Emu8086

- **Open emu8086 (Figure 1)**

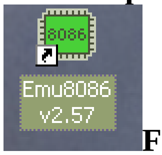


Figure 1

- **Click Continue… (Figure 2)**

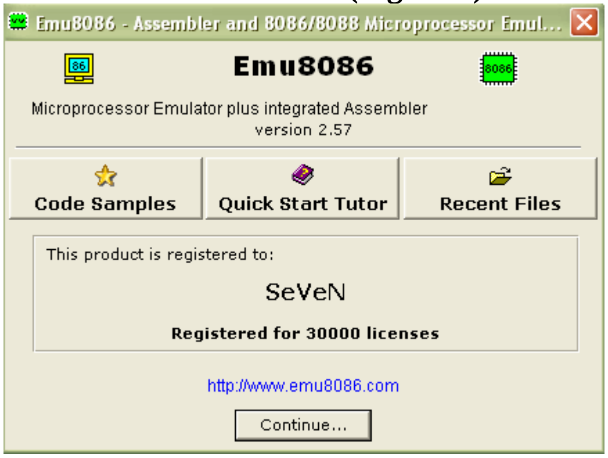

Emu8086 - Assembler and 8086/8088 Microprocessor Emul...
Emu8086
86
8086
Microprocessor Emulator plus integrated Assembler
version 2.57
Code Samples
Quick Start Tutor
Recent Files
This product is registered to:
SeVeN
Registered for 30000 licenses
http://www.emu8086.com
Continue...

Figure 2

- **From Tool Bar choose "NEW" (Figure 3)**

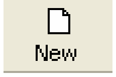


Figure 3

- **Choose COM or EXE Template**

A) COM File (Figure 4)

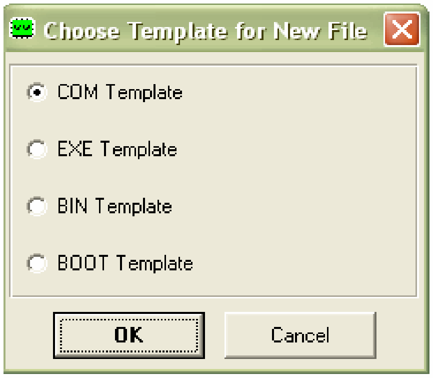
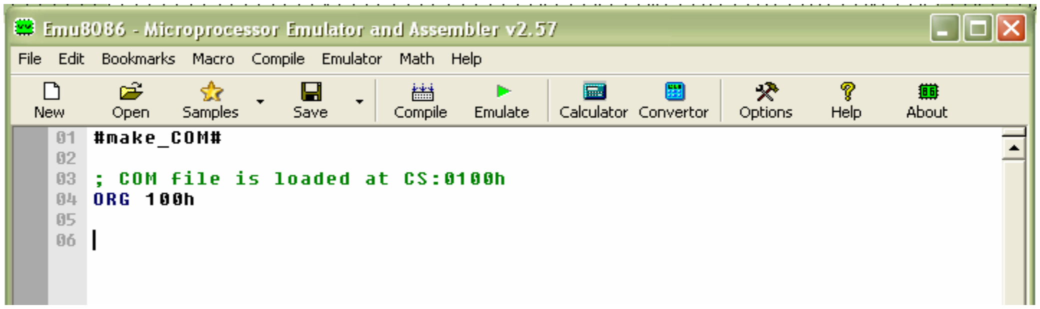


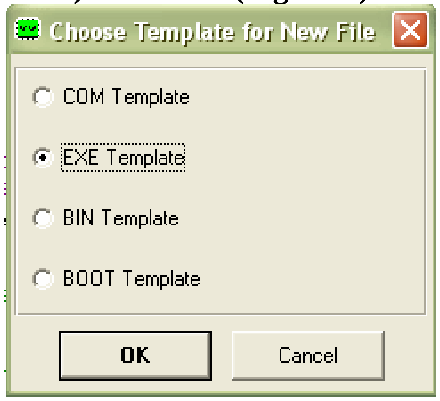
Figure 4



```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

```

Figure 5

B) EXE File (Figure 6)



