2017

Computer Engineering Design 2 Phase 1 Report

UNIVERSITY OF KWAZULU-NATAL ™

INYUVESI YAKWAZULU-NATALI

Keshav Jeewanlall
213508238

8/22/2017

Contents

1 Abstract

This report explains the details of design, development and implementation of Project 1 and Project 2. Project 1 is a PIC16F690 library file coded in assembly language. This library contains two modules that is a Binary to BCD converter and a BCD to Binary converter. Project 2 is a counter which consists of two seven-segment displays and two push button switches. The counter counts up or down in the range of 0 to 99. These project codes were written and simulated on MPLAB X IDE and implemented on a PIC16F690 microcontroller.

2 Project 1 – Binary/BCD conversion

2.1 Introduction

This project served as a way in which one could become familiar with the assembly language of the PIC16F series processors and the MPLAB X IDE. The requirement of this project was to write two modules to convert between 8 bit binary and PACKED BCD and vice versa. Modular coding was the main goal in this project in addition to developing coding style and writing efficient code.

2.2 Problem Statement

The objective of this project was to create a library which consists of two modules which converts a binary number to BCD and a BCD to a binary number. The modules written in this project had to be set up in a way in which it could be used by other projects.

2.3 Research

BCD, short for Binary Coded Decimal, is numbers 0 through 9 converted to four-digit binary.

EXAMPLE:

$29_{10} = 0001\ 1101_2$

$2_{10} = 0010_2$

$9_{10} = 1001_2$

BCD = 0010 1001

2.4 Binary to BCD Conversion

2.4.1 Algorithm

The algorithm chosen for conversion from binary to BCD is the ‘shift and add 3’ method also known as the ‘double dabble’ algorithm.

The general algorithm works as follows:

1. Shift the binary number left one bit at a time.

3. After each shift If the binary value in any of the BCD columns is greater than 4, add 3 to

   that value in that BCD column.

2. Depending on the number of shifts that have taken place, the BCD number will be in the

   respective Tens and Units columns.

4. Complete after all 8 bits are shifted.

Example: $29_{10} = 000\ 1101_2$

	TENS	UNITS	BINARY
			0001 1101
Shift #1		0	001 1101
Shift #2		00	01 1101
Shift #3		000	11101
Shift #4		0001	1101
Shift #5	0	0011	101
Shift #6	00	111	01
ADD 3		1010	
Shift #7	001	0100	1
Shift #8	0010	1001	

$2_{10} = 0010_2$

$9_{10} = 1001_2$

BCD = 0010 1001

2.4.2 Code Structure

This module is set up as a library routine which can be used in other projects. It receives input via the W register and output the result in the W register. Since this code is used as a library, memory addresses and variables cannot be specified hence code that reserves memory had to be used. The input values are limited to values in the range of 0 to 99 since this project only requires values in this range and will therefore produce incorrect results for values out of this range.

2.4.3 Code Implementation

- This code uses the following registers:
  binary – Stores the BCD number.
  bcd – Stores the binary number.
  temp – Used as temporary variable for calculations
  counter – Used to control a loop
- The value in the W register is moved to the binary register.
- A loop was used to shift 1 bit at a time from the binary register to the BCD register.
- Since the first two shifts will not produce a number greater than 4 in the lower nibble, they do not have to be inspected. Inspection will only have to be done after the third shift.
- The loop will therefore run 5 times after three shifts have already been done to shift all 8 bits.
- To check if the nibble values exceeded 4, the following is done:

  For the lower nibble, the value is added to 3 and stored in the temp register. If bit 3 is set in this register, it means that this value is 8 or greater hence the value in the lower nibble exceeded 4. If this occurs, 3 is added to the lower nibble.

  A similar approach is applied for the upper nibble. Here the value is added to 48 (00110000_2) and stored in temp. If bit 7 is set, it means that the value in the upper nibble is greater than 4 hence 3 is added to the upper nibble.

- After 5 iterations the loop exits and the result is placed in the BCD register. This is then moved to the W register.

- All code was done using assembly language in MPLAB X and the PIC16F690 instruction set.

