COMPUTER ORGANISATION

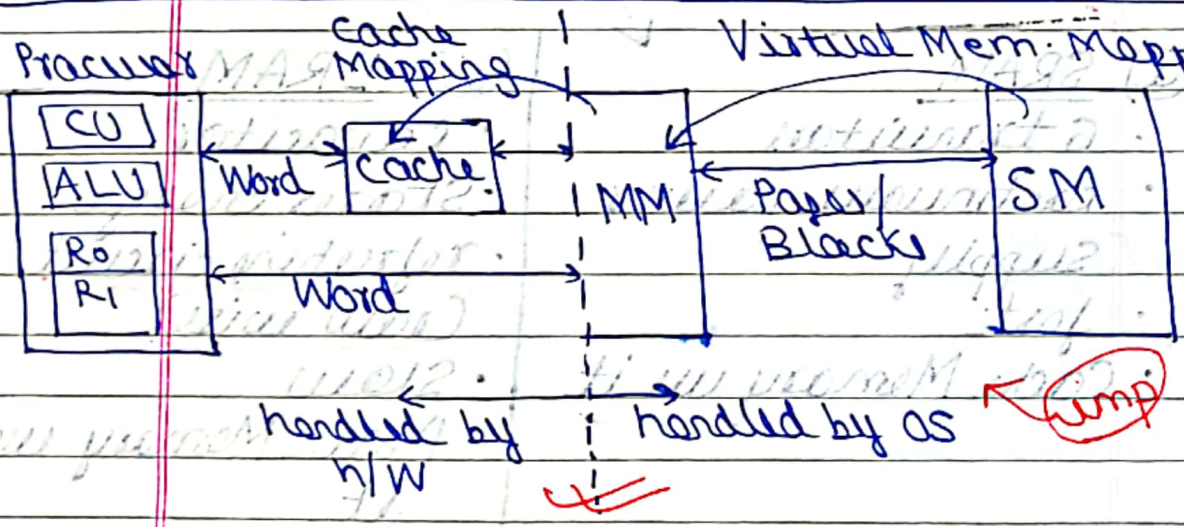
1024 bits (2^{10} bits) : 1 Kb
2^{20} bits : 1 Mb
2^{30} bits : 1 Gb
2^{40} bits : 1 Tb

4 bits = 1 nipple, 8 bits = 1 byte

* Reason for memory hierarchy:

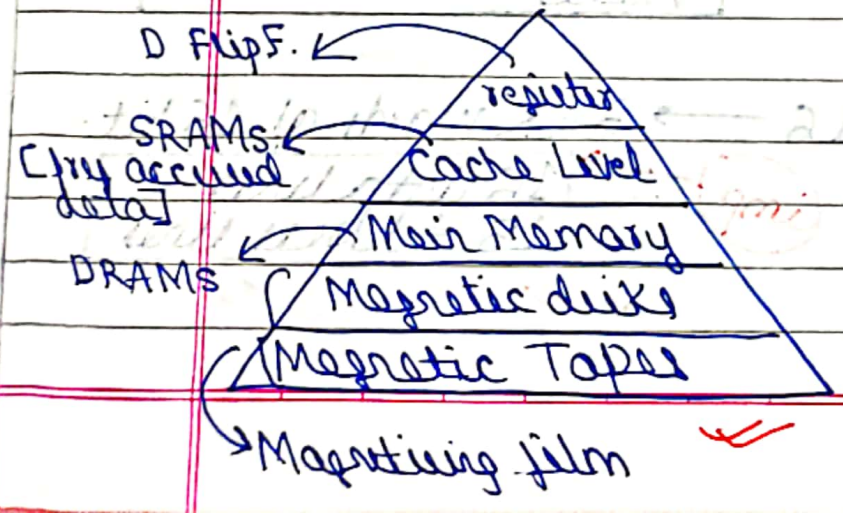
[Access time ↓, cost ↑] (trade off).

