Computer Architecture

Refers to the design, structure & organization of computer system

- It defines how processor, memory & I/O devices communicate & work together.
- it includes instruction sets, m/m hierarchy & control mechanism

Technology trends — focusses on how components like processors memory storage and network are improving with time

Ex Faster processors
improvement in wireless communication like 5G
- ~~AI, cloud computing~~
smaller components

Fast CPU with more cores

less power consumption

PAGE No.
DATE

Moore's Law - proposed by Gordon Moore in 1965

- It states that the number of transistors on a microchip doubles approx every 2 years.

- This leads to exponential growth in computing power while size and price reduce

Application - More powerful processors, better graphics, Faster running AI, Cloud computing

Parallel computers - are designed to process multiple tasks simultaneously to improve speed & efficiency

SISD - Single instruction single data
- Traditional computers
- where 1 instruction operates on single data element at a time

SIMD - Single instruction multiple data

- Used in GPU
- Single instruction operates on multiple data elements

MISD - Multiple instruction single data

Used in fault tolerant systems

- Multiple instruction operate on same data

MIMD - "M" multiple data

- Used in multiple core processors
- Each processor works independently on its own instruction and data

core in a processor is an independent processing unit within CPU

eg dual core → can handle 2 task efficiently
quad " 4 "

Think of cores of workers in Factory
single core can do only 1 job at once

Myopic view of C.A

- is a narrow perspective that focusses on improving the processor speed without considering factors like

System cost
Energy consumption
Reliability, maintainability

A balanced view is required to ensure Practical system design

Amdahl's law - used to Shows the limit of speedup by parallelising only a part of program

Speedup – ratio of time taken to run program on a single processor to the time on multiple processor

$$S = \frac{T_1}{T_n}$$

T_1 ← single processor (Sequential)

T_n ← N processor (Parallel)

measures how much faster a task executes when parallel techniques are applied.

$S = 1$ No improvement

$S = N$ Perfect speedup

$S < N$ → Practical case

Efficiency = $E = \frac{\text{Speedup}}{\text{no of processor}}$

Measures how well the processors are utilized in a // system

Formula for speed up

$$\text{speed up} = \frac{1}{(1-P) + \frac{P}{N}}$$

(1−P): sequential fraction; P: // fraction; N: no. of processors

P → fraction of program that can be parallelized

N → number of processor

eg if 80% can be // then P = 0.8

eg if 40% of it is floating pt operation and factor is 4 find speed up

Aptitude		Formula
Serially	Parallel	$\frac{1}{1-0.4+\frac{0.4}{4}} = \frac{1}{0.7}$
60%	40%	
no speed up	↓ factor 4	
so	so $\frac{0.4}{4}$	
$0.6 + 0.1$		
Earlier it was	$\frac{1}{0.7}$	
now		

if we have 1 processor

Serial (S) — time to execute serial part

Parallel (P) — time to execute // part, n = 1

$T_1 = S + P$

$T_n = \frac{S + P}{n}$ $\quad S + \frac{P}{n}$

assuming $T_1 = 1$

$1 = S + P$

$S = 1 - P$

Replacing in T_n

$T_n = (1 - P) + \frac{P}{n}$

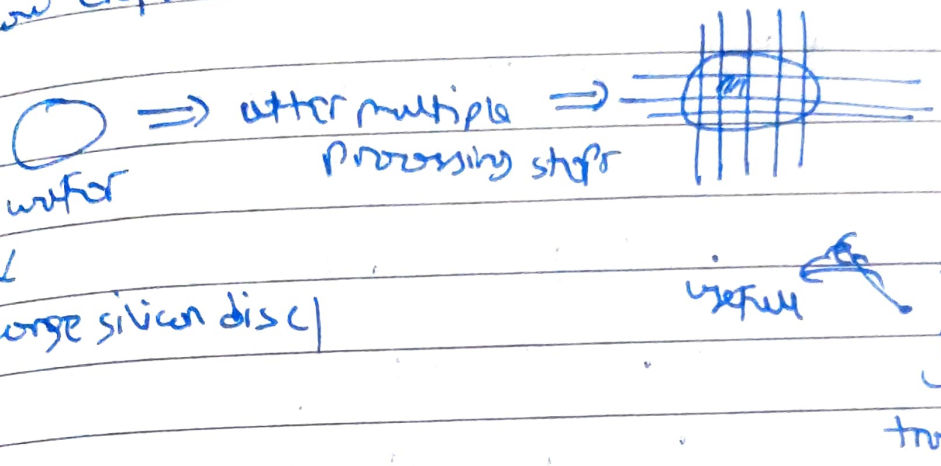
Speed up $= \frac{T_1}{T_n} = \frac{1}{(1-P) + \frac{P}{n}}$