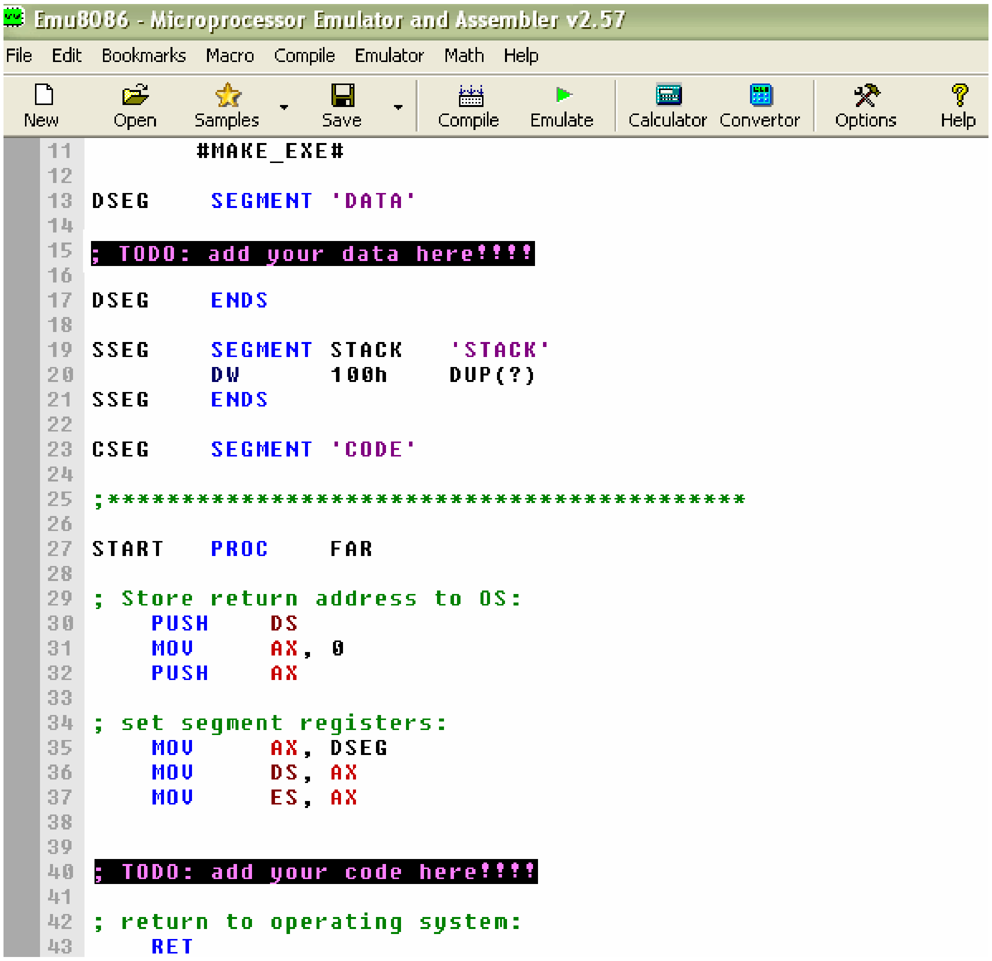
○ COM Template

● EXE Template

○ BIN Template

○ BOOT Template

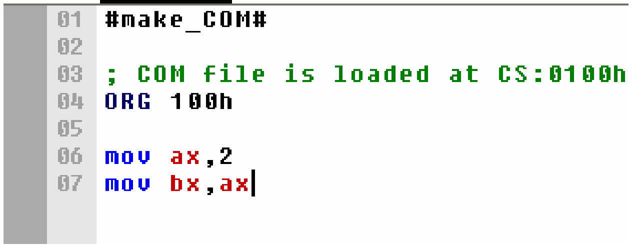
Figure 6



```
       #MAKE_EXE#

DSEG    SEGMENT 'DATA'

; TODO: add your data here!!!!

DSEG    ENDS

SSEG    SEGMENT STACK   'STACK'
        DW      100h    DUP(?)
SSEG    ENDS

CSEG    SEGMENT 'CODE'

;*******************************************

START   PROC    FAR

; Store return address to OS:
    PUSH    DS
    MOV     AX, 0
    PUSH    AX

; set segment registers:
    MOV     AX, DSEG
    MOV     DS, AX
    MOV     ES, AX


; TODO: add your code here!!!!

; return to operating system:
    RET
```

Figure7

- **Example: Create COM file and write these tow instructions as shown in (Figure 8)**



```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

mov ax,2
mov bx,ax
```

