DISK OPERATING SYSTEM (DOS)

Disk Operating System (DOS) or Microsoft Disk Operating System (MS-DOS) is a system software, which is closely associated with the computer hardware and provides the interface between the user and resources of the computer, such as central processing unit, memory, files and I/O devices. In simple words, we can say that it is a program which supervises and controls the operation of a computer.

As **DOS** was written by Microsoft Corporation, usually it is called **MS-DOS. IBM** has been licensed to use and sell the same **DOS** with their computer. In that case, when it marketed by **IBM,** the **DOS** is called **PC-DOS,** there are many operating systems, like **UNIX, OS/2, VMS, etc.,** but **DOS** is the most popular operating system.

Knowledge about **DOS** is very essential for computer users, otherwise it is not possible to use the computer efficiently. It is not essential to have a thorough knowledge about **DOS,** some of the basic commands will be sufficient for most of the purposes. The commands are discussed later. The recent versions of **MS-DOS** contain extensive online help. You can refer to it any time by typing **HELP** at the command prompt.

What is a Directory

- A directory is a list of file which is itself a file stored in the computer's memory so that users can reference it as it is required.
- Also called a catalog of files

Root Directory

- The Root directory is that directory that is automatically created when the disk is formatted
- It is the current drive that we have been working

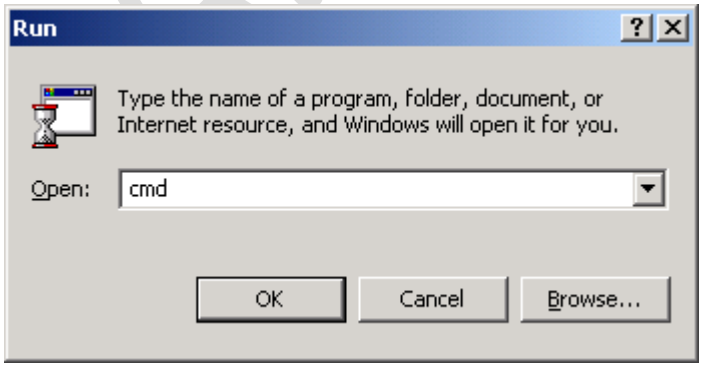
Example:

A:\ , B:\ (for Floppy Disk Drive)

C:\ , D:\ etc. (for Hard Disk Drive)

Getting into MS DOS

- To start MS DOS, click on the Start button and choose **Run**
- In the Run dialog box, type **CMD** or **COMMAND** which is the EXE file for MS DOS
- Click on the **OK** button



Types of MS-DOS commands :

There are two types :

1. Internal command
2. External command

Internal command : are the simplest , most commonly used command when you list the directory on your MS-DOS disk, you cannot see these commands because they are part of a file named (command.com). These commands were loaded into your computer's memory when you started MS-DOS.

Ex: copy , prompt ,mkdir , ren , type , time , chdir , del , ver ,cls , dir , rmdir , vol , path ,date.

External command: any filename with an extension of **.com , .exe** ,or **.bat** is considered an external command before MS-DOS can run external commands, it must read them into memory the disk. When you give an external command, MS-DOS immediately checks your working directory to find command.

Ex: label , discopy , format , more , sys , sort , print , graphics , comp , attrib ,tree.

MS-DOS commands:

1. **A:\>C:** to transfer from drive A to drive C.

   C:\>

2. **CLS** : clear the screen.

   C:\> CLS

3. **VER** : print the MS-DOS version number.

   C:\> VER

4. **VOL** : displays the volume label.

   C:\> VOL

   MS-DOS version 3.3

5. **DATE**: displays and sets the dates
   C:\>DATE

   Current date is sat 9-27-2018

   Enter new date (mm-dd-yy)

TIME : displays and sets the time

C:\>TIME
Current date is 10:45:30:86a

6. **Dir** : This command displays a list of files and subdirectories with their size in bytes and the time and date of last modification that all in the directory you specify.

   Syntax:

   Dir [drive] [Path] [File name] [/P] [/W]

   Example: dir C:\ws7\macros\student.exe **Displays** the file **student.exe** in the subdirectory 'MACROS of WS7 directory which is in drive **C. /P** and **/W** are called switches.
   /P displays one screen of listing at a time. To see next screen press any key.
   /w displays the listing in wide format with as many as five files/directory names on each line.
   /s
   /a

Using Wildcards

The symbol* (asterisk) and ? (question mark) are called wildcards in **DOS.** You can use them to display a listing of subsets of files and subdirectories. The following examples illustrate the use of wildcards.

Example:

dir C:\ws7*.txt

displays all the files and subdirectories with extensions, .txt in the parent directory 'ws7' in 'C' drive.

Another examples:

C:\>dir
C:\>dir *.*
C:\>dir filename.*
C:\>dir *.exe
C:\>dir *m.*
C:\>dir MS-DOS*.com
C:\>dir MS-DOS\?or?

7. **MD (Make Directory)*:*** Using this command, you can create a directory or a subdirectory under a parent directory.

Syntax:

MD [drive:] [Path] [directory name] [drive:]

Specifies the drive on which you want to create a 'new directory'.

[Path] [directory name]

Specifies the name and location of the new directory.

C:\>md Mosul
C:\>md \ Mosul \ Bagdad

8. CD

CD or **CHDIR,** displays the name of the **C**urrent **D**irectory or **C**hanges the current **D**irectory (**C**hanges **D**irectory) to the specified directory.

