# Lab# 04

# Variables in assembly

**OBJECTIVE:**

Variables in assembly. Arithmetic multiplication and division using variables.

**Pre-Lab Exercise**

The variables are an important component for any nontrivial program. They hold data and information for the program. Variables (global) in the 8086 assembly reside in the **Data segment** of the program.

**Assembler directive**

An assembler directive is an English word that appears in assembly language programs. It is not an instruction in assembly language but facilitates the program to complete some specific tasks through assembler. Assembler directives may vary from assembler to assembler, but instructions remain unchanged for a given microprocessor on its specific assembler. Examples of assembler directives are EQU, DB, and DW etc.

**Defining variables, arrays and constants Variables**

The syntax of assembler directives that define variables is given below:

name DB initial\_value

The assembler directive DB in above statement (note that it is not an instruction) stands for "Define Byte". Hence a variable with name ‘name’ is declared with one-byte storage

There are two types of variables in assembly; they are initialized and uninitialized variables.

Initialized variables are defined as

age DB 25

An uninitialized variable is has initial value part as '?'

**Displacement DW ?**

where

DB = define byte (1 byte)

DW = define word (2 bytes)

similarly

DD = define double (2 words) (4 bytes)

DQ = define quad (4 words) (8 bytes)

The syntax for defining variables in assembly is

***var\_name define\_bytes value***

where var\_name is the name of the variable, define\_bytes(DB) is the datatype or number of bytes required for the variable, value is the value assigned to the variable.

**Important Note**

There are some rules that must be followed when selecting the variable name. Variable name should not be an instruction such as AND, XOR, CALL etc. Variable name should not start with symbols such as +, -, % etc., though \_ (underscore can be used as starting character). Variable names should not have space in between. For example, MY VARIABLE is an incorrect variable name. If label name is to be separated for easy reading and understanding, then \_ should be used. For example, MY\_ VARIABLE will be a valid replacement of MY VARIABLE.

**Constant**

To assign a name to a constant, assembler directive EQU (equate) is used. The syntax is:

name EQU constant

For example, the following statement associates the name LF with constant 0AH.

LF EQU 0AH

Now LF could be used in place of constant 0AH which is ASCII value of Line Feed character.

**Using variables**

To use variables defined in the data segment, we first need to initialize the **DS** (data segment) register to point to the correct segment address. The following two instructions do the job,

MOV AX, @DATA

MOV DS, AX

The @DATA constant contains the data segment address for this program. The reason we did not move this constant value directly to DS register is that DS register can get a value from a register but not from a constant. So the instruction MOV DS, @DATA is illegal.

**Program:**

The program first defines variables for the multiplication and division operations. It also declares uninitialized variables for storing remainder and quotient for division operation. Finally outputs the result.

.MODEL SMALL

.STACK 100H

.DATA

NUM1 DB 2

NUM2 DB 4

DIVIDEND DW 17

DIVISOR DB 5

REMAINDER DB ?

QUOTIENT DB ?

.CODE

MAIN PROC

; INITIALIZE THE DATA SEGMENT REGISTER TO USE VARIABLES

MOV AX, @DATA

MOV DS, AX

; MULTIPLY

MOV AL, NUM1

MOV BL, NUM2

MUL BL

; OUTPUT RESULT (PRODUCT)

MOV DL, AL ; RESULT IN AX REGISTER

MOV AH, 2

ADD DL, 48 ; GET ASCII VALUE OF THE RESULTANT DIGIT

INT 21H

MOV AX, DIVIDEND

MOV CL, DIVISOR

DIV CL

; AH HAS REMAINDER, AL HAS QUOTIENT

MOV REMAINDER, AH

MOV QUOTIENT, AL

MOV AH, 2

MOV DL, REMAINDER

ADD DL, 48

INT 21H

MOV DL, QUOTIENT

ADD DL, 48

INT 21H

MOV AH, 4CH ; RETURN CONTROL TO DOS/OS

INT 21H

MAIN ENDP

END MAIN

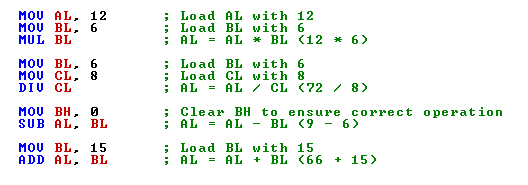
In the above program, there are two new instructions mov ax, @data and mov ds, ax. The DS (Data Segment) register is not initialized when the program loads. These instructions are used to initialize the DS register to the location of Data Segment. The first statement loads the contents of the constant @data into AX register. These contents are the address of the Data Segment. The second statement now initializes the DS register with the address of Data Segment in AX register. The DS cannot directly get @data value, so the constant is first loaded into AX register and then AX register is loaded to DS register.