Figure 8

- **Compile, correct errors if any, and then run (Emulate)… (Figure 9)**

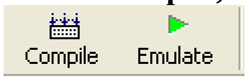
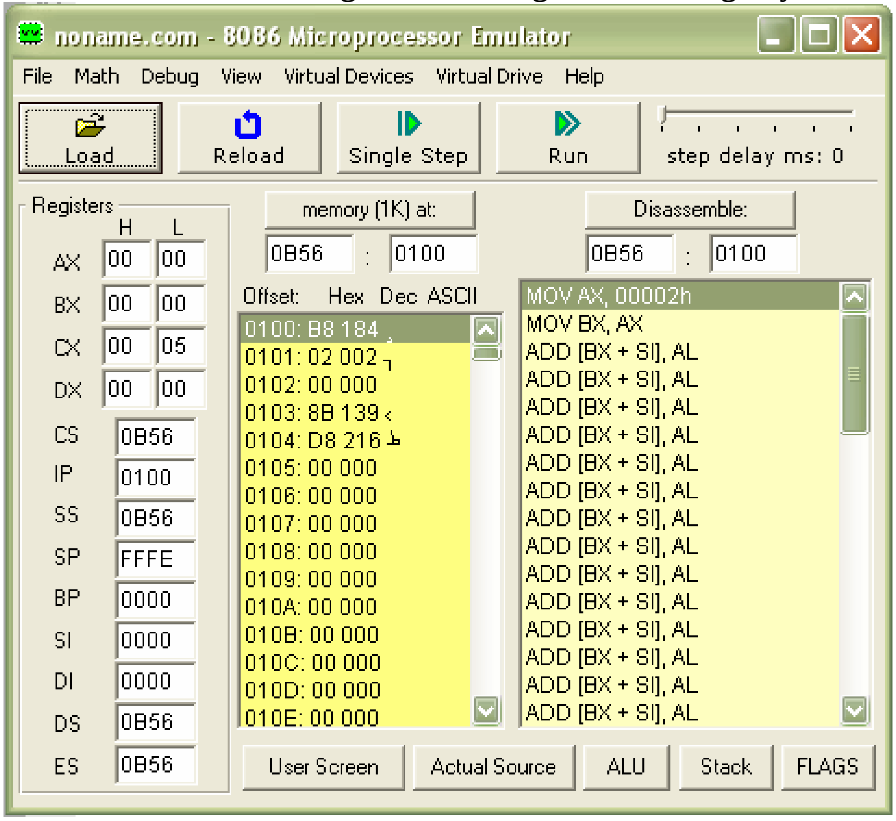


Figure 9

- **When you press Emulate, this window (Figure 10) will appear, it displays the value of all Registers, flags, block of memory (1K) at a time as Hex value, Decimal value, and ASCII char, and disassembled machine code. You can watch and change them during the executing of your code.**



	H	L
AX	00	00
BX	00	00
CX	00	05
DX	00	00

CS	0B56
IP	0100
SS	0B56
SP	FFFE
BP	0000
SI	0000
DI	0000
DS	0B56
ES	0B56

memory (1K) at: 0B56 : 0100

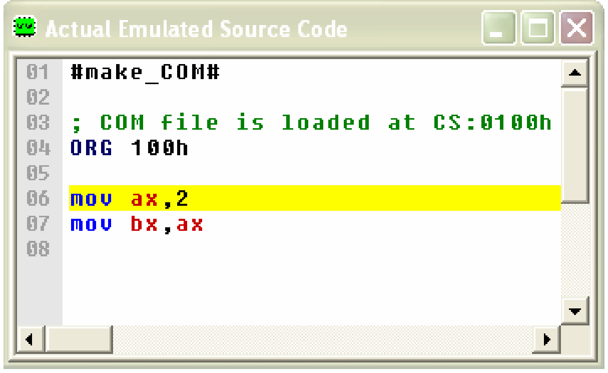
```
Offset:  Hex Dec ASCII
0100: B8 184 ¸
0101: 02 002 ┐
0102: 00 000
0103: 8B 139 ‹
0104: D8 216 ط
0105: 00 000
0106: 00 000
0107: 00 000
0108: 00 000
0109: 00 000
010A: 00 000
010B: 00 000
010C: 00 000
010D: 00 000
010E: 00 000
```

Disassemble: 0B56 : 0100

```
MOV AX, 00002h
MOV BX, AX
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
ADD [BX + SI], AL
```

Figure 10

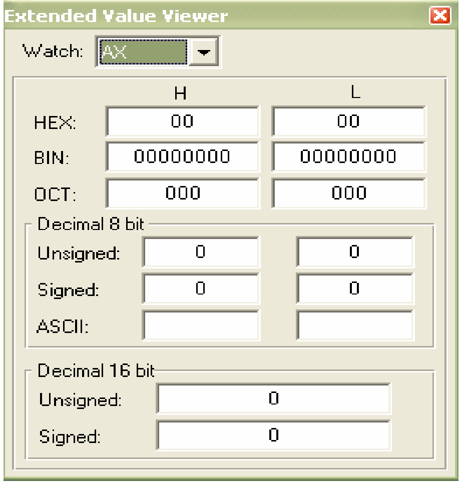
- **This window (Figure 11) views the source code; the highlighted instruction is the next one to be executed**



```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

mov ax,2
mov bx,ax
```

Figure 11

- **To change any register or any memory location, double click on the register (Figure 10), this window (Figure 12) will appear, from Watch select the register then change it as HEX, BIN, or OCT**



