Assignment No. 1

8086 Assembler

Objectives:

1. Know what assembly language is.
2. Have knowledge about number representation (hex/bin).
3. Have knowledge about **MOV** instruction.

Theory:

1.1 What is assembly language?

Assembly language is a low level programming language. You need to get some knowledge about computer structure in order to understand anything. The simple computer model, figure (1.1), as I see it:

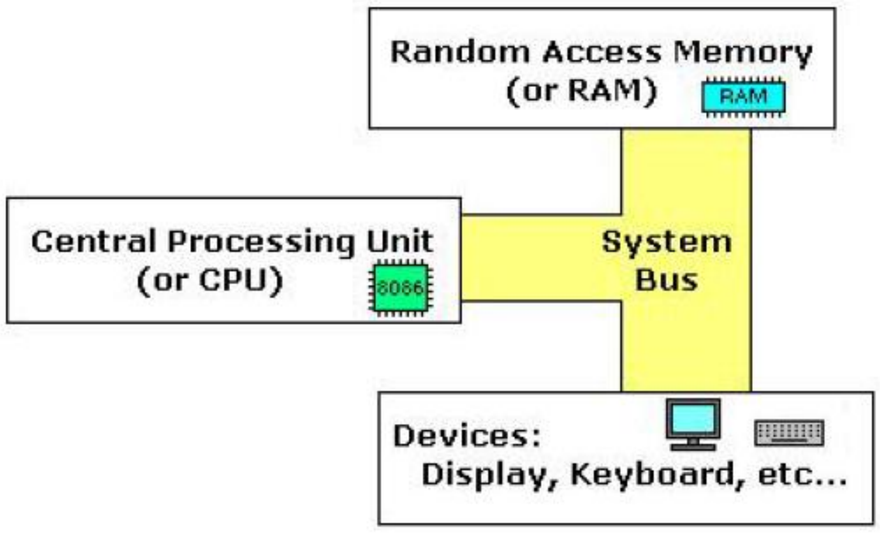


Fig. (1.1) The simple computer model

- The **system bus** connects the various components of a computer.
- The **CPU** is the heart of the computer, most of computations occurs inside the **CPU**.
- **RAM** is a place to where the programs are loaded in order to be executed.

1.2 Inside the CPU

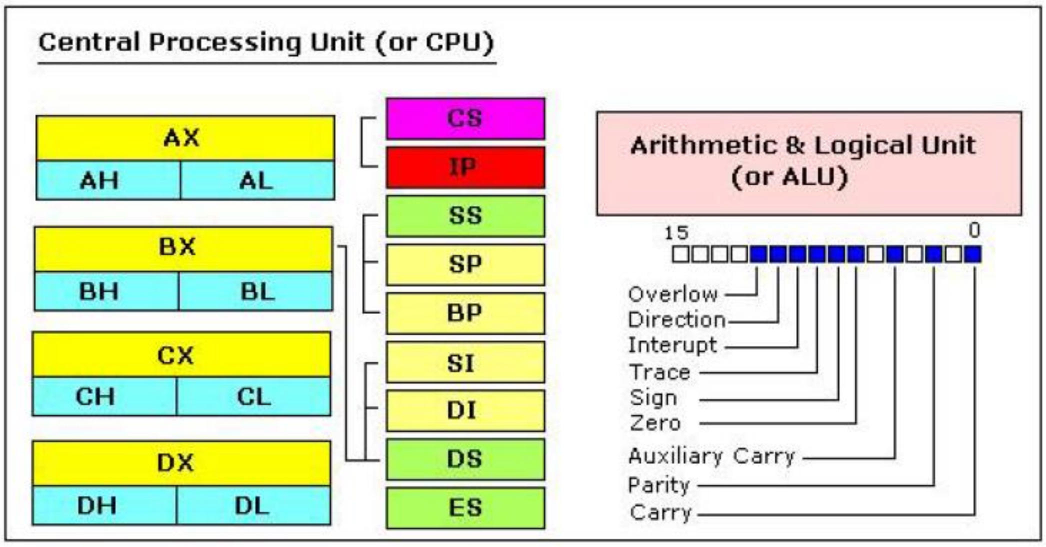


Fig. (1.2) The Central Processing Unit Registers

General purpose registers

As shown in figure (1.2) 8086 CPU has 8 general purpose registers, each register has its own name:

- **AX -** the accumulator register (divided into **AH / AL**).
- **BX -** the base address register (divided into **BH / BL**).
- **CX -** the count register (divided into **CH / CL**).
- **DX -** the data register (divided into **DH / DL**).
- **SI -** source index register.
- **DI -** destination index register.
- **BP -** base pointer.
- **SP -** stack pointer.

