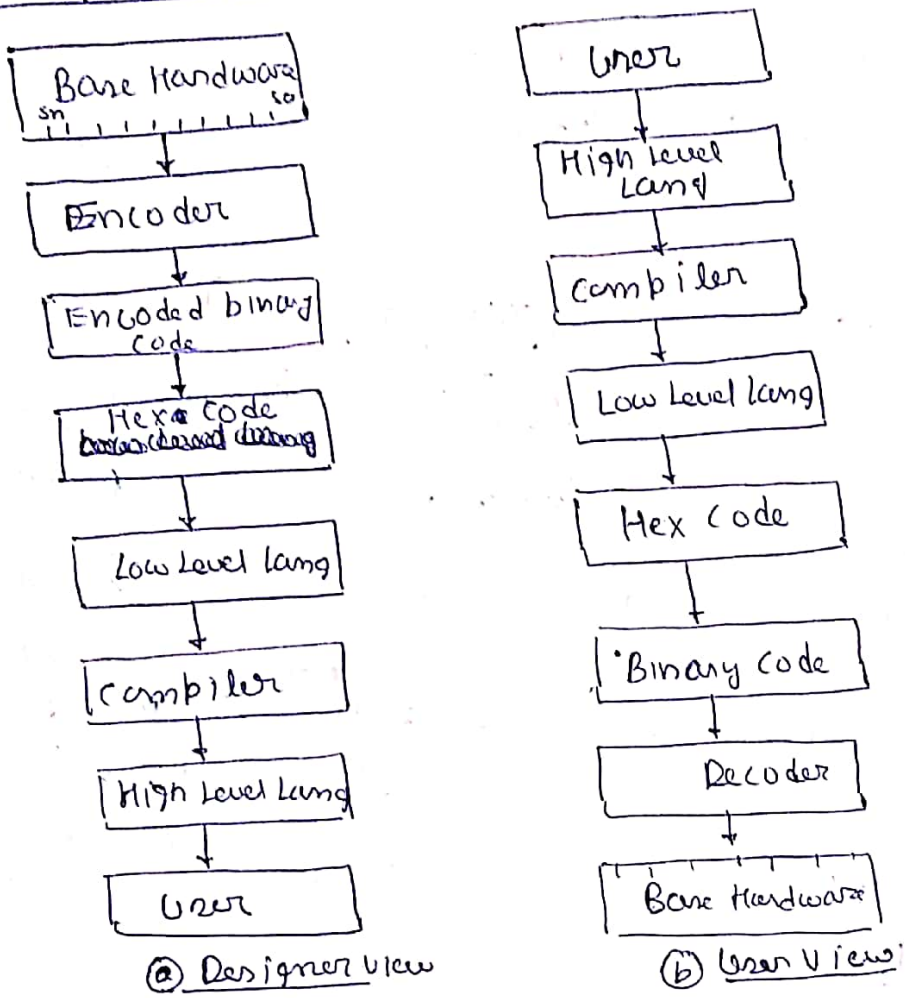
Computer Organization

Computer Organisation & Design

Module 1 : Computer Functionality

1. Basics : (i) Computer (ii) Program (iii) Instruction (iv) Encoding (v) Decoding

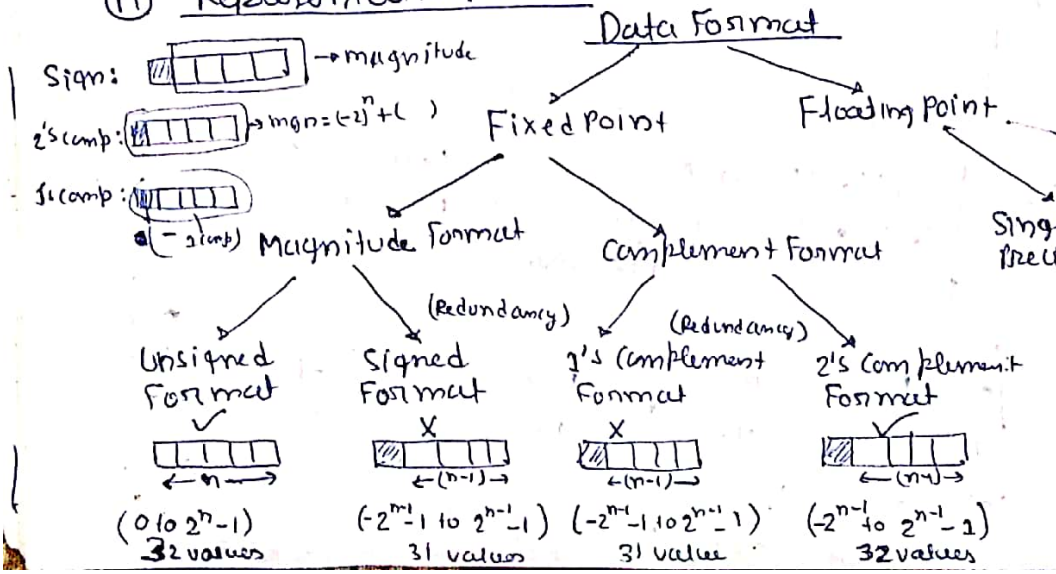
2. Designer and User View of Computer

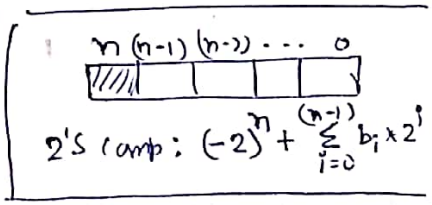


3. Data Representation

(i) Data

(ii) Representation chart





(iii) Unsigned Number: (0 to $2^n - 1$)

(a) There is no overflow case.

(b) If range is exceeded the carry flag is set.

unsigned

CY	Data

(iv) Signed Data (-2^{n-1} to $2^{n-1} - 1$)

(a) There is overflow case in it, so an overflow flag is require. to handle range exceeding.

(v) Addition: (n-bit) + (n-bit) = (n+1) bit

(vi) Multiplication: (n-bit) * (n bit) = (2n) bit ∴ total multiplication possible $(2^n \times 2^n)$ (2n) bits ROM

(vii) Overflow case Signed Data:

(a)
$$\begin{array}{r} X\ x_1\ x_2\ x_3 \\ +\ Y\ y_1\ y_2\ y_3 \\ \hline Z\ z_1\ z_2\ z_3 \end{array} \rightarrow \text{Overflow} = (\underbrace{xy\bar{z}}_{\text{Adding two negative we get '+' (which is not possible)}} + \underbrace{\bar{x}\bar{y}z}_{\text{Adding two positive we get negative no. (which is not possible)}})$$

(b) We only consider Overflow Flag in case of Signed Number, no matter if carry flag is set or not. (Recall of addition is also in)

(c) If OV = 1 (Flag is set) then:

if $\bar{X}\bar{Y}Z = = 1$ (positive overflow) :

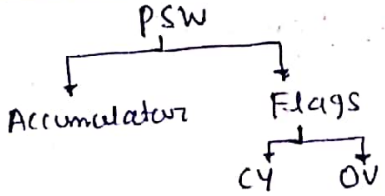
extra bit	original bit
MSB = 0	Data

if $XY\bar{Z} = = 1$ (negative overflow) :

MSB = 1	Data

include Sign bit also

(viii) Program Status Word : Use to represent data format)



(ix) Floating Point Representation

(a) $\pm M * B^{\pm e}$ M = Mantisa, B = Base, e = exponent

n = Biased Bit

Excess Bias = $\left(\frac{2^n}{2}\right)$

(b) Normalize form: $\pm \underbrace{1.}_{\text{Implicit}} M * B^{\pm e}$

Move right →

111.111 ⇒ Normalize ⇒ $1.11111 \times 2^{+2}$

0.0001111 ⇒ Normalize ⇒ 1.11×2^{-4}

← Move left

(c) Single Precision (32 bit) :

32 bit

Sign	Biased Exponent	Mantisa
1 bit	8 bit	23 bit

n = 8

Bias = $+(2^{n-1} - 1) = +127$

BE = Actual E + Bias

(d) Double Precision :

Sign	Biased Exponent	Mantisa
1 bit	11 bit	52 bit

n = 11

Bias = $+(2^{n-1} - 1) = +1023$

BE = Actual E + Bias

② Range of Floating Point

$$\pm M * 2^{(BE - Bias)}$$

Bias : +127

Sign [+] BE
- All 0's, AE : (−127) → M
  - All 0's (1.0) → 0 } round to 0
  - All 1's (1.111) } Data
- All 1's, AE : (+128) → M
  - All 0's → +∞
  - All 1's → } NAN
  - } round to +∞

(Normalize)

BE range = { −127 to +127 }

Mentisa range = { 1 to $(2 - 2^{-23})$ } ← $(2^n - 1) * 2^{Range}$

Bias : +1023

Sign [+] BE
- All 0's, AE:(−1023) → M
  - All 0's → (1.0) 0 } round to 0
  - All 1's } Data
- All 1's, AE : (+1024) → M
  - All 0's → +∞
  - All 1's → } NAN
  - } round to +∞

(Normalize)

BE range = { −1023 to +1024 }

Mentisa range = { 1 to $(2 - 2^{-52})$ }

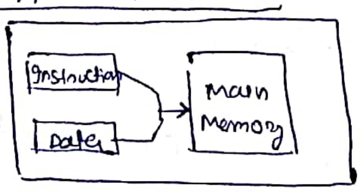
(1111.... 53 times * 2^{-52})

---

④ Computer Architecture

ⓘ Von-Neuman Architecture

* Dynamic over Program
* Program must present in main memory.

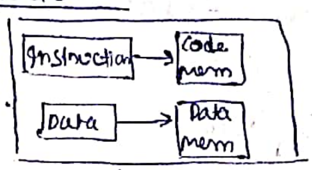


* CPU generates physical address for execution of program.

ex: 8085 MP, 8086 MP

ⓘⓘ Harvard Architecture

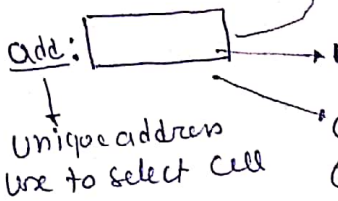
* Instructions are pramenentely present in Code Mem.



