Order:

- [x] v3 - Registers - and - Flags ... (1)
- [x] Address Registers and Address ... (2)
- [x] Machine instr. Repres. (3)
- [x] 2022 - 2's complement (4)
- [x] Overflowing concept analysis (5)
- [x] Chapter 3 Basic elem. of Assembly (6)
- [x] Chapter 4 Instructions (7)
- [ ] Exemple comentate (8) Read again
- [x] December - 22 (9) →
- [x] Multimodul Programming (10) ✓
- [x] Instruction Prefix (11)
- [x] Conversions Clasification (12)

Data definition directives:

Data segment starts: 00401000

If a db 102h => it will take

only a byte ⇒ 02h

FF ⇒ 0FFh

a — address ou 32 bits dword OR 16 bits word

---

operators — work with constant values
def. at assembly time
(ex) A.S.F. '+')

Data definition directives: (db, dw, dd)
are NOT data types def. mechanisms

Bitwise operations:

AND — & (force 0 values) , OR — | (force 1 values)

XOR, ^ (complementing values)

! → Logic negation !x=0 , x≠0 else 1

~ → 1's complement mov al, ~0 ⇒ 0ffh

-x ⇒ 2's complement

$ – position at the beginning of the line containg the current expression

$$ – beginning of the current section

SECTION directive

"+" – ASSEMBLY TIME

ADD – runtime

QWORD – memory explicitly present

Overflow: (Addition)

Ex| a,b > 128 and a+b > 256

ex1| 2 neg numbers add up to a positive
OF=1 CF=1

ex2| 2 pos. nrs add up to a negative
CF=0 OF=1

Substraction:

```
1 - - -          0 - - - -
-------   or     ---------
0 - - -          1 - - - -
```

```
neg -            pos -
pos              neg
------  OF=1     ------
pos              neg
```

CF = 1 if $a - b < 0$ in unsigned and we need to borrow!

Multiplication => can't exceed the reserved space

CF = OF everytime $\begin{cases} 0 & \text{if byte} \times \text{byte} = \text{byte} \\ 1 & \text{otherwise} \end{cases}$

Division

word : byte ≠ byte => OF -> division overflow!

=> 'Run-time error' => { Div by 0 / Div. overflow!

We don't care about the flags because it stops the exec. of the programme

---

Why do we need CF and OF in EFLAGS simultaneously?

ADD = iADD (doesn't exist) SUB = iSUB

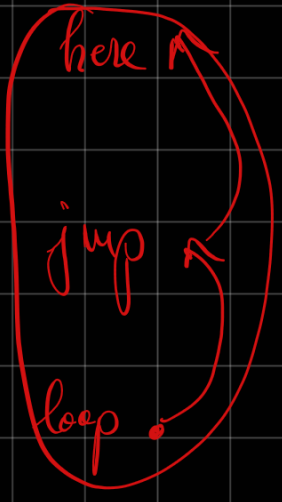
iMUL, iDiv? Why? => They work differently in signed / unsigned.

NASM short jumps (max 127 bytes)

condition imposed ONLY on the LOOP instruction! Replace it with

```
dec ecx
jne label
```

it's fine, or create a midpoint/intermediate jump location



- MOVZX <reg>, <reg/mem>
  - size(1) > size(2)
  - unsigned (zero Extension)
- MOVSX <reg>, <reg/mem>
  - signed

CWD → for (i)MUL, (i)DIV and backwards compatibility

In registers NOT Little endian

LOOP doesn't affect any flags!

Stack

Stack goes from Big → SMALL values

---

a dw 0 → ✓

a db a → X (w or dword only)

a equ a ⇒ Error, uninitialised

a - $ → ✓	⇒ POINTER
$ - a → NOT OK X	

Somewhere Else — Here is OK

Here — Somewhere Else NOT OK

Subtracting offsets of the same segment
⇒ SCALAR VALUE

Subtracting offsets from different segments
⇒ POINTER VALUE

Name of a segment = FAR ADDRESS
(available at runtime)
on 32 bits.

=> syntax error for:

a_data, $_data, data_a....

Can't be used in pointer arithmatic expressions.

16 bits => constant offset det. at assembly time

a+b accepted as scalars?

(a-$$)+(b-$$) only when in combination with scalar operations.

Here -> somewhere else — NO — Invalid operand type

Somewhere else -> Here — OK ✓

Same segment -> OK

Kinda pointer addition

mov ax, a+b → (a-$$)+(b-$$)

BUT mov ax, [a+b] → pointer addition

=> INVALID EFFECTIVE ADDRESS

a+b not allowed as DATA DEFINITION

⇒ syntax error !

---

lbl1 − lbl2 ⇒ Scalar: same label or somewhere else → here

pointer → NOT SCALAR

name of segment → address in run-time, 0.5.

⇒
- a − data ⇒ (Here − Somewhere else)
  ⇒ Syntax Error
- data − a ⇒ (Somewhere else − Here)
  ⇒ Linking Error

If a+b appears along pointer expressions

⇒ Error ! Invalid op. type.

Jump instructions

jmp is a near jump

⇒ Even if we prefix it with DS, SS ...

The jmp will STILL be DONE in CS segment because it is a NEAR JUMP

=> Prefixing only def. the far address, offset rel. to which segment it refers to.

=> CS:(EIP) → only EIP modified.

For FAR jmps => jmp far [ds:ebx]

↓

(CS):(EIP) → both modified.

FAR jumps → 48 bits memory addressing operands.

We cannot FAR jmp with registers because they contain only 32 bits < 48 bits.

---

x86 Instr. Prefix Bytes

1. REP, REPE, REPNE
2. Segment override: CS:, DS:, SS:...
3. Op. override (16/32 bit) Bits 16 Bits 32

④ Addr. override (16/32 bit addr)