Variables , Arrays and Constants

The programs consist of two types of statements: instructions and directives. The instructions are translated to the machine code by the assembler whereas directives (also called pseudo instructions) are not translated to the machine codes and they generate and store information in the memory. An assembler supports directives to define data, to organize segments, to control procedure, to define macros. Below is how to declare the variables, arrays and constants in the program.

1. ***Variable*** is a memory location. To declare the variable:

variable-name define-directive (or data-type) initial-value

Where, variable-name is the identifier for each storage space. The assembler associates an offset value for each variable name defined in the data segment. There are five basic forms of the define directive (data):

Directive	Purpose	Storage Space
DB	Define Byte	allocates 1 byte
DW	Define Word	allocates 2 bytes
DD	Define Doubleword	allocates 4 bytes
DQ	Define Quadword	allocates 8 bytes
DT	Define Ten Bytes	allocates 10 bytes

The compiler in emu8086 supports two types of variables: **byte** and **word**. The syntax for a variable declaration:

variable -name **db** *initial-value*
variable-name **dw** *initial-value*

variable-name - can be any letter or digit combination, though it should start with a letter.
initial-value - can be any numeric value in any supported numbering system (hexadecimal, binary, or decimal), or "**?**" symbol for variables that are not initialized.

Ex1. To define the variable as byte:

```
x1   db   14h
x2   db   'A'
x3   db   00110001b
x4   db    80
x5   db    ?
```

Ex2. To define the variable as word:

```
y1    dw   14h
y2    dw   'AB'
y3    dw   1111111100110001b
y4    dw   80
y5    dw   ?
```

Let's see example with **MOV** instruction:

```
ORG 100h
JMP start
var1 db 7
var2 dw 1234h
start:
MOV AL, var1      ; AL=07
MOV BX, var2    ;  BX= 1234h
RET   ; stops the program.
```

The offset of var1 is 0102 h and the offset of var2 is 0103h**.**

2.***Arrays***

Arrays can be seen as chains of variables. The syntax to declare an array :

Array_Name Data_Type Values

1. To declare an array of 5 bytes, named as Ar1 and initialize it for example:
   ar1 db 10h,20h,30h,40h,50h

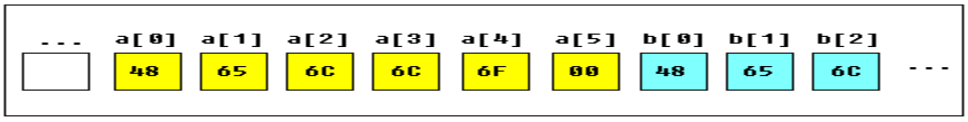
2. To declare an array of 5 word, named as Ar1 and initialize it for example:
   ar2 dw 10h,20h,30h,40h,50h

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

b is an exact copy of the ***a*** array, when compiler sees a string inside quotes it automatically converts it to set of bytes. This chart shows a part of the memory where these arrays are declared:



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]

3. If you need to declare a large array you can use **dup** operator. The syntax for dup:

Array_Name Data_Type Number DUP(?)

Array_Name Data_Type Number DUP(value(s))

Number - number of duplicate to make (any constant value).
value - expression that dup will duplicate.

Ex1. k db 5 dup(9)

Here set 5 bytes of storage in memory and give it the name as A , but leave the 5 bytes uninitialized. Program instructions will load values into these locations. Also the below declaration is an alternative way of it:

k db 9, 9, 9, 9, 9

Ex2. p db 5 dup (1, 2)
Also the below declaration is an alternative way of it:
p db 1, 2, 1, 2, 1, 2, 1, 2, 1, 2

Ex3. m db 5 dup (?)
Also the below declaration is an alternative way of it:
m db 00,00,00,00,00

Ex4. w db 100 dup(?)

Note: dw cannot be used to declare strings.

3.***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
```

The above example is functionally identical to code: MOV AX, 5

Here another example

```
ORG 100H
N   EQU   2*5+1
MOV   DX, N          ; DX = 000B h
Ret
```

****

LEA instruction

The LEA (Load Effective Address) is one of Data copy/Transfer instructions. It is more powerful because it allows you to get the address of an indexed variables. Getting the address of the variable can be very useful in some situations, for example when you need to pass parameters to a procedure. The basic syntax of LEA instruction is :

LEA Reg , Memory

REG is the destination-16 bit, the address of memory location (offset). :AX, BX, CX, DX, SI, DI, BP, SP.

Memory is the source--16 bit, [BX], [BX+SI+7], variable, etc...

Algorithm

- REG = address of memory (offset)

Example:

```
org 100h
LEA   BX ,  [0105h]                ; load offset address      BX = 0105 h
MOV   DI , 3005h                   ;  load offset address     DI = 3005 h
MOV   word ptr[BX+DI] ,  1122h     ; BX+DI= 310A h ; [310A]=22 h ; [310B]=11 h
LEA  SI , [BX+DI]                  ; SI = 310A h
MOV  CX , [BX+DI]                  ; CX = 1122 h
RET
```

Here is another example:

```
ORG 100h
MOV    AL, VAR1          ;  move the value of  VAR1  to AL  (AL=22 h).
LEA    BX, VAR1          ; load offset address    VAR1 in BX  ( BX=010D h).
MOV    byte ptr [BX], 44h    ; modify the contents of VAR1 to 44 h. (VAR1=44).
MOV    AL, VAR1          ; move the value of VAR1  to AL  (AL=44 h).
RET
VAR1   db  22 h
END
```

Both **LEA** instruction and alternative **offset** operator can be used to get the offset address of the variable. The following two instructions have the same functionality(equivalent):

LEA BX, VAR1

MOV BX, offset VAR1

XCHG instruction: Exchange value of two operands.

Algorithm: operand1 < - > operand2

The basic syntax of this instruction: XCHG op1, op2

These types of operands are supported:

	***Operand*1**	***Operand*2**
XCHG	REG	memory
	memory	REG
	REG	REG

Ex1:

```
MOV AL, 5       ; AL = 5
MOV AH, 2       ; AH = 2
XCHG AL, AH   ; AL = 2, AH = 5
XCHG AL, AH   ; AL = 5, AH = 2
```

Ex2: Write the suitable instructions to exchange the content of memory location 0200 with dx register. Assume the content of memory location 0200h=9988h and dx=2233h.

```
MOV word ptr [0200 h] , 9988 h       ; [0200 h] = 88 h and [0201h] = 99 h
MOV DX , 2233 h                      ; DX = 2233 h
XCHG DX, [0200 h]              ; DX = 9988 h and [0200 h] = 33 h and [0201 h] =22 h
```

The instructions can modify flag register

The instructions can modify flag register are **LAHF** and **SAHF**. These instructions haven't operand.

LAHF

Load AH from 8 low bits of Flags register.
Algorithm:
AH = flags register

AH bit: 7 6 5 4 3 2 1 0
[SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
bits 1, 3, 5 are reserved.

SAHF

Store AH register into low 8 bits of Flags register.

Algorithm:

flags register = AH

AH bit: 7 6 5 4 3 2 1 0
[SF] [ZF] [0] [AF] [0] [PF] [1] [CF]
bits 1, 3, 5 are reserved.

Example:

```
MOV   AH,05h
SAHF        ; flag= 05 ; 0000 0101
CLC                    ; 0000 0100
LAHF        ; AH= 04
```