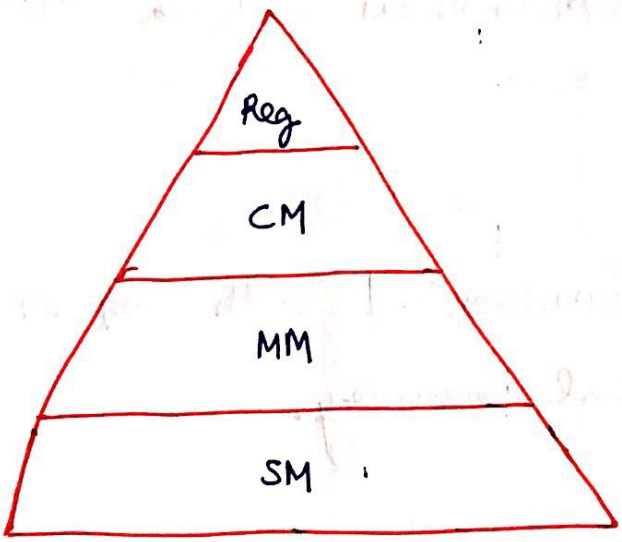
Memory hierarchy design :-

According to mem hierarchy design, system supported memory standards is as follows :-

- Level	:	1	2	3	4
- Name	:	Register	cache memory	main memory	secondary memory
- Typical size	:	< 1KB	< 16MB	< 16GB	> 100GB
- Implimentation	:	customized multiport CMOS	SRAM	DRAM	Magnetic
- Access time (ns)	:	(0.25 - 0.5)	(0.5 - 25)	(80 - 250)	(50,00,000)
- Bandwidth (MB/sec)	:	(20,000 - 1,00,000)	(5000, - 10,000)	(1000 - 5000)	(20 - 150)
- Managed by	:	Compiler	H/w	OS	OS
- Backed by	:	CM	MM	SM	Compact - Disk (CD)



Memory hierarchy design

Bottom – top approch of a memory hierarchy states:

Capacity ↓

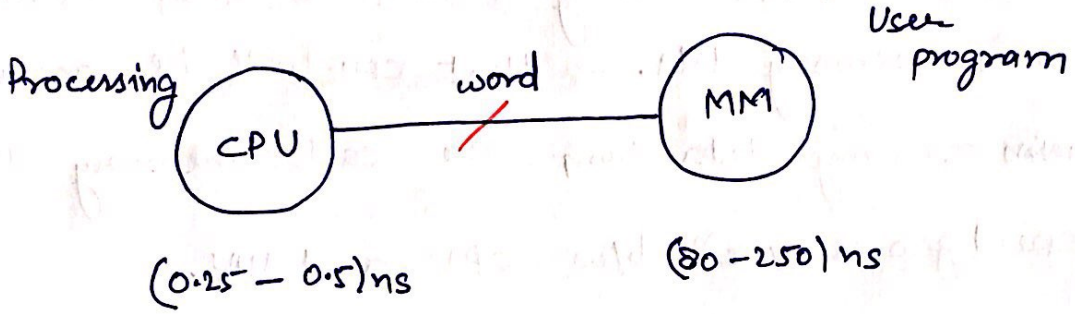
Access time ↓

Performance ↑

Cost/bit ↑

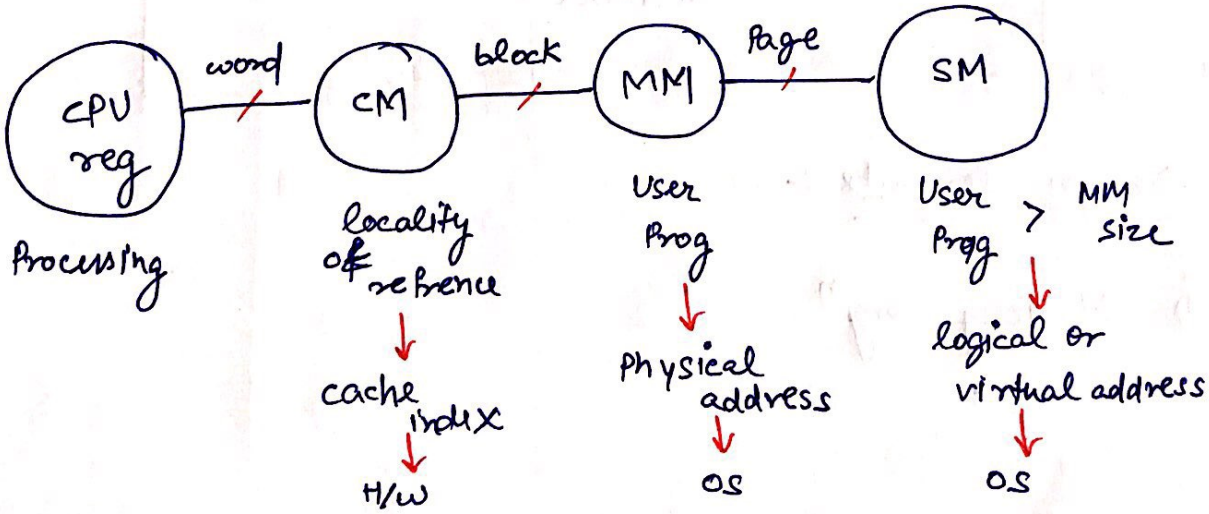
Expensive

Sys without hierarchy design :-



Speed gap ↑ b/w CPU and MM

Sys with hierarchy design :-

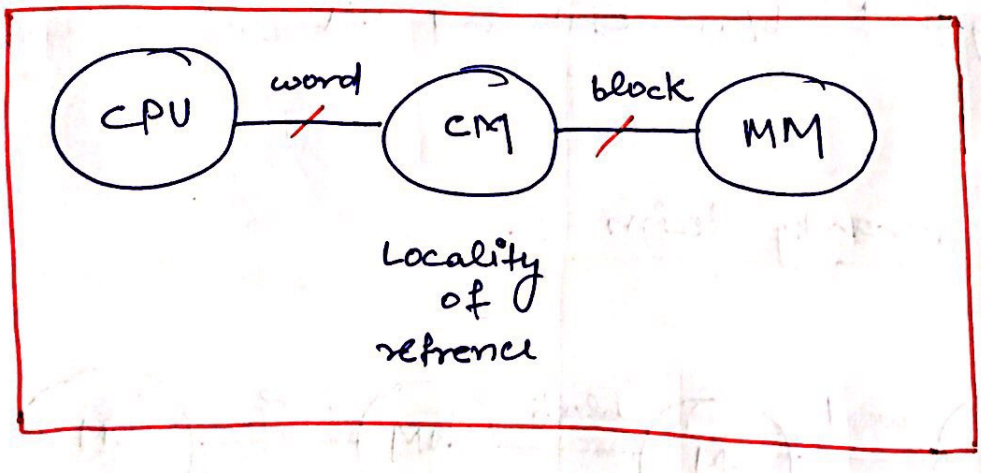


- Main memory speed is synchronizes with a CPU speed using cache memory.

- Main memory space is synchronized with any application program size using virtual memory.

Cache memory Design:-

- Cache memory is acts as an intermidiate memory b/w the CPU and the main memory, used to hold the image of a main memory data. So that CPU will be accessing the main memory data from the cache memory therefore speed gap is sync^{n} b/w CPU and MM.



Design elements :-

(i) Memory org^{n}

(ii) Mapping techniques

(iii) Replacement organization

(iv) Updating techniques

(v) Multi-level caches.

(i) Memory organization :-

- According to memory hierarchy design, data will be copied from the MM to CM in a form of "Blocks" so both of the memory are organized into a blocks based on the block-size.

- No. of block in MM = MM size / Block-size.

- No. of lines in CM = CM size / Block-size.

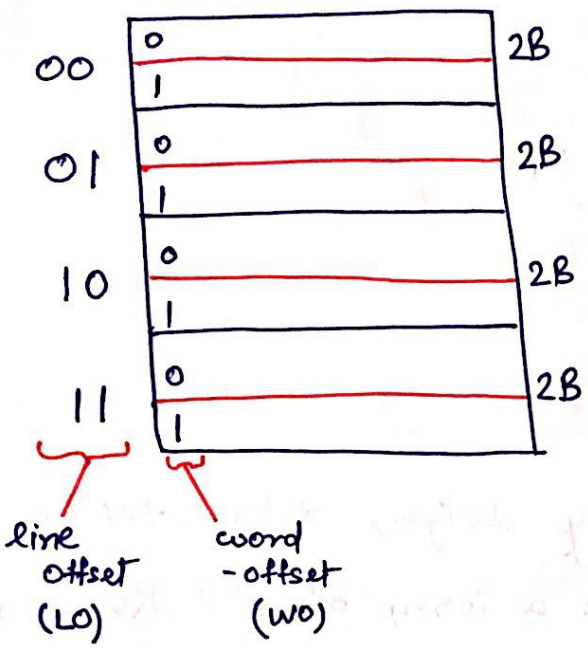
- Let us consider the system which contains 8-byte cache memory & 16-byte main memory with 2B-block

CPU ——— Cache memory (8B) ——— Main memory (16B)

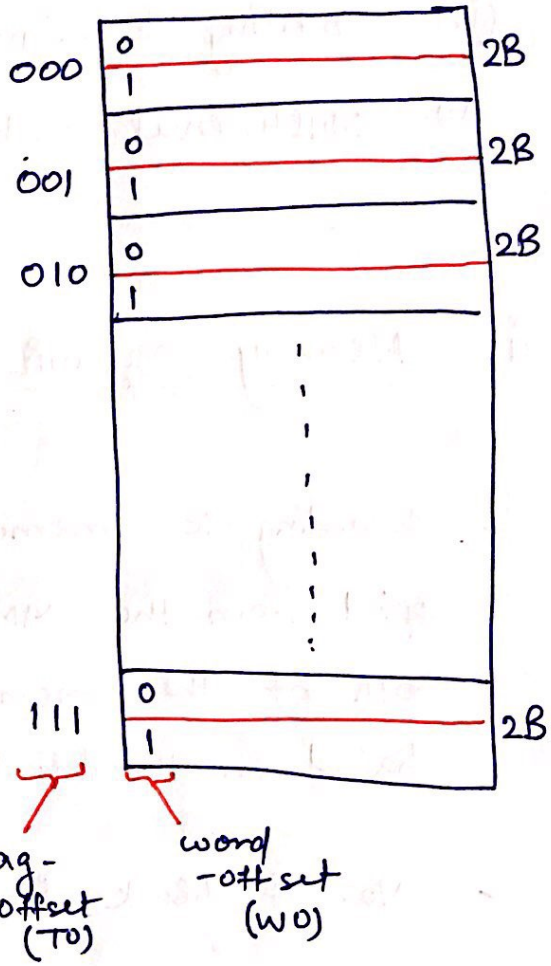
lines = $\frac{8}{2}$ = 4

blocks = $\frac{16}{2}$ = 8

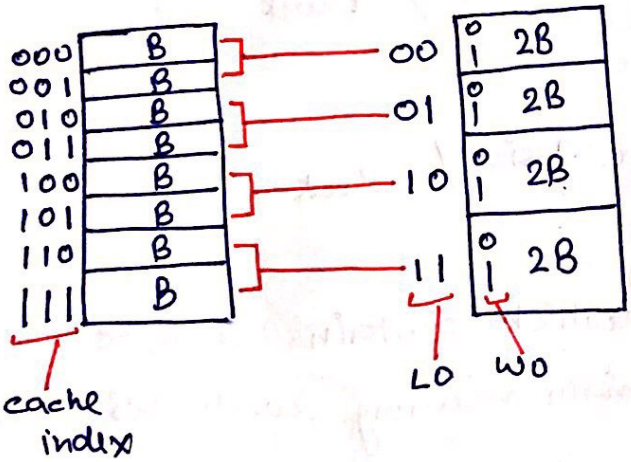
cache memory (8B)



