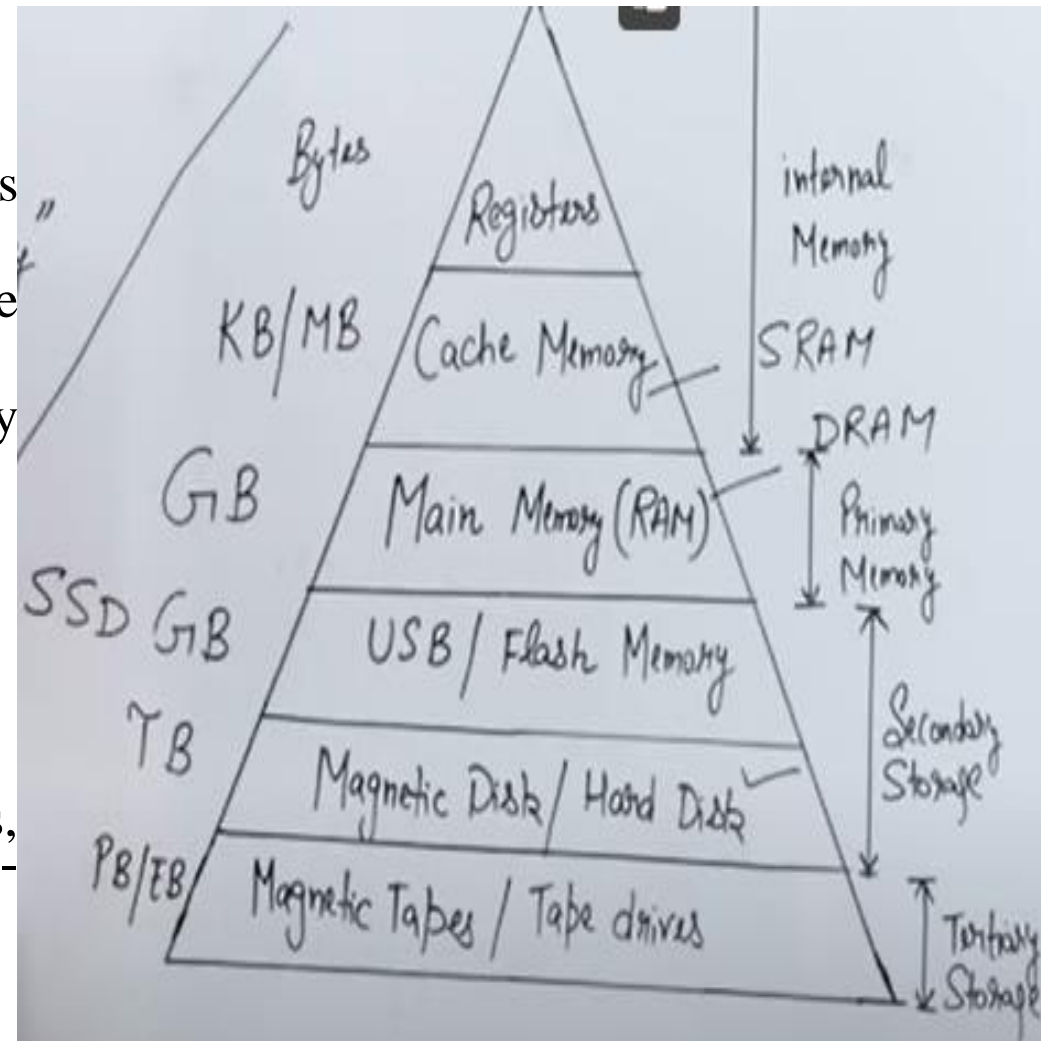


Unit 4

Memory Organization

Memory Hierarchy

- Memory organisation is done based on access time and storage capacity.
- We have all levels of memory to balance between cost and access time.
- Start with register, size – bytes (16 bit). Directly interact with CPU.
- Internal memory – directly interact with CPU.
- Back up – Secondary and tertiary.
- Cost – Register to tertiary
- Access time – register to tertiary.
- Speed – tertiary – 100 ms, secondary- 10 ms, SSD- micro second, Primary – 100 ns, cache- 10 ns, register- 1 ns.
- Frequency- register to tertiary.

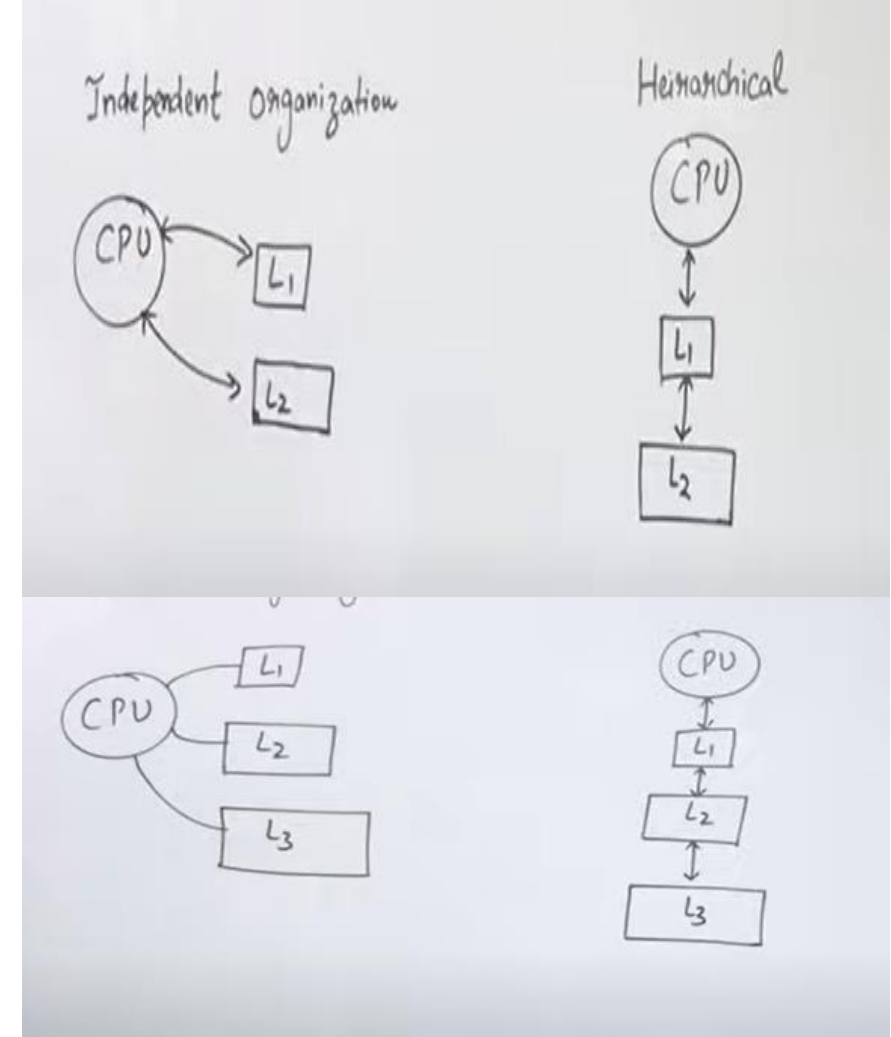


Two level memory hierarchy

- Different types of storage is considered to trade between cost and access time.
- How to set level- computer organization(how to component connect), architecture- efficiency.
- Two types
 - Independent
 - Hierarchical
- **In Independent**, both level directly interacted with CPU (parallelly and independently). L1- Cache, L2- Main or L1- Main L2- Secondary.

Assumption: $\text{size}(L1) < \text{size}(L2)$, $A.T(L1) > A.T(L2)$.

- Advantage: Parallel access(L1,L2) so minimum search time.
- If something is not in L1 then it will definitely in L2.
- $T = H1 * AT1 + (1-H1) * AT2$
- **In Hierarchical** , first search start at L1 , if not found search in L2.
- $T = H1 * AT1 + (1-H1) * (AT1 + AT2)$



3 level memory hierarchy

In Independent

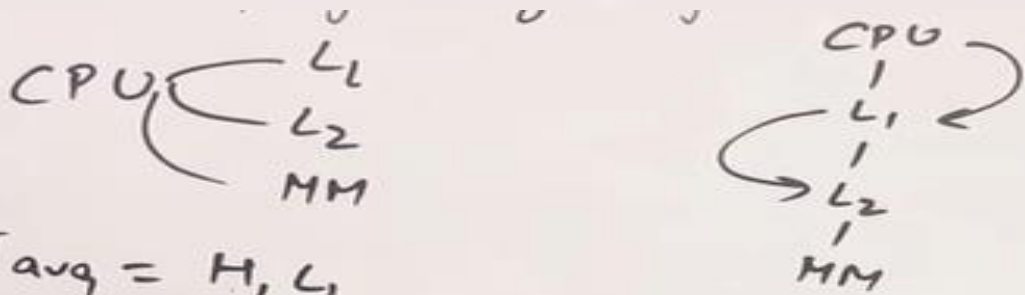
$$T = H1 * AT1 + (1-H1) * AT2 + (1-H1) * (1-H2) * AT3$$

In hierarchical

$$T = H1 * AT1 + (1-H1) * (AT1 + AT2) + (1-H1) * (1-H2) * (AT1 + AT2 + AT3)$$

Consider a system with 2-level caches. Access times of L_1 and L_2 and main memory are 1 ns, 10 ns and 500 ns. Hit ratio of L_1 and L_2 are .8 and .9. What is Avg. access time of system ignoring search time within cache?

