National University of Computer and Emerging Sciences

Lab Manual

"Introduction to Listing File, difference Between Mnemonic and Opcode, Data Declaration in Memory, and Direct Addressing Mode and Jumps"

COMPUTER ORGANIZATION AND ASSEMBLY LANGUAGE

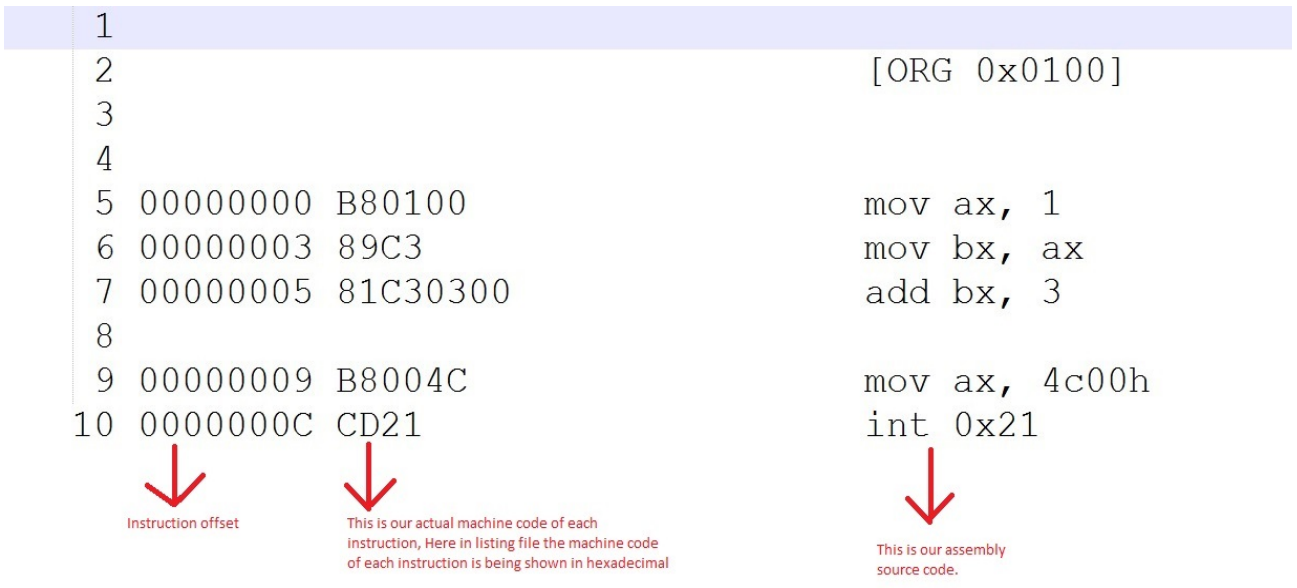
Course Instructor	Miss Aleena
Lab Instructor(s)	Maham Saleem
Section	E1,E2
Semester	Fall 2021

Department of Computer Science
FAST-NU, Lahore, Pakistan

Listing File:
Listing file is created by nasm when you enter "-l list_file_name.lst" during assembling the program, such as, if you enter the following:

```
nasm code.asm –o output.com  –l list_file.lst
```

The above command will now generate a listing file named "list_file.lst" along with the com file. Listing File will look something like this:



```
 1
 2                                            [ORG 0x0100]
 3
 4
 5 00000000 B80100                            mov ax, 1
 6 00000003 89C3                              mov bx, ax
 7 00000005 81C30300                          add bx, 3
 8
 9 00000009 B8004C                            mov ax, 4c00h
10 0000000C CD21                              int 0x21
```

Instruction offset

This is our actual machine code of each instruction, Here in listing file the machine code of each instruction is being shown in hexadecimal

This is our assembly source code.

Difference between Mnemonic and Opcode:

MNEMONIC: Human Readable words. The assembly keywords, such as, mov, add, sub, etc are MNEMONICS for programmers because they are easily understood by them. We use these MENMONICS to write programs because we can easily remember mnemonics. Mnemonics cannot be executed by the CPU, so mnemonics are always converted into some opcode which can be executed by the CPU.

OPCODE: It is a number interpreted by the CPU that represents the operation to perform. For example in the above listing file, the opcode for moving an immediate operand into AX register is B8. Can you identify the opcode for adding an immediate value into BX from the above listing file?

Purpose of [ORG 0x0100]:

It simply tells the nasm that the instructions of our program should be placed at the start of 255th byte of code segment. (The first 256 bytes are to be skipped). Note that 0x0100 is a hexadecimal number whose decimal value is 256. The reason we skip the first 256 bytes is because these bytes have some important piece of code that we do not want to overwrite with our own. This will be further clarified in some later lab session.