Main memory (16B)



Before org^n After org^n

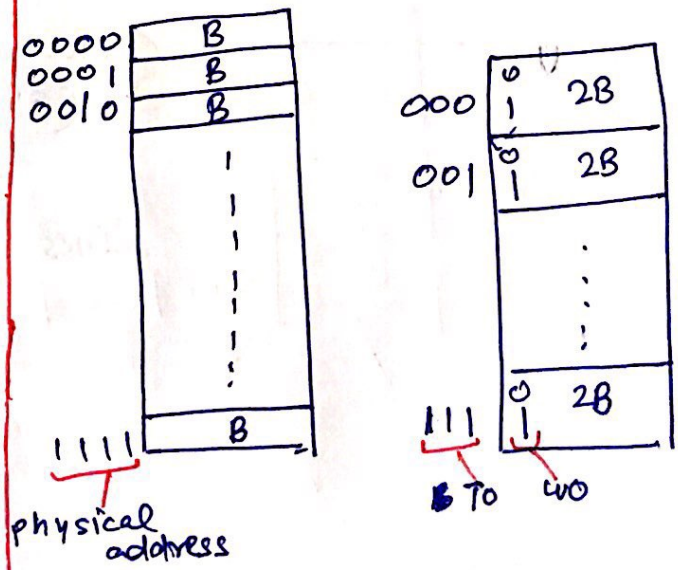


Before org^n

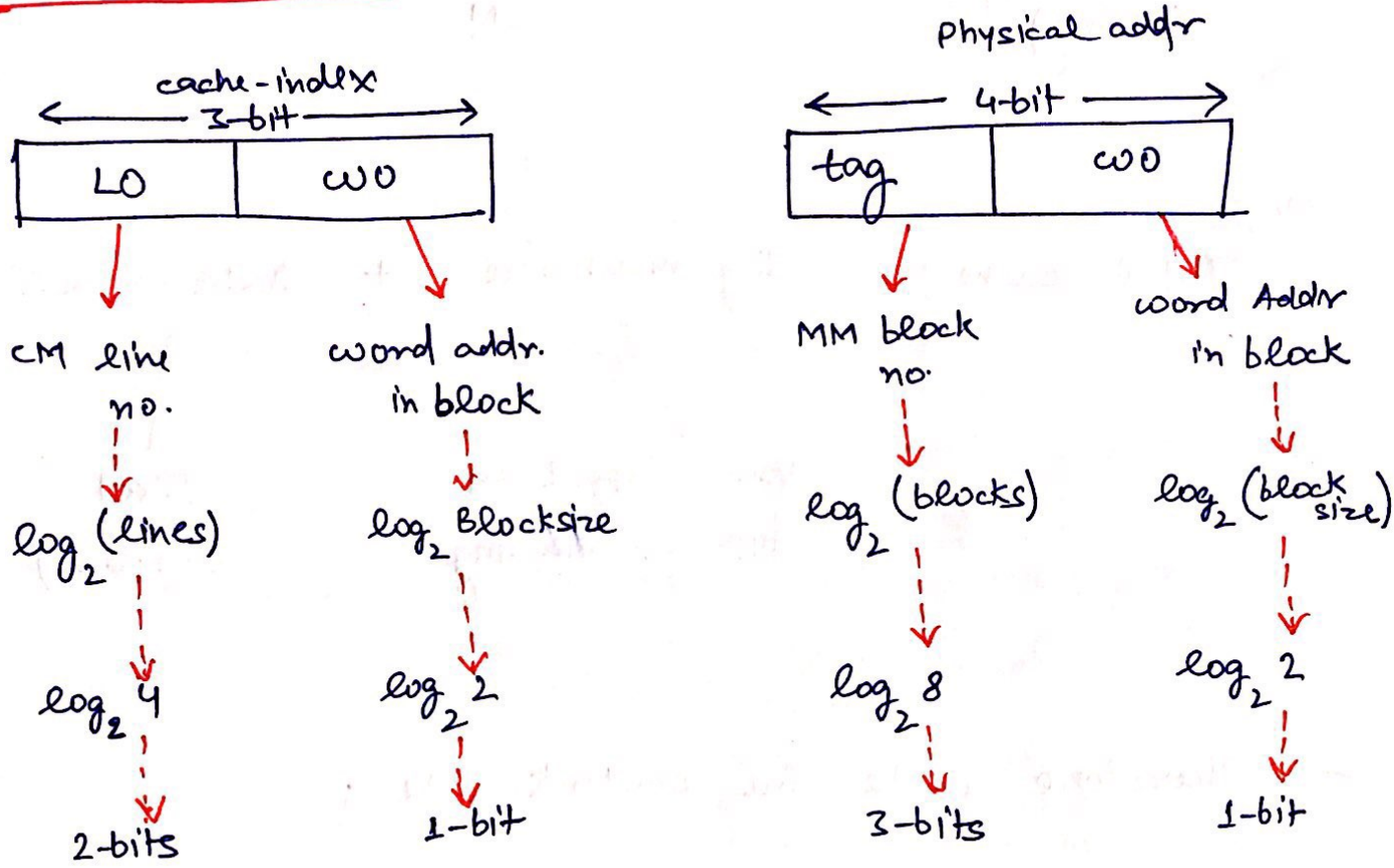
16B → MM

After org^n

Blocks = $\frac{16}{2}$ = 8



Address Formats :-



(ii) Mapping Techniques :-

- The process of a copying the data from the main memory to cache memory is called as mapping.
- In the mapping process, data-block is copied into a cache memory along with the physical address.
- To hold the physical address, storage space is required in cache line known as tag space. Therefore,

(Tag memory-size (directory) = lines in CM * tag space in the line)

(Data-memory (directory) size = lines in CM * Block-size)

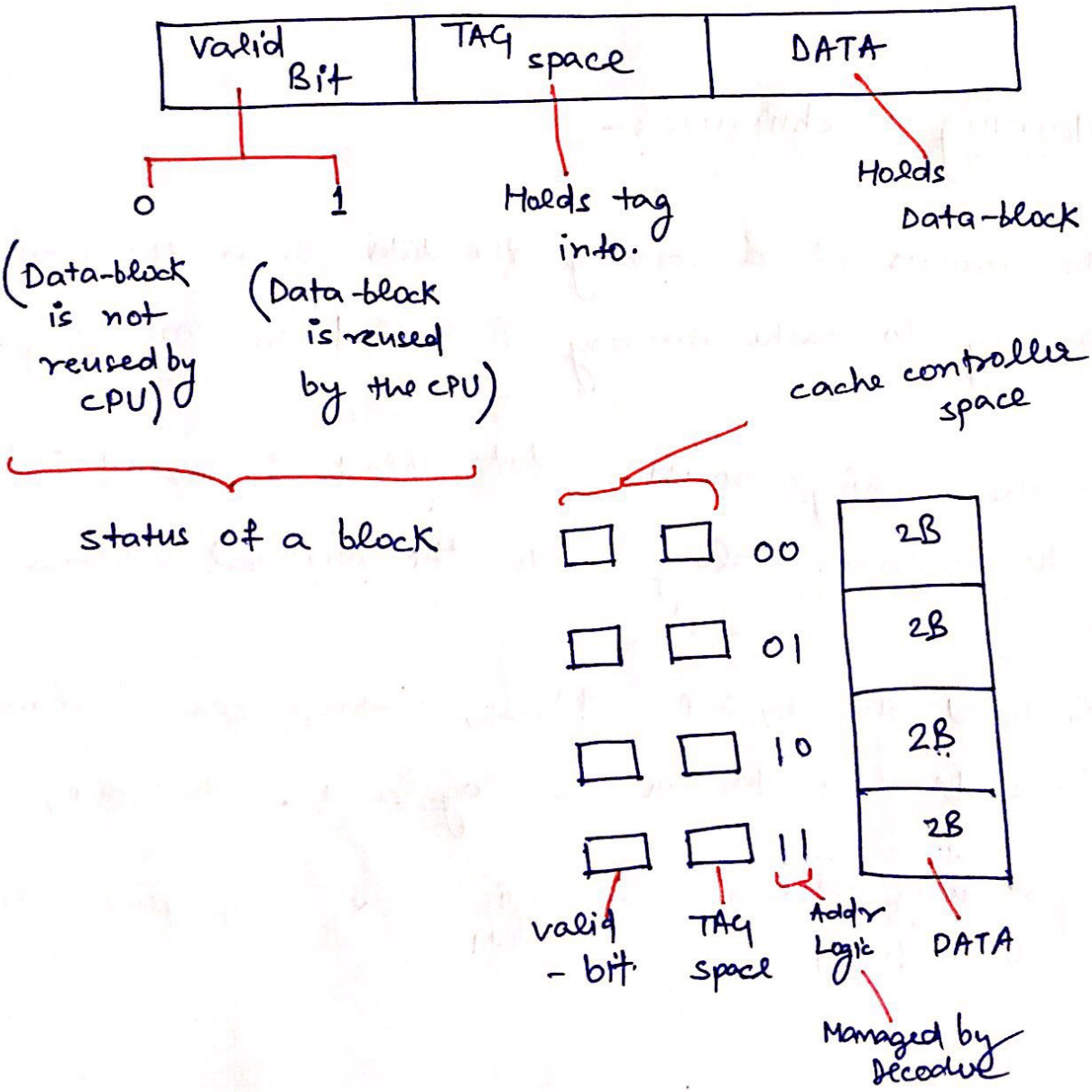
So,

[Total cache = Tag mem size + Data-mem size]

Tag mem size: varies depends on type of mapping

Data-mem size: Fixed (eg. 8 B)

- Therefore cache line contents are :



- Three kinds of mapping techniques used in cache design:-

(i) Direct mapping.

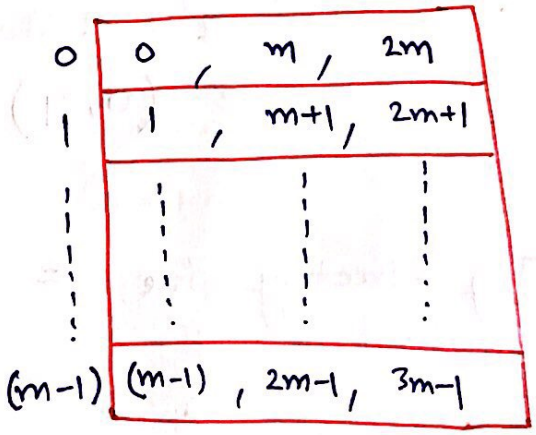
(ii) Associative mapping.

(iii) Set- Associative mapping

⇛ Direct- Mapping :- In this technique 'mod' func is used to map the data i.e $(k \bmod n = i)$.