→ Accessing from Semiconductor mem. is fast (Cache / MM). (But cost ↑)



* Memory Hierarchy

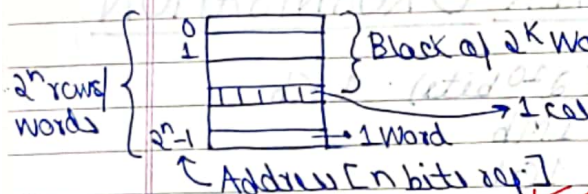
imp



- Access Time incr ↑.
- Size incr. ↑
- cost decrease ↓
- frequency dec ↓

imp

* MAIN MEMORY [any word is accessed in fixed amt of time] ∴ RAM



* if 1 word → 8 bits (byte) [Byte organis. memory]
  → 16, 32, 64 bits [Word organised memory]

imp

* Types of RAM

(i) SRAM	(ii) DRAM
• 6 transistors	• capacitor
• continuous power supply	• Store charge
• fast	• refreshing circuit (row wise)
• Cache Memory use it	• Slow
	• Main Memory use it

Vimp

* Size of RAM: [No. of Words × No. of bits/Word]

eg: if 32K × 16 → 32K words of 16 bit.

imp → 16 data lines, 15 Address lines

imp

* [No. of rows = No. of Bytes] → Byte organised Memory.

[No. of rows = No. of Words] → Word organised Memory.

Word size given

eg → Pg 13 [Q2]

eg → Pg 14 [Q2]

(Find no. of data & Add lines in Word Addressable Memory]

* Refreshment in DRAM is done row wise.

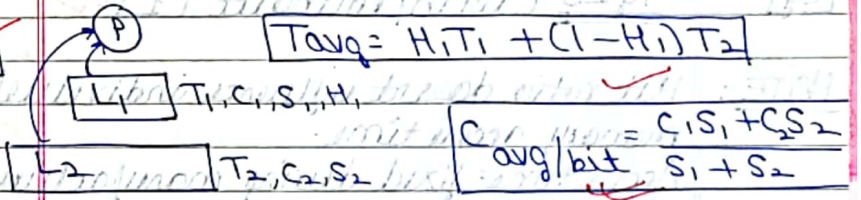
[if 2^{14} rows, time for each refresh given]

eg → Pg 16 imp

* Memory Hierarchy

1. 2 level Memory System

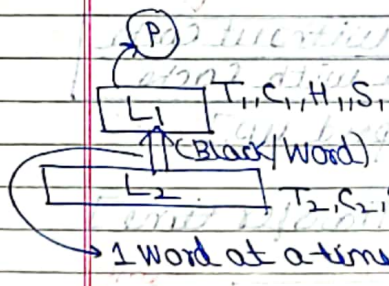
I]



$T_{avg} = H_1T_1 + (1-H_1)T_2$

$C_{avg/bit} = \frac{C_1S_1 + C_2S_2}{S_1 + S_2}$

II] Strict [Hierarchical]



$T_{avg} = H_1T_1 + (1-H_1)(T_1+T_2)$

2. 3 Level Memory System

imp

I

default

L_1 T_1, C_1, S_1, H_1

L_2 T_2, C_2, S_2, H_2

L_3 T_3, S_3, C_3

$T_{avg} = H_1T_1 + (1-H_1)H_2T_2 + (1-H_1)(1-H_2)T_3$

$C_{avg/bit} = \frac{C_1S_1 + C_2S_2 + C_3S_3}{S_1 + S_2 + S_3}$

II Strict

$T_{avg} = H_1T_1 + (1-H_1)H_2(T_1+T_2) + (1-H_1)(1-H_2)(T_1+T_2+T_3)$

eg → Pg 23 [with/without L_1]

NOTE: Hit ratio doesn't influence individual memory access time.
Access time: fixed during manufacturing.

