

# **Contorul de locații și** **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 același efect ca mai sus, deoarece în TASM, MASM  
offset(nume\_segment)=0 !!!; în NASM NU, offset(data) = 00402000 !!!)

lg db a-data ; syntax error – expression is not simple or relocatable

lg2 dw data-a; asamblorul ne lasă, însă obținem 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 20 !!!!!

x db x ; syntax error !

x1 dw x1 ; x1 = offset(x1) 09 00 la asamblare si in final 09 20

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 definiție recursivă ! (de asta e BUG !) Dacă a este primul element definit în segment consideră a=0 (offset-ul față de începutul segmentului) și va furniza lg1=0; dacă a este definit altundeva și offset(a) ≠ 0, atunci va furniza eroare de sintaxă = “Macro abuse, recursive EQUs”

g34 dw c-2 ; -8

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 și a definit aici !) – rez=POINTER !!!  
deoarece etichetele NU fac parte din același segment și este interpretată ca scădere FAR = pointer !!!

dd start-start1 ; MERGE !!! (deoarece amandouă etichetele sunt definite în același segment !!) ; rez=SCALAR !!!! pt că aici avem scădere de offset-uri definite în același 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 încapă 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 și a din data segment!!!!  
mov cx, a-\$ ; OK !!!!! Merge !! (a definit altundeva și \$ 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 încapă în 1 byte !!)

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

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

mov cx, a-start ; ok !!! (a definit altundeva și 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 definiție de date ca expresie de inițializare, ci NUMAI ca OPERAND AL UNEI INSTRUCȚIUNI – 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 în același segment

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

Scădere de offset-uri ce se raportează la același segment = SCALAR

Scădere de pointeri din segmente diferite = POINTER

Numele unui segment este asociat cu "Adresa segmentului în memorie în faza de execuție" însă aceasta nu ne este disponibilă/accesibilă, fiind decisă la momentul încărcării programului pt execuție de către încărcătorul de programe al SO. De aceea, dacă folosim nume de segmente în expresii din program în mod explicit vom obține fie eroare de sintaxă (în situații de tipul  $\$/a-data$  - “AICI – ALTUNDEVA”) fie de link-editare (în situații 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”) și NU este asociat cu "Offset-ul segmentului în faza de asamblare, fiind o constantă determinată la momentul asamblării" (așa cum este în programarea sub 16 biți de exemplu, unde nume\_segment în faza de asamblare = offset-ul său = 0). Ca urmare, se constată că în progr. sub 32 de biți numele de segmente NU pot fi folosite în expresii !!

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

Sub 32 biți 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 și la programarea sub 16 biți. Din acest punct de vedere sub 32 biți este o mare diferență între numele de variabile (al căror offset poate fi determinat la asamblare) și numele de segmente (al căror offset NU poate fi determinat la momentul asamblării ca și o constantă, această valoare fiind cunoscută la fel ca și adresa de segment doar la momentul încărcării programului pt execuție - loading time).

Dacă avem mai mulți operanzi pointeri, asamblorul va încerca să încadreze expresia într-o combinație validă de operații cu pointeri:

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

Dacă expresia nu va putea fi încadrată într-o combinație validă de operații cu pointeri, vom obține 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 și SCALAR în interpretarea **a+b = (a-\$\$) + (b-\$\$)**, doar dacă apare DE SINE STATATOR sau în combinație cu alți SCALARI , dar NU și nu în combinație 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 operații 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 și scăderea de valori imediate este corectă !!)**