- k → MM block no
- n → no. of line in CM
- i → CM line no

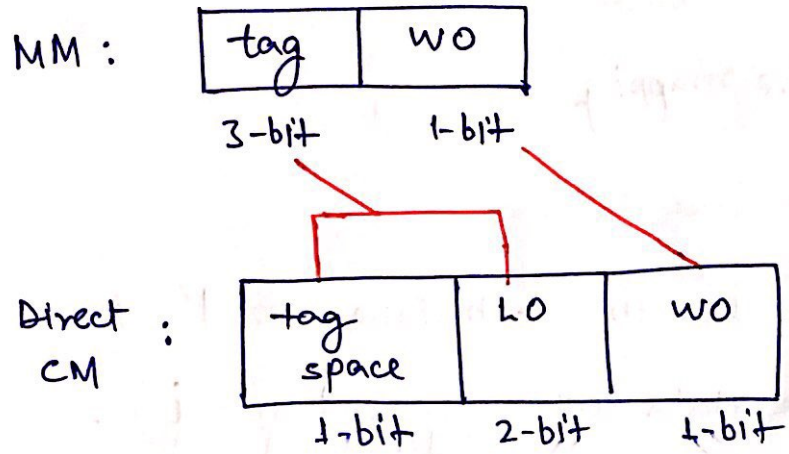
0 mod 4	=	0
1 mod 4	=	1
2 mod 4	=	2
3 mod 4	=	3
4 mod 4	=	0
5 mod 4	=	1
6 mod 4	=	2
7 mod 4	=	3



(row = line-status)

(col = CM-status)

- Mapping func^n shows the relationship b/w the block no. in MM and line no. in CM. Therefore address - binding is :



(it denotes no. of blocks assigned to given line possible single at a time)

$2^1 = 2$ blocks possible

i.e (0, 4) → line 0

Tag Directory size = lines in CM * tag-size in line

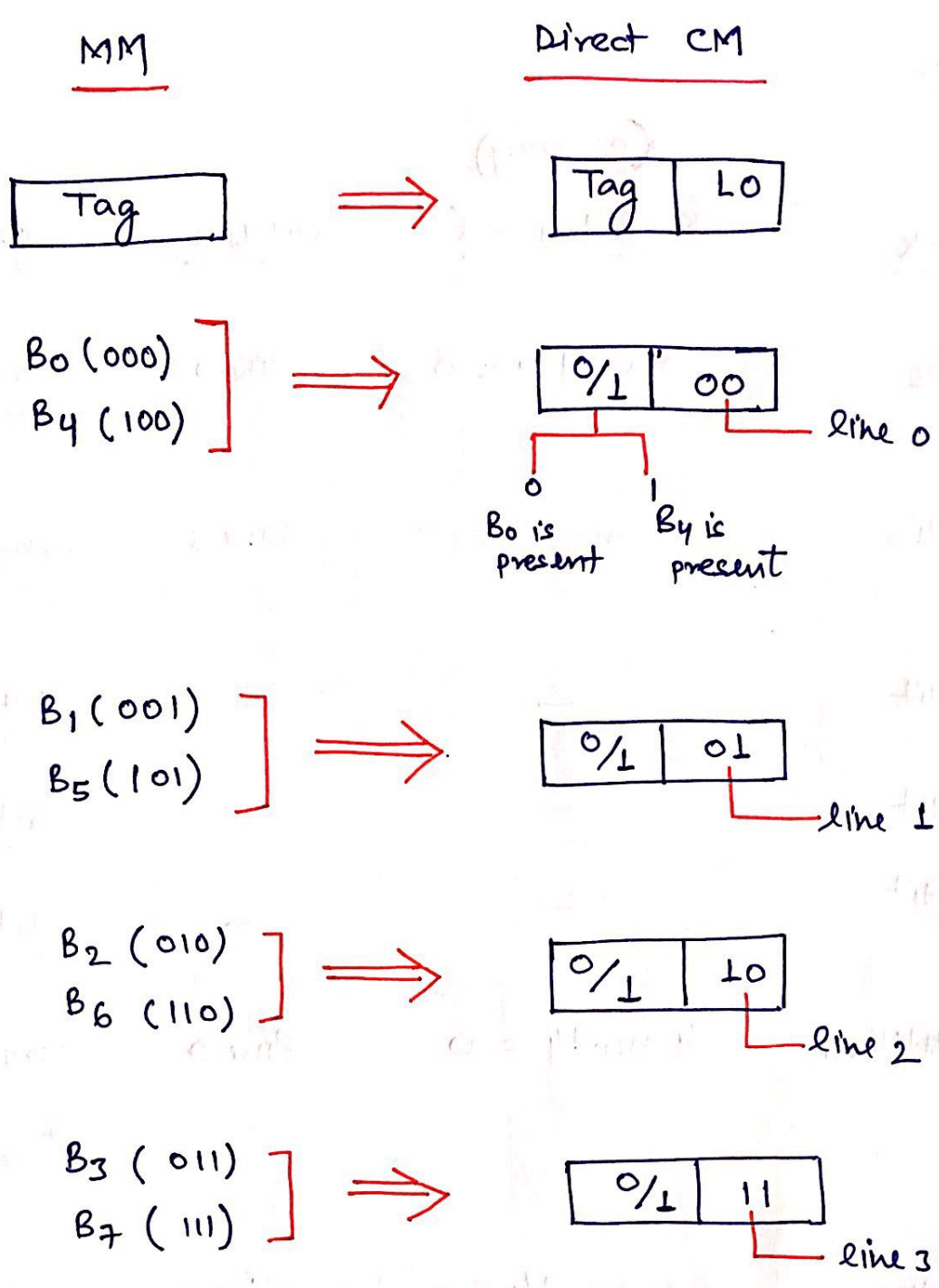
= 4 * 1-bit

= 4-bits

Total cache = Tag cache + Data cache

= 4-bits + 8-bytes

= 68-bits



Runtime analysis :-

Assume CPU generates MM ref : $B_0, B_7, B_0, B_0, B_7, B_4, B_0, B_4, B_0, B_4$.

Initially CM is empty

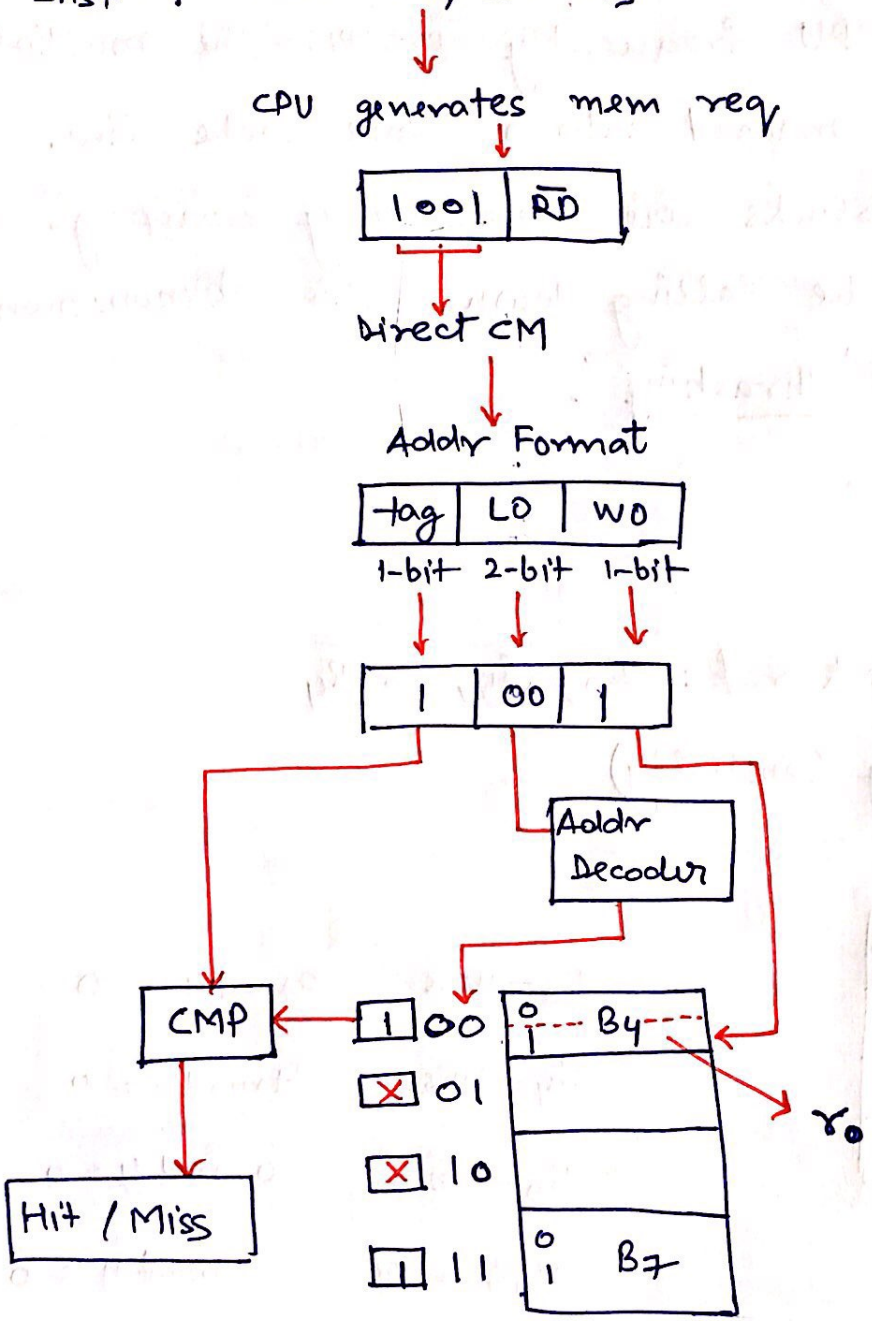
Accessing :-

MM - Block	(mapping) K mod n = i	CM Line	opⁿ
B_0 – Miss	0 mod 4 = 0	line 0	compulsory - miss
B_7 – Miss	7 mod 4 = 3	line 3	compulsory - miss
B_0 – Hit	—	—	hit
B_0 – Hit	—	—	hit
B_7 – Hit	—	—	hit
B_4 – ~~Hit~~ Miss	4 mod 4 = 0	line 0	compulsory miss + ~~replace~~
B_0 – Miss	0 mod 4 = 0	line 0	conflict miss + ~~replace~~
B_4 – Miss	4 mod 4 = 0	line 0	"
B_0 – Miss	0 mod 4 = 0	line 0	"
B_4 – Miss	4 mod 4 = 0	line 0	"

$$\text{Hit-ratio} = \frac{\text{Total Hit}}{\text{Total access}} = \frac{3}{10} = (0.3)$$

Access sequence :-

M/c Inst^n : MOV r_0, [1001]



Limitations :-

(i) In the direct cache design, each line is capable to hold only 1-block (tag) at a time therefore no. of conflict misses will be increases.

(ii) When the CPU frequently prefers the multiple blocks which are mapped into a same cache line. Then the cache blocks are continously swapping. So, hit ratio will be falling down, this phenomenon is known as 'Thrashing'.