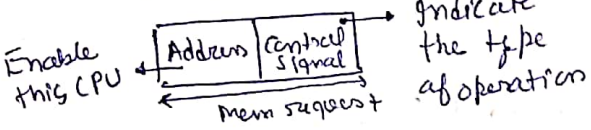
* Data is dynamic
* Execute only one type of program
ex: 8051 Microcontroller.

(5) Memory Organization

(i) Memory chip is collection of bit



(ii) Memory requests generate by CPU:



(iii) Memory Config:

M x N
- M → Total No. of Cell
- N → Size of each cell in bits

64 KB RAM → 64K, B

64K → $64 * 2^{10}$ — This is total no. of cell in memory

B → 8 bit — This is the capacity of each cell

$2^{10} \rightarrow 1K$ $\quad 2^{40} \rightarrow 1T$

$2^{20} \rightarrow 1M$ $\quad 2^{50} \rightarrow 1P$

$2^{30} \rightarrow 1G$

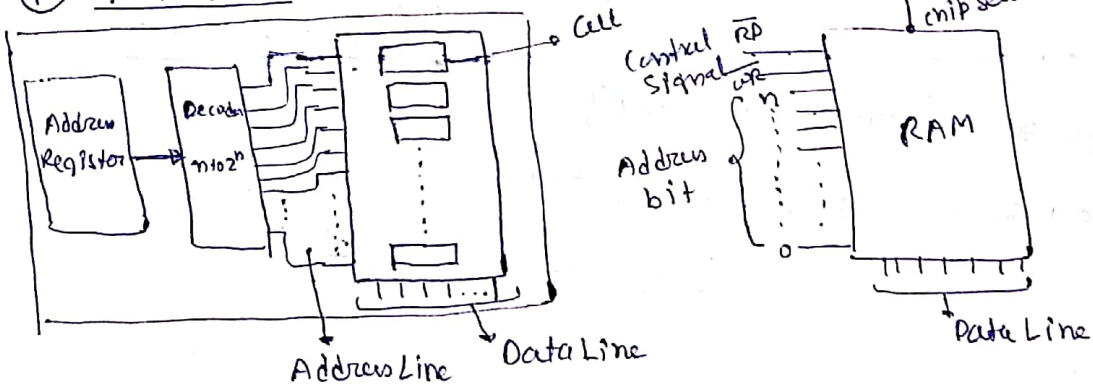
1 B → 8 bit

1 nibble → 4 bit

1 W → depends on CPU data line

$$\#\text{ of ram req} = \frac{\text{Given size RAM}}{\text{available size}}$$

(iv) Inside RAM



Data Space = $M * \left(\frac{N}{8}\right) B$

Data space in cell = N bits

Address line = $\log_2 M$ bits

(v) N bit-CPU: It means CPU process N bit data at a time. It is also word length of CPU. If CPU is 8 bit then word length is 8 bit. & If it is 16 bit, word length is 16 bit.

Word Addressable Memory: It means each cell of memory store 1 W.

Byte Addressable Memory: It means each cell of memory store 8 bit.

Nibble Addressable Memory: It means each cell of memory store 4 bit.

Bit Addressable Memory: It means each cell of mem store 1 bit.

* These all has nothing to do with Address line.

* $2^n \times 16$: It means:
  - Address line: n bit
  - Cell Size : 16 bit

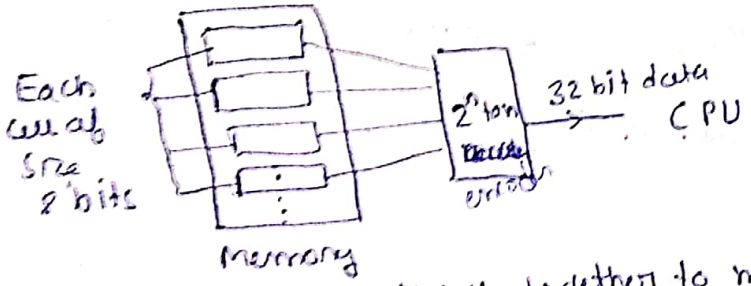
(a) In memory data always store in byte addressable Format. ex: Byte Addressable Memory means each memory cell store 8 bit.

(b) CPU always process data in word length Format.

ex: 32-bit CPU it means CPU process 32 bit data at a time.

(c) To make data in word length Parallel decoders are Used.

ex: Byte Addressable Memory & 32 bit CPU



Cell interface
↓
merge cell

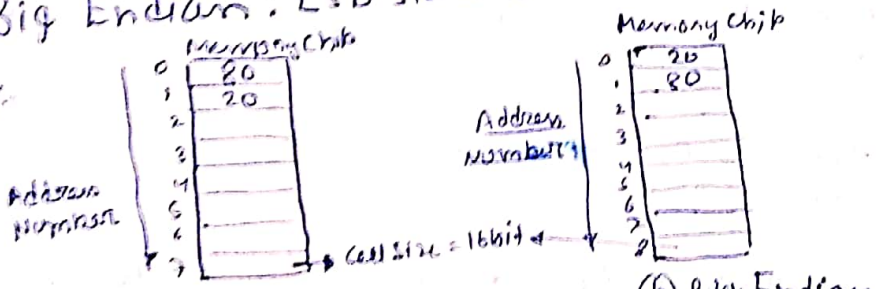
& use merge 4 cell together to make it 32 bit word length.

(iii) Interfacing of cell's to make it word bits, it cause problem in addressing to solve this problem two methods are used:

(a) Little Endian: LSB store on Lower address and MSB on higher

(b) Big Endian: LSB store on Higher address and MSB on lower

ex:



(a) Little Endian (b) Big Endian

Data: 20 80
MSB LSB

(6) CPU Architecture

(i) Processor contains set of hardware pins that are used to Operate with externally connected component.

(ii) 3 Groups of hardware pin

(a) Active High: when clock signals are high.

(b) Active Low: when clock signals are Low.

(c) Timplexed Pin: This pin is used to carry multiple data but not at same time

Address Latch Enable (ALE): It is hardware used for time multiplex pin. So that pin carry multiple data.

ALE = 0 = Pin carry data

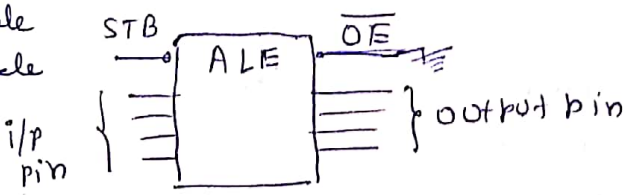
ALE = 1 = Pin carry address

(b) ALE contain 2 Pins: STB pin (Active High) (STROBE Pin)

$\overline{OE}$ Pin (Output enable.) (Active Low)

STB = 0 = i/p line disable

1 = i/p line enable

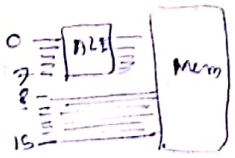


(iv) Machine Cycle: No. of CPU cycle required to complete one machine

(a) Instruction

(b) It depends on Designer.

(v) Memory interfacing 8085 ALE Mem



(vi) System Bus: It is used to provide communication b/w major component of system i.e Memory & I/O

(b) Types of line in SB

* Address Line:
- Unidirectional
- With help of it Mem capacity of system can determine

Ex: A_{15} ..... A_0 : 8085MP

A_{19} ..... A_0 : 8086MP

* Data Line
- Bidirectional
- With help of it word length of CPU can determine

Ex: A_7 ..... A_0 : 8085MP

A_{15} ..... A_0 : 8086MP

* Control Line
- Control signal: $\overline{RD}$ or $\overline{WR}$
- Timing signal: used for sync. b/w Mem/I/O. device.

(c) Common bus Cause Ambiguity. To solve it following Bus config are use:

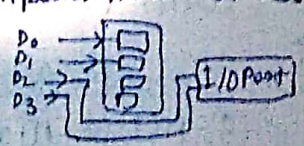
* I/O Processor (IOP)
- Seprate BUS
- Common Control Signal & Address Bus
- Different bus for Mem & I/O

* Isolated I/O [IO mapped IO]
- Extra Pin
- Common data bus & Address bus
- Different CS for Mem & IO.
- (I/O & $\overline{M}$) Pin is use.

Pin →	IO/$\overline{M}$	$\overline{RD}$	$\overline{WR}$	
	0	0	1	LOAD
	0	1	0	STORE
	1	0	1	IN
	1	1	0	OUT

* Memory mapped I/O
- Shared Mem
- Common CS & Address Bus.
- Shared Mem with I/O.
- Complete mem not use.

Ex:



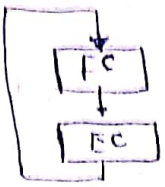
(9) Instruction Cycle

(i) Execution sequence of program.

(ii) Types:

(a) Fetch cycle (FC)

(b) Execution cycle (EC)



(iii) Fetch cycle & Execution cycle:

(a) Fetch instruction based on Program counter.
CPU — [from memory]

(b) PC is register, which hold address of instruction that is going to execute by CPU. It always increment by Step size.
Step size is fixed constant depends on CPU word length, because CPU always execute instruction in word length manner.

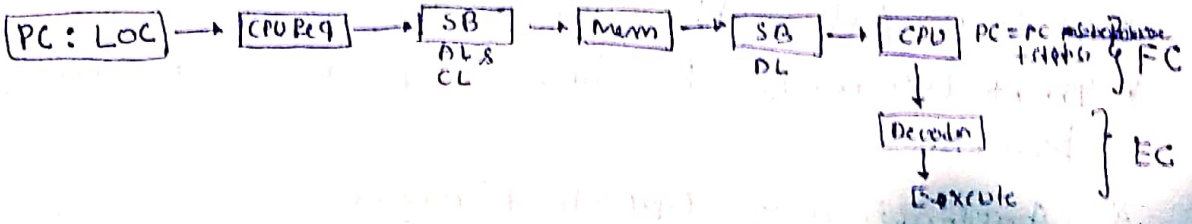
(c) -t : Starting address of program } To start execution starting Address sent to accumulator
(trace)

(d) [Opcode = word length] [opcode < word L] [opcode > word length (Not in use)]

(e) [1B Instruction] [2B Instruction] [3B Instruction]
[opcode = 8 bit] [opcode = 8 bit] [opcode = 8 bit]
[Intermediate = 8 bit] [Intermediate = 16 bit]

(f) Before starting of execution cycle, Program counter change to valid Program counter of next instruction. For example before starting execution of instruction [I_1], PC change to [I_2], then execution of I_1 begin.

