**Lab 03 Sec A 6/09/2017**

**Computer Organization and Assembly Language Fall 2017**

# **Addressing Modes**

***Register Addressing Mode:***

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

MOV AX, AX

AX = AH + AL => (1 + 1) byte

BX = BH + BL => (1 + 1) byte

CX, DX also

MOV AX, BX

MOV AX, CX

MOV AX, DX

MOV AL , BL

***Immediate Addressing Mode*:**

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

MOV AX, 0x25

MOV BX, 0x195

MOV CX, 2056

MOV DX, 1000

***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]

The MOV AX, [BX] instruction loads AX from the memory location specified by the contents of the BX register. This is an *indirect* addressing mode. Rather than using the value in BX, this instruction accesses to the memory location whose address appears in BX. Note that the following two instructions:

MOV BX, 1000

MOV AX, [BX]

are equivalent to the single instruction:

MOV AX, [1000]

Of course, the second sequence is preferable. However, there are many cases where the use of indirection is faster, shorter, and better.

***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 Types**

Variables are declared in memory.

|  |  |  |
| --- | --- | --- |
| DB | Define Byte | allocates 1 byte (0 - 28 – 1) |
| DW | Define Word | allocates 2 bytes (0 – 216 – 1) |
| DD | Define Doubleword | allocates 4 bytes (0 – 232 – 1) |
| DQ | Define Quadword | allocates 8 bytes (0 – 264 – 1) |
| DT | Define Ten Bytes | allocates 10 bytes (0 - 280 – 1) |

***Example:***

*; a program to add three numbers using memory variables*

*[org 0x0100]*

*mov ax, [num1] ;load first number in ax*

**Code Segment**

**Pointed by CS**

*mov bx, [num2] ; load second number in bx*

*add ax, bx ; accumulate sum in ax*

*mov [result1], ax ; store result in result1 variable, 15*

*mov ax, 0x4c00 ; terminate program*

*int 0x21*

*num1: dw 5 ;variables*

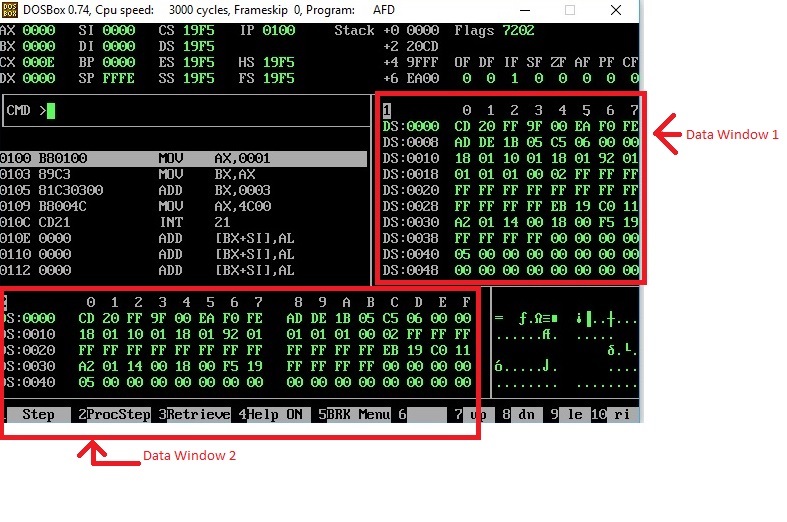
**Data Segment**

**Pointed by DS or ES**

*num2: dw 10*

*result1: dw*

**How to View Memory In AFD:**

**

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 0040, simply write m1 DS:0040 or m2 DS:0040 on AFD console.

(m1 is for window 1, and m2 is for window 2).

If you want to check your declared memory content, you have to crate listing file of your program, then note offset of your data label from listing file, and then simply write m1 DS:offset, then you will see your data label content in m1 window

**In Lab Problems**

**Problem 1:**

**Task I:**

**Which of the following mov instructions are valid or invalid?** *Do this task on a paper*

**For example**:

MOV Register1, Register2 (Valid: Mov Ax, Bx)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Destination registers  Source registers | **Immediate 2** | **Registers 2**  **(16 bit)** | **Registers 2**  **(8 bit)** | **Memory 2** |
| **Immediate 1** |  |  |  |  |
| **Register 1 (16 bit)** |  | **Valid** |  |  |
| **Register 1 (8 bit)** |  |  |  |  |
| **Memory 1** |  |  |  |  |

**Task II:**

**Following code contain lot of bugs. You need to identify bugs and re-write correct program without removing and rearranging any instruction. Logic of the program is important.** **Code add n1 and n2, then subtract n3 from result, then add n4 into n3, then subtract 10 from n3, your final answer should be 29 (1D) in n3. You are allow to add only a single instruction in code.**

*[org 0x100]*

*mov ax, n1*

*mov cx, [ax]*

*mov dx, n2*

*add cx,dx*

*mov al,n3*

*sub al,cx*

*mov n3,ax*

*add [n3], [n4]*

*sub [n3],10*

*mov ax, 0x4c00 ; terminate program*

*int 0x21*

*n1 :dw 5*

*n2: dw 10*

*n3: db 6*

*n4: db 30*

**Problem 2:**

**Task I:**

Write instructions to do the following.

**a.** Copy contents of memory location with offset 0025 into AX.

**b.** Copy AX into memory location with offset 0FFF.

**c.** Move contents of memory location with offset 0010 to memory location with offset 002F.

**Problem 3:**

**Task I:**

Make 2 word Type Arrays with 10 numbers each, add the corresponding elements of the 2 arrays and store them in a third array of type Word. Your final should be saved in Array 3.Don’t use loops

*Example:*

*Array 1 = 101, 200, 500,320,550, 632, 400, 100, 200, 900 (DB or DW?)*

*Array 2 = 50, 99, 256, 230, 550, 600, 220, 100, 200, 300 (DB or DW?)*

*Array 3 = 151, 299, 756, 550, 1100, 1232, 620, 200, 400, 1200 DB or DW?)*

**Task II:**

Again perform task-I but this time you cannot de-refer a variable for example [Array1] or [Array2] or [Array3]. Your final should be saved in array3

