- Lecture -12 - <11/10/2019> <16/10/2019>

Computer organisation and Architecture.

Summary:-

- Digital components
  - ALU
  - Memory
  - Registers
- Instruction set- Architecture
  - Formal-
  - Encoding
  - Addressing mode
- Machine & Assembly language programing

- Computer organisation :-

Question:- How to organise all the digital components such that a program could be executed?

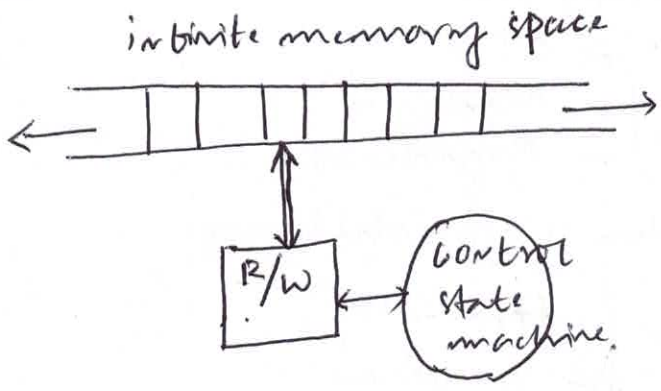
The principle:- Stored program approach

The idea:-

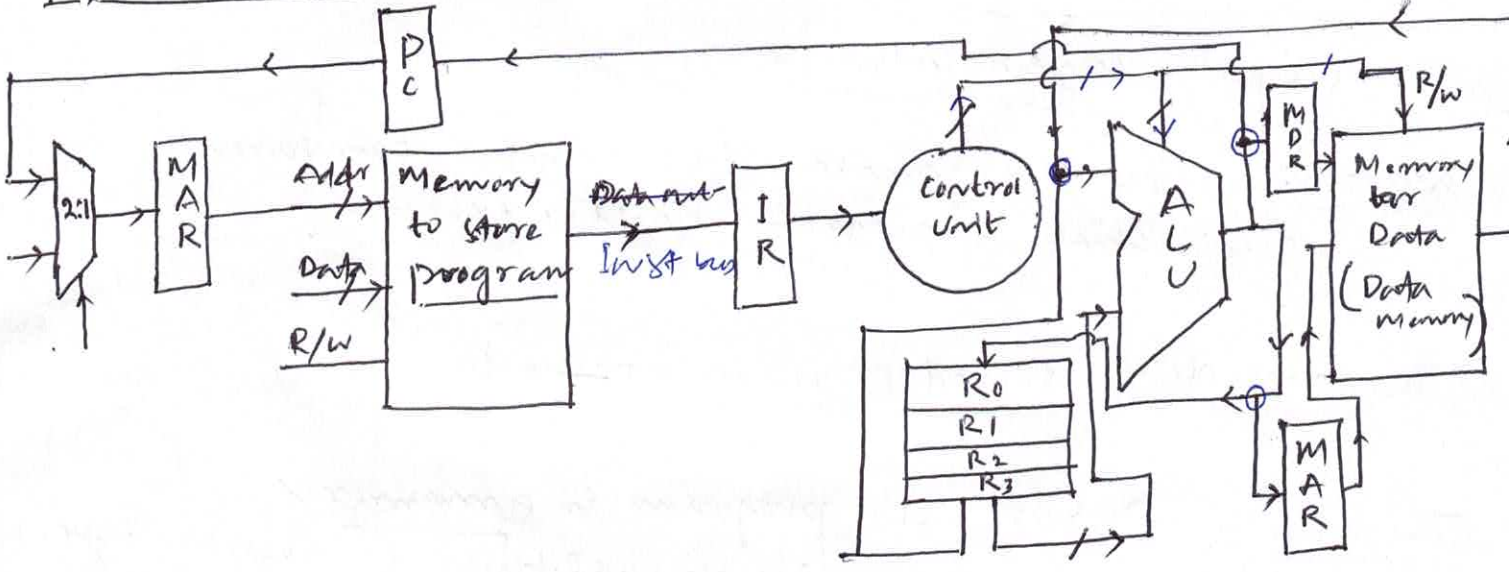
1. Store the program in a memory (Set of well defined instruction) → unambiguous
2. Fetch the instruction (~~the big one~~)
3. Decode the instruction to identify control signals and operand.
4. Fetch the operand
5. Execute the instruction using control signals and operand
6. Store back the results (Changing the current-state of the machine)

