Why they didn't implement addition and substraction with the result on the same type (byte + byte = byte) just like division and multiplication?

possible Exam question

CF is set to 1

↑ ↗ overflow

262 = 100000110

add with carry ↗

we convert from byte to word using adc

Why the designer of the processor provided 2 diff. flags for overflow (CF and OF)?

Because we have 2 possible diff. interpretation that provide diff. values

How do you tell the processor to perform an addition in the unsigned interpretation

You can't do that, bcs the binary addition will always have 2 diff. interpretation

CF — for unsigned interpretation
OF — for signed interpretation

For division and mult. you have to tell the processor if it's signed or unsigned beforehand.

Why we don't have iADD and iSUB?
their implementation is the same as ADD and SUB.

IF (Trap flag) is good for building debuggers

cli (clears the IF (interrupt flag)) → sets it to 0

mov eax...

critical section

add ebx, ecx

sti (sets the it to 1)

no one needs to have access to this

DF (direction flag) ⇒ you are supposed to set it before a set of instructions, for operating strings
if set to 0, string parsing will be in ascending order
if set to 1, string parsing will be in descending order

- Flags that show us what happened:
  - (CF), OF, PF, ZF, AF, SF
- Flags that have to be set by the programmer for influencing the execution of some instructions:
  - (CF), TF, IF, DF
    ↓ special instructions: adc, sbb

DF: CLD (DF=0)
STD (DF=1)

IF: CLI (IF=0)
STI (IF=1)

CF: CLC (CF=0)
STC (CF=1)
CMC (complement CF) ⇒ CF = !CF

There are no instructions for TF

AF is a CF for nibles

OF is used in data transmission

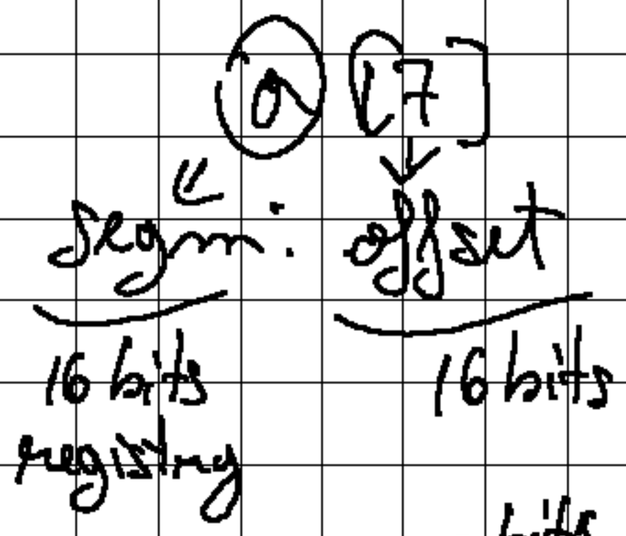
Address registers and address computation

The smallest addresable unit is the byte

Maximum memory allowed abt. 35 years ago was:

$$1MB = 2^{20}$$

it is enough to have a 16 bit registry for the starting address.



Nr.	starting address (base) 32 bits	Limit (size of that segment)
3	07F2A85Ah	1GB
8	7CD90B1h	64kB
-	- - - -	4GB
-	- - - -	
-	- - - -	

$$2^{16} = \underbrace{2^{6}}_{\text{64}} \cdot \underbrace{2^{10}}_{\text{1KB}} = 64\,KB$$

$$2^{32} = \underbrace{2^{30}}_{\text{1GB}} \cdot \underbrace{2^{2}}_{\text{4}} = 4\,GB$$

Why the majority of registers in 16 bit were extended to 32 bits but not the segment

Because these registers were holding the starting addresses of the segments but in 32 bits prog. they are not allowed to hold them, in 32 bits they are only holding a so called segment descriptor which is an index in a table of descriptors managed entirely by the OS.

So we don't need 32 bits values to express indexies in a table, 16 bits are enough

The x86 architecture allows 4 types of segments

- code segment CS
- data segment DS
- stack segment SS
- extra segment ES

CS
DS
SS
ES
} ⇒ 16 bit programming

FS
GS
⇒ 32 bit programming (2 more registers were added)

FIP

CS contains - segment selector corresponding to the code segment

Segment : offset — FAR address

NEAR Address