Syntax :

1. CD [drive] [Path]
   in case of 1, the directory will be changed to the one you specify

   Example:
   C:\> cd Mosul
   C:\ Mosul> cd Bagdad
   C:\ Mosul \ Bagdad>

2. Cd ..
   in case 2 you will be changing back to parent directory from subdirectory.

   Example:
   C:\ Mosul \ Bagdad> cd..
   C:\ Mosul >

3. Cd\
   In case 3, you will be changing the root directory from current directory. The root directory is the top of the directory hierarchy for a drive.

   Example:
   C:\ Mosul \ Bagdad> cd\
   C:\>

9. **RD (Remove Directory):** Using this command, you can delete (or remove) a directory provided the directory is empty, i.e., it does not contain any files in it.

Syntax:

RD [drive:] [Path] [directory name]

The components of this syntax has the same meaning as in MD.

Examples:
To remove a directory named (Mosul)
You must ensure that the directory is empty then you can deleted it:

C:\>cd Bagdad

C:\ Bagdad \ > cd Mosul

C:\ Bagdad \ Mosul> dir

C:\ Bagdad \ Mosul> cd ..

C:\ Bagdad \ > rd Mosul

Note : remember that if you are working in the same directory that you are trying to remove, you will receive the error message.

11.Copy*:* Copy one or more files to the location you specify.
Syntax:

Copy [source] [destination] or

Copy [drive:] [Path] [directory name]

Source: Specifies the location and name of a file or a set of files from which you want to copy.

For example, if you want to copy a file named readme.txt under 'ws7' directory in drive C: you can specify it as follows.

C:\ws7\readme.txt

destination: Destination specifies the location and name of a directory or file to which you want to copy.
For example, if you want to copy the above file, i.e., **readme.txt** to say, a directory named **txtfles** in drive **D:** you can specify it as, **D:\txtfles.** Thus, in order to copy a file from one directory (source) to another directory (destination) as indicated above, you will have to specify it as

Copy C:\ws7\readme.txt D:\txtfles

You can also use wildcards to copy a group of files from one directory to another. If you are copying from the current directory, you need not specify the source.

Another Examples:

1. copy r1.bat c:\Bagdad\Mosul
2. copy Ali.txt Omar.doc
3. copy D:*.* E:\AAA

12. Ren or rename

Changes the name of the files you specify.

Syntax:

REN [Drive:] [Path] [Filename1 Filename2]

You can also make use of wildcards to change the name of a group of files that matches your specification You cannot change name of a directory or subdirectory and move the files across the drives using the command.

Examples:

Ren ali.bas ahmad.txt
Ren D:\Mosul\ ali.bas C:ahmad.txt

10. DEL or **ERASE:** This command deletes the files you specify.
Syntax:

DEL [Drive:] [Path] [Filename] [/P]

[Drive:] [Path] [Filename]
specifies the location and name of the file or set of files you want to delete [/P], Prompts (asks) you for confirmation before deleting the specified file. You can also make use of wildcards to delete more than one file from the directory (location) you specify at a time.

Examples:

1. Del Ali.txt
2. Del D: *.*
3. Ali.*
4. Del re??.doc

DEBUG, Register and Memory Commands

The DEBUG program is a program-execution/debug tool that operates in the IBM-compatible PC's disk operating system (DOS) environment. The DEBUG program is found in any PC working with the 8088/8086 microprocessor's family. The DEBUG program, which is part of the PC's disk operating system (DOS), helps us to load, assemble, execute, test, and debug programs written in assembly or machine code.

Moreover, it gives us the ability to directly access the internal registers and memory locations of the microprocessors and make any necessary modifications to these registers or memory locations. The keyboard is the input unit of the debugger and permits the user to enter command to lead data such as the machine code of a program; examine or modify the state of the microprocessors internal registers; or execute a program. All we need to do is to type in the command and then press the enter key. These debug commands are the tools a programmer needs to enter, execute and debug programs. In this experiment, you will study DEBUG commands that can be used to examine and modify the content of microprocessor registers and memory. The ability to examine modify the content of registers and memory is essential for debugging programs. The complete command set of the DEBUG program can be found in Table 1. It is very important to refer to this table whenever you need information about the command set of the DEBUG program.

Running the DEBUG program

1. The fastest way to lunch the debug program is to use the Run command. To do so, press Win button and R button (Win + R), then type cmd or cmd.exe .

   Or go to start menu, and type cmd in search program and files.

2. This will open window working in the DOS environment , then go to root directory (e.g c:\ ) using DOS command (cd\ ↲)
3. Type the command debug (debug ↲)

DEBUG is then executed and its prompt (-) is displayed waiting for accepting commands.

Debug commands

A) Memory and Register Manipulation Command

1. R (Register command):

It is used to examine or modify the contents of registers.

Ex1: Display the content of all registers.

-R

```
AX=0000  BX=0000  CX=0446  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=6897  ES=6897  SS=6897  CS=6897  IP=0100   NV UP DI PL NZ NA PE NC
6897:0100 E96B01       JMP     026E
```

Note : if you have a file , CX contains the length of this file (0446h or 1094d). If the file were larger than 64K, BX would contain the high order of the size. This is very important to remember when using the Write command, as this is the size of the file to be written. Remember, once the file is in memory, DEBUG has no idea how large the file is, or if you may have added to it. The amount of data to be written will be taken from the BX and CX registers.

If we want to change one of the registers, we enter R and the register name.

- **R [register name]**

Ex2: Display the content of AX register then modify it with the new value 1234 h

```
-  R  AX          R and AX register
   AX  0000       Debug responds with register and contents
: 1234          : is the prompt for entering new contents. We respond 1234
-                 Debug is waiting for the next command.
```

Now if we display the registers, we see the following:

```
AX=1234  BX=0000  CX=0446  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=6897  ES=6897  SS=6897  CS=6897  IP=0100   NV UP DI PL NZ NA PE NC
6897:0100 E96B01       JMP     026E
```

Note that nothing has changed, with the exception of the AX register. The new value has been placed in it, as we requested. One note. The Register command can only be used for 16 bit registers (AX, BX, etc.). It cannot change the 8 bit registers (AH, AL, BH, etc.). To change just AH, for instance, you must enter the data in the AX register, with your new AH and the old AL values.

