Using and Programming the 7 Segment Display Prototype Circuit

Micros, Peripherals and Interfaces
ACDA4004
Bryan Skeen (2307828)

Abstract

Microcontrollers are powerful devices and provide a simple way to interface between inputs and outputs. Exploring the use of microcontrollers, the Assignment Board circuit has been used to understand how microcontrollers can be applied within a circuit and provide control of how the circuit operates. The Assignment Board circuit contains the following component: a PIC1fF84A microcontroller, voltage regulator, crystal oscillator, input buttons, a piezo buzzer with driving circuitry and a 7-segment display. Each of these are discussed to understand their operation and potential use with the circuit. Of particular focus is interfacing with the 7-segment display where a method has been devised by directly connecting seven pins of a port of the microcontroller directly to the seven cathode pins of the 7-segment display. Following understanding the Assignment Board circuit, six programs have been created in PIC assembly to control the circuit in various ways. The programs create achieve the following operations: displaying a counting value to the 7-segment display, using the button inputs and outputting corresponding glyphs to the 7-segment display, latching the button input states, displaying a counting value on the 7-segment display and increasing or decreasing the value with the two buttons, sounding the piezo buzzer and sounding the piezo buzzer with a changing signal.

Table of Contents

Introduction 5
PIC16F84A Assignment Board Circuit 5
- PIC16F84A 5
- Voltage Regulator 6
- Crystal Oscillator 6
- Piezo Buzzer 8
- Input Buttons 9
- Seven Segment Display 10

Interfacing with the 7-Segment Display 11
Programming Circuit Operations 12
- Delays 13
- Interrupts 14
- Assignment Board PIC Assembly Programs 15
  - Counting on 7-Segment Display with Modified Count.asm 15
  - Using Button Inputs 16
  - Latching Button Inputs 17
  - Counting Up and Down with Buttons 18
  - Sounding A Piezo Buzzer 19
  - Changing Piezo Buzzer 20

Conclusion 21
References 21
Appendix A: Counting on Seven Segment Display with Modified Count.asm - PIC Assembly Code 23
Appendix B: Using Button Inputs - PIC Assembly Code 25
Appendix C: Latching Button Inputs - PIC Assembly Code 27
Appendix D: Counting Up and Down with Buttons - PIC Assembly Code 29
Appendix E: Sounding a Piezo Buzzer - PIC Assembly Code 32
Appendix F: Sound Piezo Buzzer with Changing PWM Signal - PIC Assembly Code 35

Figure 1. Schematic of the Assignment Board circuit created in Proteus 8. 5
Figure 2. The 7805 regulator from the Assignment Board circuit with supporting capacitors on the input and output. 6
Figure 3. Equivalent electrical circuit to model a crystal oscillator. 7
Figure 4. Graphs showing the reactance and absolute reactance against frequency curve of a crystal oscillator 7
Figure 5. The internal components of a piezoelectric buzzer 8
Figure 6. Piezoelectric buzzer with supporting circuitry used within the Assignment Board circuit. ... 8
Figure 7. Input buttons connected to 2 pins of Port A of the PIC16F84A in the Assignment Board circuit with pull-up resistors connected to each button. 9
Figure 8. Signal seen at a pulled-up input from a button being pressed and experience bounce upon pressing and releasing 9
Figure 9. Image of the HDSP-7801 Broadcom Green 7-Segment LED Display component package. 10
Figure 10. Internal circuit of the HDSP-7801 common anode seven segment display with decimal point 10
Figure 11. Section of the Assignment Board circuit showing the connection between the PIC16F84A and 7-segment display 11
Figure 12. Flowchart showing the logical flow of the delay routine used to created delays on the PIC16F84A. 13
Figure 13. Program flow showing jump to a service routine following interrupt signal 14
Figure 14. Flowchart illustrating the logical operation of the Counting on the 7-Segment display PIC assembly code. 15
Figure 15. Flowchart illustrating the logical operation of the Using Button Inputs PIC assembly code. 16
Figure 16. Flowchart illustrating the logical operation of the Counting on the Latching Button Inputs PIC assembly code. 17
Figure 17. Flowchart illustrating the logical operation of the Counting with Button Inputs PIC assembly code. 18
Figure 18. Flowchart illustrating the logical operation of the Sounding the Piezo Buzzer PIC assembly code. 19
Figure 19. Flowchart illustrating the logical operation of the Changing the Sound of the Piezo Buzzer PIC assembly code. 20

Table 1. The different characters to be displayed on the 7-segment display, with the required segments to be illuminated and the Port B bits 0-6 configuration to achieve this. 12

Introduction

Producing circuits to perform desired tasks typically involves interfacing between inputs and outputs. A simple method for achieving this is through the use of a microcontroller that is programmed to process inputs, perform operations and output necessary results. To explore using microcontrollers to control and interface inputs and outputs (I/O), an example circuit is presented containing a microcontroller, various I/O devices and other supporting circuitry. This report analyses the Assignment Board circuit at component level, followed by the use of the microcontroller within the circuit and an exploration on how it can control different operations.

PIC16F84A Assignment Board Circuit

For a platform to explore the use of microcontrollers and interfacing with different I/O components, the Assignment Board circuit has been used and is shown in *Figure 1*. To understand how the circuit can be controlled, each component has been analysed to show their purpose within the circuit and ability to be interfaced with.

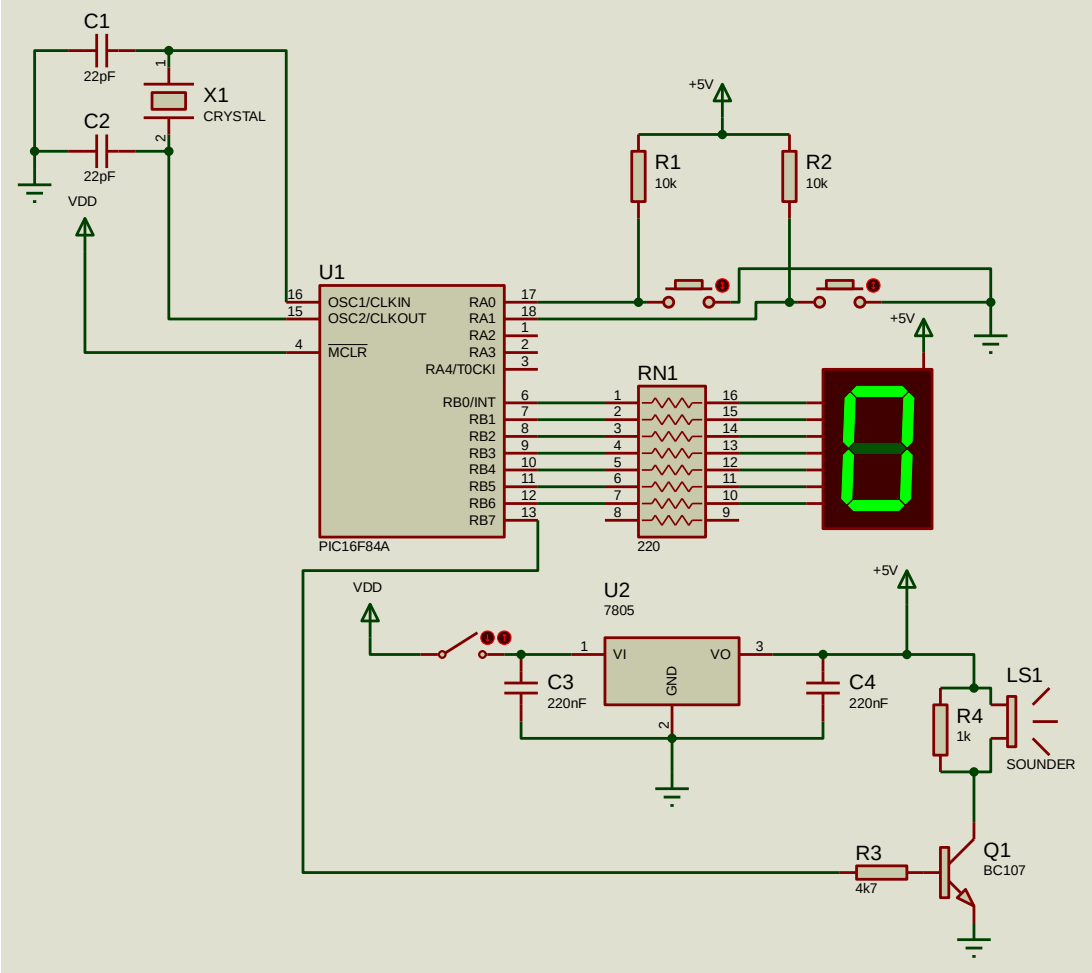


Figure 1. Schematic of the Assignment Board circuit created in Proteus 8.

PIC16F84A

Interfacing between I/O's and having programmable control of the Assignment Board circuit, is provided by a PIC16F84A microcontroller. Microcontrollers are single integrated circuits which consist of a processor, volatile and non-volatile memory, supporting circuitry such as oscillators and often various peripherals which could include analogue-to-digital converters, component drivers, or

communication interfaces like I^2C [1, pp. 1053-1096]. The PIC16F84A microcontroller used incorporates these components with having the following features [2]:

- 1024 program words,
- 68 bytes of general-purpose registers,
- 64 bytes of Electrically Erasable Read-Only Memory (EEPROM),
- 13 input/output pins across 2 ports,
- And an 8-bit timer.

Controlling the circuit with the PIC16F84A, it must be loaded with a program listing the instruction for it to carry out. PIC microcontroller's use a reduced instruction set computer (RISC) processor architecture meaning that there are only is a small set of optimised instruction taking 1 to 2 instruction cycles to complete. RISC architecture provides a number of benefits which lend itself to use within the PIC microcontrollers with its high energy efficiency and reduced complexity requiring less transistors resulting in a smaller processor. In the mid-range PIC family, of which the PIC16F84A resides, the processor has 35 instructions implemented that can be combined to produced programs which control the circuit [2] [3].

Voltage Regulator

Within the Assignment Board circuit there is a 7805 5V voltage regulator, shown in *Figure 2*. This component works to provide and maintain a fixed and stable output voltage, without impact from any changes to the input voltage. Having this function means that voltages from power supplies can be controlled to ensure that an input voltage to a circuit is stable and falls within a compatible range [4]. A 7805 regulator example is the LM7805 by Texas Instruments [5] which can take input voltages from 7.5V to 20V and output a constant 5V. This regulator can be driven in various configurations, but for the Assignment Board circuit it is operated with a fixed output voltage. Configuring the regulator in this way involves connecting the input pin to the power supply, the output pin to the circuit and the ground pin to ground. It is recommended to add additional bypass capacitors on the input and output to help aid with stability of the regulator, with the input capacitor being particularly important if regular is distant from the power supply. Bypass capacitors offer this additional stability due to removing any unwanted noise, through creating a low impedance route to ground for high frequency signals, and preventing any voltage drops or spikes by storing charge and resisting changes to the voltage level [6].

