**Examples reflecting the implicit rules for prefixing an offset with the corresponding segment register**

Mov eax, [ebx+esp] ; ESP – base… EBX – index ;EAX ß …SS:…

Mov eax, [esp + ebx] ; ESP – base… EBX – index ;EAX ß …SS:…

Mov eax, [ebx+esp\*2] ; syntactic error BECAUSE ESP can be ONLY a base register and here is specified as an index register because scale is present with it !

Mov eax, [ebx+ebp\*2] ; mov eax, DWORD PTR [DS:EBX+EBP\*2]

Mov eax, [ebx+ebp] ; …DS…

Mov eax, [ebp+ebx] ; …SS…

Mov eax, [ebx\*2+ebp] ; …SS…

Mov eax, [ebx\*1+ebp] ;…SS…

Mov eax, [ebp\*1+ebx] ; …DS…

Mov eax, [ebx\*1+ebp\*1] ; ;…SS…

Mov eax, [ebp\*1+ebx\*1] ; …DS…

Jmp et1 ; …CS:et1…

Jmp [et1] ; JMP short [DS:0f6795B4] - I have to take 4 bytes as the needed correct offset to be referred to the current CS !!!

* I go in memory to the address DS:0f6795B4 , because of [] I will take THE CONTENTS from this address (for example 0BA2F5C4) and BECAUSE of JMP this contents will be THE TARGET OFFSET to which I (the processor) will perform this JMP (this offset being relative to the current CS). So, the JMP will be made to the address CS: 0BA2F5C4 !!!!

What you will be as programmers confronted with in your checkings will be that DS=ES=SS=GS , a slightly different value for CS and a different value for FS (this comes from THE FLAT MEMORY MODEL !!!).

CS:correct\_offset ; “Access violation” run-time error !

JMP DWORD PTR [DS…] – to be performed at CS:the correct identified offset

JMP 5 ….?? (to be discussed !)

* **CS** for code labels target of the control transfer instructions (jmp, call, ret, jz etc);
* **SS** in SIB addressing when using EBP or ESP as *base* (no matter of *index* or *scale*);
* **DS** for the rest of data accesses;

Definition of a variable/subroutine = Declaration (to specify its attributes) + ALLOCATION (it’s a ONE TIME ONLY process)

**The steps followed by a program from source code to run-time:**

* Syntactic checking (done by assembler/compiler/interpreter)
* OBJ files are generated (….)
* Linking phase (performed by a LINKER = a tool provided by the OS, which checks the possible DEPENDENCIES between this OBJ files/modules); The result à .EXE file !!!
* You (the user) are activating your exe file by clicking or enter-ing…
* The LOADER of the OS is looking for the required RAM memory space for your EXE file. When finding it, it loads the EXE file AND performs ADDRESS RELOCATION !!!!
* In the end the loader gives control to the processor by specifying THE PROGRAM’s ENTRY POINT (ex: the start label) !!! The run-time phase begins NOW…

Mark Zbirkowski – semnatura EXE = ‘MZ’