Faculty **Dated:** _____

Member:_____

Semester:_____ **Section:** _____

School of Electrical Engineering and Computer Science

EE-222 Microprocessor Systems

Lab1-B: Introduction To Assembly Language And MASM

Name	Reg. No.	Viva / Quiz / Lab Performance 5 Marks	Analysis of data in Lab Report 5 Marks	Modern Tool usage 5 Marks	Ethics and Safety 5 Marks	Individual and Team Work 5 Marks	Total 25 Marks

EXPERIMENT 01

INTRODUCTION TO ASSEMBLY LANGUAGE AND MICROSOFT ASSEMBLER (MASM)

OBJECTIVES:-

1. Getting introduced to assembly language
2. Learning some basic commands
3. Introduction to the syntax of assembly language programming
4. Learning the use of Microsoft assembler (MASM)

EQUIPMENT:-

SOFTWARE:

- Microsoft assembler (MASM)

DISCUSSION:

Before starting coding in assembly we should get familiarized with some basic coding parameters, assembly language syntax and some basics of microprocessors.

Introduction to Registers:

There are four type of registers in microprocessors:

1. AX
2. BX
3. CX
4. DX

AX, BX are mainly used for arithmetic operations and saving address. CX is used for saving the values for counts which is used in executing loop instructions and DX is mainly used for I/O operations.

PROGRAM STRUCTURE:-

MEMORY MODELS:

The size of code and data a program can have is determined by specifying memory model using the .MODEL directive. The models used are TINY, SMALL, LARGE, and HUGE but the appropriate one is .SMALL. The model directive should come before any segment definition.

DATA SEGMENT:

The data segment is used for all the variables definitions. We use **.DATA** directive followed by variable and constant declaration.

STACK SEGMENT:

The purpose of stack segment is to set aside a block of memory to store the stack. The declaration syntax is:

.stack size

If we write:

.stack 100h

100 bytes would be reserved for stack. If size omitted 1KB is the default size.

CODE SEGMENT:

Code segment contains all the programming instructions. The declaration syntax is:

.code

So with minor variations this form may be used

.model small

.stack 100h

.data

; Data definitions go here

.code

Main proc

; Instructions go here

Main endp

End main

Here main proc and main endp delineate the procedure. The last line here should be END directive followed by name of main procedure.

REAL MODE:

In real-address mode, only 1 MByte of memory can be addressed, from hexadecimal 00000 to FFFFF. The processor can run only one program at a time, but it can momentarily interrupt that program to process requests (called interrupts) from peripherals. Application programs are permitted to access any memory location, including addresses that are linked directly to system hardware. The MS-DOS operating system runs in real-address mode, and Windows 95 and 98 can be booted into this mode.

PROTECTED MODE:

In protected mode, the processor can run multiple programs at the same time. It assigns each process (running program) a total of 4 Giga Byte of memory. Each program can be assigned its own reserved memory area, and programs are prevented from accidentally accessing each other's code and data. MS-Windows and Linux run in protected mode.

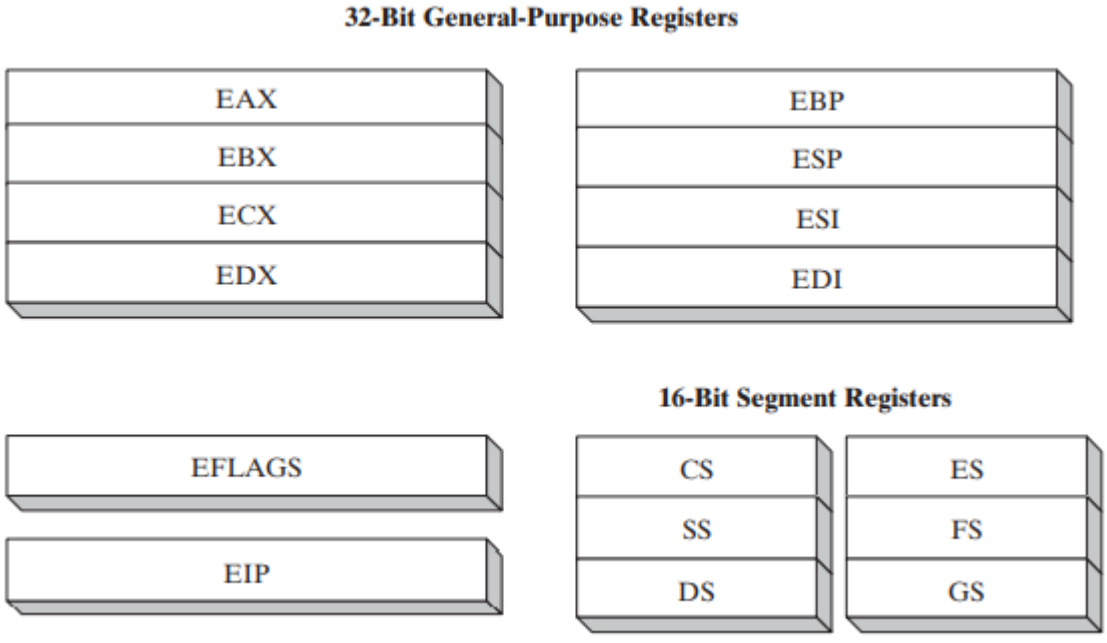
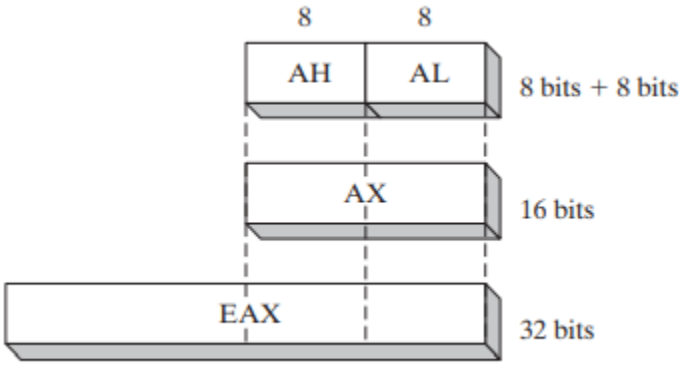