Watch: AX

	H	L
HEX:	00	00
BIN:	00000000	00000000
OCT:	000	000

Decimal 8 bit

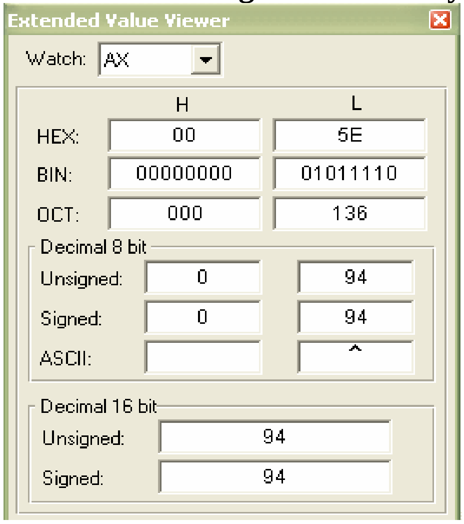
Unsigned:	0	0
Signed:	0	0
ASCII:		

Decimal 16 bit

Unsigned:	0
Signed:	0

Figure 12

Here we changed the lower byte of AX to 5E instead of 00 (Figure 13)



Watch: AX

	H	L
HEX:	00	5E
BIN:	00000000	01011110
OCT:	000	136

Decimal 8 bit

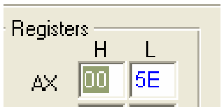
	H	L
Unsigned:	0	94
Signed:	0	94
ASCII:		^

Decimal 16 bit

Unsigned:	94
Signed:	94

Figure 13

Close the window in Figure 13; the value of AX is changed (blue color) (Figure 14)

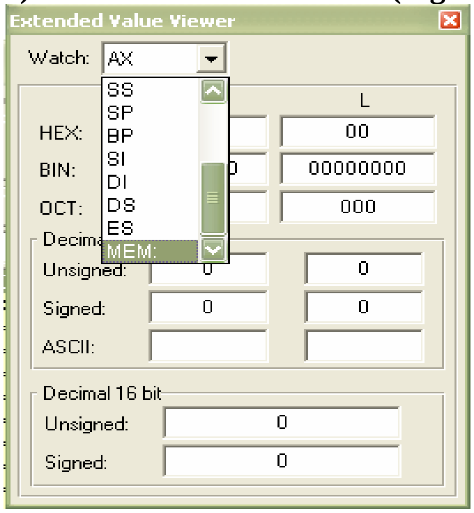


	H	L
AX	00	5E

Figure 14

- **Here we will change tow memory locations at 0B56:0106**

1) Select MEM from Watch (Figure 15).

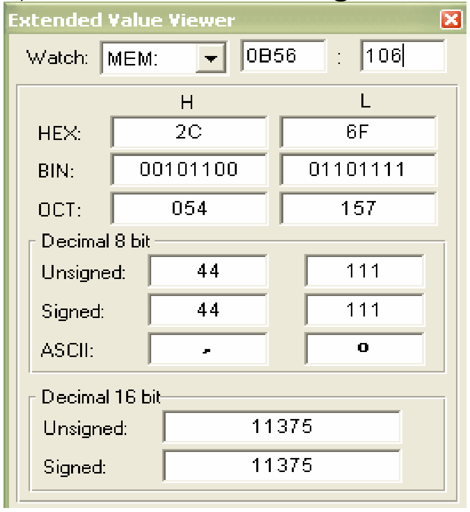


	L
HEX:	00
BIN:	00000000
OCT:	000
Unsigned:	0
Signed:	0

Decimal 16 bit Unsigned: 0 Signed: 0

Figure 15

2) Write the address “segment: offset” 0B56:0106 (Figure 16)



Watch: MEM: 0B56 : 106

	H	L
HEX:	2C	6F
BIN:	00101100	01101111
OCT:	054	157

Decimal 8 bit

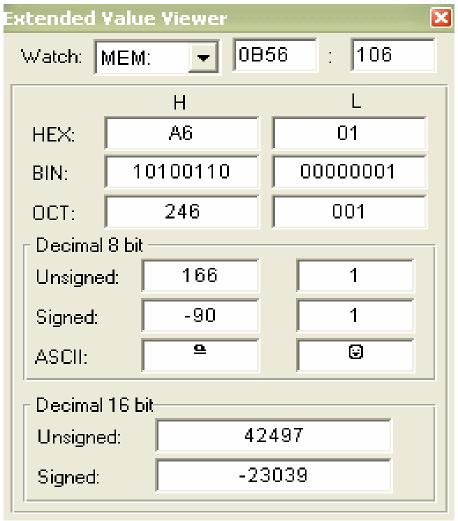
	H	L
Unsigned:	44	111
Signed:	44	111
ASCII:	,	o