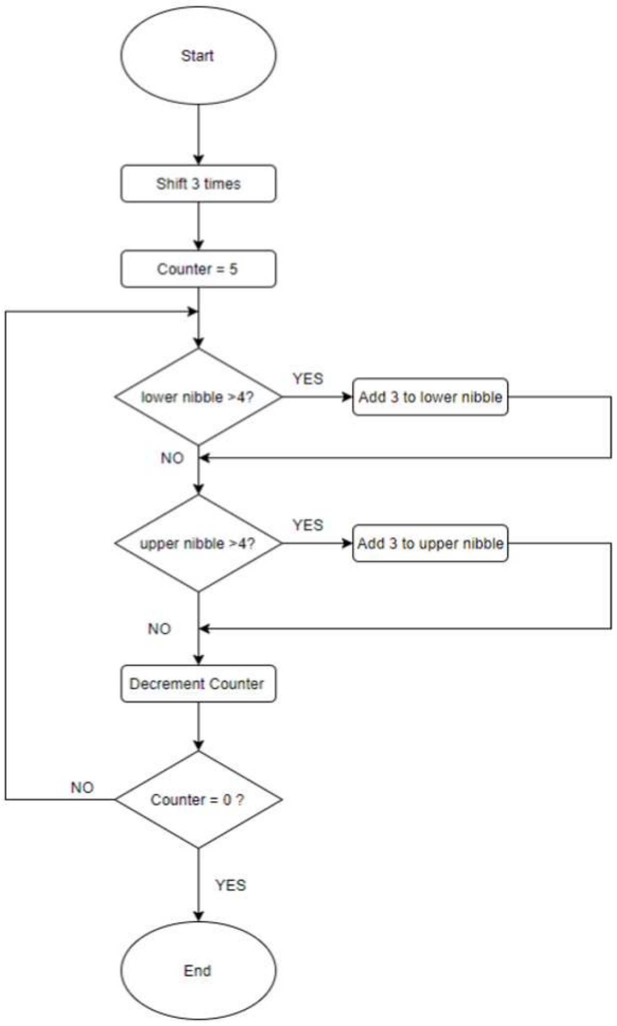
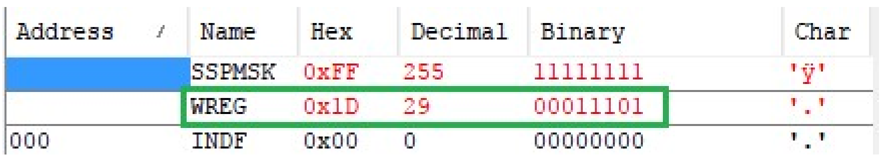


Figure 1: Binary to BCD Flowchart

2.4.4 Code Testing

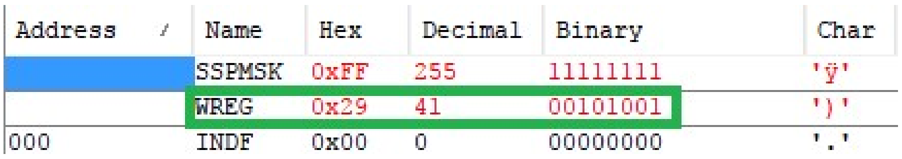
The code was debugged using the simulator in MPLAB X. Full execution of the code was analyzed to ensure the correct results were produced. Modular testing was implemented by including the library in a new project and a few random test values were used. The code produced the following results:

Value of $0001\ 1101_2$ was loaded



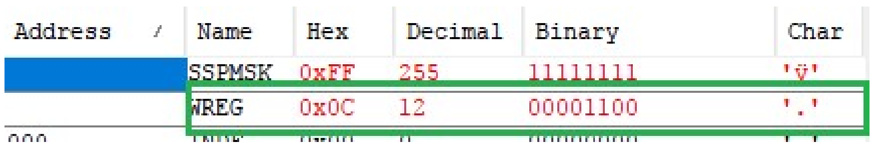
Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x1D	29	00011101	'.'
000	INDF	0x00	0	00000000	'.'

Expected value is $0010\ 1001_2$



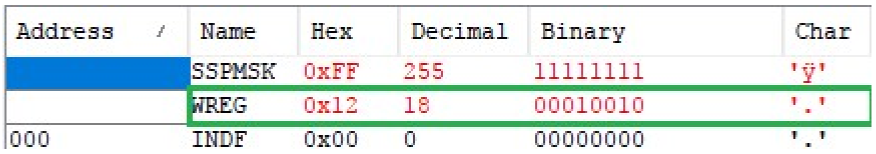
Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x29	41	00101001	')'
000	INDF	0x00	0	00000000	'.'

Value of 0000 1100_2 was loaded



Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x0C	12	00001100	'.'

Expected value is 0001 0010_2



Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x12	18	00010010	'.'
000	INDF	0x00	0	00000000	'.'

2.4.5 Code Performance

- 34 lines of code used excluding labels and comments.
- Program executes in 97 cycles & 97µs

See Appendix A for the code.

2.5 BCD to Binary Conversion

2.5.1 Algorithm

To convert from BCD to binary, firstly we must know that packed BCD is in the style [(16 * tens + units) - (6 * tens)] for example the number 29 in packed BCD is 0010 1001. Taking the tens and units which in this case is 2 and 9 respectively, using the formula (16 * 2 + 9) - (6 * 2) = 29 which is 0001 1101 in binary. Therefore, in order to convert BCD to binary we need to convert the BCD number into tens and units. This is done as follows:

1. Swap the nibbles of the BCD number and multiply by 0000 1111 to obtain the tens, move this value into a file register (temp).

2. Implement the code to obtain a value of 6*tens

3. subtract this value of 6*tens from your BCD number in order to obtain the binary equivalent

Example:

29 in BCD = 0010 1001

Swapping nibbles = 1001 0010

Multiplying by 0000 1111 using logical AND we obtain 0010 which is the tens

6 * tens = 0000 1100

0010 1001 - 0000 1100 = 0001 1101 which is 29 in binary

2.5.2 Code Structure

This module is set up as a library routine which can be used in other projects. It receives input via the W register and output the result in the W register. Since this code is used as a library, memory addresses and variables cannot be specified hence code that reserves memory had to be used. The input values are limited to values in the range of 0 to 99 since this project only requires values in this range and will therefore produce incorrect results for values out of this range.