Thrashing -

Assume MM Block ref : B_0, B_4, B_0, B_4

CM : empty (initially)

B_0 – miss $0 \bmod 4 = 0$

B_4 – miss $4 \bmod 4 = 0$

B_0 – miss $0 \bmod 4 = 0$

B_4 – miss $4 \bmod 4 = 0$

$$\text{Hit ratio} = \frac{0}{4} = 0$$

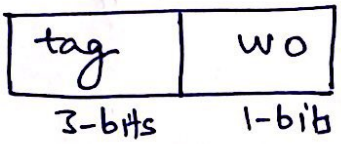
$H = 0$ → Thrashing

- For preventing from Thrashing alternative mapping is required i.e Associative mapping.

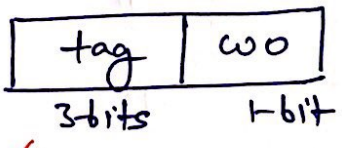
⇉ Associative mapping :- It is designed without address called as content - addressable memory.

- In this cache design, sequence func^n (counter logic) is used to map the data so any MM block can be mapped into a any CM line in a sequence.
Therefore address binding is :-

MM Addr :



Associative CM :



(tag-space)

Tag Directory size = lines in CM * tag space in the line

= 4 * 3 bits

= 12-bits

So, Total cache = Tag cache + Data cache

= 12- bits + 8 Bytes

= 76 bits

Assume,

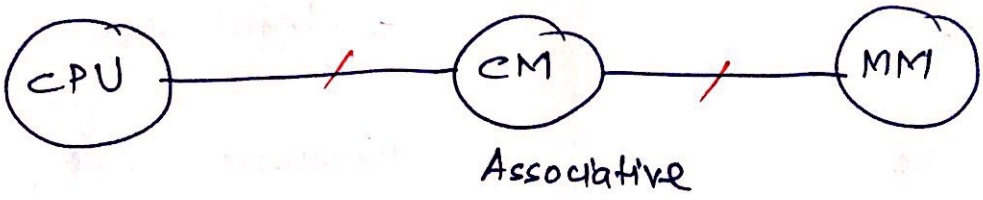
MM Block ref: $B_0, B_4, B_0, B_4, B_3, B_7, B_3, B_7, B_0, B_2$

CM: empty

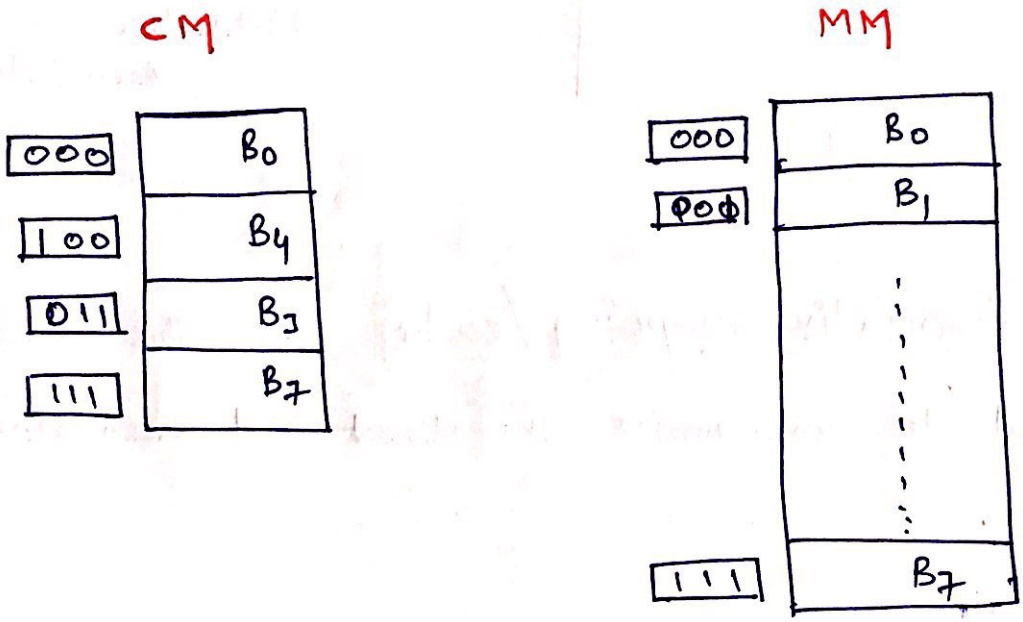
MM Block ref	Seq - func^n
B_0 – Miss	Any line in a sequence
B_4 – Miss	//
B_0 – Hit	//
B_4 – Hit	//
B_3 – Miss	//
B_7 – Miss	//
B_3 – Hit	//
B_7 – Hit	//
B_0 – Hit	//
B_2 – Miss	replacement policy – FIFO, LRU

* FIFO & LRU policies are required to replace the data when the cache is FULL.

$$\left(\text{Hit - Ratio} = \frac{5}{10} = 0.5\right)$$

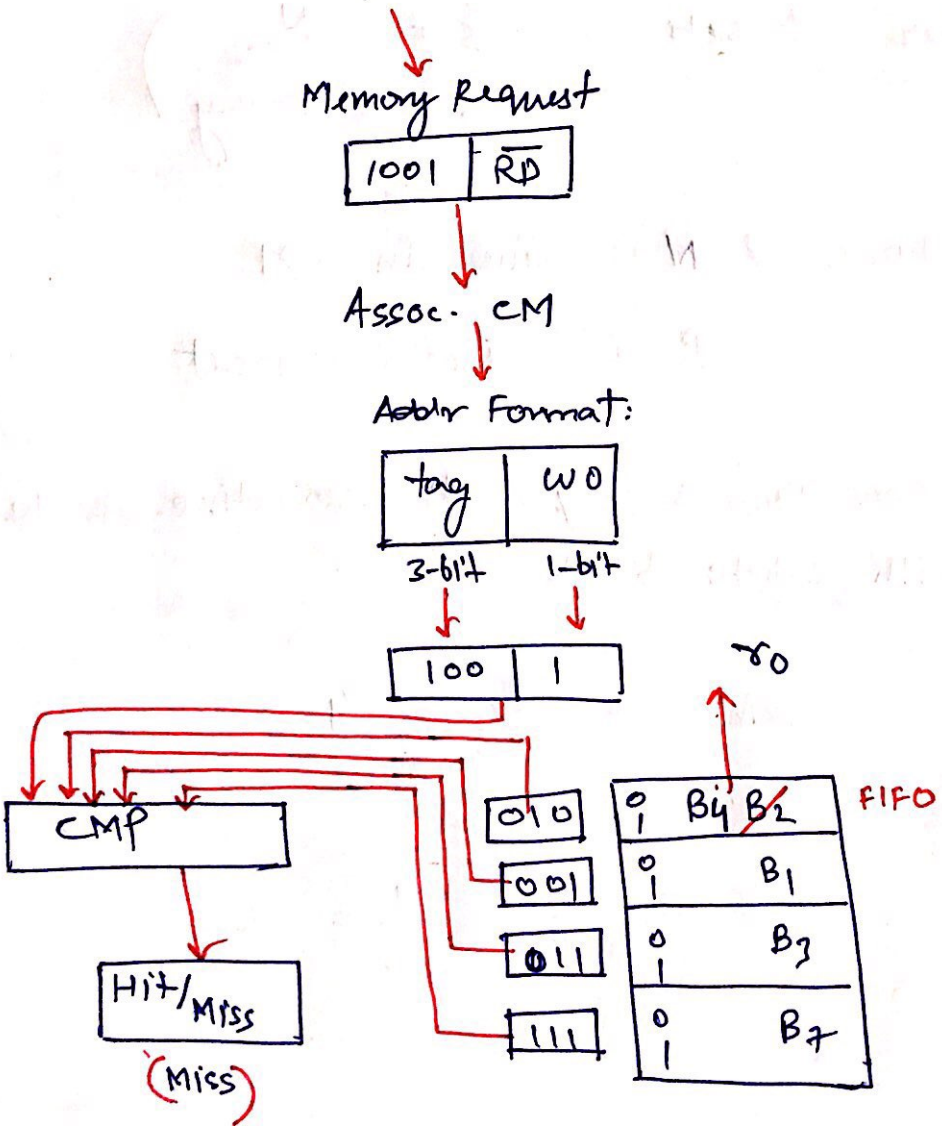


Associative



Accessing Sequence –

M/C inst^{n} : MOV r0, [1001]



Advantage:-

Fastest cache

Disadvantage:-

Expencive cache, required alternative i.e Set-Associative.

⇒ **Set-Associative mapping/cache :-** This cache structure is used to compromise the direct and associative cache designs.

- In this cache design, lines are grouped into a sets. to accomodate multiple blocks in the set.

- Number of sets, $\left(S = \frac{N}{P\text{-way}} \right)$

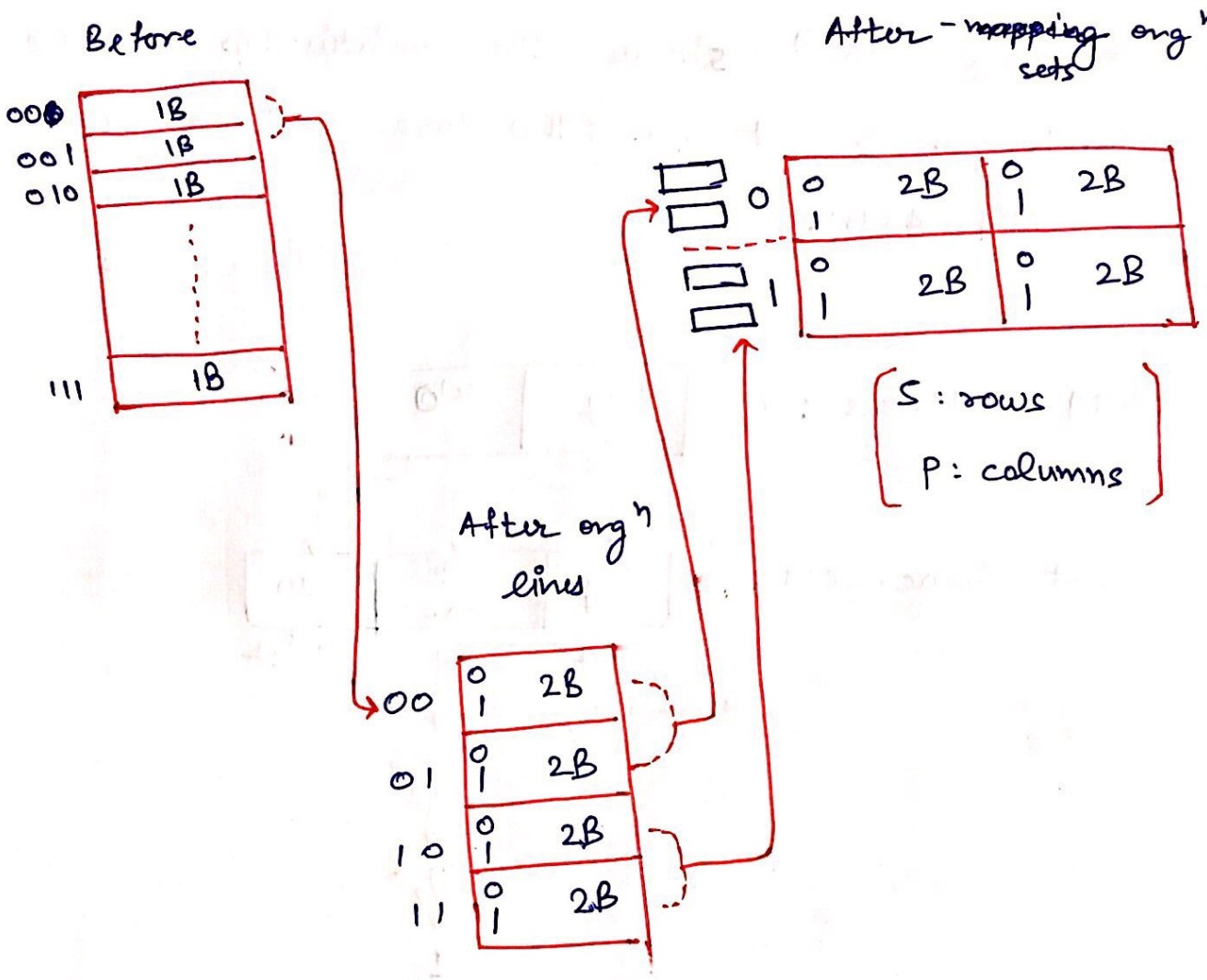
where N is lines in CM

P is line in a set

Eg:- consider 2-way set associative cache of 8-byte with 2-byte block

$$\text{lines} = 8/2 = 4$$

$$\text{sets} = \frac{N}{P} = \frac{4}{2} = 2$$



- In this design 'mod' func^n is used to map the data

i.e (K mod S = i).

