EL3002-MICROPROCESSOR INTERFACING & PROGRAMMING

LAB MANUAL

DEPARTMENT OF ELECTRICAL ENGINEERING,

FAST-NU, LAHORE

Created by: Samia Mahmood

Date: August, 2022

Last Updated by: Maaz Rizvi

Date: July, 2024

Approved by: Head of Electrical Engineering Department, Lahore Campus

Date: July, 2024

Table of Contents

List of Equipment

Sr. No.	Description
1	EMU8086
2	RIMS 8051 Based Microcontroller Trainers
3	Computer System
4	MP Lab IDE
5	Proteus Simulator
6	7-Segment Displays
7	Telephone Keypads
8	6V DC Motors

EXPERIMENT 1

INTODUCTION TO EMU8086

OBJECTIVES:

- To learn how to use EMU8086 for writing assembly language programs for PIC
- To learn how to use Watch Window for debugging programs in EMU8086

EQUIPMENT/TOOLS:

- EMU8086

INTRODUCTION:

8086 Microprocessor Emulator, also known as EMU8086, is an emulator of the program 8086 microprocessor. It is developed with a built-in 8086 assembler. It has an instruction queue, which is capable of storing six instruction bytes from the memory resulting in faster processing. emulator runs programs like the real microprocessor in step-by-step mode. it shows registers, memory, stack, variables and flags.

Table 1: Summary of Registers

Segment Registers		
CS	Code Segment	16-bit number that points to the active code-segment
DS	Data Segment	16-bit number that points to the active data-segment
SS	Stack Segment	16-bit number that points to the active stack-segment
ES	Extra Segment	16-bit number that points to the active extra-segment
Pointer Registers		
IP	Instruction Pointer	16-bit number that points to the offset of the next instruction
SP	Stack Pointer	16-bit number that points to the offset that the stack is using
BP	Base Pointer	used to pass data to and from the stack
General-Purpose Registers		
AX	Accumulator Register	mostly used for calculations and for input/output
BX	Base Register	Only register that can be used as an index
CX	Count Register	register used for the loop instruction
DX	Data Register	input/output and used for multiply and divide
Index Registers		
SI	Source Index	used by string operations as source
DI	Destination Index	used by string operations as destination

PROCEDURE:

1. Launch EMU8086 by double clicking on short cut of EMU8086 on your desktop or by running Emu8086.exe.

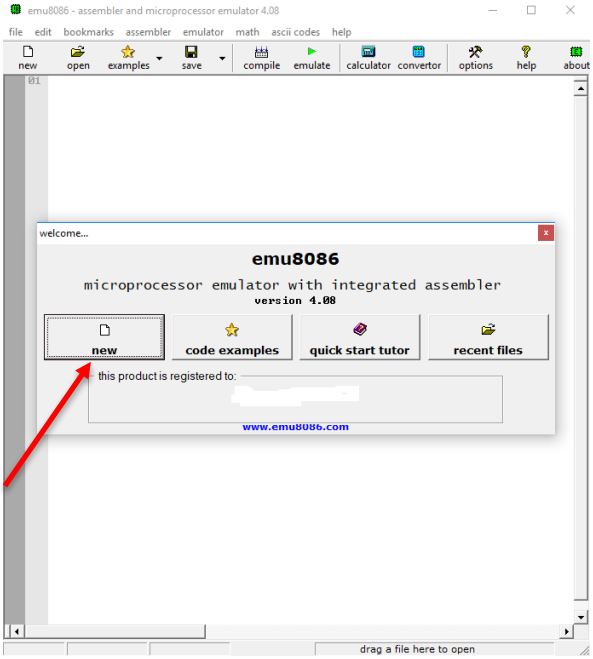


Figure 1: EMU8086 Interface

2. Select empty workspace and click OK button as shown in figure 2.

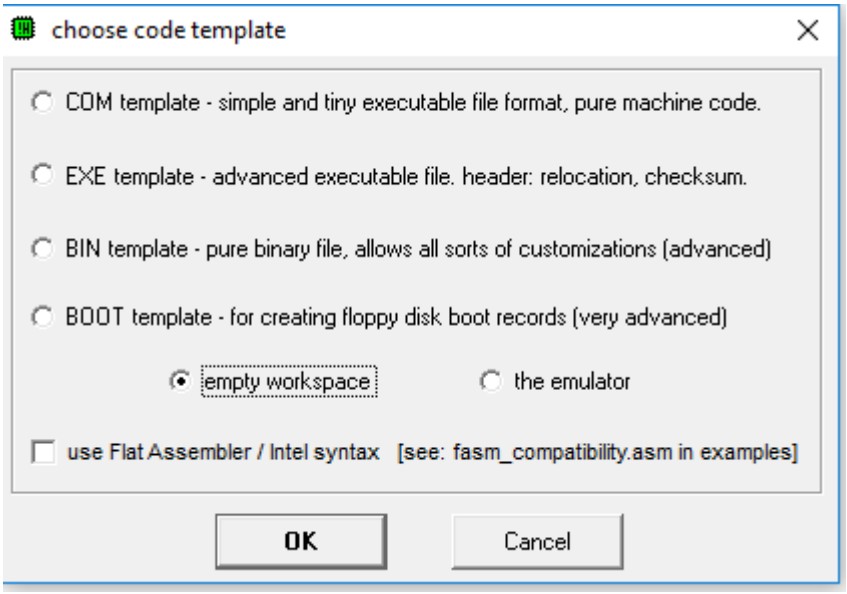


Figure 2: EMU 8086 Code template

3. Type the code given below in the empty workspace (editor).

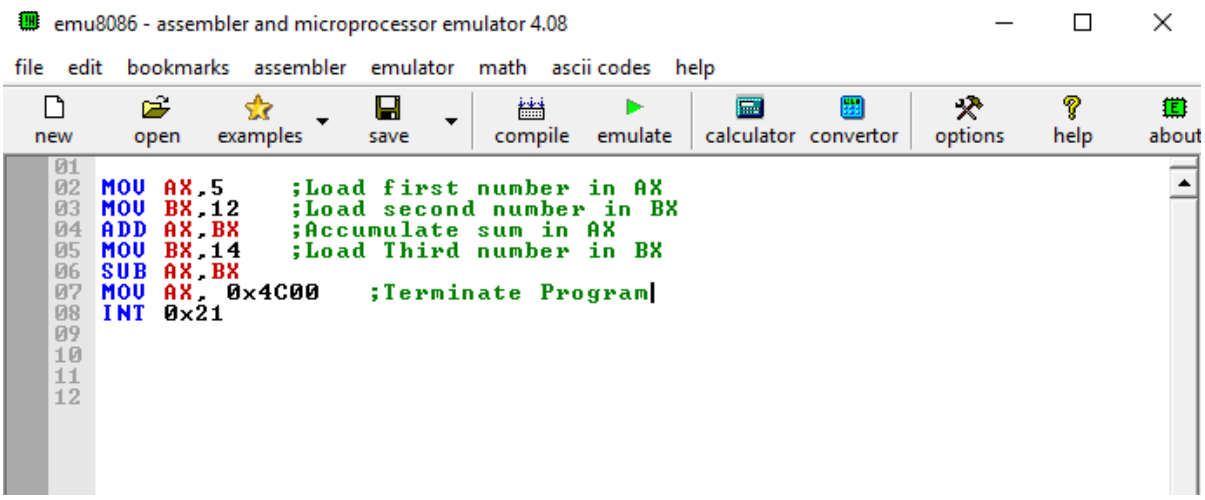


Figure 3: EMU 8086 Workspace

4. Click on the emulate button on the taskbar that starts assembling and linking of the source file. Binary file and original source code of emulator will appear as shown in figure 4:

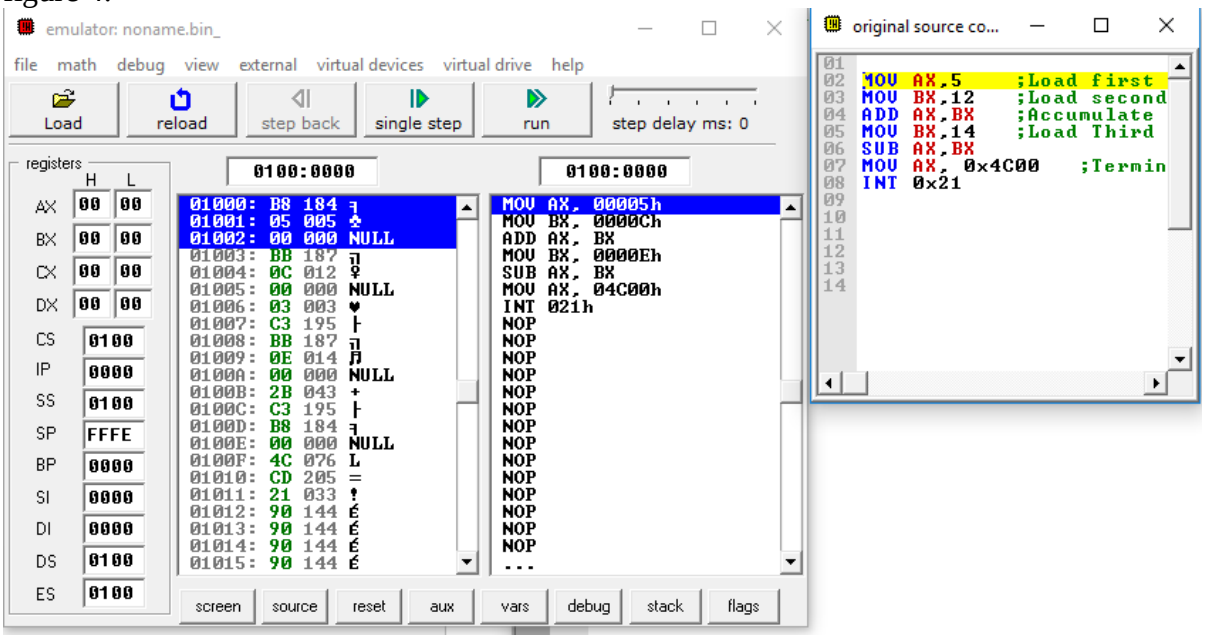


Figure 4: Emulator of EMU8086

The interface of the debugger is easy to understand and use. Using F8 or the single step button we can step through the code and see the changes taking place in the data registers.