Die = chip

PAGE No.
DATE

Fabrication cost

how chips are made



Fabrication yield - refer to the % of Functional chips produced upon the total no of chips

$$\text{yield} = \frac{\text{Number of good chips}}{\text{Total no of chips on wafer}} \times 100$$

Eg Cost of wafer = ₹2000

Number of dies per wafer = 1000

yield = 80%.

ie only 800 good die

$$\text{Cost per chip} = \frac{2000}{800} = 2.5$$

$$\text{Cost per chip} = \frac{\text{Cost of wafer}}{\text{Total no of good chip}}$$

$$yield = e^{-ad}$$

a → die area

d → defect density

Eg defect density = 0.08 / cm²

wafer area = 150 cm²

No of wafer per run = 500

Cost per wafer = ₹ 1000

Dies per wafer = 1000

$$yield = e^{-\text{defect density} \times \text{area of die}}$$

area of die / wafer → $\frac{150}{1000}$ → 0.15

$$e^{-0.08 \times 0.15}$$

$$\doteq 0.988$$

yield = 98.8 %.

Cost of production

Cost = No of wafer × cost per wafer

200 × 500

Cost per die = $\frac{\text{Cost}}{\text{No of functional die}}$

No of functional die = $\frac{98.8}{100} \times 1000 = 988$

Cost per die = $\frac{\text{wafer cost}}{\text{no. of dies}}$ $\frac{2000}{987} = 2.024$

CPI → clock cycles per instruction

It is a performance metric that tells us the average number of clock cycles required to execute a single instruction in a program

Lower CPI → Faster execution
Higher CPI → Slower execution

$$CPI = \frac{\text{Total clock cycles}}{\text{instruction count}}$$ ← total cycles taken to execute prog

