Date: __/__/20__ Lec 2 Day: M T W T F S

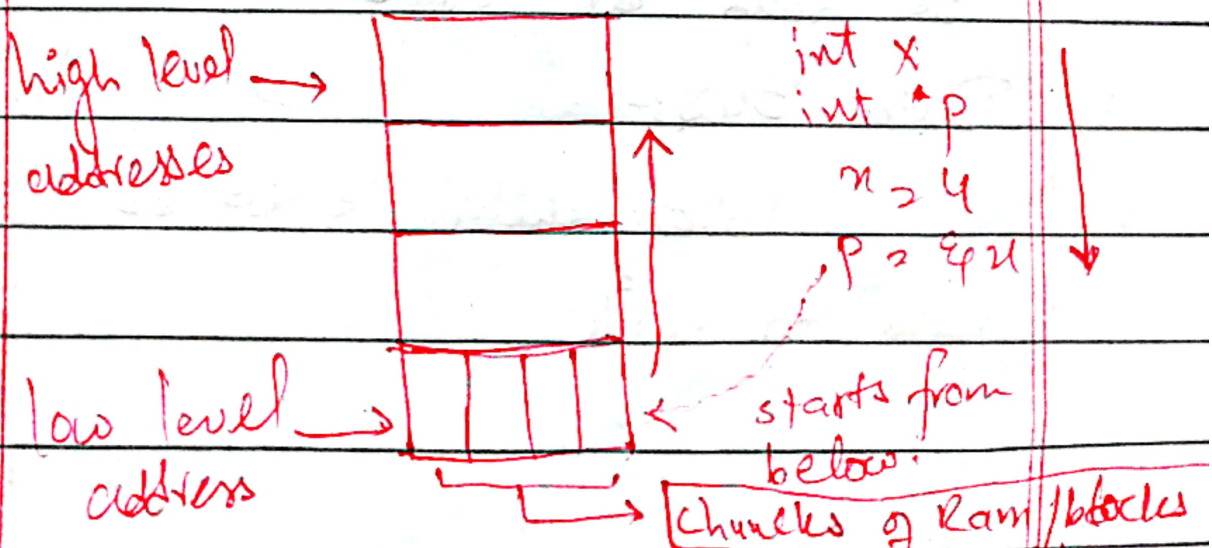
PF/OOP → DLD
↳ 00010
00100

a = 2
b = 2
c = a+b

Source code .cpp (high level lang) → compiler (converts high level lang to assembly lang) → assembler → machine (Hexa codes) → Hard disk → Executes

↳ Assembly lange:-
Code understandable by both humans & machines

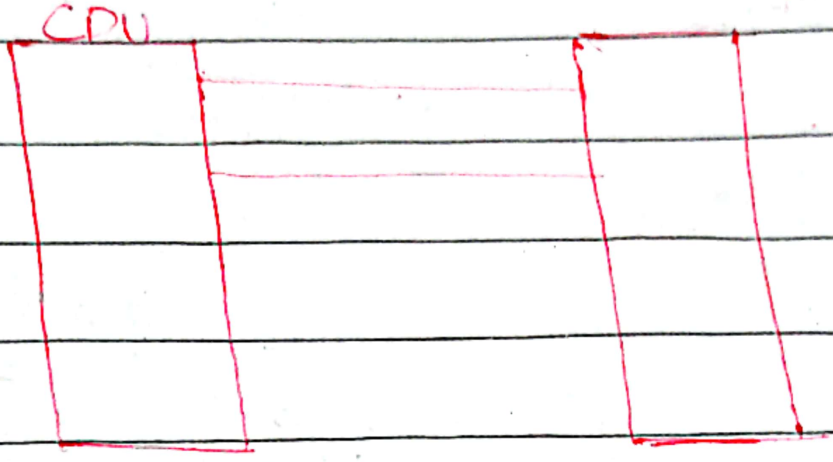
↳ Memory Structure:- (RAM structure)



↳ 1 int is 32 bit
↳ 1 byte is 8 bits

Date: __/__/20__ Day: M T W T F S

↳ addresses are always in hexa decimal form



↳ Read = 00001
write = 000010

① address Bus
② Data Bus
③ Control Bus

execution process.