- A) 13.0 ns
- B) 12.8 ns
- C) 12.6 ns
- D) 12.4 ns



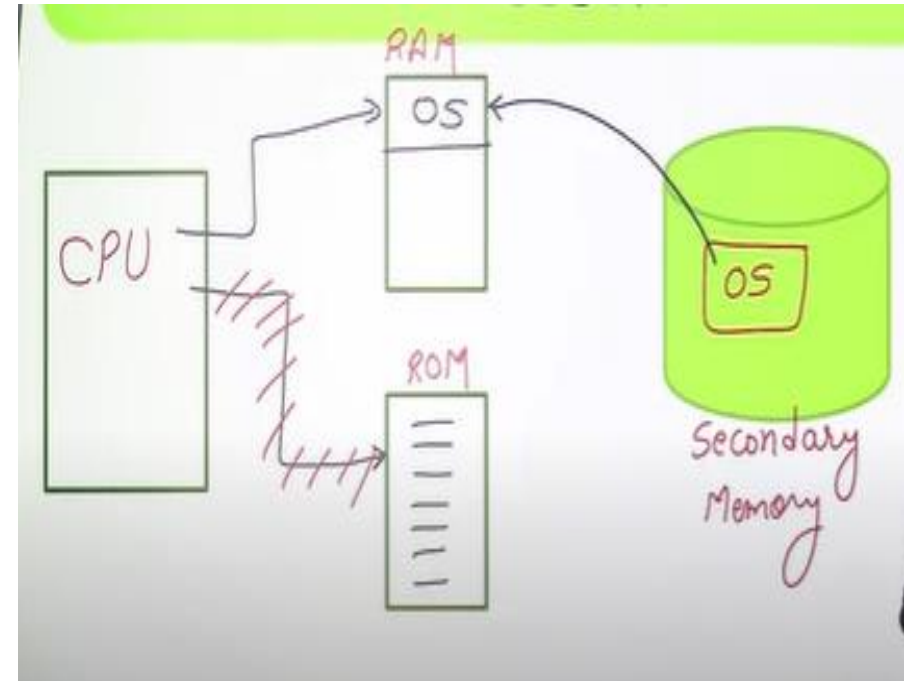
The diagram illustrates a 2-level cache system. On the left, a CPU is connected to a vertical stack of components: L_1 , L_2 , and MM (Main Memory). On the right, a similar stack is shown with a curved arrow indicating a hit in the L_1 cache, bypassing L_2 and MM. Below the diagrams, the average access time calculation is shown:

$$\begin{aligned}
 T_{avg} &= H_1 T_1 + (1-H_1) H_2 T_2 + (1-H_1)(1-H_2) T_3 \\
 &= .8 \times 1 + .2 \times .9 \times 10 + .2 \times .1 \times 500 \\
 &= .8 + 1.8 + 10 \\
 &= 12.6 \text{ ns.}
 \end{aligned}$$

Note : If in question nothing is given, then prefer independent hierarchy.

Main Memory(Primary)

- Programs reside in main memory and is brought to CPU for execution.
- CPU fetches and execute instruction from memory and this cycle repeats.
- Used to store current running program and their data.
- Basically it is of 2 types
 - RAM(Random Access Memory)- directly goes to address and fetch their content.
 - ROM(Read only memory)- also random but only enable read operation.
- RAM is volatile.(Content persist until power is there).
- ROM is non-volatile(Content persist even power is off).
- OS reside in RAM, and it is compulsory to run OS when system runs.
- So when power off , RAM content is flushed out(Even OS also).
- When PC turn on, CPU first access ROM.
- When Accessing ROM , it perform two operations
 - P.O.S.T(Power on Self Test)- checks which component(hardware) is connected to CPU.
 - Booting – load OS(secondary memory) to RAM.

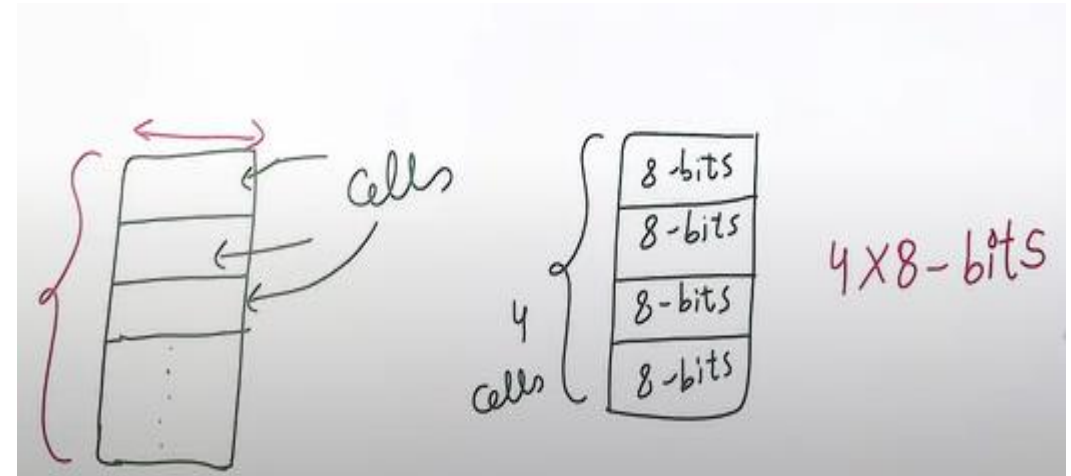


- A program residing in ROM responsible for booting operation is known as BootStrap.
- During Booting, you cannot interact with the system.
- After loading RAM, no further interaction with ROM is possible.
- RAM: to store running program.

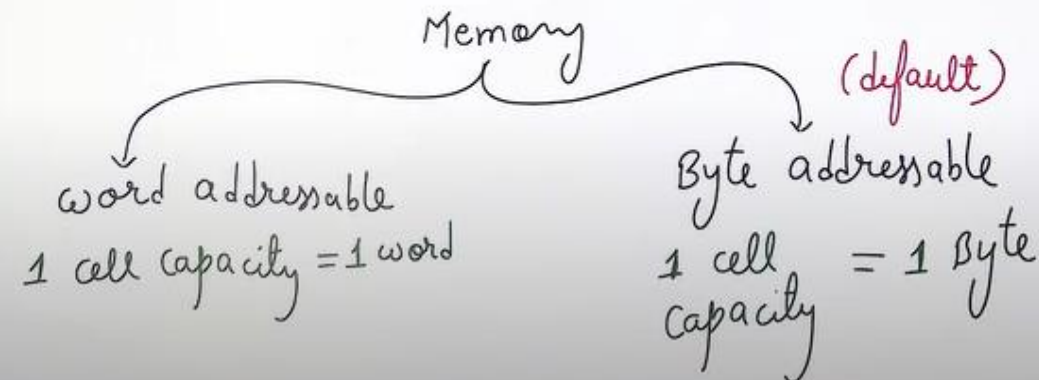
Static RAM(SRAM)	Dynamic RAM (DRAM)
Made of flip-flops	Made of capacitors Storage in form of electric charges
No refresh	Periodic refresh is required to keep content. (human brain)
Fast	Slow(during refresh no read/write)
Expensive	Less expensive
Used for cache	Used for main memory implementation.
Low idle power consumption	High idle power consumption
High operational power consumption	Low operational power consumption

Memory Representation

- Memory = Number of Memory Cells/Slot * one cell capacity (each cell is of same size)
= Number of memory locations * Bits per location
- 3 info- Number of cells, cell capacity, overall capacity.
- CPU doesn't need whole program. It only needs 1 byte or 1 word.
- Cell partition is made for accessing each cell uniquely, randomly and independently.
- So, each cell is assigned a number (address) (Home number).
- Address bit = $\log_2(\text{Number of slot})$



No. of cells	4	8	16	2^n	x
Address size	2-bits	3-bits	4-bits	n -bits	$\log_2 x - 1$



Byte Addressable

$$k = 2^{10}$$
$$M = 2^{20}$$
$$G = 2^{30}$$

- Each Memory location store 1 byte information.

Assume,

$$128K \times 8\text{-bits}$$

or

$$128K \times 1\text{ Byte}$$

or

$$128K\text{ Byte}$$

$$b \Rightarrow \text{bits}$$
$$B \Rightarrow \text{bytes}$$

Default unit of storage = Bits

A memory has 16-bits address bus. Then how many memory locations are there?

- a) 64K
- b) 65536
- c) 216
- d) All

A 32-bits wide memory has 24-bits addresses to be accessed. Maximum memory capacity is _____ MB?



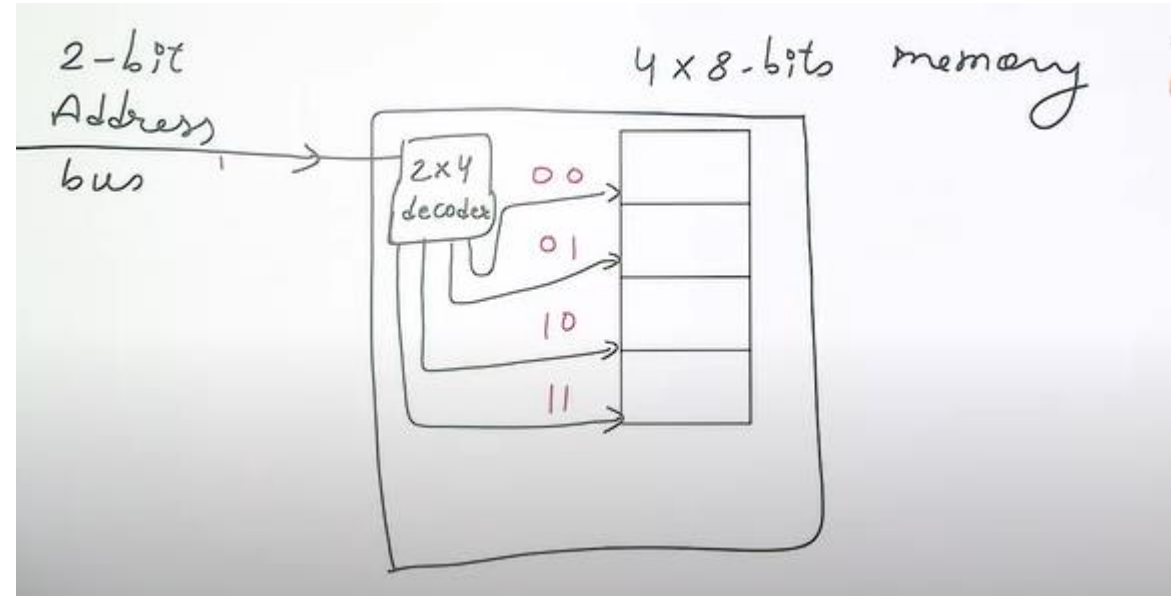
$$2^{24} \times 32\text{-bits} = 2^{24} \times 4B$$
$$= 2^{26} B$$
$$= 64MB$$

Memory is represented as?

