

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

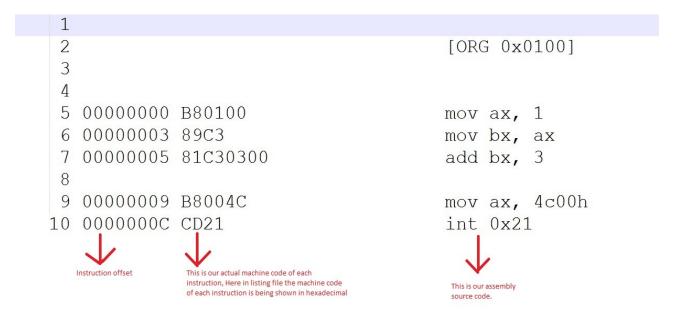


Department of Computer Science FAST-NU, Lahore, Pakistan

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:



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:

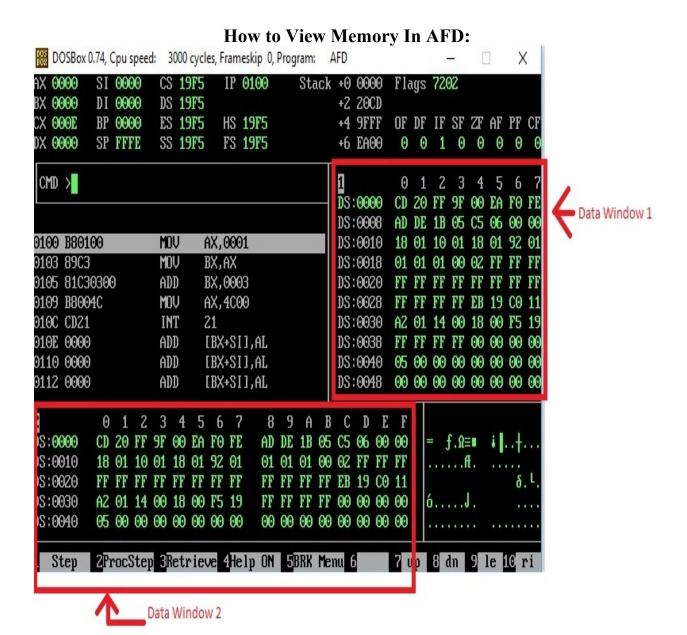
C04F (0c is the least significant byte and placed at lower address, whereas F4 is most significant byte and placed at higher address)

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





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

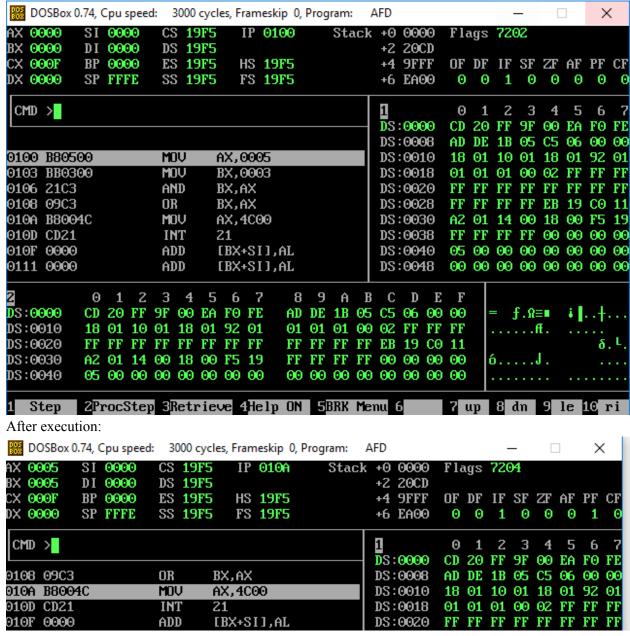
Problem 1: Write instructions that perform the following operations.

- a. Copy BL into CL mov cl,bl
- b. Copy DX into AX Mov ax,dx
- c. Store 0x12 into AL Mov al,0x12
- d. Store 0x1234 into AX Mov ax,0x1234
- e. Store 0xFFFF into AX Mov ax,0xFFFF

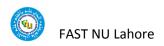
Problem 2: Logical Operations (AND, OR, XOR, NOT)

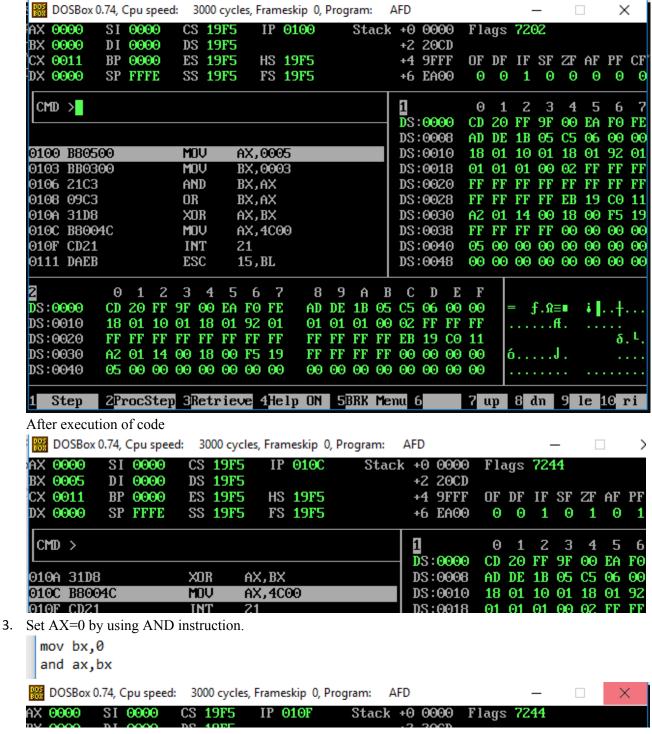
1. Copy the value of AX into BX using a logical instruction. (You are not allowed to use mov BX, AX)