Despite the name of a register, it's the programmer who determines the usage for each general purpose register. The main purpose of a register is to keep a number (variable). The size of the above registers is 16 bit, it's something like:
0011000000111001b (in binary form), or **3039h** in (hexadecimal form).

4 general purpose registers (**AX, BX, CX, and DX**) are made of two separate 8 bit registers, for example if AX= **0011000000111001b**, then AH=**00110000b** and AL=**00111001b**. Therefore, when you modify any of the 8 bit registers 16 bit register is also updated, and vice-versa. The same is for other 3 registers, "H" is for high and "L" is for low part.

Because registers are located inside the CPU, they are much faster than memory. Accessing a memory location requires the use of a system bus, so it takes much longer. Accessing data in a register usually takes no time. Therefore, you should try to keep variables in the registers. Register sets are very small and most registers have special Purposes which limit their use as variables, but they are still an excellent place to store temporary data of calculations.

Segment registers

- **CS -** points at the segment containing the current program.
- **DS -** generally points at segment where variables are defined.
- **ES -** extra segment register, it's up to a coder to define its usage.
- **SS -** points at the segment containing the stack.

Special purpose registers

- **IP -** the instruction pointer.
- **Flags register -** determines the current state of the microprocessor.

IP register always works together with **CS** segment register and it points to currently executing instruction.

Flags register is modified automatically by CPU after mathematical operations, this allows to determine the type of the result, and to determine conditions to transfer control to other parts of the program.

Generally you cannot access these registers directly, the way you can access AX and other general registers, but it is possible to change values of system registers using some tricks that you will learn a little bit later.

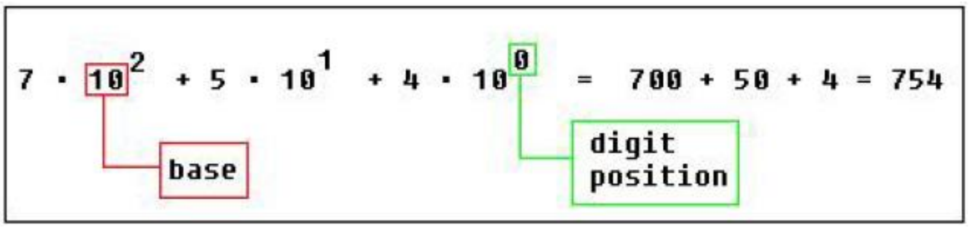
1.3 **Numbering Systems**

1.3.1 Decimal System

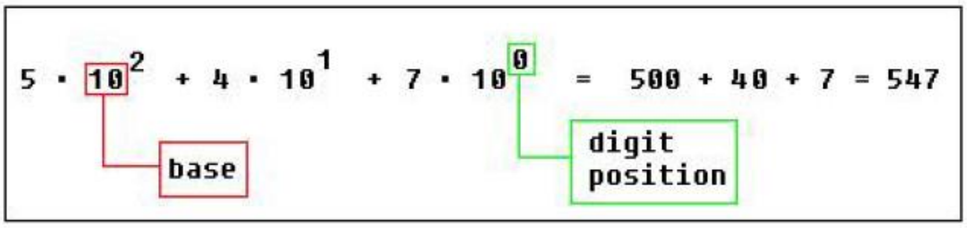
Most people today use decimal representation to count. In the decimal system there are 10 digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**

These digits can represent any value, for example: **754**.