↳ Control Bus :-

wether to read or write

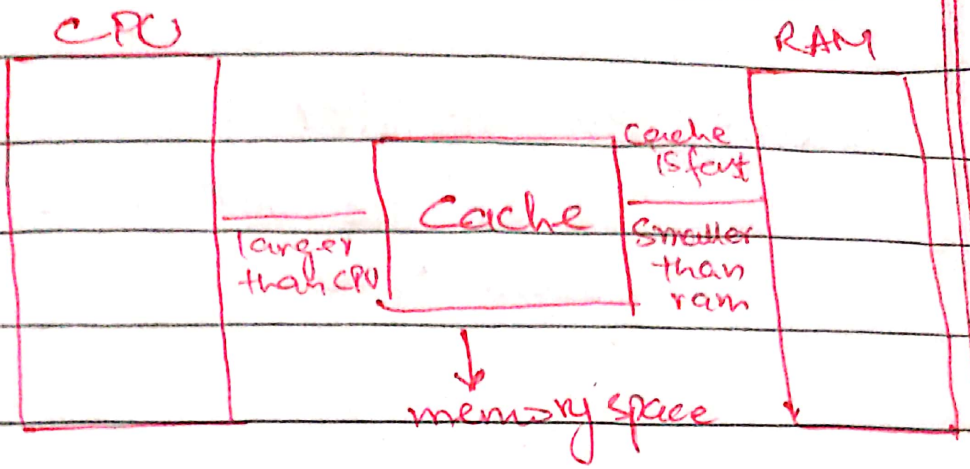
↳ Address Bus :-

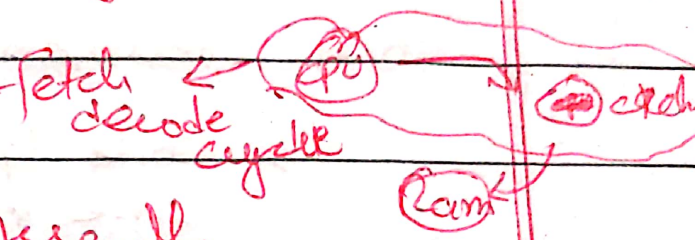
which address is to be read or write

↳ Data Bus :-

what data is to be read or write

Date: __/__/20__ Day: M T W T F S





↳ Hello World of Assembly

↳ read from ram and bring to registers.

↳ mov ax, 25 {Example}

operator | operand

register

move 25 value to ax

↳ inc ax

operation operand

ax value is incremented

Date: __/__/20__ Day: M T W T F S

↳ Accumlator registor
↳ size of accumlator is system size

↳ Vonn Neumann Architecture:
↳ code & data both in same memory
↳ general purpose registers.

↳ ① first it was fetch execute cycle
② now it is vonn neuman

page 55 — i7
page 57 — GP.
page 67 — CPU
page 76 — registers
AX is accumulator registor.

Date: __/__/20__ Day: M T W T F S

↳ DS = Data Segments

↳ IP = Instruction Pointer

↳ System on instruction

---

* org [0x100] → originate from 100

* machine code

↳ opcode of instruction.

0000000

↓

* location * System Calls

mov ax, 4c00

int 0x21

cpu.

instruction list ← handler

↳ exit code instruction

Little endian architecture followed by intel.

lowest significant bit at lowest position.

Date: __/__/20__

- mov ax, [num1]

[0507A]

[num1]
↓
go to num1 location
↓
fetch data and show.

- Segments / Memory Frame CS/DS

0000
↓
nibble
2 bytes
16 bits

1 Location → 1 byte

* functions ← stack
* dynamic runtime memory ← heap
+ extended segment :- data
Data stored on other space code

in ram can be accessed easily.

Date: __/__/20__ Day: M T W T F S

* Code segment :-

CS → stores base/codeseg

DS → data segment

CS | DS → location saving registers.

processor adds 0 ~~to~~ an extra nibble automatically.

1DPD0 → offset
0100 → logical address.
1DE D0 → actual address.

~~~~~

dh dl
↑ ↑
ax 0000 divide register
bx
cx
dx

flag. zf → in previous execution.
any logical / arithmetic logic
xor
and
or
makes register zero

* addressing by registers
~~~~~

Date: __/__/20__ Day: M T W T F S

* flag is either set or un-set. (0/1)
  1 0

"jnz" - jump not zero
if condition is not zero.

"cmp" - Compare.

* Clear registers befor carrying process

method 1 :- moving zero

method 2 :- XOR

* Bus can write at time or read at time.

* Assembler treats jnz and jne as same.

* "SF" sign flag, works according to sign value

Date: __/__/20__ Day: M T W T F S

* Nop :- No operand
  ↳ no instruction.
  ↳ operand in hexa (90)

* Jumps :- (takes 3 bytes)
  ① Conditional jumps
  ② unconditional jumps.

~~loop~~ jmp
~~data~~ data → direct jump with no condition.
~~loop~~ start

① ↳ ~~Also purchase shop supply~~
~~Amount to be paid are so days.~~
~~Usage in 4 month~~

* Unconditional jump:-
  when no condition
  is there, e.g (90)

Day: M T W T F S

Date: __/__/20__

* No memory to memory swapping is possible.