* Illusion of unlimited fast memory.

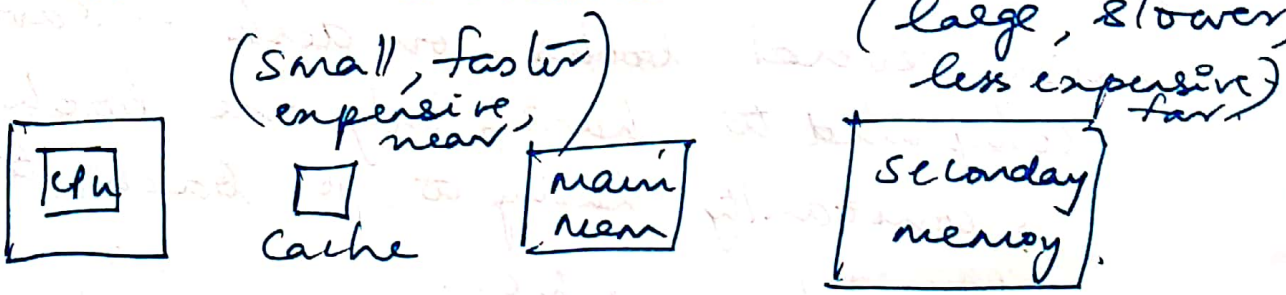
eg: of library:
If we want to read a particular topic – collect related books, bring it to desk and start reading.
Having several books on desk – saves time compared to having only one book & constantly having to go back to shelf to take another.

≈ a pgm does not access all of its code or data at once

Principle of locality – pgm accesses a relatively small portion of their addr space at any instant of time.

Two types:
① temporal locality (in time) – If an item is referenced, it will tend to be refer again soon. eg: loops.
② spacial locality (in space) – If an item is referenced, items whose addresses are close by will tend to be referenced soon. eg: arrays

By taking advantage of principle of locality, memory of a computer is implemented as a mem hierarchy.
- a structure that uses multiple levels of memories. , as distance from CPU increases, size & access time both increases.



size - increases →
cost - decreases
speed - increases (access time)
distance - increases

Mem hierarchy - multiple levels

Two level hierarchy:
- One focus - only on two levels.
- upper level - closer to CPU, smaller, faster, expensive than lower level.

* Minimum unit of info that can either be present or not in 2-level hierarchy - block or line.

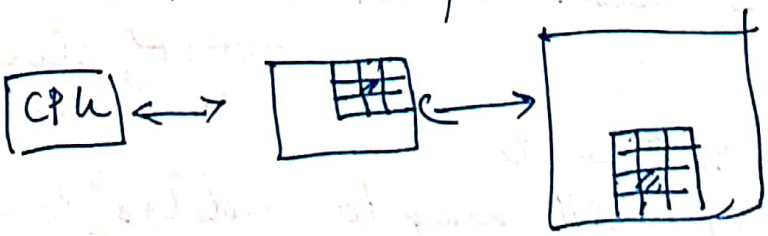
* if data requested by CPU, appears in some block in upper level - hit.
not found - miss.

hit ratio (rate) - fraction of mem accesses found in upper level (measure of perf of mem hierarchy)
Total hits / Total accesses.

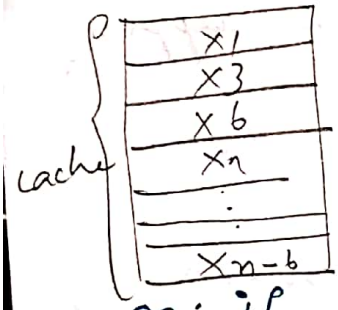
miss ratio (rate) - fraction of mem accesses not found in upper level.

Hit time - time reqd to access a level of mem hierarchy including time needed to determine whether the access is a hit a miss.

miss penalty - time reqd to fetch a block into a level of mem hierarchy from the lower level, including the time to access the block, transmit it from one level to other, insert it in the level that experienced the miss.

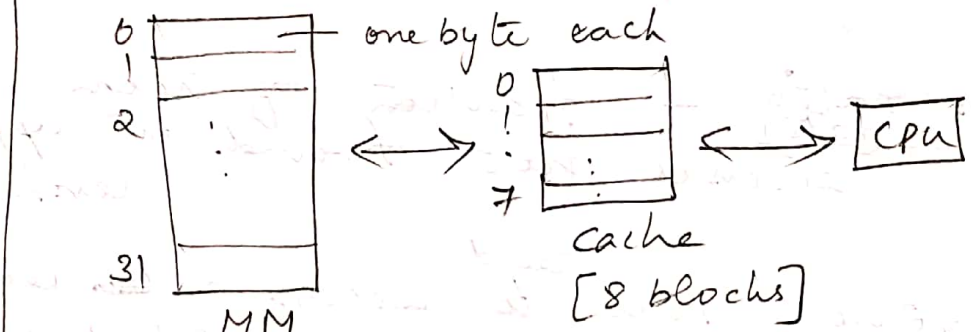


Cache - level of mem hierarchy b/w CPU & MM.



eg: if main memory has 32 blocks and cache is only 8 blocks.

which block in MM will go to which block in cache?



[here block addrs = byte addrs]

* data to be processed by CPU has to brought to cache first.