Decimal 16 bit

Unsigned:	11375
Signed:	11375

Figure 16

3) Change 0B56:0106 to 01 and 0B56:0107 to A6h. (Figure 17)



Watch: MEM: 0B56 : 106

	H	L
HEX:	A6	01
BIN:	10100110	00000001
OCT:	246	001

Decimal 8 bit

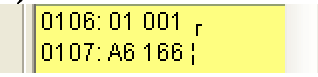
	H	L
Unsigned:	166	1
Signed:	-90	1
ASCII:	ª	☺

Decimal 16 bit

Unsigned:	42497
Signed:	-23039

Figure 17

4) Close the window in Figure 17; the value at offset 0106 and 0107 is changed (Figure 18)



```
0106: 01 001 ┌
0107: A6 166 ¦
```

Figure 18

- **To start Running:**

Press on Single Step (Figure 19)

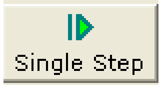
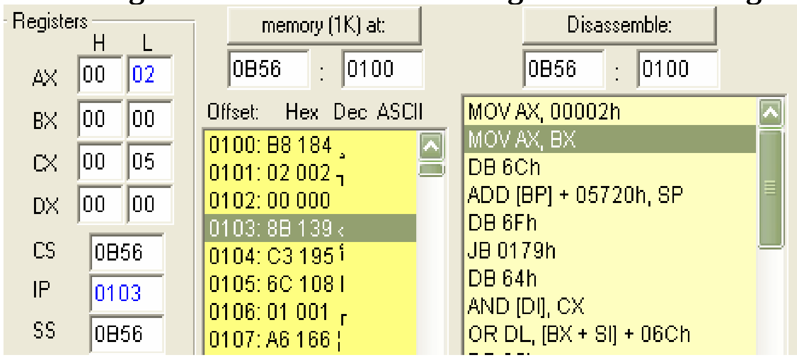


Figure 19

At CS:IP one instruction will be executed at a time.

- **Figure 20 & 21 show the changes after executing MOV AX, 2**



Registers

	H	L
AX	00	02
BX	00	00
CX	00	05
DX	00	00
CS	0B56	
IP	0103	
SS	0B56	

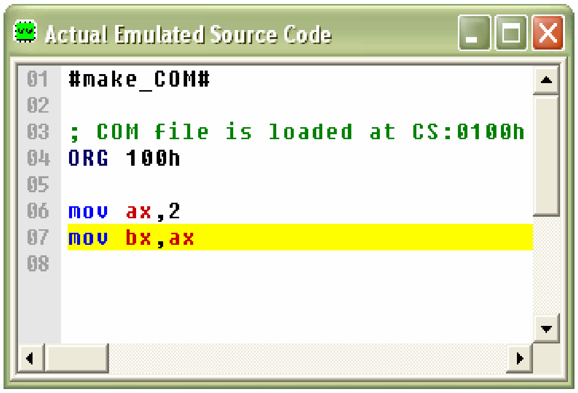
memory (1K) at: 0B56 : 0100

```
Offset:   Hex  Dec  ASCII
0100: B8 184 ╕
0101: 02 002 ☻
0102: 00 000
0103: 8B 139 ï
0104: C3 195 ├
0105: 6C 108 l
0106: 01 001 ☺
0107: A6 166 ª
```

Disassemble: 0B56 : 0100

```
MOV AX, 00002h
MOV AX, BX
DB 6Ch
ADD [BP] + 05720h, SP
DB 6Fh
JB 0179h
DB 64h
AND [DI], CX
OR DL, [BX + SI] + 06Ch
```

Figure 20

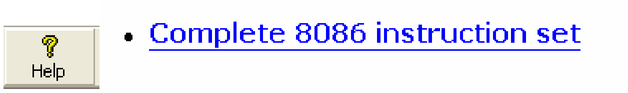


```
#make_COM#

; COM file is loaded at CS:0100h
ORG 100h

mov ax,2
mov bx,ax