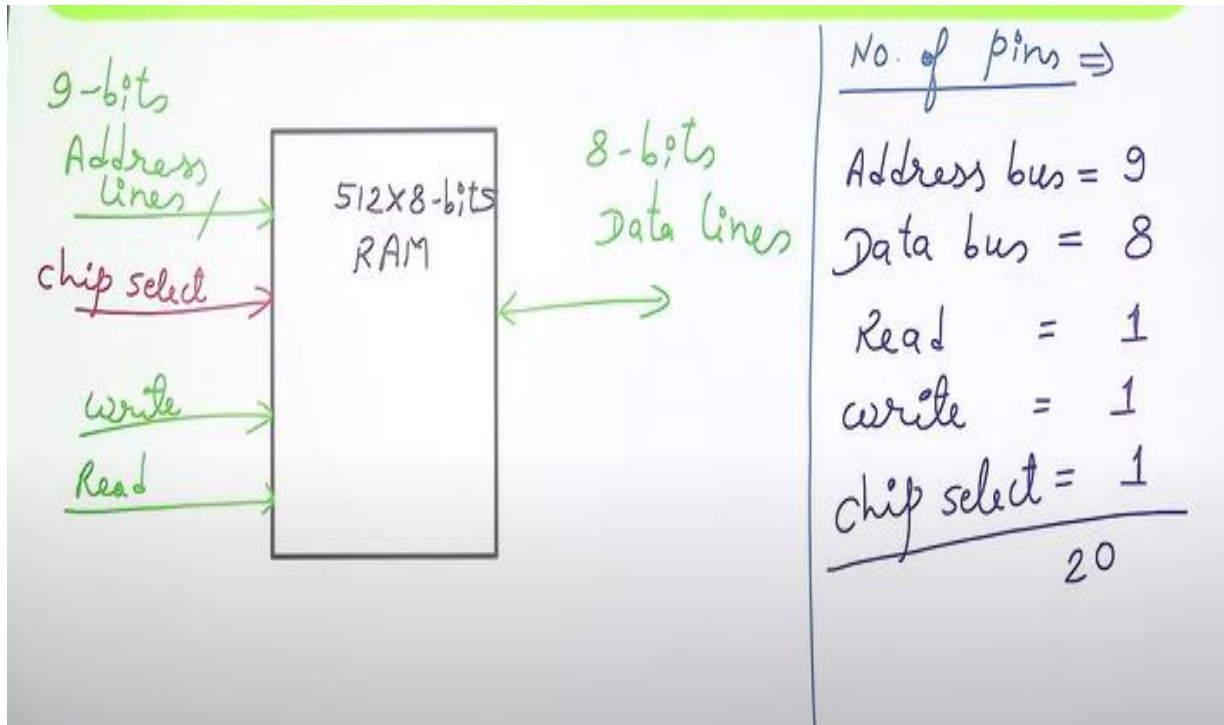
- a) $A \times B$ where A = No. of memory locations, B = No. of bits in each location
- b) $2^a \times B$ where a = No. of address bits, B = No. of bits in each location
- c) $B \times A$ where, B = No. of bits in each location, A = No. of memory locations
- d) (a) & (b) both

Memory access

- To memory access , we need decoder.
- Memory = 128k * 16 bits
= $17 * 2^{17}$ decoder



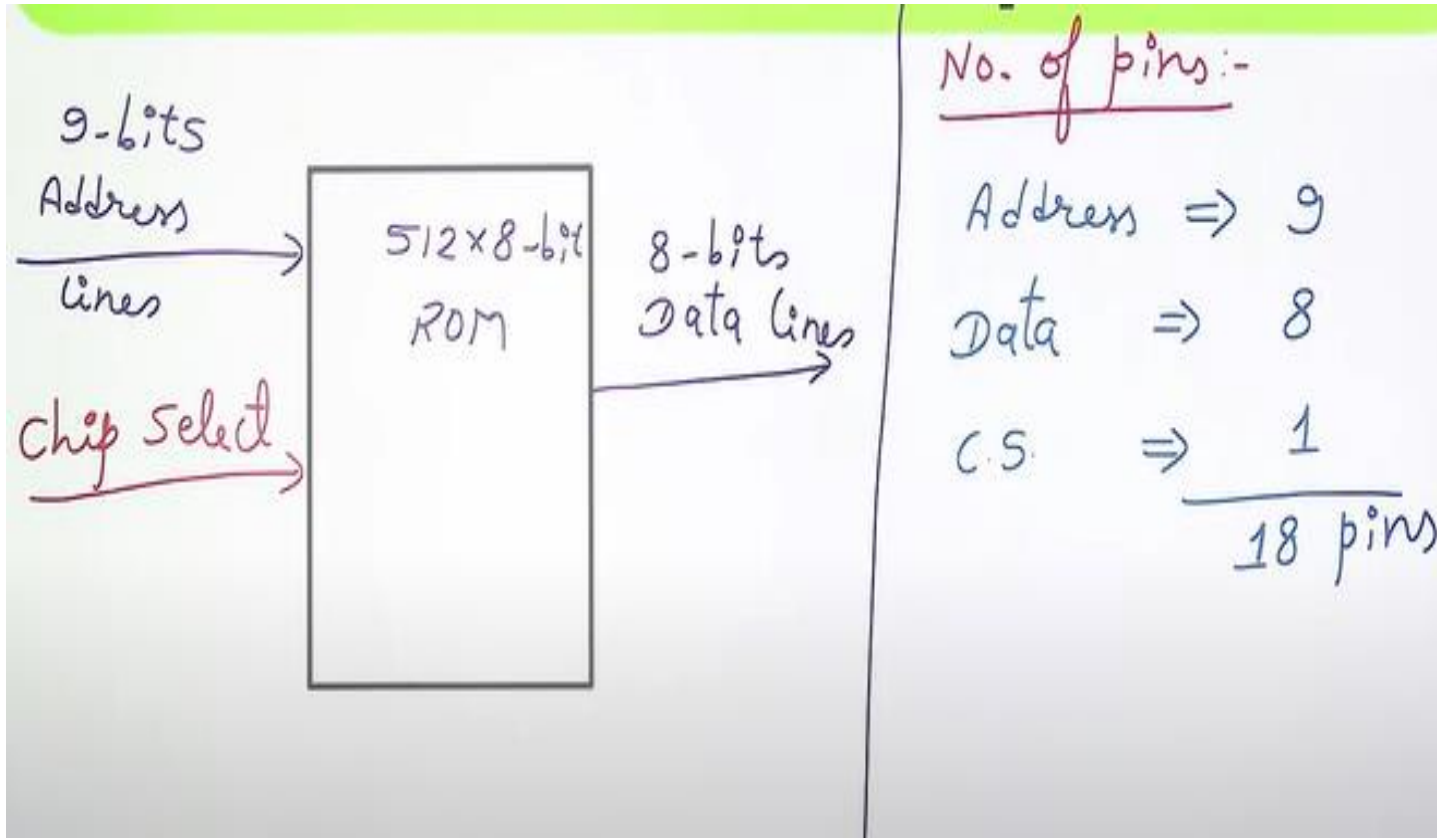
RAM Chip



CS	RD	WR	Operation
0	X	X	No operation
1	0	0	No operation
1	0	1	write operation
1	1	X	Read operation

- Write operation only creates conflict

ROM Chip

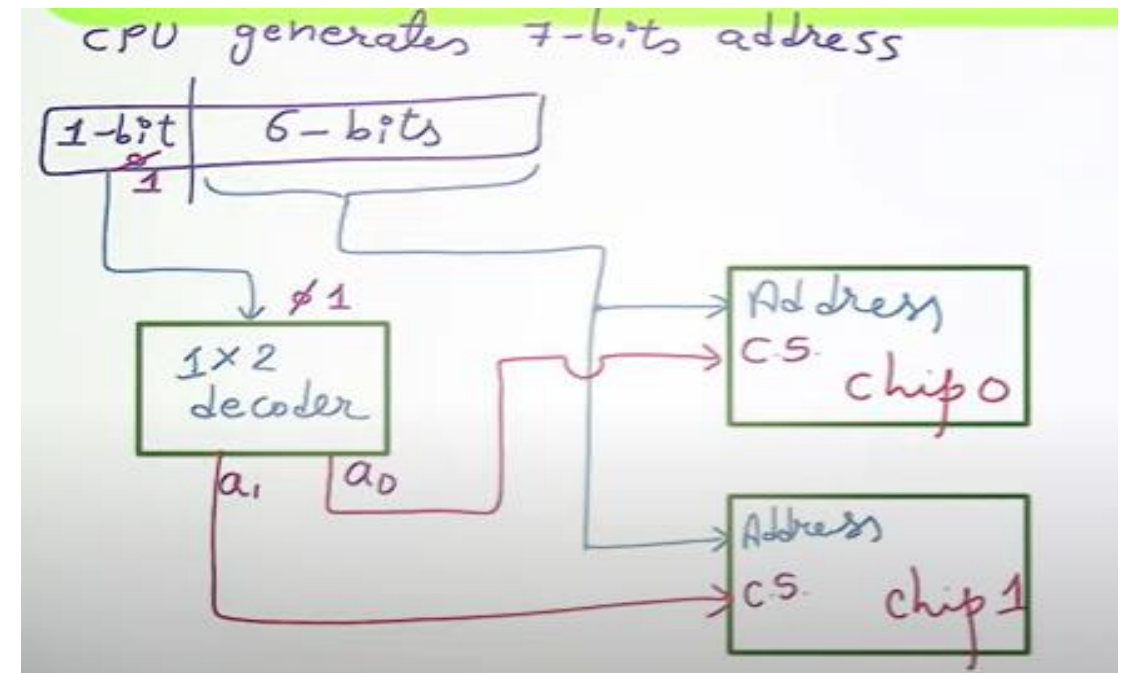
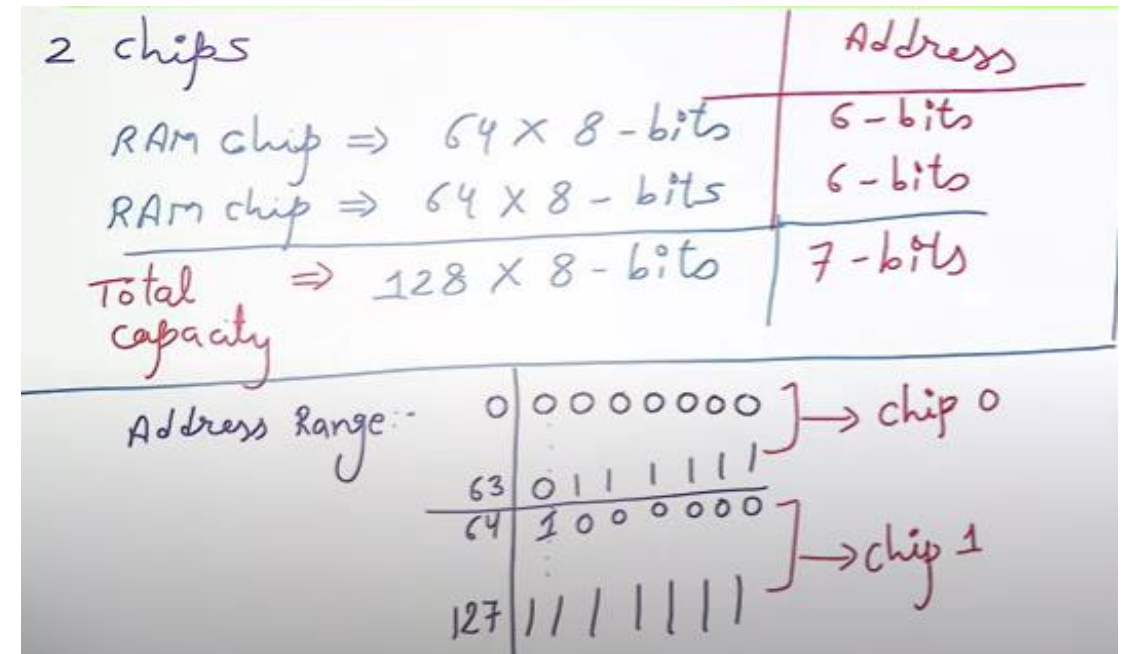