Von-Neumann Architecture | Turing Machine –

Turing Machine :-



Von-Neuman Architecture :-



→ The connections in the above figure are conceptual

→ The exact data & control path with multiplexer steering logic & interconnect will be drawn later on.

The Microoperations :-

– The micro steps required to execute an instruction through the five major steps.

Micro operations for "HACK" processor (Nand to Tetris processor) :-

① Storing program at the first place

- This is a work of resource manager (operating system)

MAR ← Address of the stored program

Memory address bus ← [MAR]

Memory Data bus ← ~~Addr~~ Instruction as a byte or word

All these microoperations be repeated to load a program from Hard disk to main memory

(in the modern "processor" computer this is being done by DMA - Direct memory Access)

* In the HACK processor the program is being loaded into ROM memory through the means of simulator

② Instruction Fetch :-

(IF)

- ~~Address of the instruction required~~
- ~~place to keep the instruction once fetched~~
- ~~Any post processing~~.

- Requirement :-

- Address of the instruction to be fetched.
- place to keep the fetched instruction.
- post processing (updating PC content)

[Address bus] ← Content of [PC]

~~R~~/w ← 0

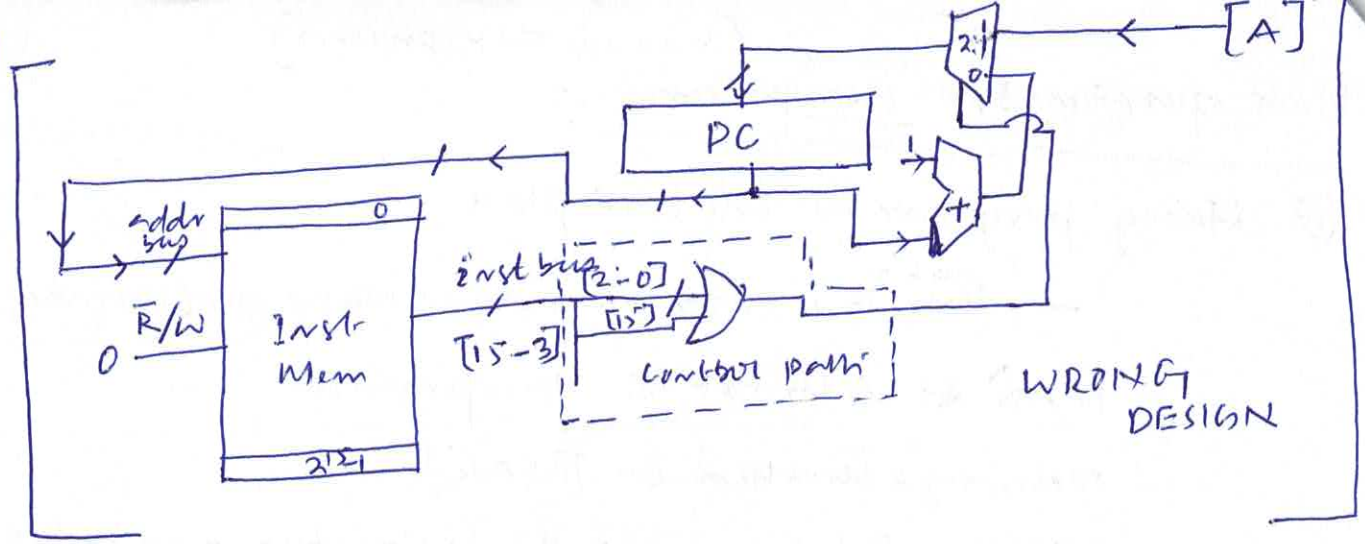
[Instruction bus] ← Memory [Add bus]