↓. $H_R \downarrow \quad T_{avg} \uparrow$ imp

* Speed up = $\frac{\text{Time without cache}}{\text{Time with cache}}$

eg → Pg 27 [Find Speed up]

eg → Pg 28/29 [Bus Transfer time] (Given Bus Size)

default = 1W at a time imp

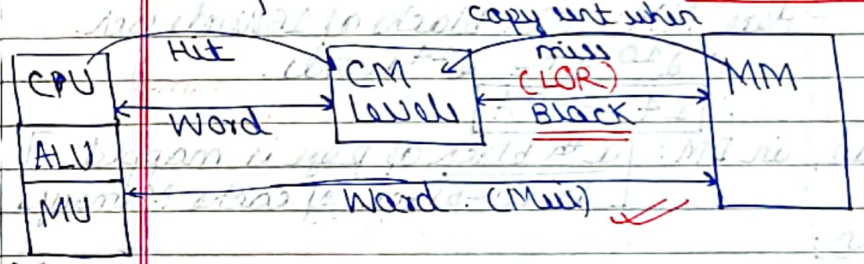
imp

* $T_{avg} = f_R \cdot T_{avgR} + f_W \cdot T_{avgW}$

eg → Pg 30 [out of 200 int., 140 were read int.] (IF & OF)
$f_R = 140/200$

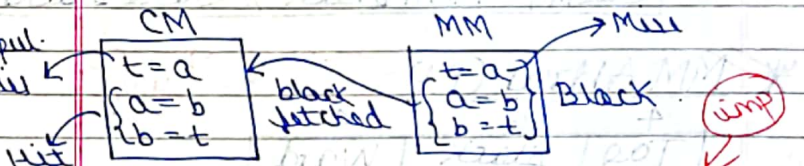
* CACHE MEMORY

- Small & fastest (SRAM)
- Frequently used data & instructions are present in cache, to minimise T_{avg} imp



* $HR = \frac{\text{no. of hits}}{\text{no. of hits} + \text{no of miss}}$; HR ↑ perf ↑

* Cache works on LOR



HR = 2/3

Block Size ↑ HR ↑ imp

Vidyalekhan™ DATE PAGE

* Performance of cache depends on:
  - Cache Size (↓) ⇒ better
  - Cache Block Size (↑)
  - No of levels of Cache (↓) (Imp)
  - Cache Mapping technique
  - Cache Replacement Policy
  - Cache updation Policy

* Cache Mapping Techniques

(i) Direct Mapping (Simple)

- Here MM: 1024 blocks of 16 words each.
  ∴ $2^{10} \cdot 2^4 = 2^{14}$ words.
  ∴ 14 bit PA (imp)
- Now, in DM: [i th block of page is mapped to ith block of cache Memory]

Given:

1 Page = 64 blocks. [Page Size = # Blocks/C]
∴ No. of pages = 16 pgs.
Cache Size = 64 blocks (Page Size)

* Cache Location of Memory Block:
  CL(M) = Mem Block % Cache Blocks

* MM Address (PA) ← LA → PA (OS), PA (H/W)

Tag	Block	Word
4	6	4

(imp)

- Tag → No of pages (16)
- Block → No of blocks/page (64) [No of cache blocks]
- Word → No of words/Block (16)

diag → Pg 11 [Rev copy]

Cache index = $\log_2$(No of Cache Blocks)

Vidyalekhan™ DATE 7 PAGE

0-63.
64-127

PA = 0001 000001 XXXX (65th block)?
[1st block of 1st Page] [Search in 1st block]

- The higher order bits are comp. with the tag after ✓ match hit [Bring from MM] (-2)
  ✗ not : fault (replace)

∴ Max Tag comparison = 1

- Accessing same block from diff pages lead to conflict miss. (imp)
- HR law, efficiency law

(ii) Associative Mapping

- A MM Block is mapped to any location in cache.
- Replacement done when cache is full.

