University of Jordan

School of Engineering

Department of Mechatronics Engineering

Microprocessor and Microcontroller Laboratory / 0908432

Lab Manual

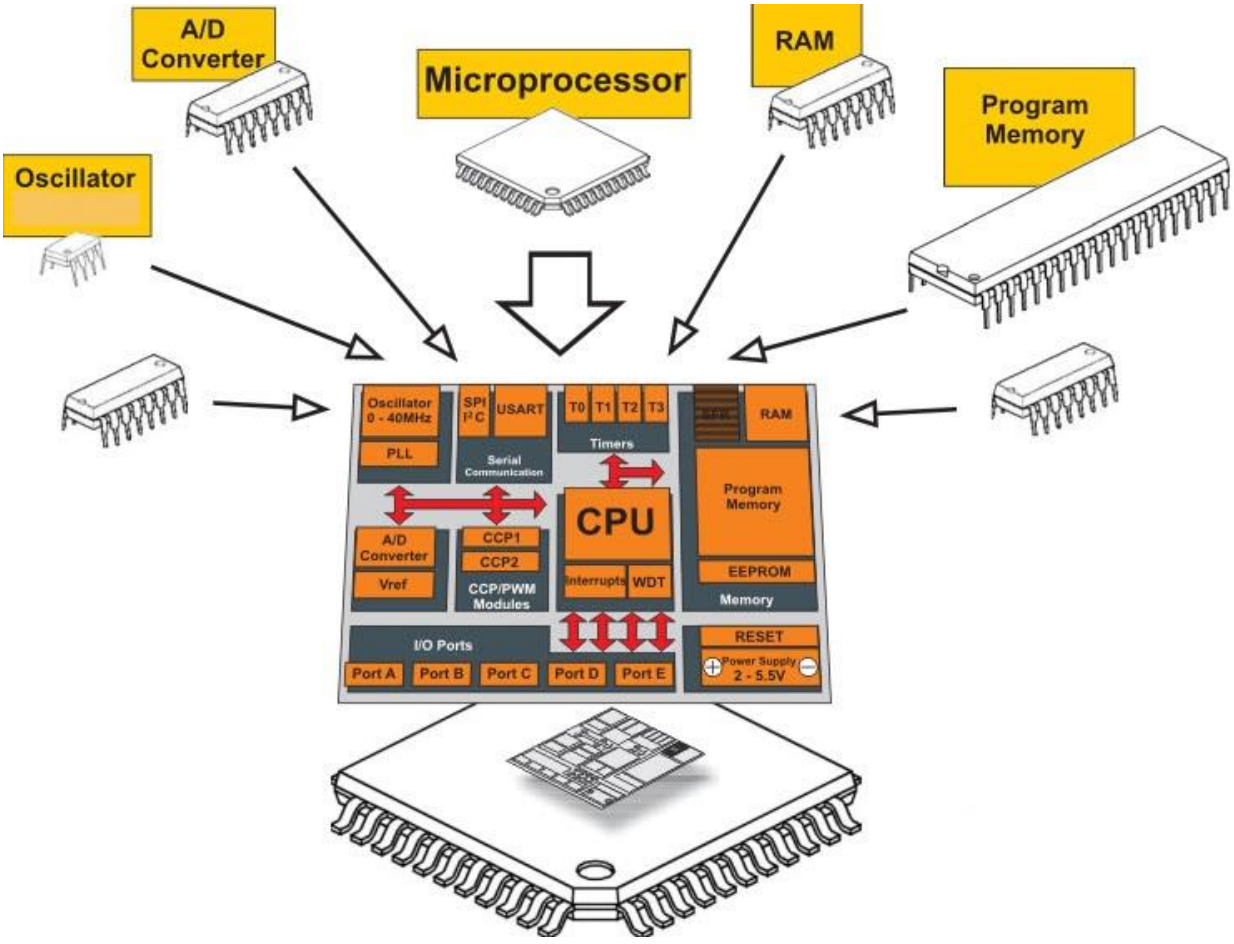


Table of Contents

University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Introduction to the Lab

Administrative Policy of the Laboratory

1) You are not allowed to smoke, eat, or drink in the Laboratory. You are expected to conduct yourself professionally, and to keep your bench area ***clean*** **and** ***neat***.

2) MXE432 is a time controlled closed lab, therefore, you are expected to ***write*** and ***test*** your code or ***build*** your circuit in the lab within the allotted time. You cannot ***write*** and ***test*** the code or build the circuit ahead of time. However, make sure to solve and bring all prelab material before hand.

3) Lab reports must be submitted at the beginning of the next experiment only. **No reports will be accepted after that time.**

4) You can discuss the experiment and the results with your colleagues, but each student must submit her/his own personally written report. Cheating and copying of reports is strictly prohibited and will be taken very seriously. The student will earn a **ZERO** in the lab when caught.

5) All questions should be solved in order. Moreover, each student is expected to demonstrate her/his solution fully and clearly whenever required.

6) No one can leave the lab until she/he has cleaned and arranged her/his bench and turned off the PC she/he used.

7) Always ask your instructor to check your setup before turning the power on.

8) The above mentioned polices should be strictly followed. Note that disregarding any of the rules above will seriously affect your grade!

9) **Makeup Midterm**: There will be no make-up for the midterm. In case of medical/ or other disabling emergencies, the instructor should be notified **before** the midterm and his approval for missing the midterm should be obtained before the midterm. If for any reason the instructor could not be reached, the department secretary should be notified before the midterm. The phone number is 535-5000 Extension 23025

10) Grading Corrections: Ask the instructor for any grading correction requests within a week of returning the report/exam/quiz papers. After that, your grade will not be adjusted. If you find any mistake in grading, please let the instructor know. Your grade will not be lowered.

11) Class Attendance: Class attendance will be taken. University regulations regarding attendance will be strictly enforced.

Prelab and Report Instructions:

- ✓ Some experiments contain requirements that need to be prepared before coming to the lab.
- ✓ You are required to prepare all prelab work before the beginning of the lab session. For the written questions you are required to write the solutions on separate sheet. **Remember** no copying from other student is allowed.
- ✓ For any questions, you can submit your questions to the instructor using email. The instructor's emails are as follows:

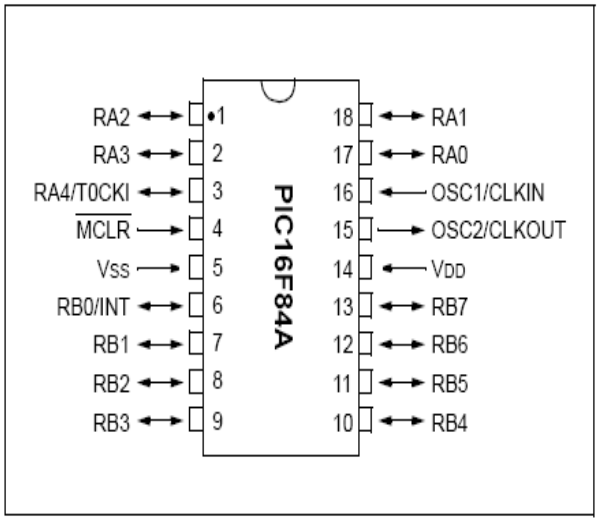
  Dr. Musa Alyaman: m.alyaman@ju.edu.jo

  Eng. Hisham Hatem hishamhatem89@gmail.com

  Please make sure to put in the **subject line** of your email message **your full name** and **your lab session numbe**r for identification.

PIC16F84A 8-bit Microcontroller

Important PIC16F84 Features:

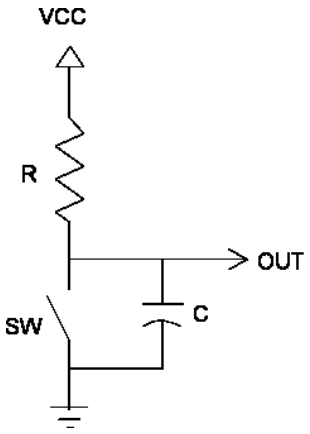


- Only 35 single word instructions
- Operating speed: DC - 20 MHz clock input
- 1024 words of program memory
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
- 10,000 erase/write cycles Enhanced FLASH Program memory typical
- Low power, high-speed technology

Mechanical Switch De-bouncing

The push-button switches are often used to provide input to digital systems. However, mechanical switches do not open or close cleanly. When a switch is pressed, it makes and breaks contacts several times before settling into its final position. This causes several transitions or "bounces" to occur. To correct this situation a de-bounce circuit is connected to the switches, thus removing the series of pulses generated by the mechanical action of the switch.

The most basic circuit used to de-bounce a switch is shown below. It consists of a resistor and a capacitor in series. The resistor and capacitor values must be chosen such that the RC time constant is greater than the bounce time.



Objective

To be familiar with Microchip MPLAB Integrated Development Environment (IDE) and the whole process of building a project, implementing, modifying simple codes, compiling the project, and simulating the code.

Pre-lab Preparation:

1- Read the PIC16F84A data sheet chapters 1, 2 especially (2.1, 2.2, 2.3).
2- Review the sections in the book regarding the Memory (Chapter 2) and MPLAB (Chapter 4).

Procedure:

This lab experiment is composed of two parts. All parts involved using MPLAB and implementing codes to learn key issues.

Part 1

In this section we will learn the steps necessary to create a project using MPLAB and then once created, we will learn how to compile it to create the necessary files that will allow us to simulate the project or alternatively, to program the microcontroller with the machine code generated.

[ ] Create a directory on the PC in D drive under the Lab2 folder in which to store all of your work.

[ ] Open a new text document and write the following:

```
movlw 06
movwf 01
 nop
 nop
   end
```

[ ] Save the text file, you created with **Lab2_P1.asm (Make sure that the extension is .asm and not .txt).**

[ ] Start the MPLAB software on your PC.

[ ] To create a project in MPLAB, follow the following simple steps: Select the Project → Project Wizard menu item. In the device selection menu, choose 16F84A. Click next. In the Active Toolsuite, choose Microchip MPASM Toolsuite. Click next. Name the project Lab2Part1. For the project directory, make sure to browse to your created subdirectory. Click next. Add the file called Lab2_P1.asm. Make sure to check the box next to the name after the file has been added. Click next. Click Finish.

[ ] From the window Tab, select the Lab2Part1.mcw window. Double click on the Lab2_P1.asm file name in the project file tree. The file Lab2_P1.asm should open now in the editor window. This is where you will usually write your programs, debug them and simulate them.

[ ] To compile your project; from the Project Tab, choose Build All. Your program should compile now, and you should see a small window showing details of the compilation process and the following message "Build Succeeded" (Note the new files generated in your directory named: Lab2_P1.hex, Lab2_P1.lst, and Lab2_P1.err).

Exercise 1:

Write a code segment that initializes the INTCON register with the value 5. (You can get the address of INTCON register from the data memory map in PIC16F84A Datasheet.) Then compile it and make sure that there are no error messages.

Note: save the file as EX1.asm and the project Lab2_ex1.

Part 2

In this part we will use the same code used in Part1.

Simulation is a very powerful tool in the hands of the embedded system developer. It allows us to run the code we have written on the computer and check whether it is working properly as expected without having to program the chip. In this part we will see an example of some of the abilities of the simulator.

[ ] From the Debugger Tab, choose Select Tool, and then enable MPLAB SIM for the simulator. Then from the same Tab, go to Settings. Select the Osc / Trace Tab and set the desired processor frequency to 4 MHz. This will tell the simulator in MPLAB to assume that Fosc is 4 MHz. Click OK to close the settings window.

[ ] From the Window tab, select the Tile Horizontally. This will show you all the current active windows in your project.

[ ] Select the Debugger → Reset, and then choose the Processor reset menu item or press F6. The software should highlight (with a green arrow). In this step, you have told the simulator to start behaving as if the microcontroller has just been given power. So, it is now ready to start execution of your program. It is now ready and waiting at the reset vector for your next command. Note that the program is not running yet.

[ ] Select the View → Watch Window. From the SFR drop down list, choose TMR0. Click the Add SFR button. Repeat the same procedure but for INTCON and OPTION _REG. You should now note that your new Watch Window has these four register names listed, along with their addresses and contents. Select the TMR0 row. Right Click and select the Properties button. Note that you can view a register as hex, decimal, binary, or ASCII.

[ ] Hit F7 to step through the program one instruction at a time. Notice the PCL register counting in the status bar. The PCL register is the low byte of the program counter and shows what address in memory the microcontroller is going to execute next. This method of using the simulator is very useful when you want to check for errors and for debugging purposes. (Note that the value of TMR0 register after the execution of these two instructions is 06).

Exercise 2:

Write a code segment that initializes the TRISB register with the value 8. You can get the address of TRISB register from the data memory map in PIC16F84 Datasheet. To ensure that you wrote a correct code view the TRISB register from Watch window, step through the code, and notice if the value of TRISB is being initialized correctly, if not why?

Exercise 3:

Write a code segment that initializes the register with address 0X186 with the value 10, step through the code, and notice if the value of register is being initialized correctly, if not why?

……

……

Exercise 4:

Write a code segment that initializes the register with address 0X1E with the value 13, step through the code, and notice if the value of register is being initialized correctly, if not why?

……

……

Exercise 5:

Write a code segment that initializes the register with address 0X9A with the value 7, step through the code, and notice if the value of register is being initialized correctly, if not why?

……

……

Exercise 6:

Write a code segment that initializes the register with address 0X6A with the value 9, step through the code, and notice if the value of register is being initialized correctly, if not why?

……

……

Objective

To be familiar with assembly language programming and the Microchip PIC 16 series instruction set.

Pre-lab Preparation:

Review Experiment 2 thoroughly.
Read chapter 7 of the PIC16F84A data sheet.
Review the Status Register- Section 2.2.1 in the book

Procedure:

This lab experiment is composed of two Parts. The first part introduces the theory behind assembly language programming and machine code format. The second part is an interactive one where you will be introduced to some PIC instructions and investigates their syntax, parameters, and usage. The experiment involves using MPLAB and implementing codes to learn key issues.

Part 1: (Theory)

Introduction to Assembly Language and the PICMicro ISA (Instruction Set Architecture)

Embedded systems combine both hardware and software aspects. The hardware evolved to a high degree of integration that has been mostly integrated in modern ICs. In addition, programming also evolved from directly writing machine codes to assembly and higher-level languages such as C.

Why use assembly while we have the high-level-language "HLL" alternatives?
Assembly once learnt and professionally used offers several advantages over HLL programming in that the professional programmer can use it to write **smaller codes** in comparison with that produced by HLL code compilers "this is due to compiler inefficiency". **Shorter codes execute fast and therefore beneficial when it comes to real-time application requirements**. Moreover, to keep costs low and reduce power consumption, memories integrated into microcontrollers are small, so it is important for the programmer to write minimal codes for his complex programs to fit in.
On the other hand, using HLL reduces code complexity, simplifies code debugging and leads to faster product development which offers shorter time to market. Such aspect is important in today's competitive market.