```

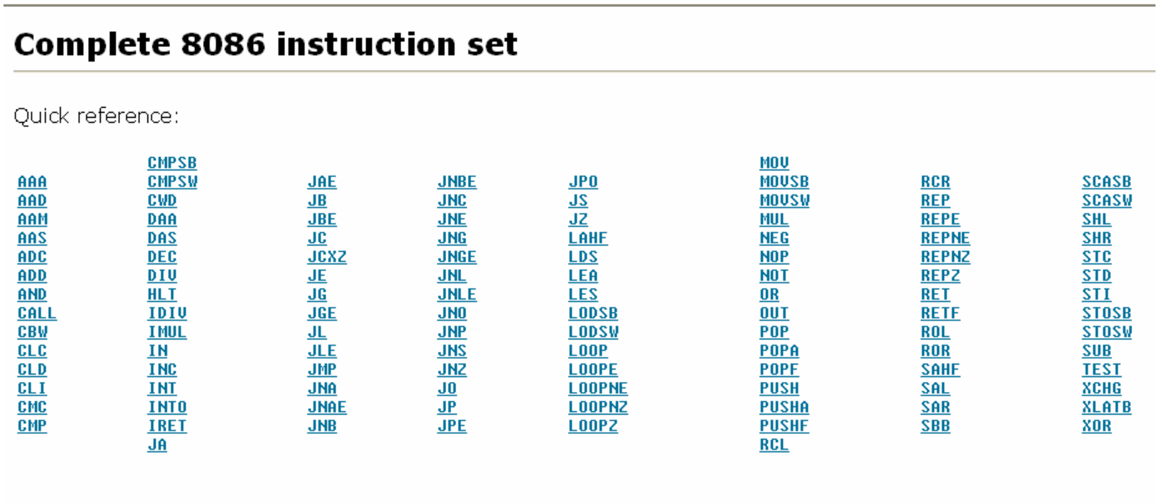
Figure 21

- **Then press again on Single Step for the next instruction until you finish your code.**

- **NOTE: Emulator has a complete 8086 instruction set, press on HELP**



- Complete 8086 instruction set



Complete 8086 instruction set

Quick reference:

AAA
AAD
AAM
AAS
ADC
ADD
AND
CALL
CBW
CLC
CLD
CLI
CMC
CMP

CMPSB
CMPSW
CWD
DAA
DAS
DEC
DIV
HLT
IDIV
IMUL
IN
INC
INT
INTO
IRET
JA

JAE
JB
JBE
JC
JCXZ
JE
JG
JGE
JL
JLE
JMP
JNA
JNAE
JNB

JNBE
JNC
JNE
JNG
JNGE
JNL
JNLE
JNO
JNP
JNS
JNZ
JO
JP
JPE

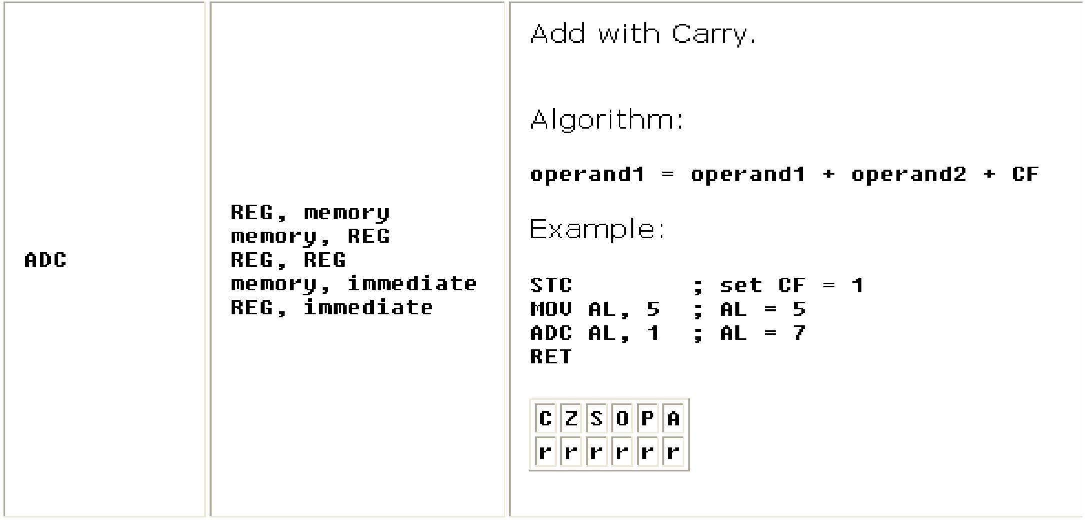
JPO
JS
JZ
LAHF
LDS
LEA
LES
LODSB
LODSW
LOOP
LOOPE
LOOPNE
LOOPNZ
LOOPZ

MOV
MOVSB
MOVSW
MUL
NEG
NOP
NOT
OR
OUT
POP
POPA
POPF
PUSH
PUSHA
PUSHF
RCL

RCR
REP
REPE
REPNE
REPNZ
REPZ
RET
RETF
ROL
ROR
SAHF
SAL
SAR
SBB

SCASB
SCASW
SHL
SHR
STC
STD
STI
STOSB
STOSW
SUB
TEST
XCHG
XLATB
XOR

- **Choose the instruction you want (i.e. ADC)**



ADC

```
REG, memory
memory, REG
REG, REG
memory, immediate
REG, immediate
```

Add with Carry.