DM: replacement done when conflict miss
AM: replacement done when capacity miss (imp)

diag → Pg 43

* PA = 000010 0000 XXXX [32nd block]
  Tag, Word
  will be compared with all the cache Tags sequentially
  ∴ Max Tag comparisons = no. of cache blocks

* Cache index = 0 (imp)

* Hit Ratio ↑ (imp)
complexity of H/W ↑
∴ (Time/Search overhead ↑)

(iii) Set Associative Mapping

- K way SAM : K blocks/set

imp → i^th block of page is mapped to i^th set of cache Memory.

- No of Sets (p) = n/K n: no of blocks in cache [64].
- CL(M) = M % p
  CL(M) = Mem Block % Sets

diag → Pg 46 [2 Way SAM]

PM Address : 14 bit PA

Tag	Set	Word

5 bit [No of pages in MM = 32].
No. of Sets (32) [5 bit]
16 Words/Block (4 bit)
imp → Page Size = # Sets

* replacement done when Set is full.
Every MM block has K alternatives.
PA : 00001 00000 XXXX.
[Search in 0th Set] ↑ [Set offset]

imp → Max Tag comparison = K
Set index = log2(No of Sets)

(1) HR ↑ (than DM)
(2) Complexity ↓ (than AM).

* Higher order bits are compared with all the tag blocks in the Set (K).

NOTE: As K ↑, P ↓ (imp), Tag Size ↑, Complexity ↑

eg → Pg 49 [Tag size in DM, AM, SAM]. [4 Way SAM ≥ Either 2 way SAM]

* Cache Capacity = Cache Memory Size + Cache directory Size

* Cache directory Size = Tag Memory + Valid bit + Ref bit + dirty bit

eg → Pg 53 [All Qs] (imp)

Tag Memory = Tag bits * No of cache Blocks.

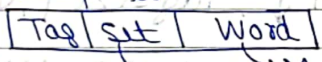
eg → Pg 57 [Tag Memory + VB + ref bit.]

eg → Pg 59 [Direct Mapped cache of 2^{12} Lines]

Tag	Line	Bytes

eg → Pg 61 (imp)
64 KB cache with word size = 32 bits
Block Size = 16 words.
∴ Word Addressable MM
Word Addressable [imp]

eg → Pg 62
Size of PA Space = 2^P bytes
Word length = 2^W bytes
No of words = 2^{P-W} ∴ (P−W) bit PA



if Word Add. [Find no. of Words]

∴ No. of blocks = $\frac{2^N - W}{2M}$

* Delay of Tag Comparator.

Delay of Tag comparator = Settling time / Hit latency / Hit delay

⇒ A 2 way SAM uses 2x1 mux.

eg → Pg 66 [DM cache & a 2 way SAM cache] (imp)

* Cache Replacement Policies

- aimed to minimize miss penalty for future references. (imp)
  [FIFO, LRU, LFU]
- No replacement strategy for DM.

*

Miss	DM	AM	SAM
Compulsory	✓	✓	✓
Conflict	✓	X	✓
Capacity	✓	✓	✓

↳ when cache is full

eg → Pg 69 [write the misses for each request]

eg → Pg 70 [Associative Mapping] (imp)

eg → Pg 71 [4 way Set Associative cache]

* HR = $\frac{\text{no. of hits}}{\text{Total references}}$ (imp)

∴ In AM → no Conflict Misses

eg → Pg 75 (Q 1)

* CACHE MEMORY & ARRAY

Steps
1) Memory Structure of Arrays.
2) Cache Structure.

A(0)	A(1)	A(2)	A(3)	...	A(99)

Vidyalekhan
DATE PAGE 12

eg → Pg 84 [Both Qs]

* 2D Array A[10][10]

00|01|02 ... 09|10|11 ...

- Array Structure
- Memory Structure (1 block brought)

A(0)(0)
A(0)(1)
A(0)(2)
A(0)(3)
⋮

- Access way

(Both HR)

eg → Pg 87 (imp)

eg → Pg 91 [Vimp Q] (imp)

* SECONDARY STORAGE

1) Magnetic Disk

- Very thin circular metal plate
- Basic unit of MD = 1 sector
- Same no. of sectors in each track.