Data Declaration in Memory:
Suppose that we want to declare a variable and initialize that variable with 10. To do this, we have to add following code at the end of our assembly code.

```
 data:  dw 10
```

Here Data is simply a label of the memory location where 10 is stored. dw (define word) means that 10 will be stored in 2 bytes. We can also store it in 1 byte by using db (define byte). Now To access this 10, we can use the following

```
Mov AX, [data]   ;this will move 10 to AX register.
MOV AX, data     ;this will move the address (offset) of 10.
```

We can also declare memory contiguously like an array, such as:

```
 data: dw 10, 3, 4, 5, 6
```

Since 10 is stored in 2 bytes, so to access 3, we will have to skip the first 2 bytes.

```
Mov AX, [data+2]   ; this will move 3 to AX
```

Similarly, to access 4, we will have to skip first 4 bytes:

```
MOV AX, [data+4]  ;this will move 4 to AX
```

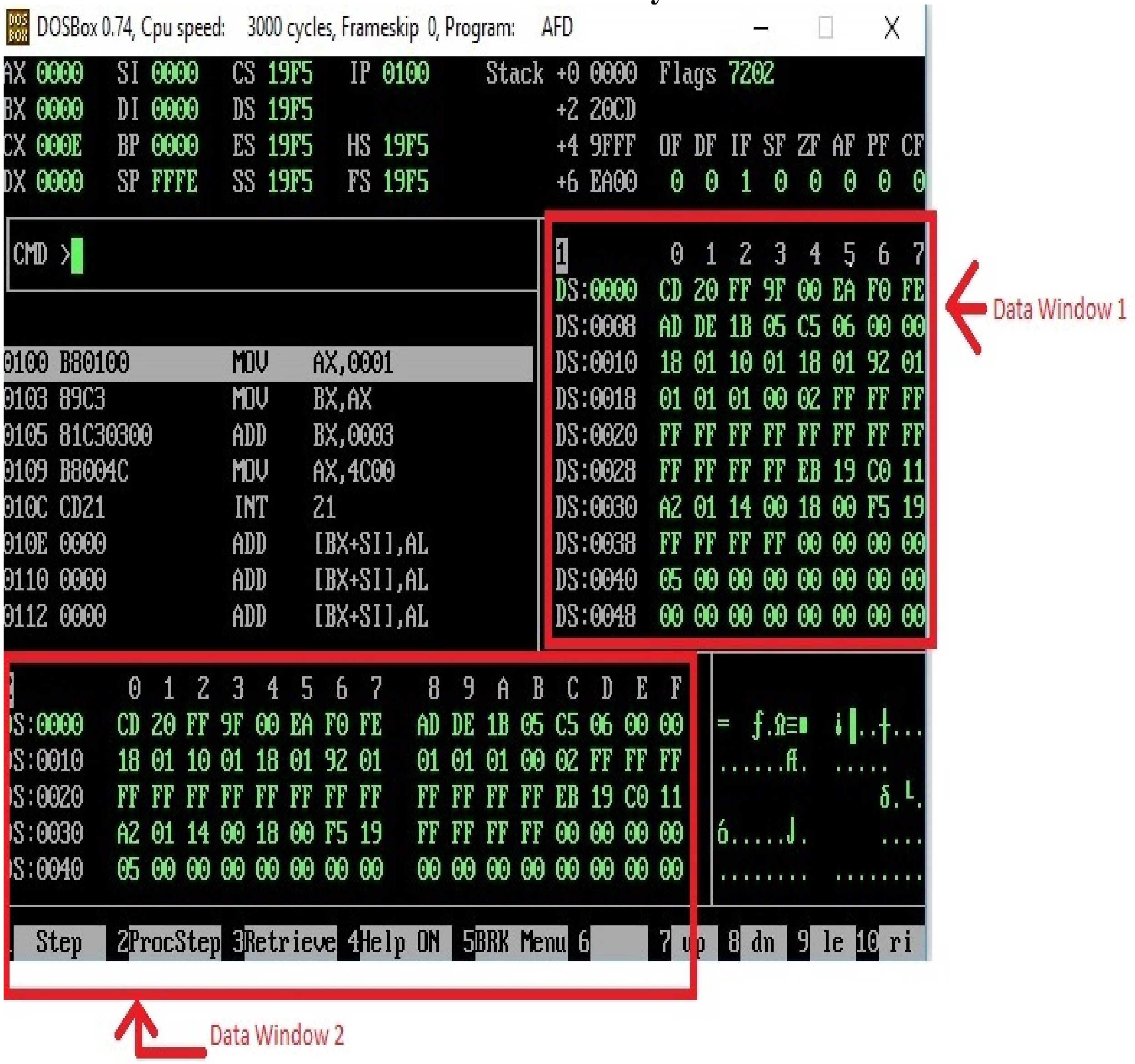
Endianness:
Endianness refers to the notation in which a number's bytes are placed in memory. There are two types