(g)



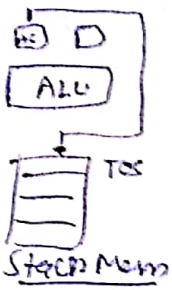
(10) Execution cycle

(a) CPU organization classified into 3 part based on ALU Data path:

* Stack CPU * Accumulator CPU * General Register CPU

b) Stack CPU

- All Operand present on Stack.
- It support 0-Addressing Mode only Opcode require.
- Opcode: PUSH, POP, ADD, MUL, DIV, SUB
- Stack memory require.
- Always fetch data from TOP of Stack.



S_1 S_2 D
TOS TOS TOS

c) Accumulator CPU

- In it One Operand is present in Accumulator and Other can be present in a register or memory.
- It support 1-Addressing mode, because other operand.
- Opcode: LOAD, STORE, ADD, MUL ...

S_1 S_2 D
ACC Reg/Mem ACC

S_1 = Source 1
S_2 = Source 2
D = Destination

- Register File: It store all register.
  Reg File = N., it means N registers are present.
- Memory spill: · Storing of data from Register to Memory.
  · Unwanted memory accen.
- Order of execution matter to optimize code.
  Ex: (A+B) / (C+D) → It require 1 Mem Spill.

d) Expand Opcode Technique

* It is useful in Fixed length Instruction set.
* In it we merge free bits to Opcode, so maximum opcode possible.
* Instruction Size = Opcode Size + Address Size

* Rules: (a) Identify higher Order Instruction, i.e smallest opcode in instruc (Promitive)
  (b) Identify total Instruction possible
  (c) Identify total free Instruction after promitive instruction.
  (d) Assign free Instructions to Lower order Instruction.

# of Instruction = No. of Free Opcode * $2^{\text{Address bit}}$

General Purpose Register CPU

(i) Two kind of architecture possible:
- * Register to memory reference
- * Register to Register reference

(ii) It is of 2-Address type instruction.

Inst: | Opcode | Addr | Addr |

(iii) Register to Memory reference

Inst: | Opcode | Reg | Mem/ |

S1 Reg, S2 Mem, D Reg

(iv) Register to Register reference

Inst: | Opcode | Reg | Reg |

S1 Reg, S2 Reg, D Reg

(B) Register to Register Reference CPU (More registers)

(i) It is 3-Address type Instruction, all are register operand.

Inst: | Opcode | reg1 | reg2 | reg3 |

reg1 → D, reg2 → S1, reg3 → S2

S1 reg2, S2 reg3, S3 reg#

(ii) 4-Address Instruction Format

| Opcode | Addr1 | Addr2 | Addr3 | Address to Next Inst |

Opcode → Operation, Addr1 → D, Addr2 → S1, Addr3 → S2

(8) Instruction Execution Sequence

IF	IE				
	ID	OF		PD	WB
		S1	S2		

{OF & PD} combinely to ALU

- IF → Mem
- ID → Decoder
- OF & PD → ALU
- WB → Register or Mem

(i) Instruction Fetch (IF)
(ii) Instruction Decode (ID)
(iii) Operand Fetch (OF)
(iv) Process Data (PD)
(v) Write Back (WB)

* CPU Fetch Instruction from memory, in Word Size Format.

| Decoder | — CS to SB

CS to Control BUS, CS to ALU

| WZ register pair |

(9) Program Status Word

(i) It is collection of Accumulator & Flags.

(ii) **Flag**: It is a Flip flop hold only one bit.

Types of Flags:

(a) Control Flag (CPU → User)

* **Carry Flag:**
  Extra bit out of MSB.
* **Parity Flag:**
  Even no. of one's in ALU O/P.
* **Auxillary Carry:** (Used in BCD)
  - Extra bit from lower nibble to higher nibble.
  - Carry bit is used to hold range exceeding condition of higher nibble
  - Unpack BCD & Pack BCD
    1 digit → (8 bit) 1 digit → (4 bit)
  - CY = 1 or AC = 1, then Digit > 9
    add 6 to make valid BCD.
* **Zero Flag** [CY Result]
  - ALU output is 0.
* **Signed Flag:**
  - MSB of ALU content 1.
* **Overflow Flag:**
  - $\bar{x}\bar{y}z + xy\bar{z}$

(b) Conditional Flag (User → CPU)

* **Trap Flag**
  0: Output after execution.
  1: Step by step output
  - g → 0 (go)
  - t → 1 (trace)
* **Interrupt Flag**
  0: Don't check any interrupt (Disable interrupt)
  1: Check interrupt continues (Enable interrupt)
* **Direction Flag**
  - 0: Auto increment (Clear direction)
  - 1: Auto decrement (Set direction)
    - → Starting Address
    - → Last Address.

(10) Addressing Mode

(i) It gives location of object.
(ii) It contains Effective address or address which lead to EA.
(iii) Effective Address is address of actual Object which is either data or Instruction.
(iv) Mode of Addressing (a) Implicit (b) Explicit

(a) Implicit: opcode itself indicate type of AM

(b) Explicit: | Opcode | AM Address |
AM bit use to indicate type of Addressing Mode.

(v)
- I/# → Immediate AM
- Regname: Register AM
- [] → Direct AM
- @/() → Indirect AM
- Index / Stack Name → Indexed AM
- +/- → Autoindexed AM

Types of Addressing Mode

(1) Sequential Control Flow AM

(i) It focus on data location. so data transfer and data manipulation both are design with this AM.

(a) Implied / Implicit AM : | Opcode | → Type of opⁿ, Data Info

ex: ADD ; (on stack CPU)

ex: STC, CLC {carry}

(b) Immediate AM : | Opcode | AM | (AM → Data)

ex: MOV r0 #23 (#23 → Constant)

- Range Limited : Unsigned & Signed Constant

EA : where program store #23

(c) Register AM : | Opcode | Reg Name | EA : Reg Name [ ] : value at

(d) Direct / Absolute AM : | Opcode | Addr | EA : Addr

(e) Indirect AM | Opcode | Addr | EA : Address present at [Addr]

|1MR, 2MR|

ex: MOV r0, @4000

(f) Register Indirect AM | Opcode | Reg Name | EA : Present in Reg Name

|1RR, 1MR, [illegible]|

(g) Indexed Based AM | Opcode | Reg Name | +/- Reg Na |

- Post Auto Inc
- Post Auto Dec
- Pre Auto Inc
- Pre Auto Dec

|1RR, 1ALU, 1MR|

Base Addr (r_i) → Base Addr + r_i → EA → [EA] ⇒ Data

ex: MOV r_1, #4 | MOV | r0 | 2000(r_1) |

2000(r_1) → 2000 + r_1

ex: MOV r_1, #1FFF

| MOV r0, +r_1 |

(h) Indexed Indirect AM

|1 RR, 1 ALU, 1MR, 1MR|

Addr (r_i) → Addr + r_i → [Pos where EA Present] → EA → [EA] → Data

(i) Autoindexed Addressing Mode

BASE Addr (+/-) stepsize

|1 RR, 1 ALU, 1MR|

| Base Addr |

| Opcode | Base Addr |

(ii) Transfer of Control AM

(i) Its purpose is location of Next instruction.

(ii) During execution program transfer from current location to target location.

(iii) It uses Transfer of Control instruction (TOC). to go to (Branch Address)

(iv) We divide Memory in segments for better utilization.

(v) If Transfer of Control occur within same segment then it is Intra Segment TOC.

If Transfer of Control occur between two segment then it is Inter Segment TOC.

(vi) To implement Intra Segment TOC (a) PC Relative Addressing Mode is Use.

PC Relative Addressing Mode: EA = PC + Relative Address

(current location − Target location)

* Relative Address present in Address field of instruction: Opcode | Addn → Relative value.

(vii) To implement the Inter Segment TOC (b) Base Register AM is used

EA = [BA] + Relative Address

[BA] → (content of Base register)

Relative Address → present in Address field of instruction.

(Best for independent code)

(ii) Instruction Set

(i) Types of Instruction

(a) Data Manipulation Instruction

* ADD, MUL, DIV, SUB, INC, DEC, CMP, AND, OR, XOR, LSL, ASL, LSR, ASR,
* Arithmetic Inst, Logical Inst, Shift Inst. → ROL, ROR, RCL, RCR

(b) Data Transfer Instruction

* MOV, LOAD
* STORE, PUSH, POP, IN, OUT

(c) Data Control Instruction

* Conditional TOC
* Unconditional TOC
* Based on previous Flag status this conditional TOC is evaluated. when condition true then control transfer to given location.

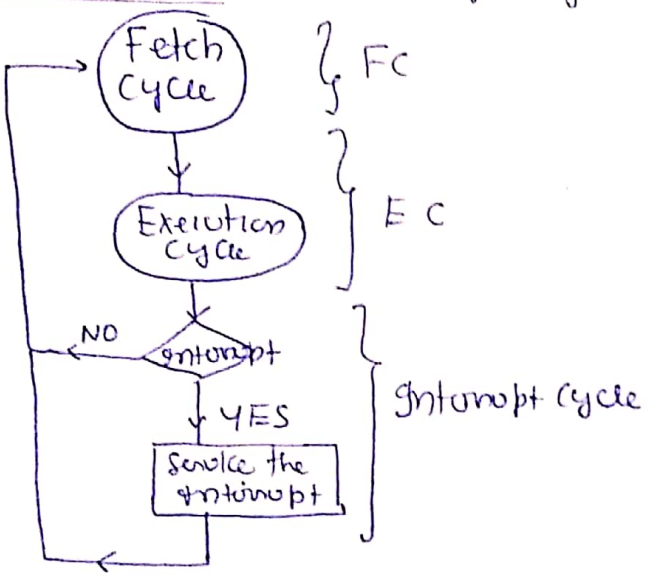
* CMP: Op2 − Op1
  - CY = 1 → Op2 is Smaller
  - Z = 0 → Op2 is not equal to Op1
  - Z = 1 → Op2 == Op1

SUB Program (Functions in Higher Language) (Procedures)

- Single Entry - Single Exit
- Main Program is suspended during execution of sub program
- CALL & RET
  - CALL → Invoke PUSH into stack before transfer of control
  - RET → Invoke POP, and return result to PC.