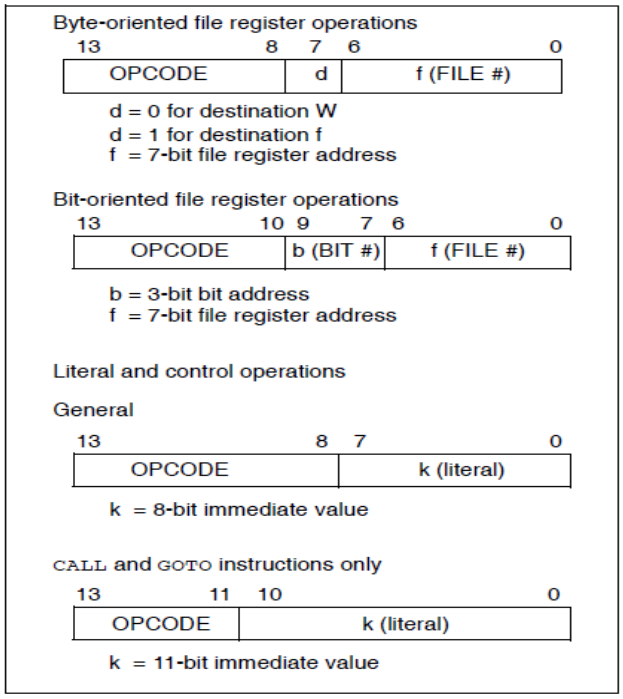
Introduction to the PIC 16 series machine code

Each PIC16XXX instruction is a 14-bit word, divided into an OPCODE which specifies the instruction type and one or more operands which further specify the operation of the instruction. Here, another classification of the instruction introduces itself according to the instruction format
The PIC16XXX instruction set is divided into:

- ***Byte-oriented*** instructions, which are so, named because they deal with whole registers (byte wide).
- ***Bit-oriented*** instructions which affect single bits in registers
- ***Literal* instructions** which contain literals (constant numbers) within the same instruction

- ***Control* instructions**, which alter the flow of operation of the programs or give direct commands to the PIC.

GENERAL FORMAT FOR INSTRUCTIONS



OPCODE FIELD DESCRIPTIONS

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1) The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W, d = 1: store result in file register f. Default is d = 1
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

The STATUS Register: The STATUS register holds the bits that are used to carry extra information about the result of the instruction most recently executed, for example whether the result is zero or a carry/borrow operation has occurred.

Part 2: (Practical)

1) The "**EQU**" directive

The equate directive is used to assign labels to numeric values. They are used to **DEFINE CONSTANTS** or to **ASSIGN NAMES TO MEMORY ADDRESSES OR INDIVIDUAL BITS IN A REGISTER** and then use the name instead of the numeric address.

Example1: - In this part we will learn the **equ** directive and how we can use it in our programs to make it meaningful.

A)

[ ] Open a new Text document and write on it the following:

```
Tmr0  equ 01
movlw 06
movwf Tmr0
nop
nop
end
```

[ ] Save the text file you created with Lab3_P1.1a.asm
[ ] Create a new project, Name the project Lab3Part1.1a
[ ] Compile the project and note if there are any errors, did the compiler recognize Tmr0, how?

B)

```
Num1 equ 20       ;GPR @ location 20
Num2 equ 40       ;GPR @ location 40
Movlw 5     ; move the constant 5 to the working register
Movwf  Num1; copy the value 5 from working register to Num1 (address 20)
Movlw 2     ; move the constant 2 to the working register
Movwf Num2  ; copy the value 2 from working register to Num2 (address
40)
Nop
End
```

2)The "**include**" directive

Suppose we are to write a huge program that uses all registers. It will be a tiresome task to define all Special Function Registers (SFR) and bit names using "equate" statements. Therefore, we use the include directive.

- The **include** directive calls a file which has all the equate statements defined for you and ready to use, its syntax is

#include "PXXXXXXX.inc" where XXXXXX is the PIC part number

Older version of include without #, still supported.

Example: #include "P16F84A.inc"

Example2: - In this part we will learn the **include** directive and how can we use it in our programs to make them meaningful.

[ ] Open a new Text document and write on it the following:

```
# include "p16f84.inc"
  movlw 0x3f
  movwf STATUS
  movlw 12
  movwf OPTION_REG
  nop
  nop
  end
```

[ ] Save the text file you created in folder Lab3 with name Lab3_P1.2.asm.
[ ] Create a new project , Name the project Lab3Part1.2, and add the file Lab3_P1.2.asm
[ ] Compile the project and note if there are any errors, did the compiler recognize *OPTION_REG*, how?
[ ] View the *OPTION_REG* register from the Watch window, step through the code and notice if the value of *OPTION_REG* is being set to 12, if not why?

3- " MOVF" instruction.

The PIC-Micro instruction set has several instructions that are used to move data, literals as shown below:

MOVF	f, d	Move f
MOVWF	f	Move W to f
MOVLW	k	Move literal to W

MOVF:- This instruction moves the data stored in the register to either the working register or to the register itself. **This is the only data movement instruction that affects the STATUS register.**

MOVF	**Move f**
Syntax:	[*label*] MOVF f,d
Operands:	$0 \le f \le 127$ $d \in [0,1]$
Operation:	(f) → (destination)
Status Affected:	Z
Description:	The contents of register f are moved to a destination dependant upon the status of d. If d = 0, destination is W register. If d = 1, the destination is file register f itself. d = 1 is useful to test a file register, since status flag Z is affected.

[ ] Create a new project and name it Lab3Part1.3

[ ] Write the following code, and save it as Lab3_P1.3.asm

```
Location equ 0C
MOVLW 0
MOVWF Location
MOVF Location, 0
END
```

[ ] Build the project

[ ] Open the watch window, add the **WREG**, **STATUS** registers, and add the symbol Location.

[ ] Run and simulate the project. Did the value of Status Register change? If yes, why, and what is the affected bit? If not, explain.

[ ] What is the equivalent machine code of instruction MOVF TMR0, 0.

Exersice1: -

Write a code that initiates the memory location 0x0D with the value 0 and moves its contents to itself. Use the equ directive to give the GPR (General Purpose Register) a name and use it in your program. Watch the **STATUS** register, did its value change? **Explain**.

Exersice2: -

Write a code to copy the contents of location 0x0E to the location 0x1F

Exersice3: -

Write a code to exchange the contents of location 0x33 with location 0x11

4- Arithmetic instruction.

ADDWF	f, d	Add W and f
ADDLW	k	Add literal and W
SUBWF	f, d	Subtract W from f
SUBLW	k	Subtract W from literal
INCF	f, d	Increment f
DECF	f, d	Decrement f

Example1: -

```
 include "p16f84a.inc"

cblock 0x30
     Num1
     Num2
     Result1
     Result2
endc

               org 0x00
Main
     Movlw 9
     Movwf Num1
     Movlw 8
     Movwf Num2
     movf  Num1, W
     addwf Num2, W
     Movwf Result1
     Movlw 1
     Movwf Num1
     Movlw D’255’
     Movwf Num2
     movf  Num1, W
     addwf Num2, W
     Movwf Result2
     nop
     end
```

Example2: -

```
 include "p16f84a.inc"

cblock 0x30
     Num1
     Num2
     Result1
     Result2
endc

               org 0x00
Main
     Movlw 4
     Movwf Num1
     Movlw 8
     Movwf Num2
     movf  Num1, W
     subwf Num2, W
     Movwf Result1
     Movlw 9
     Movwf Num1
     Movlw 7
     Movwf Num2
     movf  Num1, W
     subwf Num2, W
     Movwf Result2
     nop
     end
```

5- Logical instruction.

ANDWF	f, d	AND W with f
ANDLW	k	AND literal with W
IORWF	f, d	Inclusive OR W with f
IORLW	k	Inclusive OR literal with W
XORWF	f, d	Exclusive OR W with f
XORLW	k	Exclusive OR literal with W
COMF	f, d	Complement f

6- Branch instruction.

DECFSZ	f, d	Decrement f, Skip if 0
INCFSZ	f, d	Increment f, Skip if 0
BTFSC	f, b	Bit Test f, Skip if Clear
BTFSS	f, b	Bit Test f, Skip if Set

Example3: -

```
        include "p16F84A.inc"
cblock  0x25
        testNum
        Result
endc
        org     0x00
Main
        movf    testNum, W
        sublw   D’10’       ;10d - testNum
        btfss   STATUS, C
        goto    Greater         ;C = 0, that's B = 1, then testNum > 10
        goto    Smaller         ;C = 1, that's B = 0, then testNum < 10
Greater
        movlw   A'G'
        movwf   Result
                goto  Finish
Smaller
        movlw   A'S'
        movwf   Result
Finish
        nop
        end
```

Exercise1: -

Write a code to check if the MSB (High Nibble) is greater than LSB (Low Nibble) or not for a certain number in location 0x0E, if the result is true set the value of variable RESULT to "G", else set the value of variable RESULT to "S".

Examples: -

Number1:0x49	*Result =S*
Number2: D'100'=0x64	*Result =G*
Number3: B'00110011'	*Result =S*

Discussion and Follow-up

1.How many bits are in a nibble? How many nibbles are in a byte?

2.Refer to Chapter 2 of the data sheet. What is the address of the PORTB register?

3.Are the names of the SFR (Special Function Register) registers used in the program case sensitive or not? Check this by changing the name of one SFR register to small letters and then compile the project.

4.Write a program that implements the following equation:

R = 3*V1+V2+2*V3

Where the addresses and values of the variables are as follows:

Name	Address	Value
V1	0x20	B'00010111'
V2	0x21	D'39'
V3	0x22	a'A'
Result	0x26	?

5.The following two codes logically perform the same function; however, the second code gives different results, why?

```
#include "p16f84a.inc"
clrf PORTB
movlw 45
movwf PORTB
swapf PORTB, f
nop
end
```

```
#include "p16f84a.inc"
clrf STATUS
movlw 45
movwf STATUS
swapf STATUS, f
nop
end
```

6.Write a simple program that implements the following pseudocode Initialize location 0x30 (LocA) with the decimal value of 15 Initialize location 0x40 (LocB) with the value of 0

LocA = LocA – LocB
LocB = LocB + 1
Repeat until LocA = 0

Include a screenshot of your work showing the watch window and displaying the final values of LocA and LocB.

Objectives

1. To be familiar with assembly language programming and the Microchip PIC 16 series instruction set.
2. To see an application of macros and methods of utilizing them.
3. To use the debugging facility of the MPLAB IDE to fix program bugs.

Pre-lab Preparation:

1. Read chapter 7 of the PIC16F84 data sheet.
2. Review the Status Register- Section 2.2.1 in the book
3. Study the assembly code listings of accompanying programs. (Very important).

Example1: - Counting the Number of Ones in a Register's **Lower Nibble** Introducing simple conditional statements.

```
include "p16f84a.inc"
cblock 0x20
testNum          ;GPR @ location 20
tempNum          ;GPR @ location 21
endc
cblock 0x30
numOfOnes        ;GPR @ location 30
endc
org 0x00
clrf      numOfOnes     ;Initially number of ones is 0
movf      testNum, W    ;Since we only need to test the number of ones in the lower
nibble,
                        ; we mask them by 0x0F (preserve lower nibble and discard higher
                        ;nibble)

andlw     0x0F   ;in case a user enters a number in the upper digit. Save masked result
movwf     tempNum       ;in tempNum
rrf tempNum, F          ;rotate tempNum to the right through carry, that is the least
                        ; Significant bit of tempNum (bit0) goes into the C flag of the
                        ; STATUS register, while the old value of C flag goes into bit 7
                        ; of tempNum
btfsc     STATUS, C     ;tests the C flag, if it has the value of 1, increment number of
                        ;ones and
incf      numOfOnes, F  ;proceed, else proceed without incrementing
rrf tempNum, F
btfsc     STATUS, C           ;Same as above
incf      numOfOnes, F
rrf tempNum, F
```

```
    btfsc     STATUS, C
    incf      numOfOnes, F
    rrftempNum, F
    btfsc     STATUS, C
    incf      numOfOnes, F
    nop
    end
```

As you can see in the above program, we did not write instructions to load **testNum** with an initial value to test; this code is general and can take any input. So, **how do you test this program with general input?**

After building your project, adding variables to the watch window, and selecting MPLAB SIM simulation tool, simply **double click on testNum** in the **watch** window and fill in the value you want. Then Run the program.

Change the value of **testNum** and re-run the program again, check if **numOfOnes** hold the correct value.

Coding for efficiency: The repetition structures

You have observed in the code above that instructions from **14 to 25** are simply the same instructions repeated over and over four times for each bit tested.

Now we will introduce the repetition structures, similar in function to the “for” and “while” loops you have learnt in high level languages.

```
include "p16f84a.inc"
      cblock 0x20
            testNum
            tempNum
      endc
      cblock 0x30
      numOfOnes
      counter       ;since repetition structures require a counter, one is declared
      endc
      org 0x00
      clrf  numOfOnes
      movlw 0x04        ;counter is initialized by 4, the number of the bits to be

                        ;tested
      movwf counter
      movf  testNum, W
      andlw 0x0F
      movwf tempNum
Again
      rrf   tempNum, F
      btfsc STATUS, C
      incf  numOfOnes, F
      decfsz   counter, F     ; after each test the counter is decremented,
      goto      Again         ; if the counter reaches 0, it will skip to
                              ; “nop” and program ends
      nop                     ; if the counter is > 0, it will repeat “goto Again”
      end
```

Modular Programming

Modular programming is a software design technique in which the software is divided into several separate parts, where each part accomplishes a certain independent function. This "Divide and Conquer" approach allows for easier program development, debugging as well as easier future maintenance and upgrade.

Modular programming is like writing C++ or Java **functions**, where you can use the function many times only differing in the parameters. Two structures which are like functions are **Macros** and **Subroutines which are used to implement modular programming.**

1)Subroutines

Subroutines are the closest equivalent to functions

* Subroutines start with a **Label (**subroutine_Name**)** giving them a name and end with the instruction **return.**

*Subroutines can be written anywhere in the program after the **org** and before the **end** directives.

*Subroutines are used in the following way: **Call** subroutine_Name.

*Subroutines are stored once in the program memory, each time they are used, they are executed from that location.

*Subroutines alter the flow of the program; thus they affect the **stack**.

2)Macros

Macros are declared in the following way (like the declaration of cblocks)

```
Macro       macro Name
Instruction 1
Instruction 2
.
.
Instruction n
endm
```

*Macros should be declared **before** writing the code instructions. **It is not recommended to declare macros in the middle of your program.**

*Macros are used by only writing their name: macro Name

*Each time you use a macro, it will be replaced by its body. Therefore, the program will execute sequentially, the flow of the program will not change. **The Stack is not affected**.

Example2: -

```
; -----------------------------------------------------------
; General Purpose RAM Assignments
; -----------------------------------------------------------
cblock            0x17
InputM2
Input_TempM2
InputM4
ResultM2
Result_TempM2
ResultM4
Endc

; -----------------------------------------------------------
; Macro Definitions
; -----------------------------------------------------------
Multiply2   macro
Movf        Input_TempM2,w
Addwf       Input_TempM2,w
movwf       Result_TempM2
Endm
; -----------------------------------------------------------
; Vector definition
; -----------------------------------------------------------
                  org 0x000
                  nop
                  goto Main

INT_Routine org 0x004
            goto INT_Routine
; -----------------------------------------------------------
; The main Program
; -----------------------------------------------------------
Main        Movlw       d'15'
Movwf       InputM2
Movwf       InputM4
Movwf       Input_TempM2
Multiply2
movwf       ResultM2
Movwf       Input_TempM2
Call        Multiply4
Goto        finish
; -----------------------------------------------------------
; Sub Routine Definitions
; -----------------------------------------------------------
Multiply4
Multiply2
Movf        Result_TempM2,w
Movwf       ResultM4
Return
finish
             nop
end
```

General Multiply function: -Result = Input1 * Input2

```
General_Multiply
          Clrf      Result
Again     movf      Input2,w
          Addwf     Result,f
          Decfsz         Input1, f
          Goto      Again
Finish
     Return
```

Exercise1:

Modify **Example2** to multiply by 3 Macro(Multiply3) and multiply by 9(Multiply9) function?

Exercise2:

Write a test code for General Multiplication function for two following cases: -

1- Input1=d'9', Input2=d'7'

2-Input1=0, Input2=15?

