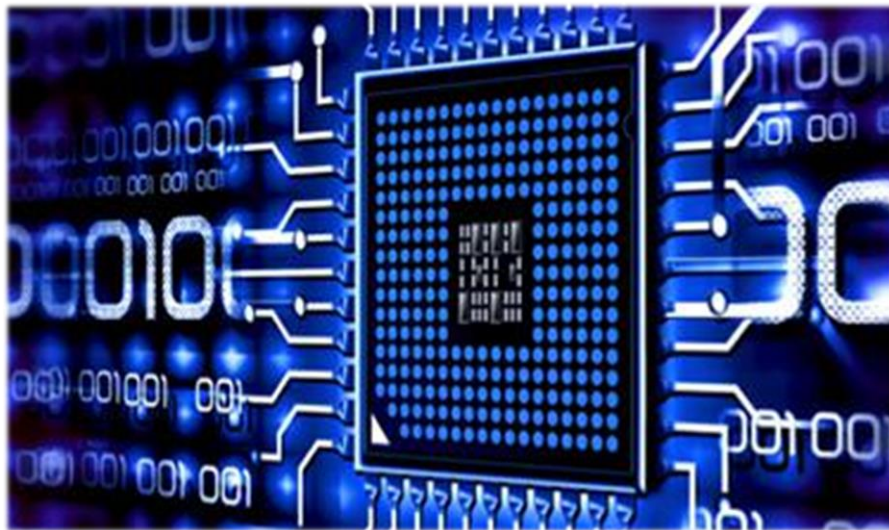




COAL-Lab 01



Instructor: Engr. Khuram Shahzad
**EL20002 – Computer Organization & Assembly
Language-Lab**

FEBRUARY 10, 2022

NATIONAL UNIVERSITY OF COMPUTER AND EMERGING SCIENCES, FAST- PESHAWAR CAMPUS
Department of Computer Science & Software Engineering

Computer Organization and Assembly Language Lab-01

OBJECTIVES OF THE LAB

- Introductions to Assemblers & Editors, Assembly Programming,
- Installation of MASM, Assembling,
- Linking and executing “.asm files” using Command Prompt (CMD), To program “Hello World” for testing of MASM successful installation
- Installation and Introduction to NASM
- First assembly code in NASM, Debug through AFD, Assembly to Machine Code, Read and Write in Memory and its visualization, Use of Interrupt
- Two programs as a task.



Computer Organization and Assembly Language Lab-01

Table of Contents

Computer Organization and Assembly Language Lab-01	0
OBJECTIVES OF THE LAB.....	0
1. Interpreter	2
2. Assembler.....	2
3. Linker.....	2
4. Loader	2
5. Cross-compiler	2
6. Source-to-source Compiler	3
7. How to Install NASM on Windows 10 How to Type and Run Assembly Language Program	3
8. To set up in Ubuntu 20.04+.....	3
9. NASM Installation on Windows 10 using DOSBOX	4
10. AFD - Advance Full Screen Debug Explain.....	5
REGISTERS' WINDOW.....	5
INSTRUCTIONS' WINDOW	6
MEMORY WINDOWS	7
11. First Program to Add two Numbers in Assembly Language	7
12. 4c00 explain	9



Computer Organization and Assembly Language Lab-01

Concept of assembler, compiler, interpreter, loader and linker.

Let us first understand how a program, using C compiler, is executed on a host machine.

User writes a program in C language (high-level language). The C compiler, compiles the program and translates it to assembly program (low-level language). An assembler then translates the assembly program into machine code (object). A linker tool is used to link all the parts of the program together for execution (executable machine code). A loader loads all of them into memory and then the program is executed.

1. Interpreter

An interpreter, like a compiler, translates high-level language into low-level machine language. The difference lies in the way they read the source code or input. A compiler reads the whole source code at once, creates tokens, checks semantics, generates intermediate code, executes the whole program and may involve many passes. In contrast, an interpreter reads a statement from the input, converts it to an intermediate code, executes it, then takes the next statement in sequence. If an error occurs, an interpreter stops execution and reports it. whereas a compiler reads the whole program even if it encounters several errors.

2. Assembler

An assembler translates assembly language programs into machine code. The output of an assembler is called an object file, which contains a combination of machine instructions as well as the data required to place these instructions in memory.

3. Linker

Linker is a computer program that links and merges various object files together in order to make an executable file. All these files might have been compiled by separate assemblers. The major task of a linker is to search and locate referenced module/routines in a program and to determine the memory location where these codes will be loaded, making the program instruction to have absolute references.