The value is formed by the sum of each digit, multiplied by the **base** (in this case it is **10** because there are 10 digits in decimal system) in power of digit position (counting from zero):



Position of each digit is very important! for example if you place "7" to the end: 547 it will be another value



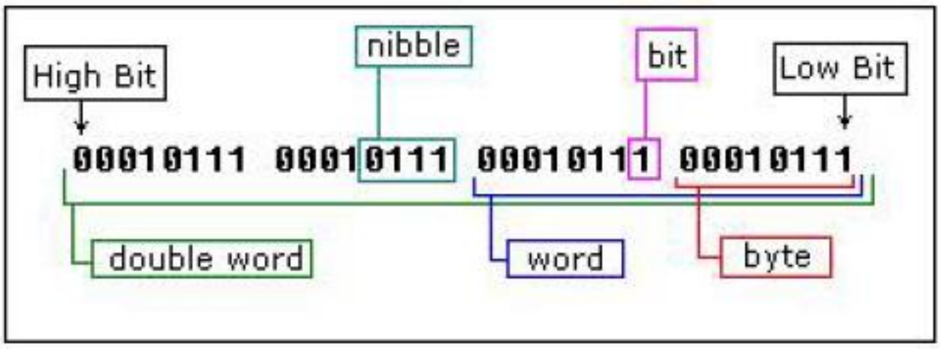
Important note: any number in power of zero is 1, even zero in power of zero is 1

$10^0 = 1$ $0^0 = 1$ $x^0 = 1$

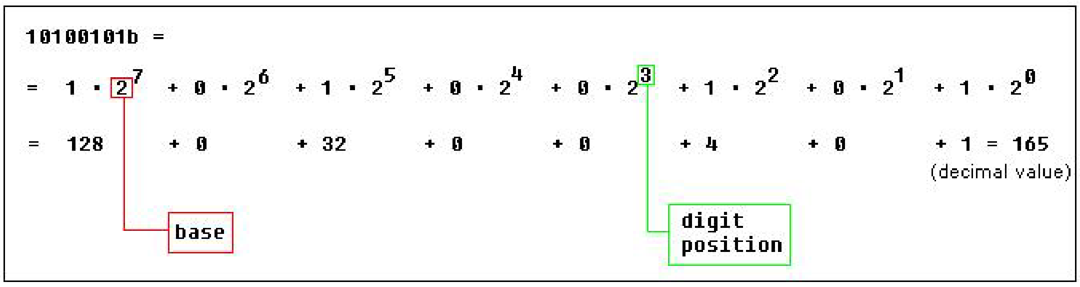
1.3.2 Binary System

Computers are not as smart as humans are (or not yet), it's easy to make an electronic machine with two states: **on** and **off**, or **1** and **0**.
Computers use binary system, binary system uses 2 digits:
0, 1 And thus the **base** is **2**.

Each digit in a binary number is called a **BIT**; 4 bits form a **NIBBLE**, 8 bits forma **BYTE**, two bytes form a **WORD**, two words form a **DOUBLE WORD** (rarely used):



There is a convention to add **"b"** in the end of a binary number, this way we can determine that 101b is a binary number with decimal value of 5. The binary number **10100101b** equals to decimal value of 165:



1.3.3 Hexadecimal System

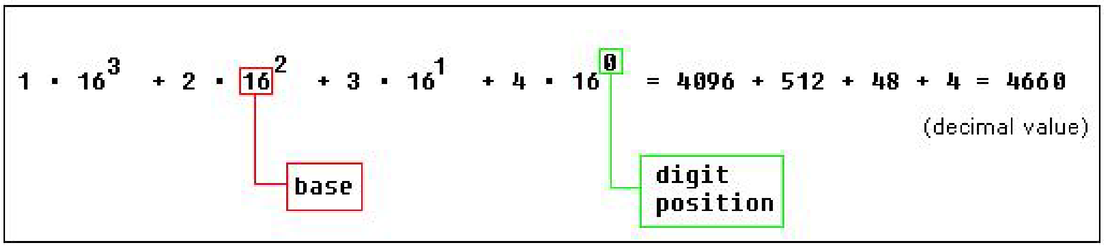
Hexadecimal System uses 16 digits: **0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.**

And thus the **base** is **16**. Hexadecimal numbers are compact and easy to read.