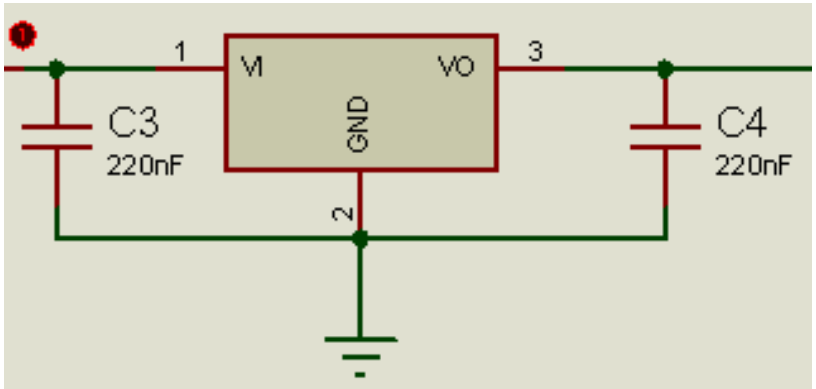


Figure 2. The 7805 regulator from the Assignment Board circuit with supporting capacitors on the input and output.

Crystal Oscillator

For the PIC16F84A to operate, an external oscillator is required to drive a clock signal to control the timing of executed instruction. There are different oscillating circuits which can be used to drive the clock signal for the PIC16F84A such as a *RC* oscillator or a crystal resonator, the latter is used within the Assignment Board circuit. A crystal oscillator is a piece of piezo-electric material which is cut to

vibrate at a specific frequency and have a special property where changes in the strain of the material (experienced when the crystal vibrates) generates voltages and vice versa. This means the mechanical oscillations of the crystal while vibrating produces a changing voltage across the crystal at the same frequency of the oscillations, which can in turn drives vibrations in the crystal. With this property of the crystal it results in it acting as an electrical resonator which can be modelled with an equivalent electrical circuit, shown in *Figure 3* [1, pp. 443-450].

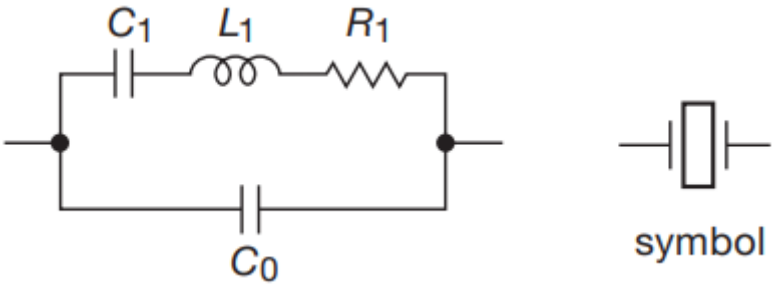


Figure 3. Equivalent electrical circuit to model a crystal oscillator [1, p. 444].

There are two operating modes of a crystal oscillator: series resonance and parallel resonance. These modes come about from the changes in the reactance of the crystal with frequency as is modelled by the equivalent circuit and shown in *Figure 4*. Series resonance occurs when the reactance of the crystal is at 0Ω and parallel resonance occurs at the point when the reactance is at its peak. In parallel mode additional load capacitance can be added to the crystal to tune the parallel resonant frequency; the load capacitance value is typically given by the crystal manufacturer for operation in parallel mode [7]. For the PIC16F84A the crystal is required to be operated in parallel resonance mode with it only having the drive circuitry for parallel cut crystals [2].

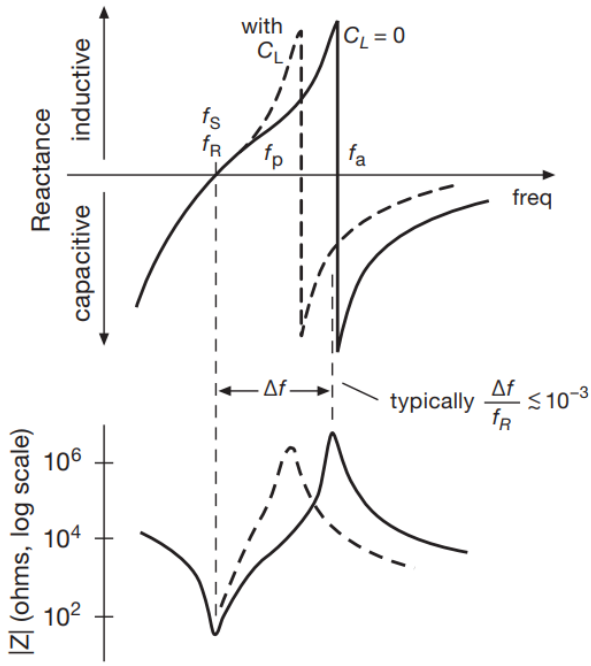


Figure 4. Graphs showing the reactance and absolute reactance against frequency curve of a crystal oscillator. The series resonant frequency is marked at fs, the parallel resonant frequency is marked at fa with no additional load capacitance and fp marks the result of the parallel resonant frequency with additional load capacitance [1, p. 444].

Piezo Buzzer

To produce audio outputs, a piezo buzzer is included in the circuit. This type of buzzer works by using a piece of piezoelectric material with attached electrodes, similar to that discussed in the Crystal Oscillator section. By applying a changing voltage at the electrodes, it causes oscillating deformations of the crystal, resulting in the generation of acoustic waves at the same frequency of the driving signal [8]. Using a digital output pin of a PIC16F84A to control the buzzer, requires creating a changing signal using the two output states available: 5V (logic 1) and 0V (logic 0). Oscillating between the two states produces a square wave signal; this causes the piezo-electric material to be deformed as shown in *Figure 5.*

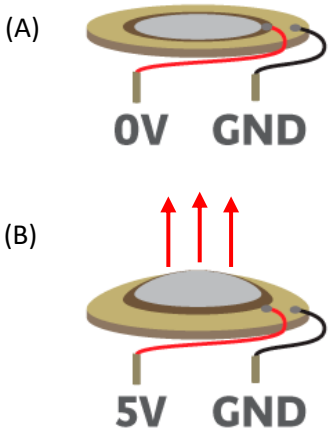


Figure 5. The internal components of a piezoelectric buzzer. (A) shows the state of the buzzer when no potential difference is experienced across the piezoelectric material, with it being undisturbed. (B) shows the result when 5V is applied, which causes the piezoelectric material to deform [15].

For the generated signal from the PIC to sound the piezo buzzer effectively, supporting circuitry is required. *Figure 6* shows the supporting components used within the Assignment Board circuit. The transistor (Q1) serves the purpose of amplifying the input excitation wave and preventing the buzzer

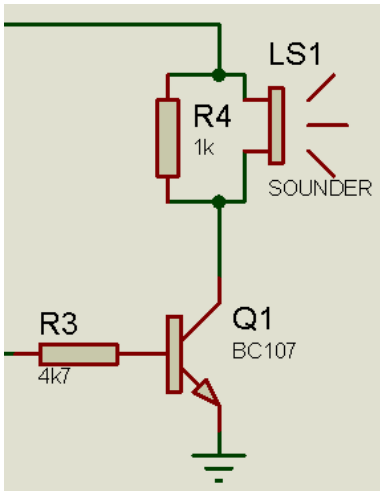


Figure 6. Piezoelectric buzzer with supporting circuitry used within the Assignment Board circuit.

from causing current or voltage spikes, while switching between states, at the microcontrollers output pin. With Q1 being a bipolar junction transistor (BJT), the resistor connected to its base (R3) is needed to regulate the current flow into the base with only a small amount, within the limits of the

microcontroller, is needed to drive the transistor. Connected in parallel to the buzzer is a resistor (R4), allowing the buzzer to reset to the undeformed state when the excitation is 0V [9].

Input Buttons

The buttons within the Assignment Board circuit, shown *Figure 7* allows for external user interactions with the circuit. Each button is connected to a pin of port A of the PIC microcontroller and act as input devices by facilitating changes in voltage at each pin. Attached to the connections between the buttons and PIC is a pull-up resistor. The purpose of this pull-up resistor is to pull the microcontroller pins to logic high when the buttons are open, creating a default input of logic level '1'. This prevents the input pins floating and reading unpredictable values.

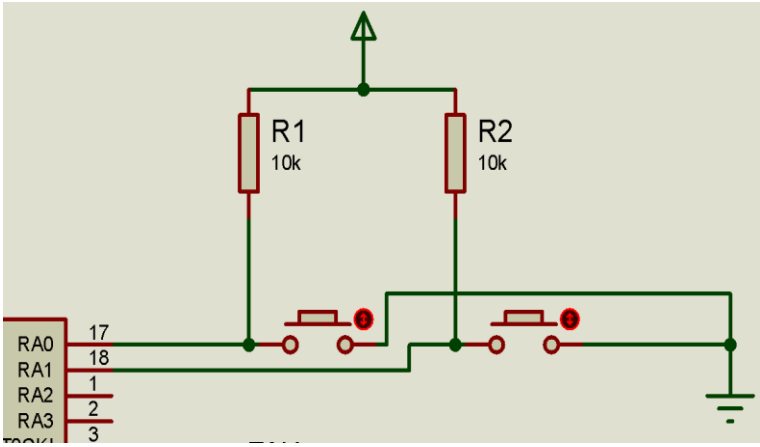


Figure 7. Input buttons connected to 2 pins of Port A of the PIC16F84A in the Assignment Board circuit with pull-up resistors connected to each button.

Using buttons as inputs introduces a problematic effect of button bounce, this is the result of mechanical vibrations when the button is pressed which causes the contacts of the button to briefly oscillate between being open and closed. The effect of this is a period of unstable input where the state of the button can be registered as multiple presses, *Figure 8* shows the signal seen by the microcontroller when the button

Figure 8. Signal seen at a pulled-up input from a button being pressed and experience bounce upon pressing and releasing [10].

is pressed. Different methods can be employed to mitigate the button bounce: using a capacitor to add a time delay component to remove the rapidly oscillating states, using a capacitor and Schmitt trigger to added time delay between state changes but with well-defined outputs, or using software techniques [10]. With the hardware included in the Assignment Board circuit, a software approach must be used to mitigate the effects; a 10ms delay should be implemented every button state read to ensure enough time passes for the bounce period to elapse.

Seven Segment Display

Providing visual character outputs can be achieved through using an arranged set of light emitting diodes (LED) which can be illuminated in different configurations to depict desired outputs. To conveniently achieve this, single packages are available which contain a collection of LED segments, laid out to display character outputs. An example component is the HDSP-7801 Broadcom Green 7-Segment LED Display, shown in *Figure 9*.

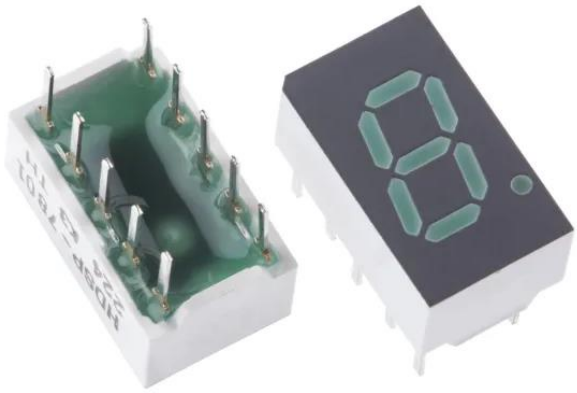


Figure 9. Image of the HDSP-7801 Broadcom Green 7-Segment LED Display component package [13].