2. D (Dump command):

It is used to display the contents of the various memory locations. It is mostly used to display data (text, flags, etc.). If we enter the Dump command at this time, DEBUG will default to the start of the program. *It uses the DS register as it's default*.

It will by default display 80h (128d) bytes of data (means one block), or the length you specify.

There are two form of this command:

i). First form:

The syntax is:

- D address

It will by default display the contents of the 128 bytes (80 h) (means one block) consecutive memory locations from the determined address.

Note: one block = 128 byte (0-127 byte) decimal
= 80h (00-7f h)

Ex1: display the content of memory locations (for one block) starting at 0100 h.

D 0100

or

D Ds:0100

Assume we will see this:

```
6897:0100  E9 6B 01 43 4C 4F 43 4B-2E 41 53 4D 43 6F 70 79   ik.CLOCK.ASMCopy
6897:0110  72 69 67 68 74 20 28 43-29 20 31 39 38 33 4A 65  right (C) 1983Je
6897:0120  72 72 79 20 44 2E 20 53-74 75 63 6B 6C 65 50 75   rry D. StucklePu
6897:0130  62 6C 69 63 20 64 6F 6D-61 69 6E 20 73 6F 66 74  blic domain soft
6897:0140  77 61 72 65 00 00 00 00-00 00 00 00 00 00 00 00  ware............
6897:0150  00 00 00 00 00 00 00 00-00 24 00 00 00 00 00 00   .........$......
6897:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
6897:0170  00 00 00 00 00 00 00 00-00 00 00 00 44 4F 53 20   ............DOS
```

Notice that the output from the Dump command is divided into three parts:

a) On the left, we have the address of the first byte on the line. This is in the format Segment:Offset.

b) Next comes the hex data at that location. Debug will always start the second line at a 16 byte boundary; that is, if you entered D 109, you would get 7 bytes of information on the first line (109-10F), and the second line would start at 110. The last line of data would have the remaining 9 bytes of data, so 80h bytes are still displayed.

c) The third area is the ASCII representation of the data. Only the standard ASCII character set is displayed. Special characters for the IBMPC are not displayed; rather periods (.) are shown in their place. This makes searching for plain text much easier to do.

Ex2: display the content of memory locations (for one block) starting at 200 from CS.

D CS:200

ii). Second form:

The syntax is:

- **D start address end address**

End address = start address + no. of bytes -1

Ex3 : display the content of two bytes of memory location at offset 300.

D 300 301

Ex4 : display the content of 13 bytes of memory location from ES started at FD00.

D ES:FD00 FD0C

3.E (Enter Command) :

It is used to examine and then modify or enter bytes (data) at various memory locations. You must supply a starting memory location where the values will be stored. The syntax is :

E [ADDRESS] enter new byte value at address

E [ADDRESS] [DATA] replace the contents of one or more bytes starting at the specified address, with the value contained in the DATA

Ex1: write a command that examine the previous content of location 0100 and modify it with the ASCII code of "A".

To examine the previous content of location 0100 :
-E 0100

In this example content of location 0100 equals FF:
0100 FF._

To loads the ASCII code of "A" in memory location 0100 :
-E 0100 "A"

Ex2: write a command that will load five bytes in memory locations starting at address 3000 h (with value 1F 2F 3F 4F 5F).

E 3000 1F 2F 3F 4F 5F

After execution the value at 2000 will be 1F, 2001 will be 2F …etc

To display these bytes:

D 3000 3004

Ex3: write a command that will load 11 bytes in memory locations (from ES) that start at address 21BC h (with value 11 12 13 14 15 16 17 18 19 1A 1B).

E ES:21BC 11 12 13 14 15 16 17 18 19 1A 1B

To display these bytes:

D ES:21BC 21C6

4.F (Fill Command)

It is used to fill consecutive memory locations all with the same value. The syntax is:

F [STARTING ADDRESS] [ENDING ADDRESS] [DATA]

Ex: write a command that will load 32 bytes in memory locations (from ES) starting at address 0100 (with value 50).

-F 0100 011F 50

5. M (Move Command)

It is used to copy a block of data from one part of memory location (source location (S.)) to another (destination (D.)). The syntax is:

M [Starting Address of S.] [Ending Address of S.] [Starting Address of D.]

Ex1 : Copy block of 24 bytes of memory location starting at address 0100 to another starting at address 0200.

-M 0100 0117 0200

-D 0200 0217

Ex2 : Copy one block (at offset 200) to another block at offset 300.

-M 200 27F 300

-D 200 27F or D 200

-D 300 37F or D 300

6.C (Compare Command)

Comparing the contents of two blocks of data to determine if they are or are not the same, the contents of corresponding address locations in each block are compared to each other. Each time unequal elements are found, the address and the content of that byte in both blocks are displayed. The syntax is:

C [Starting Address] [Ending Address] [Destination Address]

Ex : Compare block of 32 bytes of memory location starting at address 0100 with another starting at address 0200.

-C 0100 011F 0200

7.S (Search Command) Scan through a block of memory to determine whether or not it contains specific value. The result is the memory locations containing that value. The syntax is :

S [Starting Address] [Ending Address] [Data]

Ex : Scan through 5 bytes of memory location starting at address 0700 if they contain the value 04.