Figure 1: Shows extended CPU registers.



32-Bit	16-Bit	8-Bit (High)	8-Bit (Low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

32-Bit	16-Bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

Figure 2: Shows 32 and 16bit registers.

Operand	Description
reg8	8-bit general-purpose register: AH, AL, BH, BL, CH, CL, DH, DL
reg16	16-bit general-purpose register: AX, BX, CX, DX, SI, DI, SP, BP
reg32	32-bit general-purpose register: EAX, EBX, ECX, EDX, ESI, EDI, ESP, EBP
reg	Any general-purpose register
sreg	16-bit segment register: CS, DS, SS, ES, FS, GS
imm	8-, 16-, or 32-bit immediate value
imm8	8-bit immediate byte value
imm16	16-bit immediate word value
imm32	32-bit immediate doubleword value
reg/mem8	8-bit operand, which can be an 8-bit general register or memory byte
reg/mem16	16-bit operand, which can be a 16-bit general register or memory word
reg/mem32	32-bit operand, which can be a 32-bit general register or memory doubleword
mem	An 8-, 16-, or 32-bit memory operand

Above table shows the details of 8-bit, 16-bit and 32-bit register (reg), immediate (imm) and memory (mem) opernads.

Kip Irvine Library Built-in Subroutines (Functions):

FUNCTION NAME	FUNCTIONS
DumpRegs	Writes the EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EFLAGS, and EIP registers in hexadecimal, as well as the Carry, Sign, Zero, Overflow, Auxiliary and Parity flags.
DumpMem	Writes a block of memory to the console window in hexadecimal.
Clrscr	Clears the console and locates the cursor at the upper left corner.
WaitMsg	Displays a message and waits for a key to be pressed.
INPUT	
ReadChar	Waits for a single character to be typed at the keyboard (without echo) and returns the character.
ReadDec	Reads an unsigned 32-bit decimal integer from the keyboard, terminated by the Enter key.
ReadHex	Reads a 32-bit hexadecimal integer from the keyboard, terminated by the Enter key.
ReadInt	Reads a 32-bit signed decimal integer from the keyboard, terminated by the Enter key.
ReadKey	Reads a character from the keyboard's input buffer without waiting for input.
ReadString	Reads a string from the keyboard, terminated by the Enter key.
OUTPUT	
WriteChar	Writes a single character in register AL to standard output.
WriteDec	Writes an unsigned 32-bit integer to the console window in decimal format.
WriteHex	Writes a 32-bit integer to the console window in hexadecimal format
WriteInt	Writes a signed 32-bit integer to the console window in decimal format.
WriteString	Writes a null-terminated string to the console window.

Assemble – Link – Execute Cycle:

The process of editing, assembling, linking, and executing assembly language programs is summarized in Figure below. Following is a detailed description of each step.

Step 1: A programmer uses a **text editor** to create an ASCII text file named the *source file*.

Step 2: The **assembler** (file ML.exe) reads the source file and produces an *object file,* a machine-language translation of the program. Optionally, it produces a *listing file*. If any errors occur, the programmer must return to Step 1 and fix the program.

Step 3: The **linker** (file Link32.exe) reads the object file and checks to see if the program contains any calls to procedures in a link library. The **linker** copies any required procedures from the link library, combines them with the object file, and produces the *executable file*. Optionally, the linker can produce a *map file*.

Step 4: The operating system **loader** utility reads the executable file into memory and branches the CPU to the program's starting address, and the program begins to execute.

