8086 assembler tutorial for beginners (part 2)

Memory Access

To access memory we can use these four registers: **BX, SI, DI, BP**.
combining these registers inside **[ ]** symbols, we can get different memory locations. these combinations are supported (addressing modes):

[BX + SI] [BX + DI] [BP + SI] [BP + DI]	[SI] [DI] d16 (variable offset only) [BX]	[BX + SI + d8] [BX + DI + d8] [BP + SI + d8] [BP + DI + d8]
[SI + d8] [DI + d8] [BP + d8] [BX + d8]	[BX + SI + d16] [BX + DI + d16] [BP + SI + d16] [BP + DI + d16]	[SI + d16] [DI + d16] [BP + d16] [BX + d16]

d8 - stays for 8 bit signed immediate displacement (for example: 22, 55h, -1, etc...)

d16 - stays for 16 bit signed immediate displacement (for example: 300, 5517h, -259, etc...).

displacement can be a immediate value or offset of a variable, or even both. if there are several values, assembler evaluates all values and calculates a single immediate value..

displacement can be inside or outside of the **[ ]** symbols, assembler generates the same machine code for both ways.

displacement is a **signed** value, so it can be both positive or negative.

generally the compiler takes care about difference between **d8** and **d16**, and generates the required machine code.

DS Register

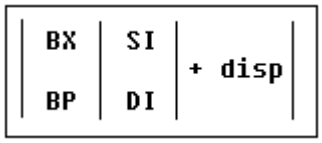
Remember that the DS (data segment) register is set to point to the beginning of a 64K segment. THE CS (code segment) denotes the beginning of the code segment. In simple EMU programs, these are often the same.

Example: Assume that **DS = 100**, **BX = 30**, **SI = 70**.
The following addressing mode: **[BX + SI] + 25**

is calculated by processor to this physical address:
100 * 16 + 30 + 70 + 25 = 1725.

by default **DS** segment register is used for all modes except those with **BP** register, for these **SS** segment register is used.

there is an easy way to remember all those possible combinations using this chart:



you can form all valid combinations by taking only one item from each column or skipping the column by not taking anything from it. as you see **BX** and **BP** never go together. **SI** and **DI** also don't go together. here are an examples of a valid addressing modes: **[BX+5]** , **[BX+SI]** , **[DI+BX-4]**

---

the value in segment register (CS, DS, SS, ES) is called a **segment**,
and the value in purpose register (BX, SI, DI, BP) is called an **offset**.
When DS contains value **1234h** and SI contains the value **7890h** it can be also recorded as **1234:7890**. The physical address will be 1234h * 10h + 7890h = 19BD0h.

if zero is added to a decimal number it is multiplied by 10, however **10h = 16**, so if zero is added to a hexadecimal value, it is multiplied by 16, for example:

7h = 7
70h = 112

---

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.
```

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.

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

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

memory: [BX], [BX+SI+7], variable, etc...

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

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

```
MOV SREG, memory
MOV memory, SREG
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.

memory: [BX], [BX+SI+7], variable, etc...

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

here is a short program that demonstrates the use of **MOV** instruction:

```
ORG 100h           ; this directive required for a simple 1 segment .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, 1101_1111b ; set CH to binary value.
MOV BX, 15Eh       ; set BX to 15Eh.
MOV [BX], CX       ; copy contents of CX to memory at B800:015E
RET                ; returns to operating system.
```

you can **copy & paste** the above program to the code editor, and press [**Compile and Emulate**] button (or press **F5** key on your keyboard).

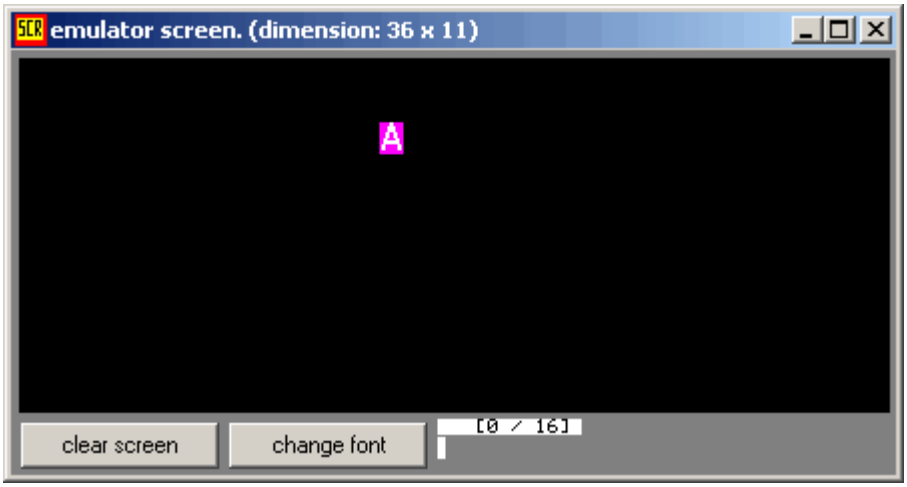
the emulator window should open with this program loaded, click [**Single Step**] button and watch the register values.

how to do **copy & paste**:

1. select the above text using mouse, click before the text and drag it down until everything is selected.
2. press **Ctrl + C** combination to copy.
3. go to the source editor and press **Ctrl + V** combination to paste.

as you may guess, "**;**" is used for comments, anything after "**;**" symbol is ignored by compiler.

you should see something like that when program finishes:



actually the above program writes directly to video memory, so you may see that **MOV** is a very powerful instruction.

<<< previous part <<< **>>> Next Part >>>**