Discussion and Follow-up

Write a General Divide function $\left[\frac{\mathbf{Input2}}{\mathbf{Input1}}\right]$ that uses multiple subtract to perform following equation: -

$$\mathbf{Input2 = (Input1) * Counter + Remains}$$

Where; $0 \leq$ **Remains**< Input1

Counter is integer number of $\frac{\mathbf{Input2}}{\mathbf{Input1}}$

Example:- Input1=3,Input2=17;we can subtract 3 five times from 17 without borrow.

$\frac{17}{3} = 3 * \mathbf{5} + \mathbf{2}; where\ \boldsymbol{counter} = \mathbf{5}\ and\ \boldsymbol{Remains} = \mathbf{2}$

University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp. 4: Implementing Instructions (II)

Objectives

1. To be familiar with assembly language programming and the Microchip PIC 16 series instruction set.
2. To see an application of macros and methods of utilizing them.
3. To use the debugging facility of the MPLAB IDE to fix program bugs.

Pre-lab Preparation:

1. Read chapter 7 of the PIC16F84 data sheet.
2. Review the Chapter 3 in the book
3. Study the assembly code listings of accompanying programs. (Very important).

Procedure:

This lab experiment is composed of three parts. **The first part** discuss how we can handle with EEPROM. **The second part** introduces the theory behind BCD numbers used in this experiment. **The third part** is an interactive one where you will be introduced to some PIC codes and to investigate their parameters and usage. The experiment involves using MPLAB and implementing codes to learn key issues.

Part 1:

1.1) Writing to the EEPROM Data Memory code

```
Movlw     0xFF
Movwf     EEADR
Movlw      A'H'
Bcf
STATUS,RP0
Movwf      EEDATA
Incf       EEADR,f
Bsf        STATUS, RP0
Bcf        INTCON, GIE
Bsf        EECON1, WREN
Movlw       0x55
Movwf      EECON2
Movlw      0xAA
Movwf      EECON2
Bsf        EECON1,WR
Bsf        INTCON, GIE
Test
Btfsc      EECON1,WR
Goto       Test
```

```
Movlw      A'I'
Bcf
STATUS,RP0
Movwf      EEDATA
Incf       EEADR,f
Bsf        STATUS, RP0
Bcf        INTCON, GIE
Bsf        EECON1, WREN
Movlw       0x55
Movwf      EECON2
Movlw      0xAA
Movwf      EECON2
Bsf        EECON1,WR
Bsf        INTCON, GIE
Test1
Btfsc      EECON1,WR
Goto       Test1
```

Exercise1: -

Modify the code using modular programing technique to write your name in EEPROM data memory. In addition, initial All GPR with first letter of your name.

1.2) Test code to Reading from the EEPROM Data Memory code

```
Bcf         STATUS, RP0
Clrf        EEADR
Bsf         STATUS, RP0
Bsf         EECON1, RD
Bcf         STATUS, RP0
Clrf        Counter
Bsf         STATUS, RP0
Clrf        TRISB
Bcf         STATUS, RP0
Movlw       A'H'
Subwf       EEDATA,w
Btfsc       STATUS,Z
Goto        Finish
Incf        Counter,f
Movlw       A'M'
Subwf       EEDATA,w
Btfsc       STATUS,Z
Finish      Incf  Counter,f
Call        Look_Up
Movwf       PORTB
Loop
Goto        Loop
```

```
Look_Up
Movf        Counter,w
Addwf       PCL,f
Retlw        B'11000000'
Retlw        B'11111001'
Retlw        B'10100100'
Retlw        B'10110000'
Retlw        B'10011001'
Retlw        B'10010010'
```

Part 2:

Binary Coded Decimal (**BCD**) is an encoding scheme for decimal numbers in which each digit is represented by its own binary sequence. Its main advantage is that it allows easy conversion to decimal digits for printing or display and faster decimal calculations. Its drawbacks are the increased complexity of circuits needed to implement mathematical operations and a relatively inefficient encoding (6 wasted patterns per digit).

In BCD, a digit usually represented by four bits which, in general; represent the values 0 - 9. To BCD-encode a decimal number using the common encoding, each decimal digit is stored in a four-bit nibble.

Decimal:	0	1	2	3	4	5	6	7	8	9
BCD:	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

Since most computers store data in eight-bit (byte), there are two common ways of storing four-bit BCD digits in those bytes:

1. Unpacked BCD: where each digit is stored in one byte and the other four bits are then set to all zeros.
2. Packed BCD: where two digits are stored in each byte.

Part 3:

[ ] Create a directory on the PC in drive D inside folder lab4 in which to store all of your work. Use YOUR name as the name of the directory.

[ ] Copy the Example3.asm file to your created directory from the lab5directory.

[ ] Create a new project with Example3.asm. Build the project.

[ ] Select the Debugger → MPLAB SIM→ Reset, and then choose the Processor reset menu item or press F6. The software should highlight (with a green arrow) the "nop" instruction at address 0 just preceding the "goto Main" instruction in the code. In this step, you have told the simulator to start behaving as if the microcontroller has just been given power. So, it is now ready to start execution of your program. It is now ready and waiting at the reset vector for your next command. Note the program is not running yet.

[ ] Select the View → Hardware Stack menu item. This allows you to view the PIC microcontroller hardware stack, which stores return addresses after each interrupt and function call. Note that the stack is empty since nothing is happening.

[ ] Select the View → Watch Window. From the Symbols drop down list, choose BCD1. Click the Add Symbol button. Repeat the same procedure but for BCD2, BCD_Result. You should now note that your new Watch Window has these three register names listed, along with their addresses and contents.

[ ] Select the Window → Tile Horizontal menu item. Close the Output window if it is still open. Select Window again → Tile Horizontal. Make sure that the Editor, Hardware Stack and Watch windows are observable clearly.

[ ] Select the Debugger → Animate menu item. Observe the code running one instruction at a time, and the stack updating as routines are called. The status bar at the bottom of MPLAB shows the program counter (pc), which contains the address of the instruction that is to be executed next. The status bar also shows the value of the W register, the processor (PIC16F84A), the Status register flag settings, and a couple of other important items.

[ ] Select the Debugger → Halt menu item (F5).

You have just seen one aspect of the simulator. Using the animate function, you can easily simulate the whole program when it is running and watch how the registers are changing. You can also observe and see whether the program is executing correctly, calling the functions correctly. You can also see if the hardware stack is overflowing or not. Therefore, you can generally get an idea if the ideas and logic you have implemented in the software are working correctly as you expect or not.

Exercise2: -

A certain assembly application uses 1 digit packed BCD numbers to represent temperature. Before processing the temperature, it should be checked against the valid range of 14 – 87 Cº. Write a procedure that implements this checking functionality. The procedure should load the memory location Result in location 0x22 with the following:

Result=00 if the temperature falls in range, Result=FF if the temperature falls out of range

Discussion and Follow-up

Important Note: You must use comments to clarify your code. Use :equ" and include directives to specify the SFRs and GPRs. Utilize Macros, Subroutines and "CBlocks" in your code.

An **unpacked** 3-digit BCD number is stored in memory starting at the location 0x38 (Least Significant Digit "LSD"). You are required to convert the unpacked 3-digit BCD number into **binary format** and store the value in location 0x25. ***Use multiply by the shifting method.***

-Assume that the number is already in the range (0 – 255).

Hint: 105 = 5 + 0*10 + 1 *10^2 and the value stored in 0x25 is 01101001

```
;Program 5.1 unpacked BCD number.
;*********************************************************
;* FUNCTION: Test a BCD returning a boolean result
;* ENTRY : BCD in F20h *
;* EXIT : Result=00 if valid BCD number , Result=FF if invalid BCD number
;* *********************************************************

#include "P16F84A.inc"

BCD    equ 20h  ; The BCD number is in File 20h
Result equ 21h  ; The result is in File 21h
; ---------------------------------------------------------
BCD_Test
                    clrf Result         ; Clear Result file
                    movf BCD,W          ;
                    iorlw 0xF0   ; Mask the last four digits of the BCD File
                    addlw 6             ; Add six
                    btfsc STATUS,C  ; Needed IF produced a carry
                    comf Result,F       ; Make Result FF (Invalid BCD Number)
                    nop
                    nop

                    END
*****************************************************************************

;Program 5.2 packed BCD number.
;*********************************************************
;* FUNCTION: Test packed BCD returning a boolean Final_Result
;* ENTRY : BCD in F20h *
;* EXIT : Final_Result=00 if Two digits valid Packed  BCD number
;*        Final_Result=0F if invalid BCD First digit number and valid BCD Second
;digit number
;*        Final_Result=F0 if valid BCD First digit number and invalid BCD Second
;digit number
;*        Final_Result=FF if Two digits invalid Packed  BCD number
;* *********************************************************

#include "P16F84A.inc"

BCD                  equ 20h  ; The BCD number is in File 20h
Result               equ 21h  ; The result is in File 21h
Final_Result  equ 22h  ; The Final Packed BCD number result is in File 22h
; ---------------------------------------------------------
BCD_Test     macro
                    clrf Result         ; Clear Result file
                    movf BCD,W          ;
                    iorlw 0xF0   ; Mask the last four digits of the BCD File
                    addlw 6             ; Add six
                    btfsc STATUS,C  ; Needed IF produced a carry
                    comf Result,F; Make Result FF (Invalid BCD Number)
                    nop
                    nop
                    endm
; ---------------------------------------------------------

       BCD_Test                ;Test the First Digit
       movf Result,W
       andlw 0x0F
       movwf Final_Result        ;Save the First digit result
       swapf BCD,F
       BCD_Test                        ;Test the Second Digit
```

```
       swapf BCD,F
       movf Result,W
       andlw 0xF0
       addwf Final_Result,F      ;Save the Second digit result
       nop
       nop

       END
*******************************************************************************
```

;Program 5.3 packed BCD number Subtraction and Sddition.

```
;*************************************************************
;* FUNCTION: Test Two packed BCD returning a boolean Final_Result if the two BCD
;numbers are valid numbers
;* then Subtract the first digits and add the second digits else ends
;* ENTRY : BCD1         in  F30h
;*            : BCD2         in  F40h
;*            : BCD_Result   in  F45h
;* EXIT  : if two Valid BCD numbers BCD_Result   in  F50h
;* *************************************************************
#include "P16F84A.inc"
; -------------------------------------------------------
; Local equates for your own symbolic designators
; -------------------------------------------------------
BCD                  equ 20h        ; The BCD number is in File 20h
Result       equ 21h         ; The result is in File 21h
Final_Result    equ 22h   ; The Final Packed BCD number result is in File 22h

; -------------------------------------------------------
; General Purpose RAM Assignments
; -------------------------------------------------------
                   cblock 0x30
                          BCD1            ; The First BCD number is in File 30h
                          Final_Result1
                   endc

                   cblock 0x40
                          BCD2            ; The Second BCD number is in File 40h
                          Final_Result2
                   endc

                   cblock 0x45
               BCD_Result      ; The Final BCD number result is in File 50h
                   endc
; -------------------------------------------------------
; Macro Definitions
; -------------------------------------------------------
BCD_Test     macro
                   clrf Result        ; Clear Result file
                   movf BCD,W         ;
                   iorlw 0xF0   ; Mask the last four digits of the BCD File
                   addlw 6            ; Add six
                   btfsc STATUS,C  ; Needed IF produced a carry
                   comf Result,F    ; Make Result FF (Invalid BCD Number)
                   nop
                   nop
                   endm
```

```
; ---------------------------------------------------------
; Vector definition
; ---------------------------------------------------------
                  org 0x000
                  nop
                  goto Main

INT_Routine  org 0x004
                  goto INT_Routine
; ---------------------------------------------------------
; The main Program
; ---------------------------------------------------------
Main         movf  BCD1,W
             movwf BCD
             call  Packed_Test         ; Test the First Number
             movf  Final_Result,W
             movwf Final_Result1

             movf Final_Result1,F
             btfss STATUS,Z      ; Test if valid First BCD continue else ends
             goto Finish

             movf  BCD2,W
             movwf BCD
             call  Packed_Test         ; Test the Second Number
             movf  Final_Result,W
             movwf Final_Result2

             movf Final_Result2,F
             btfss STATUS,Z      ; Test if valid Second BCD continue else ends
             goto Finish

             movf BCD2,W
             xorlw 0x0F
             addwf BCD1,W              ;BCD_ResultL = BCD1L - BCD2L
             addlw 0xf1                ;BCD_ResultH = BCD1H + BCD2H
             movwf BCD_Result

       goto Finish

; ---------------------------------------------------------
; Sub Routine Definitions
; ---------------------------------------------------------
Packed_Test
       BCD_Test               ;Test the First Digit
       movf Result,W
       andlw 0x0F
       movwf Final_Result        ;Save the First digit result
       swapf BCD,F
       BCD_Test                        ;Test the Second Digit
       swapf BCD,F
       movf Result,W
       andlw 0xF0
       addwf Final_Result,F      ;Save the Second digit result
           nop
       nop
         return
; ---------------------------------------------------------
; The main Program end
; ---------------------------------------------------------
Finish
             END
```

Objectives

1. To become familiar with the process of writing an assembly language program for the PIC.
2. To demonstrate different methods of handling the I/O process.
3. To demonstrate different methods of handling interrupts.
4. To use the debugging facility of the MPLAB IDE to test programs.

Pre-lab Preparation:

1. Read chapter 7 of the PIC16F84 data sheet.
2. Read Appendix 1 carefully.
3. Write the required assembly language programs **carefully** with proper documentation.

Procedure:

Write an assembly code, which operates a PIC16F84A that control a bottle labeling and packing machine.

Machine Sequence

1. The bottles pass through a conveyor belt, when the photocell sensor detects a bottle (External Interrupt), the label actuator (Solenoid) starts working and stops after 1.2 Sec.
2. 7-Segments show the numbers of labeled bottles, when the number of labeled bottles passing through the photocell sensor reaches nine the conveyor belt motor stops.
3. Nine bottles are packed into a cartoon. The cycle starts again when packing process is finished (Packing process needs to 3.6 Sec. to finished by Electro-Mechanical mechanism).
4. When the number of packed bottles reaches nine (Show the numbers of packed bottles on another digit of 7-Segments) the conveyor belt motor stops until the START pushbutton is pressed to start the cycle again.

General Guidelines for Writing your Programs

a. Always start with visualizing in your mind the process that should take place in the hardware.
b. Determine the inputs and outputs for the hardware.
c. Assign PIC ports to the hardware I/O.
d. Remember and always keep in mind the data flow cycle.
e. Never start writing code immediately in MPLAB, it wastes time and will very rarely give you what you want.
f. Start always with a flowchart on paper keeping in mind the above points. Try at first a general flowchart and then attempt to expand the flowchart into more detail to reflect the requirements of the program.
g. If you have done the above properly you will find that the flowchart will divide the program naturally into parts. You should now be able to write the code for each part.
h. Writing assembly takes time and needs patience, so be patient and careful with your code.
i. Write your comments to the code immediately with the code.
j. Study the programs that you have used in other experiments for writing style, hints and ideas.
k. Use the simulator in MPLAB to simulate your program.
l. You should demonstrate your working programs on the board to the lab supervisor.

