National University of Computer and Emerging Sciences



Laboratory Manual

for

Computer Organization and Assembly Language Programming

(EL 213)

|  |  |
| --- | --- |
| Course Instructor | Ms. Aatira Anum |
| Lab Instructor(s) | M. Salman Mubarik  Rasaal Ahmed |
| Section | H |
| Semester | Fall 2023 |

Department of Computer Science

FAST-NU, Lahore, Pakistan

## Objectives

After performing this lab, students shall be able to:

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

**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*

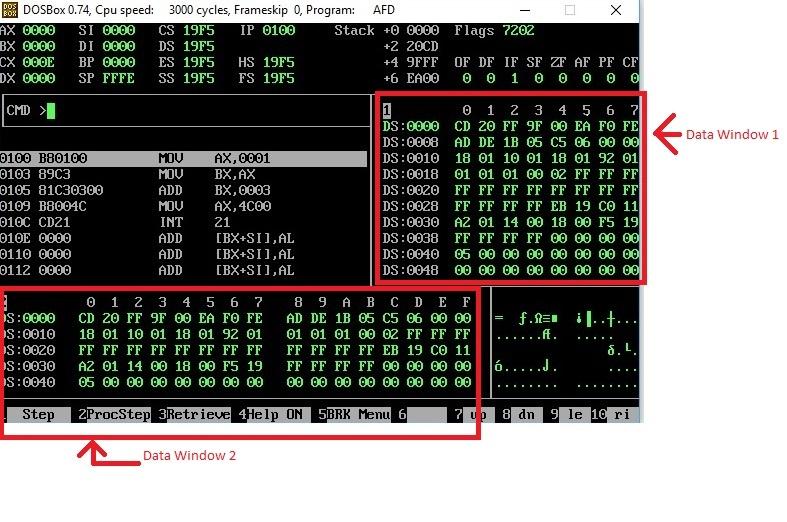
**Data Segment**

**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]

**Activity 1:** Write a program to generate the sum of first five entries of table of 3, using registers, and watch its execution in the debugger.

**Help:** [Approach 1] Can you do this using two registers only? [Approach 2] Can you do this using one register only if we have **add ax, 3** available in our instruction set? Try both of these approaches and watch the first five entries of table of 3 in AX.

**Activity 2 :** Write a program that rotates the value of three registers clockwise twice i.e. given these initial values: ax=10, bx=20, cx=30.

**Help:** First draw data registers on paper. Solve this problem on paper then write your code accordingly. After writing your code verify the execution on paper first then check the execution on AFD. You may use DX.

**Activity 3:** 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.

**Activity 4:** Develop an assembly program that reads 1 number each from 5 different data labels to a register and stores their sum in some other memory location labeled as **result**.

**Post Lab Activities**

**Activity 5:** 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 (quotient) at different memory locations labeled as “**mresult**” and “**dresult**”.

**Activity 6:** Write a program to generate first 10 terms of the Fibonacci Series. The generated terms are to be placed at memory location named “**Fib**” using indirect addressing