K — MM Block no

S — sets in CM

i — CM set number

0 mod 2 = 0 S_1

1 mod 2 = 1 S_2

2 mod 2 = 0 S_1

3 mod 2 = 1 S_2

4 mod 2 = 0 S_1

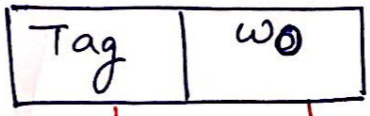
5 mod 2 = 1 S_2

6 mod 2 = 0 S_1

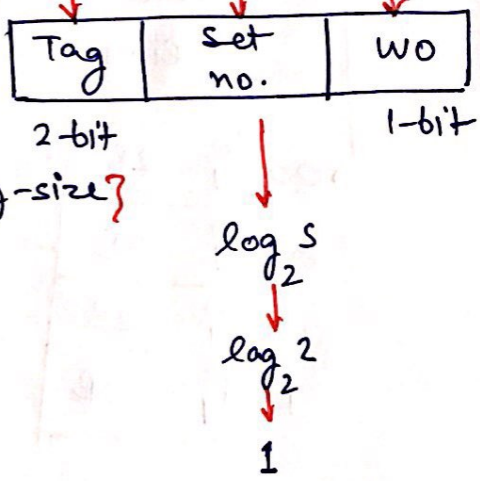
7 mod 2 = 1 S_2

- mapping func^n shows the relationship b/w the block no. & set no. Therefore address binding is MM Addr.

MM Address :



Set-Assoc. CM :



Tag-Directory size = lines in CM * tag-space in the line

$= N * 2\text{-bits}$

$= (p * s) * 2$

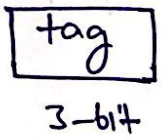
$= (2 \times 2) \times 2$

$= 8\text{-bits}$

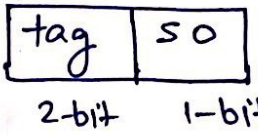
Type	Tag-Directory size
Direct	4-bits
Associative	12-bits
Set-Associative	8-bits

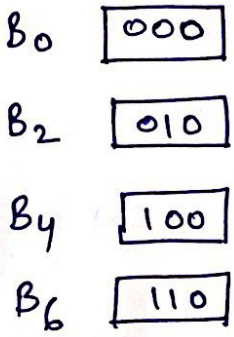
NOTE:- In this tag-space = 2-bits so, 2^n i.e $2^2 = 4$ different blocks are belongs to each set but only 2 at a time.

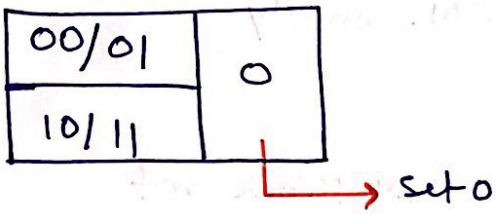
MM

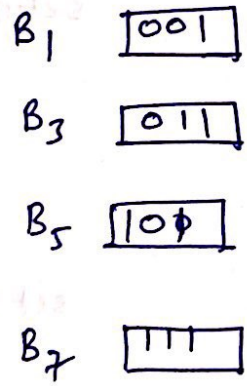


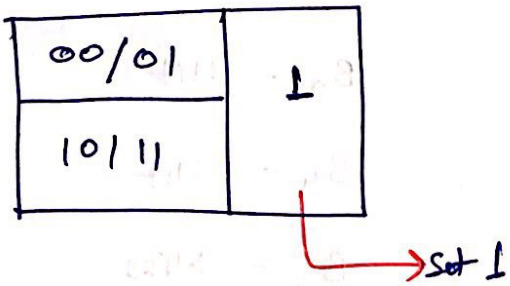
Set-Assoc CM

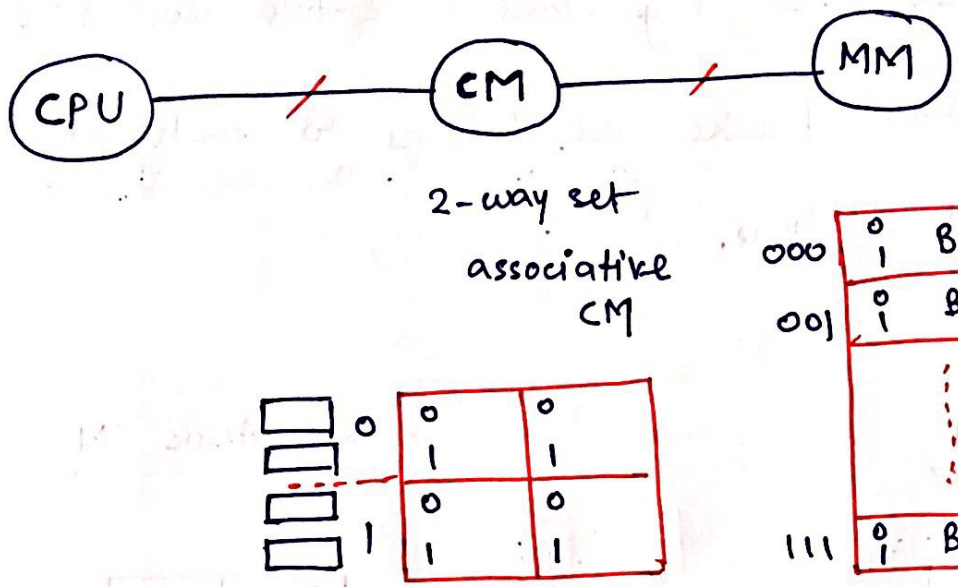












Assume,

MM Block ref : $B_0, B_4, B_0, B_4, B_3, B_7, B_3, B_7, B_0, B_2$

CM : empty

MM Block ref	k mod S = i	CM set
B_0 – Miss	0 mod 2 = 0	set 0
B_4 – Miss	4 mod 2 = 0	set 0
B_0 – Hit		
B_4 – Hit		
B_3 – Miss	3 mod 2 = 1	set 1
B_7 – Miss	7 mod 2 = 1	set 1
B_3 – Hit		
B_7 – Hit		
B_0 – Hit		
B_4 – Miss	2 mod 2 = 0	set 0

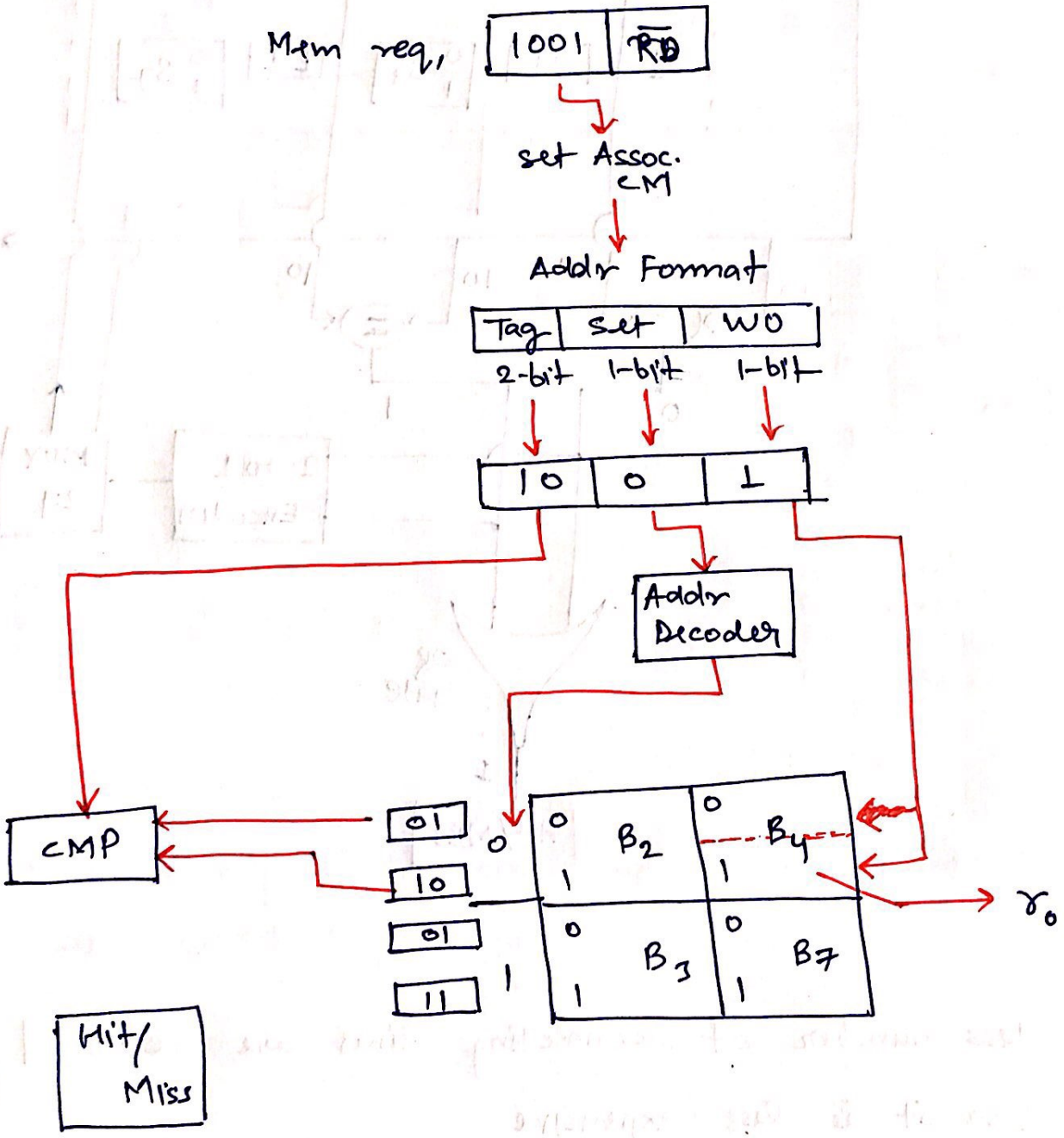
set 0 → FIFO: ~~B_0~~ B_2 ; LRU: ~~B_4~~ B_2

(Hit-Ratio $H = \frac{5}{10} = 0.5$)

NOTE:- FIFO & LRU policies are used to replace the data when the set is full.