CS	Operation
0	No operation
1	Read

Multiple Chips in Single system

- Just like building house using bricks.
- Here we build higher capacity RAM using low sized RAM.
- E.g. 32 GB RAM = 8*4 GB RAM.
- **Total capacity = Number of Chips * Capacity of each chip.**
- Vertical Arrangement

used when no. of addresses required in memory system is more than in single chip.



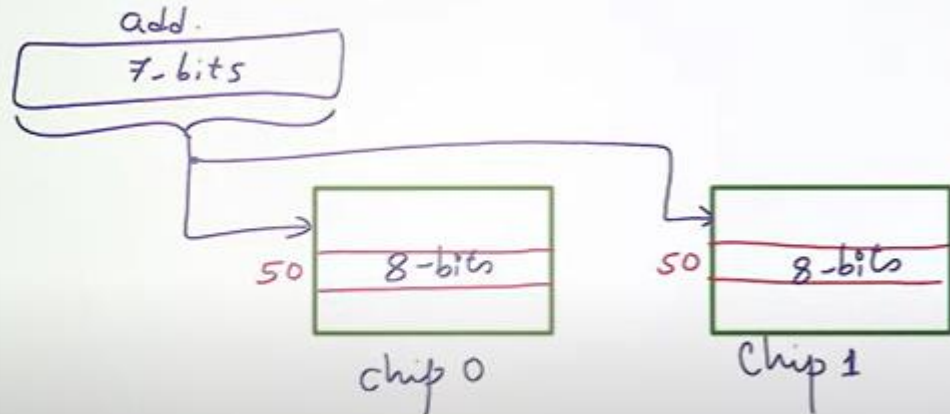
- Horizontal

(Munna Bhai MBBS)

128*8 RAM, MM- 128*16

used when data per location is required more than one chip data.

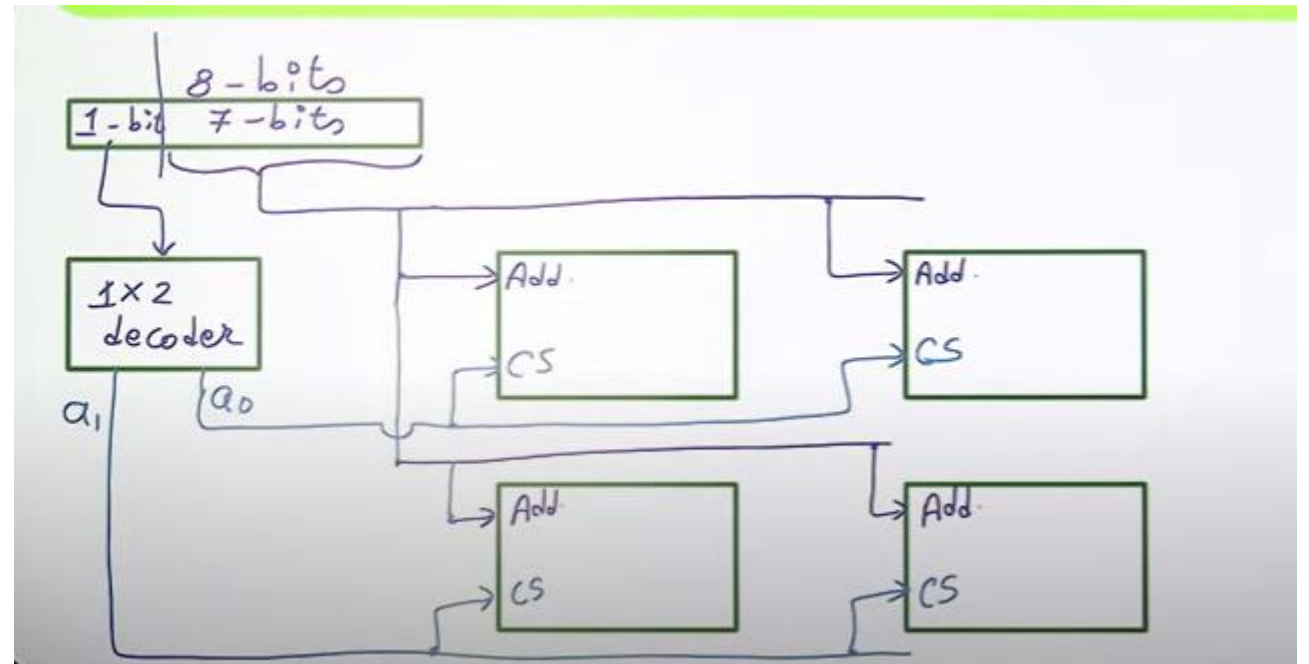
CPU generates 7-bits add.



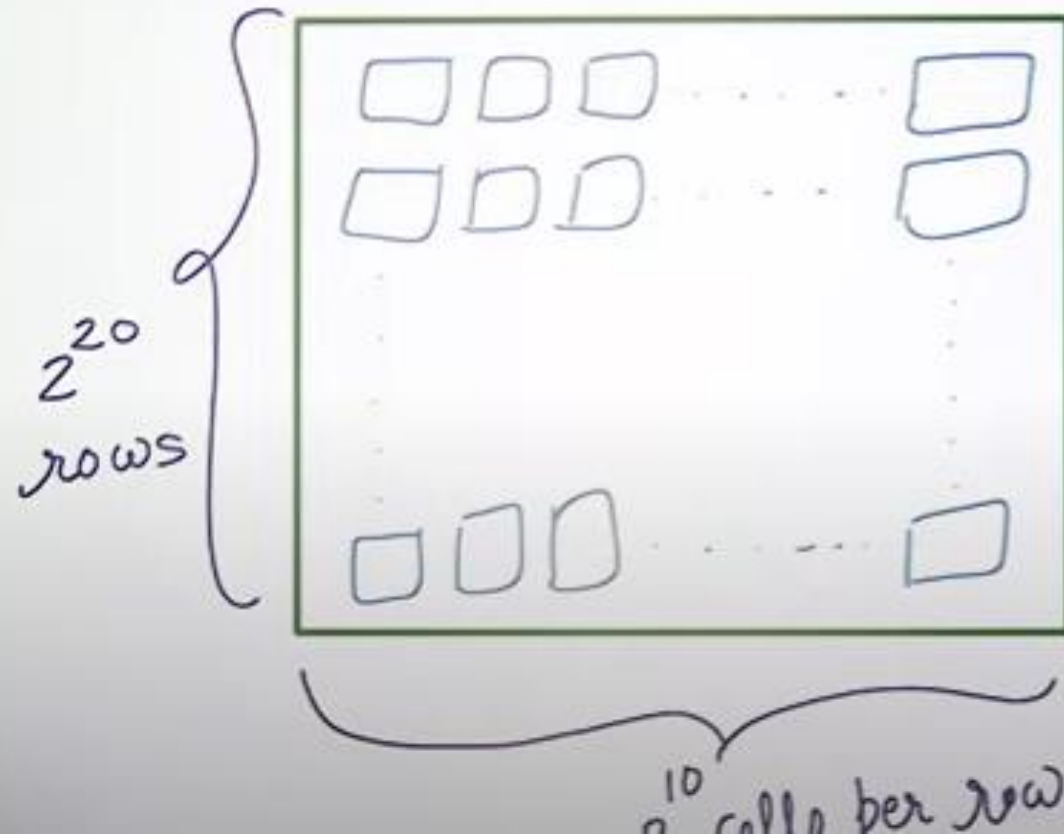
Hybrid arrangement

Used when both address and data per location is more than single chip

Eg 128*8 RAM size, 256 *16 bit MM



DRAM



Example:-

$$\text{DRAM} = 1\text{G} \times 1\text{B}$$

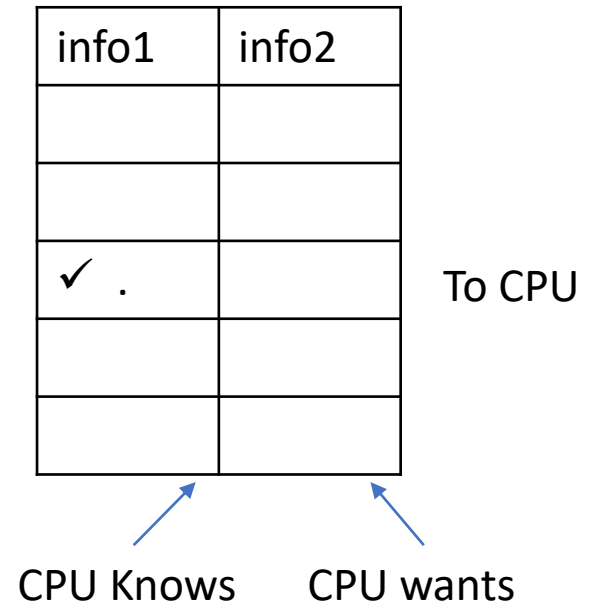
Note :-

In one refresh operation one row of cells can be refreshed.