(12) Interrupt Cycle : It is a signal generated by hardware.

(b)



(c) CPU checks INTRPIN after each execution cycle.

(d) When interrupt occur PC value store in Stack and vector address is loaded into PC.

(e) Nested Interrupt.

(f) Types of Interrupt :

(i) Hardware Interrupt

- External Interrupt
- Internal Interrupt
- RST 4.5, 5.5, 6.5, 7.5
  EH L ML H
  INR
  EL
  H : High
  Extreme : E
- Maskable & Non Maskable Interrupt
- Vector & Non Vector
- Level trigger & Edge trigger Interrupt.

(ii) Software Interrupt

- System calls.

Time require to
+ Service : Response time + Service Time

(13) RISC and CISC :

RISC

- More register
- less addressing mode
- Fixed length instruction
- It support Seccessful Pipeline
- Cl per instruction (CPI) = 1
- It used in real time app.

RISC (Reduce Instruction Set Comp)

- It contain smaller instruction set.
- It support hardwire control unit.

CISC (Complex Inst. Set Comp)

- Supports less register
- It support more addressing mode
- It support variable length instruction.
- It support unsuccessful pipeline.
- CPI ≠ 1
- It used in personal app.
- It use larger instruction set.
- It support microprogram control unit.

(14) Register Organisation RISC Computer

(i) RISC register classified into 4 Groups.

(a) Global register (G)
(b) Local register (L)
(c) In register (C)
(d) Out register (C)

(ii) Window Size

It is form by grouping of Local, In and Out register

$$\text{window size} = L + 2C + G$$

(iii) Global register is shared by all.

(iv) Window organize in overlap order, one window out is used as in for other.

(v) Register File Size : $W(L+C)+G$ W : # of windows.

Module 2

(15) Computer Components

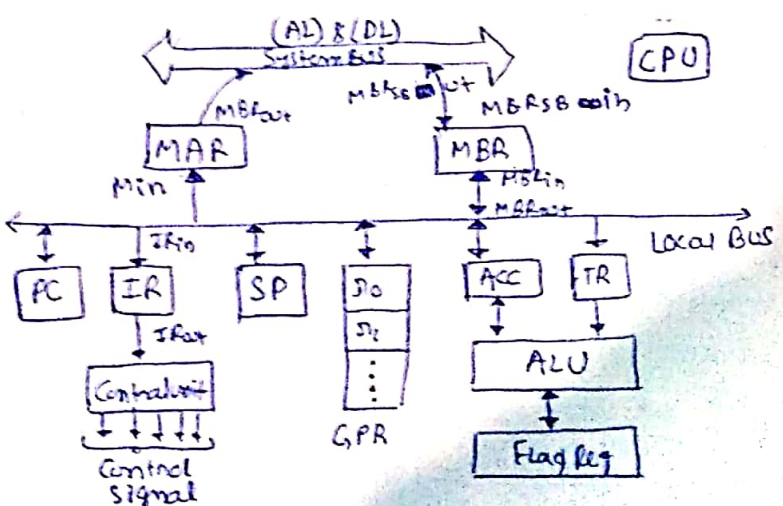
(i) CPU (ii) Memory (iii) I/O

(i) CPU : It is processing unit.

It contains 3 things :

(a) Register
(b) ALU
(c) Control Unit

- Program Counter (PC)
- Instruction Register (IR)
- Accumulator (ACC)
- Temporary Reg. (TR)
- Memory Address Reg (MAR)
- Memory Buffer Reg (MBR)
- Stack Pointer (SP)
- Flag Reg
- General Purpose Reg. (GRP)



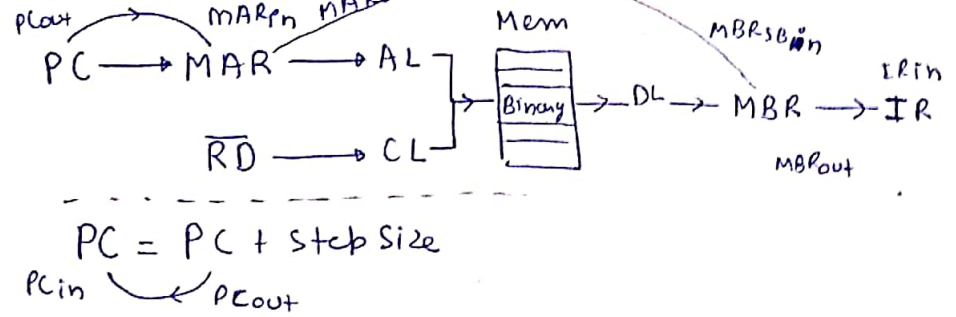
Micro-Operations : (a) It is fundamental operation in Bare hardware.
(b) There operation which are directly executed on bare hardware without further decomposition. is micro-operation.
(c) Operations ran on bare hardware in 1 cycle is μoperation.
(d) For User μ Operation is register to register transfer.
(e) Micro Instruction is collection of one-or more micro-operation, still each Micro Instruction take 1 cycle only because all Micro-operation are running in parallel.
(f) Control signal require to execute μ-operation.

M/C Instn → Register Transfer Language → H/w design → μoperation List → μprogram → Controlsignal

*** Microprogram :**
(a) It is sequence of μoperations which is used to perform meaningful action. in hardware.
(b) In Instruction cycle Common μprogram is used to execute Instruction:

- Instruction Fetch (IF)
- Instruction Decode (ID)
- Operand fetch (OF)
- write Back (WB)
- Interrupt SUB Program Initialization

*** Instruction Fetch :**



PC = PC + step size
PCin PCout

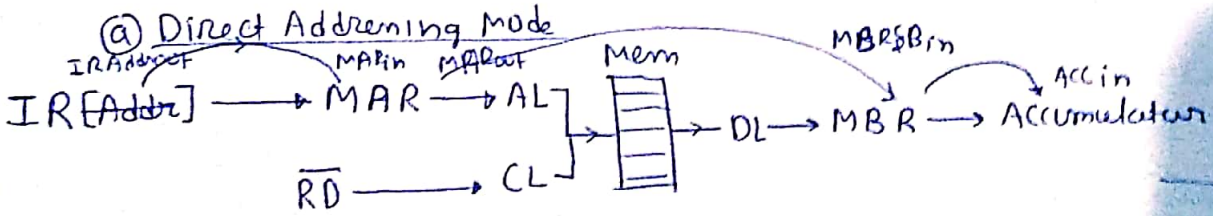
T1 : PC → MAR ;
T2 : M[MAR] → MBR ;
PC + StepSize → PC
T3 : MBR → IR

*** Instruction Decode**

IRout RecodeIn
IR → Decoder T1 : IR → Decoder

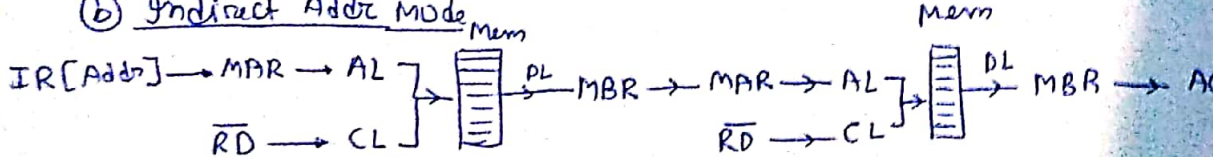
*** Operand Fetch**

(a) Direct Addressing Mode



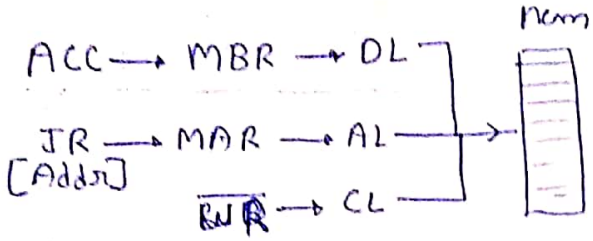
T1 : IR[Addr] → MAR
T2 : M[MAR] → MBR
T3 : MBR → ACC

(b) Indirect Addr Mode



T1 : IR[Addr] → MAR
T2 : M[MAR] → MBR
T3 : MBR → MAR
T4 : M[MAR] → MBR
T5 : MBR → ACC

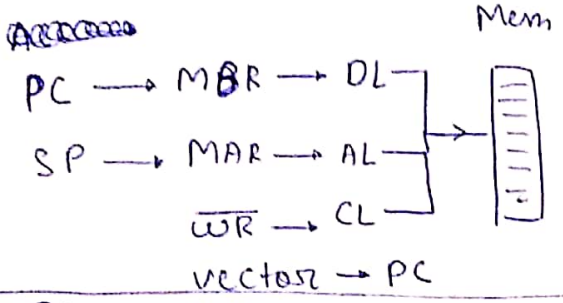
(d) Write Back | STA | 200



T1 : ACC → MBR
T2 : IR[Addr] → MAR
T3 : MBR → M[MAR]

This two statements can be swap without any effect.

