<u>Location counter and pointer arithmetic (discussed with you)</u>

SEGMENT data

a db 1,2,3,4 ; 01 02 03 04 (offset (a) =0)
lg db $-a ; 04; offset (lg) = 4
lg db $-data ; in TASM will be the same effect as above because in TASM, MASM offset(segment-name)=0 !!!; in NASM NO WAY, offset(data) = 00401000 !!!

- Here I will have syntax error !

lg db 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
d equu  a-$    ;  0-6 = -6 = FA
e  db a-$      ;  0-6 = -6 = FA
x dw x         ; 07 10 !!!!!
x db x         ; syntax error !

   db lg-a        ; 4-0 = 4
   db a-lg        ; 0-4 = -4
   db [$-a]              ; syntax error !
   db [lg-a]             ; syntax error !

 ;x dw x     ; x = offset(x)  09 00 la asamblare si in final 09 10

 lg1  EQU lg1          ;  lg1=0 !!!  (because it is a NASM BUG !!!)
 lg1  EQU lg1-a        ;  lg1=0 !!! (because offset(a) =0); if we define something before a
we will obtain …..syntax error !

   b  dd  a-start   ;  syntax error ! (a – defined here, start – defined somewhere else)
      dd  start-a  ;   OK !!!! IT WORKS !!! (start – defined here, a – defined somewhere
else)
      dd  start-start1 ; OK !!! (because both labels belong to the same segment !!!)
```

```
segment code use32
 start:
   mov ah, lg1    ; AH=0
   mov bh, c      ; c is a constant def by EQU, so BH=-6

   mov ch, lg         ; syntax error
   mov ch, lg-a       ; 4
   mov ch, [lg-a]    ;  mov ch, byte ptr DS:[4]    - most probably memory access violation

   mov cx, lg-a        ;  4
   mov cx, [lg-a]     ; mov ch, word ptr DS:[4]    - most probably memory access violation


   mov cx, $-a  ;  syntax error ! ($ – CODE SEGMENT's location counter, a – defined
somewhere else)
   mov cx, a-$  ; ok !!! Possible warning…

   mov ch, $-a  ;  syntax error ! ($ – CODE SEGMENT's location counter, a – defined
somewhere else)
   mov ch, a-$  ;  a-$ IS OK, BUT a-$ IS A POINTER !!!!! – syntax error !!!

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

   mov ch, $-start  ; ok !! – because it is a SCALAR !!!!!
   mov ch, start-$  ; ok !! – because it is a SCALAR !!!!!


   mov cx, a-start   ; ok !!
   mov cx, start-a   ; syntax error ! (start – defined here, a – defined somewhere else)


   start1:

   mov ah, a+b   ; NASM accepts THIS !!! NO SYNTAX ERROR !!! MERGE – DAR NU E
ADUNARE POINTERI !!!!!
```

a+b = (a-$$) + (b-$$) = SCALAR + SCALAR = SCALAR
mov ax, b+a ; idem – ok
mov ax, [a+b] ; Invalid effective address – THIS IS REALLY A POINTER ADDITION !!!

Location counter and pointer arithmetic (prepared by me in advance)

SEGMENT data

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

lg db $-a ; 04
lg db $-data ; syntax error – expression is not simple or relocatable
lg db a-data ; syntax error – expression is not simple or relocatable
lg2 dw data-a; the assembler allows it but we obtain a linking error – "Error in file at 00129 – Invalid fixup recordname count…."
db a-$; = -5 = FB
c equ a-$; = -6 = FA
db lg-a ; 04
db a-lg ; = -4 = FC
db [$-a] ; expression syntax error – a memory contents is not a constant computable at assembly time
db [lg-a] ; expression syntax error – a memory contents is not a constant computable at assembly time

lg1 EQU lg1 ; lg1 = 0 ! (BUG NASM !!!!)
lg1 EQU lg1-a ; lg1 = 0 – WHY ????? It's a BUG !!! It works like that only because offset(a) = 0; if we put something else before a in data segment we will obtain a syntax error : "Recursive EQUs, macro abuse"
g34 dw c-2 ; -8

b dd a-start ; expression is not simple or relocatable !!!
dd start-a ; OK !!!!!!!! - POINTER DATA TYPE!!!!! (because it represents a FAR pointers subtraction !!!!)

dd start-start1 ; OK !!! – because they are 2 labels from the same segment! – Result will be a SCALAR DType !!!!

