→ Next week: what are the exact cases when we use the choose complement?

Registers are general because we can do whatever with them

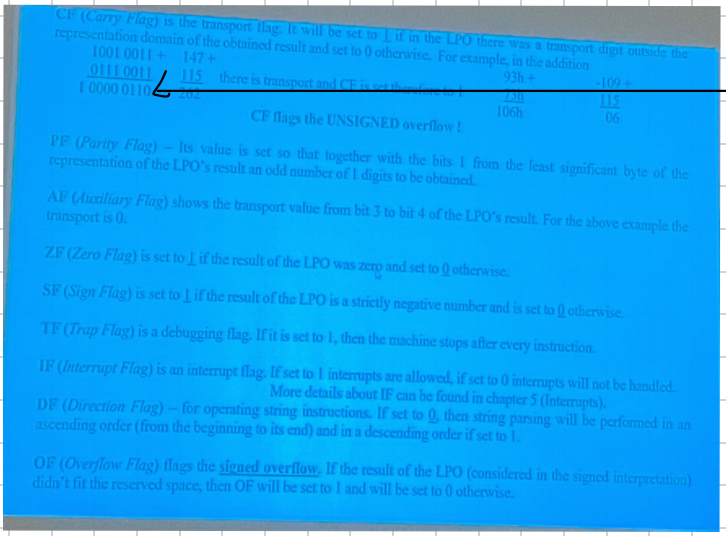
04:20 smt about the stack registers

ESI
EDI } → strings

There are 6 segment registers of size 16 bits + 8 32 address registers

Flags

you are supposed to know all 9 of them BUT for the exam you need: OF, DF, SF, ZF, CF



1 0000 0110,
only this fits in assembly, the (1) is found in CF

* Why didn't they implement ADD and SUB with the result on the same size, not like for MUL or DIV?
↳ Because the only allowable overflow is a (1)
Therefore it is useless to assign 1 byte for only one bit

The flags tell us, from a mathematical and semantic POV, that the result was incorrect for the operation!

CF signals you that the operation was not performed well.

We are targeting a conversion from a byte to a word with CF (14:20 smt)
↳ adc (add with CF)

base 2 → representation

base 16 → compressed representation

* Why did the designers provide 2 different flags for overflow? (16:40)
↳ the addition is only one but we have 2 different interpretations that give us 2 diff. values.

CF signes the overflow in unsigned

OF signes overflow for signed

18:00 Question for exam

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

You cannot, because it will always have two possible interpretations

---

* For MUL and DIV you have to decide BEFORE the implementation what the interpretation is gonna be: MUL / IMUL, DIV / IDIV

* Question exam 21:30

Why do we not have IADD and ISUB?

↳ cause the addition goes the same and the implementation goes the same way

---

PF = parity flag

↳ used in data translations (24:00), to check that the transmition was made in a correct way

↳ how many 1s (bits 1) do we have in the lower part of the value (25:25)

the last result: 0000 0110, because? see definition

AF = auxilary flag

↳ a "carry flag" for nibbles

↳ shows transport from bit 3 to bit 4

↳ we look at posible transport digits in hexa

↳ a half of a byte is a nibble

ZF = zero flag

↳ is set to 1 if the result was ZERO and 0 otherwise

* ASM is a value oriented language ⇒ everything that flows in this language is a numerical value

SF = signed flag

↳ was the value represent a strictly negative number or not?

↳ the sign bit's value goes into SF

TF = trap flag

↳ debugging flag. if it's set to 1, the machine stops after every instruction

↳ every time you press F7, TF is set to 1

IF = interrupt flag

↳ not usable for 32 bits programming

↳ every action is initiated from an interrupt

↳ viruses are just redirects of casual interrupts

↳ what's it useful to?

↳ (41:20)

↳ mov eax ...
⋮
add ebx, ecx
} critical section → nobody can interrupt

↳ CLi = clear interrupt that works directly with the IF

↳ STi = set interrupt flag

DF = direction flag

↳ strings are not datatypes that the processor is aware of

↳ each "string" can be parses ascending or descending

↳ DF = 1 → ascending
DF = 0 → descending

Most flags show us what happend in the previous operation and some ask to be set for future operations

↳ flags that show us what happend:

- (CF), OF, PF, ZF, AF, SF

↳ flags that have to be set by the programmer for influencing the execution of some instructions

- TF, IF, DF, (CF) - part of both

* adc, sbb (add with carry, subtract with ?)

For DF <
- CLD (clear direction flag : DF = 0)
- STD (set direction flag : DF = 1)

CF <
- CLC (CF = 0)
- STC (CF = 1)
- CMC (complement carry flag $CF = \overline{CF}$)

We have (7) instructions to change the value for 3 Flags : (CF), (DF), IF.
↑ shouldn't go to that

Hour 2

Address of a memory location = number of consecutive **bytes** from the beginning of RAM memory and the beginning of that memory location

↳ 35 years ago you had access to $1 MB = 2^{20}$ bytes

* It's enough to have a 16 bits register to have the start of an address and another 16 bits for the offset segm : offset
↳ starting from your specification, BIU will give you the end address
↳ the (segm : offset) pair is an address specification

---

selector table

Nr	Starting Address (base) (32 bits)	Limit (size of that segm.)
3	0752 ABh	1 GB
8	7CD 90Bh	64 MB
		4 GB
		⋮

13 : 50

There's no such a rule that the numbers have to come consecutively or start from 1

* memory validation error
↳ if the processor doesn't find that specific index

↳ a block of memory of discrete size, called physical segment. The number of bytes:

a) 64 KB for 16 bits processor

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

b) 4 GB for 32 bits processor

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

CS
DS
SS
ES
FS
GS
EIP

a segment descriptor !!, they were not extended, therefore they are 16 bits

* Why the majority of registers were extended.
  ↳ they hold only segment descriptors
  ↳ they were holding the starting addresses of that segment, but in 32 bits prog. they are not allowed to hold the address. Now they hold a so-called segment descriptor which is an index in a table managed entirely by the operating system. ⇒ we don't need 32 bits values for expressing some indexes in a table

---

Nr	Starting Address (base) (32 bits)	Limit (size of that segm)
3	0752 ABh	1 GB
8	~~4CD 90B h~~ 2000	64 MB

36:40 → la de la început

address specification: 8 : 1000 h

The difference between physical and logical:

↳ there are logical segments put in a large physical segment

The x86 arch. allows 4 types of segments:

- code segment, which contain instructions
- data segments, data with which instructions work
- stack segment
- extra segment, (supplementary data segment)

ONLY ONE SEGMENT OF EACH CAT. ACTIVE at a TIME

The values contained on CS, DS, SS, ES are the segment selector for the active segment (43:05)

CS - Contains the Segment SELECTOR corresponding to the active segment

*exam question ↑

EIP - tells you which is the offset of your currently executed instruction

you CANNOT change EIP, it is managed directly by BIU

* Last slide Vanica