CPU Execution Time = C.P.I × instruction count × clock cycle time

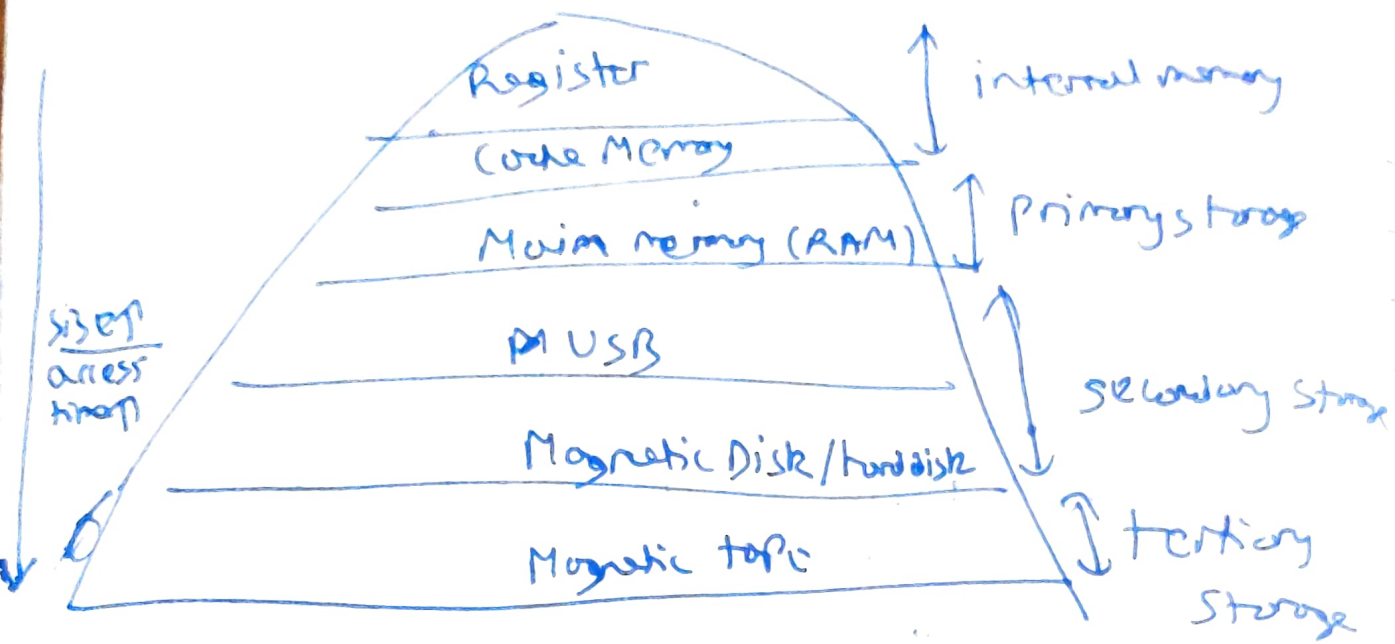
or

$$\frac{\text{C.P.I} \times \text{instruction count}}{\text{clock rate}}$$

in Hz ← clock rate (freq)

Clock cycle time = $\frac{1}{\text{clock rate}}$

M/M hierarchy



Locality of reference

temporal locality : Recently accessed data will be used again

spatial locality : data near recently accessed data will be accessed soon

Coherence - In multiprocessor system multiple cache have copies of same data

Cache coherence ensures that all processor see the most recent copy

Cache - Small, Fast memory located b/w CPU & main memory

it stores copies of frequently used data so that CPU can access them quickly instead of going to main memory

↳ built using SRAM (Static RAM)

↳ works on principle of locality of reference

Levels of cache L1 - closest to CPU, Fastest

L2, slightly slower

L3 - Larger, slower than L1/L2 but faster than RAM

Reduce CPU access time

Mapping techniques

- Direct mapping
- Fully associative mapping
- Set associative mapping

Unit for memory is word

Cache is divided in lines

A block from main memory is placed in the cache

basic concept

ram se cache m block kaise aayega and cache m kaha rakhna and CPU kaise access karega ye sab is mapping techniques

When CPU want data

1) it checks cache → if the block is there

if yes → cache hit

if no → cache miss (Fetch from RAM)

2) if block not in cache, CPU decides where in cache to place it

This is called cache mapping

Cache mapping decides how RAM blocks are placed in cache, so CPU knows where to look when it needs data

Direct Mapping

Each block of RAM can go to only Fixed line in cache

- Simple, Fast

Disadvantage: Frequently used blocks may collide

1 word = 1 Byte

Line size = block size

PAGE No.
DATE

CPU searches all lines in // to find block

Advantage - No conflict

Dis - Slower searching expensive hardware

Key Set Associative - Cache is ÷ into sets and each set has a few lines

A block of ram is mapped to one set but inside that set it can go in any line

Adv - Better b/w direct and associative

dis - Slightly more complex

	Direct	Fully associative	Set associative
Flexibility	Low	High	medium
Hardware cost	Low	High	med
Conflict	High	V. low	low
Speed (lookup)	Fast	Slow	balanced
Hit rate	Lowest (due to high conflict)	Highest	medium
Suitability	Suitable for simple systems (micro controllers)	Suitable when hit rate is critical Eg CPU	Used in general purpose CPU like laptop, desktop

1 word = 1 byte

sizeOf block = size of cache line

K way set associative

→ means no of lines in a set

eg eg K = 2, 4, 8

means that divide lines in

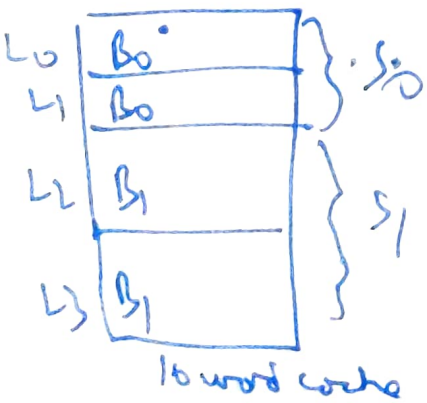
eg Cache has 16 lines 2 way set associative