2.5.3 Code Implementation

- This code uses the following registers:
  - binary – Stores the BCD number.
  - bcd – Stores the binary number.
  - temp – Used as temporary variable for calculations

- Move input value to bcd register
- Swap nibbles of bcd register
- AND this value with 0x0F (00001111) to obtain tens value
- move tens value to temp register
- Add this value to itself. This will result in double of the tens value
- Add the double tens value to temp register to obtain triple tens value
- Rotate this value left through carry once. This will result in 6 times the tens value
- Subtract this value from the bcd register. This binary subtraction will result in the required binary output.

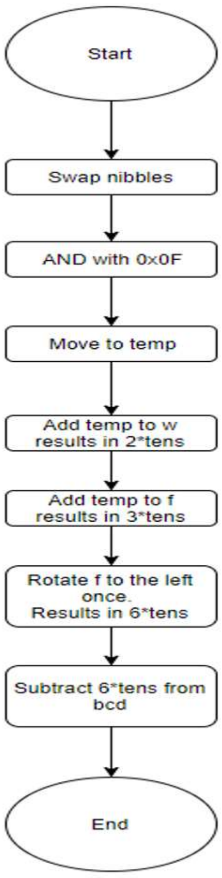
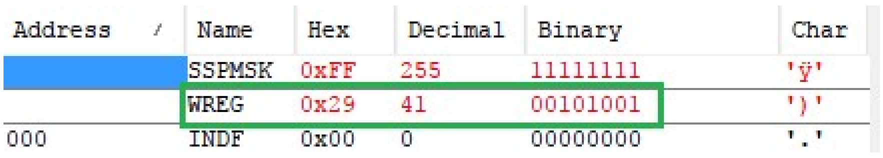

Start
Swap nibbles
AND with 0x0F
Move to temp
Add temp to w results in 2*tens
Add temp to f results in 3*tens
Rotate f to the left once. Results in 6*tens
Subtract 6*tens from bcd
End

Figure 2: BCD to Binary flowchart

2.5.4 Code Testing

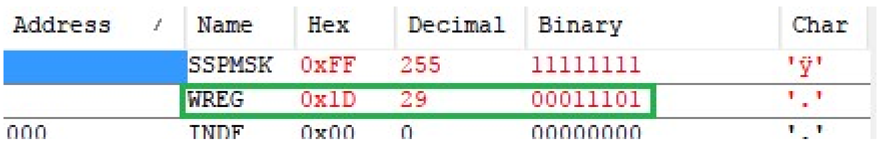
The code was debugged using the simulator in MPLAB X. Full execution of the code was analyzed to ensure the correct results were produced. Modular testing was implemented by including the library in a new project and a few random test values were used. The code produced the following results:

Value of $0010\ 1001_2$ was loaded



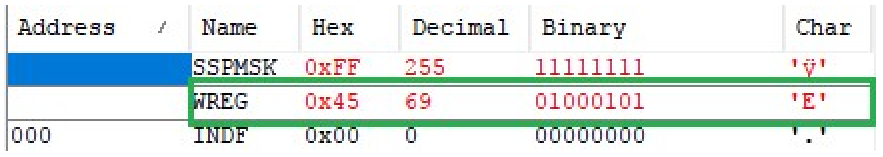
Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x29	41	00101001	')'
000	INDF	0x00	0	00000000	'.'

Expected value is $0001\ 1101_2$



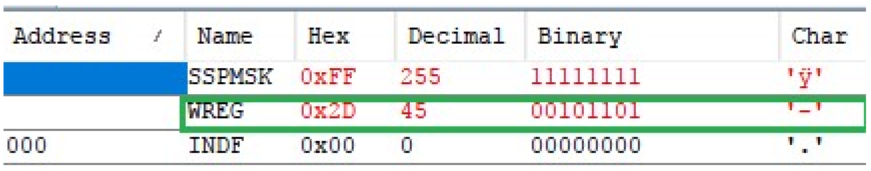
Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x1D	29	00011101	'.'
000	INDF	0x00	0	00000000	'.'

Value of $0100\ 0101_2$ was loaded



Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x45	69	01000101	'E'
000	INDF	0x00	0	00000000	'.'

Expected value is $0010\ 1101_2$



Address	Name	Hex	Decimal	Binary	Char
	SSPMSK	0xFF	255	11111111	'ÿ'
	WREG	0x2D	45	00101101	'-'
000	INDF	0x00	0	00000000	'.'

2.5.5 Code Performance

- 9 lines of code used excluding labels and comments.
- Program executes in 11 cycles & 11µs

See Appendix A for the code.

3 Project 2 – Version 2

3.1 Introduction

This project served as a way in which to gain an insight of the structured approach to design. Therefore, version 1 had to be completed first and then version 2. A top down approach was used in planning this design. In-Circuit Serial Programming (ICSP) was implemented throughout this project. All coding was done in assembly language through MPLAB X IDE and a PicKit3 programmer was used to program the PIC16F690 microcontroller.