(e) Interrupt SUB Program

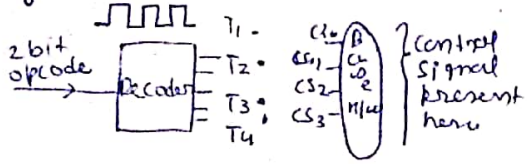


T1 : PC → MBR
T2 : SP → MAR
T3 : MBR → M[MAR] } System Bus
Vector Addr → PC } Local Bus

(iii) Control Unit

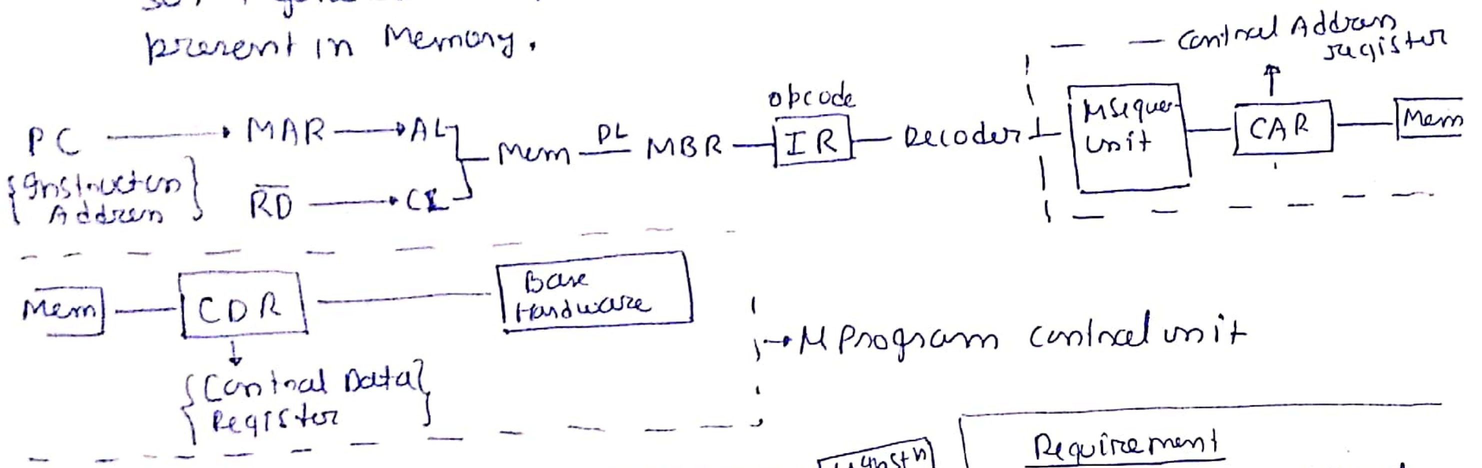
Require :
(a) How many Control Signal present in Base H/w.
(b) How many Instruction are implemented in Base H/w.
(c) How many MOperation are require for each instruction.
(d) What are control signal require for each MOperation.

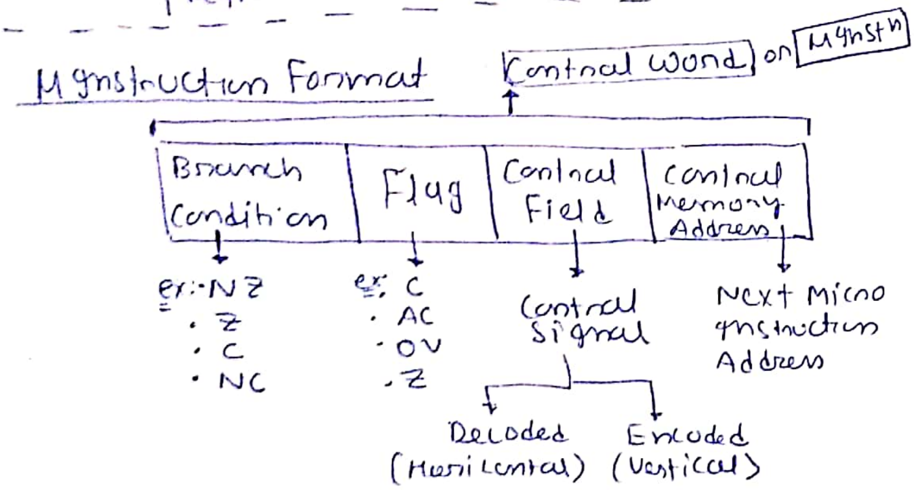
(A) Hardwired Control Unit : Used in RISC control unit.
- It is the fastest CU. * It used in Real time application.
- It expressed in term of SOP. For each Control Signal we find SOP.
- Control Signal perform operation in hardware on the basis of MOperation
  If we have n control signal then we can make 2^n Instruction we can also take its subset but at max 2^n Operation we can make.
  ex: If there is 32 Instruction and only 2 control signals then it is not possible because with 2 CS we can make at max 4 Instruction.
  Instruction → MOperation → CS
  (ex. +, -, /, *)
- SOP Form : ControlSignal 1 → MOperation$_1$ (Instruction which are using CS1) + MOperation$_2$ (Signal which are using CS2) + ...
  If CS used by all Instruction, we just write MOperation no.
  It by default means all Instruction are using it.



Microprogrammed Control Unit

* In this design microprograms are present in Control Memory.
* In it a μSequencer unit is present, in which decoded Instruction is input and on the basis of instruction it have information about what are μOperation require to complete the Instruction. So, it generate μOperation as μInstruction address which is present in Memory.





Requirement

* How many Control signal present in Base H/w?
* How many Instruction is present?
* How many μoperation or Instruction require for each Instruction?
* What Control signal req. for each μInstruction to execute.

* Types of μProgrammed Instruction
  → Horizontal μProgram Instruction
  → Vertical μProgram Instruction

(By default we consider vertical to design μInstruction)

Horizontal

* N-bits are require to represent N Control signal

Instruction No:

μOperation1 : | – | – | $CS_1\ CS_2\ CS_3$ | – | (0 1 1)

μOperation2 : | – | – | $CS_1\ CS_2\ CS_3$ | – | (0 0 1)

* High degree of parallelism.
* Faster due to no decoder
* It is not uses.
* By default μInstruction Design with 1 Control Signal.
  ↳ It cause effect in vertical if n signal active at a time bits = $n \times \log_2 N$ bits

Vertical

* $\log_2 N$ bits are require to represent for each N Control Signal.

Instruction No

μOperation : | – | – | 00 | 01 | 10 | – | ($CS_0\ CS_1\ CS_2$)

10 → Signal (4 bit)
at a time 3 active
| S_1 | S_2 | S_3 |
4bit 4bit 4bit
– 12 bit
→ 1 μInst

* Low degree of parallelism.
* Slower due to decoder.
* It uses in CISC computer.
* Function Code : μInstruction
  ($\log_2 N$)
* By default μInstruction Design with 1 Control Signal.

i.e what by default design with 1 signal

(16) Performance Analysis

(i) Performance $\propto$ $1/\text{Execution}$

(ii) Execution time = No. of second / Prog

No. of second = No. of instruction/Prog * No. of cycle require for each instruction / instruction * No. of second / cycle

ET = Instruction Count * Cycle Per Instruction * Cycle Time

ET = IC * CPI * Cycle Time

$$ET = \left(\sum IC_i * CPI_i\right) * \text{Cycle Time}$$

(iii) **Speed UP** : It is a factor which is use to measure performance gain.

$$S = \frac{\text{Performance of } X}{\text{Performance of } Y} = \frac{1/ET_x}{1/ET_y} = \frac{ET_y}{ET_x}$$

---

(17) Amdhal's Law

* According to Amdhal's, rather than developing new system, improve the performance of exsisting system.
* Due to enhancement of current system, it divided into two part. Unenhanced part and enhanced part.
* Overall system performance : $S_{overall} = \frac{ET_{old}}{ET_{new}}$ (Performance of new) / (Performance of old)
* ET_{new} = ET of unenhanced Part + ET of enhanced Part
* Enhance those part which used most.
* ET_{new} depends on (Fraction enhance) i.e how much part of system enhance and (Speed Up enhance) i.e how much performance gain by enhancement. F : enhance Fraction

(1-F) : unenhance Fraction

$$S = \frac{ET \text{ of } old \text{ } F}{ET \text{ of } new \text{ } F} = \left[ET \text{ of new } F = \frac{ET \text{ of old } F}{\text{Speed Up}}\right]$$

$$S_{overall} = \frac{ET_{old}}{ET \text{ of unenhance} + \frac{ET \text{ of old } F}{S}} = \frac{100\%}{(100\% - F) + \frac{F}{S}} = \frac{1}{(1-F) + \frac{F}{S}} = \left((1-F) + \frac{F}{S}\right)^{-1}$$

Multible enhance $S_{overall} = \left[(1 - \sum F_i) + \sum \frac{F_i}{S_i}\right]^{-1}$ ← Non enhanced fraction

High Performance CPU Design

(a) Concurrency is present, means 2 or more process can run at same time.

(b) Four type of Architecture according to Flynn's:

(i) Serial Instruction Stream and Serial Data Stream (SISD)

(ii) Serial Instruction Stream and Multiple Data Stream (SIMD)

(iii) Multiple Instruction Stream and Single Data Stream (MISD)

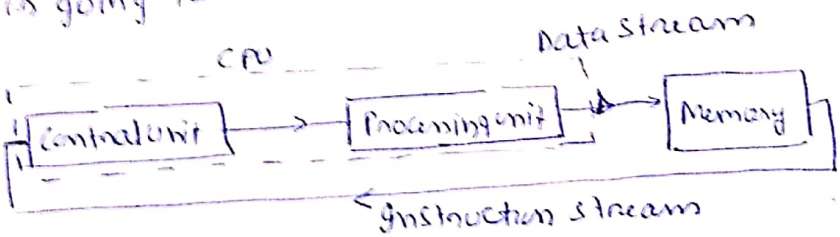
(iv) Multiple Instruction Stream and Multiple Data Stream (MIMD)

(c) MISD is never used and not implement.

(d) SIMD & MIMD need additional hardware, so it is by default support Concurrency.

(e) SISD is having no extra hardware support, so with help of pipeline it is going to have concurrency.

SISD



---

(14) Pipelining

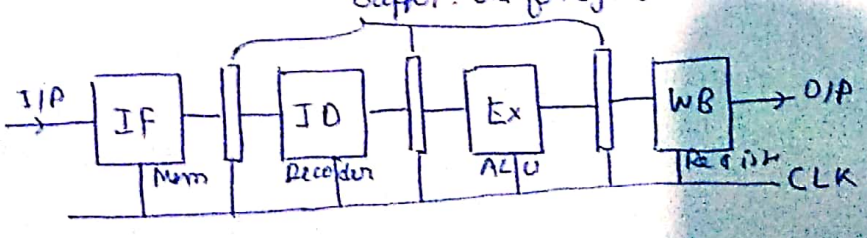
- It is hypothetical pipeline concept.
- It is successful pipeline
- It works for Fixed length instruction

(i) Basic Pipeline:

(a) It is consider to have 4 Stages. i.e IF, ID, Ex and WB.
IF works on Memory.
ID works on decoder
Ex works on ALU
WB works on Memory or Register.

(b) In Non-Pipeline each instruction execute serially. one after another, so in each cycle only one operation of one instruction is present.