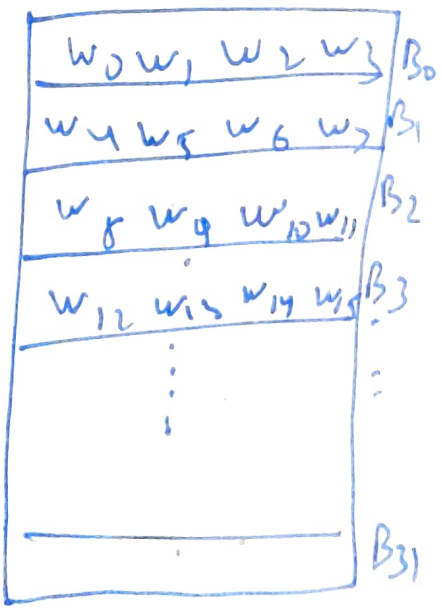
so no of set = 8

each with 2 lines

eg 16 word cache 128 word M/M

Each block can have 4 word





so no of block = $\frac{128}{4} = 32$

lines => slots

PAGE No.
DATE

block size = 4 word
so line size also 4 word
so no of lines = $\frac{16}{4} = 4$

if 2 way associative ie 2 lines per set
~~K mod n~~
no of set = $\frac{\text{Total line}}{K} = \frac{4}{2} = 2$ sets

(K) mod (n)] ← ki block kause set
↙ ↘ set   m jayega
block

eg 0 mod 2 B_0 in set 0
1 mod 2 B_1 in set 1
2 mod 2 B_2 in set 0

and set m block
kahi bhi ho sakta h   (associativity concept)

main memory

tag	set no	
4	1	2

↗ Block offset

B NO	Block offset
5	2

~~bit~~ to represent 2 we need 1 bit

eg tag 2 ↓, ↙ set 0
0010 0 00
ie block.

now in set 0 it will look for tag
in any of lines it can get it

Other than offset rest is block no

Eg offset = 4 bit
block size = 4 word

Eg P.A = 1011 0110 1100
Tag index offset

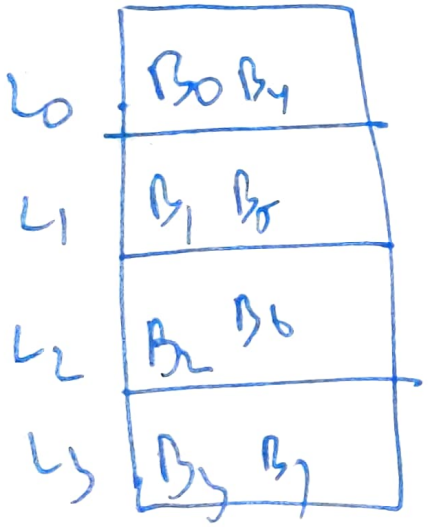
8	4
block no	offset

Now this is into tag and index

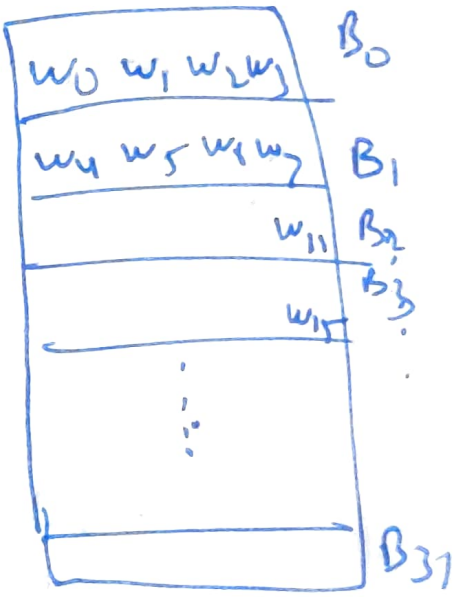
in line 0110 ie 6
Check the tag 1011 if it is there block is Tag + index
1011 0110 =