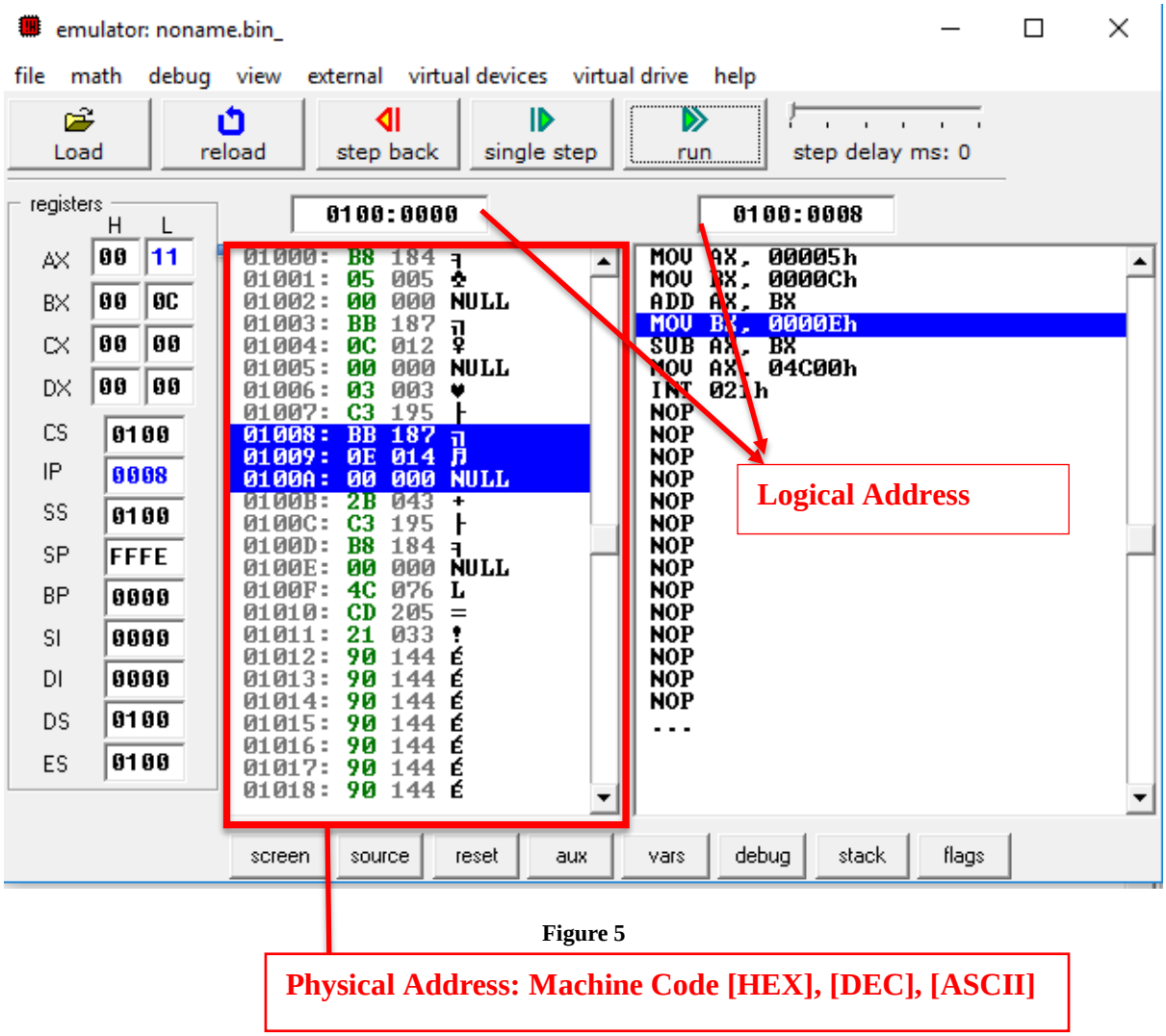


Figure 5

5. Click on the aux->memory button to observe values at different memory locations as shown in the figure 6:

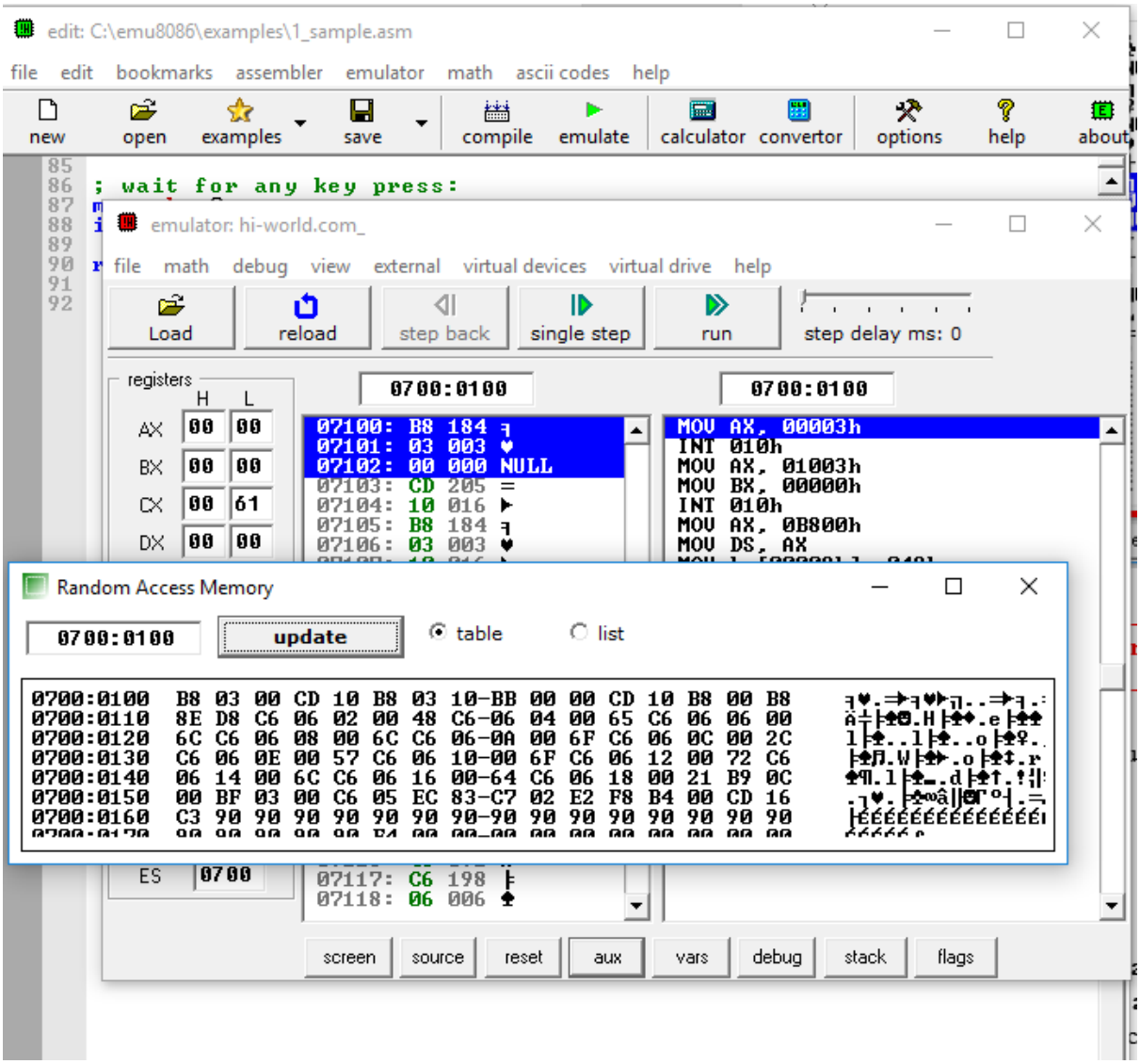


Figure 6: Memory Window

LAB TASK:

Write an assembly language program to move first four digits of your roll numbers at memory locations shown below:

03000H	03002H	03004H	03006H	03008H

Write the code in the space provided below:

POST LAB QUESTION:

Write an assembly language program to store characters of your name, at memory locations shown below:

03000H	03002H	03004H	03006H	03008H

Write the code in the space provided below:

EXPERIMENT 2

DATA MOVEMENT USING DIFFERENT ADDRESSING MODES

OBJECTIVES:

- To learn to use EMU8086 for writing assembly language programs for 8086
- To learn to write and execute assembly language program using various addressing modes.

EQUIPMENT/TOOLS:

- EMU8086

8086 Addressing Modes:

The x86 processors support **register addressing mode**, **immediate addressing mode**, **indirect addressing mode**, **indexed addressing mode**, and the **direct addressing mode**. The following paragraphs explain each of these modes:

Register Addressing Mode*:* Register operands are the easiest to understand. Consider the following forms of the MOV instruction:

MOV AX, BX
MOV AX, CX
MOV AX, DX

Immediate Addressing Mode: Constants are also pretty easy to deal with. Consider the following instructions:

MOV AX, 25H
MOV BX, 195H
MOV CX, 2056H
MOV DX, 1000H

Direct Addressing Mode: There are three addressing modes which deal with accessing data in memory. These addressing modes take the following forms:

MOV AX, [1000]

The first instruction above uses the direct addressing mode to load AX with the 16 bit value stored in memory starting at location 1000 hex.

Indirect Addressing Mode:

MOV AX, [BX]
MOV AX, [1000+BX]

The MOV AX, [BX] instruction loads AX from the memory location specified by the contents of the BX register

Indexed Addressing Mode: The last memory addressing mode is the indexed addressing mode. An example of this memory addressing mode is

MOV AX, [1000+BX]
This instruction adds the contents of BX with 1000 to produce the address of the memory value to fetch. This instruction is useful for accessing elements of arrays, records, and other data structures.

Data Transfer Instructions: These instructions are used to transfer the data from source operand to destination operand. All the store, move, load, exchange, input and output instructions belong to this group.

General purpose byte or word transfer instructions:

- **a. MOV:** To move data from source to destination that can be one of the following:
  - i. From immediate value (8 or 16-bit) to destination register (8 or16-bit)
  - ii. From immediate value (8 or 16-bit) to destination memory (8 or16-bit)
  - iii. From source register (8 or 16-bit) to destination register (8 or16-bit)
  - iv. From source memory (8 or 16-bit) to destination register (8 or16-bit)
  - v. From source register (8 or 16-bit) to destination memory (8 or16-bit)
- **b. PUSH:** To move any value from an instruction to the top of stack segment
- **c. POP:** To pop out any value from the top of stack segment to the referred register
- **d. PUSHA:** To push the contents of all registers to the top of stack segment.
- **e. POPA:** To pop out the contents of stack to all the registers
- **f. XCHG:** To interchange or exchange the data between source and destination register

PROBLEMS:

1. Write an assembly language program to store last four digits of your roll number to the following location, using Immediate Addressing Mode.

03000H	03002H	03004H	03006H

Write the code in the space provided below:

2. Write an assembly language program to add the numbers stored in the locations mentioned in question no. 1 and save the result at 03008H memory location:
   a. using indirect addressing mode.
   b. using direct addressing mode.

c. using indexed addressing modes.

Write the code in the space provided below:

POST LAB QUESTION:

1. Use first four digits of your roll number values to initialize all registers of 8086.
   a. Write a program to send these register values to stack segment one by one using PUSH statement
   b. Pop the in same order one by one and check the results.
   c. Use PUSHA and POPA to do the tasks mentioned in parts a and b.

Write the code in the space provided below:

EXPERIMENT 3

ARITHMETIC AND LOGIC INSTRUCTIONS IN EMU8086

OBJECTIVES:

- To learn to write arithmetic instruction in EMU8086
- To learn to use EMU8086 to write and execute assembly language programs for addition, subtraction, multiplication and division.

EQUIPMENT/TOOLS:

- EMU8086

Arithmetic and Logic Instructions:

Arithmetic instructions are used to perform addition, subtraction, multiplication and division. Logical instructions include logical AND, logical OR, logical XOR, OR complement etc.

Arithmetic Instruction	Operands D and S are Destination and Source (can be register data or memory address)	Description
ADD	D,S	Add the provided byte to byte/word to word
ADC	D,S	Add with carry
INC	D	Increment the provided byte/word by 1
SUB	D,S	Subtract the byte from byte/word from word
SBB	D,S	Perform subtraction with borrow
DEC	D	Decrement the provided byte/word by 1
NEG	D	Negate each bit of the provided byte/word and add 1 or 2's complement
MUL	8 bit register	Multiply unsigned byte by byte/word by word.
IMUL	8 or 16 bit register	Multiply signed byte by byte/word by word.
DIV	8 bit register	Divide the unsigned word by byte or unsigned double word by word
IDIV	8 or 16 bit register	Divide the signed word by byte or signed double word by word

Logical Instruction	Operands D and S are Destination and Source (can be register data or memory address)	Description
AND	D,S	Perform logical AND operation between two operands and stores the result back into the destination operand
OR	D,S	Perform the OR operation between two operands and stores the result back into the destination operand
NOT	D	Implement the bitwise NOT operation

XOR	D,S	Gives 1 when D and S are different and 0 if they are same
ROR	D, Count	Rotate bits of byte/word towards the right, i.e. LSB to MSB and to Carry Flag [CF]
ROL	D, Count	Rotate bits of byte/word towards the left, i.e. MSB to LSB and to Carry Flag [CF]
RCR	D, Count	Rotate bits of byte/word towards the right, i.e. LSB to CF and CF to MSB
RCL	D, Count	Rotate bits of byte/word towards the left, i.e. MSB to CF and CF to LSB

PROBLEMS:

1. Write and execute an assembly language program to solve following arithmetic expression.

$$(2\mathrm{x} + 3\mathrm{y} + 4\mathrm{z})(4z + 3y)$$

TEST VALUES:

1) $x = 2,\ y = 3,\ z = 5$
2) $x = 200,\ y = 3,\ z = 5$
3) $x = 100,\ y = 3,\ z = 5$

Write the code in the space provided below:

2. Write an assembly language program to:
    a. reverse an 8-bit number.
    b. reverse 16 bit number using 8 bits operation. Assume 16 bit number is stored at memory location 2050H and 2051H.
    c. show masking of lower and higher nibbles of an 8-bit number.

Write the code in the space provided below:

POST LAB QUESTION:

1. Develop a sequence of instructions to subtract 00FFH from 00EEH.

Write the code in the space provided below:

EXPERIMENT 4

PROGRAM CONTROL FLOW INSTRUCTIONS in 8086 MICROPROCESSOR

OBJECTIVES:

- To understand behavior of different program control flow instructions
- To learn to write programs using different program control flow instructions in EMU8086

EQUIPMENT/TOOLS:

- EMU8086

Program control flow instructions:
These instructions are used to transfer/branch the instructions during the execution of a program. These instructions use conditional/unconditional jump and loop instructions to alter the program flow.

Unconditional transfer instructions:

1. CALL: Call a procedure, save return address on stack
2. RET: Return from procedure to the main program.
3. JMP: Go to specified address to get next instruction

Condition Transfer Instructions:

Mnemonic	Jump Condition	Description
JA	CF=0 and ZF=0	Jump if Above
JAE	CF=0	Jump if Above or Equal
JB	CF=1	Jump if Below
JBE	CF=1 or ZF=1	Jump if Below or Equal
JC	CF=1	Jump if Carry
JCXZ	CX=0	Jump if CX Zero
JE	ZF=1	Jump if Equal
JG	ZF=0 and SF=OF	Jump if Greater (signed)
JGE	SF=OF	Jump if Greater or Equal (signed)
JL	SF != OF	Jump if Less (signed)
JLE	ZF=1 or SF != OF	Jump if Less or Equal (signed)
JMP	unconditional	Unconditional Jump
JNA	CF=1 or ZF=1	Jump if Not Above
JNAE	CF=1	Jump if Not Above or Equal
JNB	CF=0	Jump if Not Below
JNBE	CF=0 and ZF=0	Jump if Not Below or Equal
JNC	CF=0	Jump if Not Carry
JNE	ZF=0	Jump if Not Equal
JNG	ZF=1 or SF != OF	Jump if Not Greater (signed)

JNGE	SF != OF	Jump if Not Greater or Equal (signed)
JNL	SF=OF	Jump if Not Less (signed)
JNLE	ZF=0 and SF=OF	Jump if Not Less or Equal (signed)
JNO	OF=0	Jump if Not Overflow (signed)

Loop instructions:

1. LOOP: Loop until CX is zero and decrements CX
2. LOOPZ/LOOPE: Loops until CX is zero and ZF=1 and decrements CX
3. LOOPNZ/LOOPNE: LOOPZ/LOOPE: Loops until CX is zero and ZF=0 and decrements CX

PROBLEMS*:

1. Write and execute an assembly language program to calculate the factorial of YH (an 8-bit number). Store the result of factorial at memory location 03002+X H.
   For example, factorial of a number four is calculated as 4! = 4 x 3 x 2 x 1 = 24. The final result 24 should be stored at memory location 03002+X H.
2. Write an assembly language code to sum first YH integers using

   (a) Jump and compare instructions

   (b) Loop instructions.
3. Write an assembly language code to find the maximum of three numbers present in three different registers. The code should store the maximum number at memory location 03002 + XH.

*Note: Let X = digits of your roll number
Y= sum of digits of your roll number
For example (If your roll number is LXX-1234; Take **X=1234H** and **Y=**1+2+3+4=**10**)

Attach the codes of the above mentioned problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Write an assembly language program to sum first 10 even numbers and store the result at memory location 03002+XH.