3.2 Problem Statement

Design and program a microcontroller to produce a system that counts up and down from 0 to 99 using two SPST pushbuttons and two Seven-segment displays. If the count is 99 and the count up button is pressed, the count will reset to 0 and If the count is 0 and the count down button is pressed, the count will reset to 99. The value displayed on the two seven-segment displays are to be multiplexed representing the tens and units values. Debouncing of the switches is to be implemented in software.

3.3 Setup

3.3.1 Components Used

- PIC16F690 microcontroller
- 2 x SPST Push button switches
- 2 x Pull up resistors (1kΩ)
- 2 x Common anode seven-segment displays
- 7 x Current limiting resistors (330Ω)

3.3.2 Display Setup

Common anode displays are used in order to make this design more efficient. The common anode displays will pull less current from the microcontroller and hence produce a current sink because the segments light up when given a low signal. Due to the limit on the number of pins, the SSDs need to be multiplexed.

The following table contains the values to be sent to the SSD in order to display each digit from 0-9.

	g	**f**	**e**	**d**	**c**	**b**	**a**	**CODE (hex)**
0	1	0	0	0	0	0	0	40
1	1	1	1	1	0	0	1	79
2	0	1	0	0	1	0	0	24
3	0	1	1	0	0	0	0	30
4	0	0	1	1	0	0	1	19
5	0	0	1	0	0	1	0	12
6	0	0	0	0	0	1	0	02
7	1	1	1	1	0	0	0	78
8	0	0	0	0	0	0	0	00
9	0	0	1	1	0	0	0	10

3.3.3 PIC16F690 Pin Configuration

This project needs receives two inputs, one from each pushbutton. The PIC16F690 has only one external interrupt therefore one of the pushbuttons needs to be read using a polling method. The Count up button was chosen to be connected to the external interrupt pin RA2 of the microcontroller and the Count down button was chosen to be connected to RA5 on the microcontroller which will be polled.

The two SSDs will be connected to PORTC as this is the only port with 8 pins. Pin RC0 to RC6 will be connected to pin a-g on the SSD. Pin RC7 will be used as an enable pin for the tens SSD and pin RA4 will be used as an enable pin for the units SSD. Switching these two SSDs between opposite logic will allow for multiplexing of the SSDs.

3.3.4 Switch Setup and Debouncing

The switches are set up as pull up switches, this sends a low signal to the microcontroller when the switch is pressed.

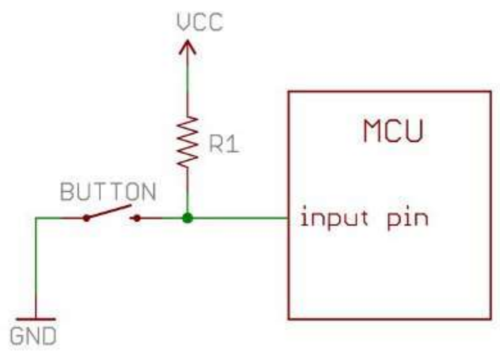


Figure 3: Switch setup circuit

The switches are debounced in software by using a loop to check if the switch is still being pressed. It checks for this and then commences with the rest of the code. The loop produces enough time for the switch to debounce and then commence with the rest of the code.

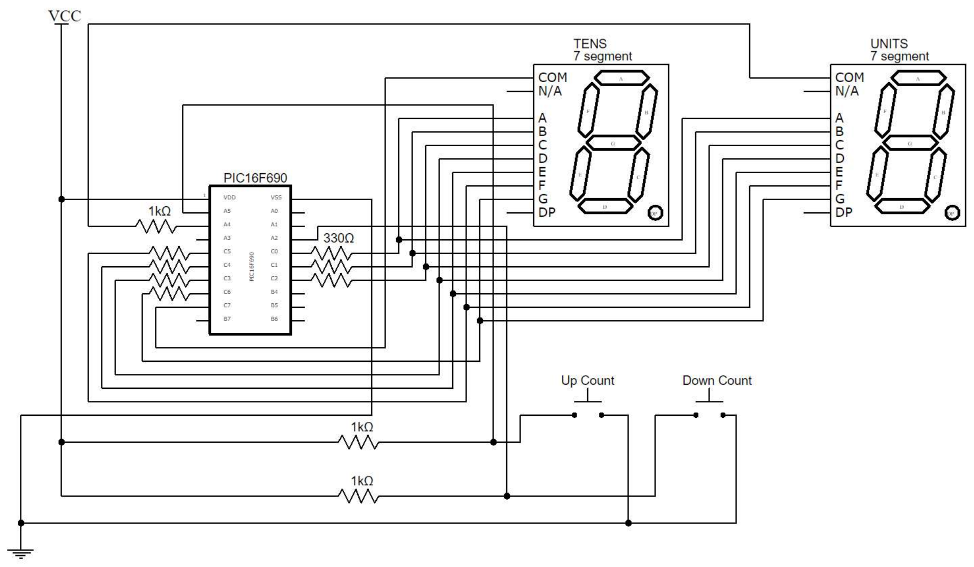


Figure 4: Schematic of the system

3.4 Code

3.4.1 Code Structure & Implementation

First the necessary registers and ports need to be setup in order to allow for input, output and external interrupts. The signals received are digital inputs therefore microcontroller needs to be set up for digit I/O. This is all done in a setup phase which is executed as soon as the microcontroller powers on. Before the setup code, the code for the Interrupt Service Routine is set up to increment the count. This count resets to 0 if the value of 99 is exceeded. An endless display loop is then executed after the set up phase. In this loop, the Display subroutine is called which contains all code required for the microcontroller continuously send the specified signals to the SSDs in order to display the count value.