-S 0700 0704 04

B) Program Creation / Debugging Commands

1. A (Assemble Command)

Assemble the instruction of program one after another into machine code, and store them in memory. The syntax is:

A [STARTING ADDRESS]

-A 0100
13D7:0100 MOV AX, 02
13D7:0103

2. U (Unassemble Command

Converting machine code instructions to their equivalent assembly language source code statement. The syntax is;

U [STARTING ADDRESS] [ENDING ADDRESS]

Ex: Unassembled 4 bytes beginning at the address 0100.

-U 0100 0103

3. T (Trace Command)

Executed the one or more instructions at a time. It also called single stepping mode. The syntax is:

T [STARTING ADDRESS] [NUMBER]

NUMBER: number of instruction to be executed.

Ex(1): Execute 3 instructions at a time. Starting at address 0100.

-T 0100 3

Ex(2): Execute 1 instruction at a time. Starting at address 0100.

-T 0100

C) Miscellaneous Command

1. H (Hexadecimal Addition and Subtraction Command)

It is used to perform addition and subtraction of two hexadecimal numbers with a single command. The syntax is:

-H [NUM1] [NUM2]

-H 0a 06

The result 0010 0004

2. Q (Quit Command)

It is used to quit the DEBUG.

-Q

write a program that will be add two byte (E1,D2) these word stored in memory location starting as f900 .then store the result in memory location

Sol:

```
Mov  byteptr [f900],E1

Mov  byteptr[f901],d2

Mov  al,[f900]

Add  al,[f901]

Mov  [f902],al

Mov  byteptr[f903],0

Adc  byteptr[f903],0
```

Write a program that will be add two word (9034,d2c1) these word stored in memory location starting as 6000 .then store the result in memory location

Sol:

```
Mov  wordptr [6000],9034

Mov  wordptr[6002],d2c1

Mov  ax,[6000]

Add  ax,[6002]

Mov  [6004],ax

Mov  byteptr[6006],0

Adc  byteptr[6006],0
```

Write program that will be subtract the content of memory location 0801 from location 0800 (are byte) then store the result also in memory location ?

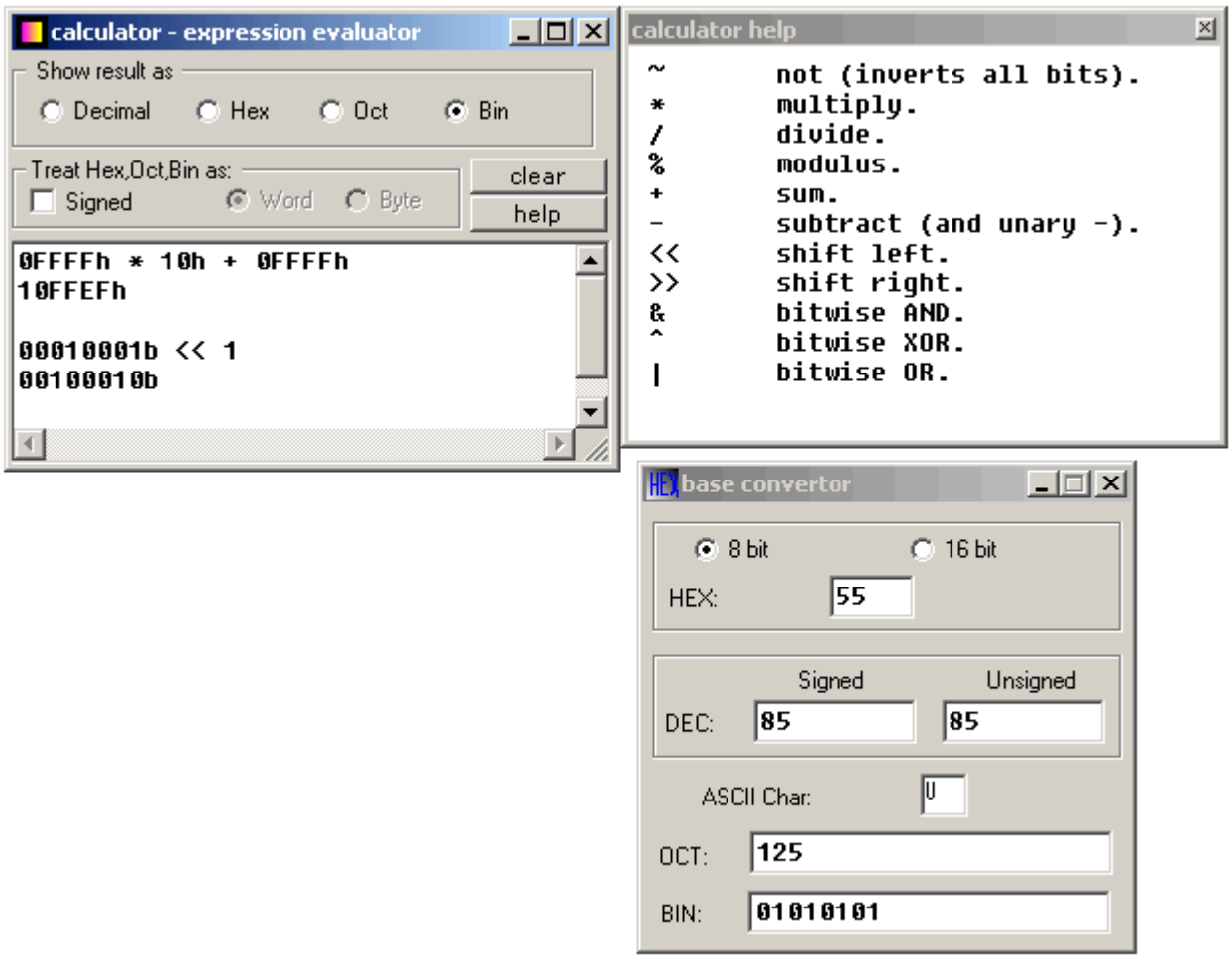
Note: the content of 0800=7f ,0801=8E

Sol/

```
Mov  byte ptr[0800] ,7fh

Mov byte ptr[0801],8Eh

Mov al,[0800]

Sub al,[0801]

mov [0802],al
```

Emulator (emu8086)

There are some handy tools in emu8086 to convert numbers, and make calculations of any numerical expressions, all you need is a click on **Math** menu:



Base converter allows you to convert numbers from any system and to any system. Just type a value in any text-box, and the value will be automatically converted to all other systems. You can work both with **8 bit** and **16 bit** values.