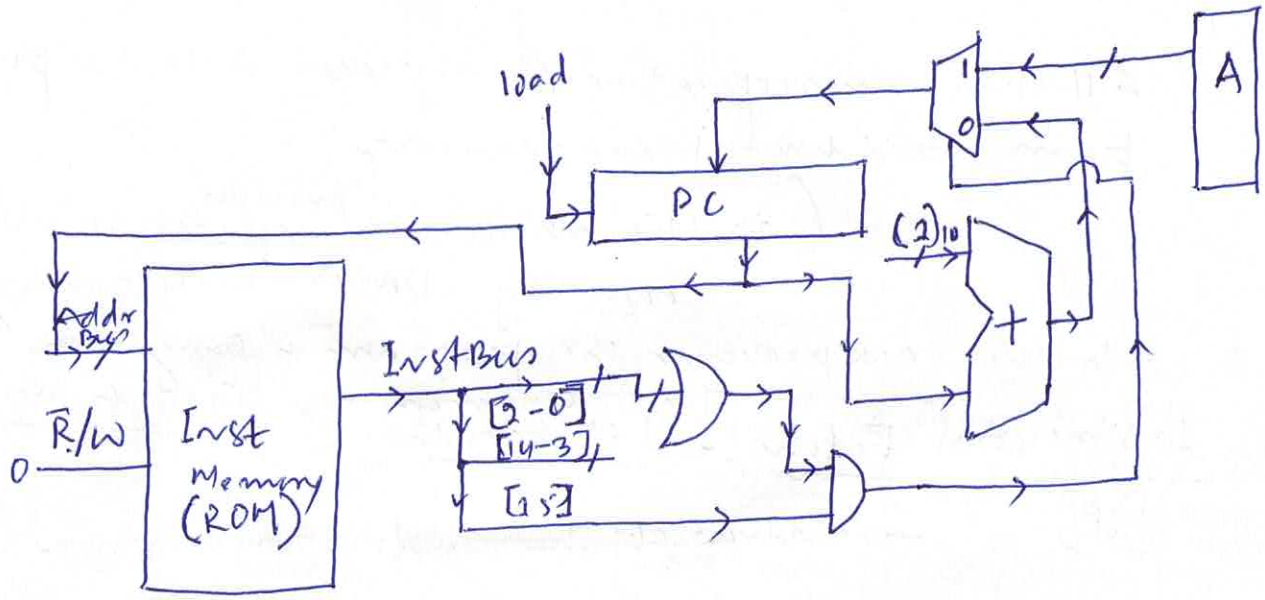
if instruction [2|1|0] ≠ 0 then

PC ← [A]

else PC ← PC + 1

(4)





Instruction Decode :- (ID)