∴ Each Track = same no. of Bytes (imp)

∴ same no of sectors.

track capacity is constant

Vidyalekhan
DATE PAGE 13

* Storage density:

ρ = No of Bytes/cm

ρ ↑ (outer to inner)

* Data transfer time (imp)

D = No of Bytes/sec

* Seek time: time to move r/w head to desired track no. (ts)

* Rotational latency: time to move r/w head to desired sector beginning (t_R)

$t_R = 1/2 * \text{Rotation time}$

* Access time of disk (Avg) (Vimp)

$t_{avg} = t_s + t_R + t_{dt} + t_{diskovh}$

(over 1 sector.)

diag → Pg 97 [Multi disk organisation]

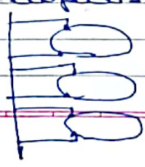
* Types of Magnetic disks:

(i) Single Sided / double Sided disks
[5 disk → 10 surfaces]

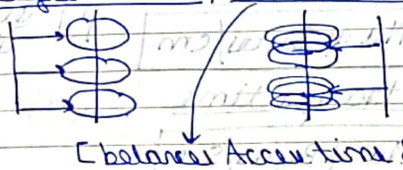
(ii) fixed r/w Head / movable r/w Head

1 r/w Head per surf (imp)
[1 rotation covers n Tracks, n = surfaces]
Hardware ↑

[1 rotn covers 1 track]
Access time ↑



(iii) Single Platter / Multi Platter :



[balances Access time & complexity]

* [No. of cylinders = No. of Tracks] imp

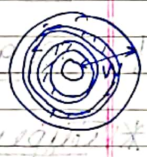
eg → Pg 99 [Q 1] [Solve all Qs]

* 1 rotation covers 1 track.

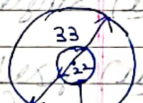
imp [data transfer rate = no. of bytes / s]

⇒ [Fixed r/W Head transfer rate = no. of * movable r/W head T.

* w = width of Storage
x = width of track
y = Gap.



→ No. of tracks = w/x+y.
→ If no Gap = w/x.
→ If only Gap = w/y



eg → Pg 104 imp

Max Storage density = 2000 bits/cm.
[Refers to circumference of innermost track]

* <c, t, s>
  - s → No. of sectors / track.
  - t → No. of track surfaces / cylinder.
  - c → No. of cylinders.

eg → Pg 106 [<400, 16, 29> corresponds to which sector number]

* Data Organization on Disk
  - Surface: <surface no., track no., sector no.>
  - Cylinder: <cylinder no., track sur/no., sector no.>

eg → Pg 110 [Rotation latency & data transfer Rate]

eg → Pg 113 [Avg. Access time] imp

$T_{avg} = t_S + t_R + t_{OH} + t_{dt}$

time in POW(10)

time required a sector.

* Disk Structures → CDs
  - Constant Angular Velocity
    [Same Track capacity]
    [ρ varies]
    [Low Access time]
  - Constant Linear Velocity
    [ρ constant]
    [Track capacity varies outer ↑]
    [More Access time]

Vimp

I/O INTERFACING

CPU ⟷ MM
CPU involves in data transfer.
I/O

⇒ Data Transfer modes:

(i) Programmed I/O / programmed control
- I/O devices don't have direct access to mem
- CPU involves for:
  - device status (polling) · status of transf
  - I/O initialization
  - data transfer
- inefficient.

(ii) Interrupt driven I/O
- when device ready, sends interrupt
- CPU involves for:
  - I/O initializ^n · status of transfer
  - data transfer
- → 1 with less overhead

(iii) DMA Transfer
- large volumes of data transfer
- processor relieved completely
- → only I/O initial^n done by by CPU.

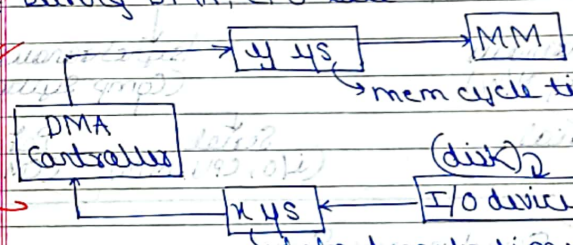
diag → Pg 127 (jump)

DMA [Direct Memory Access]

DMA Modes: The way, DMA takes control of CPU / System bus.