4. Loader

Loader is a part of operating system and is responsible for loading executable files into memory and execute them. It calculates the size of a program (instructions and data) and creates memory space for it. It initializes various registers to initiate execution.

5. Cross-compiler

A compiler that runs on platform (A) and is capable of generating executable code for platform (B) is called a cross-compiler.

6. Source-to-source Compiler

A compiler that takes the source code of one programming language and translates it into the source code of another programming language is called a source-to-source compiler.

7. How to Install NASM on Windows 10 | How to Type and Run Assembly Language Program

The Netwide Assembler is an assembler and disassembler for the Intel x86 architecture. It can be used to write 16-bit, 32-bit and 64-bit programs. NASM is considered to be one of the most popular assemblers for Linux.

NASM stands for the net assembler. if you want to type edit and execute an assembly language program you need to install NASM on Windows 10 using DosBox. in this tutorial you will be guided about how to install NASM on Windows 10 using dosbox. you will also be guided to type an assembly language program and how to assemble it and then execute the Assembly language program in AFD.

NASM can be used to write 16 bit, 32-bit and 64-bit programs. NASM is one of the most popular assemblers for Linux

8. To set up in Ubuntu 20.04+

- # Install Doxbox
- `sudo apt install dosbox`
- `dosbox`
- # Inside dosbox (give it some path in your host machine)
- # Put extracted contents of the zip file and .asm files in this folder on your host machine
- `mount c /home/nam/ee213`
- `c:`
- # Assemble and run
- `nasm c.asm -o c.com`
- `afd c.com`
- # Assemble with Listing
- `nasm c.asm -l c.lst -o c.com`
- # If after changing files, they do not show up in dosbox, hit Ctrl+F4 inside the dosbox window.

To see the all file in virtual machine folder type **dir** command

9. NASM Installation on Windows 10 using DOSBOX

Here are steps to Install NASM on Windows

- Click to [download NASM and DOSBOX](#)
- Install DOXSBOX
- Extract NASM in a Folder
- Run DoxBox
- Mount NASM folder to a Drive using “mount [driveletter] [NASM Path]

For example we have extracted NASM on F:\Assebmly then here is command to mount it to X Drive

```
---> mount X f:\assembly  
Type x:  
x:\> af
```

10. AFD - Advance Full Screen Debug Explain

The screenshot shows the DOSBox 0.74 AFD interface. The top bar displays 'DOSBox 0.74, Cpu speed: 4000 cycles, Frameskip 0, Program: AFD'. Below this, a red-bordered window shows the state of registers: AX=0000, SI=0000, CS=19F7, IP=0100, Stack=+0 0000, Flags=7202; BX=0000, DI=0000, DS=19F7, +2 20CD; CX=0000, BP=0000, ES=19F7, HS=19F7, +4 9FFF, OF=DF, IF=SF, ZF=AF, PF=CF; DX=0000, SP=FFFE, SS=19F7, FS=19F7, +6 EA00, 0 0 1 0 0 0 0 0. Below the registers is a command line 'CMD >' with a green cursor. To the right of the command line is a green-bordered window showing memory location 1 (DS:0000) with a hex dump and ASCII representation. Below the command line is a yellow-bordered window showing disassembly of instructions at addresses 0100 to 010E, all being 'ADD [BX+SI],AL'. At the bottom is a white-bordered window showing a larger memory dump from DS:0000 to DS:0040, including hex values and ASCII characters like 'f.n', '...', 'w.x', 'u...', and '...'. The bottom status bar shows navigation keys: 1 Step, 2 ProcStep, 3 Retrieve, 4 Help ON, 5 BRK Menu, 6, 7 up, 8 dn, 9 le, 10 ri.

These 4 parts are described as follows

Red - Displaying All the Registers

Green - Displaying Memory Location 1

Yellow - Displaying Memory Location 2

Whereas White for disassembly.

REGISTERS' WINDOW

The 1st window (Red One) is for displaying the values in all the registers. The values in registers updates after each execution step. like if we have an instruction

MOV AX,0014

It will change the value of AX in first window to "0014" (hexadecimal representation).

Similarly operations on each register will change its value respectively .

But, What if you want to change the value of the register, manually, on the fly ?

In CMD You can simply type

R <register>=value or <register>=value (both will work)

e.g.