Direct mapping

Example ~~Workstation~~
4 ~~words~~



Cache size 16 word



128 words

line size = block size = 4 word
so no of line

$\frac{\text{cache size}}{\text{line size}} = \frac{16}{4} = 4$

To block size = 4 word
so Total block $= \frac{128}{4} = 32$

128 → 0 – 127

no of bits $= 2^7 = 128$

no of bits = 7

Ek time pe ek hi block in cache

Direct Mapping

Formula : k mod n

k → block number

n → no of cache lines

Ki konsa block konsi line mein jayega

block 0 → 0 mod 4 → 0th line

Main mm → how to get?

Physical address

size of mm = 128

⇒ 7 bits

Physical

block no	block offset

(PA) 7 bits

1 bit 2 bit

P.A	block no	offset

each block mein words

⇒ 4

to get 4 we need 2 bit

Eg. if CPU says give 2

0001010 ie I need 10th word

0001010

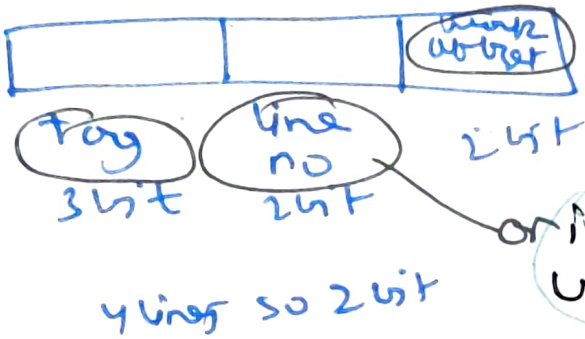
5 2

ie 2nd block 2nd word

Why P.A of same size of of memory in cache? Ans

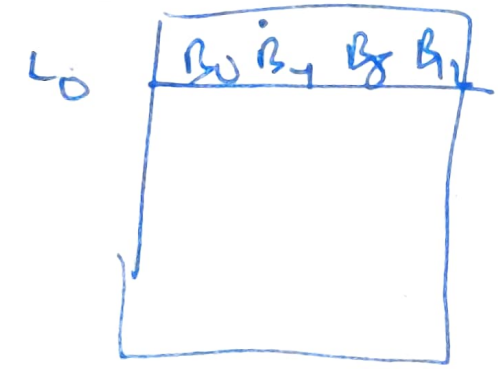
now For cache

The size remains same 7 bits



bit offset remains same as each line = block size

Tag tells that Ki line m konsa block



Block number = Tag + line no

Og line no

Tag line no bo offset

001 00 00

16 8 4 2 1

ie 4th word is in 16 line 0

See Tag & line no one word

001	01	00	w_{20}
001	01	01	w_{21}
001	01	10	w_{22}
001	01	11	w_{23}

Ans a line can have only 1 block out of my to check which block is present see the possible word in line 1 right now

The stores copy of block with same block address as in M/M

P.A → actual address of data in memory

In Fully associative

P.A

same ex

5	2
block no	offset

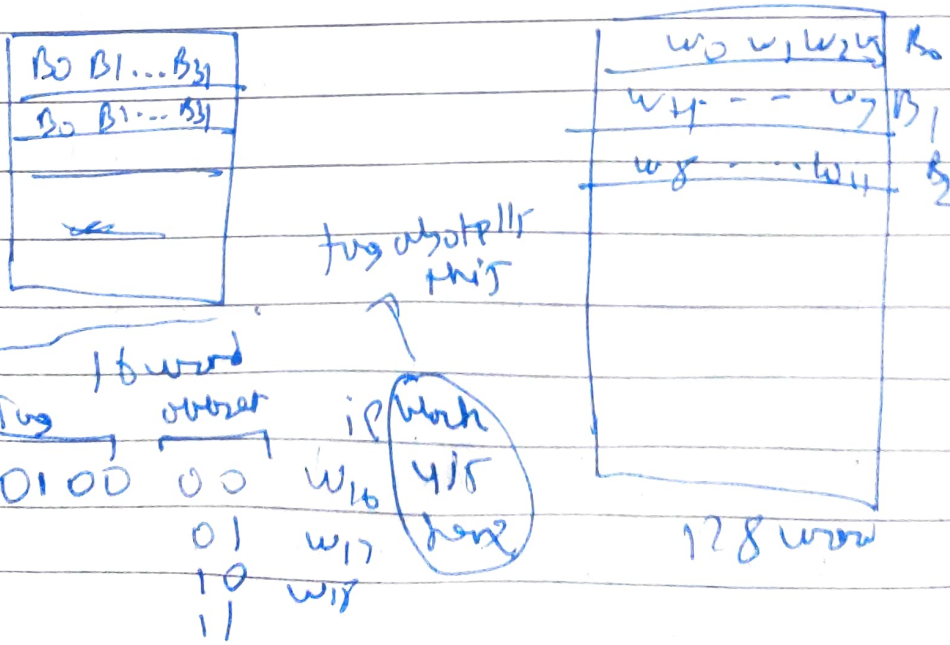
Now since in cache any block can go to any line so line no is not required