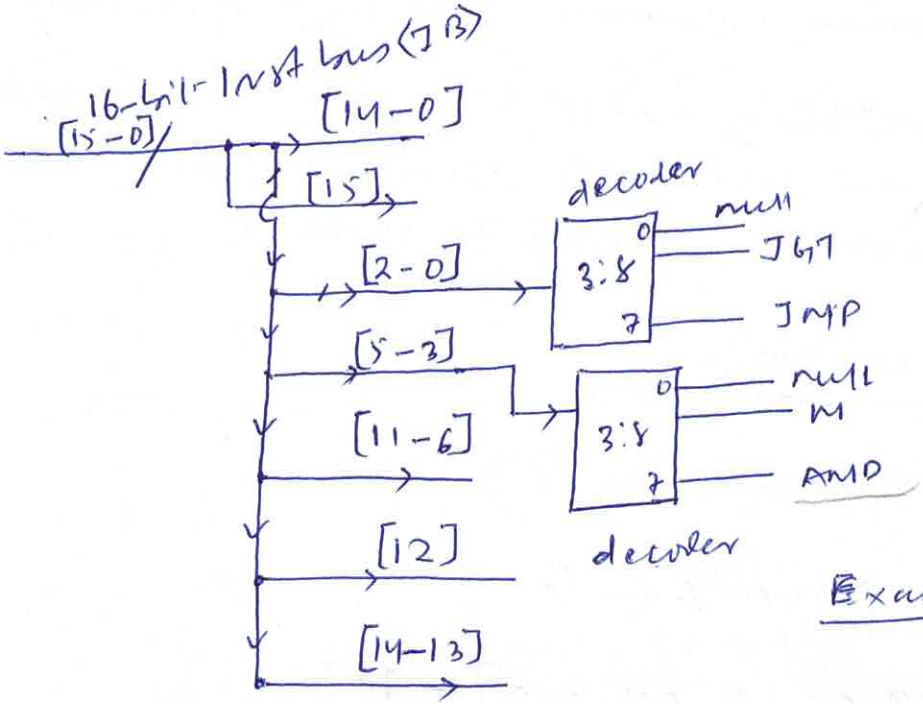
Decoding is essential to understand the control signals and operands.

Micro operations :-

Control logic ← [Instruction bus] ← content of

Control signals ← (Control logic) ← function of

"partial" control path design :-



Example :-

x86 (CISC) - microprogram control

RISC - Hardware controlled

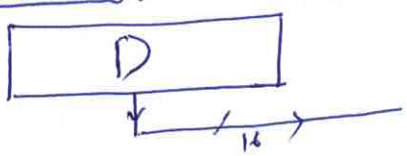
Operand Fetch (OF)

Two register → 1) - a data register

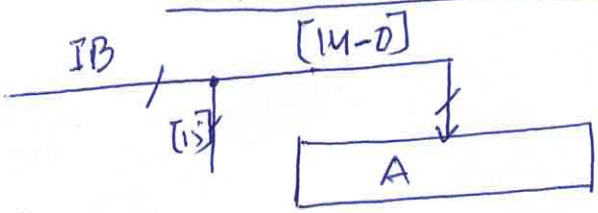
A - can be used as data register also as address register

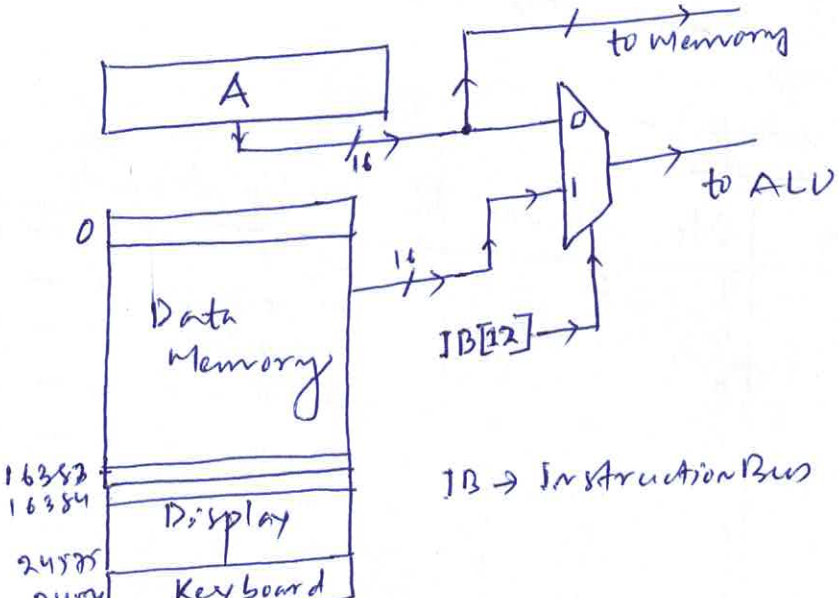
Memory → M - memory

ALU instruction :-



Address/Mem instruction





IB → Instruction Bus

(6)