2. Set AX=0 by using XOR instruction only.

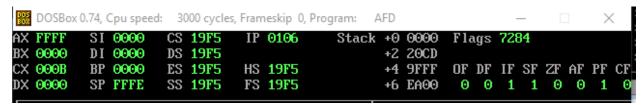




4. Set AX=FFFF by using OR instruction.

mov ax,0 or ax, 0xFFFF





5. Invert the bits of AX register using a single line instruction

```
mov ax,5
 not ax
SI 0000
                CS 19F5
                       IP 0105
        DI 0000
BX 0000
               DS 19F5
CX 000A
        BP 0000
               ES 19F5
                       HS 19F5
DX 0000
        SP FFFE
               SS 19F5
                       FS 19F5
```

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
```

ANSWER:

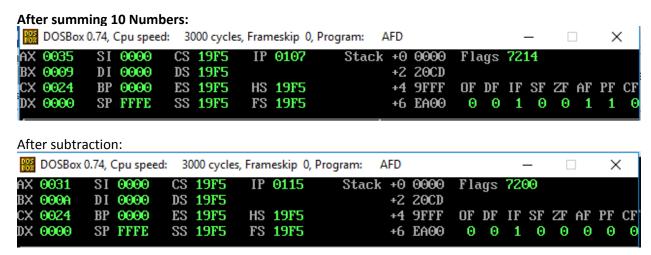
The logical error is in the brackets around data. The value to be added should be 2 each time not 1 and 3. This is because the variable 'data' is stored in dw Which takes 2 bytes. Hence, to access its next value 2 should be added to the current address.

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

```
[org 0x0100]
mov bx,0
11: add al,[data+bx]
add bx,1
cmp bx,10
jne 11
sub al,[data+2]
mov ax,0x4c00
int 0x21
data: db 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.

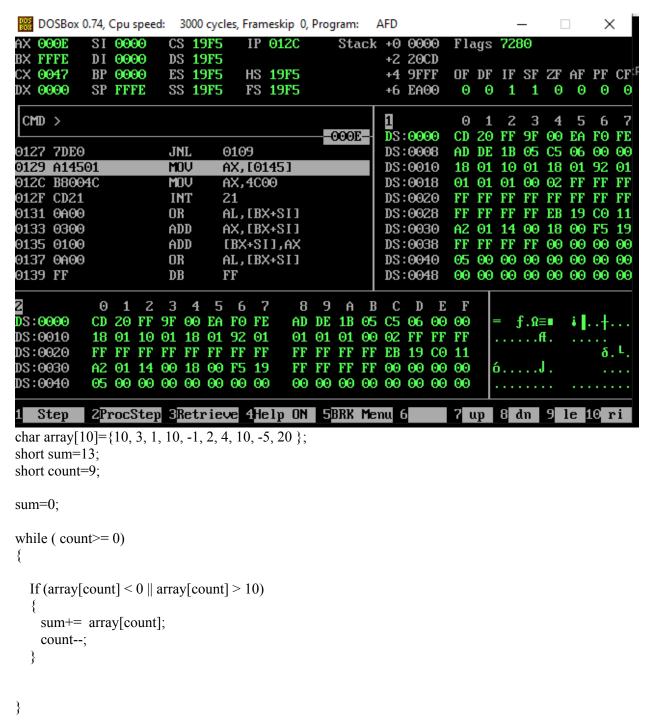
```
[org 0x0100]
mov bx,0
mov cx,data
11: add al,[data+bx]
add cx,1
add bx, 1
cmp bx,10
jne 11
sub al,[data+2]
mov ax,0x4c00
int 0x21
data: db 1,3,4,5,9,10,6,4,1,10
808 розрох олт<del>т, сра эрсса.</del> В вооб сустез, гтаптезкір ю, гтодіать
                       CS 19F5
   0035
           SI 0000
                                   IP 0118
                                                Stac
BX 000A
           DI 0000
                       DS 19F5
CX 012B
           BP 0000
                       ES 19F5
                                  HS 19F5
DX 0000
           SP FFFE
                       SS 19F5
                                  FS 19F5
```

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



```
File Edit Format View Help
[org 0x0100]
mov bx,18
mov word[sum],0
11:
mov ax,[data+bx]
cmp ax,10
JLE 13
add word[sum],ax
13: cmp ax,0
JGE 12
add word[sum],ax
12:
sub bx,2
cmp bx,0
JGE 11
mov ax, [sum]
mov ax,0x4c00
int 0x21
data: dw 10,3,1,10,-1,2,4,10,-5,20
sum: dw 13
```





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).