Within the package each LED is connected to a common anode and their cathode connects to a separate external pin, as shown in *Figure 10*. The operation of each LED is similar to a regular diode, where when a positive voltage is applied across the anode to cathode above a threshold, current is able to flow, whereas when applied from cathode to anode the diode is reversed biased preventing current flow. Illumination of the LED occurs during the forward bias state [11].

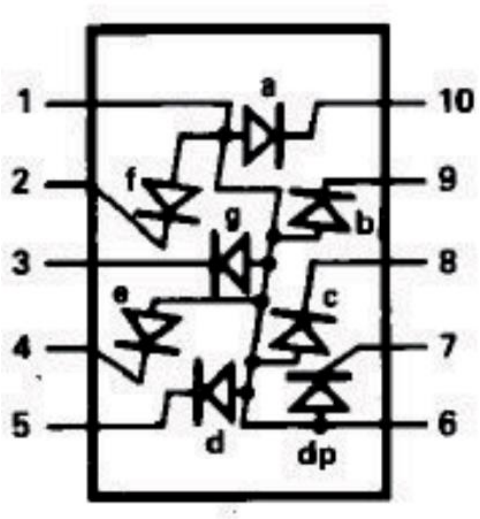


Figure 10. Internal circuit of the HDSP-7801 common anode seven segment display with decimal point [11].

Interfacing with the 7-Segment Display

Controlling a 7-segement display with the PIC16F84A can be achieved through multiple methods. Deciding how to interface with the display depends upon the requirements of the systems, the resource demands from the PIC and component budget. The method used within the Assignment Board circuit involves connecting each LED display segment's cathode directly to a pin of the PIC's general-purpose input/output (GPIO) port B through a resistor, shown in *Figure 11*. As discussed in the Seven Segment Display section above, to illuminate a segment of the display a positive voltage drop from the anode to

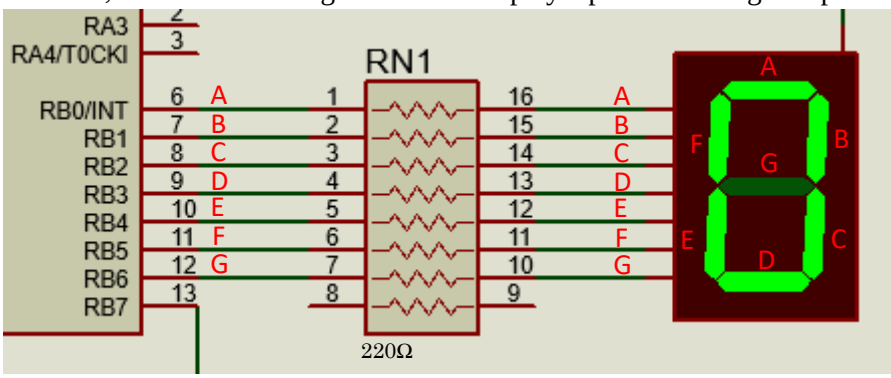
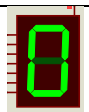
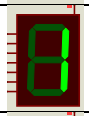
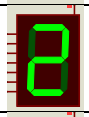
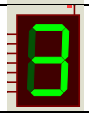


Figure 11. Section of the Assignment Board circuit showing the connection between the PIC16F84A and 7-segment display, with labels A-F showing the connections between the display's cathode and corresponding LED segment.

cathode needs to occur. This can be achieved by using the connected GPIO pins to output logic high (5V) or logic level low (0V), to either provide the voltage drop across the LED segment and illuminating it or providing 5V at both the cathode and anode, so no voltage difference, causing the segment to not illuminate. In between the PIC's GPIO pin and the cathode of a display segment, a resistor is needed to limit the current. This resistor is required as LED can only withstand a current up to a maximum limit and the GPIO pins of the PIC are only able to sink a maximum amount of current: LED segments of the example HDSP-7801 7-segment display has a maximum DC forward current rating of 30mA [11], and a PIC16F84A can only sink 25mA [2, p. 1]. For the Assignment Board circuit an 8x 220Ω resistor package is used to facilitate the connection between Port B pins 0-6 to the seven pins of the display.

Displaying meaningful characters on the 7-segment display requires the outputting of specific configurations to the pins RB0 – RB6. Achieving this involves setting bits 0 to 6 of the Port B register to then set the corresponding RB0 – RB6 pins. *Table 1* sets out the different characters to be displayed and the Port B bit configuration needed for it.

Display Character	**Illuminated Segments**	**PORT B Bits 0-6 Configuration**	**Display Output**
'0'	A,B,C,D,E,F	1000000	
'1'	B,C	1111001	
'2'	A,B,D,E,G	0100100	
'3'	A,B,C,G	0100100	

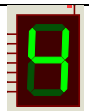
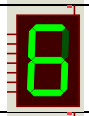
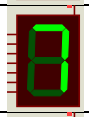
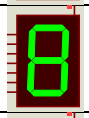
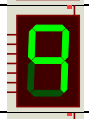
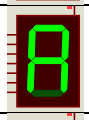
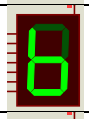
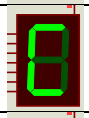
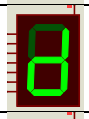
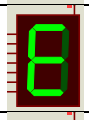
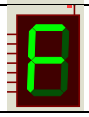
‘4’	B,C,F,G	0011001	
‘5’	A,C,D,F,G	0010010	
‘6’	A,C,D,E,F,G	0000010	
‘7’	A,B,C	1111000	
‘8’	A,B,C,D,E,F,G	0000000	
‘9’	A,B,C,F,G	0011000	
‘A’	A,B,C,E,F,G	0001000	
‘B’	C,D,E,F,G	0000011	
‘C’	A,D,E,F	1000110	
‘D’	B,C,D,E,G	1010001	
‘E’	A,D,E,F,G	0000110	
‘F’	A,E,F,G	0001110	

Table 1. The different characters to be displayed on the 7-segment display, with the required segments to be illuminated (segment labels are defined in Figure 12.) and the Port B bits 0-6 configuration to achieve this (the least significant bit is on the right and corresponds to segment A).

Programming Circuit Operations

For the PIC16F84A in the Assignment Board circuit to control the circuits operation, programs must be created using the available 35 instructions and loaded to the PIC to enable it to perform logical operations and interface with inputs and outputs. By using PIC assembly and the MPASM compiler, six programs have been created to control the circuit in different ways. The operations explored within these programs include taking button inputs, outputting to a 7-segment display using the method outlined in the Interfacing with the 7-Segment Display section, internal register control, delay routines, interrupts and generating pulse-width modulated (PWM) signals.

Delays

A particularly important operation used within all of the created PIC assembly programs is the delay routine. This routine creates delay by performing a specific number of instructions to take up the desired amount of time. Within the created programs, the delay subroutine does this through decrementing through two general purpose file registers, with the initial value of the register determining the length of the delay. The logical operation of the described delay subroutine is shown in *Figure 13*. Given the PIC16F84A is operated at a frequency of 4MHz it means one instruction cycle takes 1us, and with the number of instructions cycles in the delay routine used having been determined to be calculated with *Equation 1*, the delay length can be simply calculated with *Equation 2.*

$$With: \quad x_1, x_2 = 1 \leq x \leq 256$$

$$No.of\ Instructions\ Cycles\ in\ Delay = 3x_1 + 770x_2 - 762 \qquad \textit{Equation 1.}$$

$$Delay\ Length = Instruction\ Cycle\ Time \times No.of\ Instruction\ Cycles \qquad \textit{Equation 2.}$$

Where:

x_1 = value loaded into the delay counter 1,
x_2 = value loaded into the delay counter 2.

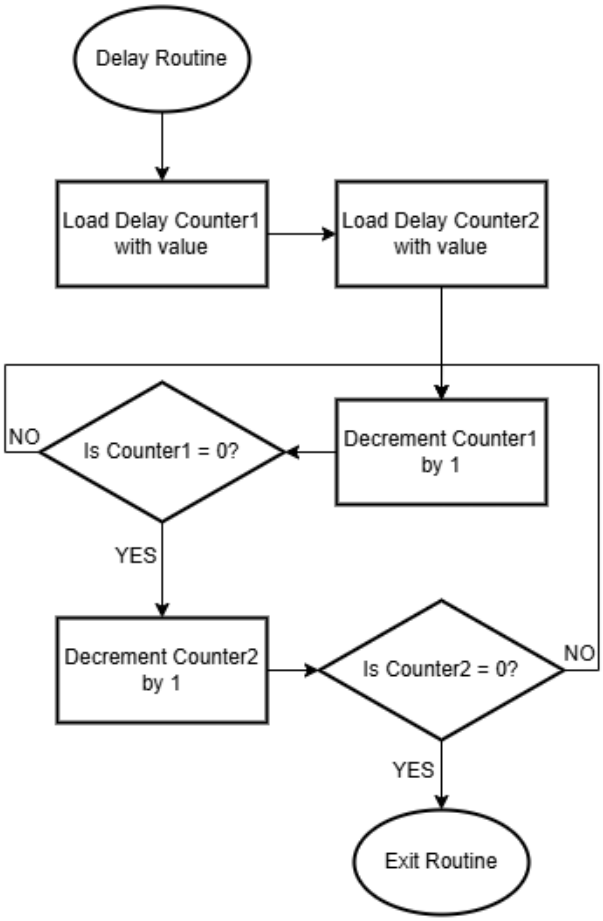


Figure 12. Flowchart showing the logical flow of the delay routine used to created delays on the PIC16F84A.

Interrupts

Another important operation used within some of the created programs is the interrupt feature. Interrupts are specific events that when detected by the microcontroller causes the normal program flow to be paused and a different routine be performed and then returning to the normal program flow, this is shown in *Figure 13*[16]. There are multiple ways for an interrupt to be triggered in a PIC microcontroller, these are external interrupt triggers, change of state of a port or by a timer overflow. Once one of the interrupt events occur, a corresponding interrupt flag is set and the specified service routine is performed. The trigger used within the following programs is the timer overflow which triggers when the timer counter reaches its maximum value and resets to 0 [12].

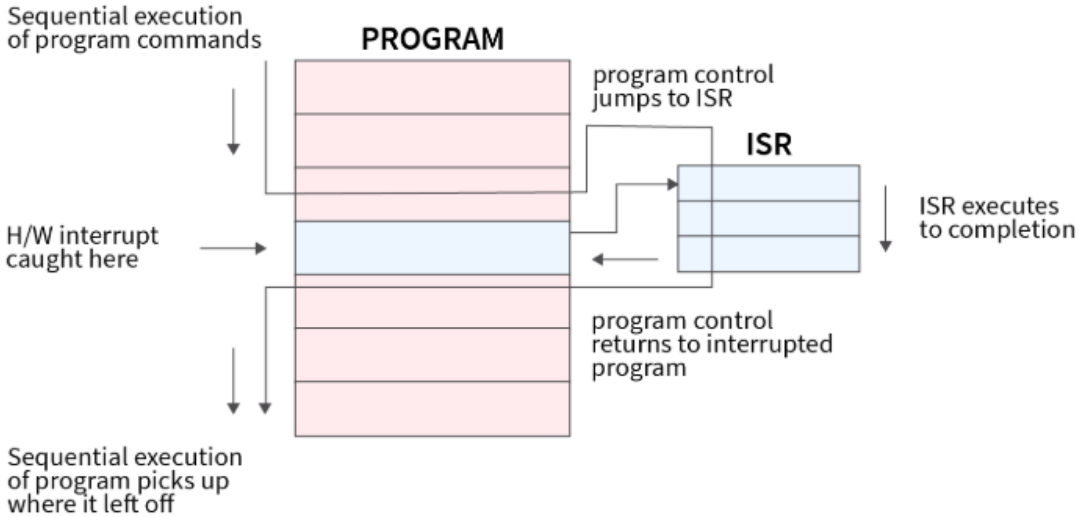


Figure 13. Program flow showing jump to a service routine following interrupt signal [16].