```
 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 – most probably…

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

    mov cx, $-a  ;  invalid operand type !!!!!
    mov cx, a-$  ;  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   ; NO syntax error !!!!!!!!!! BUT …. It is NO pointer arithmetic, NO
pointers addition
                       a+b = (a-$$) + (b-$$)
    mov ax, b+a   ; AX = (b-$$) + (a-$$) – SCALARS ADDITION !!!!

    mov ax, [a+b]  ; INVALID EFFECTIVE ADDRESS !!!! – THIS represents REALLY a
pointers addition which is FORBIDDEN !!!!!!
```

Conclusion:

Expressions of type et1 – et2 (where et1 and et2 are labels – either code or data) are syntactically accepted by NASM,

- Either if both of them are defined in the same segment
- Either if et1 belongs to a different segment from the one in which the expression appears and et2 is defined in this latter one. In such a case, the expression is accepted and the data type associated to the expression et1-et2 is POINTER and NOT SCALAR (numeric constant) as in the case of an expression composed of labels belonging to the same segment.

  Subtracting offsets specified relative to the same segment = SCALAR
  Subtracting pointers belonging to different segments = POINTER

The name of a segment is associated with "Segment's address in memory at run-time", but this value isn't accessible to us, this being decided only at loading-time by the OS loader. That is why, if we try to use the name of a segment explicitly in our programs we will get either a syntax error (in situations like $/a-data) either a linking error (in situations like data-a). In 16 bits programming a segment's name is associated to its offset and this offset is zero ! In 32 bits programming, the offset of a segment is NOT a constant computable at assembly time and that is why we cannot use it in pointer arithmetic expressions !

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

You can check for instance that NASM and OllyDbg always provide for start offset (DATA) = 00401000, and offset(CODE) = 00402000. So here the offsets aren't zero as in 16 bits programming !! So from this point of view there is a big difference between VARIABLES NAMES (which offsets can be determined at assembly time) and SEGMENTS NAMES (which offsets can NOT be determined at assembly time as a constant, this value being known exactly like the segment's address ONLY at loading time).

If we have multiple pointer operands, the assembler will try to fit the expression into a valid combination of pointer arithmetic operations:

Mov bx, [(v3-v2) ± (v1-v)] – OK !!! (adding and subtracting immediate values is correct !!)

… but we must not forget that in the case of INDIRECT ADDRESSING the final result of the expression with pointer operands in parentheses must finally provide A POINTER !!! from this reason in the case of indirect addressing operands in the form "a + b" will never be accepted !

If not, we will obtain "Invalid effective address "syntax error":

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

Mov bx, v2-v1-v ; Invalid operand type !   mov bx, v2-(v1+v)

Mov bx, v2+v1-v ; ok

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

Mov bx, v3-v2-v1-v ; OK !!! – because... Mov bx, v3-v2-v1-v = Mov bx, (v3-v2)-(v1+v) = subtracting immediate values

The direct vs. indirect addressing variant is more permissive as arithmetic operations in NASM (due to the acceptance of "a + b" type expressions) but this does not mean that it is more permissive IN POINTER OPERATIONS !!!

Mov bx, (v3±v2) ± (v1±v) – OK !!! (adding and subtracting immediate values is correct!!)

Cee ace apare in paranteze () atat in variant cu galben cat si cea verde SUNT SCALARI !!!!!!!!

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 cunoascuta 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] ;
Mov bx, v2-v1-v ;
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 immediate

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 !!)