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

; This program control a bottle labeling and packing machine.
; Photocell sensor is connected into RB0
; 7-Segments is connected to PORTB (We connect RB1 to a, RB2 to b …….And
;RB7 to g)
; Digits selection of bottles number 7-Segments is connected to RA0
; Digits selection of cartoon number 7-Segments is connected to RA1
; Conveyor belt motor is connected to RA2 (Connect to LED1 on board)
; Label actuator is connected to RA3 (Connect to LED2 on board)
; START pushbutton is connected to RA4
; The program uses a PIC16F84A running at crystal oscillator of frequency
;4MHz.
;*******************************************************
 Include "p16f84A.inc"
;*******************************************************
; Macro definitions

push  macro

      movwf       WTemp       ; WTemp must be reserved in all banks
      swapf       STATUS,W    ; store in W without affecting status bits
      banksel     StatusTemp        ; select StatusTemp bank
      movwf       StatusTemp        ; save STATUS
      endm

pop   macro

      banksel     StatusTemp        ; point to StatusTemp bank
      swapf       StatusTemp,W      ; unswap STATUS nibbles into W
      movwf       STATUS            ; restore STATUS
      swapf       WTemp,F           ; unswap W nibbles
      swapf       WTemp,W           ; restore W without affecting STATUS
      endm
;*******************************************************
; User-defined variables
      cblock            0x0C  ; bank 0 assignments
      WTemp
      StatusTemp
      ;……              Add all variables here.
      endc
;*******************************************************
; Start of executable code
      org   0x00        ;Reset vector
      nop
      goto        Main
      org   0x04              ;
      goto        INT_SVC
;;;;;; Main program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Main
 call  Initial          ;Initialize everything
MainLoop
 call  Bottle_Number  ;Check if the number of bottles reaches to nine
 call  Caroon_Number  ;Check if the number of packing bottles reaches to9.
      goto  MainLoop  ;Do it again
```

```
;;;;;;;     Initial subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine performs all initializations of variables and registers.
Initial
      Return
;;;;;;; Bottle_Number subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine Test if the number of bottles reaches to nine.
Bottle_Number
      Return
;*********************************************************
;;;;;;; Caroon_Number subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine Test if the number of packing bottles reaches to nine.
Caroon_Number
      Return
;*********************************************************
;;;;;;; Delay subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine to get a delay with 1.2 Sec.
Delay
      Return
;*********************************************************
; ;;;;;;;  Bottle_Labeling subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine start labeling and counts the number of bottles
Bottle_Labeling
      bcf       INTCON, INTF  ;Clear the External interrupt flag
;     write the code here
      goto        POLL        ;Check for another interrupt
;*********************************************************
INT_SVC
push
POLL
      btfsc       INTCON, INTF      ; Check for an External Interrupt
      goto        Bottle_Labeling
;     btfsc ...                           ; Check for another interrupt
;     call  ...
;     btfsc ...                           ; Check for another interrupt
;     call  ...
      pop
      retfie

;*********************************************************
      End
```

University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp. 6: Frequency Measurement

Objectives

1. To become familiar with hardware timers in 16F877A PIC.
2. To demonstrate the use of internal interrupts linked with the timer1 module of the 16F877A.
3. To use the debugging facility of the MPLAB IDE to fix program bugs.

Pre-lab Preparation:

1. Review the sections in the book regarding methods of generating time delays.
2. Review the sections in the book dealing with the timer/counter peripheral (Chapter 5).
3. Review the instruction set of the PIC 16.
4. **Study the assembly code listings of accompanying programs. (Very important).1**

Procedure:

The principle of frequency measurement

Frequency measurement is a very important application of both counting and timing. Fundamentally, frequency measurement is a measure of how many times something happens within a certain known period, as illustrated in Figure 1. The use can be as diverse as how many counts are received per minute in a Geiger counter, how many cycles per second (i.e. Hertz) in an electronic or acoustic measurement, or how many wheel revolutions there are per unit of time in a speed measurement. Both a counter and a timer are needed, the timer to measure the reference period of time and the counter to count the number of events within that time [(1)].

In this experiment we use Timer0 as counter and Timer1 for periodic time.

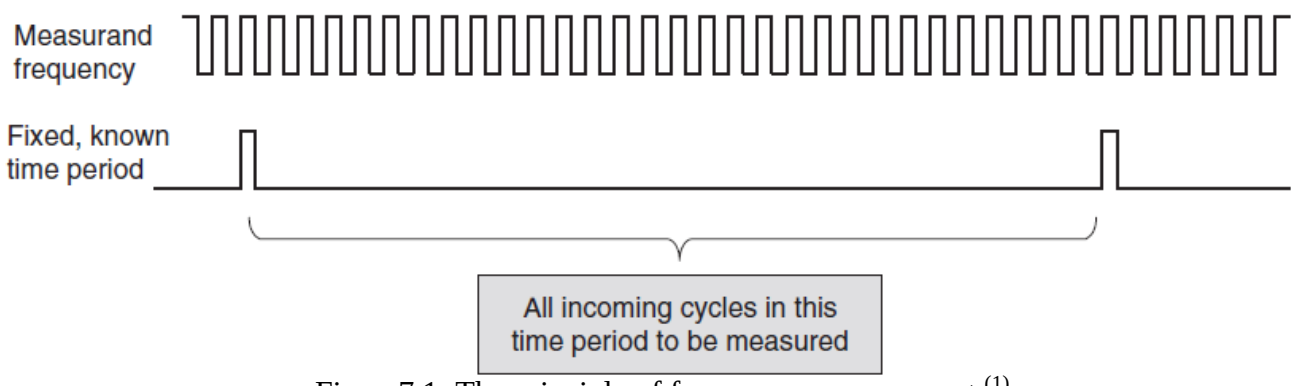


Figure7.1: The principle of frequency measurement [(1)]

(1)Designing Embedded Systems with PIC Microcontrollers: Principles and applications

Frequency Range	TMR1H	TMR1L	7-Segments
0-511	0000 0001	xxxx xxxx	0
512-1023	0000 001x	xxxx xxxx	1
1024-2047	0000 01xx	xxxx xxxx	2
2048-4095	0000 1xxx	xxxx xxxx	3
4096-8191	0001 xxxx	xxxx xxxx	4
8192-16383	001x xxxx	xxxx xxxx	5
16384-32767	01xx xxxx	xxxx xxxx	6
>32767	1xxx xxxx	xxxx xxxx	7

Discussion and Follow-up:

1. Suppose the oscillator frequency is 8 MHz.
   -What is the longest possible time between Timer0 interrupts? In addition, how would TMR0 and OPTION_REG be initialized in this case?
   ……

   ……

   -What is the shortest possible time between Timer0 interrupts? In addition, how would TMR0 and OPTION_REG be initialized in this case?
   ……

   ……
2. In the PUSH macro, why did we have to use the **swapf** instruction to save the status register?

   ……

   ……
3. In the **cblock** definitions at the beginning of the program, why did we need to reserve four different locations to save the W register? Was it necessary? Why or why not?
   ……

   ……
4. In the pop macro, WTemp is restored to W using two swapf instructions. Why are two swapf instructions used instead of the simpler movf instruction?
   ……

   ……
5. How many instructions affect the zero bit of the STATUS register?

   ……
6. What does the Master Clear input do when asserted? What pin number on the PIC is the Master Clear input? Is it active high or active low? What voltage should be connected to the Master Clear input under normal operation?
   ……

7. Why it is important to save the W and STATUS registers at the beginning of an interrupt, and restore them at the end of the interrupt?

   ……

   ……

   ……

Programming the Chip

[ ] Copy the *FileName.asm* to your directory.

[ ] Start the MPLAB software on your PC and build the project.

[ ] Select the Configure menu → Select device and make sure the following is selected:

- ❖ Device is PIC16F877A

[ ] Select the Configure menu → Configuration B**its** and make sure the following are selected:

- ❖ Oscillator is XT
- ❖ Watchdog Timer, Power Up Timer, Brown Out Detect can all be off/disable for this tutorial
- ❖ Low Voltage Program should be disabled.
- ❖ Code Protect Data EE, and Code Protect - turn all off
- ❖ Flash Memory Write 00000-FFFFFh.

[ ] Select Programmer → Select Your Programmer (**PICKit2** or MPLAB ICD2). You should see in the Output window a message regarding the availability of a new operating system for the programmer. Ignore this message. If you get any other error messages, ask for the supervisor's help.

[ ] Place the chip in the Zero Insertion Force (ZIF) socket carefully, making sure it is oriented correctly such that pin 1 is on the left upper side of the ZIF. The chip should sit without pushing and very easily. Pull the handle of the ZIF socket to the upright position to hold the chip in place firmly.

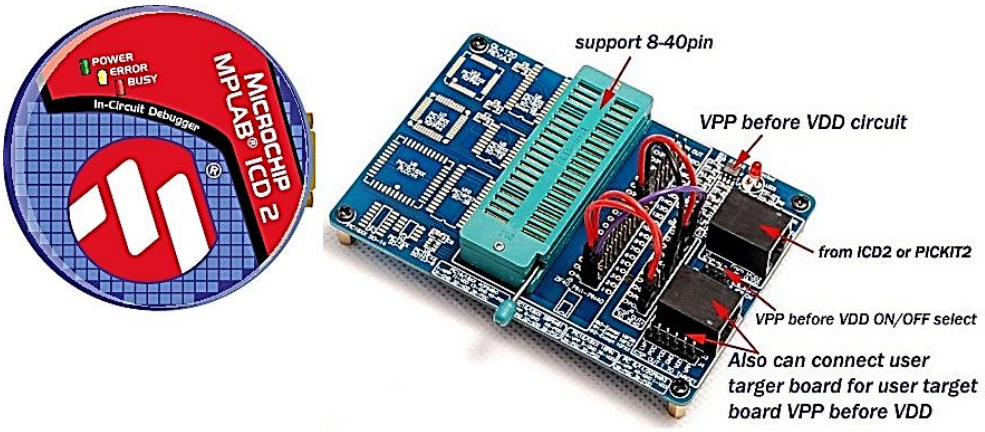
[ ] From the Programmer menu, select Erase Flash Device to make sure that the PIC you are using is blank.

[ ] Select Programmer → Program. This will start the programming process of the chip. Watch the bottom of the MPLAB IDE program for progress information.

[ ] If the programming was successful, then you should see a pass message near the MPLAB IDE icons. If there are any problems, contact the supervisor for help.

[ ] Remove the chip from the ZIF socket. It is now ready to be tested on the development board.

PICKit2 Programmer



MPLAB ICD2 Programmer

```
;*******************************************************
; This program used to measure the frequency of  periodic signal
;(like shaft encoder signal) using Timer0 and Timer1
;The signal is connected to RC0 as a clock to Timer1(Timer1 as counter)
;Timer0 used to get 1 Sec of time to measure the frequency of signal
;7-Segments is connected to PORTB (We connect RB0 to a, RB1 to b …….And
;RB6 to g)
;to return the rang of frequency as a hexadecimal numbers
;The program uses a PIC16F877A running at crystal oscillator of frequency
;4MHz.
;*******************************************************
 include "p16f877A.inc"
;*******************************************************
; Macro definitions
push  macro

      movwf       WTemp       ; WTemp must be reserved in all banks
      swapf       STATUS,W    ; store in W without affecting status bits
      banksel     StatusTemp  ; select StatusTemp bank
      movwf       StatusTemp  ; save STATUS
      endm

pop   macro

      banksel     StatusTemp  ; point to StatusTemp bank
      swapf       StatusTemp,W; unswap STATUS nibbles into W
      movwf       STATUS            ; restore STATUS
      swapf       WTemp,F           ; unswap W nibbles
      swapf       WTemp,W           ; restore W without affecting STATUS
      endm
;*******************************************************
Zero        equ     B'00111111'     ; 7-Segment Code for Zero
One         equ     B'00000110'     ; 7-Segment Code for One
Two         equ     B'01011011'     ; 7-Segment Code for Two
Three       equ     B'01001111'     ; 7-Segment Code for Three
Four        equ     B'01100110'     ; 7-Segment Code for Four
Five        equ     B'01101101'     ; 7-Segment Code for Five
Six         equ     B'01111101'     ; 7-Segment Code for Six
Seven       equ     B'00000111'     ; 7-Segment Code for Seven
Eight       equ     B'01111111'     ; 7-Segment Code for Eight
Nine        equ     B'01101111'     ; 7-Segment Code for Nine
LetterA     equ     B'01110111'     ; 7-Segment Code for A
LetterB     equ     B'01111100'     ; 7-Segment Code for B
LetterC     equ     B'01011000'     ; 7-Segment Code for C
LetterD     equ     B'01011110'     ; 7-Segment Code for D
LetterE     equ     B'01111001'     ; 7-Segment Code for E
LetterF     equ     B'01110001'     ; 7-Segment Code for F
;*******************************************************
; User-defined variables
            cblock            0x20        ; bank 0 assignnments
                  WTemp       ; WTemp must be reserved in all banks
                  StatusTemp
                  Timer1Counts
                  TMR0_Counter
      endc
```

```
      cblock            0x0A0       ; bank 1 assignnments
                  WTemp1            ; bank 1 WTemp
      endc

      cblock            0x120       ; bank 2 assignnments
                  WTemp2            ; bank 2 WTemp
      endc

      cblock            0x1A0       ; bank 3 assignnments
                  WTemp3            ; bank 3 WTemp
      endc

;*******************************************************
; Start of executable code
      org   0x00        ;Reset vector
      nop
      goto        Main
      org   0x04
      goto        INT_SVC
      ;;;;;;; Main program ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
Main
      call  Initial                       ;Initialize everything
MainLoop
      call  Range_Test
      goto  MainLoop                ;Do it again
;;;;;;;     Initial subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine performs all initializations of variables and registers.
Initial
      banksel     TRISA             ;Select bank1
      clrf  TRISB                   ;set all bits of port B to output
      bsf         TRISC,RC0

      movlw 0x07
      movwf OPTION_REG        ;Prescaler is assigned to the Timer0 module
                              ;Prescaler TMR0 (1:256)
      bsf   INTCON,GIE        ;Enable Global Interrupt
      bsf   INTCON,PEIE       ;Enable peripheral interrupts
      bsf   INTCON,TMR0IE

      banksel     PORTA        ;Select bank0
      movlw D'70'       ; initialize TMR0 with 70 counts
      movwf TMR0        ; to get interrupt every 47.64ms

      movlw 0x02
      movwf T1CON       ;TMR1 Prescale (1:1),TMR1 Oscillator is shut-off
                        ;External clock from pin RC0/T1OSO/T1CKI (on the
                        ;rising edge)
      bsf   T1CON,TMR1ON      ;Enables Timer1

      movlw Zero
      movwf PORTB             ;initialize 7-Segments to Zero
      clrf  TMR0_Counter
      clrf  Timer1Counts
      Return
```

```
;;;;;;;     Range_Test subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine performs test the range of counts and show them on the
;7-Segments.
Range_Test

      btfsc Timer1Counts,7          ;Freq less than 32767Hz
      goto  seven
      btfsc Timer1Counts,6          ;Freq less than 16384Hz
      goto  six
      btfsc Timer1Counts,5          ;Freq less than 8192Hz
      goto  five
      btfsc Timer1Counts,4          ;Freq less than 4096Hz
      goto  four
      btfsc Timer1Counts,3          ;Freq less than 2048Hz
      goto  three
      btfsc Timer1Counts,2          ;Freq less than 1024Hz
      goto  two
      btfsc Timer1Counts,1          ;Freq less than 512Hz
      goto  one
      goto  zero

zero
      movlw Zero
      movwf PORTB
      goto  Finish
one
      movlw One
      movwf PORTB
      goto  Finish
two
      movlw Two
      movwf PORTB
      goto  Finish
three
      movlw Three
      movwf PORTB
      goto  Finish
four
      movlw Four
      movwf PORTB
      goto  Finish
five
      movlw Five
      movwf PORTB
      goto  Finish
six
      movlw Six
      movwf PORTB
      goto  Finish
seven
      movlw Seven
      movwf PORTB
Finish
      Return
```