1. Cycle Stealing mode: DMA returns bus after a word transfer
2. Block mode return bus after block transfer.
3. Burst mode: after complete data transfer.
4. Interleaved mode: [½ cycle → DMA, ½ cycle → CPU]

NOTE: During DMA, CPU idle.



jump

$\%\ CPU = \frac{x}{x+y} \times 100$; $\%\ CPU_{idle} = \frac{y}{x+y} \times 100$

eg → Pg 132 [In cycle stealing mode, DMA CPU unit returns bus after 1 word transfer]
Given: mem cycle time (y).

eg → Pg 135 [16 bit count register]

$P \propto D$ (jump) ; $P \propto \frac{1}{\text{Transfer time}}$

P → system ; D → data transfer rate

eg → Pg 136 [2015] (Vimp)

⇒ [Peak data transfer rate = no. of words transferred / Src]

[Overhead = Time to transfer 1 word/byte]

eg → Pg 139 [Vimp]

* I/O TRANSFER

- Synchronous (Comp N/W) → Serial
- Asynchronous (Comp System)
  - Serial (i/O, CPU, MM)
  - Parallel (CM, pipeline)

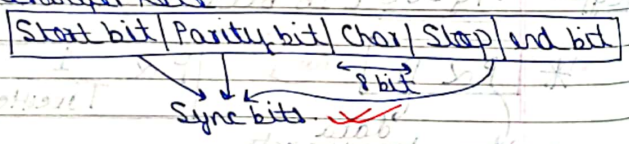
1) Synchronous Serial

- Synchronous characters must be excluded from data transfer Rate.

(Imp)

2) Asynchronous Serial

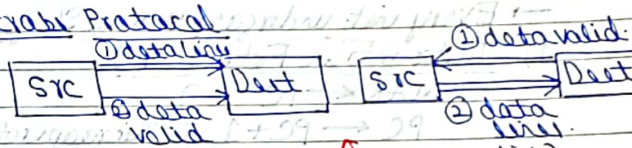
- Sync. bits are included in data transfer Rate.



* Rate at which Serial info is transmitted
⇒ baud Rate.

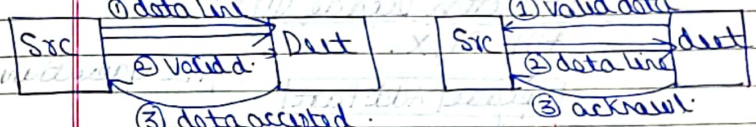
3) Asynchronous Parallel → Strobe, Handshake

(i) Strobe Protocol



(ii) ~~Rate at which info is~~ (Na ack.)

(iii) Handshake Protocol



eg → Pg 144 (Vimp)

* MACHINE INSTRUCTIONS

- How CPU executes 1 instruction.

diag → Pg 145 (Imp)

PC : holds address of next inst.
IR : holds current inst. code
MAR : holds address of memory to be referenced.
MDR : holds data to be read / write

* INSTRUCTION CYCLE

→ Every inst. undergoes some stages.

1) **Instruction Fetch (IF)**
   - MAR ← PC
   - PC ← PC+1
   - Read
   - IR ← MDR

   } Microoperations. (imp)

   Aim: Getting curr. inst. in IR.

2) **Instruction Decode (ID)**
   - ADD R1, X.
   - opcode | Add. Part
   - 1011 → Decoder → Adder circuit activate

   No operation

3) **Operand Fetch (OF)**
   - MAR ← LOC(X)
   - READ
   - Ri/ALU ← MDR

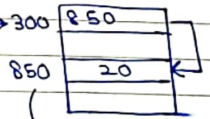
   X: operand. Ri: register.