Execute (EX)

- To perform arithmetic and logic operation
- The necessary control signals are appropriately connected to steer the operand to ALU and execute the instruction according to op-code.

micro operations :-

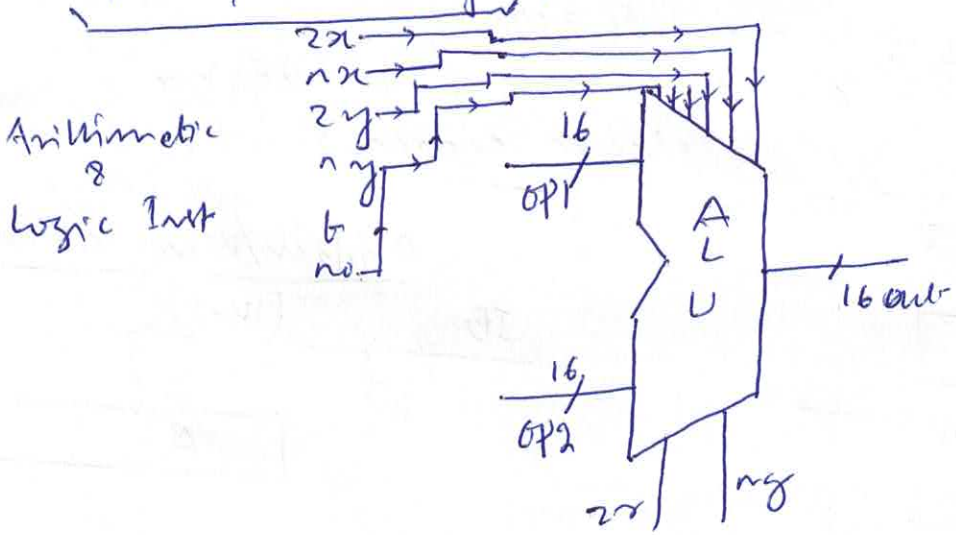
operand1 ← [ID]

operand2 ← [AM]

[zx, nx, zy, ny, f, no] ← IB[11-6]

databus ← ALUout

Data path Design :-



Data path for Jump :-

J1	J2	J3	Jd0	Jd1	Jd2	Jd3	Jd4	Jd5	Jd6	Jd7	
0	0	0	1	0	0	0	0	0	0	0	— Null
0	0	1	0	1	0	0	0	0	0	0	— out > 0
0	1	0	0	0	1	0	0	0	0	0	— out = 0
0	1	1	0	0	0	1	0	0	0	0	— out ≥ 0
1	0	0	0	0	0	0	1	0	0	0	— out < 0
1	0	1	0	0	0	0	0	1	0	0	— out ≠ 0
1	1	0	0	0	0	0	0	0	1	0	— out ≤ 0
1	1	1	0	0	0	0	0	0	0	1	— JMP

