Vineri 15 dec 18-21 !!

ex 1:

```
mov al, 200
mov bl, -1
cmp al, bl
```

← compares nothig

sets the flags for future jumps

why a subtraction?

$a < b$

$a - b \overset{?}{<} 0$

```
jl et1
```

signed interpretation

; $-56 < -1$

⇒ the jump will be performed

```
ja et 2
```

unsigned interpretation

; $200 > 255$

⇒ the jump will NOT be performed

```
jb et 3
```

unsigned

; $200 < 255$

⇒ the jump will be performed

```
jg et 4
```

signed

; $-56 > -1$

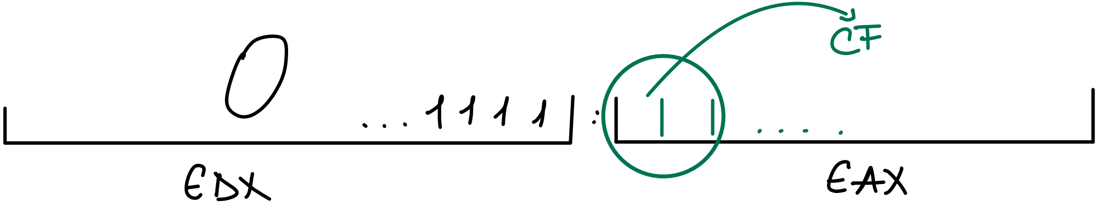
⇒ NO

* You can only compare values part of the same admissabl representation interval

Signed word: $[-32768, +32767]$!

ex 2:

```
xor  edx, edx
mov  dl , 0Fh
```



write a following sequence which multiplies by 4 the value in EDX:EAX

"SHL EDX:EAX, 2" not a valid instruction

the multiplication will remain within the limits of the qword

```
SHL  EAX, 1     ; left most
RCL  EDX, 1

shl  EAX, 1
rcl  EDX, 1
```

not the same as SHL EAX, 2
RCL EDX, 2 !

* Every shift to left multiplies the original value by 2

actual start of lecture

1st recording is the written exam pattern

```
SEGMENT  data
                                       0    1    2    3
offset of a: 0   | a   db  1, 2, 3, 4   ;  01   02   03   04
                                       4
offset of lg: 4  | lg  db  $-a          ;  04
5                |     db  a-$          ;  FB
6                | ( c  equ a-$         ;  -6 = FA
                 | ( e  db  a-$         ;  -6 = FA        OR
7                | x   dw  x            ;  07 00
8                | x1  db  x1
```

⇒ syntax error because "obj format can
only handle 16 or 32 bit

mov ax, x ⟺ lea ax, [x]

mov ah, x1

db lg-a ; 04

db [$-a] } syntax error
db [lg-a]

* rules for prefixing

mov ah, DS:[lg-a] ⟹ runtime error
memory violation error

* you don't have memory at assembler time

you cannot use

the value of an expression MUST always be determinable at .asm time

lg1 equ lg1   lg1= 0   recursive macro echo abo

↑ it is a NASM bug, it works

lg1 equ lg1-a   lg1 = 0

lg1 equ lg1-e

should've been a syntax error

different segments

b1 dd a-start; syntax error

* here - somewhere else

dd start-a; OK!

somewhere else - here works

dd start - start1; OK!

```
segment code
    start:
        mov ah, lg1      ; ≡ 0
        mov ah, lg       ; Syntax error! Cannot put an offset on a byt
        mov ch, lg-a     ; ≡ 04
        mov ch, [lg-a]   ; DS:[04] memory violation error
                           not syntax error

        mov cx, $-a;       syntax error
```

* every segment has its own $ ⇒ here - somewhere else

```
        mov ch, a-$ ;    syntax error because a FAR address doesn't
                         fit a byte
                correct
```

* only if they are part of the same segment they are determinable at assembly time

```
        mov cx, a-$ ;    OK!
        mov cx, $-start
        mov cx, start-$
```

Q: which one works and which one doesn't?