Write the code in the space provided below:

EXPERIMENT 5

STRINGS IMPLEMENTATION IN EMU8086

OBJECTIVES

- To be able to write assembly languages programs for string manipulation in EMU8086
- To understand operations including storing strings in memory, loading strings from memory, comparing strings, and scanning strings

EQUIPMENT/TOOLS:

- EMU8086

The first three instructions mentioned in the table below are used to copy a block of bytes/words from one location in memory to another. The source is pointed to by DS:SI and the destination is pointed to by ES:DI.

Mnemonic	Description
MOVSB	Move byte DS:[(E)SI] to ES:[(E)DI]
MOVSW	Move word DS:[(E)SI] to ES:[(E)DI]
MOVSD	Move dword DS:[(E)SI] to ES:[(E)DI]
CMPS	compares two string in memory
SCAS	scans a string for an element
LODS	loads an element from a string
STOS	stores an element into a string

PROBLEMS:

1. Write and execute an assembly language program to:
    a. move your first name in data segment 03002+X H.
    b. copy the string saved in part a. to extra segment 03008+X H

*Note: Let X = digits of your roll number
If your roll number is LXX-1234; Take **X=1234H**

2. Write and execute an assembly language program to scan letter “a” in the string stored in question no.1. If the letter ”a” is present in the string save the hex value 61 in AX register else save 0. The program should also find the number of times letter “a” appears in the string. The result of the count should be saved in register BX.

    Attach the codes of the above mentioned problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Write an assembly language program to find the length of the string saved in problem 1 part a.

Write the code in the space provided below:

EXPERIMENT 6

INTRODUCTIOB TO MPLAB

OBJECTIVES

- To be able to write and execute programs using MPLAB software
- To be able to simulate the program on PROTEUS

EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS

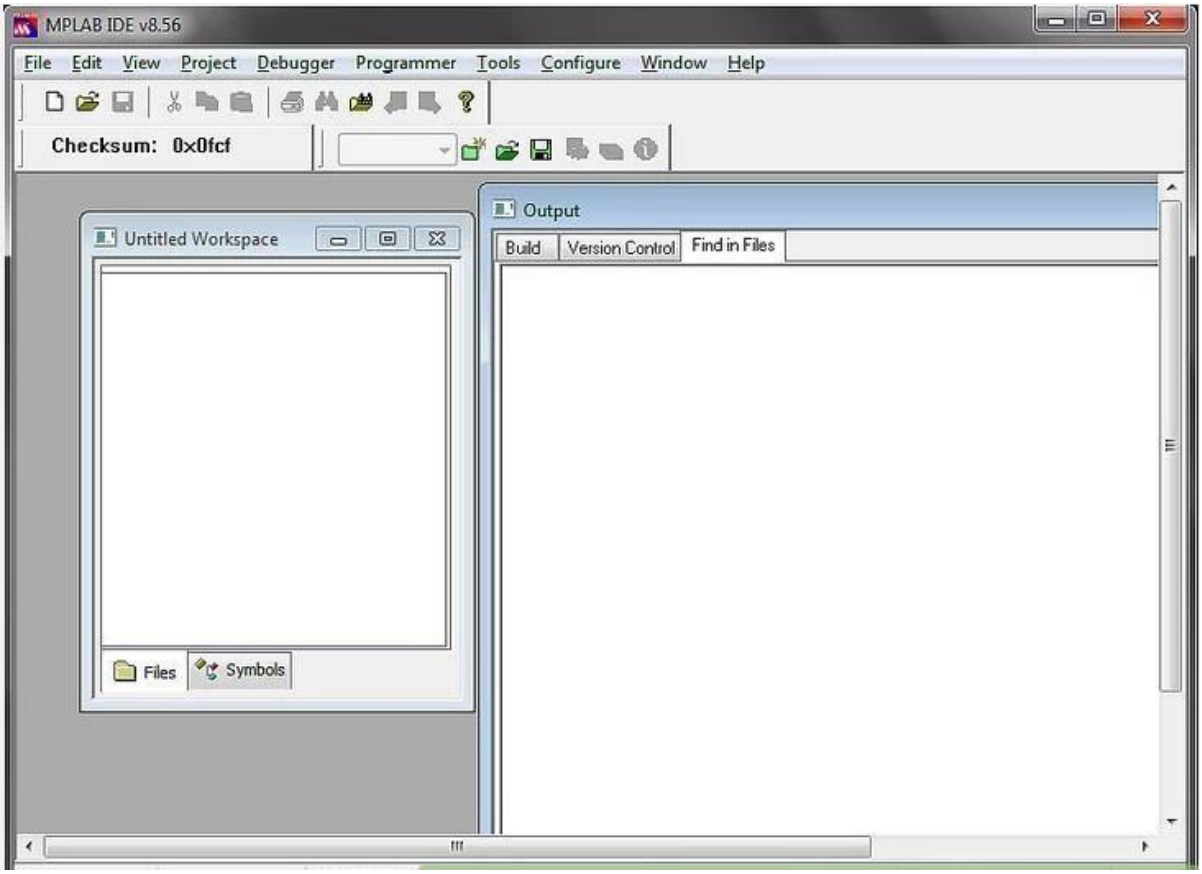
Template for Assembly language Code for PIC18F877

```
list      p=18f452     ; list directive to define processor
#include <p18f452.inc>     ; processor specific variable definitions

__CONFIG _CP_OFF & _WDT_ON & _BODEN_ON & _PWRTE_ON & _RC_OSC &
_WRT_ENABLE_ON & _LVP_ON & _DEBUG_OFF & _CPD_OFF

; '_CONFIG' directive is used to embed configuration data within .asm;
file.
; The lables following the directive are located in the respective; .inc
file.
; See respective data sheet for additional information on; configuration
word.

;***** VARIABLE DEFINITIONS
w_temp        EQU     0x70        ; variable used for context saving
status_temp   EQU     0x71        ; variable used for context saving

;**********************************************************************
            ORG     0x000         ; processor reset vector
            clrf    PCLATH        ; ensure page bits are cleared
            goto    main          ; go to beginning of program


            ORG     0x004        ; interrupt  vector  location
            Movwfw_temp       ; save off current W register contents
            movf  STATUS,w        ; move status register into W register
            movwf status_temp     ; save off contents of STATUS register

; isr code can go here or be located as a call subroutine elsewhere

Movfstatus_temp,w     ; retrieve copy of STATUS register
movwf STATUS            ; restore pre-isr STATUS register contents
swapfw_temp,f
swapfw_temp,w          ; restore pre-isr W register contents
retfie                    ; return from interrupt

main
```

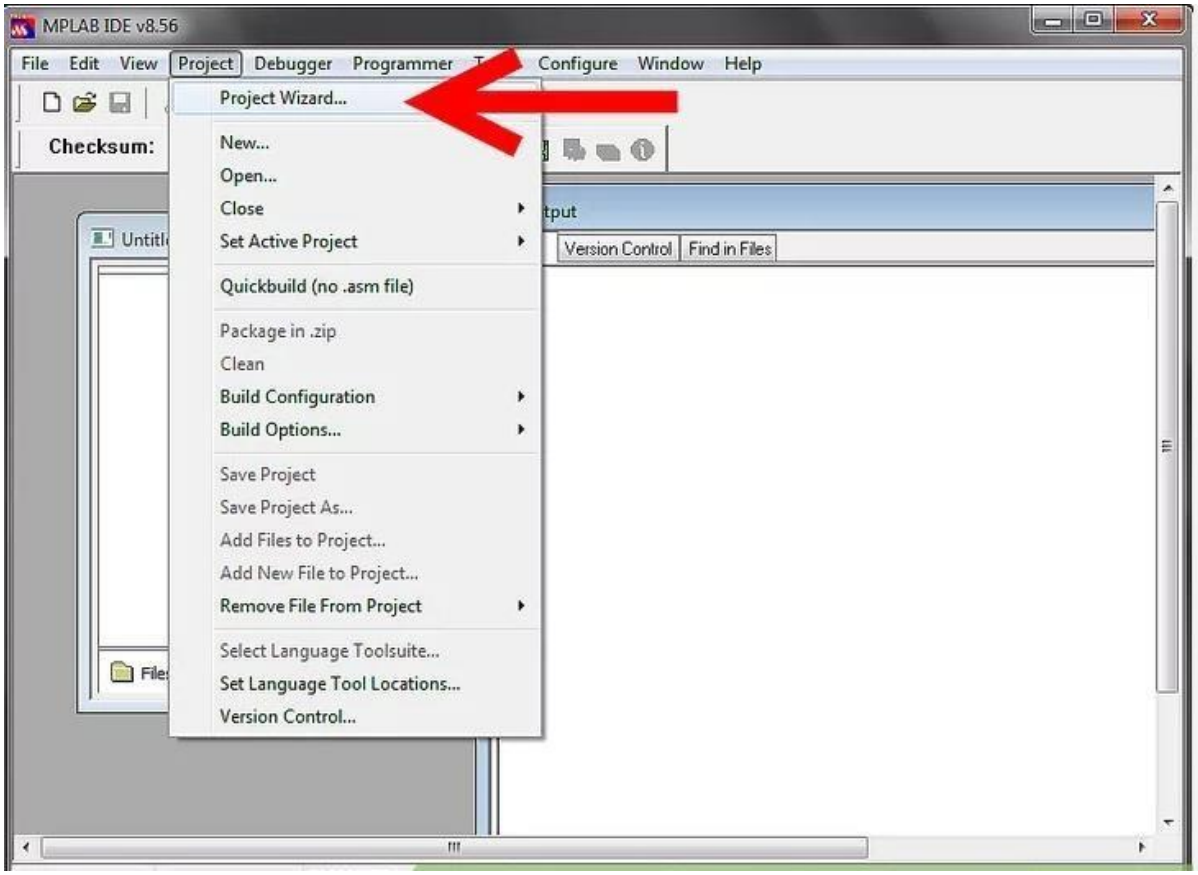
```
; remaining code goes here

END                          ; directive 'end of program'
```

Creating a New Project using MPLAB:

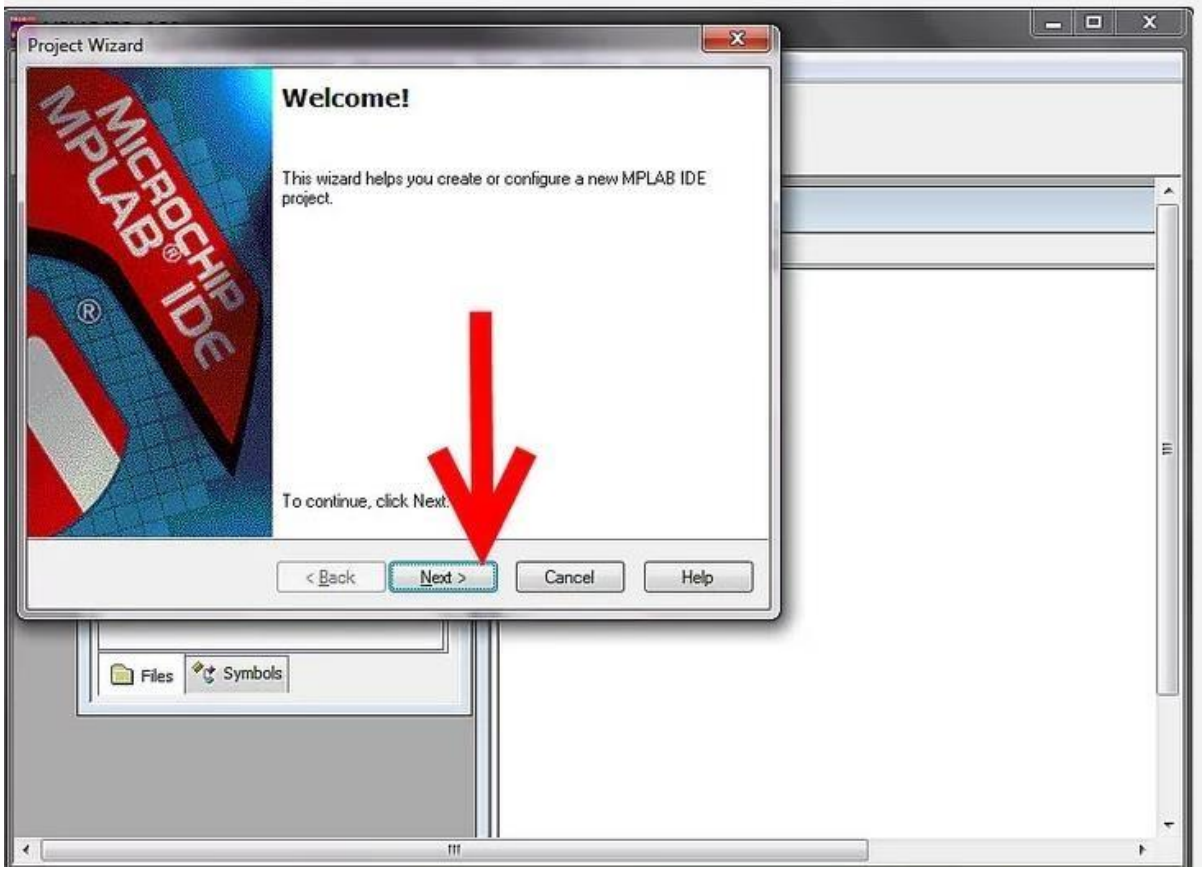
1. Launch the MPLAB, once loaded it should look like the this:



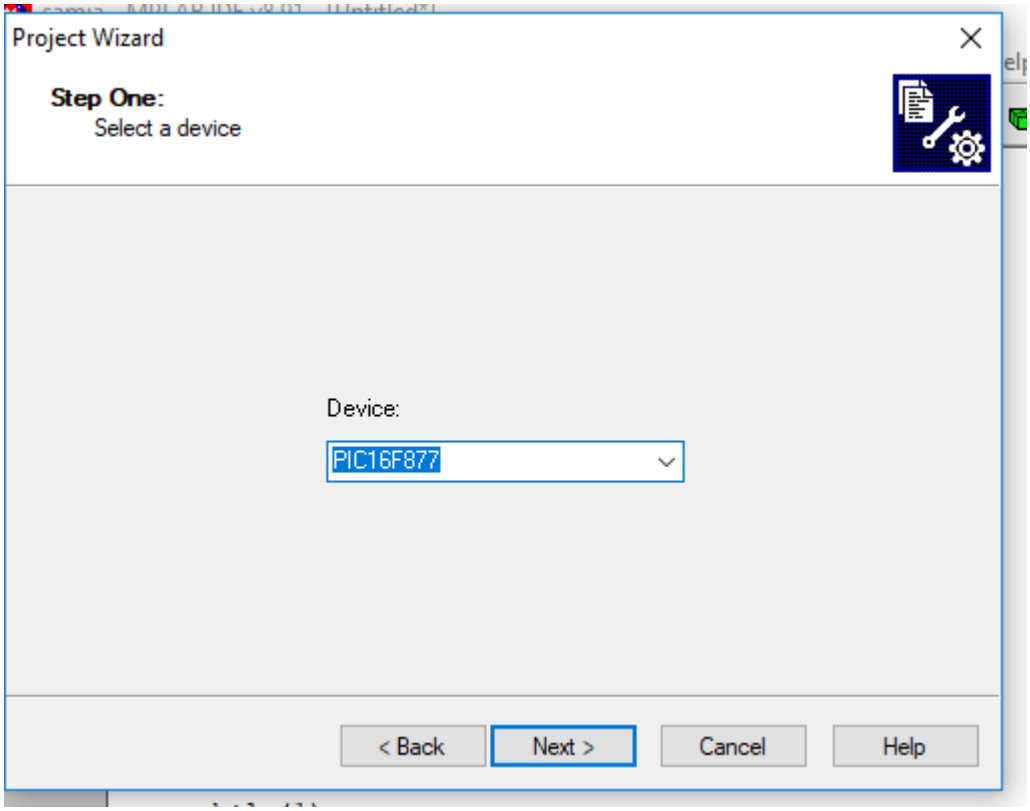
2. Click “Project” tab and select “Project Wizard”.