6 General Registers are used:

- count - Stores the count value
- tempW - Used for contents saving (WREG)
- tens - Stores the 'tens' value to be displayed
- units - Stores the 'units' value to be displayed
- temp - used during switch debouncing
- temp2 - used during switch debouncing

Setup:

- In Bank 1, the Global interrupts and external interrupt services are enabled, PORTC is setup as output, in RA2 and RA5 is configured for input, pin RA4 is set up as output and bit 6 in OPTION_REG is set to allow interrupt on RA2 to occur on falling edge of signal.
- In Bank 2, the digital I/O on all ports are enabled here by setting ANSEL to zero.
- In Bank 0, all general purpose registers used are cleared.

Display_loop:

- The Display subroutine is called.
- Pin RA5 is polled. If low signal detected, this means the switch is pressed hence call Counting_Down subroutine.

Display:

- The Convert_to_BCD subroutine is called.
- The Convert_to_BCD subroutine moves the count value to the W register and calls the library subroutine Binary_To_BCD.
- The Binary_To_BCD converts the binary value to BCD
- The Convert_to_BCD subroutine then takes this value and separates it into tens and units and then stores the tens and units values in their respective registers
- The tens value is then moved to the W register. A table which returns the required code in the W register is then called.
- A value of 128_{10} (10000000_2) is added to the W register, this enables the tens SSD by setting the MSB (PORT<7>) to 1. This is then moved to PORTC to display the tens.
- A similar method is used to display the Units although by leaving bit 7 in PORTC as 0, this will disable the Tens SSD. By setting PORTA<4> to 1, this will enable the Units SSD so the units can be displayed.

Counting_Down:

- Debounce the switch.
- Decrements the count.
- If count reaches -1, count will reset to 99. This is done by checking if bit 7 in count is set, if so, it means the number has reached -1.
- Wait for switch to be released. This is done in a loop that constantly checks if the switch is pressed, releasing the switch exits the loop.

Count_Up_Interrupt:

- Backup W register contents in a temp register.
- Debounce the switch. This is done by checking if the button is pressed using a loop.
- Increment the count.
- Check if number exceeds 99. This is done by adding the count to 28. If bit 7 in this register is set, it means number is greater or equal to 100. Therefore, reset the count to 0.
- Re-enable the interrupt by clearing bit 1 in the INTCON register.
- Restore the W register contents.
- Return from the interrupt.

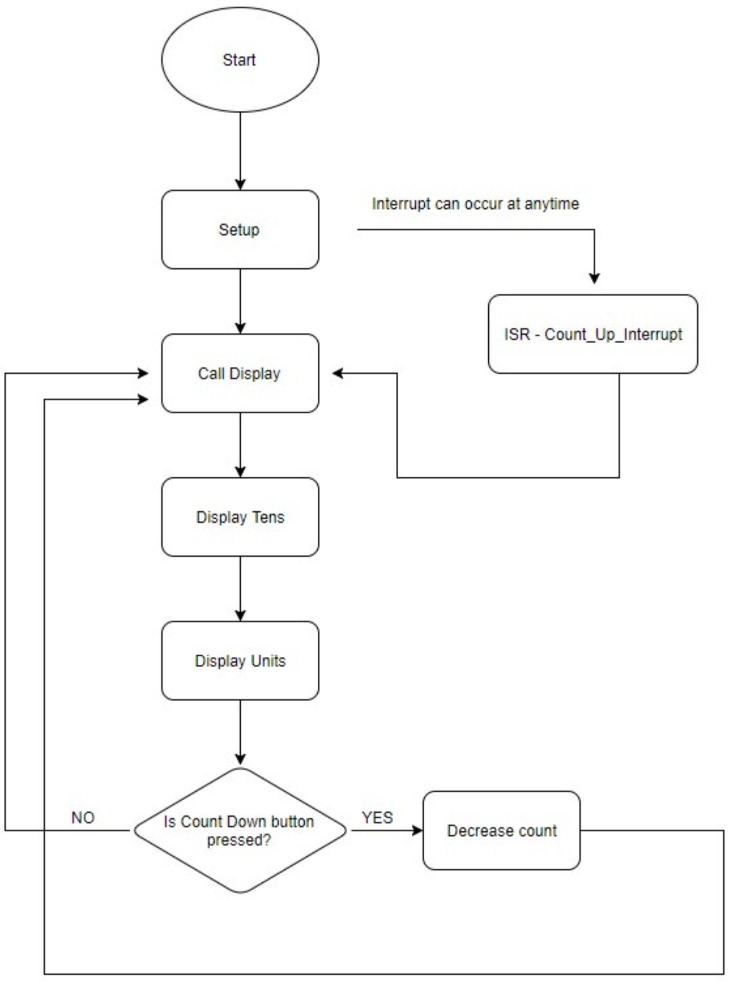


Figure 5: Project 2 code flowchart

3.4.2 Testing

All debugging and testing was done on a PIC16F690 microcontroller with all the components connected to it. Debugging using the MPLAB X simulator was not the most efficient way to debug this project as hardware interaction was involved. The code was programmed to the microcontroller using a PicKit3. The system performed to most specifications.

