National University of Computer and Emerging Sciences



Laboratory Manual

for

Computer Organization and Assembly Language Programming

(EL 213)

|  |  |
| --- | --- |
| Course Instructor | Ms. Tazeem Haider |
| Lab Instructor(s) | Mr. Gullsher Chaudhary  Ms. Nimra Abbas |
| Section | CS-3L |
| Semester | Fall 2022 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

## Objectives

After performing this lab, students shall be able to:

* Learn different types of addressing modes
* Differentiate between code data segment
* Declare and use variables in assembly language

# Addressing Modes

1. ***Register Addressing Mode:***

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

MOV AX, AX

MOV AX, BX

MOV AL , BL

1. ***Immediate Addressing Mode*:**

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

MOV AX, 0x25

MOV DX, 1000

1. ***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.

1. ***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 following 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.

1. ***Indexed Addressing Mode*:** The last memory addressing mode is the *indexed* addressing mode with the base register. 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*

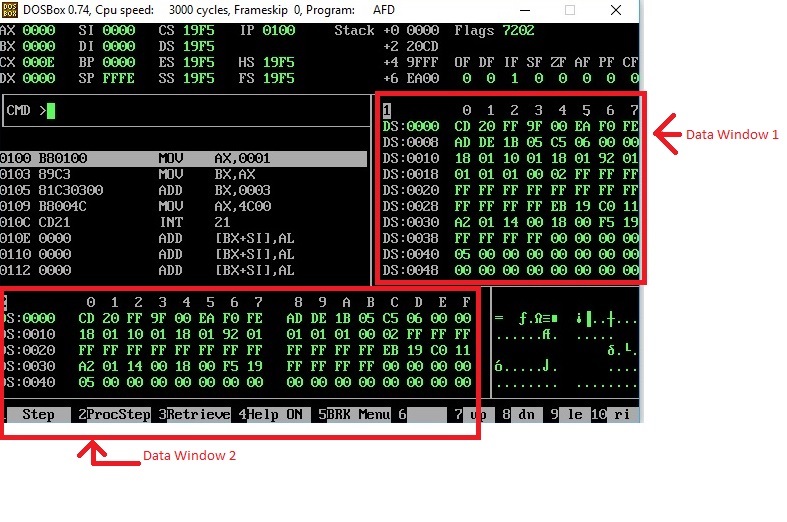
**Data Segment**

**Pointed by DS or ES**

*num2: dw 10*

*result1: dw*

**How to View Memory in AFD**



In above screenshot, 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 create 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. Offset value will be calculated after the addition of 0x0100 in the instruction address displayed by listing window since we ask assembler to start writing machine code from the address 0x0100 using the first line of code: [org 0x0100]

**Exercise 1:** Write instructions to do the following. Visualize the memory contents using memory windows to see if instruction is executed correctly. (Use m2 DS:offset to visualize the memory contents at the specified offset)

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.

**Exercise 2:** Make 2 Arrays with 10 numbers each, add the corresponding elements of the 2 arrays and store them in a third array. Your final should be saved in Array 3. You have to use Indexed Offset Addressing (e.g., mov ax,[array1+bx] where bx is the offset from the base address of array1 label) to implement the task. Use BX as your Offset Register.

Don’t use loops for this question

Given the numbers in example below, determine yourself whether Array should be byte type or word type?

Example:

Array1 = 101, 200, 500,320,550, 632, 400, 100, 200, 900 (DB or DW?)

Array2 = 50, 99, 256, 230, 550, 600, 220, 100, 200, 300 (DB or DW?)

Array3 = 151, 299, 756, 550, 1100, 1232, 620, 200, 400, 1200 (DB or DW?)

**Exercise 3:** Make two Byte type arrays named array1 and array2 and one Byte Type array named array3, each containing only 5 elements such that following rules are satisfied for each element of all the arrays:

Array1 [i+1] = Array1 [i] + Array3 [i]

Array2 [i+1] = Array3 [i] - Array2 [i]

Array3 [i] = Array1 [i] + Array2 [i]

Here “i” denotes the element number in each array and it will range from 0 to 4 for five elements. Use BX as your Offset Register. Don’t use loops for this question

You have to use two Initial Values: Array1 [0]= 0 and Array2[0]=1

The final arrays should look like this:

Array1: 0, 1, 2, 5, 12

Array2: 1, 0, 1, 2, 5

Array3: 1, 1, 3, 7, 17

**Exercise 4 (Self Exercise):** Move a number (6 for this question) from a memory location in AX, move 4 into BX then find num \* 4 using ADD instruction and then divide that answer by 3 using SUB instruction. Store the results of multiplication and division at different memory locations labeled as “mresult” and “dresult”. Do not use loops for this question too.