(c) With Pipeline:



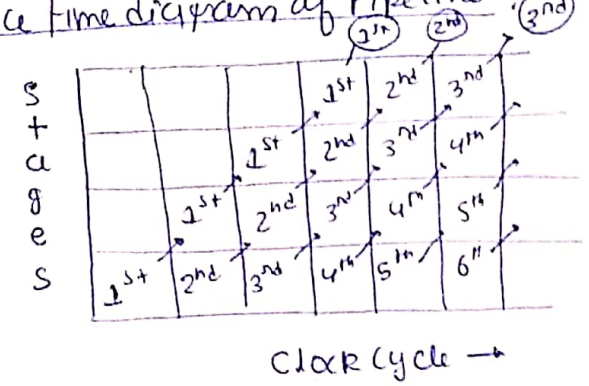
(d) Function of Pipeline

(i) It decompose problem into Independent Subproblem.

(ii) It then run all those Independent Subproblem on Independent Hardware.

(iii) It provide Overlaping Execution, means it insert new input before the completion of old input.
This way new input always inserted in each clock cycle, so CPI for Pipeline is 1.

Space time diagram of Pipeline:



(e) Delays in pipeline:

(i) If all stages have Uniform delay
Cycle time (t_p) = Stage delay

(ii) If all stages have non uniform delay
Cycle time (t_p) = Max (Stage Delay)

(iii) If buffer delay present in pipeline
Cycle time (t_p) = Max (Stage Delay + Buffer delay)

(iv) If setup time or skew time present in pipeline
Cycle time (t_p) = t_p + overhead time

(f) Performance of Pipeline

$ET_{Pipe} = (K+(n-1)) \times \text{cycles} = (K+(n-1)) \cdot t_p$

$ET_{Nonpipe} = n \cdot t_n$ So, S (Performance gain) = Performance of pipe / Performance of nonpipe

$S = \frac{ET_{Nonpipe}}{ET_{Pipe}} \Rightarrow S = \frac{n \cdot t_n}{(K+(n-1))\, t_p}$ when, n is finite and CPI ≠ 1,,

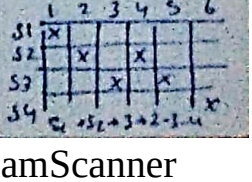
$S = \frac{t_n}{t_p}$ when $n \to \infty$ CPI = 1,,

Perfectly balanced pipeline: $t_n = K\text{-cycle} \Rightarrow t_n = K \cdot t_p$ so $S = K$

Efficiency $(\eta) = S/K$ and Throughput $= n/(K+(n-1))\, t_p$

(g) Types of Pipeline:

- **Linear Pipeline:**
  - It is synchronize.
  - In it latency is 1 (difference b/w two successive initiation in
  - It contain pipeline in only forward computation pipeline)
- **Non linear pipeline:**
  - It is asynchronous.
  - In it latency depends on reservation table.
  - Reservation table helps in execution order.



RISC Pipelining

a) To analyze what are the problem associated with pipeline, we consider a successfull pipeline implementation i.e RISC pipeline.

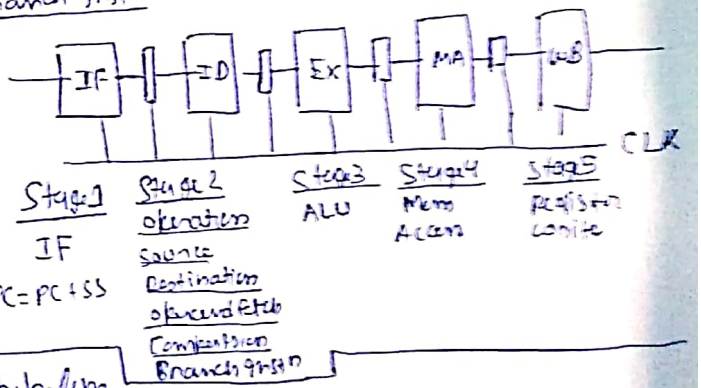
b) Instruction set of RISC

(i) Data Transfer Instruction
* It use Load and Store
* It use Indexed addressing mode to access memory.
  ex: Load R0, 3(R1)
      Store 3(R1), R0

(ii) Data Manipulation Instruction
* In it ALU operation perform only on register data.
* ex: Add R0, R1, R2

(iii) Transfer of control Instruction
* Unconditional TOC
* Condition TOC: RISC has its own instruction for it. ex: DJNZ (Decrement, Jump NO Zero)
  Ex: DJNZ R0, 2000

c) RISC uses 5 stage Pipeline:
(i) Instruction Fetch (Memory) {Along with Fetch PC increment for next Instruction Due to Fixed length Instruction}
(ii) Instruction Decode (Decoder & Operand Fetch (comparison) & Branch instn)
(iii) Execute Instruction (ALU)
(iv) Memory Refernce (Mem)
(v) Register Access (Write Back)



d) * Dependency is major problem in pipeline.
* Dependency cause extra cycle and in these extra cycle no new instruction is loaded this is also known as Stall cycle.
* When Stall cycle present then CPI ≠ 1.

e) Types of Dependencies:

(i) Structural Dependency
* It occurs due to resource conflict in pipeline.
* Resource Conflict is when two or more instruction try to use same memory or function or any Functional unit.

* In RISC memory resource conflict is present because in fetch we access memory as well as in stage 4 we also access memory.
* To solve memory conflict, Memory divided into 2 parts Code Memory and Data Memory. So in fetch time code memory is access and stage 4 data memory is access.
* If we not devide memory in 2 parts, one instruction have to wait before accessing memory, it should be free. that wait cause stalls.
* Dividing memory is also called as Renaming mechanism.
* When pipeline full always 4 instruction overlap.

Problem $\xrightarrow{\text{solve}}$ STALLS $\xrightarrow{\text{optimize}}$ Final solution
(Structural Hazard)

(ii) Data Dependency:
* It occurs when a instruction I later to read dependent data before instruction i writes it. It cause unsafe condition.

ex: $I_1: R_1 = R_1 + R_2$; I
$I_2: R_3 = R_1 + R_2$; J

Serial (non pipeline): No problem
Pipeline: Data dependency problem (J read data at same time when I executing)

* To solve this problem a algorithm is used known as TOMASULO Algo.

Previous dependency is also known as true data dependency.
i.e. $I_1 - I_2$ & $I_2 - I_3$

Tomasulo algorithm

Sno	Operation	(Independent) Operand 1	(Dependent) Operand 2	Dependent op1	Dependent op2	Status 0 = incomplete 1 = complete	Allocation	Just
1	ADD	R_0	R_1	-	-	0	R_0 = value R_1 = value allocated	

* Problem with help of this algorithm data dependency problem solved, but it cause (Raw Hazard or Data Hazard).
* To solve (Raw or Data Hazard) hardware technique is use i.e Operand Forwarding by front circuiting for this buffer is use.

(iii) Control Dependency: It occurs when unwanted instruction is execute in pipeline.
- To solve this problem Flush operation is used, it insert nop (no operation) instruction in place of unwanted instruction. (Control Hazard)
- Use of unwanted instruction depends on, on which stage branch statement is handle. For RISC Branch handle on second stage, so unwanted 1 instruction is always 1. It is also known as Penalties (no of unwanted inst^n)
- No. of Stalls or NOP = (Branch Frequency) * (Branch Penalties)
- Branch prediction (Hw technique): If this technique is used then branch target available at First Stage.
- Software Technique (Compiler): (a) Code rearrangement (Safe sequence in our target not order)
  (b) NOP Substitution

overwrite in same cycle

Instruction Scheduling

(i) **In order execution**: If any instruction is dependent then rest of instruction also share stalls.

(ii) If Operand forwarding not support, then run independent instruction first, but it cause problem known as Out of Order execution.

True data Dependency (RBW) (read before write) (adjacent instn)

(a) ANTI Data Dependency (WBR) {write before read}

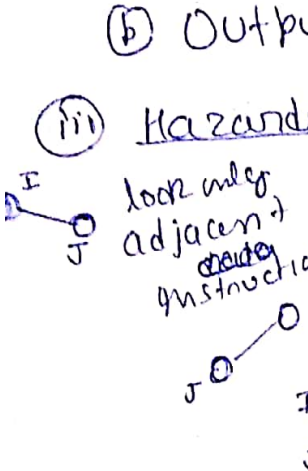
(b) Output Data Dependency. (WBW) {write before write}

(iii) **Hazards**:

(a) Read Before Write → Problem
Read After Write → Solution → Hazard } True data LOSS

(b) Write Before Read → Problem
Write After Read → Soln → Hazard } ANTI Data LOSS

(c) Write Before Write → Problem
Write After Write → Soln → Hazard } Output Data LOSS



(22) Performance Analysis:

$$S = \frac{CPI_{Nonpipe} * CycleTime}{(1 + \#stalls/instn) * Cycle\ time}$$

$$ET_{avg} = (1 + \#Stall/In) * Cycle\ Time$$

When Perfectly balance: $(T_n = K \cdot t_p)$

$$S = \frac{K}{1 + \#stall/instn}$$

when, $\eta = 100\%$

$$S = \frac{K}{1+0} = \boxed{S = K}.$$

$$\#\text{ line in CM} = \frac{CM\ Size}{Block\ Size}$$

$$\#\text{ line in MM} = \frac{MM\ Size}{Block\ Size}$$

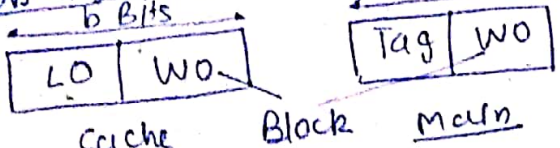
(23) Memory Organization (Module:3)

* Sequential Memory Organization
* Hierarchical Memory organization

(i) **Hit ratio**: (# Hits / Total # of accen)

(ii) **Cache Memory**

(a) Address Format



Tag space: Extra memory to store tag information.

Tag Directory Size = # lines in CM * tag space in line

Data Mem Size = # lines in CM * Block size

CM size = Tag Directory Size + Data Mem Size

(b) Status of cache = column content of cache
Status of Line = row content of cache.

(c) Mapping

Main Mem : | tag (3bit) | WO (1bit) |

Cache Mem : | Tag space (1bit) | LO (2bit) | WO (1bit) |

* Initial Tag space is empty.
* Tag space is use to identify wether block is present or not.

