Contorul de locaţii şi aritmetica de pointeri (discutate impreuna)

SEGMENT data

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

lg db $-a ; 04
lg db $-data ; syntax error
lg db a-data ; syntax error

lg2 dw data-a; este acceptata de care asamblor !!!! in schimb da eroare la link-editor !!!

db a-$; 0 – 5 = -5 = FB

c equ a-$; 0-6 = -6 = FA
db lg-a ; 4-0 = 4
db a-lg ; 0-4 = -4 = FC
db [$-a] ; expression syntax error
db [lg-a] ; expression syntax error

lg1 EQU lg1 ; lg1=0 ; nasm considera ca e corect ! (IT IS A NASM BUG !!!)

lg1 EQU lg1-a ; 0-0 =0 ; NASM va initializa constanta lg1 cu 0 !!! Deci MERGE!!!! (daca in schimb am orice definit inaintea lui a, a.i. offset(a) dif. 0 NU MAI MERGE !!!....

g34 dw c-2 ; -8 = F8

b dd a-start ; syntax error ! (a definit aici, start altundeva)
dd start-a ; MERGE !!! (start definit altundeva si a definit aici !) – rez=POINTER !!! deoarece etichetele NU fac parte din acelasi segment si este interpretata scaderea ca scadere FAR = pointer !!!
dd start-start1 ; MERGE !!! (deoarece amandoua etichetele sunt definite in acelasi segment !!!) ; rez=SCALAR !!!! pt ca aici avem scadere de offset-uri de variabile definite in acelasi segment !!!

segment code use32

start:

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

mov ch, lg         ; offset NU incape in 1 byte !!
mov ch, lg-a       ; CH=4
mov ch, [lg-a]     ;   mov byte ptr DS:[4] ?? – Cel mai probabil Memory access violation

mov cx, lg-a        ;  CX = 4
mov cx, [lg-a]     ; mov word ptr DS:[4] ?? – Cel mai probabil Memory access violation

mov cx, $-a  ;  Syntax error !! $ = contorul de locatii al lui CODE segment
                ($ generat aici, a altundeva)
mov cx, a-$  ;  OK !!! MERGE !!! (a definit altundeva si $ generat aici !)

mov ch, $-a  ;  syntax error
mov ch, a-$  ;  offset nu incape pe 1 byte

mov cx, $-start  ; ok !
mov cx, start-$  ; ok !

mov ch, $-start  ; ok, pt ca e scalar !!!
mov ch, start-$  ; ok

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

start1:

mov ah, a+b   ; ok ! MERGE !!! DAR NU ESTE ARITMETICA de POINTERI !!!
                E adunare de scalari !!! a+b = (a-$$) + (b-$$) !!!

mov ax, b+a   ; ibidem
```

mov ax, [a+b] ; ASTA CHIAR O INTERPRETEAZA CA ADUNARE DE POINTERI !!!, deci e INTERZISA !!! – deci e syntax error !!!

Daca nu am definite sectiuni explicite , atunci $$=inceputul segmentului respectiv

Contorul de locaţii şi aritmetica de pointeri (exemple si explicatii pregatite in avans)

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

lg2 dw data-a; asamblorul ne lasa insa obtinem eroare la link-editare – “Error in file at 00129 – Invalid fixup recordname count….”

   db a-$      ; = -5 = FB

c  equ  a-$    ; = -6 = FA
   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

lg1  EQU lg1-a        ; lg1 = 0 – DE CE ????? Bug NASM ! Orice se evalueaza la zero in dreapta este acceptat chiar daca este definitie recursive ! (de asta e BUG !) Daca a este primul element definit in segment considera a=0 (offset-ul fata de inceputul segmentului) si va furniza lg1=0; daca a este definite altundeva si offset(a) ≠ 0, atunci va furniza eroare de sintaxa = Macro abuse, recursive EQUs
```

g34 dw c-2 ; -8

b dd a-start ; expression is not simple or relocatable !!!
dd start-a ; OK !!!!!!!! - va fi POINTER !!!!!
dd start-start1 ; OK !!! – pt ca e vorba despre 2 etichete din cadrul aceluiasi segment! – TD va fi SCALAR !!!!

