Contorul de locaţii şi aritmetica de pointeri

```
SEGMENT data

a db 1,2,3,4 ; 01 02 03 04

lg db $-a       ; 04
lg db $-data ; syntax error – expression is not simple or relocatable
(în TASM $-data=4 – deci acelaşi efect ca mai sus, deoarece în TASM, MASM
offset(nume_segment)=0 !!!; în NASM NU, offset(data) = 00401000 !!!)
lg db a-data ; syntax error – expression is not simple or relocatable

lg2 dw data-a; asamblorul ne lasă, însă obţinem eroare la link-editare – “Error in file at 00129 – Invalid fixup recordname count….”

   db a-$      ; = -5 = FB

c  equ  a-$    ;  0-6 = -6 = FA
d equ  a-$    ;  0-6 = -6 = FA
e  db a-$      ;  0-6 = -6 = FA
x dw x         ; 07 10 !!!!!
x db x         ; syntax error !
x1 dw x1     ; x1 = offset(x1)  09 00 la asamblare si in final 09 10

   db lg-a        ; 04
   db a-lg        ; = -4 = FC
   db [$-a]            ; expression syntax error
   db [lg-a]           ; expression syntax error

 lg1  EQU lg1          ; lg1 = 0 NASM consideră că e corect ! (IT IS A NASM BUG !!!)

lg1  EQU lg1-a        ; lg1 = 0 – DE CE ????? Bug NASM ! Orice se evalueaza la zero în dreapta este acceptat chiar daca este definiţie recursivă ! (de asta e BUG !) Dacă a este primul element definit în segment consideră a=0 (offset-ul faţă de începutul segmentului) şi va furniza lg1=0; dacă a este definit altundeva şi offset(a) ≠ 0, atunci va furniza eroare de sintaxă = “Macro abuse, recursive EQUs”

g34 dw c-2   ; -8 = F8

b  dd  a-start   ; syntax error ! (a definit aici, start definit altundeva)  expression is not simple or relocatable !!!
```

dd start-a ; MERGE !!! (start definit altundeva şi a definit aici !) – rez=POINTER !!! deoarece etichetele NU fac parte din acelasi segment şi este interpretată ca scădere FAR = pointer !!!

dd start-start1 ; MERGE !!! (deoarece amandouă etichetele sunt definite în acelaşi segment !!) ; rez=SCALAR !!!! pt că aici avem scădere de offset-uri definite în acelaşi segment !

segment code use32

start:

```
mov ah, lg1    ; AH = 0
mov bh, c      ; BH = -6 = FA

mov ch, lg         ; OBJ format can handle only 16 or 32 byte relocation
                     (offset NU începe în 1 byte !!!)
mov ch, lg-a       ; CH = 04
mov ch, [lg-a]     ;  mov ch, byte ptr DS:[4] – Mem. access violation – cel mai probabil…

mov cx, lg-a        ;  CX = 4
mov cx, [lg-a]      ; mov WORD ptr DS:[4] – Mem. access violation – cel mai probabil…

mov cx, $-a  ;  invalid operand type !!!!! ($ generat aici, a altundeva)

mov cx, $$-a ; invalid operand type !!! ; aici avem $$ din code şi a din data segment!!!!
mov cx, a-$  ;  OK !!!!!! Merge !! (a definit altundeva şi $ generat aici !)

mov ch, $-a  ;  invalid operand type !!!!!  ($ generat aici, a altundeva)
mov ch, a-$  ;  OBJ format can handle only 16 or 32 byte relocation
        a-$ is OK, dar a-$ = POINTER !! – syntax error ! (pt că offset NU începe în 1 byte !!)

mov cx, $-start  ; ok !!!
mov cx, start-$  ; ok !!! (ambele scăderi ok - pt că etichetele sunt din acelaşi segment !!!)

mov ch, $-start  ; ok !!! – pt ca REZ este scalar ! (dacă era pointer nu mergea !!)
mov ch, start-$  ; ok !!!

mov cx, a-start   ; ok !!! (a definit altundeva si start definit aici !)
mov cx, start-a   ; invalid operand type !!! (start definit aici, a altundeva)
```

start1:

mov ah, a+b ; MERGE !!!!!!!!!! DAR …. NU ESTE ADUNARE DE POINTERI !!!
E adunare de scalari !!! a+b = (a-$$) + (b-$$)

mov ax, b+a ; AX = (b-$$) + (a-$$) – adunare de SCALARI !!!!

mov ax, [a+b] ; INVALID EFFECTIVE ADDRESS !!!! – ASTA CHIAR E ADUNARE DE POINTERI !!!!!! - deci e INTERZISA !!! – deci e syntax error !!!

var1 dd a+b ; syntax error ! - expression is not simple or relocatable
(deci NASM nu permite ca "a+b" să apară într-o definiţie de date ca expresie de iniţializare, ci NUMAI ca OPERAND AL UNEI INSTRUCTIUNI – cum se observă mai sus !)