Associative Memory

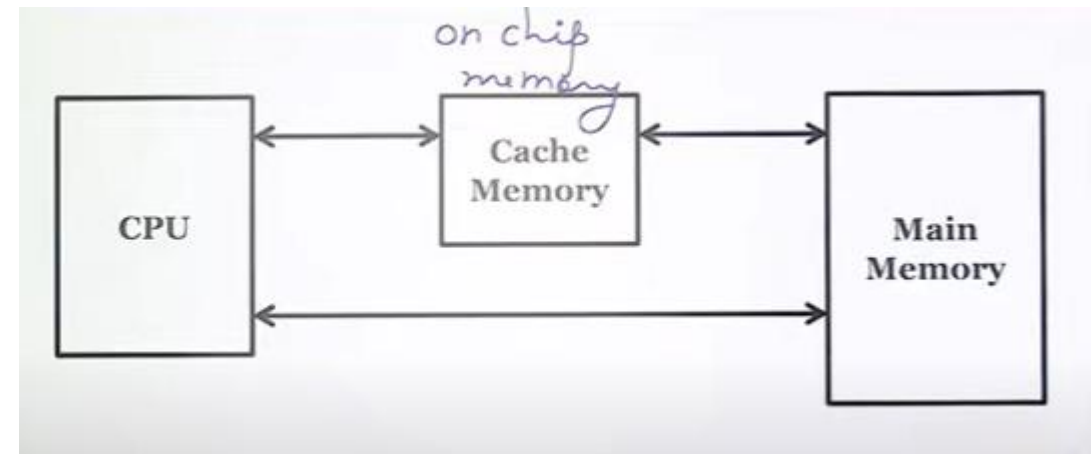
Address

- Known as Content Addressable memory.
- Addressable – cell has number, CPU generates it to fetch instructions.
- Cell don't have address.
- Searching ?- through content inside memory.
- E.g. Searching a person whose we don't know address or name(village).
- But one info – He is connected to someone(Rahul).
- Each cell contain two info.
- We designed cell in such way that it contain unique info.
- Here search info is matched parallely. Thus, require hardware to compare , making it fast but costly.
- Faster than SRAM
- Used to implement TLB(Translation lookaside Buffer) or cache memory.



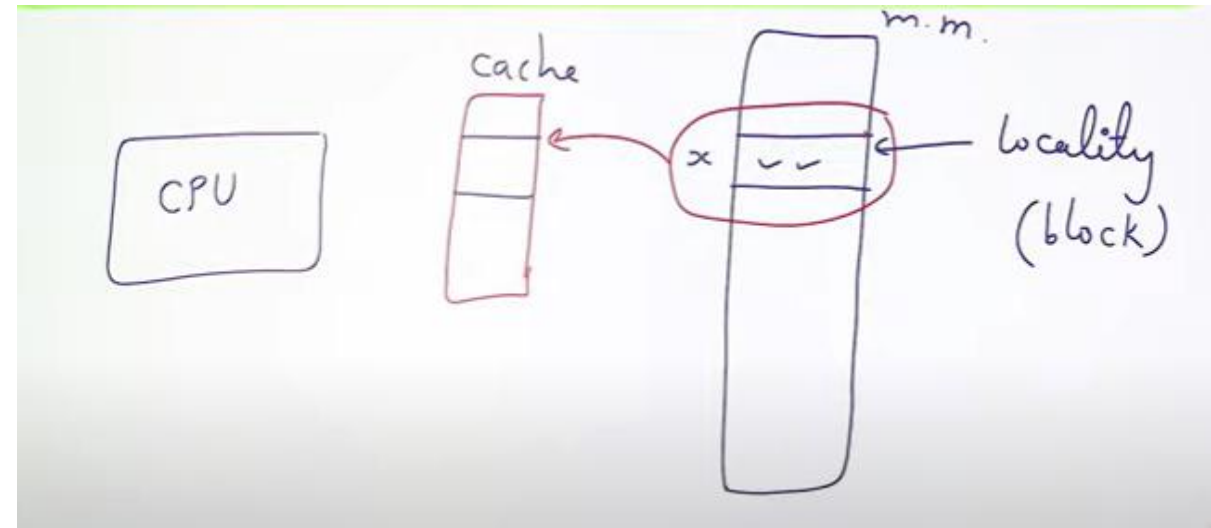
Locality of Reference

- Locality – neighbors, Reference – CPU refers main memory (Program reference).
- If CPU has requested one address for reference, then that particular address or nearby address will be access soon.(Locality of Reference).
- E.g. question asking
- If refer same address after time – temporal (based on time), if nearby add.-spatial(based on space).
- CPU access small chunks of whole program.
- Based on LOR, current demanded localities are kept into fast and small memory known as cache memory.
- Use of cache improves memory access time.
- In case of miss, whole block is transferred to Cache(collection of content(byte or words).



Working of Cache

- Cache Hit- CPU demanded content is present in the cache.
- Cache miss- CPU demanded content is not present in the cache.
- High Cache hit for improve memory access time.
- Performance of cache is given using hit ratio.
- Hit ratio= Number of hits/ Total memory references.
- Hit ratio is 0-1. Hit ratio + Miss ratio = 1
- E.g. bday decoration using boy.
- Locality deals in terms of blocks.
- Prediction that in future nearby memory address will be accessed/required.
- In case of miss, CPU goes to main memory, access the required content and bring the block (containing demanded content) from main to cache.



General formula:-

$$T_{avg} = H * \text{Mem. access time for each hit} + (1-H) * \text{Mem. access time for each miss}$$

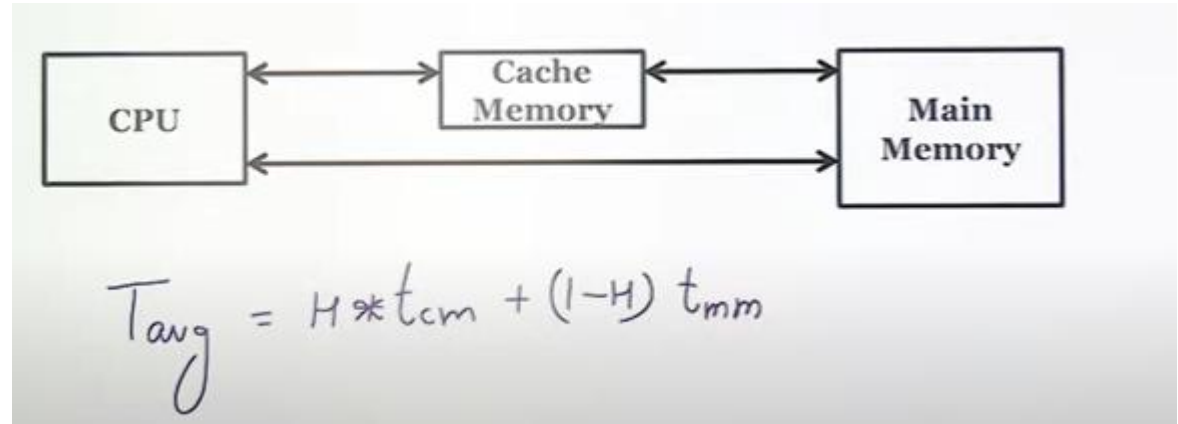
..... (1)

Example:- CPU referred memory 200 times
hit = 160 times
miss = 40 times

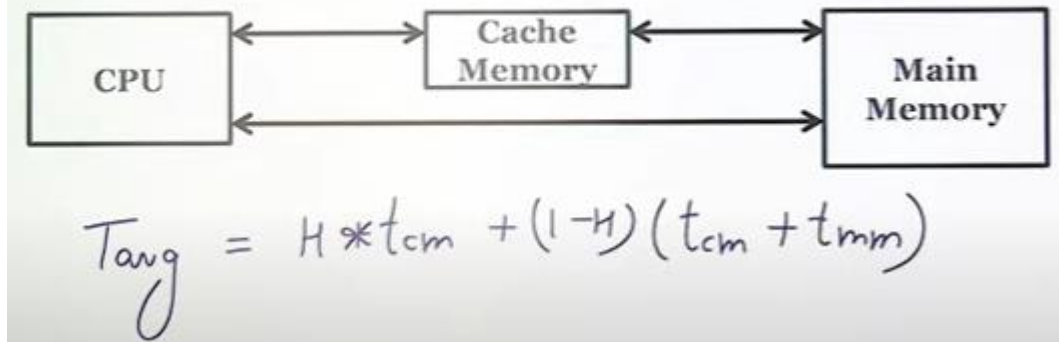
each hit requires = 10nsec
each miss requires = 100nsec

Type of cache access

- Simultaneous access: parallel access to both cache and main memory.
- Less access time as cache size is small , if miss parallelly accessing main memory.
- Hierarchical – first search in cache then search to main memory
- If in question explicit given cache and main access time , this formula.



2. Hierarchical Access (Serial)



When Block inclusion time included

Simultaneous $t_{avg} = H * t_{cm} + (1-H) (t_{block} + t_{cm})$

T_{block} = number of
block size * T_{mm}

Hierarchical:- $t_{avg} = H * t_{cm} + (1-H) (t_{cm} + t_{block} + t_{cm})$

Note: In one access of cache (T_{cm} time) , entire block can be accessed.

Assume that for a certain processor, a read request takes 50 nanoseconds on a cache miss and 10 nanoseconds on a cache hit. Suppose while running a program, it was observed that 90% of the processor's read requests result in a cache hit. The average read access time in nanoseconds is _____?

In a two-level hierarchy, if the top level has an access time of 30 ns and the bottom level has an access time of 150 ns, what is the hit rate on the top level required to give an average access time of 45ns?