AX=15 (will set AX to 0015)

BX=002A (will set BX to 002A)

```
AX 0000 SI 0000 CS 19F7 IP 0100
BX 0000 DI 0000 DS 19F7
CX 0000 BP 0000 ES 19F7 HS 19F7
DX 0000 SP FFFE SS 19F7 FS 19F7

{R} reg=value
CMD >AX = 29
```

```
AX 0029 SI 0000 CS 19F7 IP 0100
BX 0000 DI 0000 DS 19F7
CX 0000 BP 0000 ES 19F7 HS 19F7
DX 0000 SP FFFE SS 19F7 FS 19F7

CMD >
```

if you need to change flag registers you can do it in two ways

1st: writing directly to 16bit flag register by typing

FL=<value>

e.g. FL=0060 (it will set the value of Zero Flag, also shown in the image below)

2nd: writing directly to a single bit

e.g. CF = 1 (setting carry flag)

```
Flags 0060
OF DF IF SF ZF AF PF CF
0 0 0 0 1 0 0 0
```

The different flag bits are shown in the image to the left.

INSTRUCTIONS' WINDOW

Let's move to the 2nd window that is the White one. This window shows the instructions in the program. The current instruction is always highlighted with white color. You can execute the instruction using two function keys F1 and F2. These are the most frequently used keys when working in AFD.

Their use is as follows,

F2 is used for step by step execution of instructions

whereas pressing F1 also executes instructions step by step until we encounter a "call"

instruction (that is used to call a procedure) it will "step into" rather than stepping over the line (

like in case of F1 key). Non technically, it means that pressing F1 will make AFD to jump to the first instruction of the procedure and highlight that instruction (to keep things simple I am skipping the use of stack in call instruction).

For code Window we have discussed Step Over and Step Into (a.k.a procedure Step) keys. I will cover "alter/assemble code in AFD" in advance section (next tutorial) although it is very useful as well as powerful feature but , again , to keep things simple let's discuss it later .

MEMORY WINDOWS

The other two windows are used for displaying the content in memory. The window colored green is "m1" and the window colored yellow is "m2". Memory2 or m2 has an advantage that it shows large area of memory and on its right side you also have character representation of hex values. This is useful in debugging string data. In a nutshell when our program has some data in main memory we need m1 and m2 to watch that data.

Note: To save yourself from writing the commands again and again you can use 'F3' as AFD maintains history of all the commands user had used. Pressing F3 repeatedly will take you through all of them.

11. First Program to Add two Numbers in Assembly Language

Type following first program of Assembly Language in any Editor. Like Notepad

```
;assembly language program to add two numbers

org 0x100

mov ax,5      ; move the constant 5 int to register ax
mov bx,10     ; move the constant 10 int to register bx
add ax,bx     ; add value of bx into ax

mov bx, 15
add ax,bx

mov ax, 0x4c00 ; exit

int 0x21      ;... is what the OS should do for me
```

- Save it with name "p1.asm" in the same folder you have installed NASM
- Open DosBox and Assemble it using NASM (like compile in C++)

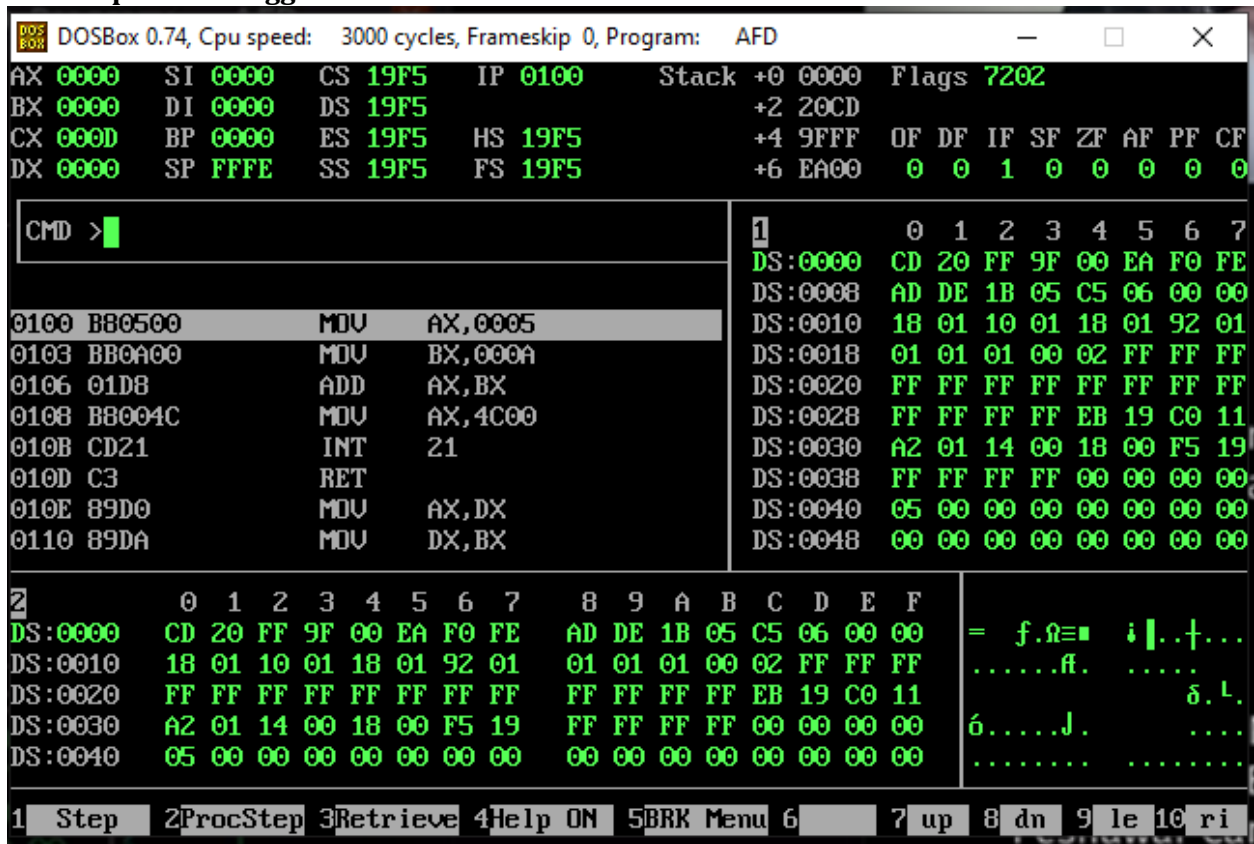
```
x:\>NASM p1.asm -o p1.com ;
```

```
x:\>AFD pl.com
```

