Date _____ Day _____

CHAPTER # 02

VON NEUMANN ARCHITECTURE :

→ The classical von neuman architecture consist of
① Main memory ② CPU ③ Processor / Core ④ Interconnection b/w mem & CPU.

↳ Main Memory consist of collection of locations used for storing instruction & data

↳ CPU is divided in control unit & ALU.

CU → deides which instr. to execute thru a program counter registes.

ALU → oesponsible for executing actual instr.

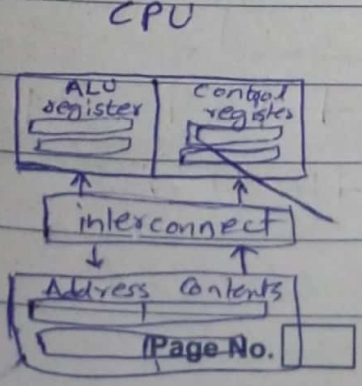
CPU contains register which contains state of a program

↳ Bus acts as an interconnection b/w mem & CPU

→ A von neuman machine executes a single instr. at a time & each instr operates on only a few pieces of data.

→ Transfer of data from mem to CPU is called fetch or read.

→ Transfer of data from CPU to memory is called written to mem or stored.



Date _____ Day _____

→ The seperation of mem and CPU is often called Von Neuman bottleneck, since the interconnect determines the rate at which instr & data can be accessed. Therefore the rate of data transfer through interconnected bus is way slower then the CPU's processing speed. CPU

Solution:

① Using cache memory which are fast & close to CPU.
② Pipelining i.e to break down inst into stages
③ Parallelism i.e to use multiple processing

"The Basics Of Caching"

Cache is a collection of memory locations that can be accessed in less time than some other memory locations.

↳ The principle that an access of one location is followed by an access of a nearby loc. is called locality.

↳ Nearby location → Spatial locality

↳ Location to be used in near future → Temporal locality

Cache Block / Lines: Memory access will effectively operate on blocks of data and instr instead of single data item / inst.

Levels of Cache: Cache is usually divided into levels.

① L1, L2, L3....

L1 being the smallest & the fastest.

Date _____ Day _____

Multilevel cache don't duplicate information that's available in another level. When data is needed by CPU, it searches from top to bottom & then in main mem.

→ If the data is found in the cache it's called cache hit or L1 hit or hit.

→ If not found then cache miss / L1 miss / miss.

→ If a value in cache is updated, the val in cache & main mem are inconsistent. Two approaches to deal with this:

↳ Write-through caches, the line is written to main mem when it is written to the cache.

↳ Write-back caches, the data isn't written immediately. Rather, the updated data is marked (dirty), and when the cache line is replaced by a new cache line from memory, the dirty line is written to memory.

"Cache Mappings"

Direct Mapping ↓ Fully Associative ↘ n-way set Associative

① Direct Mapping; → Ismein no. of hits ~~kam or miss~~ zyada hojate hai ek sabkei loc fixed hai to empty space zyada hota hai

-> Is mein hum har ek block of main memory ko ek fixed cache line assign krdete hai. Assign krne k liye we use K mod n, where K = Block No. & n = line no.

→ Words & Bytes both are same
→ Jese main memory mein frames hote hai wese hi cache mein lines hoti hai. Line or block/frame ka size same hoga.

Date Day

Cache (ek waqt mein inmein se koi ek hi ayega.)

L_0	B_0 B_4 B_8 B_{12} - B_{28}
L_1	B_1 B_5 B_9 B_{13} - B_{29}
L_2	B_2 B_6 B_{10} B_{14} - B_{30}
L_3	B_3 B_7 B_{11} B_{15} - B_{31}

16 Words

Line Size = Block Size

$\frac{16}{4} = 4$

Main Memory

W_0	W_1	W_2	W_3	B_0
W_4	W_5	W_6	W_7	B_1
W_8	W_9	W_{10}	W_{11}	B_2
W_{12}	W_{13}	W_{14}	W_{15}	B_3
⋮				⋮
W_{124}	W_{125}	W_{126}	W_{127}	B_{31}

128 Words

Each block have 4 words

so total no. of blocks = $\frac{128}{4} = 32$

$B_0 \to 0 \bmod 4 = 0$

$B_2 \to 2 \bmod 4 = 2$

$B_{31} \to 31 \bmod 4 = 3$

• Ab jab bhi B_1 cache mein jayega to wo L1 mein jayega. Sab blocks ki location fixed hoti hai.

• Finding Physical Address in Main Mem:

→ As we have total 128 words so no. of bits required to store the address of 128 words = 2^7

Block Number	Block offset
5 bits	2 bits

⇒ Physical Add. (7 bits)

0001010 → W_{10}

2 2 (block off = 2)

Block no = 2 set 4 2nd index

Page No.

Date Day

• Finding Address in Cache

7 bit

Tag	Line No	Block Offset
3 bits	2 bits	2 bits

ek line mei total 8 blocks mein se koi bhi aasakta hai. To shuru ki 3 bits will help us to find the block no.

Tag	Line	Block Off	
001	01	00	→ W_{20}
001	01	01	W_{21}
001	01	10	W_{22}
001	01	11	W_{23}

ye sab words B_5 mein hai to ~~iska mtlb cache ki~~ L1 mein B_5 pada hai

② Fully Associative Mapping:

→ Ismein koi bhi block kahi par bhi aasakta hai.