zr = 1 if out = 0 else zr = 0

ng = 1 if out < 0 else ng = 0

out = 0 → zr = 1

out < 0 → ng = 1

out > 0 ⇒ out ≠ 0 & out ≮ 0

zr = 0 & ng = 0

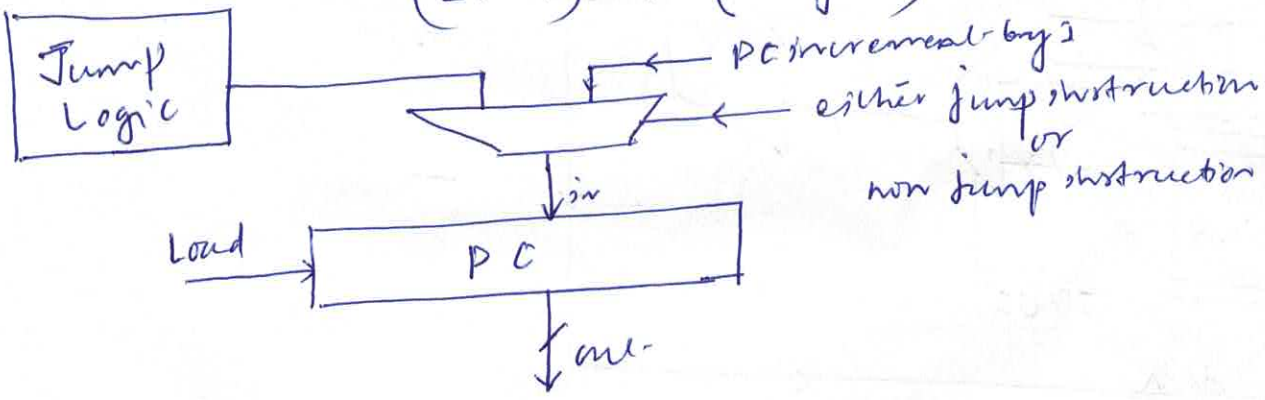
out ≥ 0 ⇒ out = 0 OR ~~out < 0~~ out > 0

(zr = 1) OR (zr = 0 & ng = 0)

out ≠ 0 → zr = 0

out ≤ 0 ⇒ out = 0 OR out < 0

(zr = 1) OR (ng = 1)



Home work :- Complete the design of JUMP Logic using the above conditional control signals.

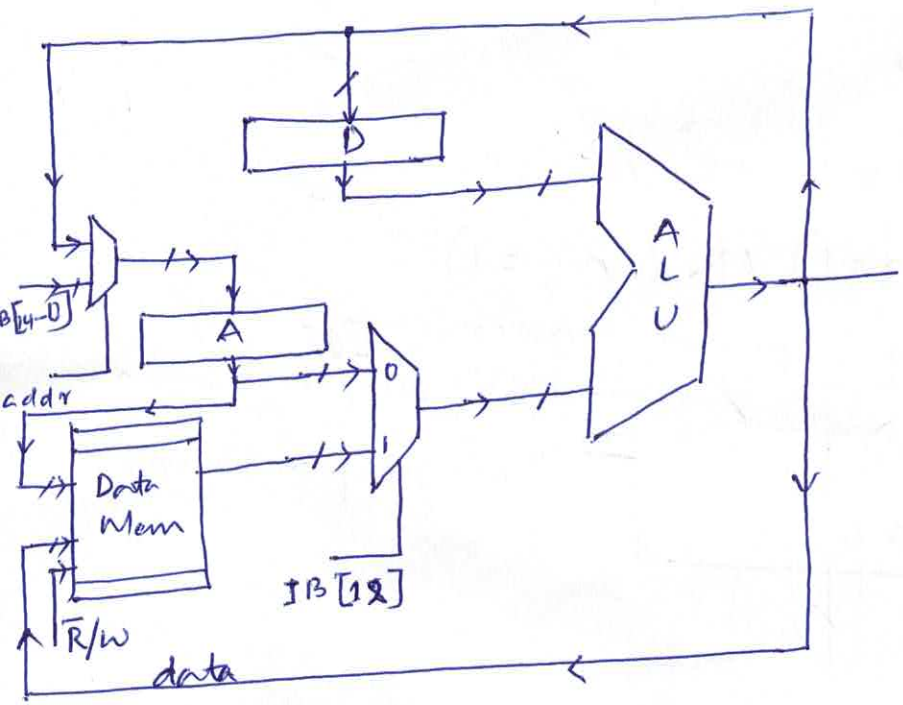
(8)

Writing back the result (WB) :-

- Writing back the result computed by ALU to respective destination.

Note :- Address / memory instruction doesn't require write back

ALU & Jump Instruction :-



updating program counter :-