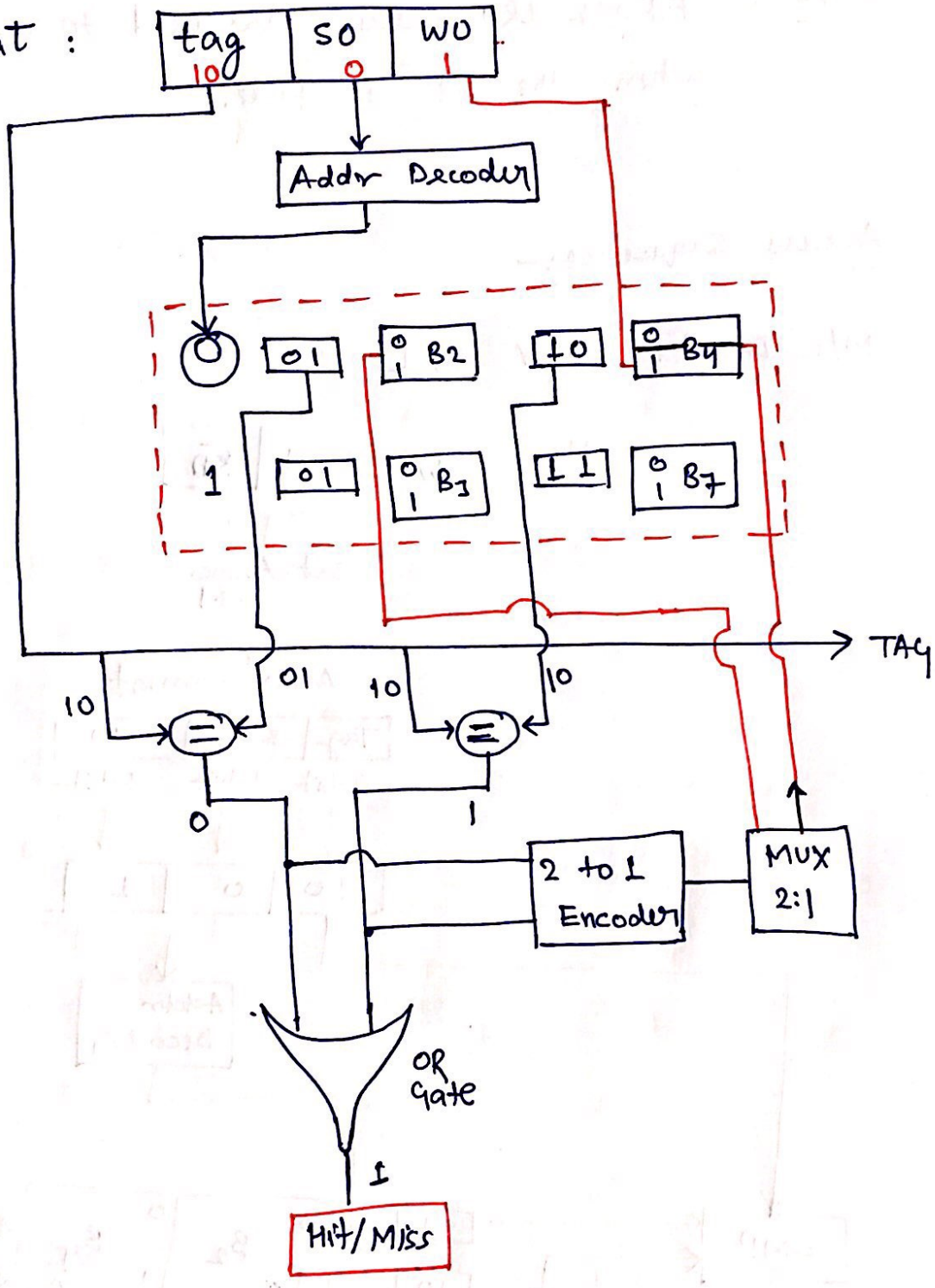
Access Sequence:-

M/c Instⁿ: MOV r0, [1001]



comparision circuit :-

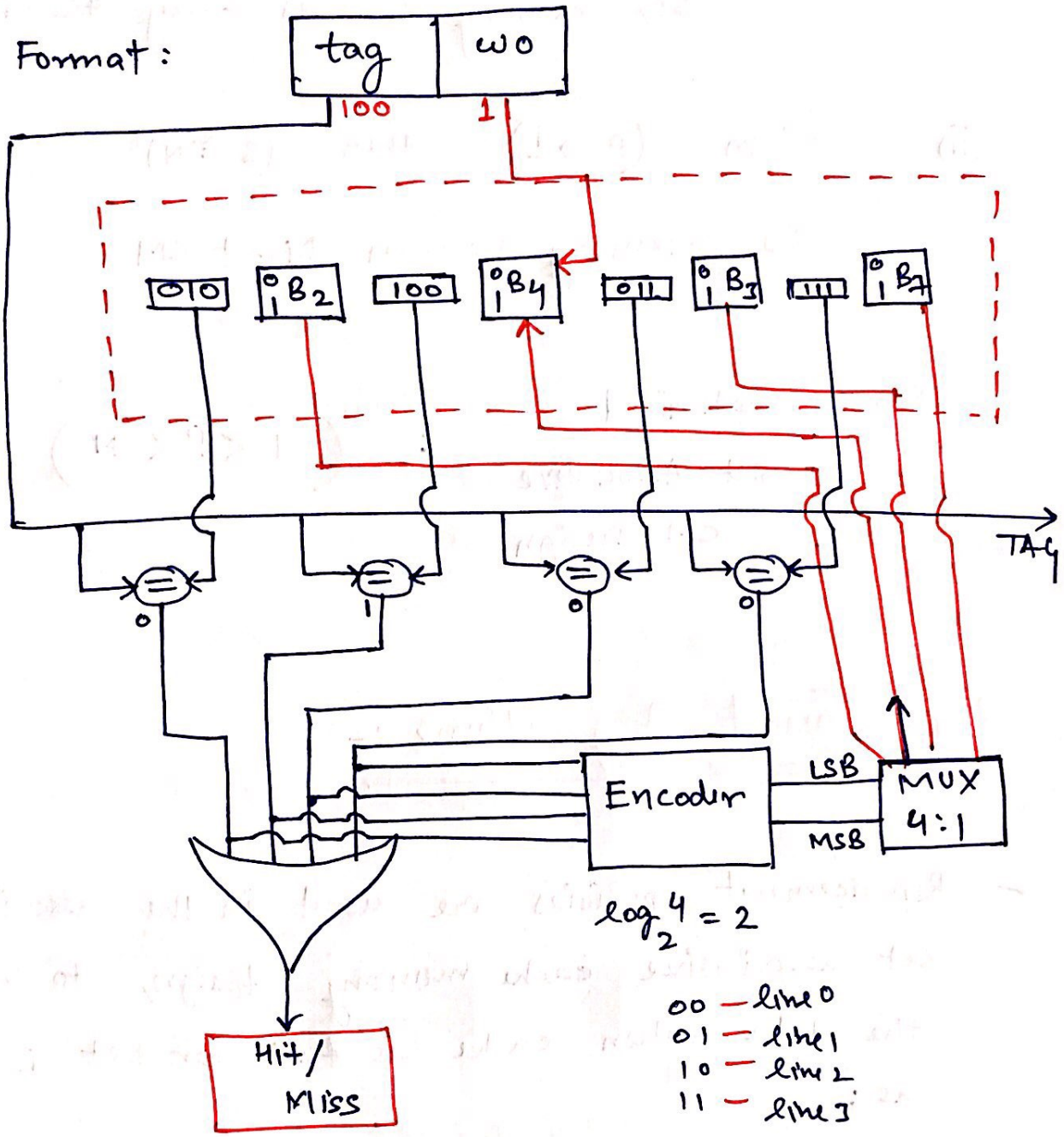
Addr Format :



* Less number of connecting lines are required so it is less expensive

Comparision ckt of Associative CM:-

Addr Format:



* More no. of CMP ckt and more no. data & control lines are required so it is expensive.

NOTE :- (i) When $(P \not\rightarrow N)$ then $(S \cong 1)$

So, memory becomes fully Associative CM

(ii) when $(P \rightarrow 1)$ then $(S \cong N)$

So, memory becomes Direct CM

So, optimized set Associative CM Design : $(1 < P < N)$

Replacement Algorithms :-

- Replacement policies are used in the associative & set associative cache memory design, to replace the data when cache is full or set is full name as :

  (i) FIFO

  (ii) LRU

- In FIFO, replace the block which ~~is present~~ enters in the cache ~~longest time without refrence~~. firstly.

- In LRU, replace the block which is present in the cache longest time without refrence.

Q. Consider 32KB direct cache organized into a 256 B blocks. Cache memory data is subset of a 2^{28} B storage space. In the cache controller, each tag compromising of 1 invalid bit and 1 update bit. what is the tag directory in the cache controller?

CM size = 32KB

Direct CM

Block-size = 256 B

MM size = 2^{28} B

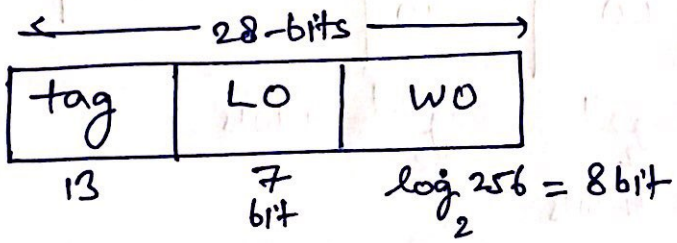
Physical addr size = 28-bits

N (no. of lines in CM) = $\frac{\text{CM size}}{\text{Block-size}}$

$N = \frac{32\ KB}{256\ B}$

$N = 2^7$

So, Addr Format



Tag directory size = N * tag-space

$= 2^7 (13 + 1 + 1)$

$= 2^7 * 15$ bits

Q: Consider 64-bit hypothetical CPU which supports 8KB direct mapped cache org^n into 64 words blocks. To which cache line (in decimal) the following MM data is mapped. 0x FC3D6D

CM size = 8KB = 2^{13} B

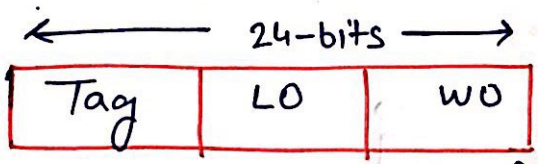
Direct mapping –

Block size = 64 w

= 64 x 8B = 2^9 B

No. of lines (N) = $\frac{2^{13}}{2^9}$ = 2^4 lines

Addr – format:–

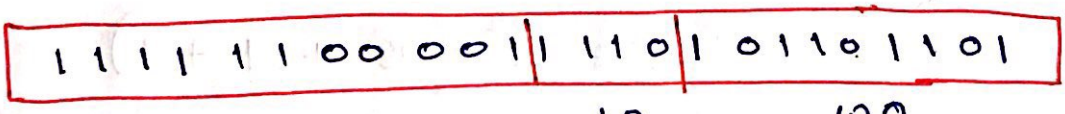


11-bits $\log_2 2^4$ = 4-bits $\log_2 2^9$ = 9-bits

0x FC3D6D

4 = 4x6 = 24-bits

FC3D6D



line = 14

Q. Consider 32-bit CPU supports 8-way set assoc. cache of 64KW memory, organized into a 256 B blocks. CPU generates 32-bit physical address to access the data. In the cache controller each tag is consist of 1 valid bit, 1 update bit, 2-replace bits. What is the size of tag directory.