4) **Execute (EX)** → perform ALU opn

5) **Write Back (WB)**
   - MAR ← LOC(Z)
   - MDR ← R1-R2/ALU/Ri
   - WRITE.
   - [Write an Add. loc^n Z in MM]

6) **Indirect Cycle**

ADD R1, #300



∴ Pointer (2 mem cycles)

→ effective add.

4) **Interrupt Cycle**: If interrupt occurs, CPU needs to serve interrupt.

NOTE: Processor performs any micro opn in reg. only (be it Arithmetic/logic/Shift/register transfer).

If an inst. takes 7 CC, it means it has 7 micro-operations. [7 Steps]

1 operation = 1 CC | 1 Steps = 1 CC

(imp)

5) INTERRUPT PROCESSING

→ Current Program suspended. (Vimp)

→ Stores (register Values, CPU Mode (PSW), Status flags, return Address) in Stack (present in MM) → used in Branch inst.

→ IP is an overhead to CPU.

diag → Pg 152

NOTE: Prog: [I1, I2, I3, I4]
New interrupt occurs when CPU is exec. I2.
CPU first completes I2, then Stores Return Address (I3) (gets from PC). (Vimp)

* CPU Modes

- User (Non-Privileged Mode)
- Kernel/System (Privileged mode). [perform OS related SERV.]

* Interrupt Types:

(ii) External / H/W Interrupt:
- due to timer / I/O device
- priority encoders. (simultaneous i/p activation).

(iii) Internal / Prog. interrupts: (imp)
[divide by zero
Invalid opcode
register overflow]

(iii) Software interrupt:
Arised due to context Switching

* T_p = No. of cc * t_c
T_p: Time to complete the prog.
t_c: cc time

eg → Pg 164 [Byte Addressable mem]
- If interrupt occurs during HAULT (last inst.); PC is not updated (imp)

eg → Pg 165 [Prog given]

* PIPELINING

- It helps to speed up execution
(I) non-Pipeling:
Inst → {IF, ID, EX, WB} (4 Phase)
Assume → Each Phase: 1CC (imp)
3 inst? ⇒ 3x4 = 12 CC

* Non-Pipeline: At any time, only one instruction is executed. (imp)

* Phase-Time diag. (Pipeline)
[9 instructions]
- each unit of CPU takes care of each Phase (parallel functioning)

diag → Pg 13 [At CLK 7, I_7 fetched inst, I_6 in ID, I_5 has begin execution, & I_4 result is write back]
∴ 4 inst. simult. at 1 CLK.

Aim: Aim of Pipelining is to make: (imp)
CPI ≈ 1
or, IS = 1 instruction
(Here after 4th clock, CPI = 1)

* Performance of Pipeline

Speed up = $\frac{T_{wp}\ (\text{Time without Pipeline})}{T_p\ (\text{Time with Pipeline})}$ (imp)

⇒ Speed up = $\frac{4 \times 9\ CC}{12\ CC} = 3$

m * efficiency = Speed up.
(m: no. of stages)
∴ efficiency / utilization = 3/4.
(used box / Total box)

If efficiency = 1, Speed up = m.
Practically, Speedup < m

* Speed Up :

'n' = Total instruction
'm' = Stages
$T_p = 1 \times m + (n-1) \times 1$ ← imp
$T_{wp} = m \times n$
Total CC

∵ 1st unit takes m clock cycles

Speed up = $\frac{T_{wp}}{T_p} = \frac{mn}{m+(n-1)} = \frac{m}{1+\left(\frac{m-1}{n}\right)}$

[every clock we get 1 unit] → ∴ if many instruct^n : Speed up = m

eg → Pg 19

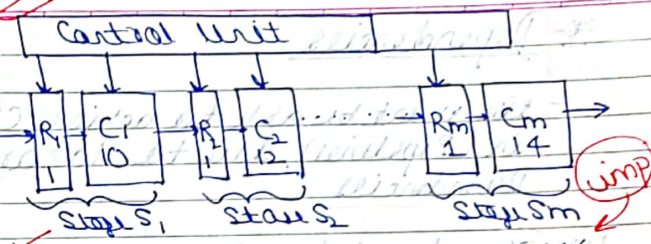
* Non-Pipeline (CPU → 1 unit) (T_1)
Pipeline (CPU divided into units & buffers ∴ delay) (T_2).
if 1 instruction
$T_2 \geq T_1$ (Pipeline takes more time)