Assignment Board PIC Assembly Programs

Counting on 7-Segment Display with Modified Count.asm

The Counting on the 7-Segment Display program has been created in PIC assembly, see Appendix A: Counting on Seven Segment Display with Modified Count.asm - PIC Assembly Code, to increment a counter every 200ms from 0 to 16, looping back to 0 after 16, and displaying the count hexadecimal value to the 7-segment display in the Assignment Board circuit. *Figure 14* show a flowchart detailing the logical flow of the program.

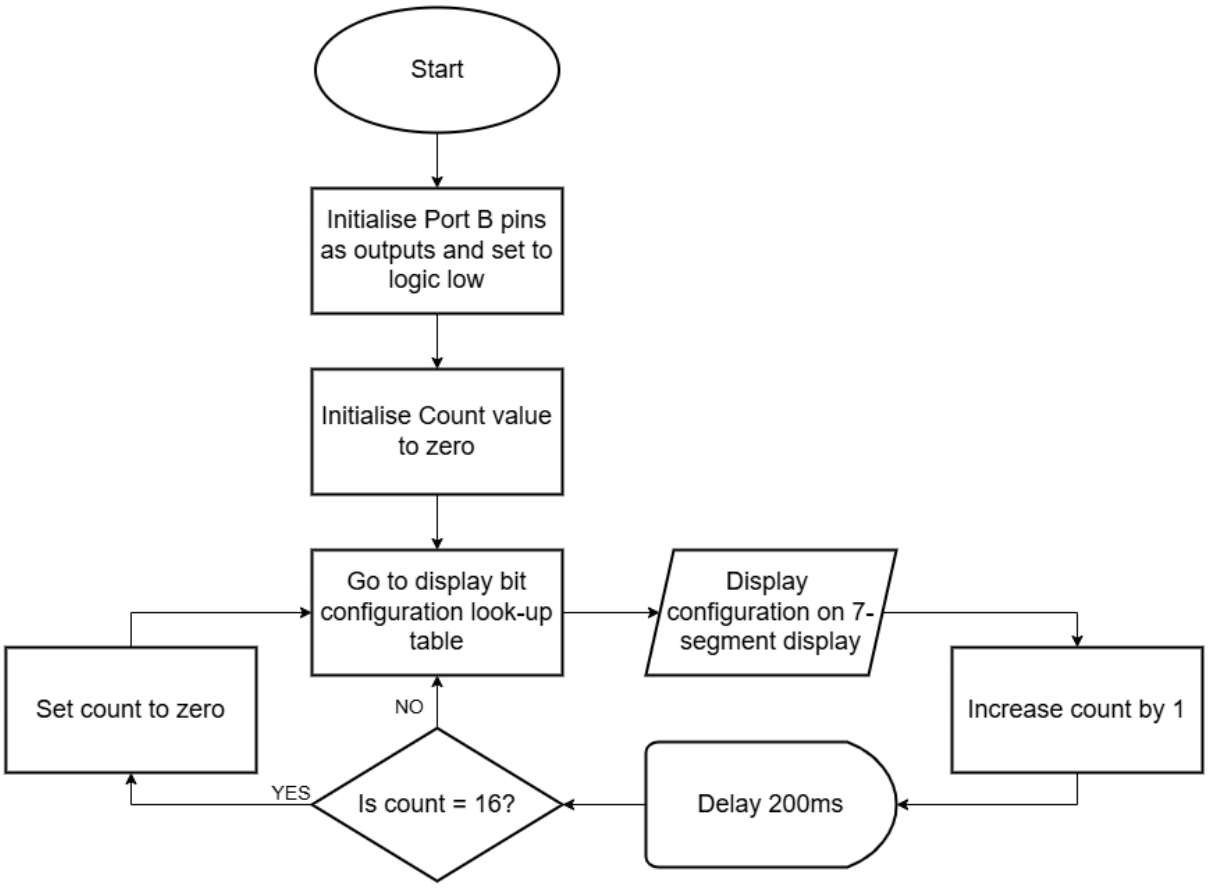


Figure 14. Flowchart illustrating the logical operation of the Counting on the 7-Segment display PIC assembly code.

Using Button Inputs

The Button Inputs program controls the Assignment Board circuit by interfacing with the input buttons and using the state of each button to display a ‘L’ or and a ‘r’ on the 7-segment display. The logical flow for this program is shown in the flowchart in *Figure 15* and is implement in the PIC assembly code in *Appendix B*.

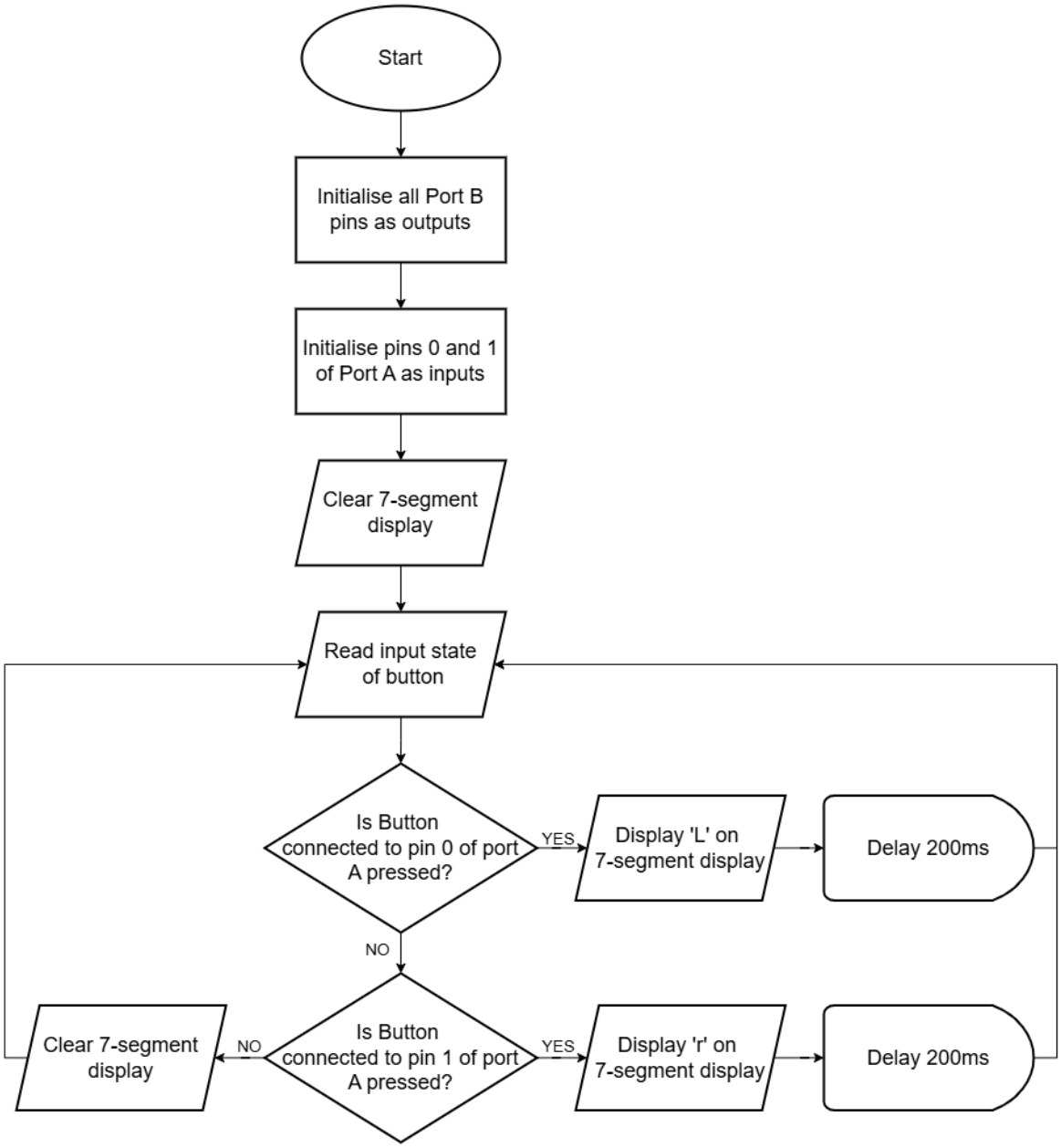


Figure 15. Flowchart illustrating the logical operation of the Using Button Inputs PIC assembly code.

Latching Button Inputs

The Latching Button Input program controls the circuit by latching the display output to the input buttons pressed. The operation of this program takes the states of the input button pins and if the left button is pressed all display segments are illuminated and if the right button is pressed all of the display segments are turned off. When neither button is pressed the display output latches to the last display state. Logical program flow is show in the flowchart in *Figure 16* and is implemented in the PIC assembly code in *Appendix C*.