Multi base calculator can be used to make calculations between numbers in different systems and convert numbers from one system to another. Type an expression and press enter, result will appear in chosen numbering system. You can work with values up to **32 bits**. When

Signed is checked evaluator assumes that all values (except decimal and double words) should be treated as **signed**. Double words are always treated as signed values, so **0FFFFFFFFh** is converted to **-1**.
For example you want to calculate: 0FFFFh * 10h + 0FFFFh (maximum memory location that can be accessed by 8086 CPU). If you check **Signed** and **Word** you will get -17 (because it is evaluated as (-1) * 16 + (-1) . To make calculation with unsigned values uncheck **Signed** so that the evaluation will be 65535 * 16 + 65535 and you should get 1114095.
You can also use the **base converter** to convert non-decimal digits to signed decimal values, and do the calculation with decimal values (if it's easier for you).

These operation are supported:

~ not (inverts all bits).
* multiply.
/ divide.
% modulus.
+ sum.
- subtract (and unary -).
<< shift left.
>> shift right.
& bitwise AND.
^ bitwise XOR.
| bitwise OR.

Binary numbers must have "**b**" suffix, example: 00011011b

Hexadecimal numbers must have "**h**" suffix, and start with a zero when first digit is a letter (A..F), example: 0ABCDh

Octal (base 8) numbers must have "**o**" suffix, example: 77o

Memory Access

In order to say the compiler about data type, these prefixes should be used:

byte ptr - for byte.
word ptr - for word (two bytes).

for example:

```
byte ptr [BX]     ; byte access.
   or
word ptr [BX]     ; word access.
```

Emu Assembler supports shorter prefixes as well:

b. - for **byte ptr**
w. - for **word ptr**

in certain cases the assembler can calculate the data type automatically.

Variables

Variable is a memory location. For a programmer it is much easier to have some value be kept in a variable named "**var1**" then at the address 5A73:235B, especially when you have 10 or more variables.

Our compiler supports two types of variables: **BYTE** and **WORD**.

Syntax for a variable declaration:

name **DB** *value*

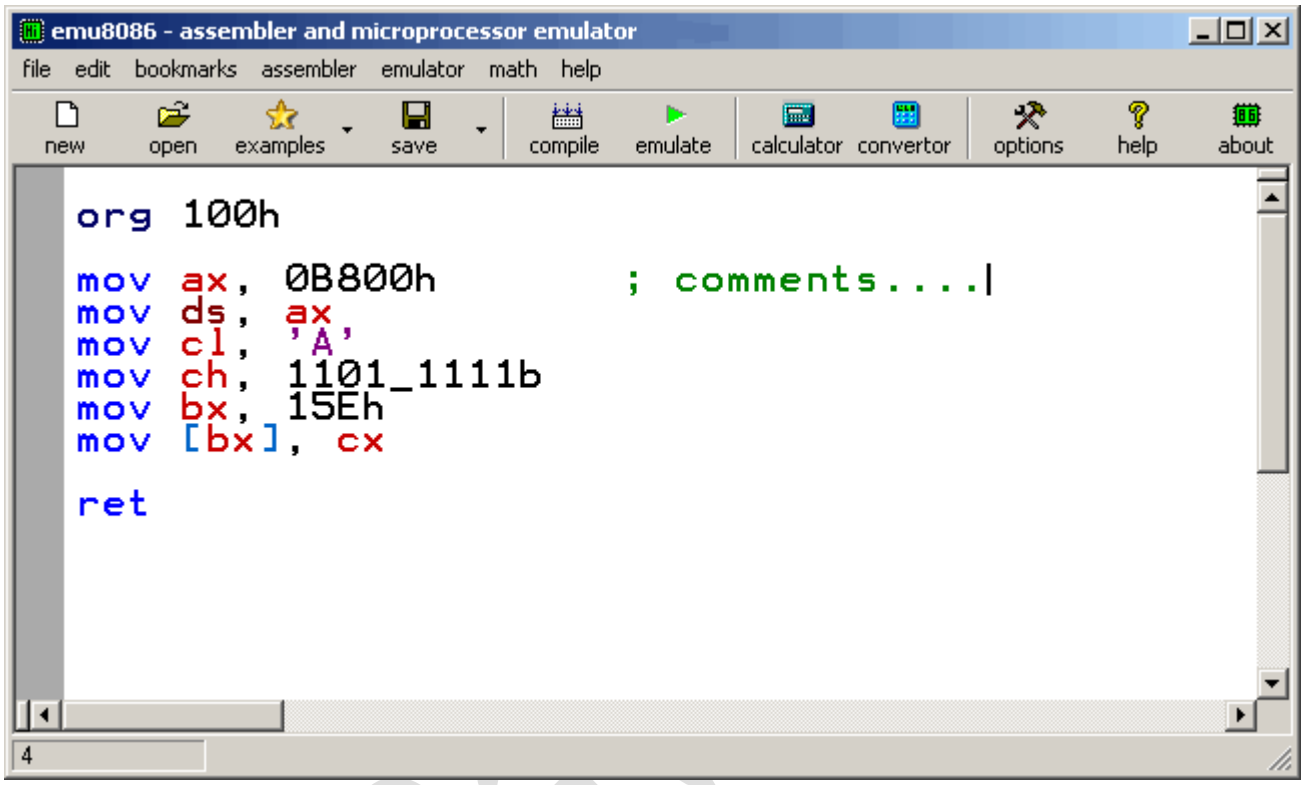
name **DW** *value*

DB - stays for Define Byte.
DW - stays 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.

