little-endian - used for memory representation
- refers to the order of the bytes

12345678 h

| 78h | 56h | 34h | 12h |
↑

- the less significant part has the smallest address.

To what is applied little-endian?

To structures or data types of at least 2 bytes (word, double words or quadwords)

---

instruction name [dest], [source]

i) <u>immediate</u> mode

mov lax, 17

ii) register mode

iii) <u>memory addressing</u> mode

offset_spec = [base] + [index * scale] + [constant]

registers

based addressing

registers

indexed addressing

direct addressing

indirect addressing

As a base you can use anyone of the 8 general registers

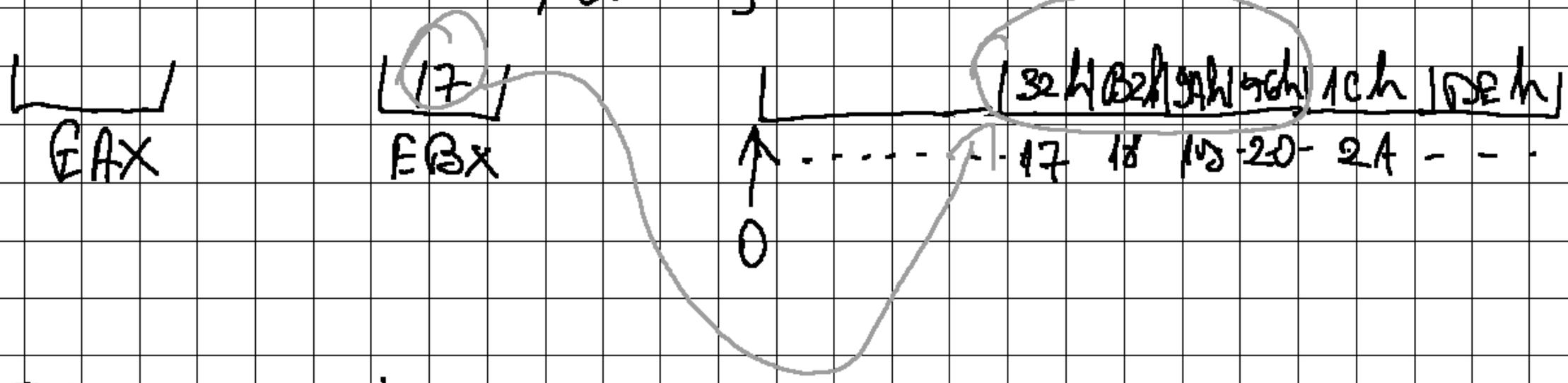
index : 7 possib, all without ESP

scale : constant (by default is 1)

Ex: mov EBX, 17

mov EAX, EBX ; eax = 17

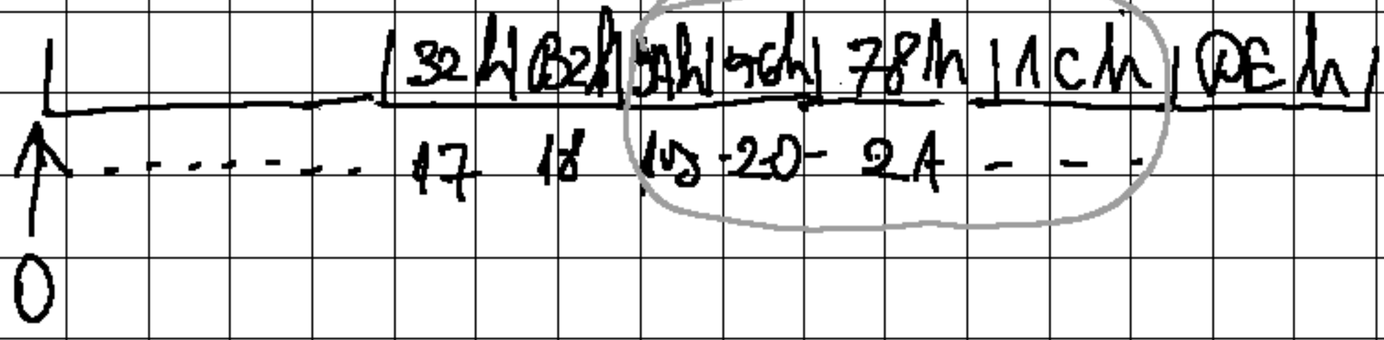
mov EAX, [EBX] : eax = 769AB232h



[] - operator dereferencing

mov eax, ebx+2 → Syntax error message!

mov eax, [ebx+2] ; EAX = 1C78569Ah



mov EAX, [EBX+2*ESP-7] - Syntax error!

mov EAX, [EBX+ESP] - CORRECT

↓

BASE (can only be base)

mov EAX, [EBX*2-7] - CORRECT

mov EAX, [EBX*3-7] ⇒ Syntax error

⇕ Same as

[EBX+EBX*2-7] correct

mov eax, [ebx-edx] → Syntax error

there are no negative registers!!

mov [ebx*2-7], eax - correct

in any programming language, any declared variable will have a fixed address
This address has 2 parts:
- segment part - you don't have access to it
- offset part

it's offset of a variable is always a constant determinable at assembly or compile time

- exe file
  [code segment
  [data segment

Pointer arithmetic:

- adding ⇒ pointers (forbidden) - does not make sense
  multiplication
  division
- substraction (walk through memory)

i) substracting two pointers
ii) adding a constant to a pointer
iii) substracting a constant from a pointer

a[7] = *(a+7) = *(7+a) = 7[a]

i = i + 1

LHS = left hand side
L-value

RHS values
R-values

Symbol := expression

identifier := expression

address_computation_expression = expression

General form

mov [EBX + EDX * 4 - 7], eax

L-value R-value