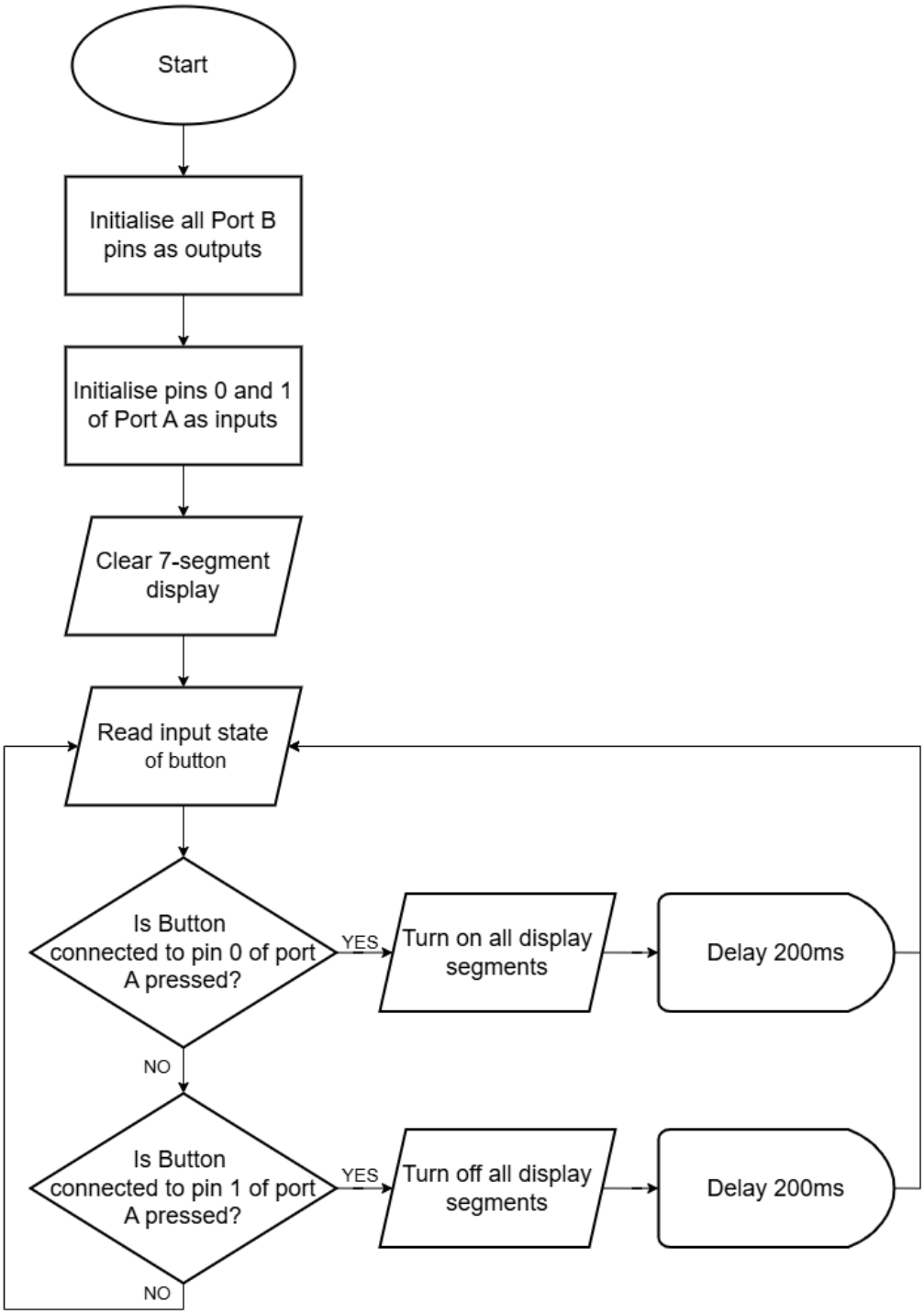


Figure 16. Flowchart illustrating the logical operation of the Counting on the Latching Button Inputs PIC assembly code.

Counting Up and Down with Buttons

The counting up and down program starts with an initial count value of 5 and uses the buttons to increase or decrease the count, which is displayed on the 7-segment display. The PIC assembly code for this program is in Appendix D and Figure 17 show the flowchart illustrating the programs logical flow. For this program to operate the EEPROM must be loaded with the bit configurations outlined in the Interfacing with the 7-Segment Display section for the numbers to be displayed on the 7-segement display when the PIC is programmed.

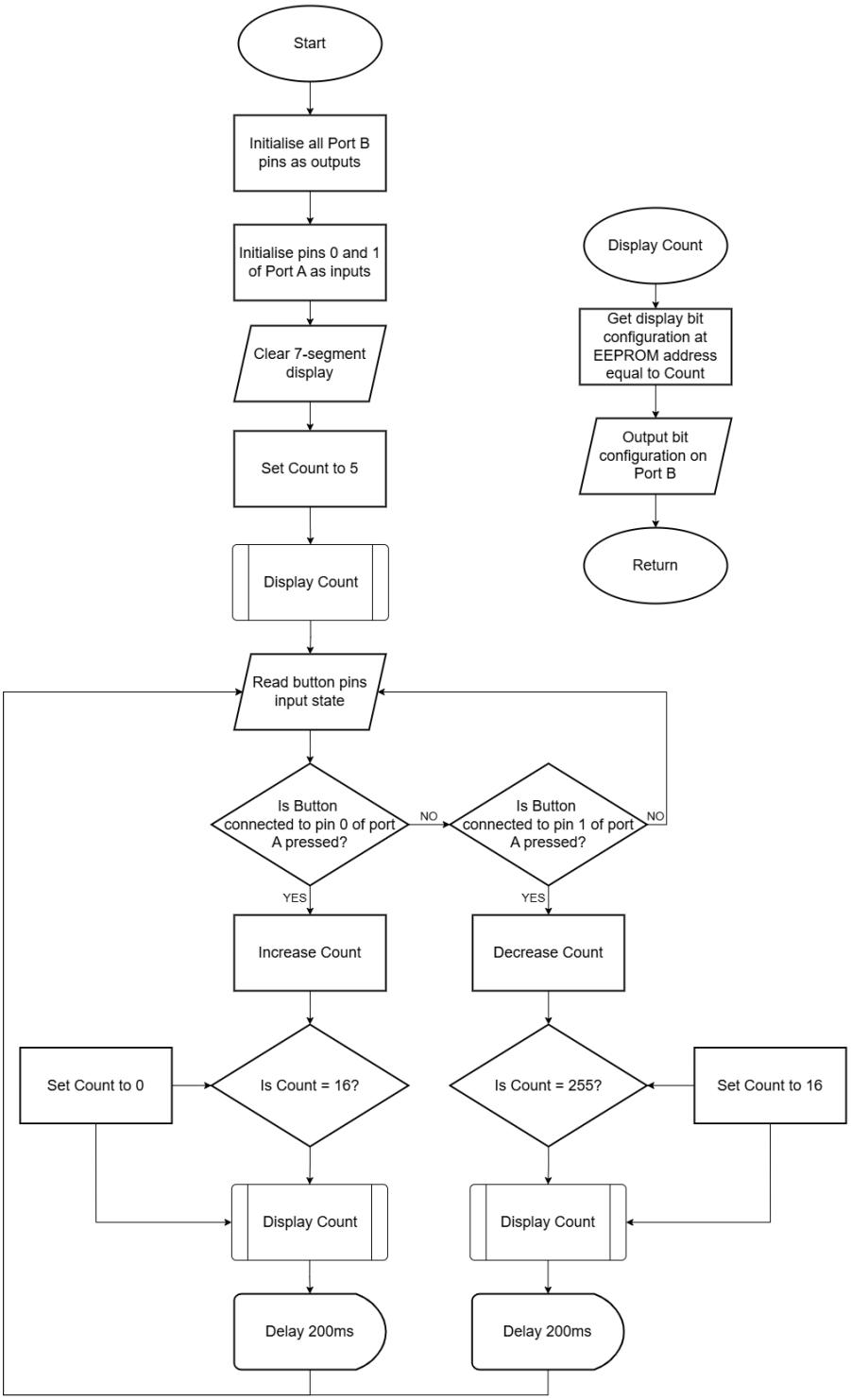


Figure 17. Flowchart illustrating the logical operation of the Counting with Button Inputs PIC assembly code.

Sounding A Piezo Buzzer

The sounding of the buzzer program takes the button inputs and if pressured the piezo buzzer is sounded by generating a PWM signal using software. *Figure 18* shows the flowchart of the logical follow of the PIC assembly program in *Appendix E*. This program makes use of timer interrupts to utilise the in-built hardware of the PIC16F84A to count the high and low sections of the PWM signal.

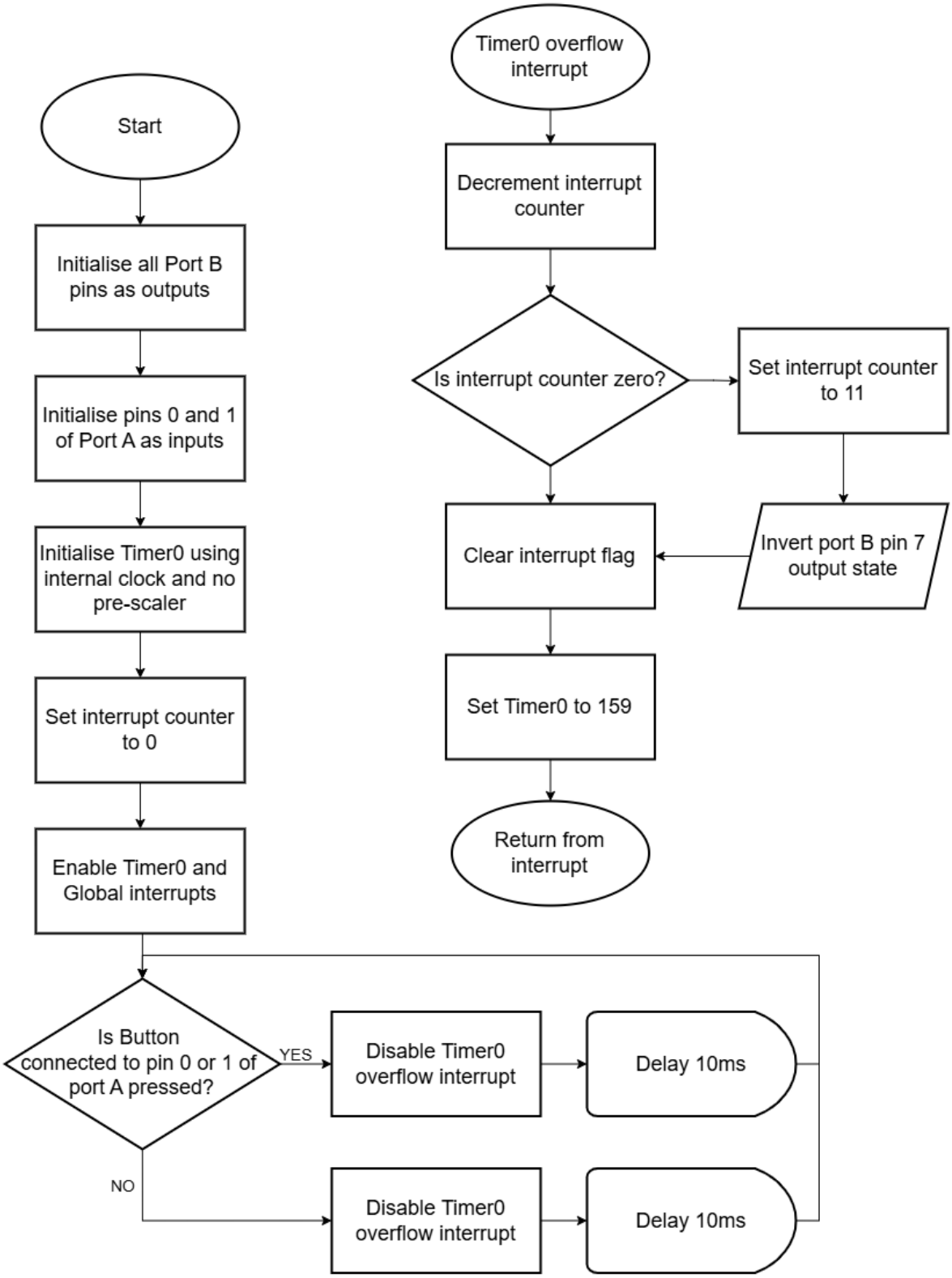


Figure 18. Flowchart illustrating the logical operation of the Sounding the Piezo Buzzer PIC assembly code.

Changing Piezo Buzzer Sound

The changing the buzzer sound program takes the button inputs and if either button is pressed then the buzzer is sounded. To sound the buzzer a software generated PWM signal is sent and to produce a changing output the duty cycle of the PWM signal is varied between 5% and 95%. *Figure 19* shows the flowchart describing the logical flow of the PIC assembly in *Appendix F*. Within this program the timer interrupt functionality is also used to count the high and low periods for the PWM signal.

