount
    movlw 0x04 ;4
    movwf mCount
```

Pulse:

```
    btfss porta, 0
    goto Pulse_Loop
    return
```

Pulse_Loop:

```
    bsf  portb,1
    call PulseDelay
```

```
    bcf  portb,1
    call PulseDelay
```

```
    btfss porta, 0 ; checks RA0
    goto Pulse_Loop
    return
```

PulseDelay: ; 240 and 1

```
    movlw 0xF0
    movwf nCount
    movlw 0x01
    movwf mCount
```

;; tried to replicate C5 with pulse having different properties it did work but made C5 way
;higher pitched... definitely not C5 anymore

delayOuter:

```
    decfsz nCount, f  
    goto  delayOuter_continue
```

```
    movlw 0xF0  
    movwf nCount  
    decfsz mCount, f  
    goto  delayOuter  
    return
```

delayOuter_continue:

```
    goto  delayOuter
```

end

Bibliography

- [1] D. Ahuja¹ and N. Chaudhary¹, "Programmable Logic Controller," *International Journal of Information and Computer Science*, vol. 1, pp. 115–120, Aug. 2012.
- [2] Microchip Technology Inc., "PIC16F84A Data Sheet," *Microchip.com*, 2024. <https://www.microchip.com/en-us/product/PIC16F84A#Documentation> (accessed Dec. 20, 2024).
- [3] M. Alba, "Vacuum Tubes: The World Before Transistors - Engineering.com," *Engineering.com*, Jan. 19, 2018. <https://www.engineering.com/vacuum-tubes-the-world-before-transistors/> (accessed Dec. 22, 2024).
- [4] "An Introduction to Microchips: Types & Manufacturing | Ansys," *Ansys.com*, 2024. <https://www.ansys.com/en-gb/simulation-topics/what-is-a-microchip> (accessed Dec. 22, 2024).
- [5] Micro Electronics, *1977 Data Catalog*. Century Electronics: General Instrument Corporation, 1977, p. 4.
- [6] AMD, *AMD RYZEN PRO7000 WX-SERIES PROCESSORS*, 1st ed. AMD, 2023, p. 4. Accessed: Dec. 22, 2024. [Online]. Available: <https://www.amd.com/content/dam/amd/en/documents/products/processors/ryzen-threadripper/amd-ryzen-threadripper-pro-workstation-raising-the-bar-white-paper.pdf>
- [7] R. A. Serway and J. W. Jewett, *Physics for Scientists and Engineers with Modern, Chapter 39*. Cengage Learning, 2009, p. 1153.
- [8] C. Yang and K. Youcef-Toumi, "Principle, implementation, and applications of charge control for piezo-actuated nanopositioners: A comprehensive review," *Mechanical Systems and Signal Processing*, vol. 171, p. 108885, May 2022, doi: <https://doi.org/10.1016/j.ymssp.2022.108885>.
- [9] L. Guo, S.-T. Han, and Y. Zhou, "Electromechanical coupling effects for data storage and synaptic devices," *Nano Energy*, vol. 77, p. 105156, Nov. 2020, doi: <https://doi.org/10.1016/j.nanoen.2020.105156>.
- [10] S. Bible and Microchip Technology Inc, "AN826 Crystal Oscillator Basics and Crystal Selection for PIC and PICmicro® Devices," p. 4, 2002.
- [11] "Common Cathode vs. Common Anode: A Detailed Comparison," *Premteco*, Jun. 29, 2024. <https://www.ptcled.com/academy/common-cathode-vs-common-anode.html> (accessed Jan. 03, 2025).
- [12] "Fundamental frequencies of Notes in Western Music | Auditory Neuroscience," *Auditoryneuroscience.com*, 2025. <https://auditoryneuroscience.com/pitch/fundamental-frequencies-notes-western-music> (accessed Jan. 08, 2025).

MPI Assignment

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

75/100

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

GRADING FORM: GJ_2019

JAMIE HAMILTON-SWINGEWOOD**376****ENGAGEMENT WITH LITERATURE**

Good range of sources referenced.
Make sure to use the exact IEEE referencing convention.

75**KNOWLEDGE AND UNDERSTANDING**

Good K&U demonstrated, but be careful not to wander off into areas that are a bit too irrelevant to the subject at hand! (no matter how interesting they are!!)

78**COGNITIVE AND INTELLECTUAL**

Some interesting creative approaches in solving various issues.

70

PRACTICAL APPLICATION

 Not sure the PWM code will do what was required (as per the video clip example provided), but a good attempt at the other sections and the circuit.

73

TRANSFERABLE SKILLS

 Very well written and presented

80