1W = 2 32-bit = 4B

set-size (P)
8 ~~no of set~~ (~~S~~) = 8 (set-associative mapping)

CM size = 64KW

= 64K x 4B

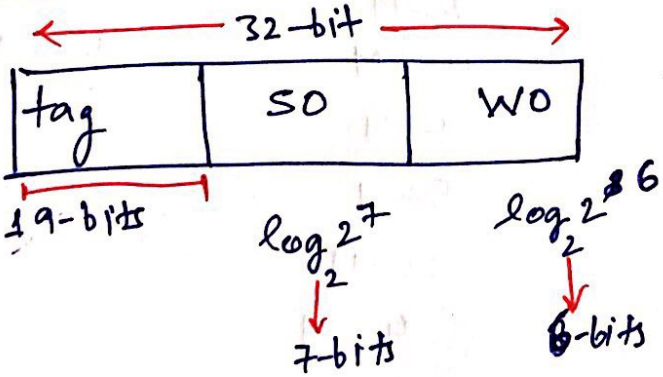
= 2^{18} B

Block size = 256 B = ~~2^8 B~~ 2^6 W

No. of lines (N) = $\frac{2^{18}}{2^8} = 2^{10}$ lines

No. of set (S) = $\frac{N}{P} = \frac{2^{10}}{2^3} = 2^7$

Addr Format:-



Tag-directory size = N * (tag-space + 1 valid bit + 1 update bit + 2 replace-bits)

$= N * (19 + 1 + 1 + 2)$

$= 2^{10} * 23$ bits

$= 23K$ bits

Q. Consider 4 blocks cache memory, initially empty with following main memory references 4, 5, 7, 12, 4, 5, 13, 4, 5, 7. Identify the hit ratio using -

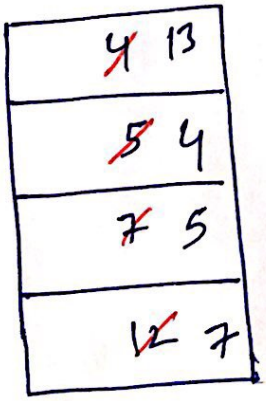
(a) FIFO

(b) LRU

(c) Direct Mapped CM

(d) 2-way set Associative with LRU

(a) FIFO sequence -

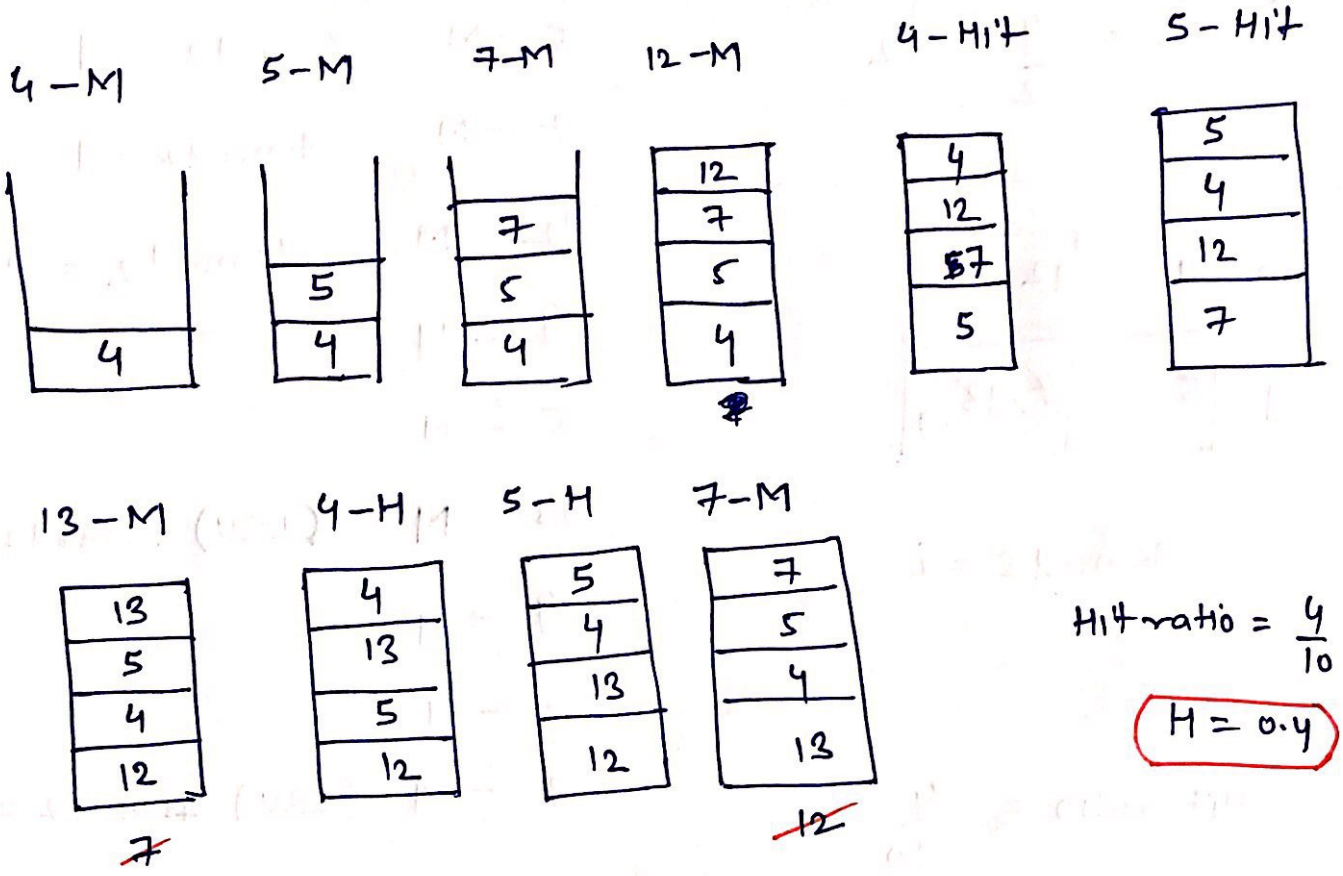


4 - M
5 - M
7 - M
12 - M
4 - H
5 - H
13 - M
4 - M
5 - M
7 - M

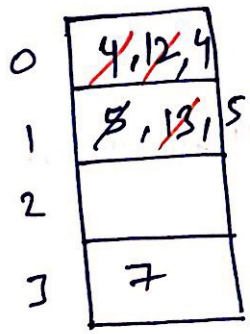
H = 2/10

H = 0.2

(b) LRU sequence



(c) Direct mapping :-



4-M 4 mod 4 = 0

5-M 5 mod 4 = 1

7-M 7 mod 4 = 3

12-M 12 mod 4 = 0

4-M 4 mod 4 = 0

5-H

13-M 13 mod 4 = 1

4-H

5-M 5 mod 4 = 1

7-H

Hit-ratio = 3/10

H = 0.3

(d) $N = 4$

$$S = \frac{4}{2} = 2$$

0	4	12
1	5	~~7~~, ~~13~~, 7

$K \bmod S = i$

4 - M	$4 \bmod 2 = 0$
5 - M	$5 \bmod 2 = 1$
7 - M	$7 \bmod 2 = 1$
12 - M	$12 \bmod 2 = 0$
4 - H	
5 - H	
13 - M	(LRU) $13 \bmod 2 = 1$
4 - H	
5 - H	
7 - M	(LRU) ~~7~~ $\bmod 2 = 1$

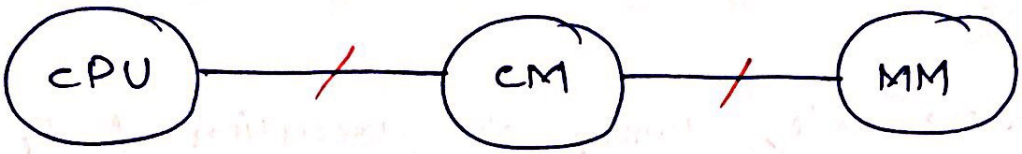
$$\text{Hit ratio} = \frac{4}{10}$$

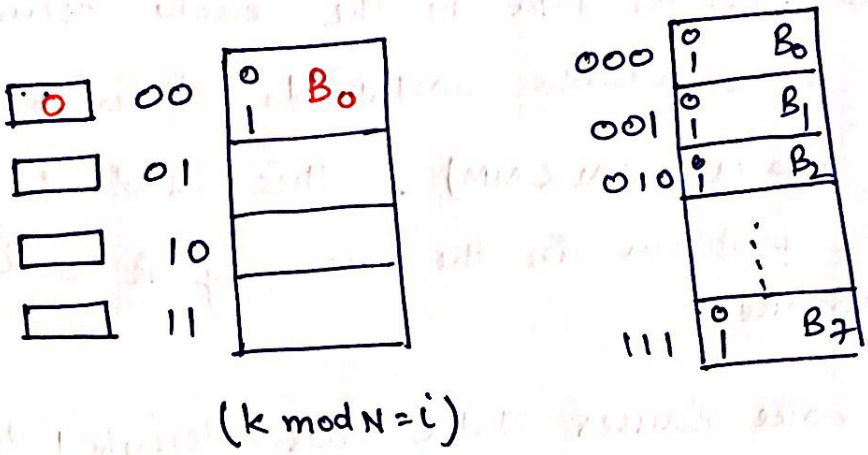
$$H = 0.4$$

Updating Techniques :-

- According to memory hierarchy design, CPU performs the READ & WRITE op^n only on a cache memory.
- In this regard, CPU always checks the availability of data in the cache memory. When it is available then op^n become READ hit or WRITE hit otherwise op^n becomes READ miss or WRITE miss.

- when miss occured then respective block is copied from the MM to CM called as READ allocate or WRITE allocate. After the allocation, CPU perform -s the READ & WRITE opn only on a cache memory.





Prog:-

I_1: MOV r_0, [0000] — B_0 – 0th byte → r_0

I_2: Add r_0, #23 — $r_0 \leftarrow r_0 + 23$

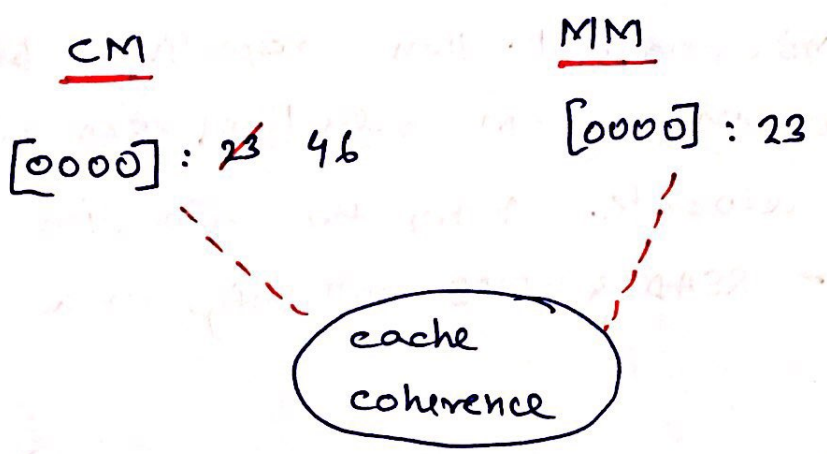
I_3: MOV [0000], r_0 — B_0 – 0th byte ← r_0

Execn :-

I_1: READ miss; READ allocate (0 mod 4 = 0) ; $r_0 = 23$

I_2: $r_0 + 23 \rightarrow r_0$; $r_0 = 46$

I_3: WRITE hit, CM : [0000] : ~~23~~ 46



- In the above code, during the execution of I_7 inst^n CPU updates the data in the cache memory only therefore same address contains different value at different places (CM & MM). This kind of data-inconsistency problem in the memory is called as cache coherence.