Algorithm:

```
operand1 = operand1 + operand2 + CF
```

Example:

```
STC          ; set CF = 1
MOV AL, 5    ; AL = 5
ADC AL, 1    ; AL = 7
RET
```

C	Z	S	O	P	A
r	r	r	r	r	r

NOTE: The stack memory area is set by SS (Stack Segment) register, and SP (Stack Pointer) register. Generally operating system sets values of these registers on program start.

"PUSH source" instruction does the following:

- **Subtract 2 from SP register.**
- **Write the value of source to the address SS:SP.**

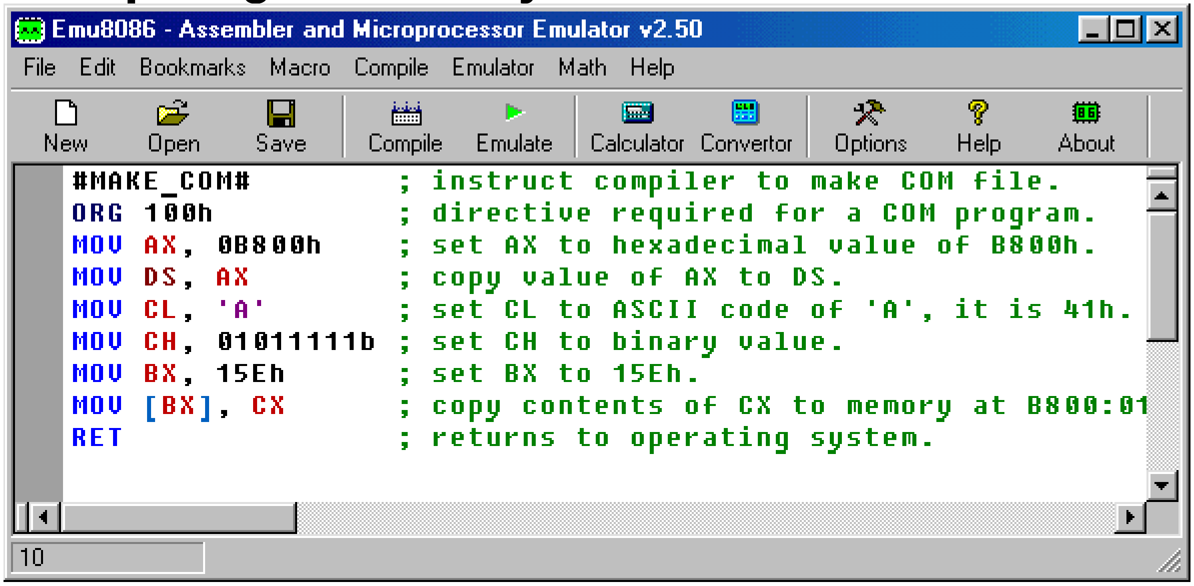
"POP destination" instruction does the following:

- **Write the value at the address SS:SP to destination.**
- **Add 2 to SP register.**

The current address pointed by SS:SP is called the top of the stack.

For COM files stack segment is generally the code segment, and stack pointer is set to value of 0FFFEh. At the address SS:0FFFEh stored a return address for RET instruction that is executed in the end of the program.

Compiling Assembly Code



```
#MAKE_COM#         ; instruct compiler to make COM file.
ORG 100h           ; directive required for a COM program.
MOV AX, 0B800h     ; set AX to hexadecimal value of B800h.
MOV DS, AX         ; copy value of AX to DS.
MOV CL, 'A'        ; set CL to ASCII code of 'A', it is 41h.
MOV CH, 01011111b  ; set CH to binary value.
MOV BX, 15Eh       ; set BX to 15Eh.
MOV [BX], CX       ; copy contents of CX to memory at B800:01
RET                ; returns to operating system.
```

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.

The Output File Type Directives:

#MAKE_COM#
#MAKE_BIN#
#MAKE_BOOT#
#MAKE_EXE#

You can insert these directives in the source code to specify the required output type for the file. Only if compiler cannot find any of these directives it will ask you for output type before creating the file.

Description of Output File Types:

#MAKE_COM# - the oldest and the simplest format of an executable file, such files are loaded with 100h prefix (256 bytes). Compiler directive ORG 100h should be added before the code. Execution always starts from the first byte of the file.
Supported by DOS and Windows Command Prompt.
ORG 100h is a compiler directive (it tells compiler how to handle the source code). This directive is very important when you work with variables. It tells compiler that the executable file will be loaded at the offset of 100h (256 bytes), so compiler should calculate the correct address for all variables when it replaces the variable names with their offsets.
Why executable file is loaded at offset of 100h? Operating system keeps some data about the program in the first 256 bytes of the CS (code segment), such as command line parameters and etc.