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
mov ch, lg-a ; CH = 04
mov ch, [lg-a] ; mov byte ptr DS:[4] – Access violation – cel mai probabil…

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

mov cx, $-a ; invalid operand type !!!!!
mov cx, $$-a ; invalid operand type !!! ; aici avem $$ din code !!!!
mov cx, a-$; OK !!!!!!

mov ch, $-a ; invalid operand type !!!!!
mov ch, a-$; OBJ format can handle only 16 or 32 byte relocation

mov cx, $-start ; ok !!!
mov cx, start-$; ok !!!

mov ch, $-start ; ok !!!
mov ch, start-$; ok !!!

mov cx, a-start ; ok !!!
mov cx, start-a ; invalid operand type !!!

start1:

mov ah, a+b ; MERGE !!!!!!!!!! DAR …. NU ESTE ADUNARE DE POINTERI !!!
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 !!!!!!

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 daca ambele sunt definite in acelasi segment
- Fie daca et1 apartine unui segment diferit fata de cel in care apare expresia, iar et2 este definita in segmentul in care apare expresia. Intr-un astfel de caz, TD asociat expresiei et1-et2 este POINTER si NU SCALAR (constanta numerica) ca si in cazul etichetelor ce fac parte din acelasi segment.

  Scadere de offset-uri ce se raporteaza la acelasi segment = SCALAR
  Scadere de pointeri din segmente diferite = POINTER

Numele unui segment este asociat cu "Adresa segmentului in memorie in faza de executie" insa aceasta nu ne este disponibila/accesibila, fiind decisa la momentul incarcarii programului pt executie de catre incarcatorul de program al SO. De aceea, daca folosim nume de segmente in expresii din program in mod explicit vom obtine fie eroare de sintaxa (in situatii de tipul $/a-data) fie de link-editare (in situatii de tipul data-a), deoarece practic numele unui segment este asociat cu adresa FAR al acestuia" (adresa care nu va fi cunoscuta decat la momentul incarcarii programului, deci va fi disponibila doar la run-time) si NU este asociat cu "Offset-ul segmentului in faza de asamblare, fiind o constanta determinata la momentul asamblarii" (asa cum este in programarea sub 16 biti de exemplu, unde nume segment in faza de asamblare = offset-ul sau = 0). Ca urmare, se constata ca in programarea sub 32 de biti numele de segmente NU pot fi folosite in expresii !!

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

Sub 32 biti offset (data) = vezi OllyDbg – Intotdeauna DATA segment incepe la offset-ul 00401000, iar CODE segment la offset 00402000. Deci offset-urile inceputurilor de segment nu sunt 0 la fel ca si la programarea sub 16 biti. Din acest punct de vedere sub 32 biti este o mare diferenta intre numele de variabile (al caror offset poate fi determinat la asamblare) si numele de segmente (al caror offset NU poate fi determinat la momentul asamblarii ca si o constanta, aceasta valoare fiind cunoscuta la fel ca si adresa de segment doar la momentul incarcarii programului pt executie - loading time).

Daca avem mai multi operanzi pointeri, asamblorul va incerca sa incadreze expresia intr-o combinatie valida de operatii cu pointeri:

Mov bx, [(v3-v2) ± (v1-v)] – OK !!! (adunarea si scaderea de valori imediate este corecta !!)

…insa nu trebuie sa uitam ca fiind vorba despre ADRESARE INDIRECTA rezultatul final al expresiei cu operanzi pointeri din paranteze trebuie sa furnizeze in final UN POINTER !!!... din aceasta cauza in cazul adresarii indirecte nu vor fi niciodata accepate expresii cu operanzi adrese de forma “a+b”

Daca nu, vom obtine eroarea de sintaxa “Invalid effective address”:

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

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

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) = scadere de valori imediate

Varianta de adresare directa vs.indirecta este mai permisiva ca operatii aritmetice in NASM (din cauza acceptarii expresiilor de tip “a+b”) insa asta nu inseamna ca este mai permisiva IN OPERATIILE CU POINTERI !!!

Mov bx, (v3±v2) ± (v1±v) – OK !!! (adunarea si scaderea de valori imediate este corecta !!)

A[7] = *(A+7) = *(7+A) = 7[A] !!!!