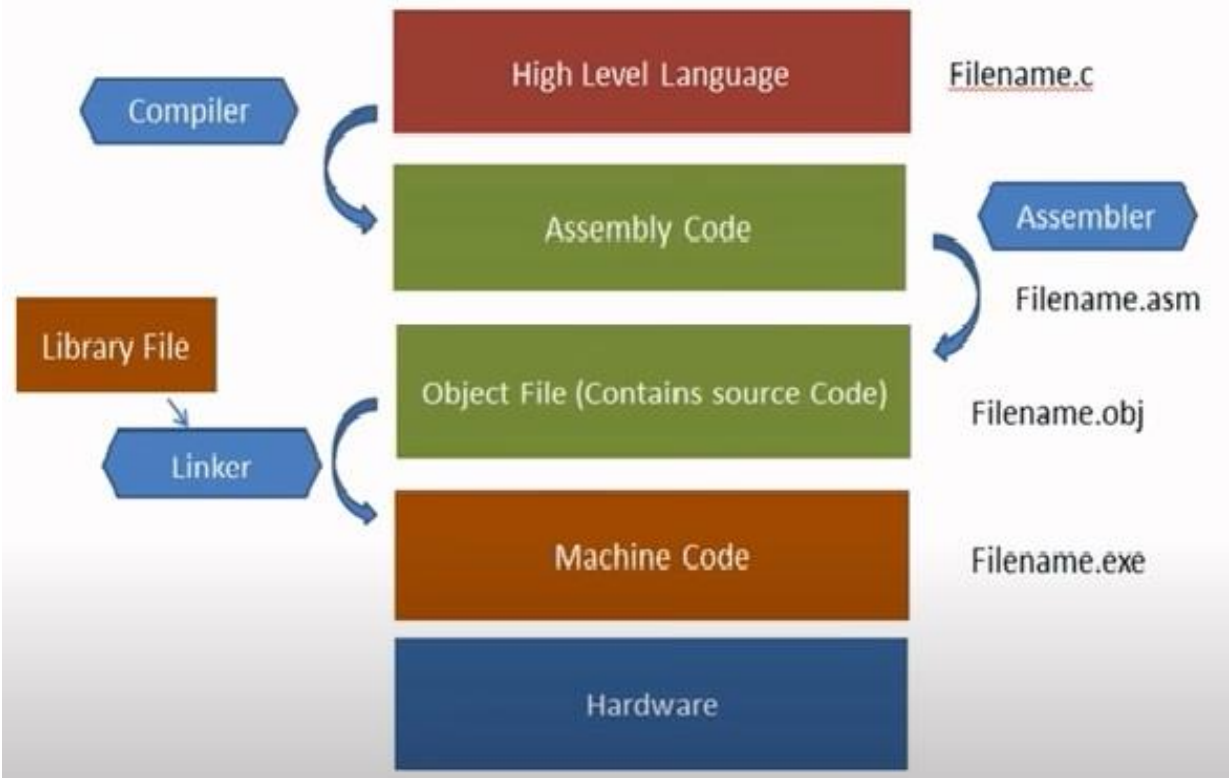


Figure 3: Shows the process of assembling and linking.

GETTING STARTED WITH MASM:

1. Download the Microsoft Assembler using following link.
https://drive.google.com/drive/folders/1duid_Q4TOdjCsujgv_M6_YipUxgF3EKs?usp=sharing
   After downloading extract this Zip file and then paste this folder inside C drive.

2. Open Notepad copy example code given below to Notepad and Save As the fie with *.asm* extension. Make sure you saved the file in the directory "C:\masm615" which is directory in which Masm is installed.

3. Open the command prompt by typing *cmd* in Run and change your current path to "C:\masm615"
   By typing following commands in command prompt.
   cd c:\masm615

4. Use make32.bat file to assembling and linking by typing following in command prompt.
   make32 example
   This will create following files.
   Example.exe
   Example.ilk
   Example.lst
   Example.obj
   Example.pdb
   Note: make32.bat is batch file containing list of commands for assembling, linking and setting the paths.

5. Run the .exe file by typing following in command prompt
   Example.exe
   This will show the output of your code.

SAMPLE CODE:

PROGRAM DESCRIPTION:

Start with displaying a **"?"** and then reading a character from keyboard and displaying it on next line.

```
Include irvine32.inc
.model small
.stack 100h
.data                   ;variable declarations go here
.code
Main Proc               ;instructions go here
call clrscr             ;clears the screen

Call dumpregs           ;displays the result on the screen by displaying all register values

mov eax,0               ;initialize eax with zero

Call dumpregs           ;displays the result on the screen by displaying all register values

mov al,0ah
call writechar

Call dumpregs           ;displays the result on the screen by displaying all register values

mov al,'?'
call writechar

call readchar
mov dl,al

mov al,0ah
call writechar

mov al,dl
call writechar

Call dumpregs           ;displays the result on the screen by displaying all register values

call WaitMsg

Exit
Main endp
End main
```

EXERCISES:

1. Edit the example code above and now move 3 into dl register and then run the program. Observe what is the output and why?
2. Manipulate the example code so that the user input that will now be printed on first position of first line. (Hint: the ? will be override by the user input)