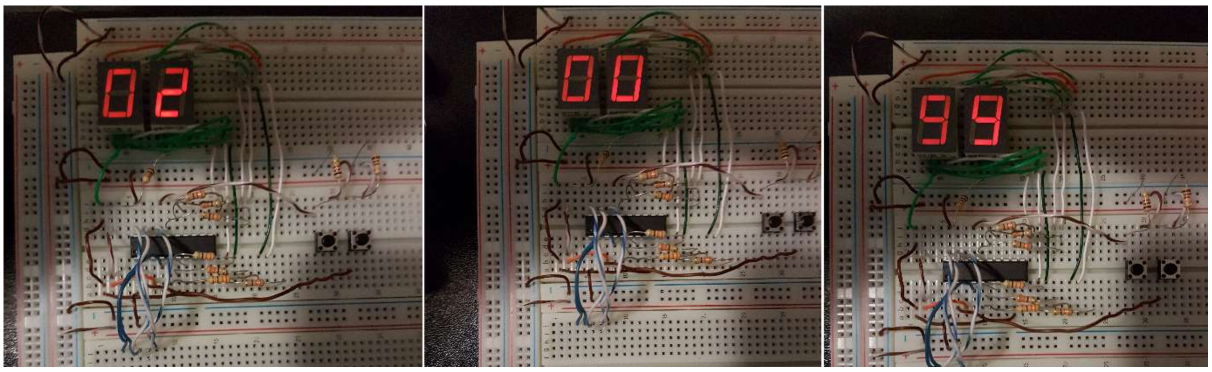


Figure 6: Breadboard design of system

See Appendix B for the code.

4 References

[1] Eng.utah.edu. (2017). Binary to BCD. [online] Available at: http://www.eng.utah.edu/~nmcdonal/Tutorials/BCDTutorial/BCDConversion.html [Accessed 27 Jul. 2017].

[2] Piclist.com. (2017). PIC Microcontoller Radix Math Method BCD packed to binary 2 digit to 8 bit. [online] Available at: http://piclist.com/techref/microchip/math/radix/bp2b-2d8b.htm [Accessed 29 Jul. 2017].

[3] PIC16F685/687/689/690 Datasheet. (2005). Microchip Technology Inc.

[4] PIC Programming in Assembly. (n.d.). [ebook] Available at: http://groups.csail.mit.edu/lbr/stack/pic/pic-prog-assembly [Accessed 19 Aug. 2017].

[5] YouTube. (2017). 8 bit binary to bcd conversion in 8086 microprocessor. [online] Available at: https://www.youtube.com/watch?v=l2R0LLG4tn0 [Accessed 29 Jul. 2017].

[6] YouTube. (2017). Shift Add 3 Method | Simple method for Binary to BCD conversion. [online] Available at: https://www.youtube.com/watch?v=IBgiB7KXfEY [Accessed 29 Jul. 2017].