It is very easy to convert numbers from binary system to hexadecimal system and vice-versa, every nibble (4 bits) can be converted to a hexadecimal digit using this table:

Decimal base (10)	**Binary base (2)**	**Hexadecimal base (16)**
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

There is a convention to add **"h"** in the end of a hexadecimal number, this way we can determine that 5Fh is a hexadecimal number with decimal value of 95. We also add **"0"** (zero) in the beginning of hexadecimal numbers that begin with a letter (A..F), for example **0E120h**.The hexadecimal number **1234h** is equal to decimal value of 4660:

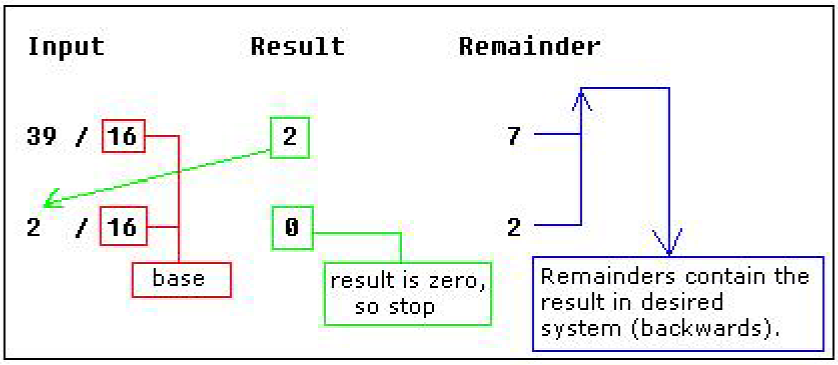

$1 \cdot 16^3 + 2 \cdot 16^2 + 3 \cdot 16^1 + 4 \cdot 16^0 = 4096 + 512 + 48 + 4 = 4660$
(decimal value)
base
digit position

1.3.4 Converting from Decimal System to Any Other

In order to convert from decimal system, to any other system, it is required to divide the decimal value by the **base** of the desired system, each time you should remember the **result** and keep the **remainder**, the divide process continues until the **result** is zero.The **remainders** are then used to represent a value in that system.

Let's convert the value of **39** (base 10) to *Hexadecimal System* (base 16):

As you



As you see we got this hexadecimal number: **27h**. All remainders were below **10** in the above example, so we do not use any letters.

<u>**Practice 1**</u>: Convert decimal number **43868** to hexadecimal

1.4 MOV instruction

- Copies the second operand (source) to the first operand (destination).
- The source operand can be an immediate value, general-purpose register or memory location.
- The destination register can be a general-purpose register, or memory location.
- Both operands must be the same size, which can be a byte or a word.

These types of operands are supported:

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

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

immediate: 5, -24, 3Fh, 10001101b

for segment registers only these types of **MOV** are supported:

```
MOV REG, SREG
MOV SREG, REG
```

SREG: DS, ES, SS, and only as second operand: CS.

REG: AX, BX, CX, DX, AH, AL, BL, BH, CH, CL, DH, DL, DI, SI, BP, SP.

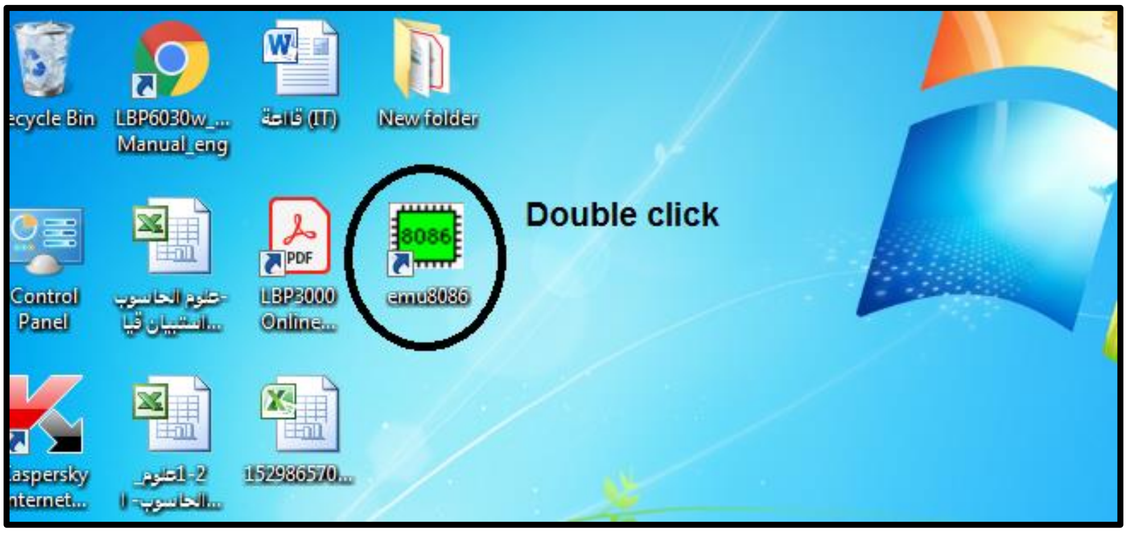
The **MOV** instruction cannot be used to set the value of the **CS** and **IP** registers.