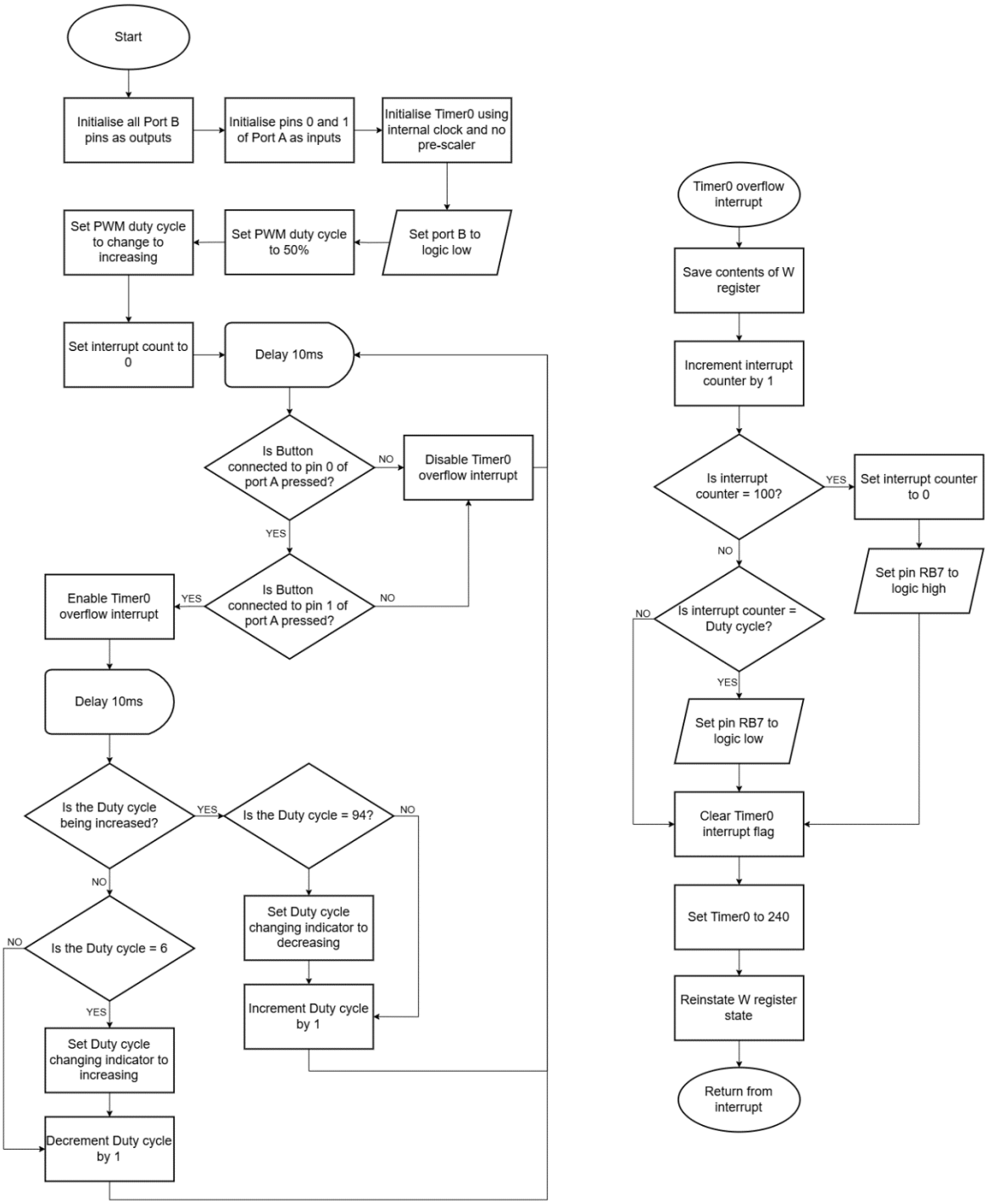


Figure 19. Flowchart illustrating the logical operation of the Changing the Sound of the Piezo Buzzer PIC assembly code.

Conclusion

To conclude, the components of the Assignment board circuit have been analysed to understand their function and purpose within the circuit. This has demonstrated how a microcontroller could be utilised to interface with inputs and outputs as seen with the included input buttons, 7-segment display and piezo buzzer along with the circuitry needed to interface with a PIC16F84A. Furthermore, this has shown supporting circuitry used for a microcontroller circuit, such as the crystal oscillator and voltage regulator. Moreover, interfacing with a 7-segment display with PIC microcontroller was explored and a method was devised to illuminate specific LED segments to display the hexadecimal values of 0 through to F, involving outputting different binary numbers to the connected port of the microcontroller. Finally, programs have been created in PIC assembly to control the Assignment Board circuit in various ways, this has demonstrated the versatility of the microcontroller and how software can be developed in different ways to alter the operation of the circuit.

References

[1] P. Horowitz and W. Hill, The Art of Electronics, New York: Cambridge University Press, 2015.

[2] Microchip Technology Incorporated, “PIC16F84A Data Sheet,” 2001.

[3] Arm Ltd, “Glosary: RISC,” [Online]. Available: https://www.arm.com/glossary/risc?utm_source=google&utm_medium=cpc&utm_content=text_txt_na_risc&utm_campaign=mk30_brand-paid_technologies_keyword&utm_term=risc&gad_source=1&gclid=CjwKCAiA9vS6BhA9EiwAJpnXw3WO_qN7oCFYnF__YBCMECR1voa-fT_C2Bq5n5QkkmHChyWusWbx. [Accessed 17 12 2024].

[4] Monolithic Power Systems, “Voltage Regulator Types and Working Principles,” [Online]. Available: https://www.monolithicpower.com/en/learning/resources/voltage-regulator-types?srsltid=AfmBOooTutiNzdq2fr0FD4hFy64flmfue03obmfEEyNU6wVKlh3j8q. [Accessed 11 November 2024].

[5] Texas Instruments, “LM340, LM340A and LM7805 Family Wide VIN 1.5-A Fixed Voltage Regulators Datasheet,” 2016.

[6] Renesas, “Choosing and Using Bypass Capacitors,” 2011.

[7] M. Ko and J. Metzler, “Crystal Basics,” CTS Corporation.

[8] RS Components Ltd., “How Do Piezo Buzzers Work,” 29 January 2024. [Online]. Available: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/how-do-piezo-buzzers-work?srsltid=AfmBOorGAZGKDuJVEUYt_dq_6nfSr1b68OQv9aszZGEkQnbO9lTs3wZW. [Accessed 16 November 2024].

[9] B. Rose, “Buzzer Basics - Technologies, Tones and Drive Circuits,” [Online]. Available: https://www.sameskydevices.com/blog/buzzer-basics-technologies-tones-and-driving-circuits#:~:text=To%20operate%20a%20transducer%2C%20the,buzzer%20a%20specified%20dc%20voltage.&text=In%20a%20magnetic%20buzzer%2C%20the%20transistor%20acts%20as%20the%20drivi. [Accessed 26 November 2024].

[10] C. Maxfield, “How to Implement Hardware Debounce for Switches and Relays When Software Debounce Isn’t Appropriate,” DigiKey, 09 Febuary 2021. [Online]. Available: https://www.digikey.co.uk/en/articles/how-to-implement-hardware-debounce-for-switches-and-relays?srsltid=AfmBOoosGNolS_GtQAHbHlGyyVixwNh0EBgWYai3c-p0FhEVxVkKKoxU. [Accessed 27 December 2024].

[11] Avago Technologies, “7.6 mm (0.3 inch) Micro Bright Seven Segment Displays,” *HDSP-740x Series Data Sheet*, 25 March 2014.

[12] Microchip, “8-bit PIC® MCU Interrupts,” 09 November 2023. [Online]. Available: https://developerhelp.microchip.com/xwiki/bin/view/products/mcu-mpu/8bit-pic/peripherals/interrupts/. [Accessed 12 December 2024].

[13] RS Components, “HDSP-7801 Broadcom 7-Segment LED Display, CA Green 6.8 mcd RH DP 7.6mm,” [Online]. Available: https://uk.rs-online.com/web/p/led-displays/0451051?srsltid=AfmBOooOzKLoTMQQ2wm1ybNyoNyo0FntYiC-JvtDOWerQy24oU-8iO5H. [Accessed 03 November 2024].

[14] J. Lindblom, “Diodes,” Sparkfun Electronics, [Online]. Available: https://learn.sparkfun.com/tutorials/diodes/real-diode-characteristics. [Accessed 03 November 2024].

[15] Arduino Verkstad, “Beep,” [Online]. Available: http://ctc-dev.verkstad.cc/en/course-literature/beep/. [Accessed 24 November 2024].

[16] S. Gaurav, “Interrupt Handling,” 15 October 2023. [Online]. Available: https://www.scaler.com/topics/operating-system/interrupt-handling/. [Accessed 12 December 2024].

Appendix A: Counting on Seven Segment Display with Modified Count.asm - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
              radix         hex
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
count         EQU    0x0C          ; Counter for 0 to 255
nCount        EQU    0x0D          ; Pause routine inner loop counter
mCount        EQU    0x0E          ; Pause routine outer loop counter

;=====================================================================
; MAIN CODE
;=====================================================================
       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 pins low
       CLRF   count         ; Initialize counter to 0

COUNTING
       CALL   LOOK_UP       ; Get display config given count from
                            ;      program memory look-up table into w register
       MOVWF  PORTB         ; Move display config to PORTB

       MOVF   count, W      ; GET COUNT VALUE IN W REGISTER
       ADDLW  1             ; ADD 1 TO W REGISTER
       MOVWF  count         ; SAVE W BACK IN COUNT

       CALL   pause         ; Delay 200ms by subroutine

       MOVF   count, W
       XORLW  0x10

       BTFSS  STATUS, Z
       GOTO   COUNTING      ; Repeat forever
```

```
        MOVLW   0
        MOVWF   count

        GOTO    COUNTING        ; Repeat forever

;=====================================================================
; SUBROUTINE
;=====================================================================
pause
        MOVLW   0x00            ; Set w = 255 decimal
        MOVWF   mCount          ; mCount = w: outer loop counter
LOAD_N
        MOVLW   0x00            ; Set w = 255 decimal
        MOVWF   nCount          ; nCount = w: inner loop counter
DEC_N
        DECFSZ nCount, F        ; nCount--
        GOTO    DEC_N           ; If nCount != 0 then repeat nCount--

        DECFSZ mCount, F        ; Else decrement Count
        GOTO    LOAD_N          ; If mCount != 0 then
                                ;      reload nCount to 255 and decrement