A: They both work !!

```
        mov ch, $-start  } work because it's a scalar and fits on CH
        mov ch, start-$
        mov cx, a-start    works
                start-a    syntax error
```

exam

```
        mov ah, a+b        NOT syntax error
                a+b = (a-$$)+(b-$$)
```

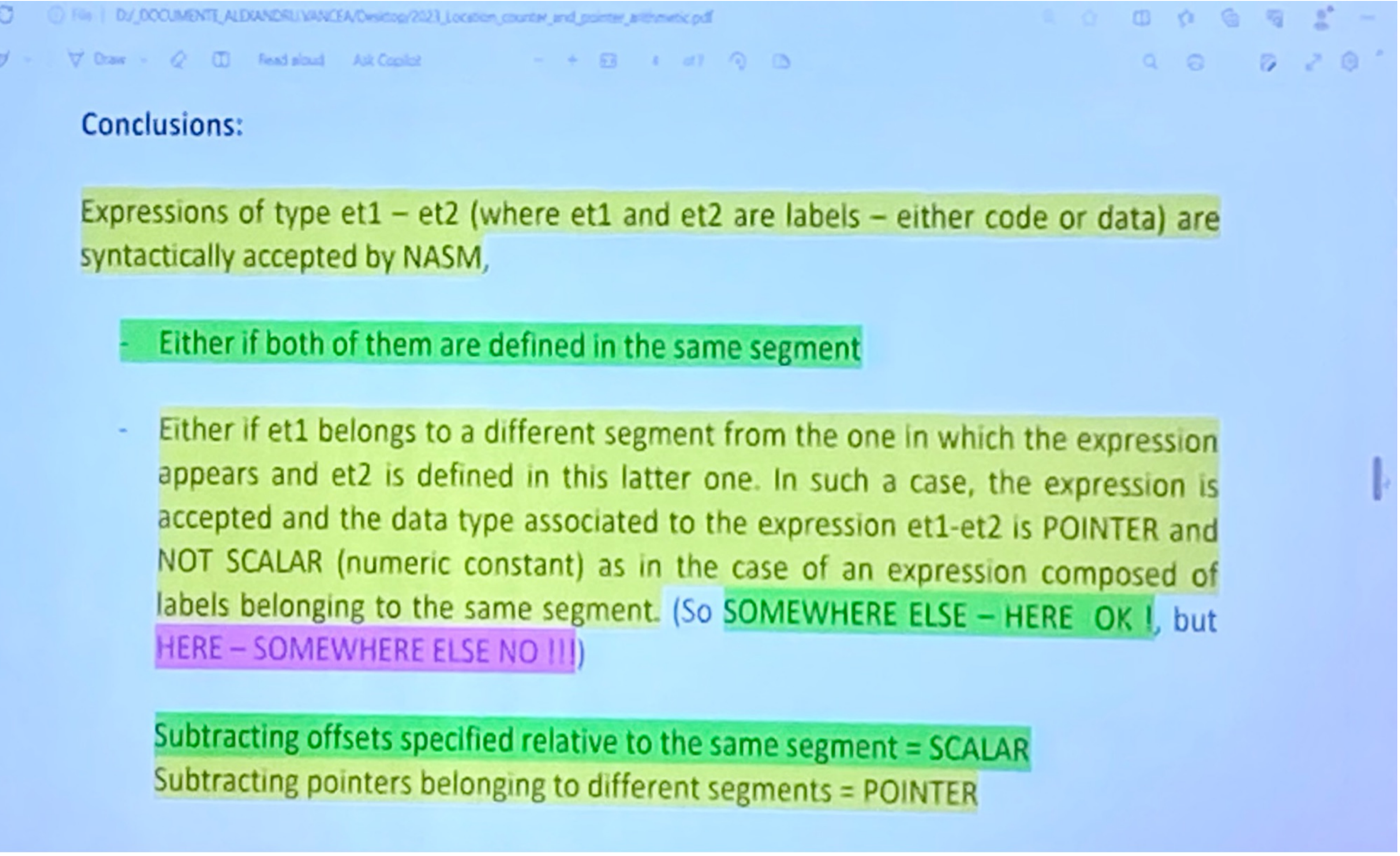
Code segment or data segment ??

NOT pointer arithmetic !!

```
        mov ax, [a+b]    syntax error !!
```

back to code segment:

var 1 dd a+b syntax error!!



Conclusions:

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. (So SOMEWHERE ELSE – HERE OK !, but HERE – SOMEWHERE ELSE NO !!!)

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