#MAKE_EXE# - more advanced format of an executable file. Not limited by size and number of segments. Stack segment should be defined in the program. You may select EXE Template from the New menu in to create a simple EXE program with defined Data, Stack, and Code segments.
Entry point (where execution starts) is defined by a programmer.
Supported by DOS and Windows Command Prompt.

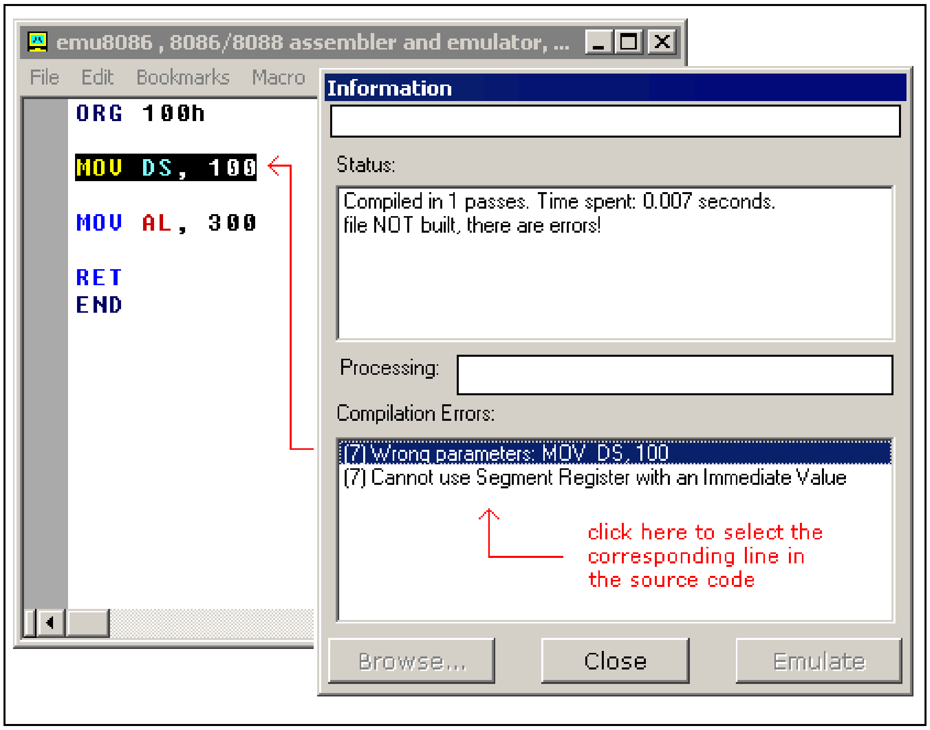
#MAKE_BIN# - a simple executable file. You can define the values of all registers, segment and offset for memory area where this file will be loaded. Execution starts from values in CS:IP.
This file type is unique to Emu8086 emulator.

#MAKE_BOOT# - this type is a copy of the first track of a floppy disk (boot sector).

Compiler directive ORG 7C00h should be added before the code, when computer starts it loads first track of a floppy disk at the address 0000:7C00.
The size of a .BOOT file should be less then 512 bytes (limited by the size of a disk sector).
Execution always starts from the first byte of the file.
This file type is unique to Emu8086 emulator.

Error Processing

Compiler reports about errors in a separate information window:



```
ORG 100h

MOV DS, 100

MOV AL, 300

RET
END
```

Status:

Compiled in 1 passes. Time spent: 0.007 seconds.
file NOT built, there are errors!

Processing:

Compilation Errors:

(7) Wrong parameters: MOV DS, 100
(7) Cannot use Segment Register with an Immediate Value

click here to select the corresponding line in the source code

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

Compiler makes several passes before generating the correct machine code; if it finds an error and does not complete the required number of passes it may show incorrect error messages. For example:

```
#make_COM#
ORG 100h

MOV AX, 0
MOV CX, 5
m1: INC AX
LOOP m1             ; not a real error!

MOV AL, 0FFFFh      ; error is here.

RET
```

List of generated errors:
(7) Condition Jump out of range!: LOOP m1

```
(9) Wrong parameters: MOV AL, 0FFFFh
(9) Operands do not match: Second operand is over 8 bits!
```

First error message (7) is incorrect, compiler did not finish calculating the offsets for labels, so it presumes that the offset of m1 label is 0000, that address is out of the range because we start at offset 100h.

Make correction to this line: MOV AL, 0FFFFh (AL cannot hold 0FFFFh value). This fixes both errors! For example:

```
#make_COM#
ORG 100h

MOV AX, 0
MOV CX, 5
m1: INC AX
LOOP m1           ; same code no error!

MOV AL, 0FFh        ; fixed!

RET
```