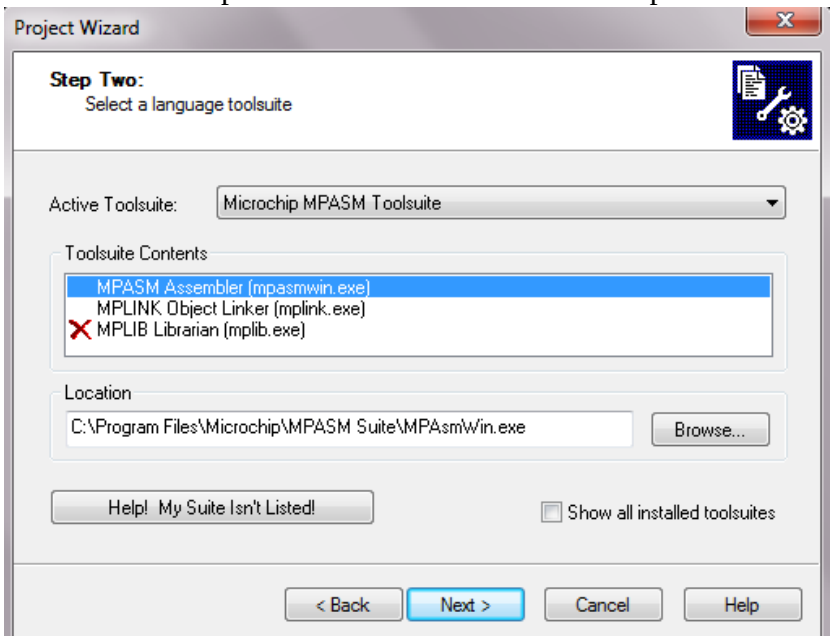
3. Click “Next” and select PIC16F877 controller, click “Next”.



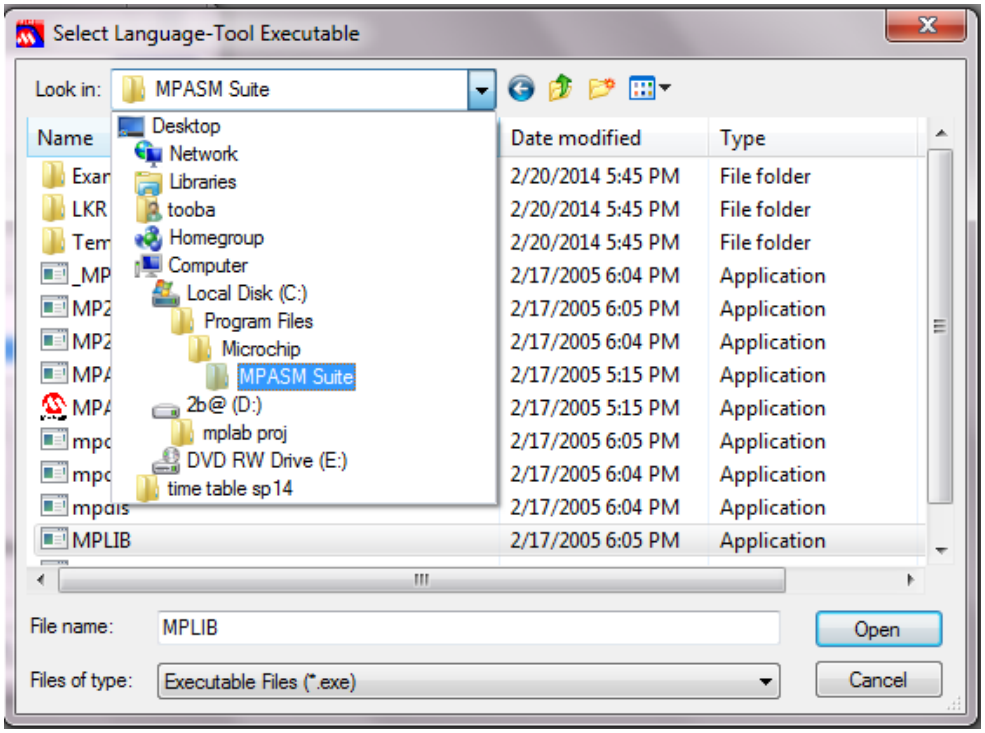
4. Select Device “PIC 16F877”

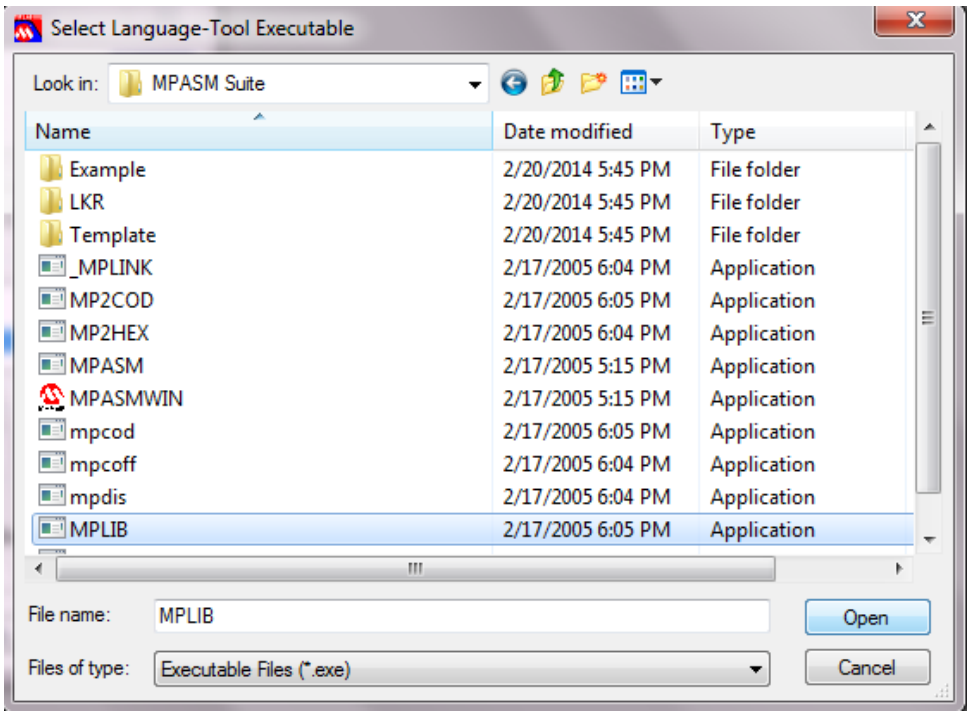


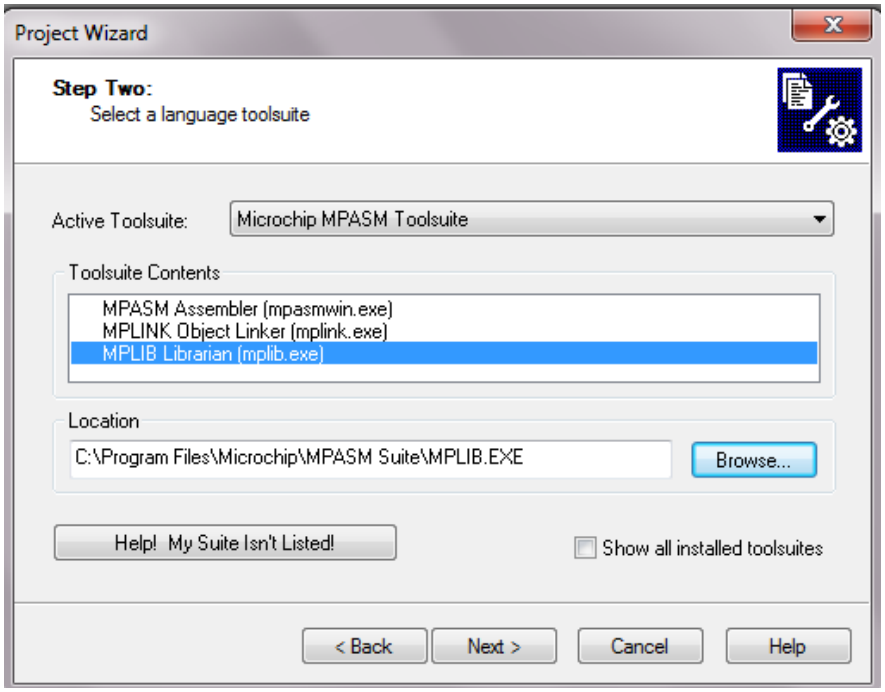
1. In the "Active Toolsuite" pull-down menu select "Microchip MPASM Toolsuite".



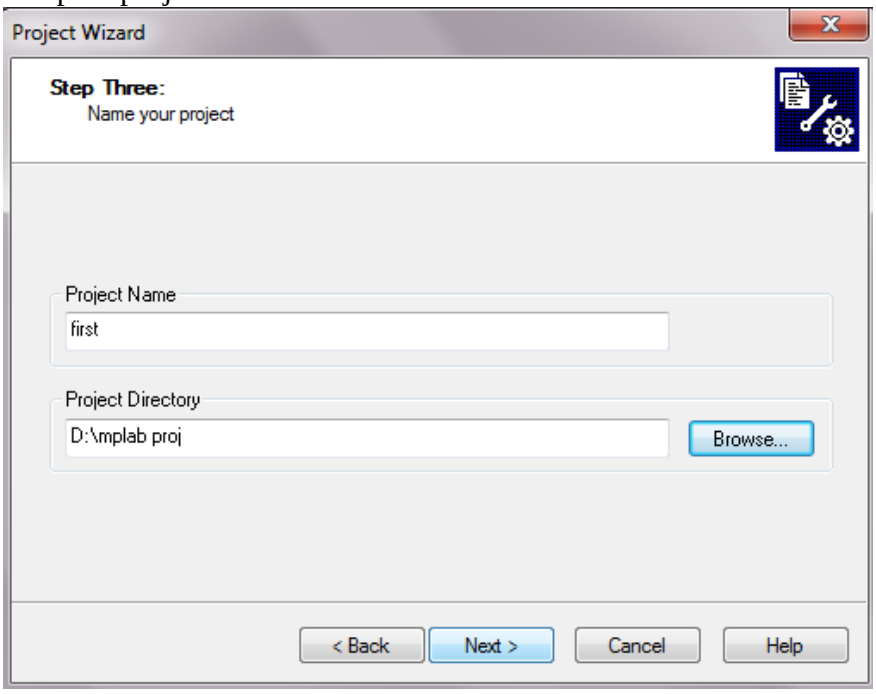
2. If you see a cross against tool suit content, you need to give the locations of the "Tool suite Contents". Select that tool suit and click on browse. Normally the path for the tool suit is: C:\Program Files\Microchip\MPASM Suite


Select Language-Tool Executable
Look in: MPASM Suite
Desktop
Network
Libraries
tooba
Homegroup
Computer
Local Disk (C:)
Program Files
Microchip
MPASM Suite
2b@ (D:)
mplab proj
DVD RW Drive (E:)
time table sp14
Name
Date modified
Type
2/20/2014 5:45 PM File folder
2/17/2005 6:04 PM Application
2/17/2005 6:05 PM Application
2/17/2005 5:15 PM Application
MPLIB
File name: MPLIB
Open
Files of type: Executable Files (*.exe)
Cancel


Select Language-Tool Executable
Look in: MPASM Suite
Name
Date modified
Type
Example 2/20/2014 5:45 PM File folder
LKR 2/20/2014 5:45 PM File folder
Template 2/20/2014 5:45 PM File folder
_MPLINK 2/17/2005 6:04 PM Application
MP2COD 2/17/2005 6:05 PM Application
MP2HEX 2/17/2005 6:04 PM Application
MPASM 2/17/2005 5:15 PM Application
MPASMWIN 2/17/2005 5:15 PM Application
mpcod 2/17/2005 6:05 PM Application
mpcoff 2/17/2005 6:04 PM Application
mpdis 2/17/2005 6:04 PM Application
MPLIB 2/17/2005 6:05 PM Application
File name: MPLIB
Open
Files of type: Executable Files (*.exe)
Cancel

