<u>Examples - implicit rules for prefixing an offset with the corresponding segment register</u>
(material discussed and developed together with you)

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

Mov eax, [ebx+esp*2] ; - syntactic error !!
Mov eax, [ebx+ebp*2] ; - ok ! …DS

Mov eax, [ebx+ebp] ;   ok ! …DS
Mov eax, [ebp+ebx] ;   ok ! …SS

Mov eax, [ebx*2+ebp] ; ok ! - …SS

Mov eax, [ebx*1+ebp]   ; …SS
Mov eax, [ebp*1+ebx]   ; …DS

Mov eax, [ebx*1+ebp*1] ; ok !...SS
Mov eax, [ebp*1+ebx*1] ; … DS

Mov eax, [ebx*1+ebp*2] ; ????
Mov eax, [ebp*1+ebx*2] ; ????

Jmp et1 ; …CS:et1…

Jmp eax;

Jmp [DS:et1] ; 4 bytes (short / NEAR jmp) will be taken from [DS:et1] and will be considered as a POINTER to which the jmp will be made IN THE SAME CODE SEGMENT… The jmp will be made to CS:[DS:et1]

JMP FAR [et1]

Jmp 5 ; syntax error because it does not follow the syntax : JMP label/register/memory address

- **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;

<u>Examples - implicit rules for prefixing an offset with the corresponding segment register</u> (material prepared by me in advance) – I left them both here for being parsed and analyzed comparatively if it helps you…

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

Mov eax, [ebx+esp*2] ; syntactic error BECAUSE ESP can be ONLY a base register !
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 !!! JMP DWORD PTR [DS…] – to be performed at CS:the correct identified offset ; JMP CS:correct_offset (taken relatively to DS) will result usually in “Access violation” run-time error ! (to be checked by you!)

- 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 within your checkings will be that DS=ES=SS=GS, a different value for CS and a different value for FS (due to the FLAT MEMORY MODEL).

Jmp 5 ; syntax error BECAUSE it does not obey the JMP syntax , 5 is not a label, nor a register and nor a memory address !!! - Relative call to absolute address not supported by OBJ format

- **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;

[eax+ebx] – indirect addressed operand ; [v] – direct addressed operand (the contents !!!)
V – is determinable at assembly time as an offset !

Bitwise operations and operators

Attention to the difference between operators and instructions !!!

Mov ah, 01110111 << 3 ; AH :=10111000b Vs.

Mov ah, 01110111
Shl ah, 3

& - bitwise AND operator	x AND 0 = 0	; x AND x = x
AND – instruction	x AND 1 = x	; x AND ~x = 0

Operation useful for FORCING THE VALUES OF CERTAIN BITS TO 0 !!!!

| - bitwise OR operator	x OR 0 = x	; x OR x = x
OR – instruction	x OR 1 = 1	; x OR ~x = 1

Operation useful for setting the values of some bits to 1 !!!

^ - bitwise EXCLUSIVE OR operator;	x XOR 0 = x ;	x XOR x = 0
XOR – instruction	x XOR 1 = ~x;	x XOR ~x = 1

Operation useful for COMPLEMENTING the value of some bits !

XOR ax, ax ; AX=0 !!! = 00000000 0000000b

Reported Error types in Computer Science

- **Syntax error – diagnosed by assembler/compiler !**

- **Run-time error (execution error) – program crashes – it stops executing**

- **Logical error = program runs until its end or remains blocked in an infinite loop … if it functions until its end, it functions LOGICALLY WRONG obtaining totally different results/output then the envisioned ones**

- **Fatal: Linking Error !!! (for example in the case of a variable defined multiple times in a multimodule program … if we have 17 modules, a variable must be defined ONLY in a SINGLE module ! If it is defined in 2 or more modules , a "Fatal: Linking Error !!! – Duplicate definition for symbol …." Will be obtained.**