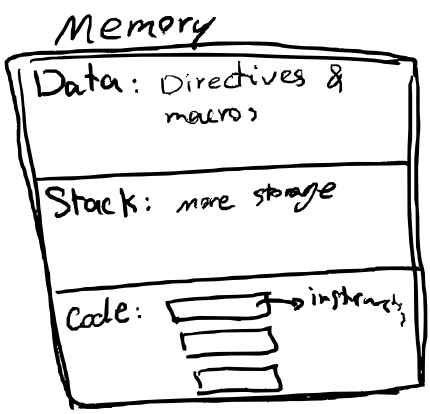
Statements

- Directives / Pseudo-OPs
- Instructions
- Macros



Instructions

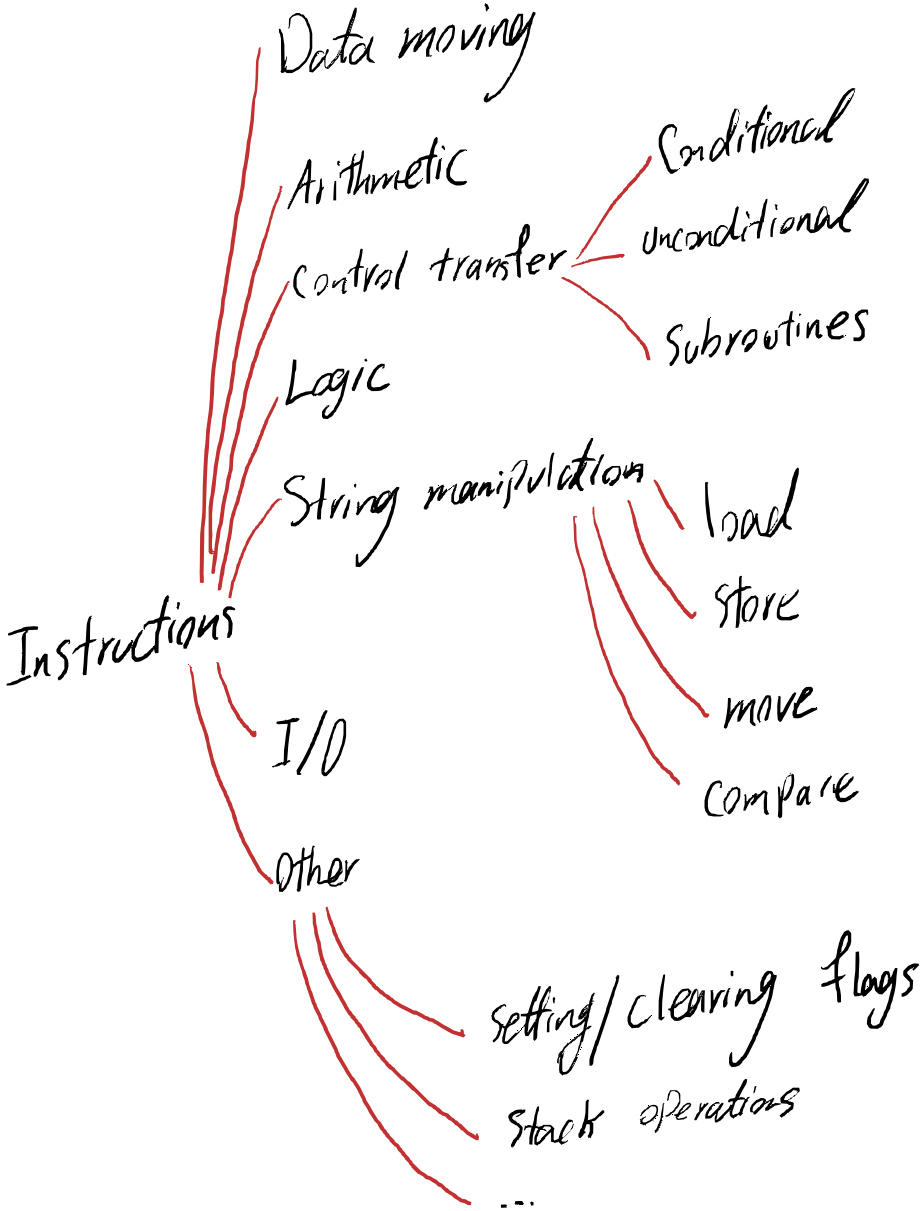
Operation [operands...]

like

```
mov     eax, ebx

mul     ebx     →     eax *= ebx
div                   /=
```

eax is implicit here.


Instructions
Data moving
Arithmetic
Control transfer
Conditional
unconditional
Subroutines
Logic
String manipulation
load
store
move
compare
I/O
Other
setting/clearing flags
stack operations
...

MOV

mov destination, source

LEA (Load Effective Address)

lea register, memory_address

* why effective? since it calculates the final address ex. lea dx, [SI +4].

XCHG (exchange)

xchg register, memory_address

* like mov (and most of other instructions) we can't use it with both memory_address.

INC (increment)
DEC (Decrement)

inc destination
dec destination

Add
Sub

add destination, source
sub destination, source

Mul

mul source

* It uses AL as destination!

* unsigned mul

IMUL

mul source

* Again AL as destination

* Signed mul

DIV, IDIV

div source
idiv source

* implicit AX destination

* It returns → Quotient AL
→ Remainder AH

$$\frac{AX}{\text{Divisor}} = \underset{AL}{\text{Quotient}}, \quad \underset{AH}{\text{Remainder}}$$

* Overflow in multiplication can be easily handled by using AX, but in division it is a problem, and it generates an special software interrupt when over flow happens.

$$\frac{\overset{\text{Dividend}}{24}}{\underset{\text{Divisor}}{13}} = \overbrace{1 \cdot 13}^{\text{Quotient}} + \overbrace{11}^{\text{Remainder}}$$

* Both multiplication & Division can be used for 32-bit operands as well by using DX register.

$$\frac{\underset{\boxed{16\text{-bit}}}{DX \quad AX}}{} = \underset{AX}{\text{Quotient}}, \quad \underset{DX}{\text{Remainder}}$$