University of Jordan
School of Engineering
Department of Mechatronics Engineering
Microprocessor and Microcontroller Laboratory
0908432
Exp. 4: MPLAB Basics

Objective

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

Pre-lab Preparation:

Review Experiment 3 thoroughly.
Read chapter 7 of the PIC16F84 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.

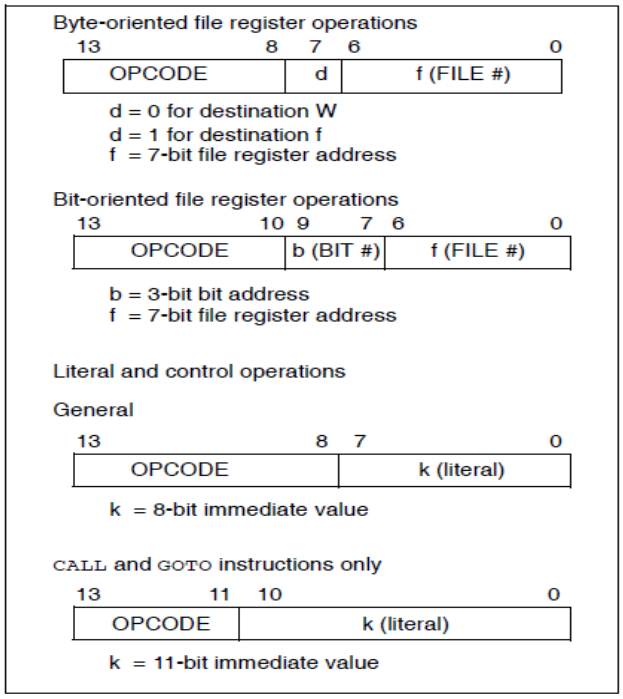
Why use assembly while we have the high-level-language "HLL" alternatives?
Assembly once learnt and professionally used offers several advantages over HLL programming in that the professional programmer can use it to write **smaller codes** in comparison with that produced by HLL code compilers "this 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)

A- 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
```

B- 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

C- 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
        movftestNum, 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 true 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'10110111'
V2	0x21	D'39'
V3	0x22	a'C'
Result	0x23	?

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 (Loc30) with the decimal value of 15 Initialize location 0x40 (Loc40) with the value of 0

Loc30 = Loc30 – Loc40
Loc40 = Loc40 + 1
Repeat until Loc30 = 0

Include a screenshot of your work showing the watch window and displaying the final values of Loc30 and Loc40.