(d) Direct Cache Mapping — (K Mod N = i)
* Compulsory Miss → First time accen
* Conflict Miss → suppose to present but not found
* Direct Mapping, there is chance of Thrashing, due to more no. of conflict mines

MM: | b | a | (B bits)

CM: | b-c | c | a |

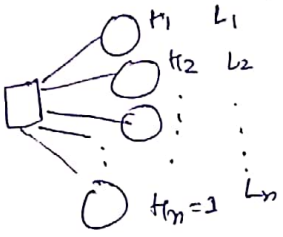
Tag Space: b-c bit

$C = \log_2\left(\frac{\text{CM size}}{\text{Block Size}}\right)$

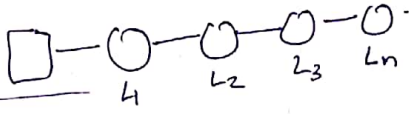
a = word offset
b = Tag bit

* Sequential Mem accen:

$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 + \cdots (1-H_1)(1-H_2)\cdots H_n \cdot T_n$



* Hercarichal Memary accen:



$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_2+T_1) + (1-H_1)(1-H_2) H_3 (T_3+T_2+T_1) + \cdots + (1-H_1)(1-H_2)\cdots H_n (T_n+T_{n-1}+T_{n-2}+\cdots+T_1)$

(e) Associative Cache
* No fixed binding like Direct map.
* It is called as Content addressable memory
* It can be map to any cache block in sequence starting from top.

Address binging

MM: | Tag (3bit) | WO (1bit) |

Cache Mem: | Tag space (3bit) | WO (1bit) |

Directory Size = # lines in CM * Tag space

* In this cache, Replacement algorithm are used because of no fixed binding. Replacement algo (LRU, FIFO)

B bits → Memory location

MM: | b | a |

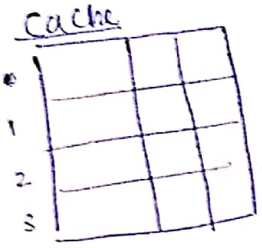
CM: | b | a |

} Main operation perform on it → illusion

Set associative Cache

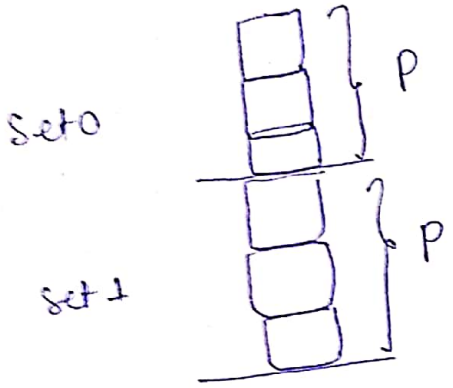
* In this cache lines are grouped into set and accomodate multiple blocks in set.

$$\#\ Sets(S) = \frac{N}{P\text{-way}}$$; N ⇒ No. of lines
P ⇒ No. of line in one set.



(S × P)
(4 × 3)
or

where S = No. of rows
P = {No. of columns / How many blocks can accomodate in one set or row.}



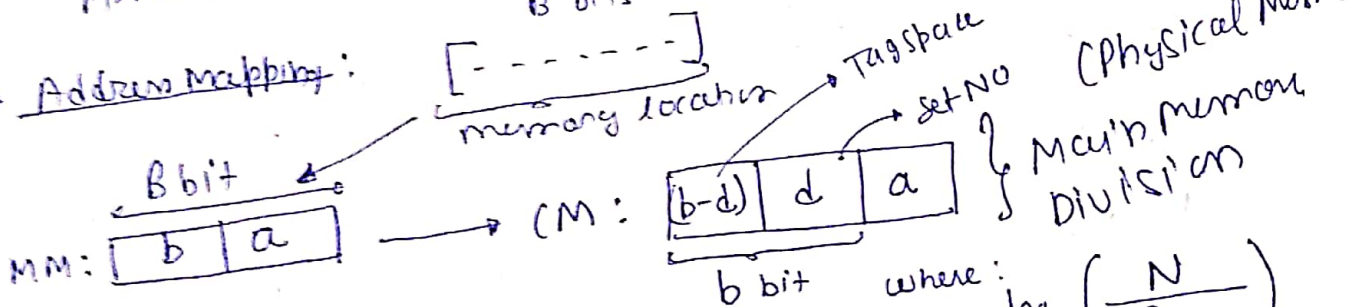
* Mod function is use to map the Block with set. after mapping with set block can accomodate any line, in sequence.

$$K\ Mod\ S = i$$

* Replacement algorith used inside set.
* This cache design is combination of Direct and Associative mapping.

a = word offset
b = tag bit

* Address Mapping:



where: $d = \log_2\left(\frac{N}{P\text{ way}}\right)$ ← No. of sets

Tag Directory Size = # lines * Tag space
= N * (b-d) bits
or
= (S×P) * (b-d) bits

$$N = S \times P$$

* $1 < P < N$

P way: P = 1 → work like Direct Mapping
P = N → work like associative mapping.

(ii) Updating Technique

* When CPU tries to read from memory location or write on memory location and if that block present in cache. it is known as Read Hit and Write Hit.

* When Block not present in cache, it is known as Read Miss or Write Miss.

And when Memory copy from Main to cache for read purpose is known as Read allocate and for write purpose known as Write allocate.

* Updating technique require because of inconsistency in data present in cache as well as in main memory.

* Two policies are used for updation technique. i.e
  * Write-Through protocal
  * Write-back protocal

* Write-Through Protocal
  → CPU perform simultaneous write operation on both cache and MM.
  → Inclusion always success.
  → Simultaneous Write time = Max(word updation time in cache , word updation time in MM)
  (Tw)

Read:

$$T_{avg\,read} = H_R T_c + (1 - H_R)(T_M + T_c)$$

H_R : if present in cache.
T_c : read time
T_M : Read allocate time of time require to transfer data from MM to CM

Write:

$$T_{avg\,write} = H_W T_w + (1 - H_W)(T_M + T_w)$$

H_W : if present in cache
T_M : main memory

Average mem access time:

$$T_{avg} = (Freq_{read} * T_{avg\,read}) + (Freq_{write} * T_{avg\,write})$$

→ when Simultaneous Memory organization is there then Hit ratio for write is always 1 [$H_w = 1$]

So,

$$T_{avg\,write} = \underbrace{T_n}_{MM\ access\ time}$$

$$T_{avg\,read} = H_1 T_1 + (1-H_1)\cdot H_2 \cdot T_2 + \cdots$$

→ If Question Contain : CM with Write-Through policie

if $H_w = 1$, then use Simultaneous Mem. Organ'

$H_w \neq 1$, then use Hierarchical Mem Organ.

* Write Back Protocal

→ In this method CPU maintain one extra bit i.e Update bit (or dirty bit)

If it is set, this means modification done on Cache Block and if it is 0, then no modification done.

→ So, when CPU going to replace a block, it first check its Update bit, if it is 1, then CPU first write it back to main memory then replace it.

Update bit → 1 (Dirty bit) } % of chance that it is dirty bit } % D

→ 0 (Clean bit) } % of chance that it is clean bit } % C

Read Time:

$$T_{avg\,read} = H_R T_C + (1-H_R)\{\%D * (\underbrace{T_M}_{\text{write back time}} + \underbrace{T_M}_{\text{Read allocate time}} + T_C) + \%C * (T_M + T_C)\}$$

(Main Mem access time)

Write Time

$$T_{avg\,write} = H_w T_c + (1-H_w)\{\%D * (T_M + T_M + T_C) + \%C * (T_M + T_C)\}$$

Average time

$$T_{avg\ WB} = (Freq_{read} \times T_{avg\,read}) + (Freq_{write} * T_{avg\,write})$$

✡ If Hit ratio of write is not given then we consider it as 1.

(iv) Multilevel Cache

(a) **Miss Penalty** : Time required to transfer data from higher level to lower level. When there is miss in lower level.

(b) L_1 CM size < L_2 CM size < L_3 ....

L_1 CM access time < L_2 CM access time < ....

(c) **Two kind of Miss rate**

* Global Miss Rate = (# Misses in Cache / Total CPU reference)
* Local Miss Rate = (# Misses in Cache / Total # access to Cache.)

Tavg = Hit ratio * Hit Time L_1 + (Miss Rate L_1 * Miss Penalty of L_1)

Miss Penalty L_1 = Hit ratio * Hit Time L_2 + (Miss Rate L_2 * Miss Penalty L_2)

Miss Penalty L_2 = Main Memory access time.

Average mem stall per instruction =
= (# of misses in L1/inst * Hit time L_2) + (# of Misses in L2/inst * Penalty of L_2)

(v) Types of Misses

* Compulsory miss
* Capacity miss
* Conflict miss.

(vi) **Cache Inclusion** : If there are multilevel cache, if content of higher level cache also present in Lower level cache then cache is said to be inclusive. whatever changes occur in lower level caches must reflect in higher level caches so for this both cache must use write-through policy

I/O Organization (Module 4)

(i) Interrupt driven I/O.

(ii) I/O Modes

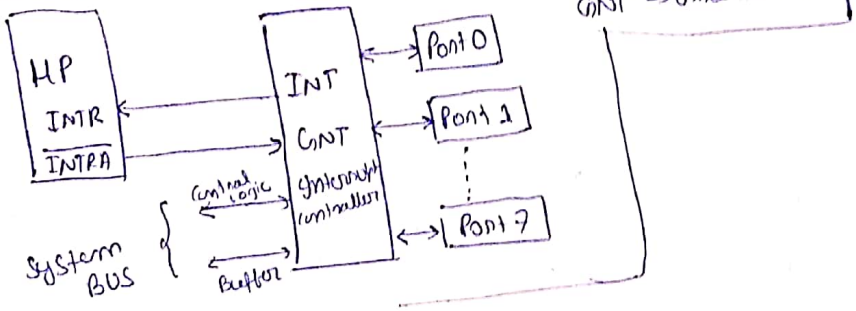
(a) Programmed I/O

ex: Banking System

* I/O operation are programmed in CPU.
* CPU waiting time. * Depends on speed of I/O device.

(b) Interrupt Driven I/O

* Depends on latency of interface.



(c) Direct Memory Access (DMA)

* Bulk amount of data transfer from I/O to MM, without involvement of CPU.