```
;*********************************************************
; TIMER0 RE-Initialize and reset
T0
      movlw D'70'       ; initialize TMR0 with 70
      movwf TMR0        ; counts to get interrupt every 47.64ms

      incf  TMR0_Counter,F
      movf  TMR0_Counter,w
      sublw D'21'
      btfss STATUS,Z
      goto  Continue
      movf  TMR1H,w
      movwf Timer1Counts
      clrf  TMR1L
      clrf  TMR1H
      clrf  TMR0_Counter
Continue
      bcf         INTCON,TMR0IF
      goto  POLL                    ;Check for another interrupt
;*********************************************************
INT_SVC
push
POLL
      btfsc INTCON,TMR0IF                ; Check for an TMR0 Interrupt
      goto  T0

      pop
      retfie

;*********************************************************
      End
```

University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp. 7: Serial Communication

Objectives

1. To become familiar with the use of serial communications through the USART.
2. To demonstrate methods of remote control using serial communications.
3. To use the debugging facility of the MPLAB IDE to fix program bugs.

Pre-lab Preparation:

1. Review the sections in the book regarding the USART.
2. Read the PIC16F877A data sheet especially chapter 10.
3. Review the instruction set of the PIC 16F877A.
4. Read the assembly programs **carefully** and try to understand the operation and the settings used.

Procedure:

we are going to use the USART of the PIC to receive a character from the PC and return (send) next character to PC again. The communication is done using the RS232 protocol by utilizing the TTL to RS232 converter IC MAX202 on the board. You will need to use a communications program on the PC to monitor the data sent by the PIC.

Exercise: -

-Modify the code to receive a number and show the next one on 7-Segments and sent it to the PC again.

-Modify the code to classify the characters into 9 groups (1-9) as following and show the number of group on 7-Segments

Group	1	2	3	4	5	6	7	8	9
Character	A, B, C	D, E, F	G, H, I	J, K, L	M, N, O	P, Q, R	S, T, U	V, W, X	Y, Z

```
;*********************************************************
; This program to receive a character from the PC and return (send) next
;character to PC again.
;*********************************************************

      include "p16f877A.inc"

__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_OFF & _XT_OSC

;*********************************************************
; User-defined variables

      cblock      0x20
            WTemp             ; Must be reserved in all banks
            StatusTemp
            Counter
      endc

      cblock      0x0A0             ; bank 1 assignnments
            WTemp1                  ; bank 1 WTemp
      endc

      cblock      0x120             ; bank 2 assignnments
            WTemp2                  ; bank 2 WTemp
      endc

      cblock      0x1A0             ; bank 3 assignnments
            WTemp3                  ; bank 3 WTemp
      endc
;*********************************************************
; Macro Assignments

push  macro
      movwf WTemp ;WTemp must be reserved in all banks
      swapf       STATUS,W    ;store in W without affecting status bits
      banksel     StatusTemp  ;select StatusTemp bank
      movwf StatusTemp  ;save STATUS
      endm

pop   macro
      banksel     StatusTemp        ;point to StatusTemp bank
      swapf       StatusTemp,W      ;unswap STATUS nybbles into W
      movwf STATUS  ;restore STATUS (which points to where W was stored)
      swapf       WTemp,F           ;unswap W nybbles
      swapf       WTemp,W           ;restore W without affecting STATUS
      endm
;*********************************************************
; Start of executable code
      org   0x00        ; Reset vector
      nop
      goto  Main
;*********************************************************
; Interrupt vector
      org   0x04        ; interrupt vector
      goto  IntService
```

```
;*********************************************************
; Main program

Main
      call  Initial           ; Initialize everything
MainLoop
      nop
      nop
      goto  MainLoop    ; Do it again

;*********************************************************
; Initial Routine

Initial
      movlw D'25'       ; This sets the baud rate to 9600
      banksel     SPBRG       ; assuming BRGH=1 and Fosc=4.000 MHz
      movwf SPBRG

      banksel     RCSTA
      bsf         RCSTA,SPEN  ; Enable the serial port
      bcf         RCSTA,RX9   ; Disable9-bit Receive
      bsf         RCSTA,CREN  ; Enable continuous receive

      banksel     TXSTA
      bcf         TXSTA,SYNC  ; Set up the port for asynchronous
                              ;operation
      bsf         TXSTA,TXEN  ; Transmit enabled
      bsf         TXSTA,BRGH  ; High baud rate
      bcf         TXSTA,TX9   ; Disable9-bit send

      banksel     PIE1        ; Enable the Timer2 interrupt
      bsf   PIE1, RCIE
      bcf   TRISC,RC6         ; Set RC6 to output Send Pin
      bsf   TRISC,RC7         ; Set RC7 to input Receive Pin

      banksel     INTCON      ; Enable global and peripheral interrupts
      bsf   INTCON, GIE
      bsf   INTCON, PEIE

      banksel     Counter
      clrf    Counter

      return

;*********************************************************
; Interrupt Service Routine
; This routine is called whenever we get an interrupt.
IntService
      push
      btfsc PIR1, RCIF  ; Check for a Timer2 interrupt
      call  RECEIVE
      pop
      retfie
```

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

RECEIVE

      movf  RCREG,w
      movf  RCREG,w
      movwf Counter
      incf  Counter,w

      banksel     TXREG
      movwf TXREG       ; Send a next character out the serial port
      banksel     TXSTA
L1    btfss TXSTA,TRMT
      goto  L1

      return

      end
```

University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp.8: - Serial with A/D protocol-based transmission

Objectives

1. To become familiar with the process of writing an assembly language program for the PIC.
2. To demonstrate different methods of handling the A/D conversion process.
3. To demonstrate different methods of handling the serial communications through the USART.
4. To demonstrate methods of remote control using serial communications.
5. To demonstrate the use of internal interrupts.

Pre-lab Preparation:

1. Review the sections in the book regarding the A/D.
2. Review the instruction set of the PIC 16F877A.
3. Review the sections in the book regarding the USART.
4. Read the PIC16F877A data sheet especially chapter 10.

Procedure:

In this lab experiment you are required to write an assembly code for a PIC16F877A which operates as data acquisition system (DAQ) for machine that has two analogue sensors:

1) Pressure Sensor
2) Temperature Sensor

Your code should transmit for each sensor a message that contains the sensor readings and a control bit to the computer. The message consists of the following fields:

1- **Control Bit**: It is a 9th bit of Transmit Data: -
   - Zero: to indicate that the message contains the pressure sensor reading.
   - One: to indicate that the message contains the Temperature sensor reading.

2- **Message data**: - It is the Most 8-Significant Bit (MSB) of the A/D conversion result.

The serial transmission for the pressure sensor is ***event*** *driven*: the reading should be transmitted whenever **absolute difference** between two readings greater than 25, the reading is ready to transmit, without considering the time spent between these different readings.

However, the serial transmission for the Temperature sensor reading is ***time*** *driven*: the readings should be transmitted every 50 ms.

Exercise1: - Modify the Pressure_Test subroutine code to make a test for ***absolute difference*** not for positive difference only.

Exercise2: - Write a subroutine code to return (Send) the digital conversion value as following: -

Conversion Value range	Return Value "Send Value"
0-25	0
26-76	1
77-127	2
128-178	3
179-229	4
230-255	5

```
;*********************************************************
; This program Serial with A/D protocol based transmission.
;*********************************************************

      include "p16f877A.inc"

__CONFIG    _CP_OFF & _WDT_OFF & _BODEN_OFF & _PWRTE_OFF & _XT_OSC
;*********************************************************
; User-defined variables

      cblock      0x20
            WTemp             ; Must be reserved in all banks
            StatusTemp
            Pressure_Reading
            Difference
            Pressure_Result
            Temperature_Reading
            Counter
      endc

      cblock      0x0A0             ; bank 1 assignnments
            WTemp1                  ; bank 1 WTemp
      endc

      cblock      0x120             ; bank 2 assignnments
            WTemp2                  ; bank 2 WTemp
      endc

      cblock      0x1A0             ; bank 3 assignnments
            WTemp3                  ; bank 3 WTemp
      endc
;*********************************************************
; Macro Assignments

push  macro
      movwf WTemp       ;WTemp must be reserved in all banks
      swapf STATUS,W    ;store in W without affecting status bits
      banksel     StatusTemp  ;select StatusTemp bank
      movwf StatusTemp  ;save STATUS
      endm

pop   macro
      banksel     StatusTemp  ;point to StatusTemp bank
      swapf StatusTemp,W      ;unswap STATUS nybbles into W
      movwf STATUS   ;restore STATUS (which points to where W was stored)
      swapf WTemp,F           ;unswap W nybbles
      swapf WTemp,W           ;restore W without affecting STATUS
      endm

;*********************************************************
; Start of executable code

      org   0x00        ; Reset vector
      nop
      goto  Main
```

```
;*******************************************************
; Interrupt vector

      org   0x04        ; interrupt vector
      goto  IntService
;*******************************************************
; Main program

Main
      call  Initial           ; Initialize everything
Mainloop
      call  Pressure_Conversion
      call  Pressure_Test
      call  Temperature_Conversion
      goto  Mainloop    ; Do it again

;*******************************************************
; Initial Routine

Initial
      movlw D'25'       ; This sets the baud rate to 9600
      banksel     SPBRG       ; assuming BRGH=1 and Fosc=4.000 MHz
      movwf SPBRG

      banksel     RCSTA       ; Enable the serial port
      bsf         RCSTA, SPEN
      banksel     TXSTA
      bcf         TXSTA, SYNC ; Set up the port for asynchronous operation
      bsf         TXSTA, TXEN ; Transmit enabled
      bsf         TXSTA, BRGH ; High baud rate
      bsf         TXSTA,TX9   ; Enable 9-bit send
      banksel     PIE1        ; Enable the Timer2 interrupt
      bsf         PIE1, TMR2IE
      bcf         TRISC,RC6   ; Set RC6/TX to output Send Pin
      bsf         TRISA,RA0   ; Set RA0 to input
      bsf         TRISA,RA1   ; Set RA1 to input

      banksel     INTCON      ; Enable global and peripheral interrupts
      bsf         INTCON, GIE
      bsf         INTCON, PEIE
      movlw D'194'      ; Set up the Timer2 Period register to get 50ms
      banksel     PR2
      movwf PR2
      movlw B'01111110' ; Set up Timer2 postscale=1:16, prescaler=16
      banksel     T2CON
      movwf T2CON

      banksel           ADCON1
      movlw B'00000100' ; A/D data left justified, 3 analog channels AN0,
                        ;AN1 and AN3
                                    ; VDD and VSS references
      banksel     ADCON0            ; Select register bank 0
      movlw       0x02
      movwf       Counter

      return
```

```
;*********************************************************
      Pressure_Conversion
      banksel     ADCON0
      movlw B'01000001' ; Fosc/8, A/D Channel 0, A/D enabled
      movwf ADCON0
      bsf         ADCON0, GO  ; Start A/D conversion
Wait
      btfsc ADCON0,2    ; Wait for conversion to complete
      goto  Wait
      movf  ADRESH, W   ; Get A/D result
      movwf Pressure_Reading
      return
;*********************************************************
      Pressure_Test

      decfsz      Counter,f         ;Send first reading regardless of the
      goto  Send                    ;value of the change in pressure
      movf  Pressure_Result,w
      subwf Pressure_Reading,w      ;Pressure_Reading - Pressure_Result
      movwf Difference
      sublw d'25'       ; 25 - Difference
      btfsc STATUS,C
      goto  Continue
Send
      movf  ADRESH, W         ; Get A/D result
      movwf Pressure_Result
      call  Pressure_Transmision
Continue
      movlw 0x01
      movwf Counter

      return

;*********************************************************
      Pressure_Transmision

      banksel     TXSTA
      bcf         TXSTA,TX9D  ;9th bit of Transmit Data = 0
      banksel     TXREG
      movf  Pressure_Result,w
      movwf TXREG       ; Send a pressure reading out the serial port
      banksel     TXSTA
L1    btfss TXSTA,TRMT
      goto  L1
      banksel     ADCON0            ; Select register bank 0
      return
;*********************************************************
Temperature_Conversion
      banksel     ADCON0
      movlw B'01001001'       ; Fosc/8, A/D Channel 1, A/D enabled
      movwf ADCON0
      bsf         ADCON0, GO  ; Start A/D conversion
Wait1
      btfsc ADCON0,2          ; Wait for conversion to complete
      goto  Wait1
```

```
      movf  ADRESH, W         ; Get A/D result
      movwf Temperature_Reading

      return
;*********************************************************
; Interrupt Service Routine
; This routine is called whenever we get an interrupt.
IntService
      push

;     btfsc PIR1, TMR2IF      ; Check for a Timer2 interrupt

      banksel     TXSTA
      bsf         TXSTA,TX9D  ;9th bit of Transmit Data = 1
      banksel     TXREG
      movf  Temperature_Reading,w
      movwf TXREG       ; Send temperature reading out the serial port
      banksel     TXSTA
L2    btfss TXSTA,TRMT
      goto  L2
      banksel     PIR1
      bcf         PIR1, TMR2IF      ; Clear the Timer2 interrupt flag

      pop

      retfie
;*********************************************************
      end
```

Objectives

1. To become familiar with Pulse Width Modulation in software.
2. To demonstrate the use of external interrupts linked with the port B on-change.

Pre-lab Preparation:

1. Review the sections in the book regarding PWM (Chapter 7).
2. Review the instruction set of the PIC 16F877.
3. Read the assembly program carefully.

Procedure:

In this experiment, we are going to use four pushbuttons on the board connected with the supply voltage to produce logic 1 (5Volt). Each pushbutton will set a different value that used to pulse width modulate, a signal is connected to the light bulb on the board to get a visual indication of the effect of the different values on the PWM signal.