$t_{cm} = 30 \text{ nsec}$
 $t_{mm} = 150 \text{ nsec}$
 $H = ?$
 $t_{avg} = 45 \text{ nsec}$

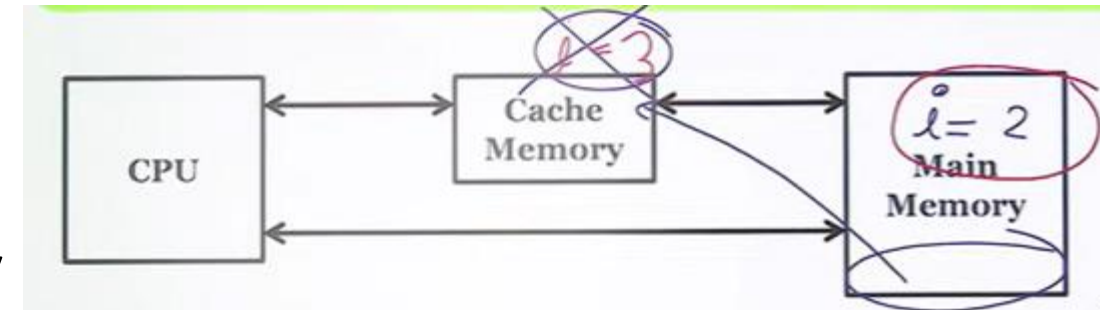
Hierarchical:-

$$45 = 30 + (1-H) 150$$

$$H = 0.9$$

Cache Write

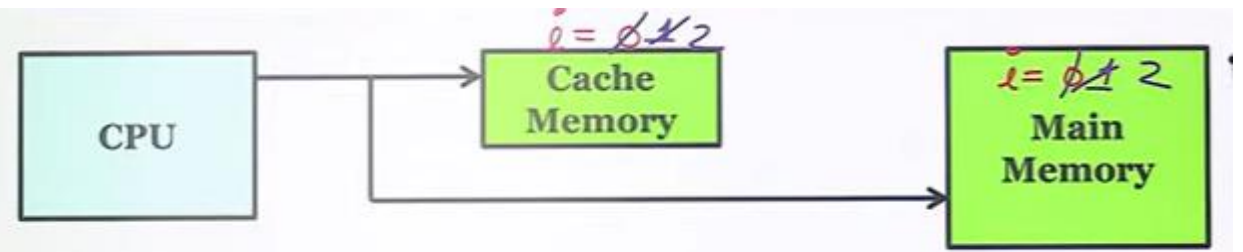
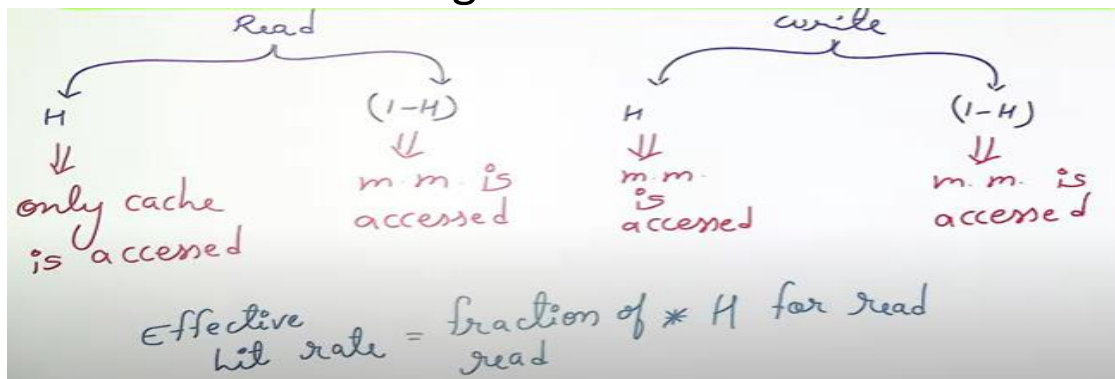
- Early we talk about cache read operation. Read operation does not create conflict.
- Write operation changes the content inside memory cell or block.
- If any change in cache, must update in main memory to data inconsistency.
- Since in cache we keep only few data, other locality will replace current content.
- Cache- copy of data, original data- MM.
- If CPU updates data in Cache, original data in MM should also be updated. (Write propagation).



- It is of 2 types:
 - Write Through
 - Write Back

Write Through

- Write update/operation is performed simultaneously to both cache and main memory.
- Advantage: Data consistency (cache and MM will have same value for content)
- Disadvantage: Time consuming as even though it is hit operation, we still have to access MM.
- $R_{avg} = H \cdot (T_{cm}) + (1-H) \cdot (T_{mm})$ (Read)
- $W_{avg} = \max(T_{cm}, T_{mm}) = T_{mm}$ (Write)
- $T_{avg} = \text{fraction of read} \cdot R_{avg} + \text{fraction of write} \cdot W_{avg}$



A system has a write through cache with access time of 100ns and hit ratio of 90%. The main memory access time is 500ns. The 70% of memory references are for read operations.

1. Average memory access time for read operations only
2. Average memory access time for write operations only
3. Average memory access time for read-write operations both
4. Effective Hit ratio

1. $t_{avg \text{ read}} = 0.9 * 100 \text{ nsec} + 0.1 * 500 \text{ nsec}$
2. $t_{avg \text{ write}} = \max(100, 500) = 500 \text{ nsec}$
3. $t_{avg} = 0.7 * 140 + 0.3 * 500 \text{ nsec}$
 $= 98 + 150$
 $= 248 \text{ nsec}$
4. Effective Hit rate = $0.7 * 0.9 = 0.63$

Write Back

- CPU updates only cache content to MM when block is replaced from cache.
- Advantage: Time saving
- Disadvantage: Data inconsistency.
- $R_{avg} / W_{avg} = H * T_{cm} + (1-H) * (T_{mm} + T_{writeback})$

Write allocate and No write allocate

For read operation

Cache miss, block is brought from MM to cache

Write allocate: for write miss operation, the block is copied to cache from MM.

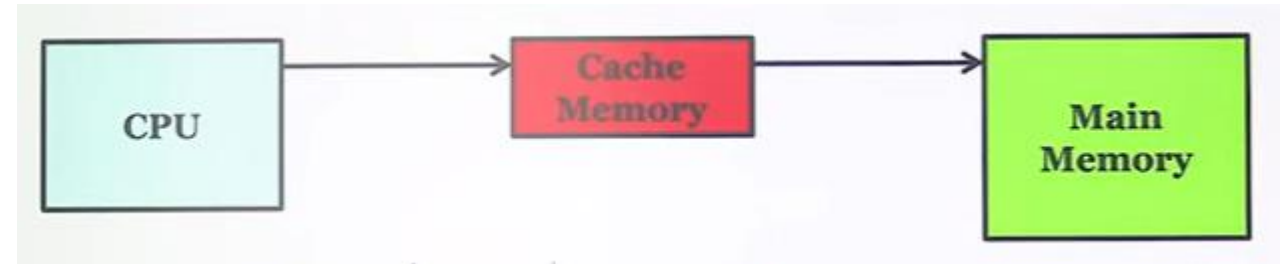
No-write allocate:

For write miss, the block is not copied instead direct update at the MM.

For Better performance:

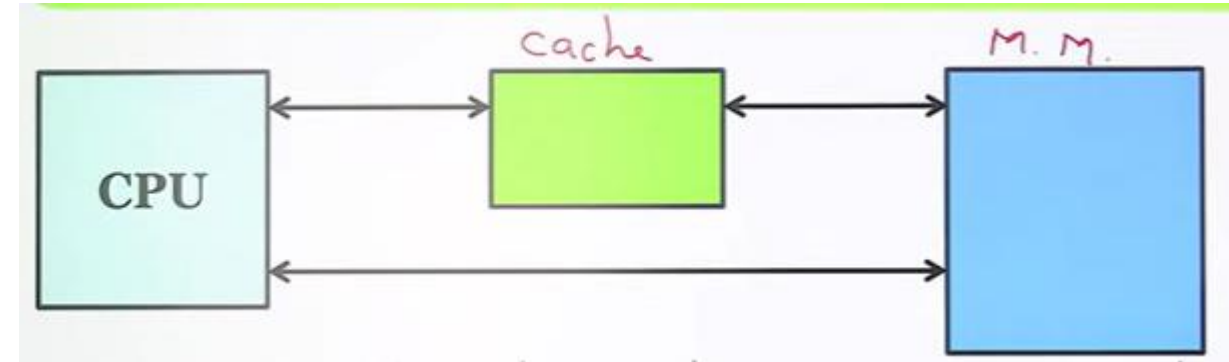
Write Through: Uses no write allocate

Write back: uses write allocate.



Cache mapping

- How to perform mapping of memory address to the cache memory.
- Since cache memory size is small, so less number of cache slot.
- Thus how CPU maps this memory address to cache address.
- Using MM address how cache memory is searched- is helped by the cache mapping.
- So we findout a pattern for mapping memory block to the cache
- Thus with help of pattern,
 - We can , map miss memory block to cache
 - Also it help is searching the cache block in future.
- Transforming of mm content to cache is known as cache mapping



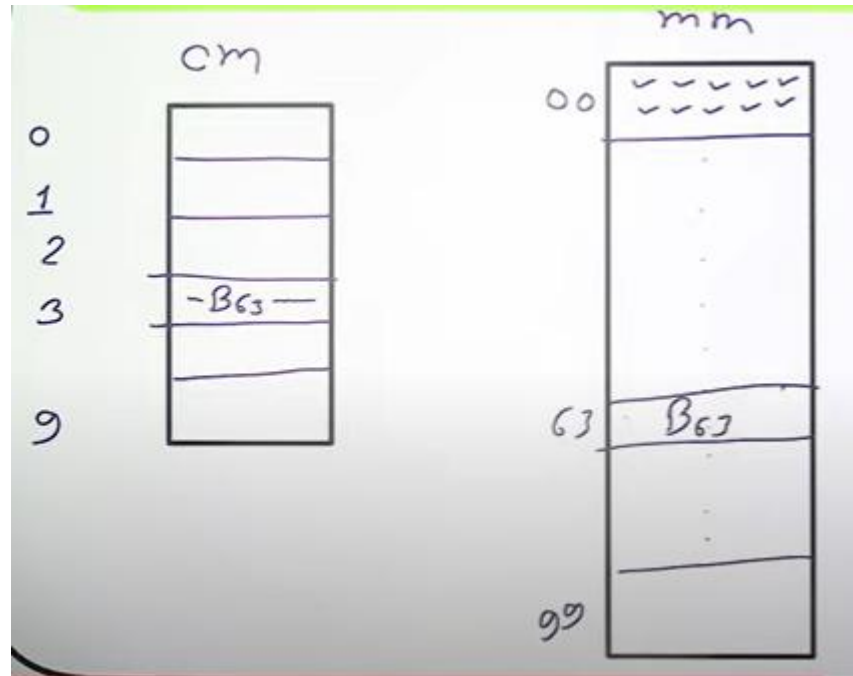
Generally 3 types

1. Direct Mapping
2. Set associative
3. Fully associative

Mapping is done on blocks.