→ Ismein hits to zyada hojate hai lekn no. of comparisions to check that whether a block is present in cache is equal to total no. of lines.

③ N-Way Associative Mapping:

→ N ki value humein given hogi like 2-way, 4-way, n-way associative mapping.

→ Hum apne cache ko n sets mein divide kordenge

→ Main memory se cache mein jo block/frame ayega wo K mod S (set no) ki basis par ayega. Ab ek set mein 2, 3, 4, -- kitni bhi lines hosakti hai. ab agr koi block S_0 mein store hona hai to wo S_0 ki kisi bhi block mein hosakta hai.

7

Tag	Set No	B offset
4 bits	1 bits	2 bits

Page No.

Date _____ Day _____

→ LRU (Least Recently Used) Algo is used to replace the cache lines.

! Revise Virtual memory, pages, TLB from OS Notes.

"INSTRUCTION LEVEL PARALLELISM"

① PIPELINING:

In CPU it allows multiple functional units such as fetching, decoding, execute, store to work on diff stages of multiple inst at a same time.
It speeds up the execution by overlapping the execution of inst.

↳ Pipelining operates at speed of slowest stage
↳ Delay causes the pipeline to stall.

② Multiple Issue:

It enhances processor performance by allowing multiple inst to be issued and executed simultaneously in a single clock cycle.
This is achieved by replicating functional units within the processor enabling it to handle more than one operation at a time.
If a processor has 2 adders then it can execute two addition operations at a time.

Date _____ Day _____

Types of Multiple Issue:

↳ Static Multiple Issue
- Inst are sheduled at compile time
- The compiler determines which inst can be executed in // and sheduler them accordingly.

↳ Dynamic Multiple Issue
- Inst are sheduled at runtime.
- The processor itself decides which instruction can be executed simultaneously, based on the current state of execution.
- processors using dynamic MI are called superscalar

- Speculation in Multiple Issue:
This includes making educated guess about certain outcomes and execute them in // with other inst.

↳ Compiler based Speculation: The compiler inserts extra code to check whether the speculation was correct and if not it performs necessary action.

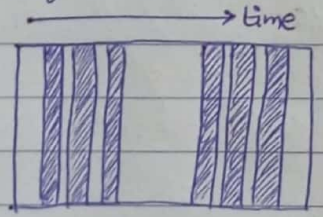
↳ Hardware based Speculation: The hardware stores speculative results in a buffer and if its correct it's comitted to register or memory. If the speculation is incorrect the results are discarded and re-executed.

Date _____ Day _____

"HARDWARE MULTITHREADING"

It is used to improve the performance of processors by allowing them to execute multiple threads of a program simultaneously.

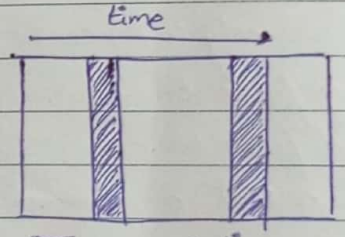
- Granularity:
  It is the ratio of computation to communication. Periods of computation are typically seperated from periods of communication by synchronization events.
- Fine-grain Parallelism: Relatively small amounts of computational work are done b/w communication events.
  - ↳ Tasks are broken down into very small units of computation.
  - ↳ Since the task are small there is frequent communication and syn b/w processors, leading to higher overhead in terms of managing the //ism.
  - ↳ Eg: Pipelining within CPUs, multithreading within loops.
  - ↳ Facilitates load balancing as it focuses more on distributing the tasks among all cores, it gives less opportunity for performance enhancement.



Instruction level //ism exploit this by executing multiple instr per clk cycle using techniques like superscalar execution or pipelining

Date _____ Day _____

- Coarse-Grain Parallelism: Relatively larger amount of computational work is done b/w communication & sync events.
  - ↳ Tasks are divided into larger units, such as whole function, process or independent computations.
  - ↳ Larges independent chunks of computation being processed in //.
  - ↳ Lower overhead compared to fine-grain bcz syn b/w task is less frequent.
  - ↳ Programs run for longer period of time.
  - ↳ Harder to load balance efficiently.



Multiprocessing where different process are assigned to seperate processors, distributed computing

- Thread-level //ism:
  Attempts to provid //ism thru simultaneous executio of diff threads, it provides a coarser-grained //ism than ILP. Threads are larger or coarser than the the finer-grained units.

Date _____ Day _____

▷ Hardware Multithreading:

Provides a means for system to continue doing useful work when the task being currently executed has stalled - for example, if the current task has to wait for data to be loaded from memory.

For this system must support very rapid switching blw threads.

↳ Fine-grained Multithreading - the processor switches blw threads after each instr., skipping threads that are stalled. Longer threads may have to wait for longer period of time.

↳ Coarse-grained Multithreading - only switches the threads that are stalled waiting for a time-consuming operation to complete (eg: load from main memory)

↳ Simultaneous Multithreading - or SMT is a variation on fine-grained multi-g. It attempts to exploit superscalar processors by allowing multiple threads to make use of multiple functional units.

"Preffered threads" → threads that have many instr ready to execute.