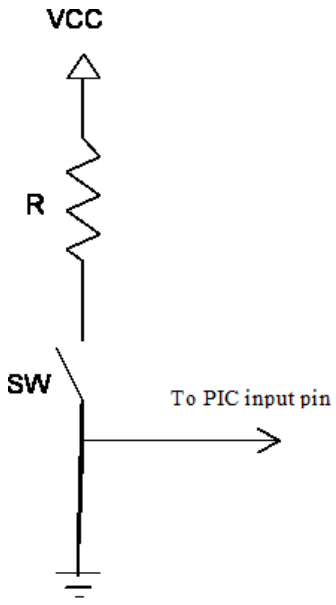


Fig.1: - Switch connecting to PIC

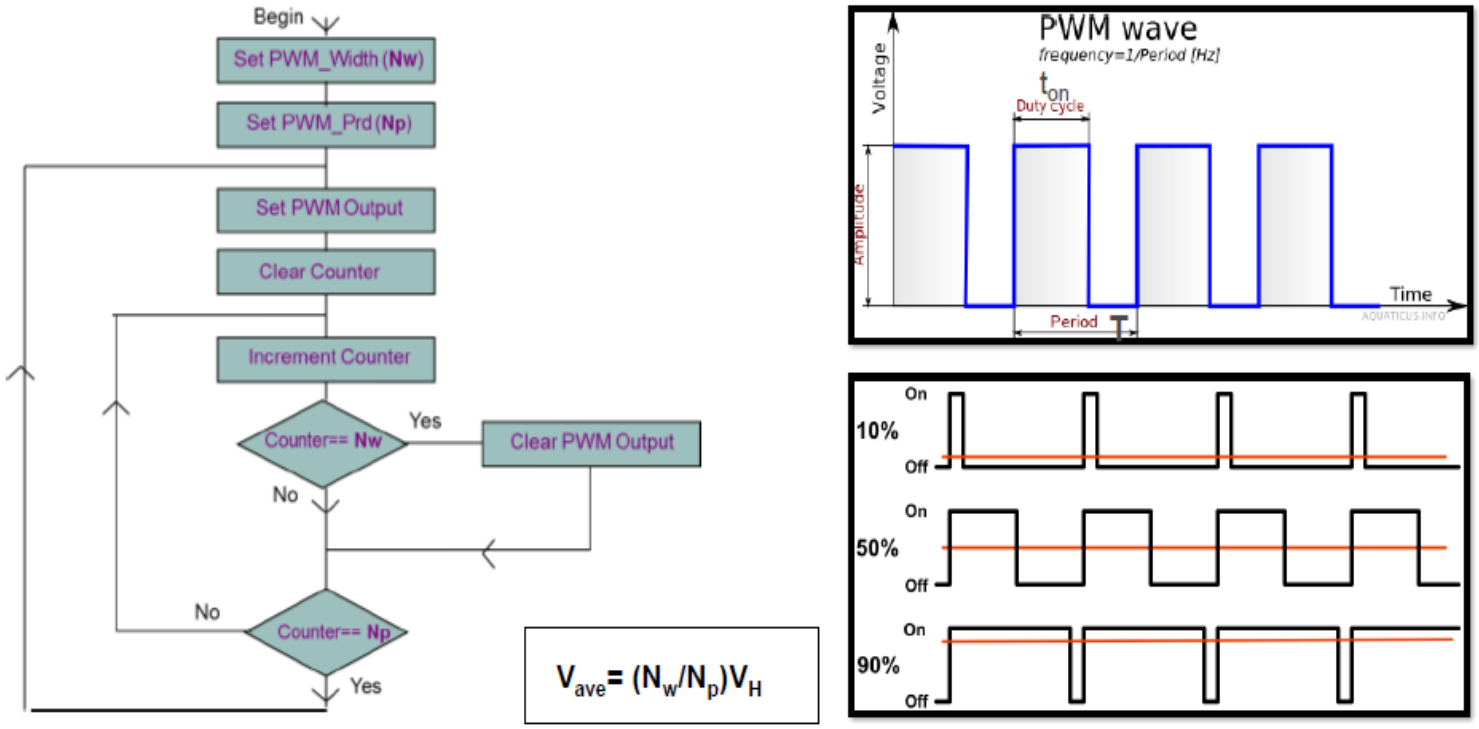


Fig.2: - PWM Flow Chart and Concept

```
;*******************************************************
; Lab10.asm
;This program operates a car lighting system using 4 pushbuttons
;Each pushbutton will set a different value that used to pulse width
;modulate
;pushbuttons are connected to the pins of port B (RB4-RB7).
; light bulb is connected to 6.
; The program uses a PIC16F877A running at crystal oscillator of
;frequency 4MHz.
;*******************************************************
      include     "p16f877A.inc"
;*******************************************************
; Macro definitions
push  macro
      movwf       WTemp       ; WTemp must be reserved in all banks
      swapf       STATUS,W    ; store in W without affecting status bits
      banksel     StatusTemp        ; select StatusTemp bank
      movwf       StatusTemp        ; save STATUS
      endm
pop   macro
      banksel     StatusTemp        ; point to StatusTemp bank
      swapf       StatusTemp,W            ; unswap STATUS nibbles into W

      movwf       STATUS            ; restore STATUS
      swapf       WTemp,F           ; unswap W nibbles
      swapf       WTemp,W           ; restore W without affecting STATUS
      endm
;*******************************************************
Sec_1       equ       D'10'   ; Number of centiseconds in a second
CountOuter0 equ       D'10'
CountInner0 equ       D'250'
;*******************************************************
; User-defined variables
      cblock            0x20        ; bank 0 assignnments
                  WTemp ; WTemp must be reserved in all banks
                  StatusTemp
                  PWM_Width
```

```
                PWM_Period
                Counter
                BLNKCNT
                CountOuter
                CountInner
        endc
        cblock          0x0A0       ; bank 1 assignnments
                WTemp1              ; bank 1 WTemp
        endc
        cblock          0x120       ; bank 2 assignnments
                WTemp2              ; bank 2 WTemp
        endc
        cblock          0x1A0       ; bank 3 assignnments
                WTemp3              ; bank 3 WTemp
        endc
;*********************************************************
; Start of executable code
        org         0x000
        nop
        goto        Initial
;*********************************************************
; Interrupt vector
        org         0x0004
        goto        INT_SVC           ; jump to the interrupt service
routine
;*********************************************************
; Initial Routine
Initial
        banksel     PORTC
        clrf        PORTC       ;Clear PORTC
        bsf         INTCON,GIE        ;Enable Global Interrupt
        bsf         INTCON,RBIE ;Enable RB Port Change Interrupt

        banksel     TRISC
        clrf        TRISC             ; All of the PORTC bits are outputs
```

```
      movlw       0xF0
      movwf       TRISB  ;Set port B pins (RB0-RB3 outputs, RB4-Rb7
                         ;inputs)
      banksel     ADCON0            ; Select register bank 0
      clrf        PWM_Width
      clrf        PWM_Period
;*********************************************************
; Main Routine
Main
      sleep
      comf        PWM_Period
L1
      bsf         PORTC,RC6         ;Set PWM signal to RC6
      clrf        Counter
L2
      incf        Counter,F
      movf        PWM_Width,w
      subwf       Counter,w
      btfsc       STATUS,Z
      bcf         PORTC,RC6         ;clear PWM signal from RC6
      movf        PWM_Period,w
      subwf       Counter,w
      btfsc       STATUS,Z
      goto        L1
      goto        L2
;*********************************************************
; Interrupt Service Routine
INT_SVC
      push
      call  Duty_Select
      pop
      retfie
;*********************************************************
; Duty_Select Routine
Duty_Select
      btfsc       PORTB,RB4
      goto        Duty_25
```

```
      btfsc       PORTB,RB5
      goto        Duty_50
      btfsc       PORTB,RB6
      goto        Duty_75
      btfsc       PORTB,RB7
      goto        Duty_100
      goto        Cont
Duty_25
      movlw d'64'
      movwf PWM_Width
      goto  Cont
Duty_50
      movlw d'128'
      movwf PWM_Width
      goto  Cont
Duty_75
      movlw d'192'
      movwf PWM_Width
      goto  Cont
Duty_100
      movlw d'255'
      movwf PWM_Width
Cont
      call  Delay
      movf  PORTB,w
      bcf         INTCON,RBIF
      return

;;;;;;; Delay subroutine ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; This subroutine to get a delay with 100 mSec.
Delay

Sec   movlw Sec_1
      movwf BLNKCNT

TenMs
      movlw CountOuter0
```

```
        movwf CountOuter

DecO
        movlw CountInner0
        movwf CountInner

DecI
        nop
        decfsz      CountInner, F
        goto  DecI
        decfsz      CountOuter, F
        goto  DecO
        decfsz      BLNKCNT, F
        goto  TenMs
        Return
;*****************************************************
        end
```

Objectives

1. Knowing the various modes of operation of the LCD (8-bit/4-bit interface, 2-lines/1-line, CG-RAM
2. Distinguishing between the commands for the instruction register and data register.
3. To become familiar with keypad Interfacing and usage.

Introduction:

1- *L*iquid Crystal *D*isplays (LCD)

What is an LCD?

A **L**iquid Crystal **D**isplays (LCD) is a thin, flat display device made up of any number of color or monochrome pixels arrayed in front of a light source or reflector. It is often utilized in battery-powered electronic devices because it uses very small amounts of electric power.

LCDs can display numbers, letters, words, and a variety of symbols. This experiment teaches you about LCDs which are based upon the Hitachi HD44780 controller chipset. LCDs come in different shapes and sizes with 8, 16, 10, 24, 32, and 40 characters as standard in 1, 2 and 4–line versions. **However, all LCD's regardless of their external shape are internally built as a 40x2 format. See Figure 2 below**

Figure 1: A typical LCD module

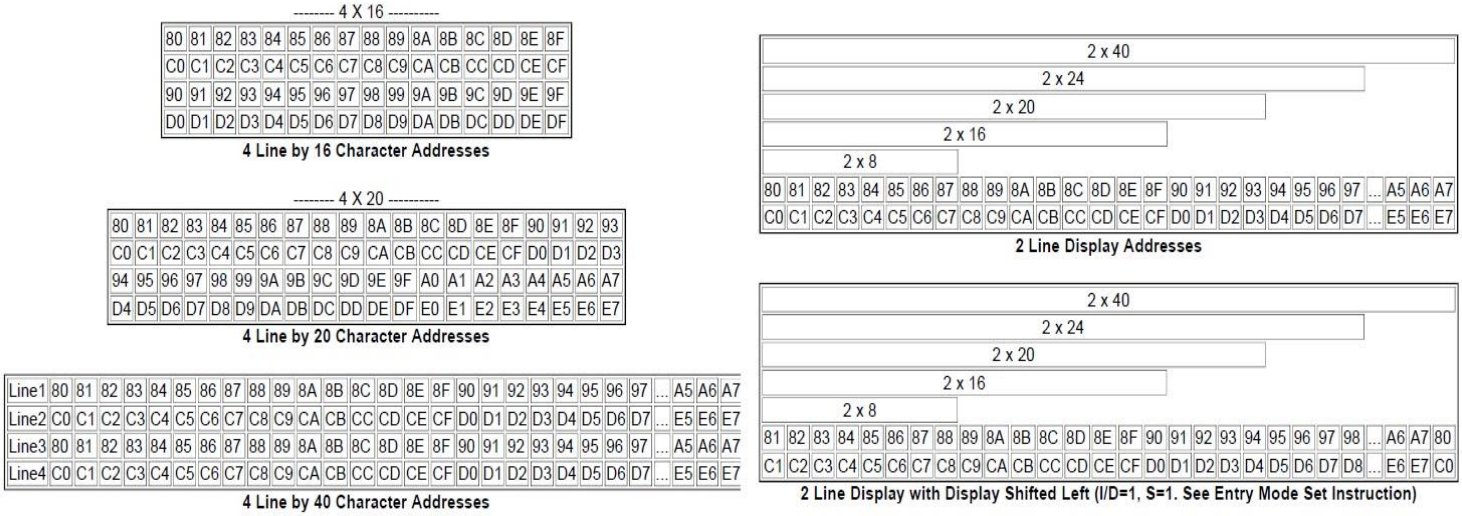


Figure 2: Different LCD modules shapes and sizes

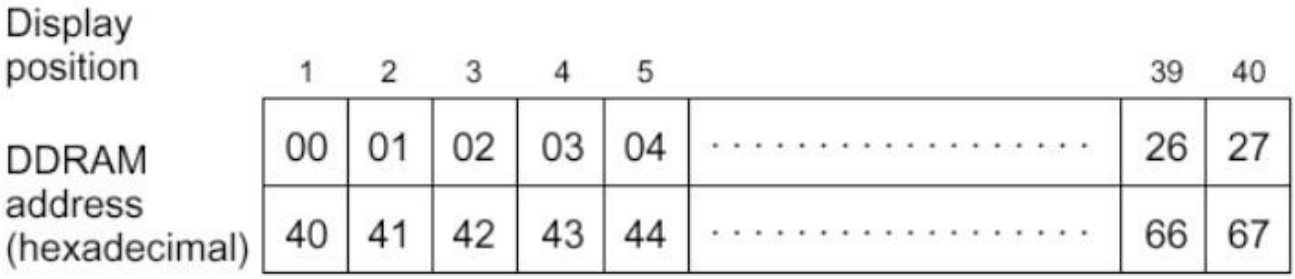


Figure 3: Display address assignments for HD44780 controller-based LCDs

LCD I/O

Most LCD modules conform to a standard interface specification. A 14-pin access is provided having eight data lines, three control lines and three power lines as shown below. Some LCD modules have 16 pins where the two additional pins are typically used for backlight purposes

Note: This image might differ from the actual LCD module, the order can be from left to right or vice versa therefore you should pay attention, pin 1 is marked to avoid confusion (printed on one of the pins).

Powering up the LCD requires connecting three lines: one for the positive power ***Vdd*** (usually +5V), one for negative power (or ground) ***Vss***. The ***Vee*** pin is usually connected to a potentiometer which is used to vary the contrast of the LCD display. We will connect this pin to the GND.

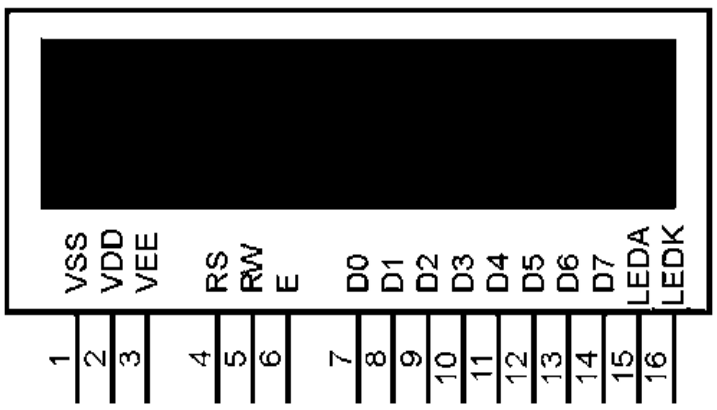


Figure 4: LCD Pinout

As you can see from the figure, the LCD connects to the microcontroller through three control lines: RS, RW and E, and through eight data lines D0-D7.

With 16-pin LCDs, you can use the L+ and L- pins to turn the backlight (BL) on/off.

Table1: LCD pin-out details

Pin No.	Pin Name	Pin Type	Pin Description	Pin Connection
Pin 1	Ground	Source Pin	This is a ground pin of LCD	Connected to the ground of the MCU/ Power source
Pin 2	VCC	Source Pin	This is the supply voltage pin of LCD	Connected to the supply pin of Power source
Pin 3	V0/VEE	Control Pin	Adjusts the contrast of the LCD.	Connected to a variable POT that can source 0-5V
Pin 4	Register Select	Control Pin	Toggles between Command/Data Register	Connected to a MCU pin and gets either 0 or 1. 0 -> Command Mode 1-> Data Mode
Pin 5	Read/Write	Control Pin	Toggles the LCD between Read/Write Operation	Connected to a MCU pin and gets either 0 or 1. 0 -> Write Operation 1-> Read Operation
Pin 6	Enable	Control Pin	Must be held high to perform Read/Write Operation	Connected to MCU and always held high.
Pin 7-14	Data Bits (0-7)	Data/Command Pin	Pins used to send Command or data to the LCD.	In 4-Wire Mode Only 4 pins (0-3) is connected to MCU In 8-Wire Mode All 8 pins (0-7) are connected to MCU
Pin 15	LED Positive	LED Pin	Normal LED like operation to illuminate the LCD	Connected to +5V
Pin 16	LED Negative	LED Pin	Normal LED like operation to illuminate the LCD connected with GND.	Connected to ground

Sending Commands/Data to the LCD

Using an LCD is a simple procedure once you learn it. Simply put you will place a value on the LCD lines D0-D7(this value might be an ASCII value (character to be displayed), or another hexadecimal value corresponding to a certain command). So how will the LCD differentiate if this value on D0-D7 is corresponding to data or command?

Observe the figure below, as you might see the only difference is in the RS signal (**R**egister **S**elect), this is the only way for the LCD controller to know whether it is dealing with a character or a command.

Command	Binary										
	RS	R/W	E	D7	D6	D5	D4	D3	D2	D1	D0
Write Data to CG or DD RAM	1	0	↓	ASCII Value							
Write Command	0	0	↓	Refer to the Command Table below							

Figure 5: Necessary control signals for Data/Commands

Displaying Characters

All English letters and numbers (as well as special characters, Japanese and Greek letters) are built in the LCD module in such a way that it conforms to the **ASCII standard**. To display a character, you only need to send its ASCII code to the LCD which it uses to display the character.