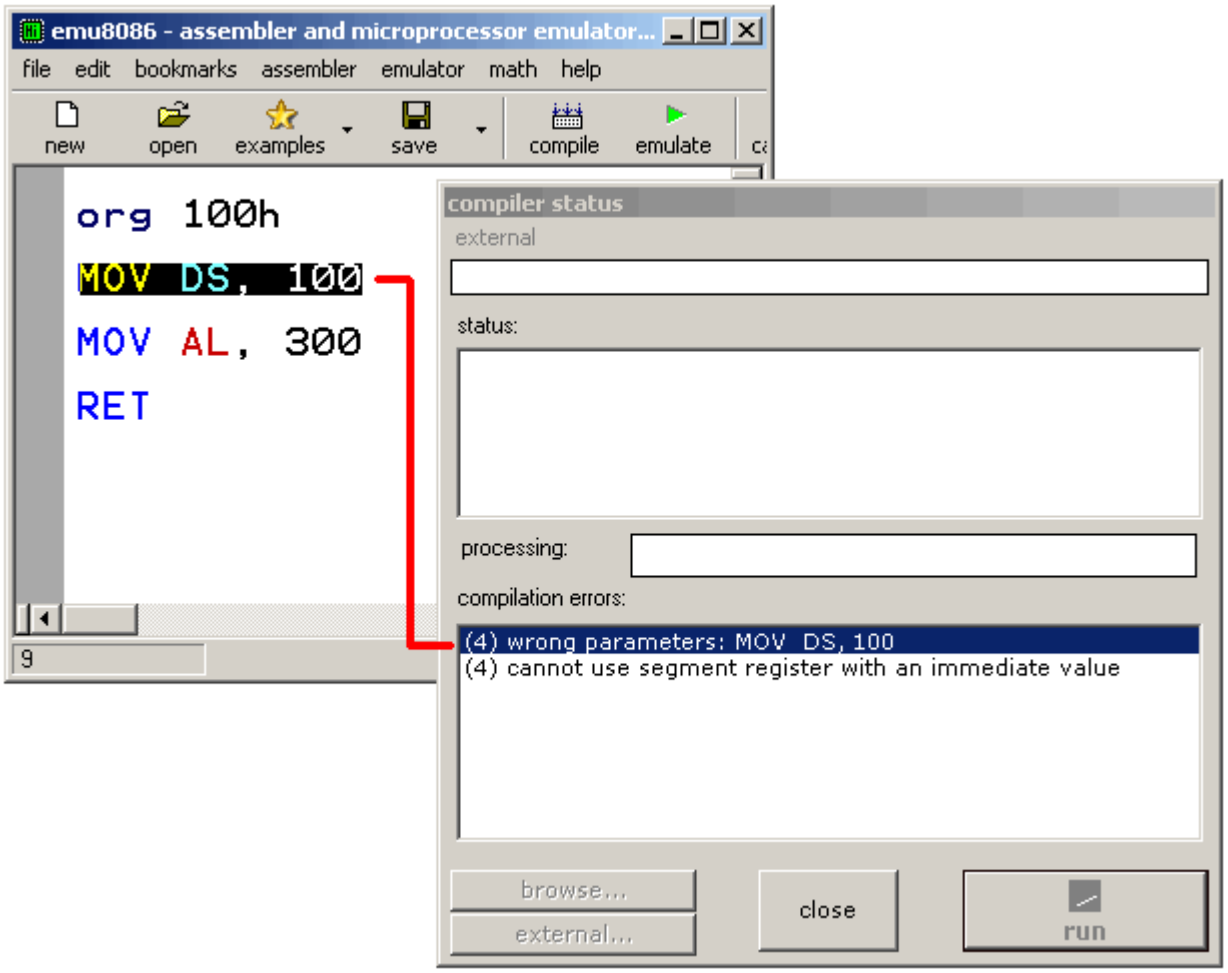
Compiling the assembly code



Type your code inside the text area, and click **compile** button. You will be asked for a place where to save the compiled file.
After successful compilation you can click **emulate** button to load the compiled file in emulator.

Error processing

assembly language compiler (or assembler) reports about errors in a separate information window:



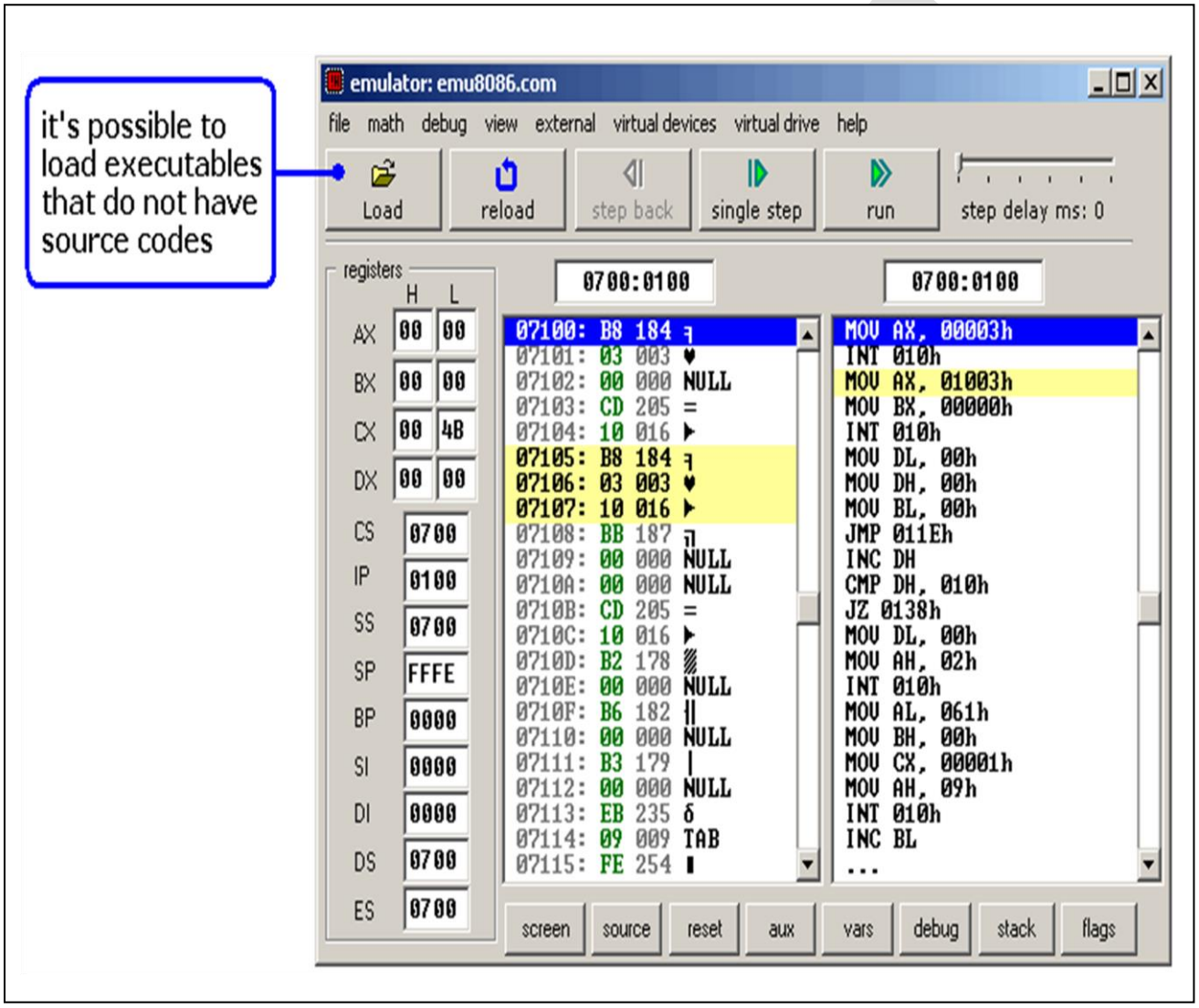
MOV DS, 100 - is illegal instruction because segment registers cannot be set directly, general purpose register should be used, for example
MOV AX, 100
MOV DS, AX

MOV AL, 300 - is illegal instruction because **AL** register has only 8 bits, and thus maximum value for it is 255 (or 11111111b), and the minimum is -128.