Explain: -o mean output and .com mean output will be and exe file, .com is exe extension in older OS version.

- Program will be loaded in Debugger. USE F2 to run the program and to see the values of registers.

✓ Output in debugger



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD

AX 0000	SI 0000	CS 19F5	IP 0100	Stack +0 0000	Flags 7202
BX 0000	DI 0000	DS 19F5		+2 20CD	
CX 000D	BP 0000	ES 19F5	HS 19F5	+4 9FFF	OF DF IF SF ZF AF PF CF
DX 0000	SP FFFE	SS 19F5	FS 19F5	+6 EA00	0 0 1 0 0 0 0 0

CMD > █

0100 B80500	MOV	AX,0005
0103 BB0A00	MOV	BX,000A
0106 01D8	ADD	AX,BX
0108 B8004C	MOV	AX,4C00
010B CD21	INT	21
010D C3	RET	
010E 89D0	MOV	AX,DX
0110 89DA	MOV	DX,BX

1	0	1	2	3	4	5	6	7
DS:0000	CD	20	FF	9F	00	EA	F0	FE
DS:0008	AD	DE	1B	05	C5	06	00	00
DS:0010	18	01	10	01	18	01	92	01
DS:0018	01	01	01	00	02	FF	FF	FF
DS:0020	FF	FF	FF	FF	FF	FF	FF	FF
DS:0028	FF	FF	FF	FF	EB	19	C0	11
DS:0030	A2	01	14	00	18	00	F5	19
DS:0038	FF	FF	FF	FF	00	00	00	00
DS:0040	05	00	00	00	00	00	00	00
DS:0048	00	00	00	00	00	00	00	00

2	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DS:0000	CD	20	FF	9F	00	EA	F0	FE	AD	DE	1B	05	C5	06	00	00
DS:0010	18	01	10	01	18	01	92	01	01	01	00	02	FF	FF	FF	FF
DS:0020	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	FF	EB	19	C0	11
DS:0030	A2	01	14	00	18	00	F5	19	FF	FF	FF	FF	00	00	00	00
DS:0040	05	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

1 Step 2ProcStep 3Retrieve 4Help ON 5BRK Menu 6 7 up 8 dn 9 le 10 ri

```
x:\>NASM pl.asm -l pl.lst -o pl.com ;
```

```
x:\>AFD pl.com
```

Output LST (Machine Code)

P1.LST - Notepad

File Edit Format View Help

```

1      ;assembly language program to add two number
2      org 0x100
3      mov ax,5
4      mov bx,10
5      add ax,bx
6      mov bx, 15
7      add ax,bx
8      mov ax, 0x4c00
9      int 0x21
  
```

B8 is operand
0500 is opcode

1 Bit	0 or 1
1 Nibble	4 Bits
1 byte	8 bits
One Word	2 bytes

12.4c00 explain