        RETURN                  ; Else exit subroutine

;=====================================================================
; 7-SEGEMENT DISPLAY CONFIGURATION LOOK-UP TABLE
;=====================================================================
LOOK_UP
       MOVF     count, W        ; Moving value of count into W register
       ADDWF    PCL, F          ; Adding count value to program counter to jump to
                                ;      corresponding bit configuration
       RETLW    B'01000000'     ; 0 – Return with bit configuration in W register
       RETLW    B'01111001'     ; 1
       RETLW    B'00100100'     ; 2
       RETLW    B'00110000'     ; 3
       RETLW    B'00011001'     ; 4
       RETLW    B'00010010'     ; 5
       RETLW    B'00000010'     ; 6
       RETLW    B'01111000'     ; 7
       RETLW    B'00000000'     ; 8
       RETLW    B'00011000'     ; 9
       RETLW    B'00001000'     ; A
       RETLW    B'00000011'     ; b
       RETLW    B'01000110'     ; C
       RETLW    B'00100001'     ; d
       RETLW    B'00000110'     ; E
       RETLW    B'00001110'     ; F
;=====================================================================
        END
```

Appendix B: Using Button Inputs - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
DELAY1        EQU    0x0C          ; Delay counter 1
DELAY2        EQU    0x0D          ; Delay counter 2

;=====================================================================
; MAIN CODE
;=====================================================================
       ORG    0x0000        ; Setting program memory location to 0x0000
START
       BSF    STATUS, RP0   ; Changing to bank 1

       CLRF   TRISB         ; Clearing TRISB reg to set PORTB to outputs

       MOVLW  0x03          ; Setting PORTA pins 0 & 1 to inputs
       MOVWF  TRISA         ;

       BCF    STATUS, RP0   ; Returning to bank 0

CLEAR_DISP
       MOVLW  0xFF          ; Setting all of PORTB outputs to high
       MOVWF  PORTB         ;

CHECKING_PINS               ; Constantly polling button pins

       BTFSS  PORTA, 0      ; Check if RA0 bit is 0 in PORTA if not
                            ;      skip next command
       GOTO   DISP_L        ; If button pressed bit is 0 so count is increased

       BTFSS  PORTA, 1      ; Check if RA1 bit is 0 in PORTA if not
                            ;      skip next command
       GOTO   DISP_R        ; If button pressed bit is 0 so count is decreased

       GOTO   CLEAR_DISP    ; Clear display repeat checking the button pins

DISP_L                      ; For increasing count

       MOVLW  B'01000111'
       MOVWF  PORTB
       CALL   PAUSE         ; 10ms delay to prevent issues with button bounce

       GOTO   CHECKING_PINS ; Go back to checking pins
```

```
DISP_R                      ; For decreasing count

       MOVLW  B'01001110'
       MOVWF  PORTB
       CALL   PAUSE         ; 10ms delay to prevent issues with button bounce

       GOTO   CHECKING_PINS ; Go back to checking pins

;=====================================================================
; SUB-ROUTINES
;=====================================================================
PAUSE
       MOVLW  0x00          ; Move DEC 256 into DELAY1 – for decrementing loop
       MOVWF  DELAY1        ;      0 acts as DEC 256

       MOVLW  0x00          ; Move DEC 256 into DELAY2
       MOVWF  DELAY2        ;

       DECFSZ DELAY1, F     ; Decrement DELAY1
       GOTO   $ - 1         ; If DEALY1 != 0 then go back one step

       DECFSZ DELAY2, F     ; Else decrement DELAY2
       GOTO   $ - 3         ; If DELAY2 != 0 then go back three steps

       RETURN

;=====================================================================
        END
```

Appendix C: Latching Button Inputs - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
DELAY1        EQU    0x0C          ; Delay counter 1
DELAY2        EQU    0x0D          ; Delay counter 2

;=====================================================================
; MAIN CODE
;=====================================================================
       ORG    0x0000        ; Setting program memory location to 0x0000
Start
       BSF    STATUS, RP0   ; Changing to bank 1

       CLRF   TRISB         ; Clearing TRISB reg to set PORTB to outputs

       MOVLW  0x03          ; Setting PORTA pins 0 & 1 to inputs
       MOVWF  TRISA         ;

       BCF    STATUS, RP0   ; Returning to bank 0

       MOVLW  0x7F          ; Setting all PORTB outputs to high
       MOVWF  PORTB         ;

CHECKING_PINS               ; Constantly polling button pins

       BTFSS  PORTA, 0      ; Check if RA0 bit is 0 in PORTA if not
                            ;      skip next command
       GOTO   SET_HIGH      ; If button pressed bit is 0 so count is increased

       BTFSS  PORTA, 1      ; Check if RA1 bit is 0 in PORTA if not
                            ;      skip next command
       GOTO   SET_LOW       ; If button pressed bit is 0 so count is decreased

       GOTO   CHECKING_PINS ; Repeat checking the button pins

SET_HIGH                    ; For turning off LED segments of display
       MOVLW  0x7F          ; Setting first 7 pins of PORTB to high
       MOVWF  PORTB
       CALL   DELAY         ; 10ms delay to prevent issues with button bounce

       GOTO   CHECKING_PINS ; Go back to checking pins
```

```
SET_LOW                     ; For turning on LED segments of display
       CLRF   PORTB         ; Setting first 7 bits of PORTB to low
       CALL   DELAY         ; 10ms delay to prevent issues with button bounce

       GOTO   CHECKING_PINS ; Go back to checking pins

;=====================================================================
; SUB-ROUTINES
;=====================================================================
DELAY
       MOVLW  0xFA          ; Loading COUNT1 with DEC 250
       MOVWF  DELAY1
       MOVLW  0x0D          ; Loading COUNT2 with DEC 13
       MOVWF  DELAY2

       DECFSZ DELAY1, F     ; Decrement DELAY1
       GOTO   $ - 1         ; If DEALY1 != 0 then go back one step

       DECFSZ DELAY2, F     ; Else decrement DELAY2
       GOTO   $ - 3         ; If DELAY2 != 0 then go back three steps
       RETURN

;=====================================================================
        END
```

Appendix D: Counting Up and Down with Buttons - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
COUNT         EQU    0x0C          ; Counter 0 to 16
DELAY1        EQU    0x0D          ; Pause routine inner loop counter
DELAY2        EQU    0x0E          ; Pause routine outer loop counter
;=====================================================================
; INITIAL EEPROM DATA
;=====================================================================
       ; Location of EEPROM within the PIC starting at EEPROM address 0x00
       ORG    0x2100

       ; Following loads values into EEPROM when programming PIC

       ; PORTB bit configuration for numbers 0 - 3 in EEPROM locations 0x00 - 0x03
       DE b'01000000', b'01111001', b'00100100', b'00110000'

       ; PORTB bit configuration for numbers 4 - 7 in EEPROM locations 0x04 - 0x07
       DE b'00011001', b'00010010', b'00000010', b'01111000'

       ; PORTB bit configuration for numbers 8 - b in EEPROM locations 0x08 - 0x0B
       DE b'00000000', b'00011000', b'00001000', b'00000011'

       ; PORTB bit configuration for numbers c - f in EEPROM locations 0x0C - 0x0F
       DE b'01000110', b'01000000', b'00000110', b'00001110'
;=====================================================================
; MAIN CODE
;=====================================================================
       ORG    0x0000        ; Setting program memory location to 0x0000
START
       BSF    STATUS, RP0   ; Changing to bank 1

       CLRF   TRISB         ; Clearing TRISB reg to set PORTB to outputs
       MOVLW  0x03          ; Setting PORTA pins 0 & 1 to inputs
       MOVWF  TRISA         ;

       BCF    STATUS, RP0   ; Returning to bank 0

       MOVLW  0xFF          ; Setting all PORTB outputs to high
       MOVWF  PORTB         ;

       MOVLW  0x05          ; Initialising count to DEC 5
       MOVWF  COUNT         ;

       CALL   DISP_COUNT    ; Do subroutine to display count on the display
```

```
CHECKING_PINS                ; Constantly polling button pins

       BTFSS  PORTA, 0       ; Check if RA0 bit is 0 in PORTA
                             ;      if not skip next command
       GOTO   INCREASE       ; If button pressed bit is 0 so count is increased

       BTFSS  PORTA, 1       ; Check if RA1 bit is 0 in PORTA
                             ;      if not skip next command
       GOTO   DECREASE       ; If button pressed bit is 0 so count is decreased

       GOTO   CHECKING_PINS ; Repeat checking the button pins


INCREASE                     ; For increasing count

       INCF   COUNT, 1       ; Increment count by 1
       MOVLW  0x0A           ; Testing if count has gone over DEC 9
       XORWF  COUNT, 0       ; If COUNT = 0x10 then result is 0
       BTFSC  STATUS, Z      ; Checking Z flag to see if XOR result was zero
       CLRF   COUNT          ; If XOR was 0 then set COUNT back to 0

       CALL   DISP_COUNT     ; Displaying COUNT on display
       CALL   PAUSE          ; 200ms delay - controls COUNT change speed and
                             ;      prevents issues with button bounce

       GOTO   CHECKING_PINS ; Go back to CHECKING_PINS


DECREASE                     ; For decreasing COUNT

       DECF   COUNT, 1       ; Decrement COUNT by 1
       MOVLW  0xFF           ; Testing if COUNT has gone below 0
       XORWF  COUNT, 0       ; If CONNT = 0xFF then result of operation is 0
       BTFSS  STATUS, Z      ; Checking Z flag to see if XOR result was zero
       GOTO   $ + 3          ; If Z flag not 0 goto current program counter + 3
       MOVLW  0x09           ; If Z flag is zero move 0x09 (DEC 9)
       MOVWF  COUNT          ;      into COUNT

       CALL   DISP_COUNT     ; Displaying COUNT on display
       CALL   PAUSE          ; 200ms delay - controls COUNT change speed and
                             ;      prevents issues with button bounce

       GOTO   CHECKING_PINS ; Go back to CHECKING_PINS
;=====================================================================
; SUB-ROUTINES
;=====================================================================
PAUSE
       MOVLW  0x00           ; Move DEC 256 into DELAY1 – for decrementing loop
       MOVWF  DELAY1         ;      0 acts as DEC 256

       MOVLW  0x00           ; Move DEC 256 into DELAY2
       MOVWF  DELAY2         ;

       DECFSZ DELAY1, F      ; Decrement DELAY1
       GOTO   $ - 1          ; If DEALY1 != 0 then go back one step

       DECFSZ DELAY2, F      ; Else decrement DELAY2
       GOTO   $ - 3          ; If DELAY2 != 0 then go back three steps

       RETURN                ; Else exit subroutine
```

```
DISP_COUNT
        MOVF    COUNT, W        ; Move COUNT to W
        MOVWF   EEADR           ; Port configuration bytes stored in EEPROM from
                                ; address 0X00 to 0X0F which correspond to COUNT value
        BSF     STATUS, RP0     ; Changing to bank 1
        BSF     EECON1, RD      ; Setting the EEPROM read bit
        BCF     STATUS, RP0     ; Changing back to bank 0
        MOVF    EEDATA, W       ; Reading EEPROM at COUNT location into W register
        MOVWF   PORTB           ; Move W to PORTB to update the display
        RETURN
;=====================================================================
         END
```

Appendix E: Sounding a Piezo Buzzer - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
COUNTER       EQU    0x0C          ; Number of interrupts before change
                                   ;      for PWM signal
DELAY1        EQU    0x0D          ; Delay counter 1
DELAY2        EQU    0x0E          ; Delay counter 2

;=====================================================================
; BEGINING OF PROGRAM MEMORY
;=====================================================================
       ORG    0x0000        ; Setting program memory location to 0x0000
       GOTO   Start         ; Go to Start