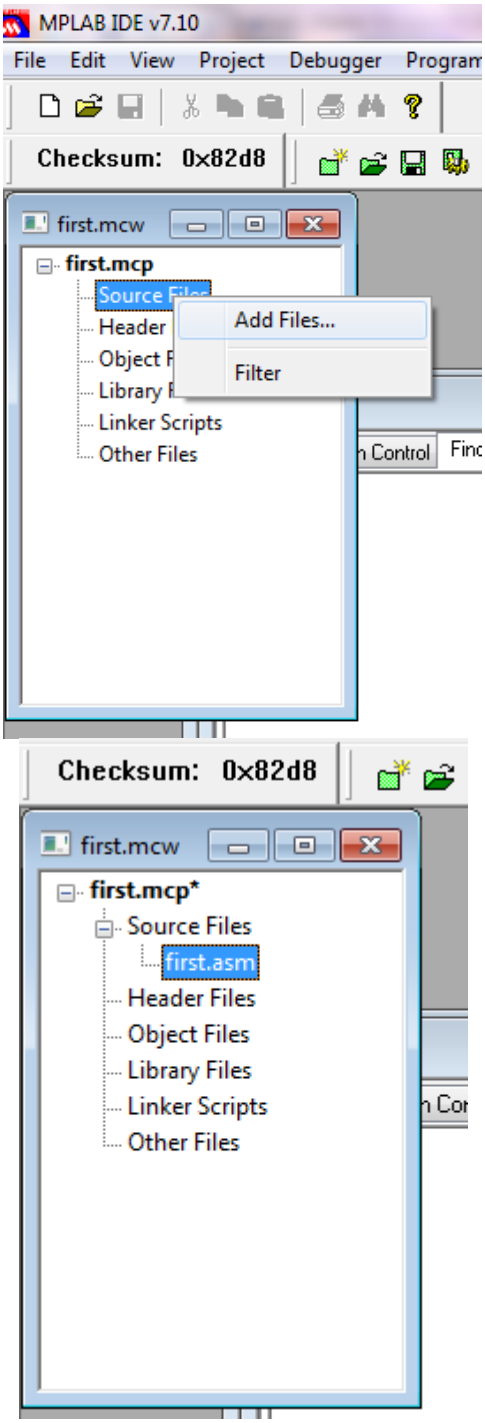


3. Once the tool suit is selected for compilation, Click Next
4. Enter project name and the directory in which you want to save the project. Remember that this directory path should not be longer than 60 characters. For example, you may save the project in D:\mplab proj\



5. After setting the directory click on next. In the next pop up window, you may add any existing code file in your project. In case you don't have any, you may just skip this step by clicking on next and then click on finish.
6. From file menu, create a new file
7. Write your code and save it.

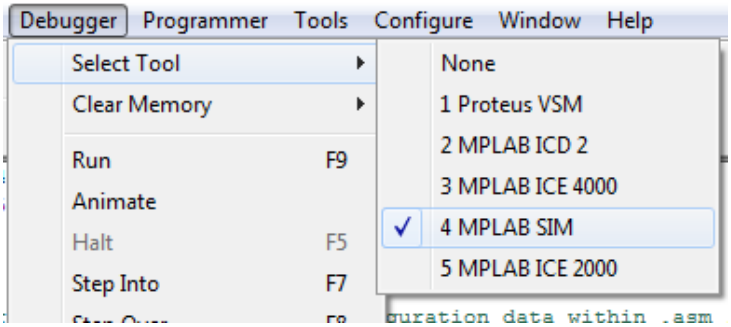
8. Project->add files to project->select your file and click open.



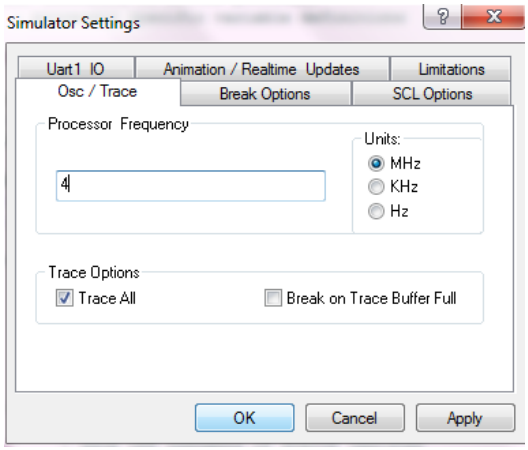
9. Project -> Make
   If the source files do not contain any errors the project will be assembled successfully.

Using MPLAB SIM

1. Debugger -> Select Tool -> MPLAB SIM



2. Debugger -> Settings... In the Simulator Settings pop-up window make sure that the processor frequency is set to 4MHz.



Different debugging options: 

- **Processor Reset**

  Debugger -> Reset -> Processor Reset 

- **Step Into** 

  Debugger->Step Into (for step wise debugging) or press F7
  It will also allow you to enter into any subroutine.

- **Step Over** 

  Debugger -> Step Over
  Will quickly run a subroutine and step to the next instruction following the subroutine call.

- **Step Out** 

  Debugger -> Step out
  It lets the control to come out of a subroutine and execute the instruction just after the subroutine call

- **Run** 

Debugger -> Run
This will run your program at high speed, so fast that you will not see what is going on.

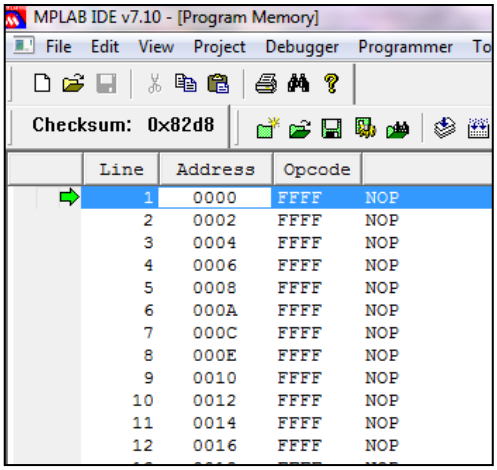
- **Halt**
  Debugger -> Halt
  How to stop the simulator after running or animating.

- **Animate**
  Debugger -> Animate
  A slow execution that will go pretty fast.

Viewing the memory

A. Program Memory

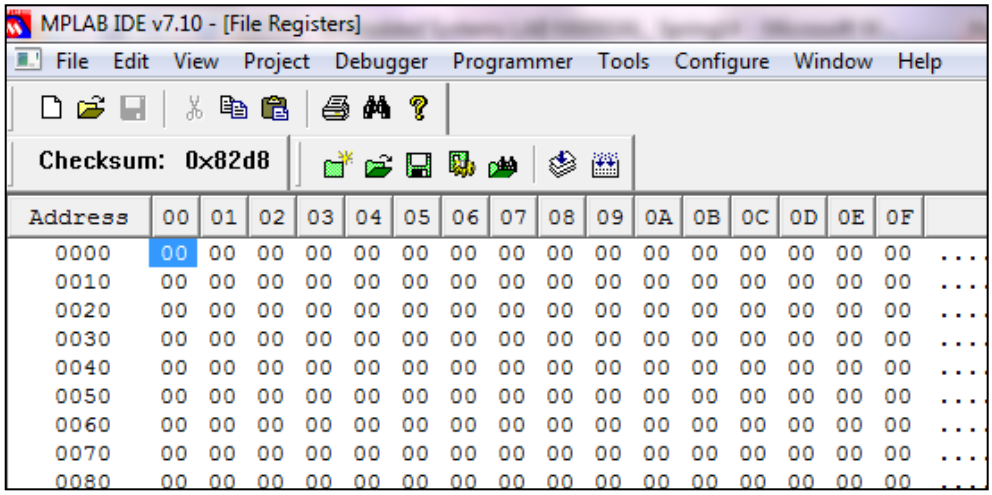
We can see the content of the program memory using following method:
View -> Program Memory



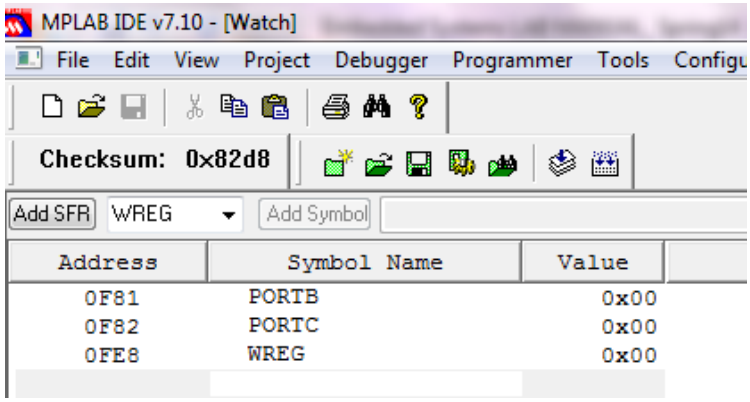
B. Data Memory

There are two methods of viewing the contents of registers.

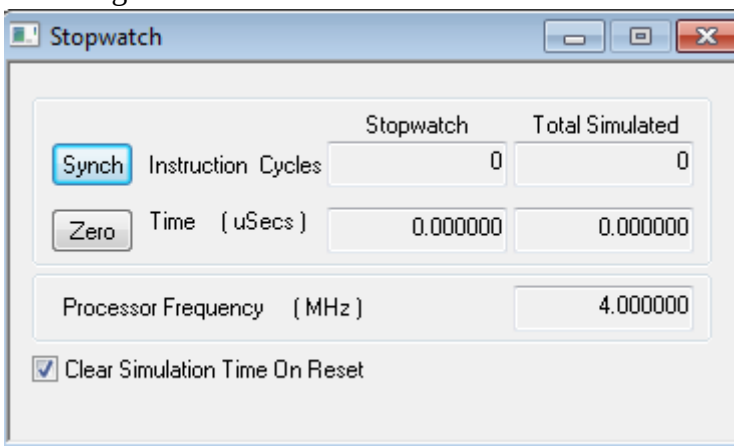
Method – 1 File Registers
File registers menu shows contents address wise in memory
View -> File Registers



Method – 2Watch window
View -> Watch
Select "PORTB" from the left pull down menu and then click on Add Symbol. Similarly add PORTC and WREG.



Using Stopwatch
Debugger -> Stopwatch displays a stopwatch which will give the actual time that is passing as instructions are executing.

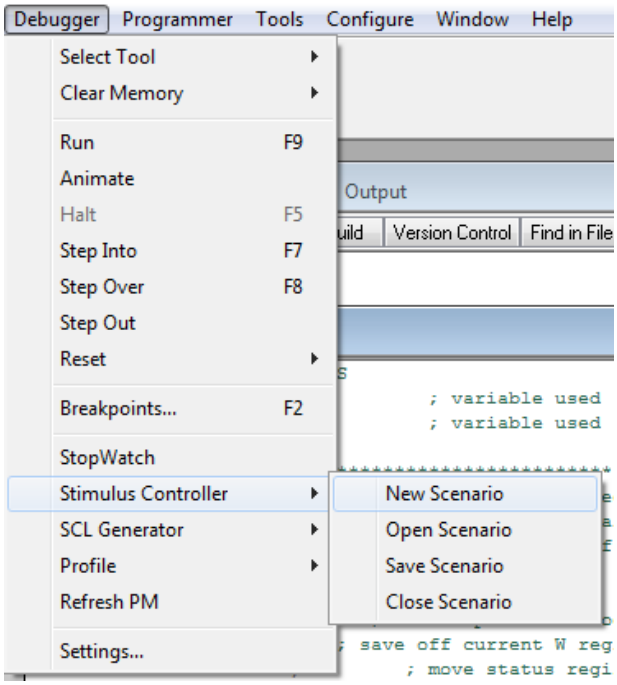


Double click on a line of code and a red "B" will appear in the margin in front of the line of code. This "B' stands for break point.

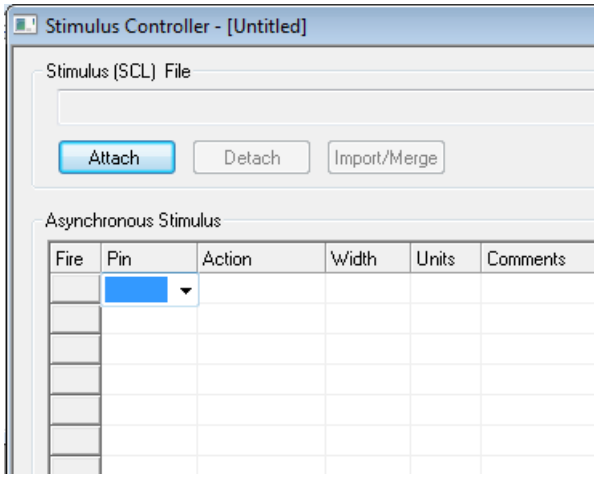
```
ORG     0x004          ; interrupt vector l
movwfw_temp          ; save off current W re
movf    STATUS,w          ; move status reg
movwf   status_temp       ; save off conten
```