To display a character on the LCD simply move the ASCII character to the working register (for this experiment) then call send_char subroutine.

Notice that from column 1 to D, the character resolution is 5 pixels wide x 7 pixels high (5x7) (column 0 is a special case, it is 1x8, but considered as 5x7, more on this later) whereas the character resolution of columns E and F is 5 pixels wide x 10 pixels high (5x10).

Figure6: Correspondence between Character Codes and Character Patterns (ROM Code: A00)

Command	Binary D7	D6	D5	D4	D3	D2	D1	D0	Hex
Clear Display	0	0	0	0	0	0	0	1	01
Display & Cursor Home	0	0	0	0	0	0	1	x	02 or 03
Character Entry Mode	0	0	0	0	0	1	1/D	S	04 to 07
Display On/Off & Cursor	0	0	0	0	1	D	U	B	08 to 0F
Display/Cursor Shift	0	0	0	1	D/C	R/L	x	x	10 to 1F
Function Set	0	0	1	8/4	2/1	10/7	x	x	20 to 3F
Set CGRAM Address	0	1	A	A	A	A	A	A	40 to 7F
Set Display Address	1	A	A	A	A	A	A	A	80 to FF

1/D:	1=Increment*, 0=Decrement	R/L:	1=Right shift, 0=Left shift
S:	1=Display shift on, 0=Off*	8/4:	1=8-bit interface*, 0=4-bit interface
D:	1=Display on, 0=Off*	2/1:	1=2 line mode, 0=1 line mode*
U:	1=Cursor underline on, 0=Off*	10/7:	1=5x10 dot format, 0=5x7 dot format*
B:	1=Cursor blink on, 0=Off*		
D/C:	1=Display shift, 0=Cursor move	x = Don't care	* = Initialization settings

Figure 7: LCD command control codes

Set CG-RAM Address command Syntax: 01AAAAAA

If you give a closer look at Figure 6, you will clearly see that the table only contains English and Japanese characters, numbers, symbols as well as special characters! Suppose now that you would like to display a character not found in the built-in table of the LCD (i.e. an Arabic Character). In this case we will have to use what is called the CG-RAM (Character Generation RAM), which is a reserved memory space in which you could draw your own characters and later display them.

Observe column one in Figure 6, the locations inside this column are reserved for the CG-RAM. Even though you see 16 locations (0 to F), you only have the possibility to use the first 8 locations 0 to 7 because locations 8 to F are mirrors of locations 0 – 7.

So, to organize things, to use our own characters, we must do the following:

1. Draw and store our own defined characters in CG-RAM
2. Display the characters on the LCD screen as if it were any of the other characters in the table

Drawing and storing our own defined characters in CG-RAM

As stated earlier, we have eight locations to store our characters in. So how do we choose which location out of these to start drawing and building our characters in?
The answer is quite simple; follow this rule as stated in the datasheet of the HD44780 controller

1. To write (build/store a character in location 00 (crossing of the row and column)), you send the CG-RAM address command as follows: 01*AAAAAA* → 01*000000* → 0x40
   2. However, to write in any location from 01 to 07, you must skip eight locations So, the CG-RAM address command will send **0x48** (to store a character in location 1**), 0x50** (to store a character in location 2) and so on...

So up to this point we have defined **where** to write our characters but not how to build them. Draw a 5x8 Grid and start drawing your character inside, then replace each shaded cell with one and not shaded ones with zero.

Append three zeros to the left (B5-B7) and finally transform the sequence into hexadecimal format. This is the sequence which you will fill in the CG-RAM SEQUENTIALLY once you have set the CG-RAM Address before.

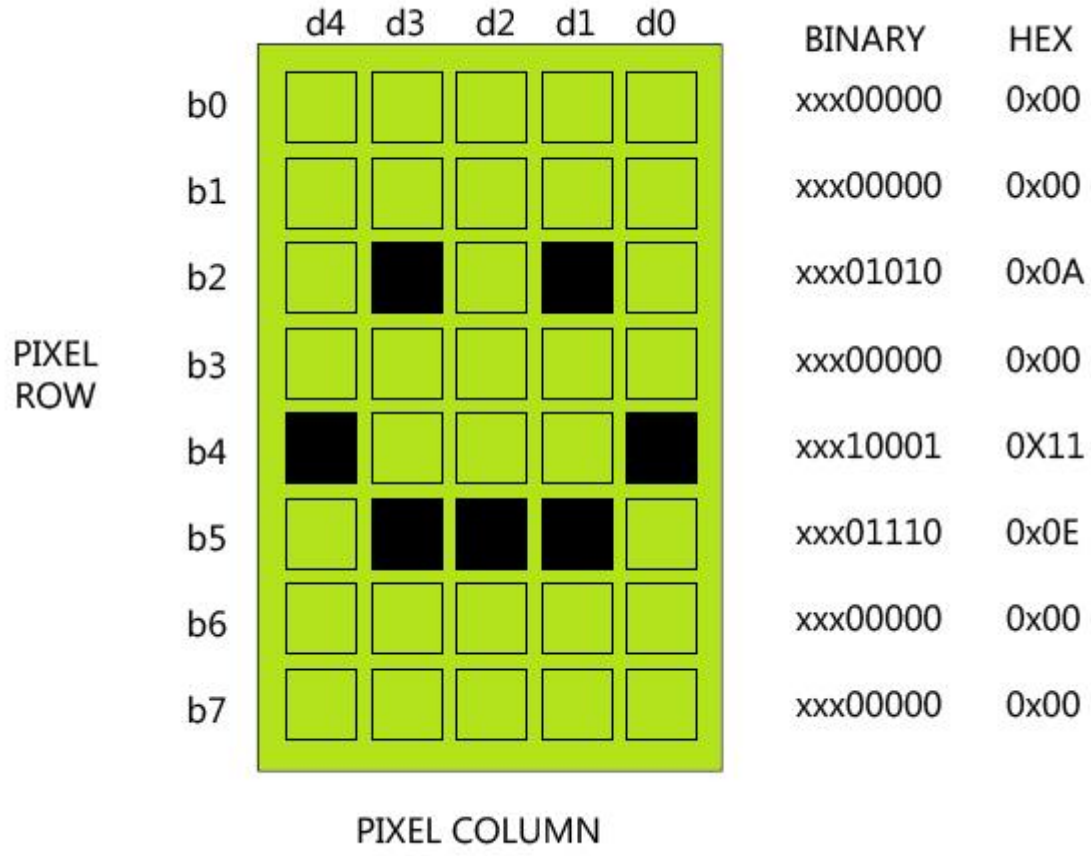


Figure 8: CG-RAM drawing example

Displaying the user generated (drawn) characters on the LCD screen

Simply, if we stored our character in location 0, we move 0 to the working register then issue the **"call send_char"** command, if we stored it in location 2, move 2 to the working register and so on ….

2- Keypad

Basic Keypad Theory

Keypads are essentially large switch arrays which allow data entries (numeric or alphanumeric) into systems. Keypads are widely used in everyday applications such as burglar alarms, cell phones and photocopiers. Keypads come in different shapes and sizes with 4x3, 4x4 buttons as common examples. It is not practical to connect each button in the keypad to its own port input as we previously did with switch and push buttons; therefore, keypads are normally constructed in a matrix format. An (n x m)

Figure 9: A 4x4 keypad

General Keypad Operation

In general, a keypad is interfaced in a way such that initially if no key is pressed you will read a certain logic level and when you press a button a signal with the negative of the original level will be read. You have two cases:
1. Fix the initial button state to be read as logic 1 (using pull-up resistors), when you press a button you will read logic 0.

2. Fix the initial button state to be read as logic 0 (using pull-down resistors), when you press a button you will read logic 1.

- Pull-up and pull-down resistors are used to limit the amount of current and protect the circuit. (not to read a floating state)

- Pull-up and pull-down resistors are normally connected externally, BUT you can make use of the internal pull-up resistors found in Microchip's PIC devices such as those in implemented in PORTB. In this experiment we will use the internal pull-up resistors.

Whether you use internal or external pull-up resistors the keypad will operate in the same way.

Technique

******It does not matter whether you start scanning rows or columns first it depends on your connections; the basic idea is the logic of the scanning technique is the same.

First the row bits are set to output, with the column bits as input. The output rows are set to logic 0. If no button is pressed all column line inputs will be read as logic 1 due to the action of the pull-up resistors. If, however, a button is pressed then its corresponding switch will connect column and row lines, and the corresponding column line will be read as low.

To detect this logic transition from high to low (that is to know whether a key has been pressed or not), we must either:
1. Keep pulling the inputs (columns) continuously until 0 is detected.
2. Make use of the interrupt (Here PORTB interrupt- on change will be beneficial)

Yet still, we have identified the column in which the key was pressed but not the button itself. So, what we do now is save the column and repeat the same procedure above with the following minor modification:

Secondly the column bits are set to output, with the row bits as input. The output columns are set to logic 0. Since the button is still pressed then its corresponding switch is still connecting column and row lines, and the corresponding row line will be read as low. If, however, the button is released all row line inputs will be read as logic 1 due to the action of the pull-up resistors. Now we have identified the row

Example

Suppose we connect the columns to PORTB 4-7 as input and the rows to PORTB 0-3 as output (with the value of 0). If we continuously read the inputs they will always be read as 1 because of the internal pull-up resistors on PORTB. If one presses number “7”, this will make us read logic 0 on RB4 (identified that we have pressed a button in the first column).

Now let us exchange the inputs for the outputs, that is we connect the rows to PORTB 4-7 as input and the columns to PORTB 0-3 as output (with the value of 0).

If we read the input, we will find that RB2 is 0. Now we have identified the location of the pressed button and ready to process what it means.

So, what comes next?
The above scanning technique let us know the position of the pressed button (in terms of its row/column intersection) but not the value corresponding to the button. So obviously the next step is to use the location to retrieve the desired value.

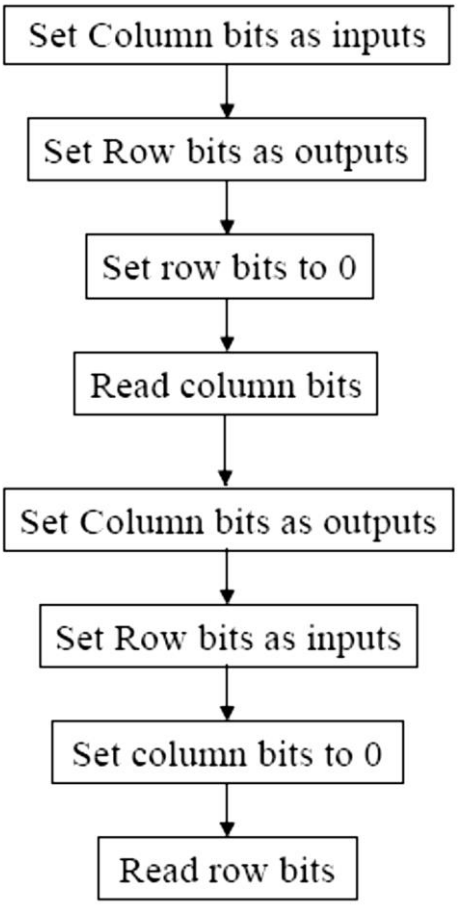


Table 2 - The values which will be read when a key is pressed

Key pressed	**Column** RB7, RB6, RB5, RB4	**Row** RB3, RB2, RB1, RB0	Look-up table index		
1	1110	1110	COL1	0	0 +0
4	1110	1101		1	0+1
7	1110	1011		2	0+2
A	1110	0111		3	0+3
2	1101	1110	COL2	4	4+0
5	1101	1101		5	4+1
8	1101	1011		6	4+2
0	1101	0111		7	4+3
3	1011	1110	COL3	8	8+0
6	1011	1101		9	8+1
9	1011	1011		10	8+2
B	1011	0111		11	8+3
F	0111	1110	COL4	12	12+0
E	0111	1101		13	12+1
D	0111	1011		14	12+2
C	0111	0111		15	12+3

Most often, you will need to display the number on a 7-segment display, LCD or use it in binary calculations. Therefore, it is natural to build a look-up table with the 7-segment representations, ASCII code or binary equivalent and use the location pattern which you saved (as in the table above) as an index to the look-up table.

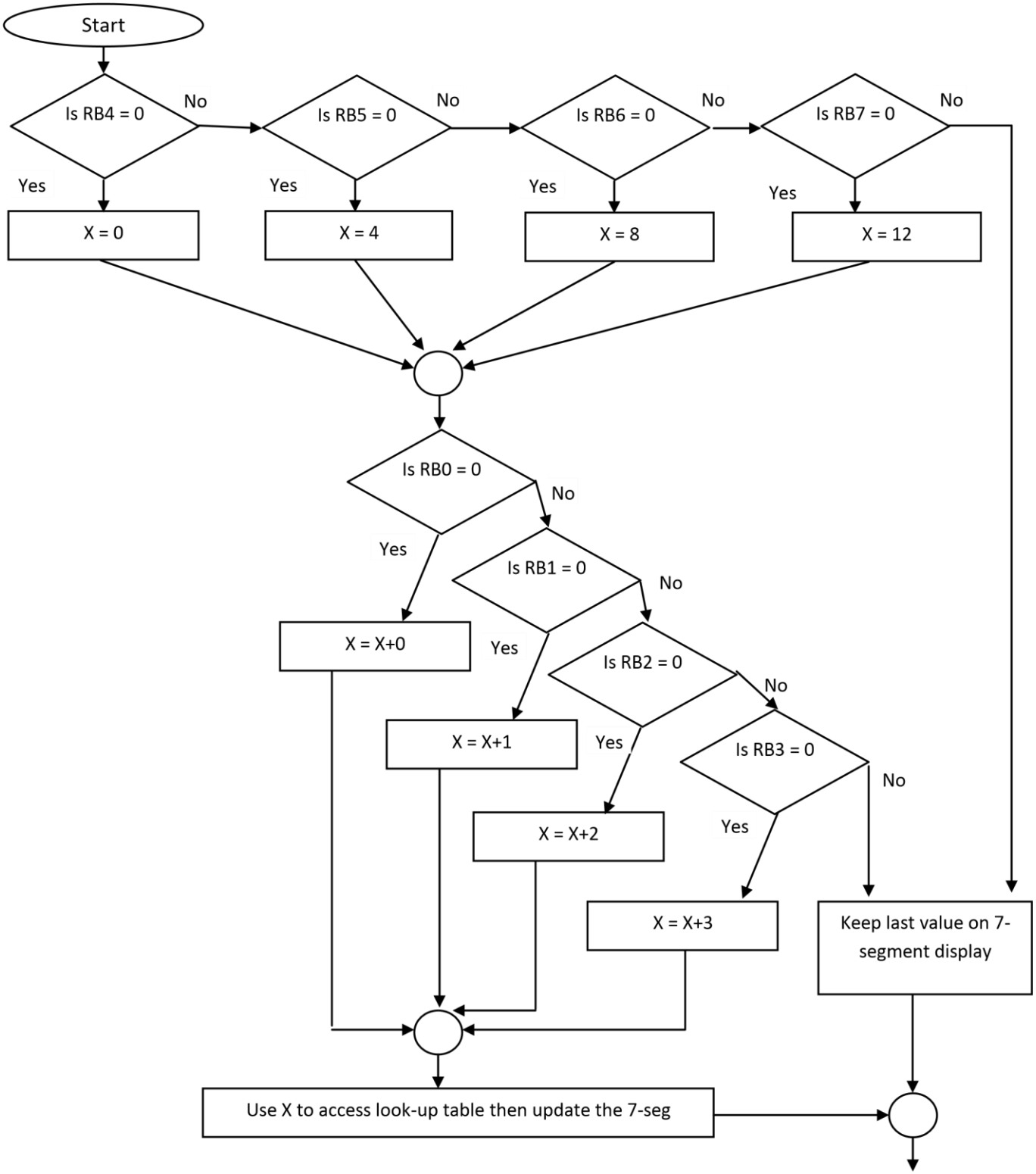
The way one organizes the look-up table entries differs from one person to another, therefore there is no specific way to translate the locations to their corresponding values. One might use a series of **btfsc or btfss** instructions or deduce a relationship and use mathematical operations or a combination of both.