Direct Mapping

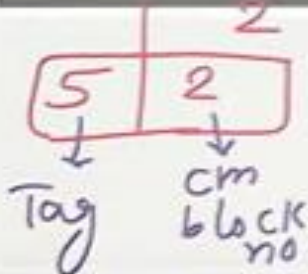
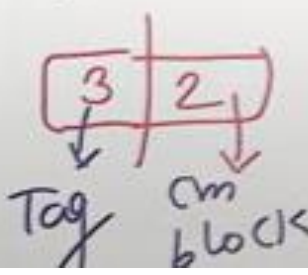
- X Blocks in cache = 10 (0-9)
- X Blocks in Main memory = 100 (00-99)



$$\text{cm block no.} = (\text{MM block no.}) \% (\text{no. of blocks in cm})$$

Cache	0	1	2	3	4	5	6	7	8	9
Main Memory	00	01	02	03	04	05	06	07	08	09
	10	11	12	13	14	15	16	17	18	19
	20	21	22	23	24	25	26	27	28	29
	30	31	32	33	34	35	36	37	38	39
	:	:	:	:	:	:	:	:	:	:
	90	91	92	93	94	95	96	97	98	99

CPU Request (MM block)	Mapping(CM block no.)	Hit / Miss	Comments
63	$63 \% 10 = 3$	MISS	Bring Block 63 from mm to cm on block no. 3.
93	$93 \% 10 = 3$	MISS	Bring block 93 from mm to cm on block no. 3, by replacing block 63.

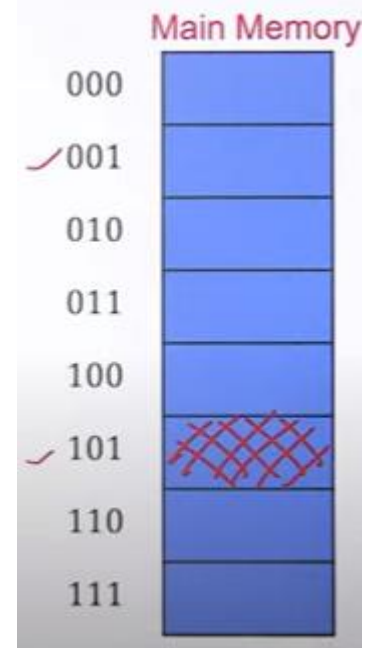
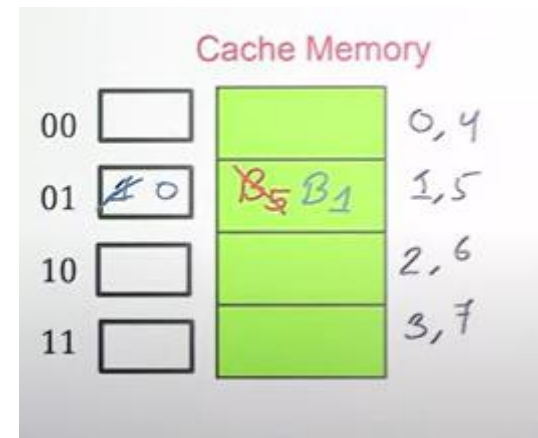
CPU Request (MM block)	Mappin (CM block no.)	Tag	Hit / Miss	Comments
52		5	Miss	Bring block no. 52 of mm into cache at block no. 2, & update tag as 5.
32		3	Miss	Bring block no. 32 of mm into cache at block no. 2, by replacing Block 52; and update tag by 3.

- To help identify block on cache no (whether block 63, or block 93), an identification bit is used. This identification bit is known as tag bit.
- Tag identifies which block is present in cache right now.

No. of tag infoⁿ stored = No. of blocks in cache (line)

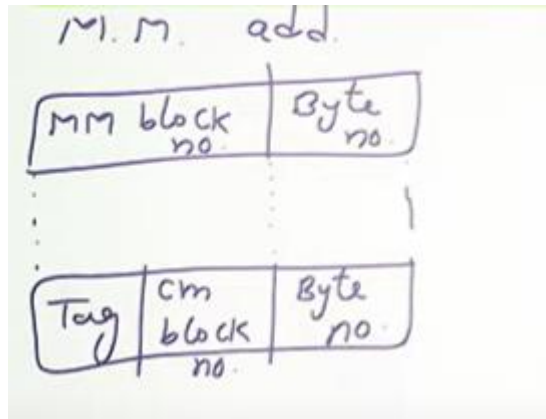
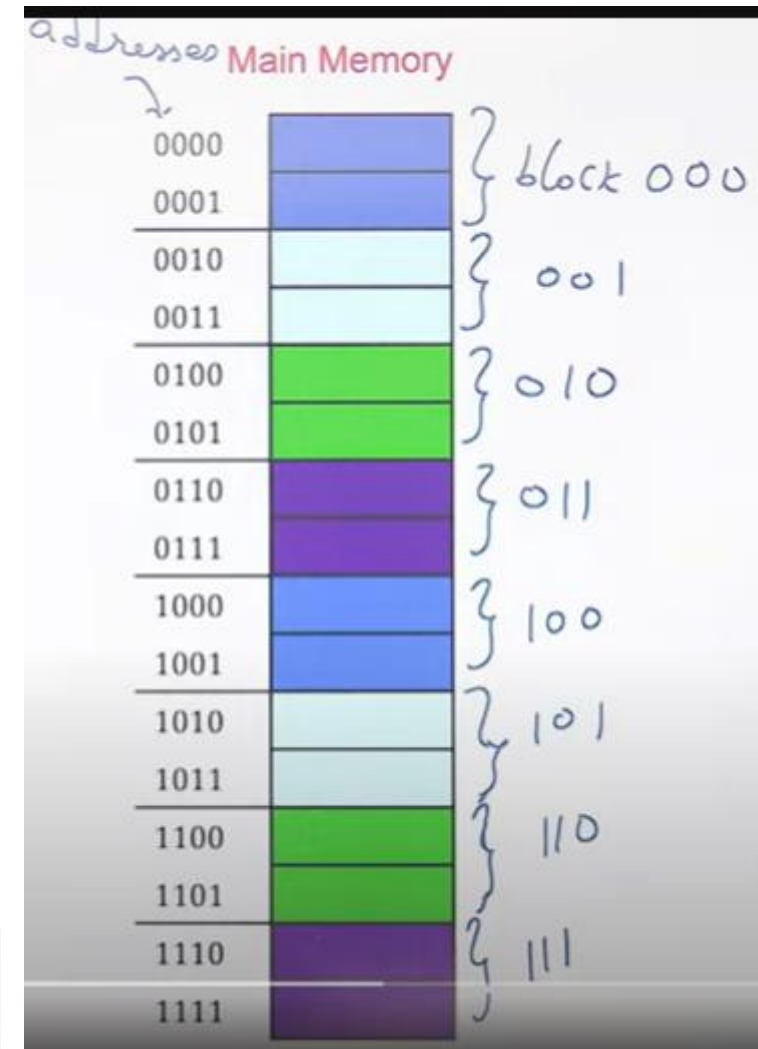
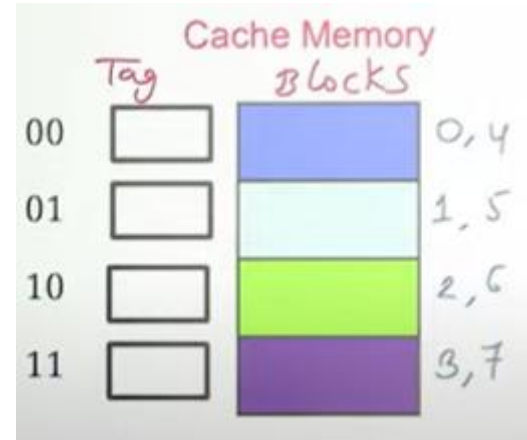
CPU Request (MM block)	Mapping(CM block no.)	Hit / Miss	Comments
63	$63 \% 10 = 3$	Miss	<u>Tag 6</u>
93	$93 \% 10 = 3$	Miss	Tag 9

CPU Request (MM block)	Mapping(CM block no.)	Hit / Miss	Comments
5 = 101	$5 \% 4 = 1$ <div style="border: 1px solid black; padding: 2px; display: inline-block;">1 01</div>	MISS	Bring mm block 5 (101) into cache at block no. 01 with tag 1.
1 = 001	<div style="border: 1px solid black; padding: 2px; display: inline-block;">0 01</div> <div style="margin-left: 10px;"> Tag ↓ ✓ cm block no. </div>	MISS	Bring mm block 1 (001) into cache at block no. 01, by replacing