**In lab Task:**

**Task 1:**

Write assembly language program for the following equations

Ans = 12\*6-6/8+15

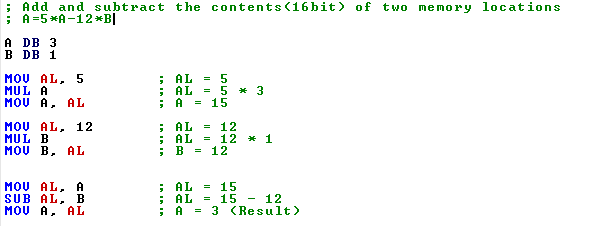


**Task 2:**

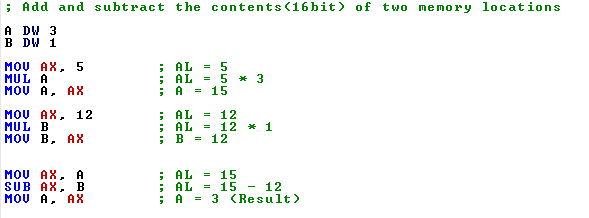
Suppose A and B are two-word variables

A=5\*A-12\*B

Add and subtract the contents(8bit) of two memory locations



Add and subtract the contents(16bit) of two memory locations



Multiply two 16-bit numbers and store the result in some variable at a memory location.

A screenshot of a computer code

Description automatically generated

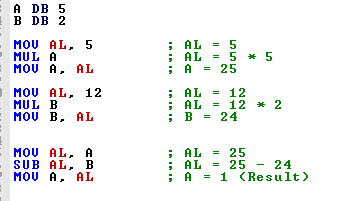
Calculate the average of numbers

A white background with green text

Description automatically generated

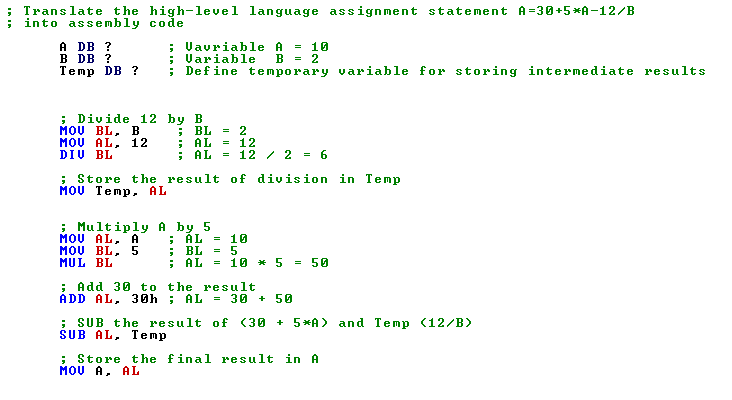
**Task 3:**

Write the same above given program in Emu8086 emulator code editor but with different operand values. Emulate and run the program to verify the outputs.



**Task 4:**

Translate the high-level language assignment statement A=30+5\*A-12/B into assembly code. Let A and B be byte variables and suppose there is no overflow also save and load program.



# Rubric for Lab Assessment

|  |  |  |  |
| --- | --- | --- | --- |
| **The student performance for the assigned task during the lab session was:** | | | |
| Excellent | The student completed assigned tasks without any help from the instructor and showed the results appropriately. | 4 |  |
| Good | The student completed assigned tasks with minimal help from the instructor and showed the results appropriately. | 3 |  |
| Average | The student could not complete all assigned tasks and showed partial results. | 2 |  |
| Worst | The student did not complete assigned tasks. | 1 |  |

**Instructor Signature: \_\_ Date:**