What have we done in this experiment?

Study the following flowchart which (in the next page) is based on the table above.

1. If the key pressed is in column 1, then X might take the values 0, 1, 2, 3
2. If the key pressed is in column 2, then X might take the values 4, 5, 6, 7
3. If the key pressed is in column 3, then X might take the values 8, 9, 10, 11
4. If the key pressed is in column 4, then X might take the values 12, 13, 14, 15

These values will be added to PCL to retrieve values from the look-up table.


Start
Is RB4 = 0
No
Yes
X = 0
Is RB5 = 0
No
Yes
X = 4
Is RB6 = 0
No
Yes
X = 8
Is RB7 = 0
No
Yes
X = 12
Is RB0 = 0
No
Yes
X = X+0
Is RB1 = 0
No
Yes
X = X+1
Is RB2 = 0
No
Yes
X = X+2
Is RB3 = 0
No
Yes
X = X+3
Keep last value on 7-segment display
Use X to access look-up table then update the 7-seg

```
;*******************************************************************************
*
;                                 Example Code
;*******************************************************************************
*
; Function:
;
;
; Connections:
;            Input:
;                  Keypad Row 1:RB0
;                  Keypad Row 2:RB1
;                  Keypad Row 3:RB2
;                  Keypad Row 4:RB3
;                  Keypad Col 1:RB4
;                  Keypad Col 2:RB5
;                  Keypad Col 3:       RB6
;                  Keypad Col 4:       RB7
;     LCD Control:
;                        RA1: RS      (Register Select)
;                        RA3: E (LCD Enable)
;     LCD Data:
;     PORTD 0-7 to LCD DATA 0-7 for sending commands/characters
; Notes:
;     The RW pin (Read/Write) - of the LCD - is connected to RA2
;     The BL pin (Back Light) – of the LCD – is connected potentiometer
;            Output:
;                  7-Segment A-G: PORTC 0-6
;                  7-Segment Digit Enable 1: Connected To RA0 On Board

__CONFIG
_DEBUG_OFF&_CP_OFF&_WRT_HALF&_CPD_OFF&_LVP_OFF&_BODEN_OFF&_PWRTE_OFF&_WDT_OFF&_X
T_OSC
;*******************************************************************************
*
#INCLUDE "P16F877A.INC"

CBLOCK 0x20
DELCNTR1            ; Used in generating 10 ms delay
DELCNTR2
KPAD_PAT            ; Holds the pattern retrieved from keypad
KPAD_ADD            ; Holds keypad address to lookup table (generated from
                        ; KPAD_PAT to get KPAD_CHAR)
KPAD_CHAR           ; Holds the 7-segment representation of the most recent
                        ; character pressed on keypad
;tempChar
;charCount
lsd                     ;lsd and msd are used in delay loop calculation
msd
ENDC
;*******************************************************************************
*
Zero         equ        B'00111111' ; 7-Segment Code for Zero
One          equ        B'00000110'     ; 7-Segment Code for One
Two          equ     B'01011011' ; 7-Segment Code for Two
Three        equ        B'01001111' ; 7-Segment Code for Three
Four         equ     B'01100110' ; 7-Segment Code for Four
Five         equ     B'01101101' ; 7-Segment Code for Five
Six          equ     B'01111101' ; 7-Segment Code for Six
Seven        equ        B'00000111' ; 7-Segment Code for Seven
Eight        equ     B'01111111' ; 7-Segment Code for Eight
Nine         equ     B'01101111' ; 7-Segment Code for Nine
```

```
LetterA             equ          B'01110111'  ; 7-Segment Code for A
LetterB             equ          B'01111100'  ; 7-Segment Code for B
LetterC             equ          B'01011000'  ; 7-Segment Code for C
LetterD             equ          B'01011110'  ; 7-Segment Code for D
LetterE             equ          B'01111001'  ; 7-Segment Code for E
LetterF             equ          B'01110001'  ; 7-Segment Code for F
;*****************************************************************************
; START OF EXECUTABLE CODE
;*****************************************************************************
      ORG    0x00
      GOTO   INITIAL
;*****************************************************************************
; INTERRUPT VECTOR
;*****************************************************************************
      ORG    0x04
      GOTO   KPAD_TO_7SEG
;*****************************************************************************
INITIAL
             BANKSEL TRISA
             CLRF    TRISA
             CLRF   TRISD
             CLRF   TRISC
             MOVLW  B'11110000'  ; PORTB initially row bits as output, column
                                 ;as input
             MOVWF  TRISB
             BCF    OPTION_REG, NOT_RBPU   ; Turn on all internal pull-ups of PORTB

             BANKSEL ADCON1
             MOVLW   0X06
             MOVWF   ADCON1             ;set PORTA as general Digital I/O PORT

             BANKSEL PORTB
             CLRF   PORTB
             movlw  Zero                   ; Send Zero to output
             movwf  PORTC
             BSF          PORTA,RA0
             BCF          INTCON, RBIF    ; Initialize and enable port-on-change
             BSF          INTCON, RBIE    ; interrupt
             BSF          INTCON, GIE
;Initialize LCD
             Movlw  0x38         ;8-bit mode, 2-line display, 5x7 dot format
             Call   send_cmd
             Movlw  0x0e         ;Display on, Cursor Underline on, Blink off
             Call   send_cmd
             Movlw  0x02         ;Display and cursor home
             Call   send_cmd
             Movlw  0x01         ;clear display
             Call   send_cmd
             call   DrawStick1
            Call    DrawStick2
             Movlw  0x01         ;clear display
             Call   send_cmd
;*****************************************************************************
***
call delay
Movlw  a'A'
   Call         send_char
```

```
Main

       movf   KPAD_CHAR,w
       sublw  One
       btfsc  STATUS,Z
       call    Page1
       movf   KPAD_CHAR,w
       sublw  Two
       btfsc  STATUS,Z
       call    Page2
       movf   KPAD_CHAR,w
       sublw  Three
       btfsc  STATUS,Z
       call    Page3

       goto   Main                ;Do it again
;*********************************************************************************
DrawStick1                        ; Setting the CGRAM address at which we draw the
stick man
       Movlw  0x40         ; Here it is address 0x00
              Call   send_cmd
       Movlw  0X0E         ; Sending data that implements the Stick man
       Call   send_char
       Movlw  0X11
       Call   send_char
        Movlw 0X0E
        Call  send_char
        Movlw 0X04
        Call  send_char
        Movlw 0X1F
        Call  send_char
        Movlw 0X04
        Call  send_char
         Movlw0X0A
         Call  send_char
         Movlw0X11
         Call  send_char
       Return

;*********************************************************************************
***
DrawStick2                        ; Setting the CGRAM address at which we draw the
stick man
        Movlw 0x48         ; Here it is address 0x01
        Call  send_cmd
        Movlw 0X0E         ; Sending data that implements the Stick man
        Call  send_char
        Movlw 0X0A
        Call  send_char
         Movlw0X04
          Call send_char
         Movlw0X15
         Call send_char
         Movlw0X0E
         Call send_char
         Movlw0X04
         Call send_char
        Movlw 0X0A
        Call  send_char
        Movlw 0X0A
        Call  send_char
       Return
```

```
;*******************************************************************************
Page1
          Movlw    0x88         ;Set display address
             Call   send_cmd
             Movlw  0x00
             Call    send_char
             Movlw  0x88         ;Set display address
             Call   send_cmd
             call   delay
             call   delay
             Movlw   0x01
       Call    send_char
       return
Page2

             Movlw  0x01         ;clear display
             Call   send_cmd
Movlw  0x02         ;Display and cursor home
             Call   send_cmd
return
Page3

             Movlw  0x01
             Call    send_char
             call   delay
             call   delay
             Movlw   0x00
       Call    send_char
       return
;*******************************************************************************
send_cmd
       movwf  PORTD  ; Refer to table 1 on Page 5 for review of this subroutine
       bcf    PORTA,RA1
       bsf    PORTA,RA3
       nop
       bcf    PORTA,RA3
       bcf    PORTA,RA2
       call   delay
       return
;*******************************************************************************
send_char
       movwf  PORTD  ; Refer to table 1 on Page 5 for review of this subroutine
       bsf    PORTA,RA1
       bsf    PORTA,RA3
       nop
       bcf    PORTA,RA3
       bcf    PORTA,RA2
       call   delay
       return
;*******************************************************************************
delay
             movlw  0x80
             movwf  msd
             clrf   lsd
loop2
             decfsz lsd,f
             goto   loop2
             decfsz msd,f
endLcd
             goto   loop2
             return
```

```
;*********************************************************************
; INTERRUPT SERVICE ROUTINE.
;*********************************************************************
; Keypad press has been detected. Does the following:
;1. Solves de-bouncing through software delay
;2. Gets keypad pattern (location of the pressed button)
;3. Converts location to table index
;4. Access look-up table with index, gets 7-seg. Code, display on 7-segment.
;5. Solves the port-on-change interrupt when button is released (2nd call delay):
;The code will enter the interrupt service routine two times for the same ;button,
;once when the button is pressed (change from 1 to 0), another when ;the button
;is released (change from 0 to 1),
;therefore we insert the second delay such that the action of pressing/releasing
;the button will occur inside ;the interrupt routine,
;and when it happens we will clear the flag only once ;and not enter the subroutine
;again for the release action.
;*********************************************************************
KPAD_TO_7SEG
            CALL   DELAY
            CALL   KPAD_RD
            CALL   KP_CODE_CONV
            CALL   DELAY
            MOVF   PORTB, W        ; READ PORTB VALUE.
            BCF          INTCON, RBIF   ; CLEAR INTERRUPT FLAG
            RETFIE

;*********************************************************************
; Function:
;      Gets the coordinates of the pressed keypad button and stores it
; Input:
;      Nibble Values from PORTB: initially high nibble of PORTB then low nibble
; Output:
;      Pressed button coordinates in the matrix in the form {Column, row} store in
;      KPAD_PAT
; REFER TO THE FLOWCHART ON PAGE 3 TO FULLY UNDERSTAND HOW THIS
; SUBROUTINE WORKS
;*********************************************************************
KPAD_RD
            MOVF   PORTB,W                    ; Read Column
            ANDLW  B'11110000'         ; Ensure unwanted bits are suppressed
            MOVWF  KPAD_PAT
            BSF    STATUS, RP0         ; Set row as input, column as output.
            MOVLW  B'00001111'
            MOVWF  TRISB
            BCF    STATUS, RP0
            CLRF PORTB                 ; Send Zero to output
            MOVF   PORTB, W            ; Read Row
            ANDLW  B'00001111'         ; Ensure unwanted bits are suppressed
            IORWF  KPAD_PAT, 1
            BSF    STATUS, RP0
            MOVLW  B'11110000'     ; Restore row as output, column as input
            MOVWF  TRISB
            BCF          STATUS, RP0
            CLRF   PORTB               ; Send Zero to output
            RETURN
;*********************************************************************
; Function:
;      Converts keypad pattern held in KPAD_PAT to an index KPAD_ADD which we use
;      to access the look-up table.
; INPUT:
;      KPAD_PAT which holds the location in terms of {Column,Row} of the pressed
;      key
```

```
; OUTPUT:
; A value in KPAD_ADD in the range of 0 to 15 which is the index to be used in the
; look-up table
; REFER TO THE FLOWCHART ON PAGE 5 TO FULLY UNDERSTAND HOW THIS
; SUBROUTINE
;*********************************************************************
KP_CODE_CONV
            CLRF   KPAD_ADD           ; Initially Base index is 0
KP0         BTFSC  KPAD_PAT, 4        ; Is Column1?
            GOTO   KP1
            GOTO   ROW_FIND
KP1         BTFSC  KPAD_PAT, 5        ; Is Column2?
            GOTO   KP2
            MOVLW  B'00000100'        ; Base index is 4
            ADDWF  KPAD_ADD, 1
            GOTO   ROW_FIND
KP2         BTFSC  KPAD_PAT, 6        ; Is Column3?
            GOTO   KP3
            MOVLW  B'00001000'        ; Base index is 8
            ADDWF  KPAD_ADD, 1
            GOTO   ROW_FIND
KP3
            BTFSC  KPAD_PAT, 7        ; Is Column4?
            GOTO   KEEP               ; If no button was pressed, display last
                                      ; character on 7-segment display
            MOVLW  B'00001100'        ; Base index is 12
            ADDWF  KPAD_ADD, 1

ROW_FIND
            BTFSC  KPAD_PAT, 0        ; Is Row1?
            GOTO   RF1
            GOTO         KEYPAD_OP
RF1         BTFSC  KPAD_PAT, 1        ; Is Row2?
            GOTO   RF2
            MOVLW  B'00000001'         ; Add 1 to the base index
            ADDWF  KPAD_ADD, 1
            GOTO         KEYPAD_OP

RF2         BTFSC  KPAD_PAT, 2        ; Is Row3?
            GOTO   RF3
            MOVLW  B'00000010'        ; Add 2 to the base index
            ADDWF  KPAD_ADD, 1
            GOTO         KEYPAD_OP

RF3         BTFSC  KPAD_PAT, 3        ; Is Row4?
            GOTO   KEEP               ; If no button was pressed, display last
                                                ;   character   on   7-segment
display
            MOVLW  B'00000011'        ; Add 3 to the base index
            ADDWF  KPAD_ADD, 1
KEYPAD_OP
            MOVF   KPAD_ADD, 0
            CALL   KP_TABLE           ; Access table with index
            MOVWF  KPAD_CHAR          ; Save character
            MOVWF  PORTC              ; Display Character
            GOTO   FIN
KEEP
            MOVF   KPAD_CHAR, W
            MOVWF  PORTC              ; If no button was pressed, display last
                                                ;   character   on   7-segment
display
FIN         RETURN
```

```
;*********************************************************************
; This table contains the common cathode 7-segment representations of the numbers
; 0 to 9 and the characters ‘A’ to ‘f’ found on the keypad.
;As seen below, the table lists the numbers in the order Col1, Col2 and son on.
;*********************************************************************
KP_TABLE
             ADDWF  PCL,1
             RETLW  One                ;'1'   COLUMN1
             RETLW  Four               ;'4'
             RETLW  Seven              ;'7'
             RETLW   LetterA           ;'A'
             RETLW  Two                ;'2'   COLUMN2
             RETLW  Five               ;'5'
             RETLW  Eight              ;'8'
             RETLW  Zero               ;'0'
             RETLW  Three              ;'3'   COLUMN3
             RETLW  Six                 ;'6'
             RETLW  Nine               ;'9'
             RETLW  LetterB            ;'B'
             RETLW  LetterF            ;'F'   COLUMN4
             RETLW  LetterE            ;'E'
             RETLW  LetterD            ;'D'
             RETLW  LetterC            ;'C'
;*********************************************************************
; Delay subroutine
; Delay of approx. 10ms which is more than enough for de-bouncing
;*********************************************************************
DELAY
             MOVLW  0X20
             MOVWF  DELCNTR1
             CLRF   DELCNTR2
LOOP2
             DECFSZ DELCNTR2,F
             GOTO   LOOP2
             DECFSZ DELCNTR1,F
ENDLCD
             GOTO         LOOP2
             RETURN
             END
;*********************************************************************
```