



Variables:

- Declaration:

Variable _Name DB/DW initial_value

- DB = Define Byte

Example:

VAR1 DB 5EH

- DW = Define Word

Example:

VAR2 DW 8DC6h

- DB/DW are variable types

- Variable _Name:

Can be any letter or digit combination, though it **must start with a letter**.

It's possible to declare unnamed variables by not specifying the name using "?" (this variable will have an address but no name)

- Initial_value:

can be any numeric value:

- Decimal number ends with an optional "D"/"d"
- Binary number ends with "B"/"b"
- Hex number ends with "H"/"h" **and must start with a decimal digit**.
Otherwise the assembler would be unable to decide whether the data represents a number or a string. (Ex: 2AH, 5C4H, 1ABCH, 0ABCDH)
- Numbers may have signs
- "?" denotes an uninitialized byte/word

Example 1: Define a byte variable with the value 35H and print the value of the variable.

```
01 .MODEL SMALL
02 .STACK 100H
03 .DATA
04     V1 DB 35H
05 .CODE
06     MOV AX, @DATA
07     MOV DS, AX
08
09     MOV AH, 2
10     MOV DL, V1
11     INT 21H
12
13     MOV AH, 4CH
14     INT 21H
```

```
01 .MODEL SMALL
02 .STACK 100H
03 .DATA
04     V1 DW 54H
05 .CODE
06     MOV AX, @DATA
07     MOV DS, AX
08
09     MOV AH, 2
10     MOV DX, V1
11     INT 21H
12
13     MOV AH, 4CH
14     INT 21H
```

@DATA is the name of the data segment defined by .DATA. The assembler translates the name @DATA into a segment number. Two instructions are needed because a number (the data segment number) may not be moved directly into a segment register.

Creating Constants:

Constants are just like variables, but they exist only until your program is compiled (assembled) because no memory is allocated for constants. After definition of a constant its value cannot be changed. To define constants EQU (equates) directive is used:

constant_name EQU constant_value (numeric value / string)

Example:

k EQU 5

MOV BX, 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).

Example:

```
VAR1 DB 48h, 65h, 6Ch, 6Ch, 6Fh, 00h
VAR2 DB 'Hello', 0
```

- You can access the value of any element in an array using square brackets

Example:

```
MOV AL, VAR1[3]
```

- You can also use any of the memory index registers BX, SI, DI, BP

Example:

```
MOV SI, 3
MOV CL, VAR2[SI]
```

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

The syntax for DUP:

NUMBER DUP (VALUE(S))

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

value - expression that DUP will duplicate.

Example:

c DB 4 DUP(9) ;is an alternative way of declaring: c DB 9, 9, 9, 9

d DB 4 DUP(1, 2) ;is an alternative way of declaring: d DB 1, 2, 1, 2, 1, 2, 1, 2

Example 2: Define a byte variable with initial value 65h and a string "HELLO!". Print those.

```
02 .MODEL SMALL
03 .STACK 100H
04 .DATA
05     V1 DB 65h
06     S DB "HELLO!$"
07 .CODE
08     MOV AX, @DATA
09     MOV DS, AX
10
11     MOV AH, 2
12     ;DISPLAY THE VARIABLE
13     MOV DL, V1
14     INT 21H
15
16     ;PUT A SPACE AFTER VARIABLE
17     MOV DL, 20H
18     INT 21H
19
20     ;PRINT THE STRING
21     ;USING FUNCTION# 9
22     LEA DX, S
23     MOV AH, 9
24     INT 21H
25
26     MOV AH, 4CH
27     INT 21H
28
```

INT 21h, function 9, expects the offset address of the character string to be in DX. To get it there, we use a new instruction:

LEA Destination, Source

where destination is a general register and source is a memory location. LEA stands for "Load Effective Address." It puts a copy of the **source offset address** into the destination.

Example:

LEA DX, MSG1

puts the offset address of the variable MSG into DX.

Instruction	Operands	Descriptions
LEA	REG, MEM	Load Effective Address. Algorithm: REG = address of memory (offset)

Example 3: create an array of size 3 and load the array with user input data.

```
01  
02 .MODEL SMALL  
03 .STACK 100H  
04 .DATA  
05     A DB 3 DUP (?), '$'  
06 .CODE  
07     MOV AX, @DATA  
08     MOV DS, AX  
09  
10     MOV AH, 1  
11  
12     ;TAKE INPUT  
13     INT 21H  
14     MOV A[0], AL  
15  
16     INT 21H  
17     MOV A[1], AL  
18  
19     INT 21H  
20     MOV A[2], AL  
21  
22     MOV AH, 4CH  
23     INT 21H  
24  
25
```

```
01  
02 .MODEL SMALL  
03 .STACK 100H  
04 .DATA  
05     A DB 3 DUP (?), '$'  
06 .CODE  
07     MOV AX, @DATA  
08     MOV DS, AX  
09  
10     MOV AH, 1  
11  
12     ;SI HOLDS THE OFFSET  
13     ;ADDRESS OF A  
14     LEA SI, A  
15     ;TAKE INPUT  
16     INT 21H  
17     MOV [SI], AL  
18  
19     INT 21H  
20     MOV [SI+1], AL  
21  
22     INT 21H  
23     MOV [SI+2], AL  
24  
25     MOV AH, 4CH  
26     INT 21H  
27
```