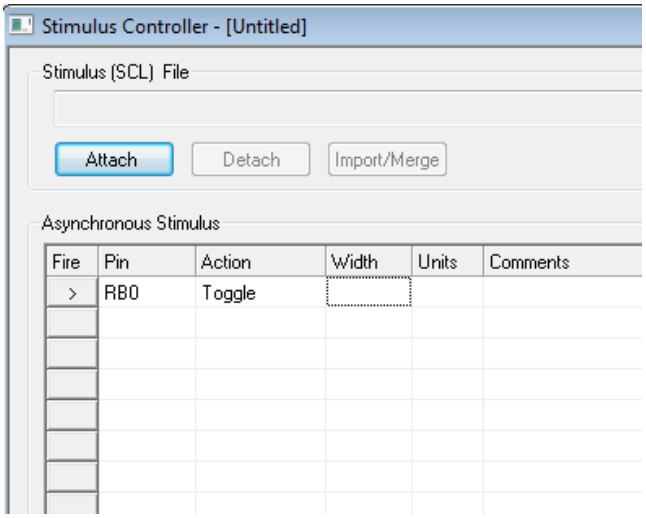
Setting input values to input ports

1. Debugger->Stimulus Controller->New Scenario



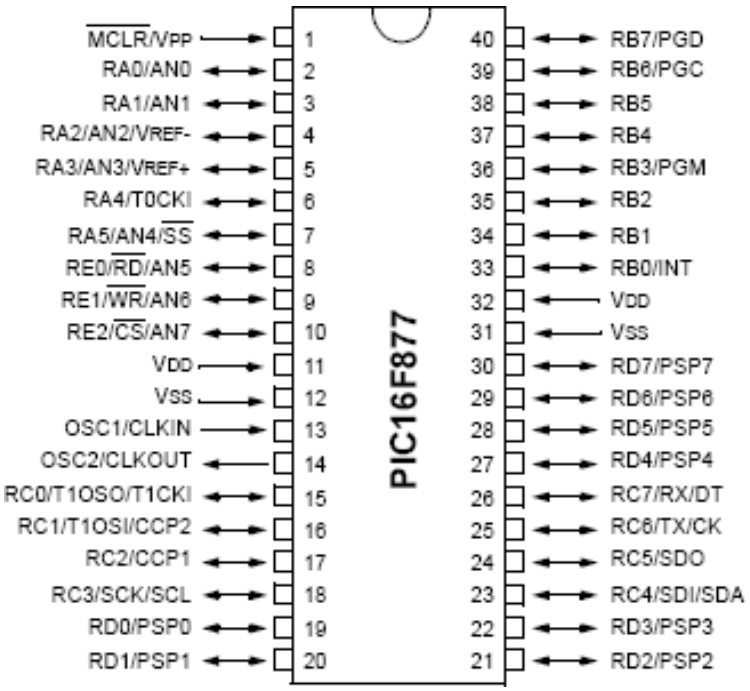
2. Click on the cell under Pin Heading. From drop down menu, select RB0. Similarly select all the pins of PORTB. (RB1,RB2,…RB7)





3. Under "Action" heading, you may select the required action. Select toggle.
4. Save scenario by clicking on save, write the name of scenario file and click ok.
5. Now by clicking on the fire button you can toggle the values of PORTB pins.
6. Animate your code and toggle during animation, you will see in changing bits in your watch window.

PIC16F877 Pin configuration



2.3 *PIC16F877* Pin description

Following table gives a detailed overview of each pin's functionality.

Pin Name	Pin #	Buffer Type	Description
OSC1/CLKIN	13	ST/CMOS	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	14	-------	Oscillator crystal output. Connects to crystal or resonator In crystal oscillator mode. In RC mode, OSC2 pin outputs CLKOUT which has 1/4 the frequency of OSC1, and denotes the instruction cycle rate.

Pin Name	Pin #	Buffer Type	Description
$\overline{\text{MCLR}}$/VPP	1	ST	Master Clear (Reset) input or programming voltage input. This pin is an active low RESET to the device.
RA0/AN0	2	TTL	RA0 is a bi-directional I/O pin. RA0 can also be analog input0.
RA1/AN1	3	TTL	RA1 is a bi-directional I/O pin. RA1 can also be analog input1.
RA2/AN2/VREF-	4	TTL	RA2 is a bi-directional I/O pin. RA2 can also be analog Input2 or negative analog reference voltage.
RA3/AN3/VREF+	5	TTL	RA3 is a bi-directional I/O pin. RA3 can also be analog Input3 or positive analog reference voltage.
RA4/T0CKI	6	ST	RA4 is a bi-directional I/O pin. RA4 can also be the clock Input to the Timer0 timer/counter. Output is open drain type.
RA5/$\overline{\text{SS}}$/AN4	7	TTL	RA5 is a bi-directional I/O pin. RA5 can also be analog input4 or the slave select for the synchronous serial port.
RB0/INT	33	TTL/ST	RB0 is a bi-directional I/O pin. RB0 can also be the external Interrupt pin.
RB1	34	TTL	RB1 is a bi-directional I/O pin. RB1 is a bi-directional I/O pin.
RB2	35	TTL	RB2 is a bi-directional I/O pin. RB2 is a bi-directional I/O pin.
RB3/PGM	36	TTL	RB3 is a bi-directional I/O pin. RB3 can also be the low Voltage programming input.
RB4	37	TTL	RB4 is a bi-directional I/O pin. Interrupt-on-change pin.
RB5	38	TTL	RB5 is a bi-directional I/O pin. Interrupt-on-change pin.
RB6/PGC	39	TTL/ST	RB6 is a bi-directional I/O pin. Interrupt-on-change pin or In-Circuit Debugger pin. Serial programming clock.
RB7/PGD	40	TTL/ST	RB7 is a bi-directional I/O pin. Interrupt-on-change pin or

			In-Circuit Debugger pin. Serial programming data.
Pin Name	**Pin #**	**Buffer Type**	**Description**
RC0/T1OSO/T1CKI	15	ST	RC0 is a bi-directional I/O pin. RC0 can also be the Timer1 Oscillator output or a Timer1 clock input.
RC1/T1OSI/CCP2	16	ST	RC1 is a bi-directional I/O pin. RC1 can also be the Timer1 oscillator input or Capture2 input/Compare2 output/PWM2 Output.
RC2/CCP1	17	ST	RC2 is a bi-directional I/O pin. RC2 can also be the Capture1 Input / Compare1 output/PWM1 output.
RC3/SCK/SCL	18	ST	RC3 is a bi-directional I/O pin. RC3 can also be the synchronous serial clock input/ output for both SPI and I2C Modes.
RC4/SDI/SDA	23	ST	RC4 is a bi-directional I/O pin. RC4 can also be the SPI Data In (SPI mode) or data I/O (I2C mode).
RC5/SDO	24	ST	RC5 is a bi-directional I/O pin. RC5 can also be the SPI Data Out (SPI mode).
RC6/TX/CK	25	ST	RC6 is a bi-directional I/O pin. RC6 can also be the USART Asynchronous Transmit or Synchronous Clock.
RC7/RX/DT	26	ST	RC7 is a bi-directional I/O pin. RC7 can also be the USART Asynchronous Receive or Synchronous Data.
RD0/PSP0	19	ST/TTL	bi-directional I/O
RD1/PSP1	20	ST/TTL	bi-directional I/O
RD2/PSP2	21	ST/TTL	bi-directional I/O
RD3/PSP3	22	ST/TTL	bi-directional I/O

RD4/PSP4	27	ST/TTL	bi-directional I/O
RD5/PSP5	28	ST/TTL	bi-directional I/O
RD6/PSP6	29	ST/TTL	bi-directional I/O
RD7/PSP7	30	ST/TTL	bi-directional I/O
VSS	12,31	ST/TTL	Ground reference for logic and I/O pins.
VDD	11,32	ST/TTL	Positive supply for logic and I/O pins.

PROBLEMS:

1. Write and execute an assembly language program for LED blinking using 4Mhz Oscillator frequency. Connect LED to pin RB1. The LED should continue to blink forever. Check the simulation of your program on PROTEUS.

Write the code in the space provided below:

POST LAB:

1. Write and execute an assembly language program to read 8 bit data (use switches to give data) from PORT B and display it on PORT C. Implement the circuit on PROTEUS.

Write the code in the space provided below and attach the screen shot of PROTEUS design

EXPERIMENT 7

DATA TRANSFER INSTRUCTIONS AND INTERFACING EXTERNAL COMPONENTS USING IO PORTS

OBJECTIVES

- To be able to write and execute programs using data transfer instructions and analyze memory contents in MPLAB software.
- To be able to interface external circuitry using IO ports of microcontroller.
- To be able to interface circuits using Optocoupler and Relays

EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS
- RELAYS, OPTOCOUPLERS

PIC 16F877 INSTRUCTION SET:

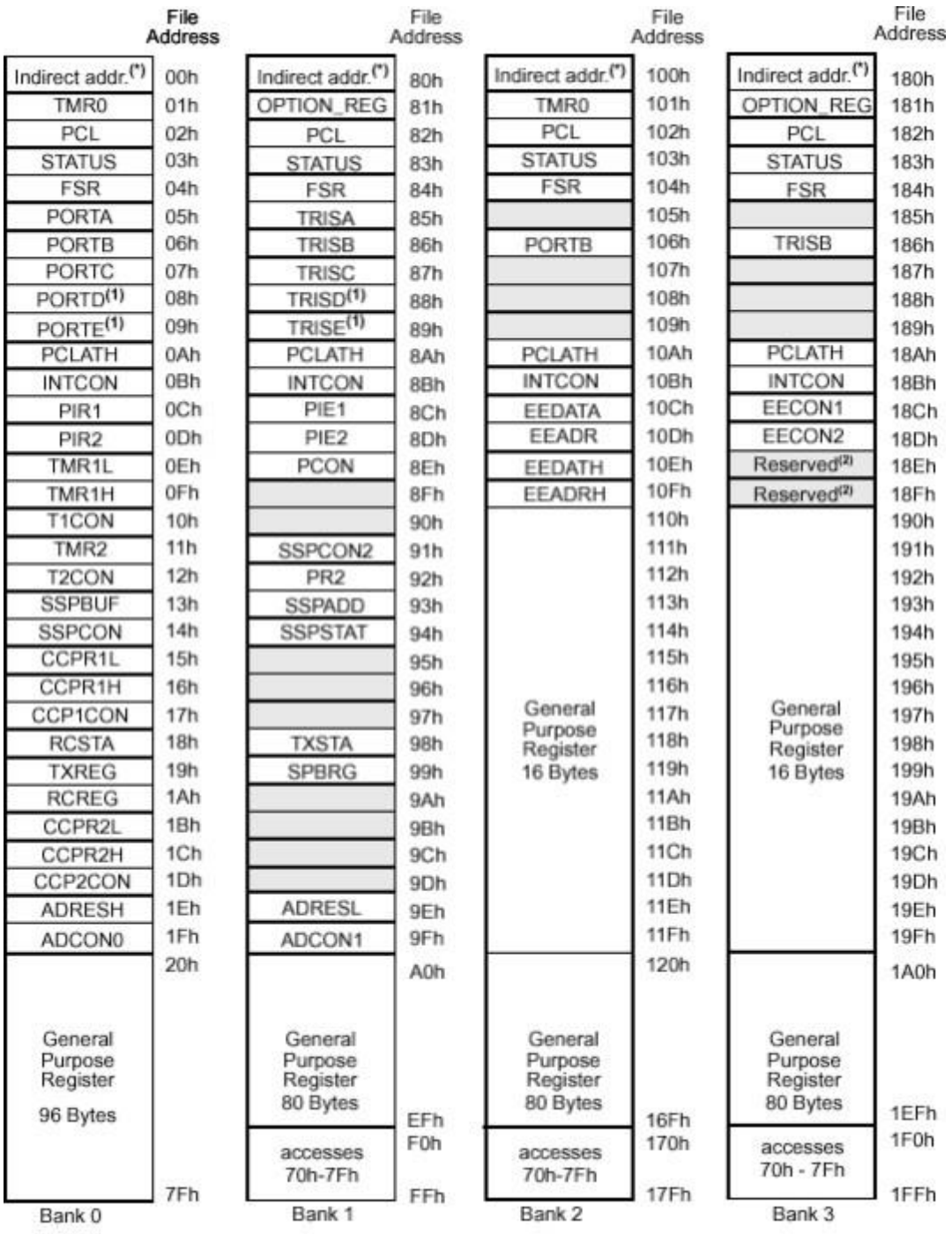
Mnemonic Operands	Description
MOVF f,d	Copy/move the contents (bits) in the flag register to D register
MOVWF f	Move the data (bits) from W register to flag register F
MOVLW k	Write constant in W register (move the value from literal to W register)
CLRW	Clearing instruction that helps to reset the values of W register to '0' (write '0' in W register).
CLRF f	Write '0' in F register that helps to reset the current status to '0'
SWAPF f,d	Swap nibbles in f

Data Memory (RAM):

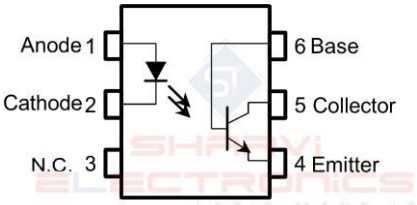
The data memory in RAM is partitioned into multiple banks (or pages) which contain the General Purpose Registers (GPR) and the Special Function Registers (SFR).
The SFRs are already internally defined by the system so users are not allowed to override these registers.
The GPRs are actually empty spaces to be used to store data or variables. In other words, when you define a variable like, x, y, or temp, then you have to allocate a space (a byte size) for each in the GPRs in RAM area.
To access registers properly, the bank location of them must be selected. In other words, you have keep track which bank of registers the CPU currently accesses. If the current bank access is Bank0, and your next register is located in Bank1, you have to select the Bank1 before accessing the register.

Accessed Bank	(RP1:RP0)
0	0 0
1	0 1

2	1 0
3	1 1

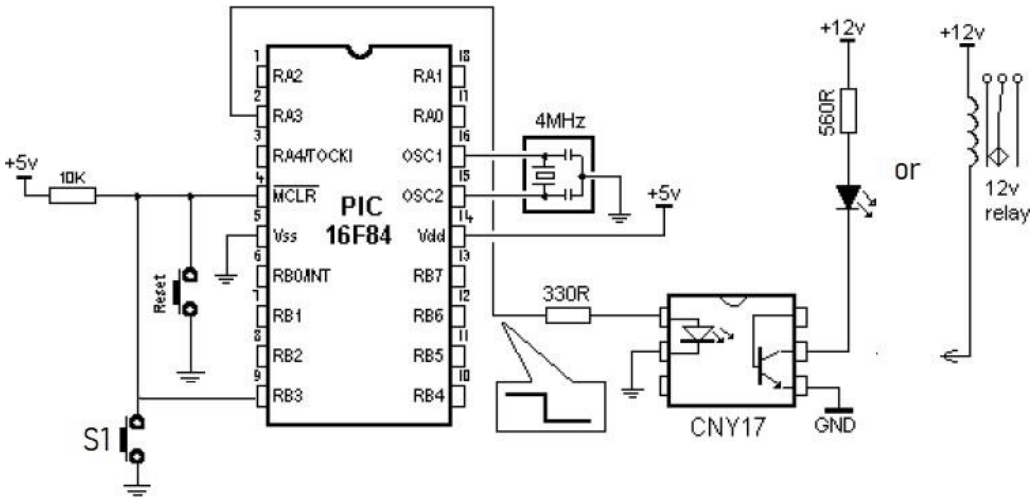


CNY17: The CNY17 is an optically coupled pair consisting of a gallium arsenide infrared emitting diode optically coupled to a silicon NPN phototransitor. The CNY17 can be used to replace relays and transformers in many digital interface applications.