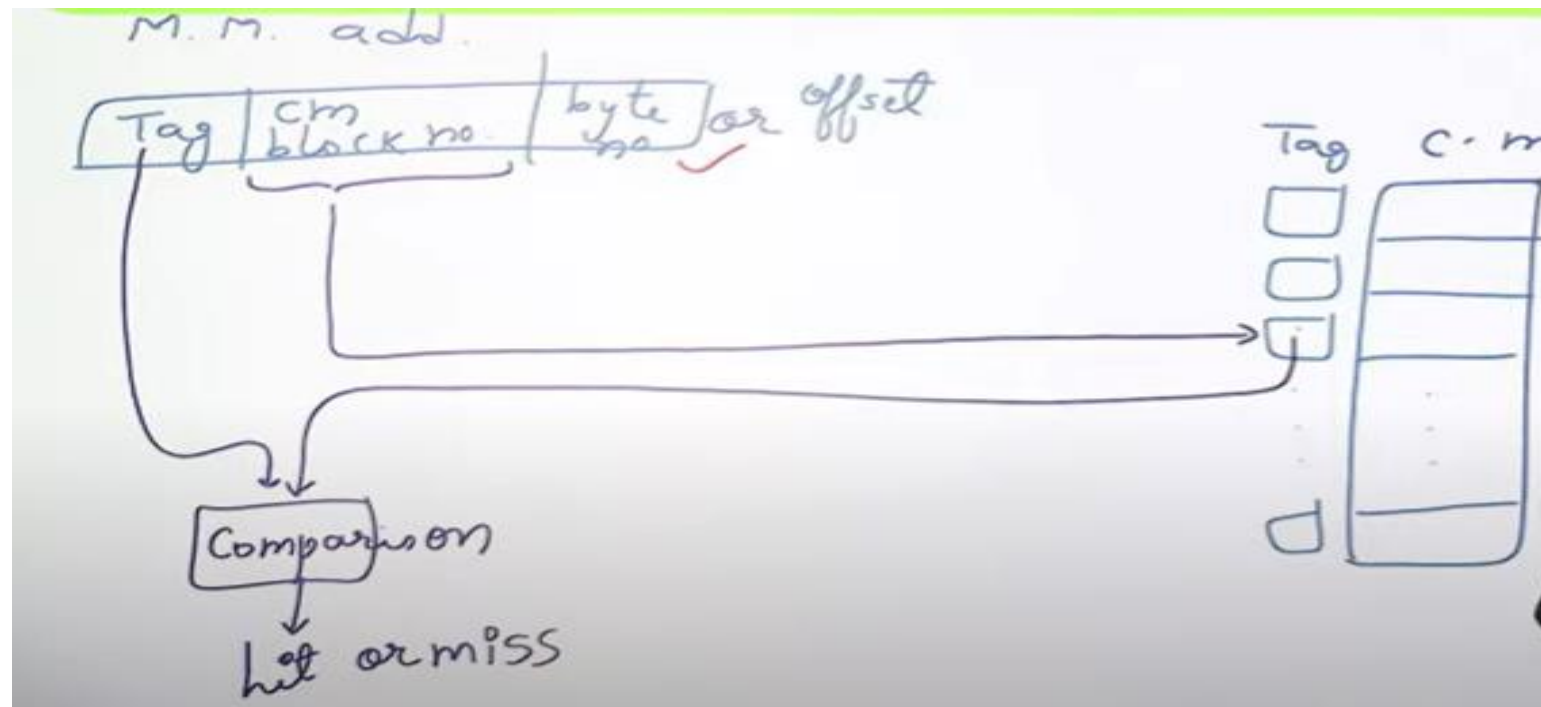
- X Blocks in cache = 4 (00-11)
- X Blocks in Main memory = 8 (000-111)
- X Block Size = 2 Bytes

- X Size of Cache memory = $4 * 2B = 8B$
- X Size of Main memory = $8 * 2B = 16B = 2^4 B$
- X Size of Main memory address = 4-bits



No. of bits in byte no. = $\log_2(\text{block size})$

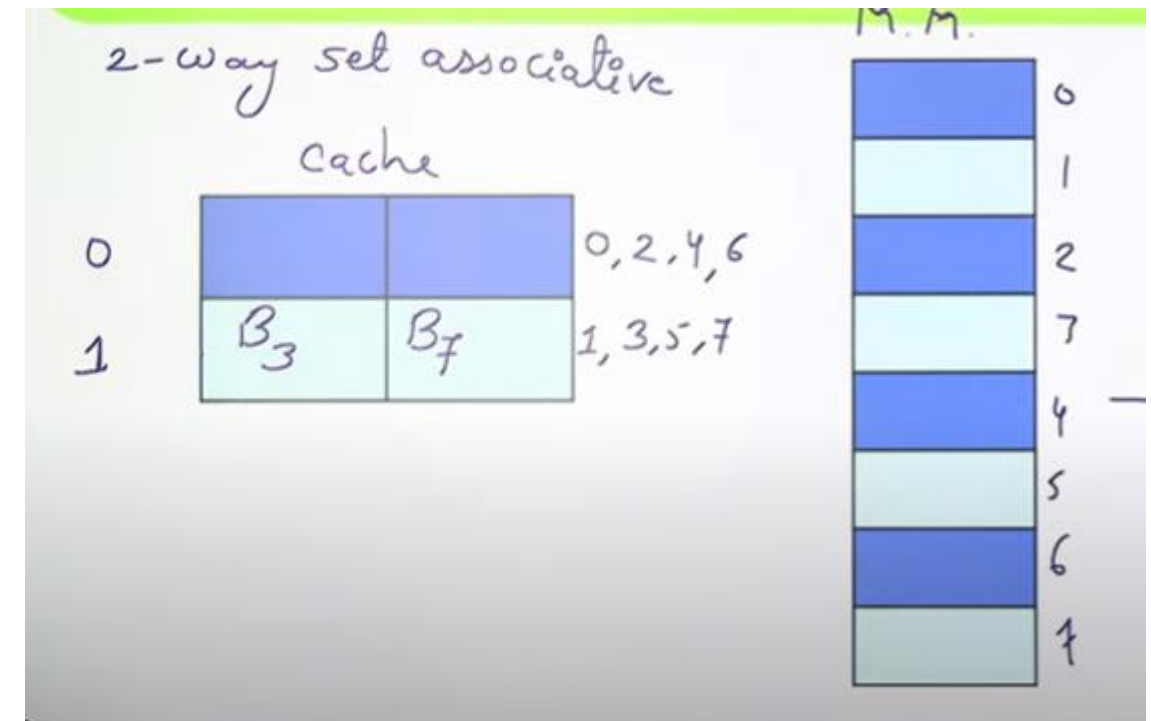
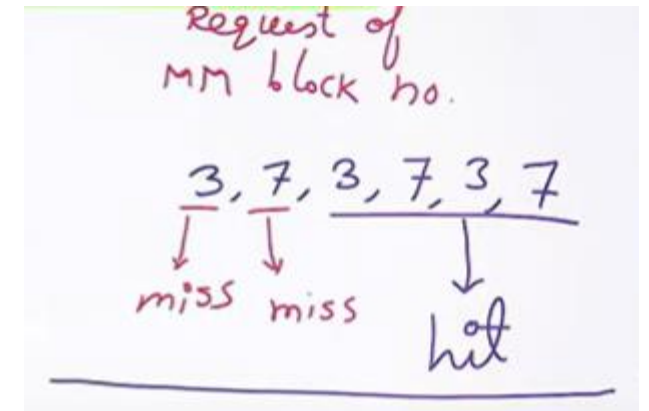
No. of bits in cm. block no. = $\log_2(\text{no. of blocks in cache})$

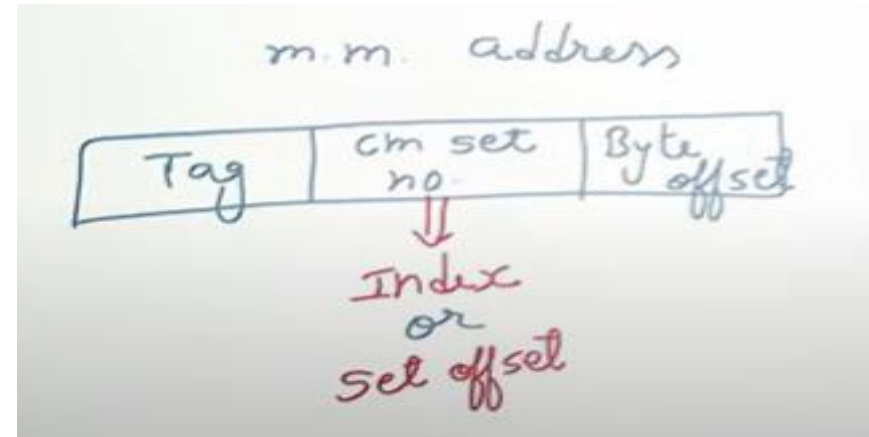
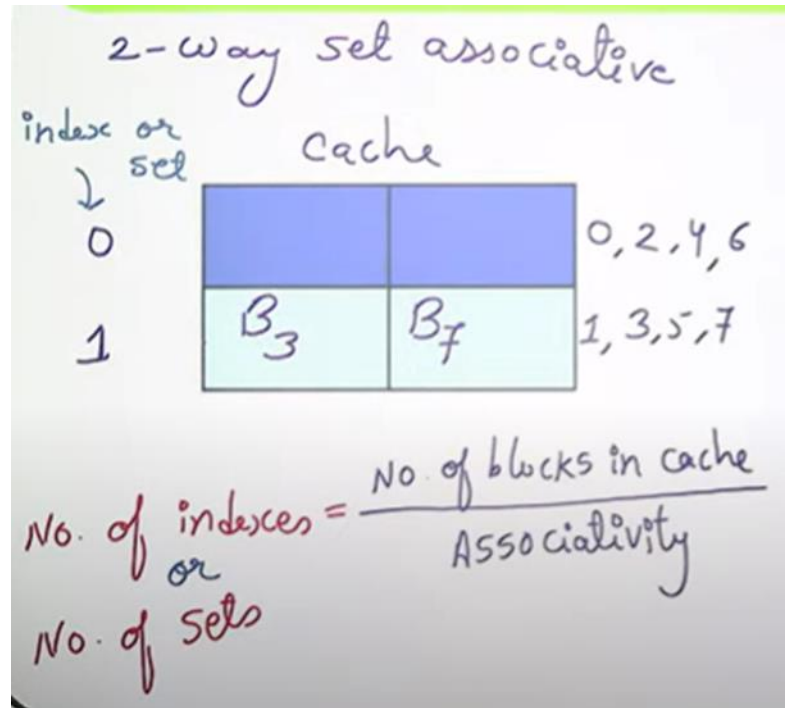


CPU Request (MM add.)	Mapping(CM block no.)	Hit / Miss	Comments
1010 <div> <div>101</div> <div>0</div> </div>	<div> <div>1</div> <div>01</div> <div>0</div> </div> <div> <div>Tag</div> <div>cm block no.</div> </div>	Miss	Bring block no. 5 from m.m. to cache block 01 with tag = 1.
1011 <div> <div>101</div> <div>1</div> </div>	<div> <div>1</div> <div>01</div> <div>1</div> </div> <div> <div>Tag</div> <div>cm block no.</div> </div>	Hit	send byte 1 of this block for CPU access.

Set Associative Mapping

- Since in direct mapping, only block can be placed in cache, block access 3,7,3,7,3,7 can result in too much miss ratio.
- Thus to avoid this issue one another mapping known as set associative mapping is proposed.
- Here we can keep more than one block in cache. The number of block in cache depends on the set size.
- If it is 2 set, then we can keep two block in one cache slot and so on.
- Mapping done is based on set while in case of direct mapping is done based on block or cache number.
- Cache set no = $(\text{mm block no}) \% (\text{no of sets in cache})$





No of tag= Number of blocks in each set

$$\text{No. of indexes} = \frac{\text{No. of blocks in cache}}{\text{Associativity}}$$

Fully associative mapping:

- Whole block here is placed in same set.
- Only tag bit is used to identify the block number.