Access Sequence:

* CPU Initalize DMA Module along with I/O command.
* DMA Control logic interprets the command and enable I/O operation.
* Based on speed of I/O device, it consume the time to prepare data, after that it enable DMAREQ signal to DMA. DMA after recieving DMAREQ, it enable HOLD signal to CPU, then CPU generate HOLDA to DMA and DMA after recieving HOLDA signal, it get control of System BUS and after that DMA generate DMAREQ,A to device, after which data transfer from I/O to MM, & continue till COUNT REG value become 0. After COUNT REG become 0, DMA cycle completed and link re establish b/w CPU & System BUS.

* In DMA CPU present in 2 states:

→ Busy State : In this state I/O device prepare data.

Let, X "Preparation time" & Y "Transfer time".

$$\% \text{ of time CPU Busy} = \left(\frac{X}{X+Y}\right) * 100\%$$

→ Blocked State : In this state Data transfer take place CPU not have accen to system Bus.

$$\% \text{ of time CPU Block} : \left(\frac{Y}{X+Y}\right) * 100\%$$

* Modes of DMA

→ Burst Mode (waiting time more)

Taken SB → BULK Transfer → Return SB

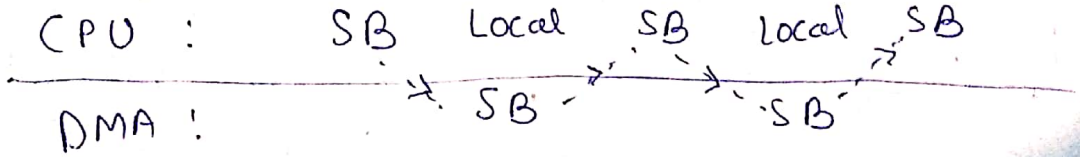
→ Cycle Stealing Mode (waiting time less than BM)

Taken SB → Small Data unit transfer → Return SB

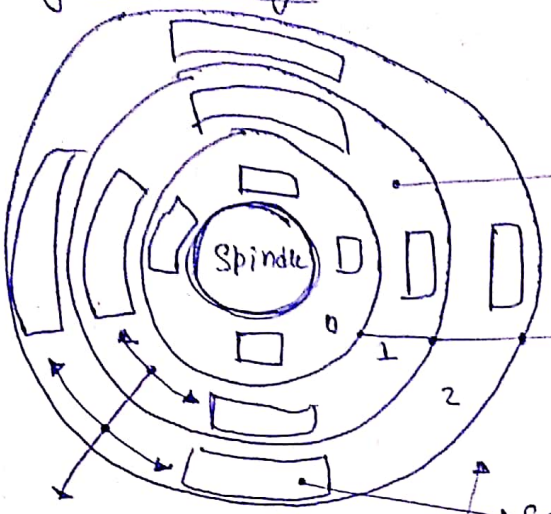
→ Interleaving Mode (efficient utilization)

Use SB when CPU Busy with local operation.

Instruction cycle : IF (Mem) ID OF (Mem) PD WB (Mem)



Secondary Storage

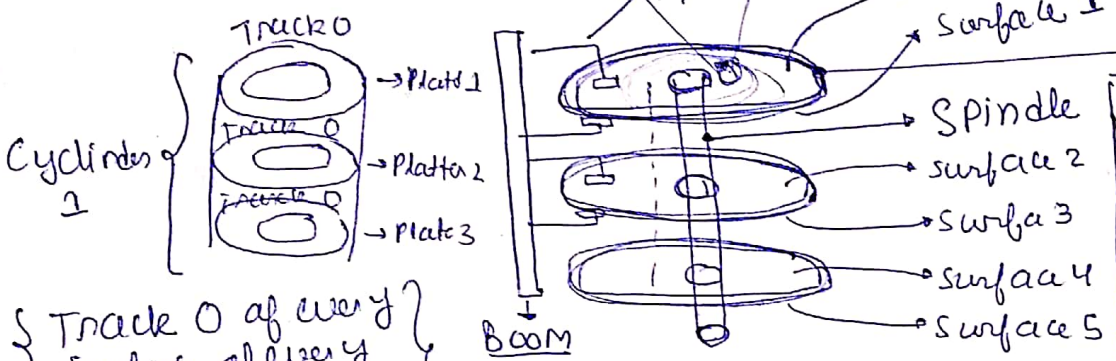


TRACKS (White Space)

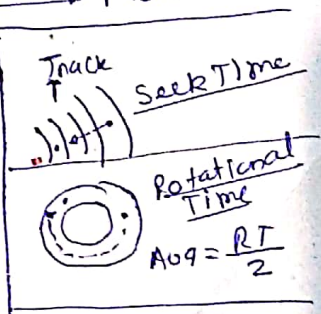
Inter track (Blue line) Distance

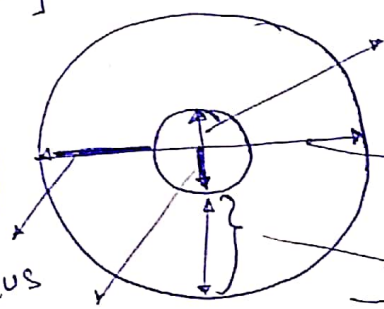
Sector

Inter Sector Distance



Track 0 of every surface of every platter combinely form a cyclinder





Storage area = [Outer Radius − Inner Radius]

tracks on surface = (Storage − inner radius) / Inter track Distance

of cylinder on disk = # tracks

Track capacity = # Sector/track * Sector capacity

Cyclinder capacity = # Surface in Disk * Track capacity

Disk capacity = # Cyclinder in Disk * Cyclinder capacity

or

Disk capacity = # Surface * # track/surface * # Sector/track * Sector capacity

Surface = 2 * # of Platter

Track width mention

track = # track − track width

Total time req to access data from Disk = Avg seek time + Average rotational lat. + Transfer time + Overhead

Sector # = (i, j, k) → Sector No.

i → Cyclinder No.

j → Surface No.

(26) Multiplication algorithm

(i) Unsigned NO : $A_3 \times B_3$

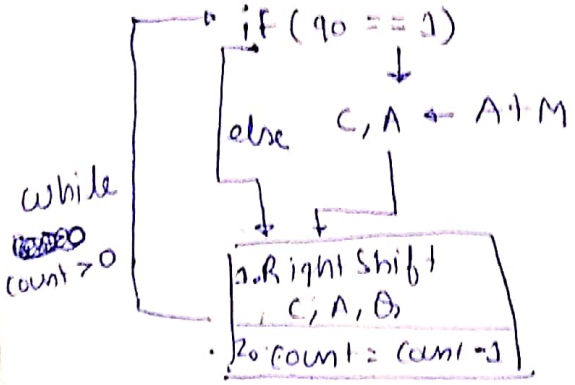
M ← Multiplicand

Q ← Multiplier

C ← 0 {1 Bit}

A ← 0 {Size of Multiplicand (M)}

count ← No of bits in multiplier (Q)



Result : AQ

(ii) Signed NO (Booth's Multiplication Algo)

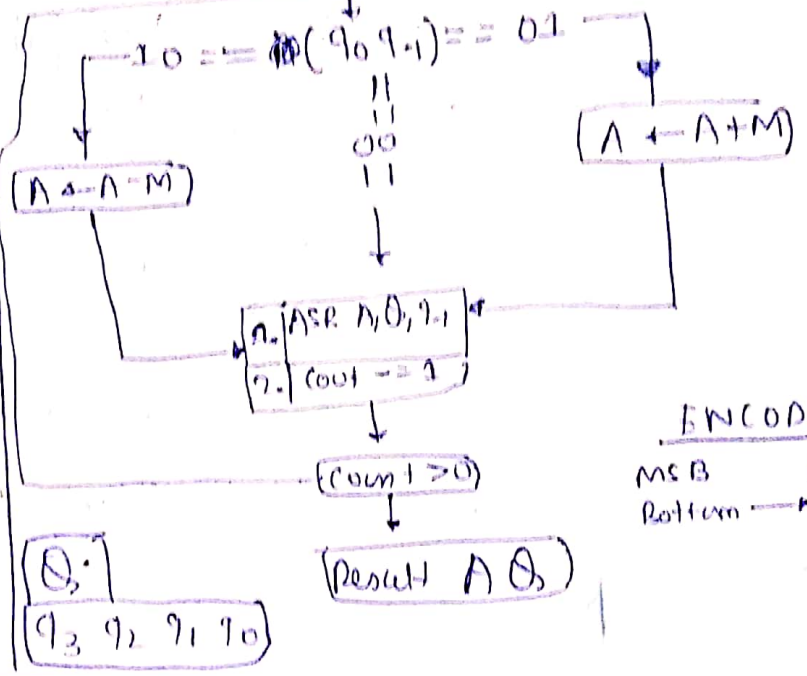
Code	q_1	q_0	
0	0	0	ASR
+1	0	1	A ← A + M, ASR
-1	1	0	A ← A - M, ASR
0	1	1	ASR

M ← Multiplicand

Q ← Multiplier

q_{-1} ← 0

A ← 0 {Size of Multiplicand (M)}



ENCODEN

MSB LSB

Pattern → 10 P

(30) Spatial Locality

(i) It means adjacent data in block access in sequence.

(ii) When array is present then there is chance of spatial locality

(iii)

Storage sequence	Access seq	Spatial Locality	
Row-wise	Row-wise	Yes	(50% Miss)
Col-wise	Col-wise	Yes	(50% miss)

# of Miss / Row = No. of Blocks / Row

# of Miss / Col = No. of Blocks / Col

ex: Element Size = 4 B

Block Size = 2 B

Element / Block = 2

Row Size = 4

Block / Row = 4/2 = 2 Block

{Row-wise Storage and row wise access}

A[4][4]

Row1 Row2

No. of element in a row

2 element in one Block

Total # Miss / Row = # row * # Block / Row

= 4 × 2

= 8 " Miss (50%)

{Row wise Storage and column wise access}

Element Size = 4 B

Block Size = 2 B

Elem / B = 2

Column Size = 4

Row Size = 4

Block / Row = 4/2 = 2 Block

But access in col. So,

Block / Col = 4

Total # Miss/col = # col * # Block/col

= 4 × 4

= 16 Miss