* Efficiency = 3/4 ;
ideally Sp could have been 4 if n → ∞ (m)
but we made it 3.

eg → Pg 18 [23 CC]

* INSTRUCTION PIPELINE → H/W

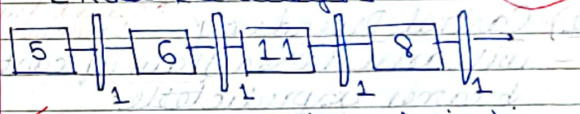
'm' Stage Pipeline : CPU is divided into m independent modules.
Buffer registers : stores result of each stage.



Take largest Stage delay as 1 CLK.
1 CLK = 15 ns ∴ 1 CC = 15 ns

eg → Pg 19 ← Stage delays
m = 4, [150, 120, 160, 140]
register delay = 5 ns
∴ 1 CC = 165 nS
∴ $T_p = [1 \times m + (n-1) \times 1]$ CCs
⇒ (4 + 999) CCs

eg → Pg 20 (Speed up ?) CPI = 1
CLK = 12 ns ∴ 12 nS per Inst.
Time without Pipeline
[Add the delays]



Speed up = $\frac{\text{Time without pipeline}}{\text{Time with Pipeline}}$
30 ns per unit → ⇒ $\frac{5+6+11+8}{12} = 2.5$
12 ns per unit →

eg → Pg 21 [2015 Q]
∴ In Pipelined unit, take CPI = 1
Test

Vidyalekhan
DATE
PAGE 26

* Dependencies

- We want be able to achive CPI = 1 in (Pipelines), due to Hazards/ Dependencies.

(1) Structural dependency

- when 1/more phase of the pipeline try to use the same resources.
  eg: IF, OL both trying to access the same resource/ cache.

	1	2	3	4	5
I_1 :	IF	OL	EX	OS	
I_2 :		IF	OL	EX	OS

(imp)

Solution: resource replication/duplication.
[have multiple copies of resources]

* Resource replication in MIPS RX000.

Ref → 9th Pic

(2) Control Dependency (caused due to branching)

- will arise when flow of control becomes unpredictable:
  - Conditional branches
  - Unconditional branches
  - Function calls/ return
  - Prog control inst that may affect PC.

→ due to this, CPI ≠ 1

Vidyalekhan
DATE
PAGE

diag → Pg 25

CAT 6th CLK, we get the next inst. Addrss, but by that time, already S4, S3, S2, S1 of inst. computed.
∴ Flushing
[we loose cycles/ penalty]. [CPI > 1]

* Solution:
  (i) Flushing
  (ii) Stalling Pipeline (leaving it empty till CLK 6)
  (iii) NO OP (Compiler recognises BI & introd. no operations). [branch delay]
  (iv) Reschedule/ rearrange: some inst. round in NOP, whose movement doesn't effect prog as a whole.

* Let there be 100 inst. [CPI = 1]
  [80 N Branch, 20 Branch Inst]

⇒ o/p of Branch Inst. will be known at the end of 2nd Stage. [CPI = 2]

	1	2	3	4	5	6	7	8	9	10
I_1	S_1	S_2	S_3	S_4						
I_2		X	S_1	S_2	S_3	S_4				
I_3				S_1	S_2	S_3	S_4			
I_4					S_1	S_2	S_3	S_4		

(Imp)

(for every CLK, 1 inst coming out)
[CPI = 2]

∴ Avg = $\frac{80 \times 1 + 20 \times 2}{100} = 1.2$ CPI

→ [From Pg 27: Refer book (Notes) [Ravi]]