1. **Little Endian Notation:**
   In this notation a number's most significant bytes are placed at higher address, and the least significant bytes are placed at lower address, such as if you declare a number 0x4FC0 (a 2-byte hexadecimal number) in memory, then, using little endian notation, it will be placed in memory in the following way:

   C04F (0c is the least significant byte and placed at lower address, whereas F4 is most significant byte and placed at higher address)
2. **Big Endian Notation:**

In big endian notation, the most significant bytes are placed at lower address, and the least significant bytes are placed at higher address. So the number 0x4FC0 will be placed as follows:
4FC0

How to View Memory In AFD:


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: AFD
Data Window 1
Data Window 2

```
AX 0000  SI 0000  CS 19F5  IP 0100   Stack +0 0000  Flags 7202
BX 0000  DI 0000  DS 19F5                  +2 20CD
CX 000E  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 >                                      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
0100 B80100        MOV    AX,0001          DS:0010  18 01 10 01 18 01 92 01
0103 89C3          MOV    BX,AX            DS:0018  01 01 01 00 02 FF FF FF
0105 81C30300      ADD    BX,0003          DS:0020  FF FF FF FF FF FF FF FF
0109 B8004C        MOV    AX,4C00          DS:0028  FF FF FF FF EB 19 C0 11
010C CD21          INT    21               DS:0030  A2 01 14 00 18 00 F5 19
010E 0000          ADD    [BX+SI],AL       DS:0038  FF FF FF FF 00 00 00 00
0110 0000          ADD    [BX+SI],AL       DS:0040  05 00 00 00 00 00 00 00
0112 0000          ADD    [BX+SI],AL       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   = ƒ.Ω≡■  ¡|..┼...
DS:0010   18 01 10 01 18 01 92 01  01 01 01 00 02 FF FF FF   ......Æ.  .....
DS:0020   FF FF FF FF FF FF FF FF  FF FF FF FF EB 19 C0 11              δ.└.
DS:0030   A2 01 14 00 18 00 F5 19  FF FF FF FF 00 00 00 00   ó.....J.      ....
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
```

In above screen shot, there are two data windows, each window is showing the contents of Memory. Such as at Offset 0000, we can see that the data is CD and at Offset 0001, the data is 20. If you want to see the data at offset 001F, simply write m1 001F or m2 001F on AFD console (m1 is for window 1, and m2 is for window 2).

InLab Problems

Problem 1: Write instructions that perform the following operations.

a. Copy BL into CL
b. Copy DX into AX
c. Store 0x12 into AL
d. Store 0x1234 into AX
e. Store 0xFFFF into AX

Problem 2: Logical Operations (AND, OR, XOR, NOT)

1. Copy the value of AX into BX using a logical instruction. (You are not allowed to use mov BX, AX)
2. Set AX=0 by using XOR instruction only.
3. Set AX=0 by using AND instruction.
4. Set AX=FFFF by using OR instruction.
5. Invert the bits of AX register using a single line instruction

Problem 3: Following is a program in assembly that adds 4 numbers declared in memory at the end of program, and stores the result in AX register. There is a logical error in the program. Find and correct the error.

```
[org 0x0100]
mov ax, [data]
add ax, [data+1]
add ax, [data+2]
add ax, [data+3]
mov ax,0x4c00
int 0x21
data: dw 10, 20, 30, 40
```

Problem 4: Develop an assembly program that reads 10 contiguously placed numbers in memory and stores their sum in AL register. After that you have to subtract 3rd number in memory from the result in AL register. Each number is of one byte.
The numbers are 1, 3, 4, 5, 9, 10, 6, 4, 1, 10

Problem 5: In problem 3, you were declaring a block of memory which stored 10 numbers. Modify the above code, so that CX register stores the address of last number, i.e. 10.

Problem 6: Convert the following C++ code into equivalent assembly code. (The logic in assembly code must mirror the logic in high level code)

```
char array[10]={10, 3, 1, 10, -1, 2, 4, 10, -5, 20 };
short sum=13;
short count=9;

sum=0;

while ( count>= 0)
{

   If (array[count] < 0 || array[count] > 10)
   {
     sum+=  array[count];
     count--;
   }


}
```

Remember: *Honesty always gives fruit (no matter how frightening is the consequence); and Dishonesty is always harmful (no matter how helping it may seem in a certain situation).*