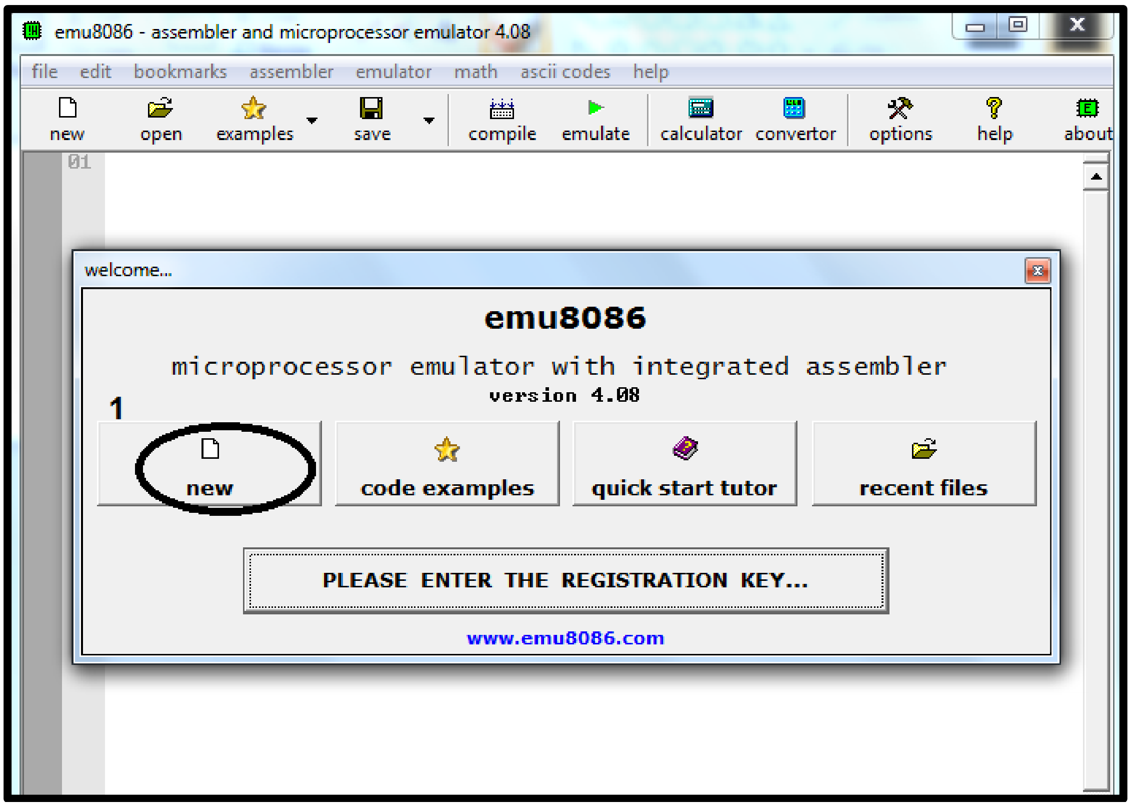
Practice2

Type the program code into the code editor:

```
org 100h
mov al,2Ah
mov ah,5fh
mov bx,ax
ret
```