- Coherence causes data loss described below -

I_4: MOV r_0, [1000] ; B_4 - 0^{th} byte $\rightarrow r_0$

I_5:

I_6:

I_7:

I_8: MOV r_0, [0000] ; B_0 - 0^{th} byte $\rightarrow r_0$

Execution:-

I_4: READ miss; READ - allocate

$K \bmod N = i$

$4 \bmod 4 = 0$

CM : ~~B_0~~ B_4

CM: B_4 → updated data is lost 'I8' Instⁿ incorrectly access the old value from the block 'B_0'. Data Loss occurred.

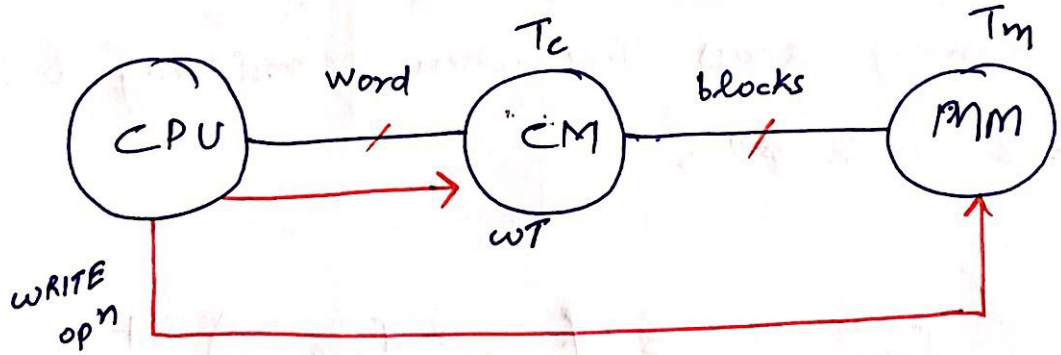
- To handle the above problem updating techniques are used in the cache design. These are of two kinds:

Write Through protocol

Write back protocol

(i) Write Through protocol :- In this protocol, CPU performs the simultaneous write opⁿ in both CM and MM so no coherence. Inclusion is always is success.

Inclusion means lower level memory data is always subset of higher level memory data.



simultaneous WRITE opn $(T_W) = \text{Max}\left(\text{word updation time in CM} \;/\; \text{word updation time in MM}\right)$

- Time required to READ one-word data from the memory is called as Read-cycle time T_{avg}.

ie $T_{avg} = H_r T_c + (1-H_r)(T_m + T_c)$

$H_r T_c$: Read Hit, Read Data; $(1-H_r)(T_m + T_c)$: Read miss, Read allocate, Read data

- Time required to WRITE 1-word data into a memory is called as write-cycle time $T_{avg_{write}}$.

$T_{avg_{write}} = H_w T_w + (1-H_w)(T_m + T_w)$

$H_w T_w$: write Hit, write simultaneous; $(1-H_w)(T_m + T_w)$: write miss, write allocate, simultaneous write

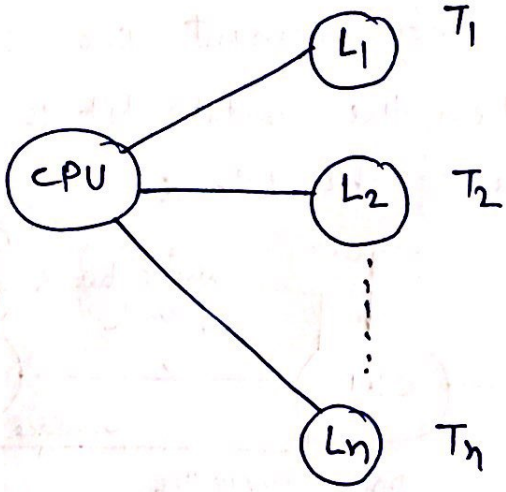
- Avg. memory access time when considering both READ & WRITE opn is

$$T_{avg_{WT}} = (f_{read} * T_{avg_{read}}) + (f_{write} * T_{avg_{write}})$$

$$\left(\eta_{WT} = \frac{1}{Tavg_{WT}} \text{ words/sec}\right)$$

NOTE:- when the write-through protocal is implemented in a simultaneous access memory org^n then H_w always becomes 1 because data block is not needed in the first level (L1 memory).

$H_w = 1$



$H_w = 1$

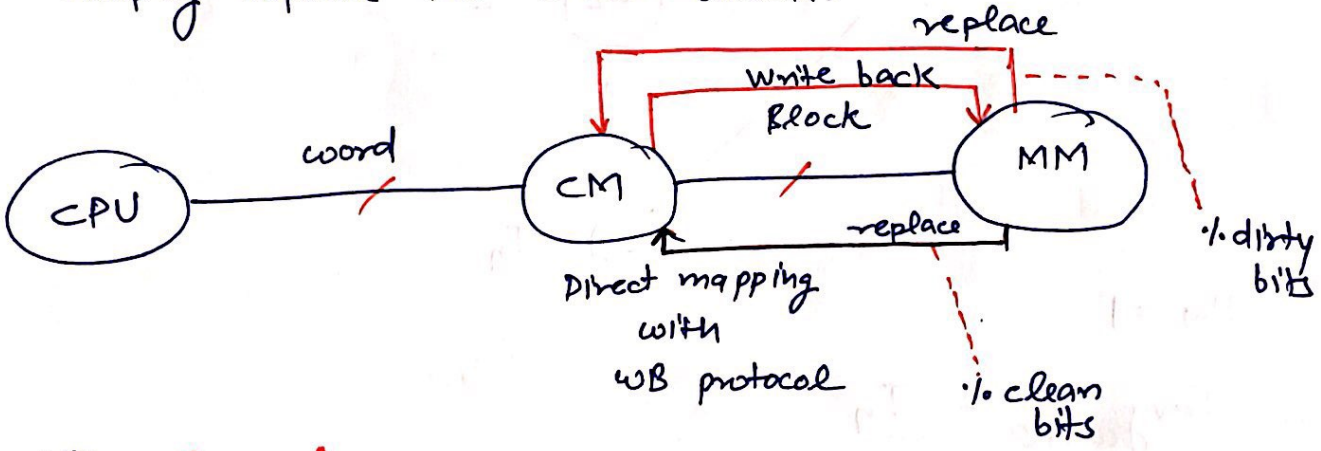
$Tavg_{write} = T_n$

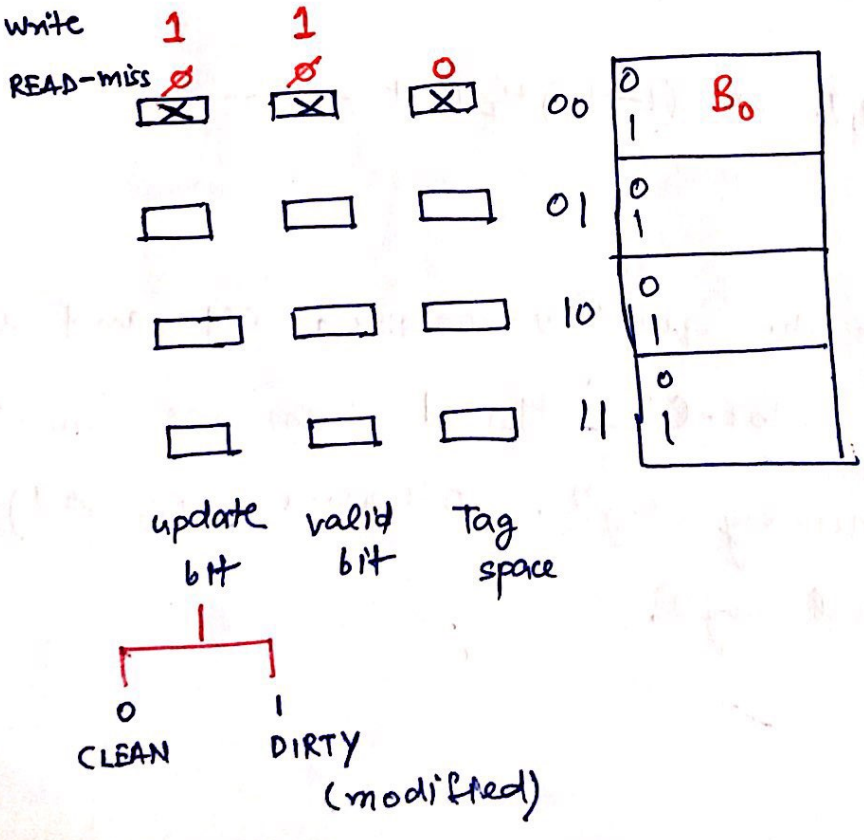
$Tavg_{read} = H_1T_1 + (1-H_1)H_2T_2 + \text{-----}$

NOTE:- when the question contains CM and write Through protocol & $H_w = 1$ then use simultaneous access memory org^n. otherwise ($H_w \neq 1$), use hierarchal org^n.

(ii) **Write-back protocal :-** In the write back cache design each line contain 1-bit storage space called as updati bit, use to hold the status of a updation. This bit is set, when the cache block is updated otherwise, reset.

- In this protocal, CPU performs the WRITE op^n only in the cache memory. So, coherence present. It doesn't causes the data loss bcz CPU reads the status of update bit before replacement c.e CPU writeback the cache block when the update bit is 1 otherwise simply replace the cache block.





- Read cycle-time $T_{avg\ read}$;

$$T_{avg\ read} = H_r T_c + (1 - H_r)\left[\%\ \underset{-bits}{dirty}\ (T_m + T_m + T_c) + \%\ \underset{-bits}{clean}\ (T_m + T_c)\right]$$

(replace) access time; T_m + T_m + T_c: write back, read allocate, read data; $T_m + T_c$: (replace only)

- Write-cycle time $T_{avg\ write}$;

$$T_{avg\ write} = H_w T_c + (1 - H_w)\left[\%\ \underset{-bits}{dirty}\ (T_m + T_m + T_c) + \%\ \underset{-bits}{clean}\ (T_m + T_c)\right]$$

$H_w T_c$: write hit, write data; $(1 - H_w)$: write miss; $T_m + T_m + T_c$: write back, write allocate, write data; $T_m + T_c$: write allocate, write data

Avg. memory access time when considering both read & write op^n is

$$T_{avg\ WB} = \left(F_{read} * T_{avg\ read}\right) + \left(F_{write} * T_{avg\ write}\right)$$

$$\eta_{WB} = \frac{1}{T_{avg_{WB}}} \quad \text{words/sec}$$