Concluzie:

Expresiile de tip et1 – et2 (unde et1 si et2 sunt etichete – fie de cod, fie de date) sunt acceptate sintactic de catre NASM,

- Fie dacă ambele sunt definite in acelaşi segment

- Fie dacă et1 aparţine unui segment diferit faţă de cel în care apare expresia, iar et2 este definită în segmentul în care apare expresia. Intr-un astfel de caz, TD asociat expresiei et1-et2 este POINTER şi NU SCALAR (constantă numerică) ca şi în cazul etichetelor ce fac parte din acelasi segment. (Deci ALTUNDEVA – AICI merge !!!!, însă AICI – ALTUNDEVA NU !!!!!)

  Scădere de offset-uri ce se raportează la acelaşi segment = SCALAR
  Scădere de pointeri din segmente diferite = POINTER

Numele unui segment este asociat cu "Adresa segmentului in memorie în faza de execuţie" însă aceasta nu ne este disponibilă/accesibilă, fiind decisă la momentul încărcării programului pt execuţie de către încărcătorul de programe al SO. De aceea, dacă folosim nume de segmente în expresii din program în mod explicit vom obţine fie eroare de sintaxă (în situaţii de tipul $/a-data - "AICI – ALTUNDEVA") fie de link-editare (în situaţii de tipul data-a - ALTUNDEVA – AICI), deoarece practic numele unui segment este asociat cu adresa FAR al acestuia (adresă care nu va fi cunoscută decât la momentul încărcării programului, deci va fi disponibilă doar la run-time; observăm astfel că adresă_segment este considerată ca "ALTUNDEVA") şi NU este asociat cu "Offset-ul segmentului în faza de asamblare, fiind o constantă determinată la momentul asamblării" (aşa cum este în programarea sub 16 biţi de exemplu, unde nume_segment în faza de asamblare = offset-ul său = 0). Ca urmare, se constată că în progr. sub 32 de biţi numele de segmente NU pot fi folosite in expresii !!

Mov eax, data ; Segment selector relocations are not supported in PE files (relocation error occurred) - syntax error!

Sub 32 biţi offset (data) = vezi OllyDbg – DATA segment începe la offset-ul 00402000, iar CODE segment la offset 00401000 (sau eventual invers, depinde de link-editor, versiunea de SO etc). Deci offset-urile începuturilor de segment nu sunt 0 la fel ca şi la programarea sub 16 biţi. Din acest punct de vedere sub 32 biţi este o mare diferenţă între numele de variabile (al căror offset poate fi determinat la asamblare) şi numele de segmente (al căror offset NU poate fi determinat la momentul asamblării ca şi o constantă, această valoare fiind cunoscută la fel ca şi adresa de segment doar la momentul încărcării programului pt execuţie - loading time).

Dacă avem mai mulţi operanzi pointeri, asamblorul va încerca să încadreze expresia într-o combinaţie validă de operaţii cu pointeri:

Mov bx, [(v3-v2) ± (v1-v)] – OK !!! (adunarea şi scăderea de valori imediate este corectă !!)

Dacă expresia nu va putea fi încadrată într-o combinaţie validă de operaţii cu pointeri, vom obţine eroarea de sintaxă “Invalid effective address”:

Mov bx, [v2-v1-v] ; Invalid effective address !

Mov bx, v2-v1-v ; Invalid operand type !

Mov bx, v2-v1-v ; Invalid operand type ! mov bx, v2-(v1+v) – de unde concluzionăm că v1+v este acceptată ca şi SCALAR în interpretarea a+b = (a-$$) + (b-$$), doar dacă apare DE SINE STATATOR sau în combinaţie cu alţi SCALARI , dar NU şi nu în combinaţie cu expresii de tip POINTER !!!

Mov bx, v2+v1-v ; ok = v2+scalar

Mov bx, [v3-v2-v1-v] ; Invalid effective address !

Mov bx, v3-v2-v1-v ; OK !!! – deoarece ?... Mov bx, v3-v2-v1-v = Mov bx, (v3-v2)-(v1+v) = scădere de valori imediate (SCALARE!)

Varianta de adresare directă vs.indirectă este mai permisivă ca operaţii aritmetice în NASM (din cauza acceptării expresiilor de tip “a+b”) însă asta nu înseamnă că este mai permisivă IN OPERATIILE CU POINTERI !!!

Mov bx, (v3±v2) ± (v1±v) – OK !!! (adunarea şi scăderea de valori imediate este corectă !!)