1.5 How to use emu8086




emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01
welcome...
1
new
code examples
quick start tutor
recent files

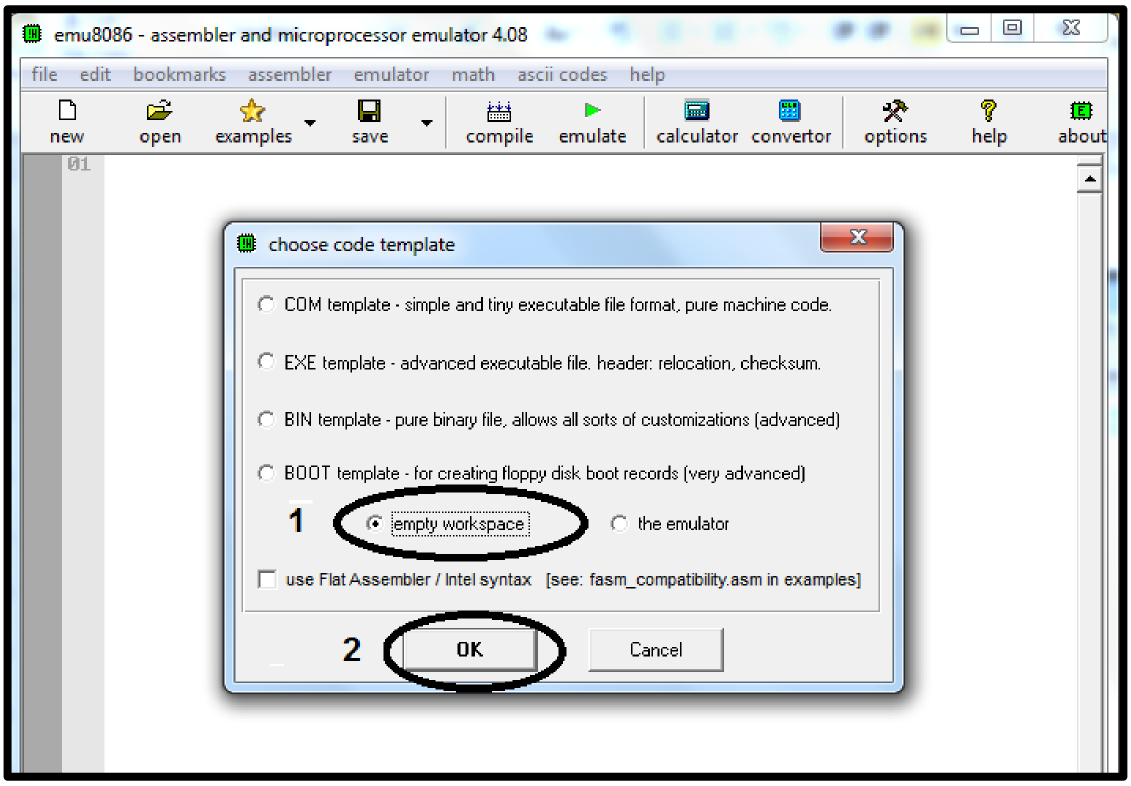
emu8086

microprocessor emulator with integrated assembler

version 4.08

PLEASE ENTER THE REGISTRATION KEY...

www.emu8086.com


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
01
choose code template
1
2
OK
Cancel

COM template - simple and tiny executable file format, pure machine code.