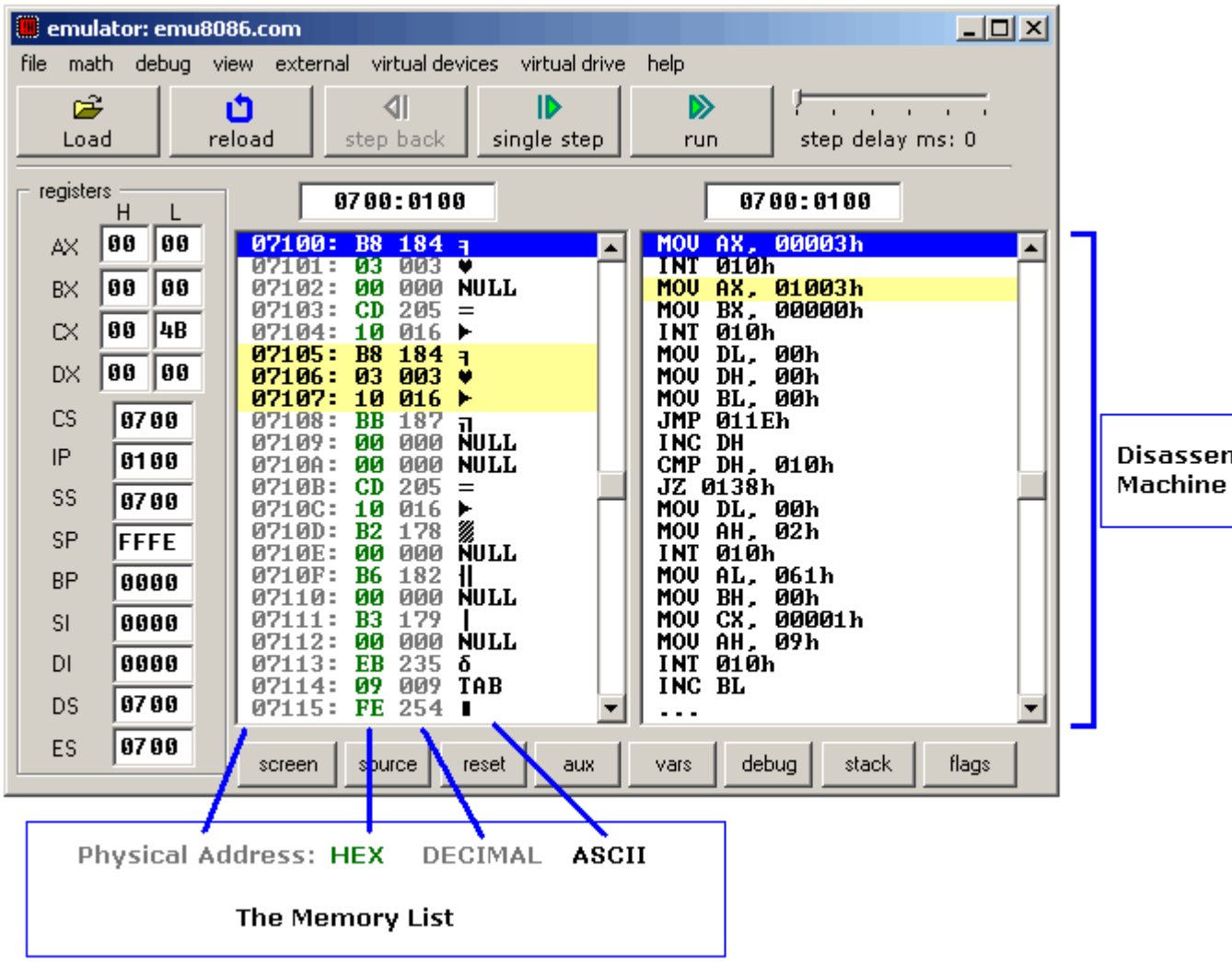
Using the emulator

If you want to load your code into the emulator, just click "**Emulate**" button .

The emulator can load executables created by other assemblers as well by select **Show emulator** from the **Emulator** menu.



And then loading files from "**MyBuild**" or any other folder. So if there are no files in "**MyBuild**" folder return to source editor, select *Examples* from *File* menu, load any sample, compile it and then load into the emulator:



[**Single Step**] button executes instructions one by one stopping after each instruction.

[**Run**] button executes instructions one by one with delay set by **step delay** between instructions.

Double click on register text-boxes opens "**Extended Viewer**" window with value of that register converted to all possible forms. You can modify the value of the register directly in this window.

Double click on memory list item opens "**Extended Viewer**" with WORD value loaded from memory list at selected location. Less significant byte is at lower address: LOW BYTE is loaded from selected position and HIGH BYTE from next memory address. You can modify the value of the memory word directly in the "**Extended Viewer**" window,

You can modify the values of registers on runtime by typing over the existing values.

[**Flags**] button allows you to view and modify flags on runtime.

Some programs:

write a program in assembly language to subtract two bytes, these bytes stored in memory locations 0800 and 0801.

Sol:

```
Mov   al, byte ptr[0800h]
Sub al,[0801h]
mov [0802h],al
ret
```

write a program in assembly language to solve the following equation:

z=x-y ,x=99 ,y=49

Sol:

```
org 100h
jmp start
x   db  99h
y   db  49h
start:
mov bl,x
mov bh,y
sub bl,bh
ret
```

Multiply

1.Byte*byte

Write a program to multiply the content of memory location 0200(as byte) with the content of memory location 0201 (as byte) then store the result also in memory location .assume the content 0200=f1,0201=02.

Sol:

```
Mov b.[0f900h],f1h

Mov b.[0f901h],02h

Mov al,[0f900h]

Mul b.[0f901h]

Mov [0f902h],ax
```

2.Word*word

Write a program to multiply the content of memory location 0200(as word) with the content of memory location 0201 (as word) then store the result also in memory location , assume the content 0200=3f11,0202=0809

```
Mov  w.[0200h],3f11h

Mov  w.[0202h],0809h

Mov ax,[0200h]

Mul  w.[0202h]

Mov [0204h],ax

Mov [0206h],dx
```

3.Word* byte

Write a program to multiply the content of memory location 0200(as byte) with the content of memory location 0201 (as word) then store the result also in memory location .assume the content 0200=50h,0201=0503h

Sol:

```
Mov  b.[ 0200h],50h

Mov  w.[ 0201h],0503h

Mov al,[ 0200h],

Mov ah,00h

Mul  w.[ 0202h]

Mov [0203h],ax

Mov [0205],dx
```

Signed Word* byte

Write a program to multiply the content of memory location 0200(as signed byte) with the content of memory location 0201 (as signed word) then store the result also in memory location .assume the content 0200=50h,0201=0503h

Sol:

```
Mov  b.[ 0200h],50h

Mov  w.[ 0201h],0503h

Mov al,[ 0200h], ; if  bit 7 of al =0 then ah =00,if bit 7 of al=1 then ah=ff

Cbw

IMul  w.[ 0202h]
```

Mov [0203h],ax

Mov [0205],dx

Write a program to copy an array (src) of 5 elements (as word) to another array (dst) .

Sol:

```
org 100h
jmp start
src  dw 100,102,500,600,700
dst dw ?, ?, ?, ?, ?
start:
lea si, src
lea di, dst
mov cx, 5
next:
mov ax,[si]
mov [di],ax
add si,2
add di,2
loop next
ret
```

Write a program to find the summation and average of an array (arr) of 5 elements (as bytes) . Store the result in the variables (summation) and (average).

```
org 100h

jmp start

length equ 5

arr db 100, 95, 100, 80, 100

summation dw ?

average db ?

start:

lea bp, arr

mov cx, length

mov bx, 0

next:

add bl, [bp]

adc bh, 0

inc bp

loop next

mov summation, bx

mov ax, summation

mov bl, length

div bl

mov average, al

ret
```

Write a program to count the numbers of even and odd number in an array (arr) of 10 elements (as word).

```
org 100h

jmp start

length equ 10

arr  dw  1,2,3,4,5,6,7,8,9,11

even db 0

odd db 0

start:

lea bp, arr

mov cx, length

next:

test w. [bp], 1

jz even_no

inc odd

jmp skip

even_no:

inc even

skip:

add bp, 2

loop next

ret
```