**Exemple ilustrand regulile implicite de prefixare ale unui offset cu segmentul aferent**

Mov eax, [ebx+esp] ; eax 🡨 dword care incepe la adresa [SS:esp+ebx]

Mov eax, [esp + ebx] ; eax 🡨 dword care incepe la adresa [SS:esp+ebx]

Mov eax, [ebx+esp\*2] ; syntax error – ESP nu poate fi index !

Mov eax, [ebx+ebp\*2] ; eax 🡨 dword care incepe la adresa [DS:ebx + 2\*ebp]

Mov eax, [ebx+ebp] ; EAX 🡨 …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….

Mov eax, [v] ; …[DS:v]

Jmp et1 ; ….CS:et1….

Jmp [et1] ; …[DS:et1]…. NU sare deloc la et1 !!!!!!!!! – si nu face nici salt in segmentul de date (DS) la eticheta et1, ci doar incarca din segmentul de date de la offset-ul et1 continutul a 4 octeti reprezentand adresa offset la care se va face saltul (in segmentul de cod !, deci acel offset extras din memorie va fi considerat relative la CS!). Cel mai probabil se va obtine Access violation error – run-time error !! – de verificat de fiecare dintre voi !!

Obs. Datorita modelului de memorie flat adoptat, dpdv practic se va obtine acelasi efect in fiecare din cazurile:

Jmp [et1] , jmp [DS:et1], jmp [ES:et1], jmp [CS:et1], jmp [SS:et1], jmp [GS:et1], insa nu si pt jmp [FS:et1], care va furniza o alta valoare, deoarece FS este intializat cu o valoare diferita fata de ceilalti registri segment (verificati va rog!).

Jmp 5 - syntax error PT CA nu se respecta sintaxa instructiunii

* “relative call to absolute address not supported by OBJ format”

JMP operand ; operand = eticheta, registru, adresa de memorie

Jmp [ebp] ; …[SS:EBP]…- la SS:[EBP] am continutul variabilei pointer… se ia acest continut (32 biti = offset, deci doar continutul a 4 octeti de la adresa [SS:EBP]) si se efectueaza saltul la CS:offset preluat de la [SS:EBP].

JMP DWORD PTR [SS:EBP] ; asa apare in OLLYDbg !!!!!

Regulile pentru asocierile implicite sunt:

* **CS** pentru etichete de cod destinaţie ale unor salturi (jmp, call, ret, jz etc);
* **SS** în adresări SIB ce foloseste EBP sau ESP drept bază (indiferent de index sau scală);
* **DS** pentru restul accesărilor de date

**Operatii si operatori pe biti**

Atentie la diferenta dintre operatori si instructiuni !!

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

Vs.

Mov ah, 01110111b

Shl ah, 3

& - operatorul SI bit cu bit x AND 0 = 0 ; x AND x = x

AND – instructiunea x AND 1 = x ; x AND ~x = 0

Operatie utila pt. ZERORIZAREA ANUMITOR BITI !!!!

Mov ah,[a]

AND AH, ….

| - operatorul SAU bit cu bit x OR 0 = x ; x OR x = x

OR – instructiunea x OR 1 = 1 ; x OR ~x = 1

Operatie utila pt. SETAREA ANUMITOR BITI la valoarea 1 !!!

^ - operatorul SAU EXCLUSIV bit cu bit; x XOR 0 = x ; x XOR x = 0

XOR – instructiunea x XOR 1 = ~x ; x XOR ~x = 1

Operatie utila pt. COMPLEMENTAREA unor biti !!!!!

Mov AH, 17^(~17) ; AH = 11111111b = 0ffh

Xor ax, ax echiv cu mov ax,0 !!!!

AH =0 (inseamna FALSE)

AH = ~0 pe 8 biti = 11111111b = 0ffh (inseamna TRUE) !!!!!

**Utilizarea operatorilor ! si ~**

! Negare logică: !X = 0 când X ≠ 0, altfel 1

~ Complement faţă de 1: mov al, ~0 => mov AL, 0ffh

Mov eax, ![a] ; expression syntax error pt ca… [a] NU este o constanta det la mom asamblarii

Mov eax, [!a] ; ! can only be applied to SCALAR values !!!!!

A = POINTER !!!!!!!

Mov eax, !a ; ! can only be applied to SCALAR values !!!!!

A = POINTER !!!!!!!

Mov eax, !(a+7) ; ! can only be applied to SCALAR values !!!!!

A = POINTER !!!!!!!

Mov eax, !(b-a) ; OK !!!!

Mov eax, ![a+7] – expression syntax error !

Mov eax, !7 ; EAX = 0 ;

Mov AH, ~7 ; 7 = 00000111b deci ~7 = 11111000b = 0f8h

Mov eax, !ebx ; syntax error !

Aa equ 2

Mov ah, !aa ; AH = 0 !!!! – MERGE !!!!!!