PROBLEMS:

1. Write and execute an assembly language program that stores first four digits of your "Roll_Number" to file registers location 0x11, 0x21, 0x31 and 0x41. Use "MPLAB SIM" to debug the code and check the output in the file registers using watch window.
2. Write and execute an assembly language program that takes the last two digits of your "Roll Number" from PORT C (as 8-bit value) and display the input data on PORT B using MPLAB. Implement the program on PROTEUS using PIC16F877 microcontroller by connecting switches and LEDs to the respective ports.
    - Note: If your roll number is **4381**, The input data on PORT A should be 1010001 (binary equivalent of 81).
3. Write an assembly code to control the output led through switch S1using MPLAB and implement it using proteus.

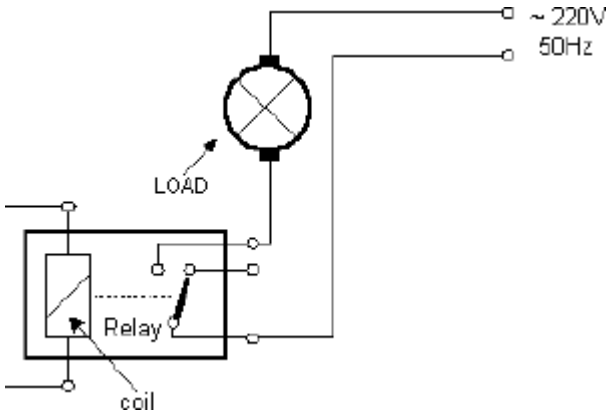


Attach the codes of the problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

Attach the screen shot of the PROTEUS design for the problems, in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Redo In lab Part 3 by replacing the optocoupler module by Relay shown below and connect a 220V light bulb or Motor in place of LED. (Relay used should be triggered by 5V otherwise 12V relay should need additional trigger circuit.)



Write the code in the space provided below and attach the screen shot of PROTEUS design

EXPERIMENT 8

ARITHMETIC AND LOGIC INSTRUCTIONS AND USE OF EXTERNAL INTERRUPT in PIC16F877

OBJECTIVES

- To be able to write and execute programs using arithmetic and logic instructions and analyze memory contents in MPLAB software
- To be able to use an external interrupt and understand its functioning in PIC microcontroller
- Advancing the Simulation skills using Proteus as a simulator
- To be able to interface circuits using relays

EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS
- RELAYS

PIC 16F877 INSTRUCTION SET:

Mnemonic Operands	Description
ADDLW k	Given constant is added with W register
ADDWF f, d	Add W register content with f register
SUBLW k	W register content is subtracted from k result is stored in W register
SUBWF f	W register content is subtracted from f
INCF f, d	Increment the content of f register, if d=0 result is stored in W register else if d=1 result is stored in f register
DECF f, d	Decrement the content of f register
ANDLW k	Given constant is AND with W register result is stored in W register
IORLW k	Given constant is OR with W register and result is stored in W register
IORWF f, d	W register is OR with f register
XORLW k	Given constant is XOR with W register and result is stored in W register
XORWF f, d	W register is XOR with f register
BCF f, b	Bit Clear f
BSF f, b	Bit set f

External Interrupt:

Interrupts are special events that requires immediate attention, it stops a microcontroller from the running task and to serve a special task known as Interrupt Service Routine (ISR) or Interrupt Handler. In PIC16F877 INT Pin Interrupt (external interrupt) is one of the interrupt sources for each peripheral module. Additionally, if the device has peripheral interrupts, then

it will have registers to enable the peripheral interrupts and registers to hold the interrupt flag bits. These registers are named as PIR and PIE.
External Interrupt on the RBO/INT pin is edge triggered. The rising and falling edge is determined by the INTDEG bit from another register known as OPTION_REG. If INTDEG is "1" interrupt occurs on rising edge of RBO/INT pin else it occurs on falling edge of RBO/INT pin.

INTCON REGISTER (ADDRESS 0Bh, 8Bh, 10Bh, 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF
bit 7							bit 0
1	0	0	1	0	0	0	0

bit 7 **GIE:** Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 4 **INTE:** RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

PROBLEMS:

1. Write and execute an assembly language program that takes two 8 bit numbers from user through PORT C and store the numbers at memory locations 0x70 and 0x71.Your program should perform the following functions:
   i. Add the two numbers and store the result at memory location 0xX H.
   ii. Subtract the smaller number from the larger number and store the result at memory location 0xX+2H+2.
   iii. Take number stored at memory location 0x70. Use AND to mask this number such that upper nibble is stored at memory location 0xX+4H+4 and the lower nibble is stored at memory location 0xX+8H+8.

*Note: Let X = second last digit of your roll number

Y= last of digit of your roll number

Use "MPLAB SIM" to debug the code and check the output in the file registers using watch window.

2. Write and execute an assembly language program that displays the last two digits of your "Roll Number" on PORT C. A switch is connected to the External Interrupt pin INT of the PIC Microcontroller. When this switch is pressed, the microcontroller is interrupted and the ISR is executed. The ISR should turn the fan ON (which is connected through a relay to pin 3 of port A).
   Note: Switch may require a pull up resistor to work properly.

Attach the codes of the problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Enable PORTB Change Interrupt which is used to identify the external Interrupts pins to RB4, RB5, RB6, and RB7. Whenever anyone of the RB4, RB5, RB6, RB7 state changes, Interrupt will happen. In ISR your program should toggle the LED connected to pin 0 of PORTD.

Write the code in the space provided below and attach the screen shot of PROTEUS design

EXPERIMENT 9

PIC TIMER CONFIGURATION TO GENERATE DELAYS AND COUNT EVENTS

OBJECTIVES

- To be able to write and execute programs using PIC16F877 timer interrupts in MPLAB software
- To be able to configure the PIC16f877 internal and external timers to generate delays.

EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS

PIC16F877 Timers:

Timers are used to measure the time or generate the accurate time delay. The timer is a simple binary counter that can be configured to count clock pulses (Internal/External). Once it reaches the max value, it will roll back to zero setting up an OverFlow flag and generates the interrupt if enabled. Timers can be used to time delay generation, measuring frequency of pulses, generating PWM signals and triggering external devices or peripherals. PIC16F877 has three timers.

- Timer0 (8-bit timer)
- Timer1 (16-bit timer)
- Timer2 (8-bit timer)

Timer 0:

Physically, timer is a register whose value is continually increasing to 255, and then it starts all over again: 0, 1, 2, 3, 4...255....0, 1, 2, 3. ....etc. The Timer0 module (timer/counter) has the following features:

• 8-bit timer/counter
• Readable and writable
• 3-bit software programmable prescaler (8-options)
• Internal or external clock select
• Interrupt on overflow from FFh to 00h
• Edge select for external clock

The Timer0 module's core register is **TMR0** which is an 8-Bit SFR register.

$$T_{out} = \frac{(4 * Prescaler * (256 - TMR0) * X)}{F_{osc}}$$

The control register for Timer0 is called OPTION_REG which is one of the core SRFs. Here is the brief functionality of each bit in this register. Prescaler is a name for the part of

microcontroller which divides oscillator clock before it will reach logic that increases timer status. The value of prescaler can be set using OPTION register.

OPTION_REG REGISTER

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{\text{RBPU}}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **$\overline{\text{RBPU}}$**

bit 6 **INTEDG**

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on T0CKI pin
0 = Internal instruction cycle clock (CLKO)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on T0CKI pin
0 = Increment on low-to-high transition on T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

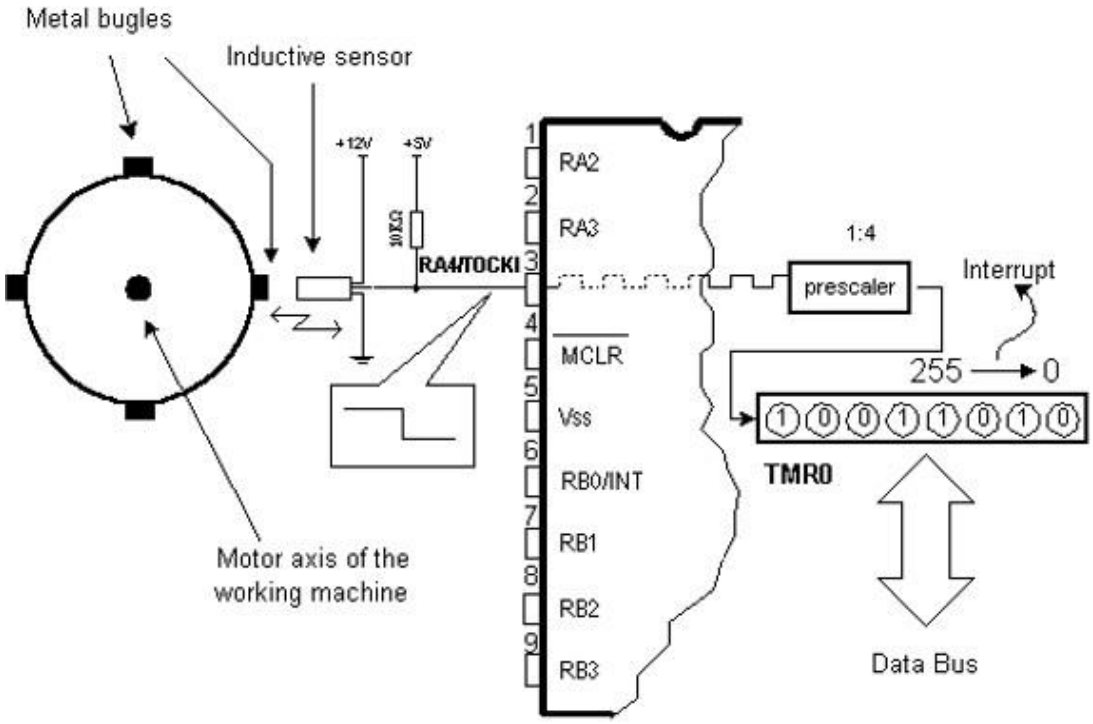
bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

The prescaler is selected by using the “PSA” bit in the “OPTION_REG” register. The “PSA” bit is cleared (=0) and the scale ratio is set as shown above:

PROBLEMS:

1. Write and execute an assembly language program that uses internal Timer0 to blink LED after every 2 seconds. The program should start blinking LED (which is driven through RC0 pin) when the push button (connected to Pin 3 of PORT D) is pressed.
2. Write and execute an assembly language program that calculates the stepper motor revolutions count. The system has four metal screws on the axis of a winder. These four screws represent metal convexity as shown in the figure below. Inductive sensor will generate the falling signal every time the head of the screw is parallel with sensor head. Each signal will represent one fourth of a full turn, and the sum of all full turns must be found in TMR0 timer. Display the revolutions count on PORT D.


Metal bugles
Inductive sensor
+12V
+5V
10KΩ
RA4/TOCKI
1
2
3
4
5
6
7
8
9
RA2
RA3
MCLR
Vss
RB0/INT
RB1
RB2
RB3
1:4
prescaler
Interrupt
255
0
1 0 0 1 1 0 1 0
TMR0
Data Bus
Motor axis of the
working machine

Attach the codes of the problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Write and execute an assembly language program that creates a delay of "X" seconds. Where "X" is the last digit of your roll number. Load "X" in TMR0 and after delay of "X" seconds display the value of TMR0 on PORT D. Continue to run the program till the push button (connected to RC5) is pressed.

Write the code in the space provided below and attach the screen shot of PROTEUS design

EXPERIMENT 10

PIC TIMER INTERRUPT, DEBOUCING OF SWITCHES AND KEYPAD INTERFACING

OBJECTIVES

- To be able to write and execute programs using PIC16F877 timer overflow interrupts in MPLAB software
- To understand the phenomena of bouncing in mechanical switches and its solution
- To be able to use mechanical switches in making a simple application
- To learn scanning of a simple keypad for input

EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS

BACKGROUND:

Push-button switches, toggle switches, and electro-mechanical relays all have one thing in common: contacts. It's the metal contacts that make and break the circuit and carry the current in switches and relays. Because they are metal, contacts have mass. And since at least one of the contacts is on a movable strip of metal, it has springiness. Since contacts are designed to open and close quickly, there is little resistance (damping) to their movement.

Because the moving contacts have mass and springiness with low damping they will be "bouncy" as they make and break. That is, when a normally open (N.O.) pair of contacts is closed, the contacts will come together and bounce off each other several times before finally coming to rest in a closed position. The effect is called "contact bounce" or, in a switch, "switch bounce" as shown in figure below. Note that contacts can bounce on opening as well as on closing.