5 Appendix A – Binary/BCD Conversion Code

```
;******************************************************************************
;
;    Student Name          : Keshav Jeewanlall
;    Student Number        : 213508238
*
;    Description           : Convert Binary to BCD Code and reverse
;
;    The following code contains two modules. Module 1 converts Binary
;    numbers to BCD code. Module 2 converts BCD code to a Binary number
;
;******************************************************************************

List p = 16f690
#include <p16F690.inc>


;Variable Declaration

GPR_Variables     UDATA
binary           RES 1
bcd               RES 1
counter           RES 1
temp              RES 1


;RES instruction is used to reserve memory since memory addresses cannot be
;specified in relocatable code.


CODE

;******************************************************************************
;
;                   Module 1 - Binary number to BCD code
;
;  This code uses the Shift and add 3 algorithm to convert binary numbers to
;  BCD
;
;  The general algorithm works as follows:
;  1. Shift the binary number left one bit at a time.
;  2. Depending on the number of shifts that have taken place, the BCD number will
;     be in the respective Tens and Units columns.
;  3. If the binary value in any of the BCD columns is greater than 4, add 3 to
;     that value in that BCD column.

;  In this code, the binary number is shifted one bit at a time into a new
;  register. This register consists of the Tens (upper nibble) and Units (lower
;  nibble). After the first 2 shifts, the two nibbles are inspected and if their
;  value is greater than 4, 3 is added to that nibble.


Binary_To_BCD

    GLOBAL Binary_To_BCD   ;Made global in order to be accessed externally
    MOVWF binary           ;Move number from W register to file register
    MOVLW 0x05             ;loop counter needs to run 5 times
    MOVWF counter          ;move the value of 5 into the counter register


    CLRF bcd              ;clear bcd register
    CLRF temp             ;clear temp register

    RLF binary,1
    RLF bcd,1
    RLF binary,1
    RLF bcd,1
```

```
 ;only after the 3rd shift nibbles need to be evaluated because the first two
 ;shifts will not result in a value greater than 4 eg. 0011 in binary is 3

    RLF binary,1
    RLF bcd,1


BCD_loop

;To check if the value in the nibble is greater than 4, 3 is added to the nibble
;and stored in a temp register. If the value in temp is greater than 7,
;it means the value in the nibble was greater than 4. Due to this, bit 3 or 7
;in temp will be set because the value in temp will be 8 or larger and we can
;test for this.


    ;Evaluating the lower nibble

        MOVFW bcd
        ADDLW 0x03
        MOVWF temp
        BTFSC temp,3

    ;if bit 3 in temp is set, call function to add 3 to lower nibble
        CALL Lower_nibble_Add3


    ;Evaluating the upper nibble

        MOVFW bcd
        ADDLW 0x30
        MOVWF temp
        BTFSC temp,7
    ;if bit 7 is set, call function to add 3 to upper nibble

        CALL Upper_nibble_Add3

        RLF binary,1
        RLF bcd,1


    ;loop must run 5 times to ensure all remaining bits are shifted after first
    ;3 shifts
        DECFSZ counter
        got BCD_loop

        MOVFW bcd
    RETURN


;function to add 3 to lower nibble and store result in bcd register
Lower_nibble_Add3
        MOVLW 0x03
        ADDWF bcd,1
    RETURN


;function to add 3 to upper nibble and store result in bcd register

Upper_nibble_Add3
        MOVLW 0x30
        ADDWF bcd,1
    RETURN


;*****************************End of Module 1***********************************
```

```
;********************************************************************************
;                  Module 2 -  BCD Code To Binary number
;
; To convert from BCD to binary, firstly we must know that packed BCD is in
; the style [(16 * tens + units) - (6 * tens)] for example the number 29 in
; in packed BCD is 0010 1001 so taking the tens and units which in this case
; is 2 and 9 respectively, using the formula (16 * 2 + 9) - (6 * 2) = 29
; which is 0001 1101 in binary.
;
; Therefore, in order to convert BCD to binary we need to convert the BCD
; number into tens and units. This is done as follows:
;
; 1. Swap the nibbles of the BCD number and multiply by 0000 1111 to obtain
;    the tens, move this value into a file register (temp).
; 2. Implement the code to obtain a value of 6*tens
; 3. subtract this value of 6*tens from your BCD number in order to obtain
;    the binary equivalent


BCD_To_Binary

    GLOBAL BCD_To_Binary
        MOVWF bcd           ;move bcd value to bcd register
        SWAPF   bcd, W      ;swap nibbles of bcd register
        ANDLW   0x0F         ;ANDLW with 0x0F, W register will contain the tens
        MOVWF   temp         ;move tens to temp register
         ADDWF   temp, W      ;this instruction results in w = 2*tens
         ADDWF   temp, F      ;temp = 3*tens (note carry is cleared)
         RLF     temp, W      ;by rotating f left through carry once W = 6*tens
         SUBWF   bcd, W       ;W = 16*tens+ones - 6*tens. Binary number obtained
         MOVWF   binary


;*********************************REFERENCES**********************************
; Dattalo, S. (n.d.). BCD packed to binary 2 digit to 8 bit. piclist.com.

;****************************End of Module 2**********************************
END
```