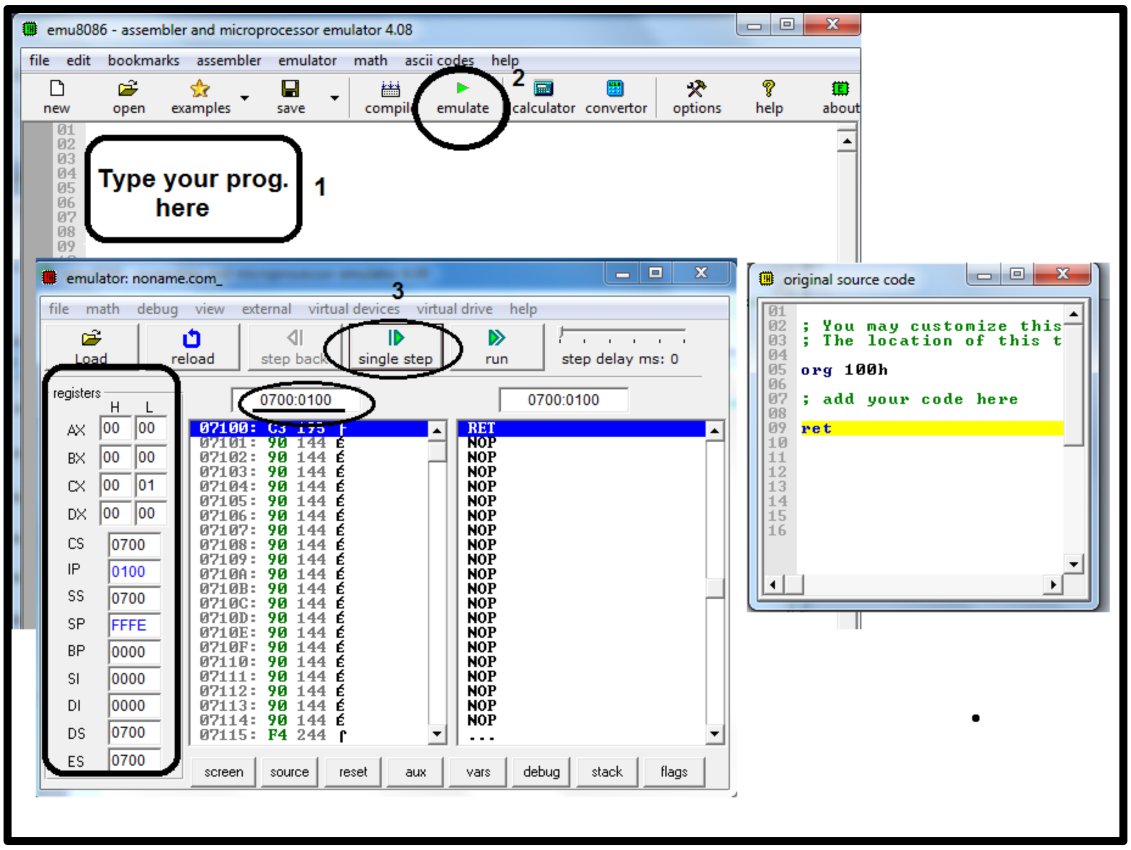
EXE template - advanced executable file. header: relocation, checksum.

BIN template - pure binary file, allows all sorts of customizations (advanced)

BOOT template - for creating floppy disk boot records (very advanced)

[✓ empty workspace] the emulator

use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]


emu8086 - assembler and microprocessor emulator 4.08
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
Type your prog. here
1
2
3
emulator: noname.com_
file math debug view external virtual devices virtual drive help
Load reload step back single step run step delay ms: 0
screen source reset aux vars debug stack flags
original source code

registers

	H	L
AX	00	00
BX	00	00
CX	00	01
DX	00	00
CS	0700	
IP	0100	
SS	0700	
SP	FFFE	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0700:0100                       0700:0100

07100: C3 195 ┟                 RET
07101: 90 144 É                 NOP
07102: 90 144 É                 NOP
07103: 90 144 É                 NOP
07104: 90 144 É                 NOP
07105: 90 144 É                 NOP
07106: 90 144 É                 NOP
07107: 90 144 É                 NOP
07108: 90 144 É                 NOP
07109: 90 144 É                 NOP
0710A: 90 144 É                 NOP
0710B: 90 144 É                 NOP
0710C: 90 144 É                 NOP
0710D: 90 144 É                 NOP
0710E: 90 144 É                 NOP
0710F: 90 144 É                 NOP
07110: 90 144 É                 NOP
07111: 90 144 É                 NOP
07112: 90 144 É                 NOP
07113: 90 144 É                 NOP
07114: 90 144 É                 NOP
07115: F4 244 ſ                 ...
```

```

; You may customize this
; The location of this t

org 100h

; add your code here

ret
```