P. A of cache →

5	2
Tag	offset

tag tells us block no i.e in a line any block can be present of 32 block so 5 bit more any block

See. Question for tag



To convert decimal
in binary
First
Decimal to hex
then write it in binary

Iron law of performance - states that the performance of a program depends on 3 Factors.

1) no of instructions
2) Clock cycle per instruction
3) Clock cycle time

CPU Execution time = Instruction count × Clock cycle time × C.P.I

Performance ∝ 1/(Time required for 1 clock cycle)

TO improve the performance / make Prog Faster

- Reduce instruction count
- Reduce C.P.I → better architecture (pipelining)
- Reduce clock cycle time → Faster Hardware

1 KB = 1024 bytes
1 MB = 1024 KB

Just for understanding

CPU → Page table → main kabhi hi nahi frame hota h
(Ki might ye frame chahiye)

Virtual memory – memory management technique used by modern O.S that gives the illusion of a large, continous main memory (RAM), even if the actual Physical m/m is smaller

it allows prog to believe it has access to large block of m/m by using secondary storage as an extension of m/m

- implemented using Paging or segmentation

How virtual m/m Abstract & Extend Physical m/m

- Abstraction means hiding the complexity and presenting a logical view

In V.M system
O.S hides the real physical m/m layout and each process sees a simple clean continous block of m/m starting from 0 even though in reality it is scattered in RAM
This is called address space abstraction

Physical m/m is limited ie 8 GB or 16 GB
byt Progams may need more space than this

V.M extends physical memory by using a portion of secondary memory as virtual m/m

How it works
Only active pages (currently used) of prog are kept in ram
inactive pages (not recently used) are moved to the disk

when a prog needs them they are brought into RAM
Then it is Page Fault and using Page Fault servic which they are brought in ram

Fundamental concept of V.M

i) V.M provides address space abstraction
- Each prog it given its own virtual address

These virtual addr are mapped to physical addr in RAM by the Memory management unit using a structure called page table

if any page is required that is not in M/M then page fault and using page fault service routine it is brought into M/M

Components of V.M system

1) Page table - is a data struct that is maintained by OS that is used to map virtual address to their physical address in the main memory

- keeps track of which virtual page is stored in which Frame of main M/M

2) P.T.E - Each entry in the page table is called page table entry

- it contain info about virtual page like
  - Frame number
  - valid / invalid bit if the page is in memory or not
  - Dirty bit if write operation is used on page
  - Reference bit if page is modified recently

Translation Lookaside buffer - small fast cache inside the CPU that stores recent Virtual to physical address translation

- Accessing the P.T for every translation is slow
- T.L.B reduces the overhead by storing the translation for frequently used Pagetable entries

CPU generates V.A

M.M.U checks TLB if found TLB hit fast access

if not TLB miss Page table consulted

how Page table Maps

- Virtual add — Page No, offset
- MMU uses Page no to find the corresponding Frame Number in the Page table
- Frame number + offset gives P.A

PAGE No.
DATE

Advantage of T.LB

- improves / reduces memory access time
- Better CPU utilization
- improved throughput
- Reduces the time required for address translation
- Stores Frequently used Page table entries translation