;=====================================================================
; INTERRUPT ROUTINE - MUST BE STORED AT PROGRAM MEMORY LOCATION 0x0004
;=====================================================================
       ORG    0x0004        ; Interrupts sets program counter to 0x0004

       DECFSZ COUNTER, 1    ; Decrement number of interrupts counter
       GOTO   RESET_TIMER   ; If not zero reset TIMER0 to give specific
                            ;      timer interrupt delay

                            ; If counter goes to zero:
       MOVLW  0x0B          ; Resetting counter to 11
       MOVWF  COUNTER       ;

       MOVLW  0x80          ; Inverting pin RB7
       XORWF  PORTB, 1      ; XOR allows the RB7 bit to be inverted and all other
                            ;      pins to be unchanged

RESET_TIMER
       BCF    INTCON, T0IF  ; Resetting the timer interrupt flag
       MOVLW  0x9F          ; Setting TIMER0 value to cause interrupt in 96us
       MOVWF  TMR0          ;

       RETFIE               ; Return from interrupt routine and re-enable the
                            ;      global interrupt bit
```

```
;=====================================================================
; MAIN CODE
;=====================================================================
Start
       BSF    STATUS, RP0   ; Changing to bank 1

       CLRF   TRISB         ; Clearing TRISB register to set PORTB to outputs

       MOVLW  0x03          ; Setting PORTA pins 0 & 1 to inputs
       MOVWF  TRISA         ;

       BCF    OPTION_REG, 5 ; Setting TMR0 clock selection bit for internal
                            ;      instruction clock
       BSF    OPTION_REG, 3 ; Assigns the timer pre-scaler to the WDT so
                            ;      no pre-scaler applied to TIMER0

       BCF    STATUS, RP0   ; Returning to bank 0

       CLRF   PORTB         ; Initialising all PORTB pin to low

       CLRF   COUNTER       ; Initialise interrupt counter to 0

       MOVLW  0xA0          ; Setting the timer interrupt and global
       MOVWF  INTCON        ;      interrupt enable bit and ensuring all other
                            ;      INTCON bits are clear


CHECKING_PINS               ; Constantly polling button pins

       BTFSS  PORTA, 0      ; If button 0 is not pressed RA0 is 1 so
                            ;      skip to check button 1
       GOTO   ACTIVATE      ; If button 0 is pressed RA0 is 0 so activate
                            ;      PWM signal
       BTFSS  PORTA, 1      ; If button 1 is not pressed RA1 is 1 so skip next
       GOTO   ACTIVATE      ; If button 1 is pressed RA1 is 0 so go to
                            ;       activate PWM signal
DEACTIVATE
       BCF    INTCON, T0IE  ; If neither button pressed turn off TIMER0
                            ;      interrupt
       BCF    PORTB, 7      ; Ensuring Buzzer is turned off
       CALL   DELAY         ; 10ms delay to avoid issues with button bounce

       GOTO   CHECKING_PINS ; Looping polling buttons

ACTIVATE
       BSF    INTCON, T0IE  ; Turning on timer interrupts
       CALL   DELAY         ; 10ms delay to avoid issues with button bounce

       GOTO CHECKING_PINS   ; Looping polling buttons
```

```
;=====================================================================
; SUB-ROUTINES
;=====================================================================
DELAY
       MOVLW  0xFA          ; Loading COUNT1 with DEC 250
       MOVWF  DELAY1
       MOVLW  0x0D          ; Loading COUNT2 with DEC 13
       MOVWF  DELAY2

       DECFSZ DELAY1, F     ; Decrement DELAY1
       GOTO   $ - 1         ; If DEALY1 != 0 then go back one step

       DECFSZ DELAY2, F     ; Else decrement DELAY2
       GOTO   $ - 3         ; If DELAY2 != 0 then go back three steps
       RETURN

;=====================================================================
        END
```

Appendix F: Sound Piezo Buzzer with Changing PWM Signal - PIC Assembly Code

```
;=====================================================================
; Processor: PIC16F84A
; Compiler:  MPASM (Proteus)
;=====================================================================
; DEFINITIONS
;=====================================================================
              list          p=16f84a
#include      p16f84a.inc                 ; Include register definition file

;=====================================================================
; DEVICE CONFIGURATION
;=====================================================================
      __CONFIG _XT_OSC & _WDT_OFF  ; Setting clock to external crystal oscillator
                                   ; and turning off the watch dog timer
;=====================================================================
; VARIABLES
;=====================================================================
W_TEMP        EQU    0x0C          ; For temporary storage of w register
                                   ;      during interrupt
COUNTER       EQU    0x0D          ; NUMBER OF INTERRUPTS BEFORE CHANGE FOR PWM
                                   ;      SIGNAL
DUTY          EQU    0x0E          ; PWM duty cycle percentage
DIR           EQU    0x0F          ; Duty cycle increment/decrement
                                   ;      direction indicator
DELAY1        EQU    0x10          ; Delay counter 1
DELAY2        EQU    0x11          ; Delay counter 2

;=====================================================================
; BEGINING OF PROGRAM MEMORY
;=====================================================================
       ORG    0x0000        ; Setting program memory location to 0x0000
       GOTO   Start         ; Go to Start
;=====================================================================
; INTERRUPT ROUTINE - MUST BE STORED AT PROGRAM MEMORY LOCATION 0x0004
;=====================================================================
       ORG    0x0004        ; Interrupts sets program counter to 0x0004

       MOVWF  W_TEMP        ; Saving current W register state

       INCF   COUNTER, F    ; Increment counter
COUNTER_CHECK
       MOVLW  0x64          ; Seeing if counter has reached max of DEC 100
       XORWF  COUNTER, W    ;
       BTFSS  STATUS, Z     ; Testing if XOR result is 0
                            ;      - it is if counter is DEC 100
       GOTO   HIGH_CHECK    ; If counter is not DEC 100 go to HIGH_CHECK
       CLRF   COUNTER       ; If counter is 100 reset back to 0
       BSF    PORTB,  7     ; Set PWM pin to high

       GOTO   RESET_TIMER   ; Go to RESET_TIMER


HIGH_CHECK
       MOVF   DUTY, W       ; Checking if counter is at the duty cycle value
       XORWF  COUNTER, W    ;
       BTFSS  STATUS, Z     ; Testing if XOR result is zero
                            ;      - it is if counter is at duty cycle value
```

```
        GOTO    RESET_TIMER    ; If not at duty cycle value go to RESET_TIMER
        BCF     PORTB, 7       ; If it is set PWM pin to low


RESET_TIMER
        BCF     INTCON, T0IF   ; Resetting the timer interrupt flag
        MOVLW   0xF0           ; Setting TIMER0 value to cause interrupt every 11us
        MOVWF   TMR0           ;

        MOVF    W_TEMP, W      ; Re-instating W register state

        RETFIE                 ; Return from interrupt routine and re-enable the
                               ;      global interrupt bit

;=====================================================================
; MAIN CODE
;=====================================================================
Start
        BSF     STATUS, RP0    ; Changing to bank 1

        CLRF    TRISB          ; Clearing TRISB register to set PORTB to outputs

        MOVLW   0x03           ; Setting PORTA pins 0 & 1 to inputs
        MOVWF   TRISA          ;

        BCF     OPTION_REG, 5  ; Setting TMR0 clock selection bit for
                               ;      internal instruction clock
        BSF     OPTION_REG, 3  ; Assigns the timer pre-scaler to the WDT so
                               ;      no pre-scaler applied to TIMER0

        BCF     STATUS, RP0    ; Returning to bank 0

        CLRF    PORTB          ; Initialising all PORTB pin to low

        MOVLW   0x32           ; Initialising duty with a value of DEC 50
        MOVWF   DUTY           ;

        CLRF    COUNTER        ; Starting counter at 0

        CLRF    DIR            ; Starting duty counting direction to be increasing

        MOVLW   0xA0           ; Setting the timer interrupt and global interrupt
                               ;      enable bit and ensuring all other INTCON bits
        MOVWF   INTCON         ;      are clear

CHECKING_PINS                  ; Constantly polling button pins
        CALL    DELAY          ; 10ms delay to avoid issues with button bounce

        BTFSS   PORTA, 0       ; If button 0 is not pressed RA0 is 1 so
                               ;      skip to check button 1
        GOTO    ACTIVATE       ; If button 0 is pressed RA0 is 0 so
                               ;      activate PWM signal
        BTFSS   PORTA, 1       ; If button 1 is not pressed RA1 is 1 so skip next
        GOTO    ACTIVATE       ; If button 1 is pressed RA1 is 0 so
                               ;      go to activate PWM signal

        BCF     INTCON, T0IE   ; If neither button pressed turn off
                               ;      TIMER0 interrupt

        GOTO    CHECKING_PINS  ; Looping polling buttons
```

```
ACTIVATE
        BSF     INTCON, T0IE  ; Turning on timer interrupts

CHECK_DIR
        CALL    DELAY         ; Wait 10ms before changing duty cycle

        MOVF    DIR, F        ; Moving DIR into itself to test if it is zero
        BTFSS   STATUS, Z     ; Testing zero flag of STATUS register for
                              ;      result of last operation
        GOTO    MIN_CHECK     ; If DIR is non-zero go to decrementing duty cycle and
                              ;      checking if at minimum otherwise skip and go to
                              ;      increasing duty cycle and checking if at max
MAX_CHECK
        MOVLW   0x5E          ; Checking if duty cycle is at DEC 94
                              ;      (the max it will reach is DEC 95)
        XORWF   DUTY,  W      ; XOR will equal zero if DUTY is equal to DEC 94
        BTFSC   STATUS, Z     ; Bit testing the Z flag of the STATUS register to
                              ;      see if XOR resulted in zero
        DECF    DIR,   F      ; If Z flag set decrement DIR to make it non-zero
                              ;      indicating to count DUTY down
        INCF    DUTY,  F      ; Increase duty cycle by 1

        GOTO CHECKING_PINS    ; Looping polling buttons

MIN_CHECK
        MOVLW   0x06          ; Checking if duty cycle is at DEC 6
                              ;      (the min it will reach is DEC 5)
        XORWF   DUTY, W       ; XOR will equal zero if duty is equal to DEC 6
        BTFSC   STATUS, Z     ; Bit testing the Z flag of the STATUS register to
                              ;      see if XOR resulted in zero
        CLRF    DIR           ; If Z flag set clear DIR to zero indicating to
                              ;      count DUTY up
        DECF    DUTY,  F      ; Decrement duty

        GOTO CHECKING_PINS    ; Looping polling buttons

;=====================================================================
; SUB-ROUTINES
;=====================================================================
DELAY
        MOVLW   0xFA          ; Loading DELAY1 with DEC 250
        MOVWF   DELAY1
        MOVLW   0x0D          ; Loading DELAY2 with DEC 13
        MOVWF   DELAY2

        DECFSZ DELAY1, F      ; Decrement DELAY1
        GOTO    $ - 1         ; If DEALY1 != 0 then go back one instruction

        DECFSZ DELAY2, F      ; Else decrement DELAY2
        GOTO    $ - 3         ; If DELAY2 != 0 then go back three instructions

        RETURN

;=====================================================================
         END
```