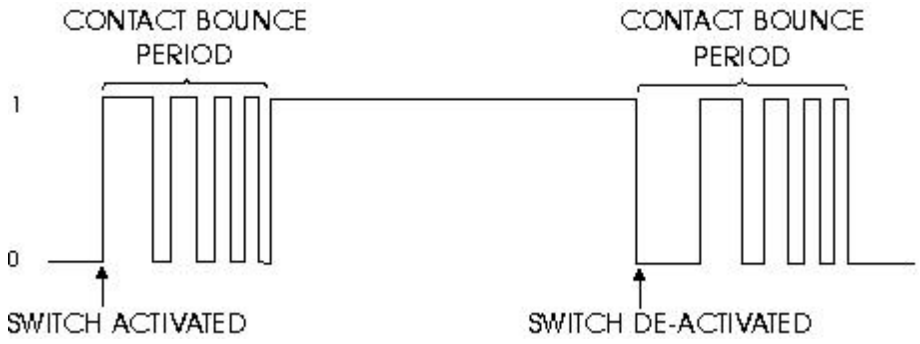


Fig. Bouncing of mechanical switches

If all you want your switch or relay to do is turn on a lamp or start a fan motor, then contact bounce is not a problem. But if you are using a switch or relay as input to a digital counter, a personal computer, or a micro-processor based piece of equipment, then you must consider contact bounce. The reason for concern is that the time it takes for contacts to stop bouncing is measured in milliseconds. Digital circuits can respond in microseconds.

As an example, suppose you want to count widgets as they go by on a conveyor belt. You could set up a sensitive switch and a digital counter so that as the widgets go by they activate the switch and increment the counter. But what you might see is that the first widget produces a count of 47, the second widget causes a count of 113, and so forth. What's going on? The answer is you're not counting widgets; you're counting how many times the contacts bounced each time the switch is activated!

There are several ways to solve the problem of contact bounce (that is, to "de-bounce" the input signal). Often the easiest way is to simply get a piece of equipment that is designed to accept "bouncy" input. In the widget example above, you can buy special digital counters that are designed to accept switch input signals. They do the de-bouncing internally. If that is not an option, then you will have to do the debouncing yourself using either hardware or software.

DEBOUNCING USING SOFTWARE:

Usually, the switch or relay connected to the computer will generate an interrupt when the contacts are activated. The interrupt will cause a subroutine (interrupt service routine) to be called.
The idea is that as soon as the switch is activated the Debounce Routine (DR) is called. The DR calls another subroutine called DELAY which just kills time long enough to allow the contacts to stop bouncing. At that point the DR checks to see if the contacts are still activated (maybe the user kept a finger on the switch). If so, the DR waits for the contacts to clear. If the contacts are clear, DR calls DELAY one more time to allow for bounce on contact-release before finishing.

ENABLING AND DISABLING AN INTERRUPT FUNCTION:
All interrupt functions are disabled by default, meaning that non will be responded by the microcontroller if they are activated. The interrupts must be enabled by software in order for the microcontroller to respond to them. The INTCON register is responsible for enabling and disabling the interrupt function.

INTCON							
GIE	PEIE	TMR0	INT	RBIE	TMR0IF	INTF	RBIF

GIE: Global Interrupt Enable bit
PIE: Peripheral Interrupt Enable bit
TMR0IE: TMR0 Overflow Interrupt Enable bit
INTE: RB0/INT External Interrupt Enable bit
RBIE: RB Port Change Interrupt Enable bit
TMR0IF: TMR0 Overflow Interrupt Flag bit
INTF: RB0/INT External Interrupt Flag bit
RBIF: RB Port Change Interrupt Flag bit

PROBLEM:

1. Write and execute an assembly language program that uses timer overflow interrupt. Configure time to increment timer after 2 oscillator clocks. Timer should start from the value equivalent to last two digits of your "Roll_Number". The program should send last two digits of your "Roll_Number" to PORT C when timer overflow is detected. Calculate the time consumed. Swap the last two digits of your "Roll_Number" and display the result on PORT C with the delay calculated.

Attach the code in the space provided below:

POST LAB:

1. Interface a 4-bit keypad with PORTB of PIC16f877. Four switches should be connected to PORT B which will be used as an input to keypad. Write and execute an assembly language program that checks the status of keypad and send it to a predefined register. In the main program, send the status to PORT A. Include bouncing effect in your program and use interrupt during keypad scanning.

Write the code in the space provided below and attach the screen shot of PROTEUS design

EXPERIMENT 11

OPEN ENDED LAB

Problem Statement

Design and implement a system to interface multiple 7-segment displays with a PIC16F877 microcontroller. Using appropriate techniques, the task is to control these displays to show desired numerical outputs.

EXPERIMENT 12

INTERFACING OF MISCELLANEOUS COMPONENTS TO PIC MICROCONTROLLER

OBJECTIVES

- To learn LCD interfacing with PIC16F877
- To be able to use shift registers to obtain the desired output

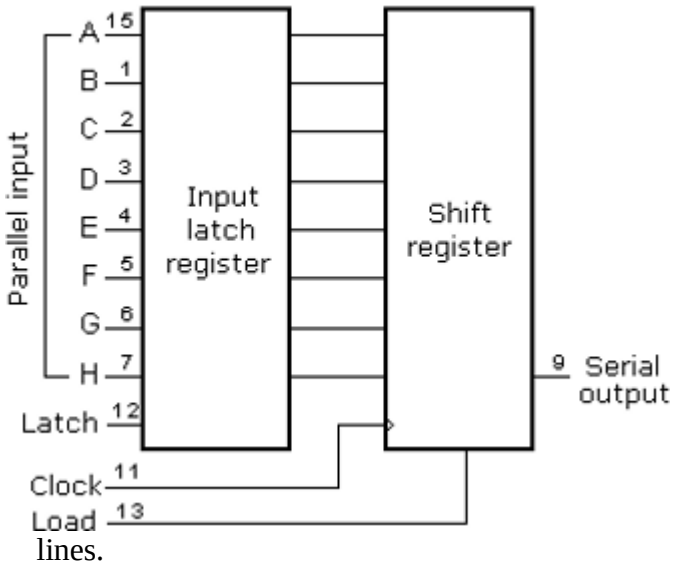
EQUIPMENT/TOOLS:

- MPLAB v8.92
- PROTEUS

BACKGROUND:

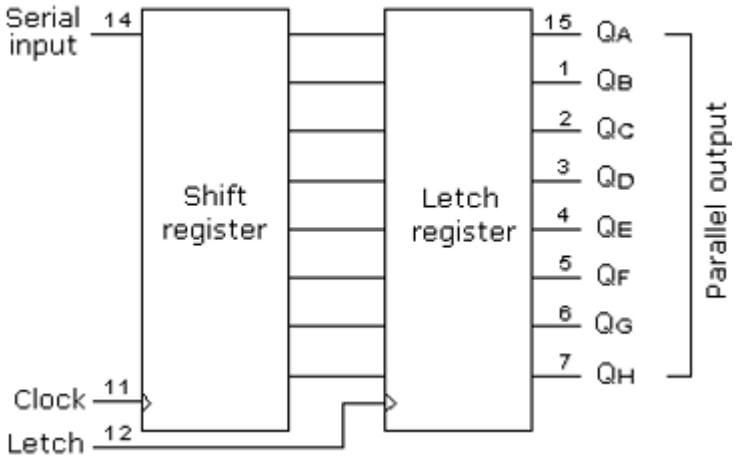
There are two types of shift registers: input and output. Input shift registers receive data in parallel, through 8 lines and then send it serially through two lines to a microcontroller. Output shift registers work in the opposite direction; they receive serial data and on a "latch" line signal, they turn it into parallel data. Shift registers are generally used to expand the number ofinput-output lines of a microcontroller.

Input shift register: Input shift registers transform parallel data into serial data and transfer itto a microcontroller. Their working is quite simple. There are four lines for the transfer of data:clock, latch, load and data. Data is first read from the input pins by an internal register through a 'latch' signal. Then, with a 'load' signal, data is transferred from the input latch register to the shift register, and from there it is serially transferred to a microcontroller via 'data' and 'clock' lines.



Output shift register: Output shift registers transform serial data into parallel data. On everyrising edge of the clock, the shift register reads the value from data line, stores it in temporary

register. On a signal from 'latch' line, data is copied from the shift register to input register,thus data is transformed from serial into parallel data.



LCD Display: Many microcontroller devices are using 'smart LCD' displays to output visual information. We can communicate to the display via an 8-bit data bus or 4-bit data bus. For a 8-bit data bus, the display requires a +5V supply plus 11 I/O lines. For a 4-bit data bus it only requires the supply lines plus seven extra lines. When the LCD display is not enabled, data lines are tri-state which means they are in a state of high impedance (as though they are disconnected) and this means they do not interfere with the operation of the microcontroller when the display is not being addressed.
The LCD also requires 3 "control" lines from the microcontroller.
The Enable (E) line allows access to the display through R/W and RS lines. When this line is low, the LCD is disabled and ignores signals from R/W and RS. When (E) line is high, the

	0 Access to LCD disabled
E	1 Access to LCD enabled
	0 Writing data to LCD
R/W	1 Reading data from LCD
	0 Instruction
RS	1 Character

Writing data to the LCD is done in several steps:

- Set R/W bit to low
- Set RS bit to logic 0 or 1 (instruction or character)
- Set data to data lines (if it is writing)
- Set E line to high
- Set E line to low Read data from data lines (if it is reading)

Reading data from the LCD is done in the same way, but control line R/W has to be high.

PROBLEMS:

1. Write and execute an assembly language program that displays the last two digits of your "Roll Number" using parallel input of the shift register and moves it serially to PORTC when enable bit is high.
2. Repeat Problem no. 1 using serial to parallel data transfer when enable bit is high.

Attach the codes of the problems in the space provided. Moreover, for each answer mention the problem no. at the top of the page.

POST LAB:

1. Write and execute an assembly language program that displays the first four digits of your roll number on 2x16 LCD.

Write the code in the space provided below:

Appendix-A: Lab Evaluation Criteria

Labs with projects

1. Class Participation 10%
2. Lab Work 40%
3. Quiz (2) 20%
4. Project 30%
    a. Project Demonstration 20%
    b. Project Report 5%
    c. Project Quiz 5%

cvb

Labs without projects

1. Class Participation 10%
2. Lab Work 40%
3. Quiz (2) 20%
4. Lab Final 30%
    a. Lab Final (Practical) 20%
    b. Lab Final (Written) 10%

Appendix-B: Safety around Electricity

In all the Electrical Engineering (EE) labs, with an aim to prevent any unforeseen accidents during conduct of lab experiments, following preventive measures and safe practices shall be adopted:

- Remember that the voltage of the electricity and the available electrical current in EE labs has enough power to cause death/injury by electrocution. It is around 50V/10 mA that the "cannot let go" level is reached. "The key to survival is to decrease our exposure to energized circuits."
- If a person touches an energized bare wire or faulty equipment while grounded, electricity will instantly pass through the body to the ground, causing a harmful, potentially fatal, shock.
- Each circuit must be protected by a fuse or circuit breaker that will blow or "trip" when its safe carrying capacity is surpassed. If a fuse blows or circuit breaker trips repeatedly while in normal use (not overloaded), check for shorts and other faults in the line or devices. Do not resume use until the trouble is fixed.
- It is hazardous to overload electrical circuits by using extension cords and multi-plug outlets. Use extension cords only when necessary and make sure they are heavy enough for the job. Avoid creating an "octopus" by inserting several plugs into a multi-plug outlet connected to a single wall outlet. Extension cords should ONLY be used on a temporary basis in situations where fixed wiring is not feasible.
- Dimmed lights, reduced output from heaters and poor monitor pictures are all symptoms of an overloaded circuit. Keep the total load at any one time safely below maximum capacity.
- If wires are exposed, they may cause a shock to a person who comes into contact with them. Cords should not be hung on nails, run over or wrapped around objects, knotted or twisted. This may break the wire or insulation. Short circuits are usually caused by bare wires touching due to breakdown of insulation. Electrical tape or any other kind of tape is not adequate for insulation!
- Electrical cords should be examined visually before use for external defects such as: Fraying (worn out) and exposed wiring, loose parts, deformed or missing parts, damage to outer jacket or insulation, evidence of internal damage such as pinched or crushed outer jacket. If any defects are found the electric cords should be removed from service immediately.
- Pull the plug not the cord. Pulling the cord could break a wire, causing a short circuit.
- Plug your heavy current consuming or any other large appliances into an outlet that is not shared with other appliances. Do not tamper with fuses as this is a potential fire hazard. Do not overload circuits as this may cause the wires to heat and ignite insulation or other combustibles.
- Keep lab equipment properly cleaned and maintained.
- Ensure lamps are free from contact with flammable material. Always use lights bulbs with the recommended wattage for your lamp and equipment.
- Be aware of the odor of burning plastic or wire.
- ALWAYS follow the manufacturer recommendations when using or installing new lab equipment. Wiring installations should always be made by a licensed electrician or other qualified person. All electrical lab equipment should have the label of a testing laboratory.

- Be aware of missing ground prong and outlet cover, pinched wires, damaged casings on electrical outlets.
- Inform Lab engineer / Lab assistant of any failure of safety preventive measures and safe practices as soon you notice it. Be alert and proceed with caution at all times in the laboratory.
- Conduct yourself in a responsible manner at all times in the EE Labs.
- Follow all written and verbal instructions carefully. If you do not understand a direction or part of a procedure, ASK YOUR LAB ENGINEER / LAB ASSISTANT BEFORE PROCEEDING WITH THE ACTIVITY.
- Never work alone in the laboratory. No student may work in EE Labs without the presence of the Lab engineer / Lab assistant.
- Perform only those experiments authorized by your teacher. Carefully follow all instructions, both written and oral. Unauthorized experiments are not allowed.
- Be prepared for your work in the EE Labs. Read all procedures thoroughly before entering the laboratory. Never fool around in the laboratory. Horseplay, practical jokes, and pranks are dangerous and prohibited.
- Always work in a well-ventilated area.
- Observe good housekeeping practices. Work areas should be kept clean and tidy at all times.
- Experiments must be personally monitored at all times. Do not wander around the room, distract other students, startle other students or interfere with the laboratory experiments of others.
- Dress properly during a laboratory activity. Long hair, dangling jewelry, and loose or baggy clothing are a hazard in the laboratory. Long hair must be tied back, and dangling jewelry and baggy clothing must be secured. Shoes must completely cover the foot.
- Know the locations and operating procedures of all safety equipment including fire extinguisher. Know what to do if there is a fire during a lab period; “Turn off equipment, if possible and exit EE lab immediately.”