6 Appendix B – Project 2 Version 2 Code

```
;*******************************************************************************

;    Student Name          : Keshav Jeewanlall
;    Student Number        : 213508238
;    Date                  : 22 / 08 / 2017
;    Description           : Count up & down (from 0 to 99) using two
;                              push button switches & two SSDs
;
; Counting up is done using a push button as an external interrupt on pin RA2.
; Counting down is done using a push button that is being polled via pin RA5.
;*******************************************************************************


    List p=16f690
#include <p16F690.inc>
errorlevel  -302
    __CONFIG    _CP_OFF & _CPD_OFF & _BOR_OFF & _MCLRE_ON & _WDT_OFF & _PWRTE_ON &
_INTRC_OSC_NOCLKOUT & _FCMEN_OFF & _IESO_OFF


;************************VARIABLE DEFINITIONS & VECTORS*************************
 UDATA
tempW           RES 1     ;temporarily stores value in W register when
                           ;interrupt occurs
count            RES 1     ;stores the count of the button press
tens             RES 1     ;stores tens digit
units            RES 1     ;stores units digit
temp            RES 1
temp2            RES 1

EXTERN Binary_To_BCD      ;library to convert binary to BCD

RESET ORG 0x00            ;Reset vector, PIC starts here on power up and reset
GOTO Setup

ORG 0x04                  ;The PIC will come here on an interrupt
                          ;This is our interrupt routine that we want
                          ;the PIC to do when it receives an interrupt

;*******************************INTERRUPT ROUTINE**********************************

Count_Up_Interrupt
    MOVWF tempW
    CALL Switch_Debounce    ;call switch debouncing module
    INCF count            ;increase count on button press
    MOVLW 0x1C            ;adds 28 to count
    ADDWF count,0         ;if count is 128 or greater, bit 7 will set
    MOVWF temp
    BTFSC temp,7          ;if bit 7 is clear, number is under 100
    CLRF count            ;reset the number count after 99
    BCF INTCON,1          ;enable the interrupt again
    MOVFW tempW          ;restore W register value from before the interrupt
    RETFIE                ;This tells the PIC that the interrupt routine
                          ;has finished and the PC will point back to the
                          ;main program


;**************************SETUP OF PIC16F690 PORTS*****************************
Setup
                      ;Select Bank 1
    BSF STATUS,5
    BCF STATUS,6

    BSF INTCON,7      ;enable Global Interrupt
    BSF INTCON,4      ;enable External Interrupt
    CLRF TRISC        ;set PORTC as output for ssds
    CLRF TRISA
    BSF TRISA,2               ;set pin RA2 as input for count up button
    BSF TRISA,5               ;set pin RA5 as input for count down button
    BCF OPTION_REG, 6  ;setting bit 6 of OPTION_REG ensures that the interrupt
                       ;occurs on the falling edge (Button Press)

                       ;select Bank 2
    BCF STATUS,5
    BSF STATUS,6

    CLRF ANSEL         ;enable digital I/O on ports

                       ;select Bank 0
    BCF STATUS,6

                       ;clear registers
    CLRF count
    CLRF tempW
    CLRF temp
    CLRF temp2
    CLRF tens
    CLRF units

    GOTO Display_loop  ;go to the main loop
```

Code

```
;***************************CODE FOR DISPLAYING ON SSDs*************************

Display_loop
   CALL Display
   BTFSS PORTA,5        ;checks if counting down button is pressed
   CALL Counting_Down   ;calls subroutine if it is pressed
GOTO Display_loop

Display
    CALL Convert_to_BCD         ;subroutine to convert count to BCD
    BCF PORTA,4                 ;clear pin RA4 to disable units SSD
    CALL SSD_Table       ;gets code for displaying the number (Tens)
    ADDLW 0x80           ;setting the MSB (Bit 7) will enable the Tens SSD
    MOVWF PORTC                 ;display Tens value

    BSF PORTA,4                 ;Set pin RA4 to enable units SSD
    MOVFW units
    CALL SSD_Table       ;gets code for displaying the number (Units)
    MOVWF PORTC                 ;displays units value

    CALL Multiplexing_Delay  ;delay for multiplexing SSDs

    RETURN


Convert_to_BCD           ;converts count to BCD
    MOVFW count
    Call Binary_To_BCD   ;uses library subroutine to get BCD value of number
    MOVWF tens
    ANDLW 0x0F           ;b'00001111 , clears upper nibble of BCD number
    MOVWF units                 ;stores the value as the units
    SWAPF tens,1         ;swaps the nibbles of the BCD number
    MOVFW tens
    ANDLW 0x0F            ;b'00001111, clears the high nibble to get tens value
    MOVWF tens            ;stores value in tens register
    RETURN
```

```
 SSD_Table
                        ;These HEX values are required because common anode SSDs
                        ;are being used
    ADDWF PCL,1
    RETLW 0x40          ;displays number 0 on SSD
    RETLW 0x79          ;displays number 1 on SSD
    RETLW 0x24          ;displays number 2 on SSD
    RETLW 0x30          ;displays number 3 on SSD
    RETLW 0x19          ;displays number 4 on SSD
    RETLW 0x12          ;displays number 5 on SSD
    RETLW 0x02          ;displays number 6 on SSD
    RETLW 0x78          ;displays number 7 on SSD
    RETLW 0x00          ;displays number 8 on SSD
    RETLW 0x10          ;displays number 9 on SSD


 Counting_Down          ;subroutine used for counting down
    CALL Switch_Debounce
    DECF count          ;Decreases the number count by 1
    BTFSS count,7       ;If -1 occurs, bit 7 will be set, reset count to 99
    GOTO No_Reset       ;else skip
    MOVLW 0x63
    MOVWF count
  No_Reset
    CALL Display        ;ensures that count is decreased on button press
  Constant_Display_loop        ;waits till button gets released to avoid changing
                        ;count when button is held down
    BTFSC PORTA,5       ;if switch is released
    RETURN              ;exit the loop
    GOTO Constant_Display_loop ;else wait until switch is released

Multiplexing_Delay      ;delay for multiplexing SSDs to allow for
                        ;equal brightness
    MOVLW 0x32
    MOVWF temp
  Multiplex_loop
    DECFSZ temp
    GOTO Multiplex_loop
    RETURN


;****************************SWITCH DEBOUNCING CODE*****************************
Switch_Debounce
    MOVLW 0xC8            ;Switch Debouncing
    MOVWF temp
    MOVWF temp2
Debounce_loop
    DECFSZ temp2
    GOTO Debounce_loop
    DECFSZ temp
    GOTO Debounce_loop    ;End of switch debouncing
    RETURN

End
```