# EEE 332/ CSE 331 Lab 2

Topics to be covered in class today:

* Creating Variables
* Creating Arrays
* Create Constants
* Introduction to INC, DEC, LEA instruction
* Learn how to access Memory.

## Creating Variable:

Syntax for a variable declaration:

name DB value

name DW value

DB - stands for Define Byte.

DW - stands for Define Word.

name - can be any letter or digit combination, though it should start with a letter. It's possible to declare unnamed variables by not specifying the name (this variable will have an address but no name).

value - can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "?" symbol for variables that are not initialized.

## Creating Constants

Constants are just like variables, but they exist only until your program is compiled (assembled). After definition of a constant its value cannot be changed. To define constants EQU directive is used:

name EQU < any expression >

For example:

k EQU 5

MOV AX, k

## Creating Arrays

Arrays can be seen as chains of variables. A text string is an example of a byte array, each character is presented as an ASCII code value (0-255).

Here are some array definition examples:

a DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h

b DB 'Hello', 0

You can access the value of any element in array using square brackets, for example:

MOV AL, a[3]

You can also use any of the memory index registers BX, SI, DI, BP, for example:

MOV SI, 3

MOV AL, a[SI]

If you need to declare a large array you can use DUP operator.

The syntax for DUP:

number DUP ( value(s) )

number - number of duplicate to make (any constant value).

value - expression that DUP will duplicate.

for example:

c DB 5 DUP(9)

is an alternative way of declaring:

c DB 9, 9, 9, 9, 9

one more example:

d DB 5 DUP(1, 2)

is an alternative way of declaring:

d DB 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

## Memory Access

To access memory we can use these four registers: BX, SI, DI, BP. Combining these registers inside [ ] symbols, we can get different memory locations.

|  |  |  |
| --- | --- | --- |
| [BX + SI] [BX + DI] [BP + SI] [BP + DI] | [SI] [DI] d16 (variable offset only) [BX] | [BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8] |
| [SI + d8] [DI + d8] [BP + d8] [BX + d8] | [BX + SI + d16] [BX + DI + d16]  [BP + SI + d16] [BP + DI + d16] | [SI + d16] [DI + d16] [BP + d16] [BX + d16] |

Displacement can be an immediate value or offset of a variable, or even both. if there are several values, assembler evaluates all values and calculates a single immediate value..

Displacement can be inside or outside of the [ ] symbols, assembler generates the same machine code for both ways.

Displacement is a signed value, so it can be both positive or negative.

## Instructions

|  |  |  |
| --- | --- | --- |
| Instruction | Operands | Description |
| INC | REG  MEM | Increment.  Algorithm:  operand = operand + 1  Example:  MOV AL, 4  INC AL ; AL = 5  RET |
| DEC | REG  MEM | Decrement.  Algorithm:  operand = operand - 1  Example:  MOV AL,86  DEC AL ; AL=85  RET |
| LEA | REG,MEM | Load Effective Address.  Algorithm:  REG = address of memory (offset)  Example:  MOV BX, 35h  MOV DI, 12h  LEA SI, [BX+DI] |

## Task 1

Create two arrays of size 5. Load one of the arrays with random numbers of your choice. The second arrays should be kept blank. Copy the contents of the first array into the second array in **reverse** order. You must not use loops to accomplish this task.