→ various schemes → direct mapping
→ set associative mapping
→ fully associative.

① Direct mapping.

cache addrs = MM block addrs % no: of block in cache

eg: 2%8 = 0

M[2] will map to addrs 0 in cache

12%8 = 4 M[12] " " 4 in cache

M[0], M[8], M[16], M[24] → cache address 0

map to

0%8, 8%8, 16%8, 24%8 = 0

≈ M[1], M[9], M[17] M[25] → 1

M[2], M[10], M[18], M[26] → 2

M[7], M[15], M[23], M[31] → 7

8 blocks in cache

In this eg. 4 mem references go to one entry in cache

∴ how do we know whether a requested word is in cache or not?

→ tag bits. → addrs info reqd to identify whether a word in the cache corresponds to requested word.

→ upper portion of addrs, not used for index.

eg: for a 32 block main mem & 8 block cache

5 bits for 32 — tag | index addrs → for cache (8 blocks)

	tag	index addrs	
M[0]	00	000	
M[8]	01	000	
M[16]	10	000	
M[24]	11	000	
	diff for each addrs	same → 0	

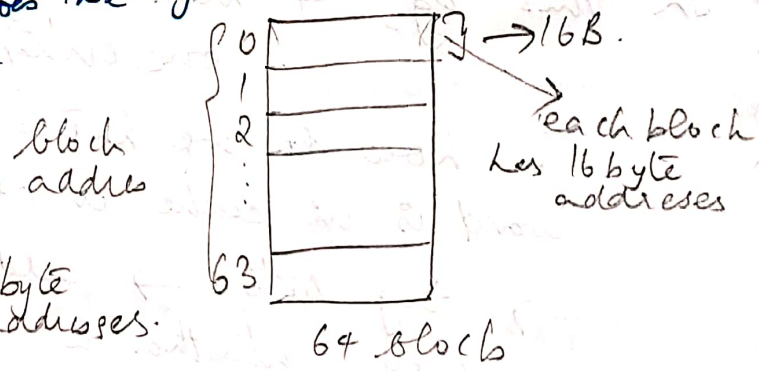
	tag	index addrs	
M[1]	00	001	
M[9]	01	001	
M[17]	10	001	
M[25]	11	001	
	diff for each addrs	same → 1	

Fig 7.6 – example page – 477.

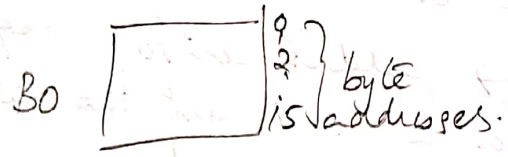
valid bit – also in cache

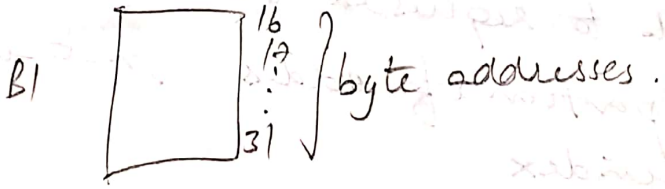
– cache contains valid bit to indicate whether an entry contains a valid address or not.

Q) Consider a cache with 64 blocks and a block size of 16 B. What block no: does the byte addres 1200 map to?



block addres





≈ other Blocks.

byte address 0 → 15 go to block 0 in cache.

Block addres 0 → 0 % 64 = 0

16 → 31 : block addres 1

≈ 32 → 47 " 2

$\lfloor$ byte address / no: of bytes $\rfloor$ = block no:

$\therefore \left\lfloor \frac{1200}{16} \right\rfloor = 75$

cache address = block address % no: of blocks

= 75 % 64 = 11

Q) cache with 128 blocks & a block size of 8B. what block no: does the byte address 2004 map to?

block no: = $\left\lfloor \frac{\text{byte address}}{\text{block size}} \right\rfloor$ = 250

cache address = 250 % 128

= 122

Size of single entry in cache =
1 (V bit) + no: of tag bits + no: of bits for data.

tag = Total main mem addrs − (index + Bo)
no: of bits for

Total cache size = total no: of blocks * size of single entry.

Total data size = total no: of blocks * data / block.

Q) For a cache with 32kB of data with and 8 word blocks, find the no: of blocks in cache?

= $\frac{32 \times 2^{10} B}{8 \times 4 B} = \frac{2^{15}}{2^5} B = 2^{10}$

Q) Assume ~~32 bit~~ 1GB main mem with a cache of 1KB blocks and 4 word block size., find the no: of bits for tag, index and BO.

4 word block size

$= 4 \times 4 = 16 = 2^4$ $\quad 1GB = 2^{31}$

$\therefore BO = 4$

Cache = 1K blocks $= 2^{10}$

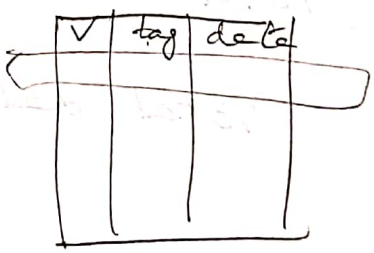
$\therefore$ index $= 10$

$\therefore$ tag $= 31 - (10+4)$

$= 17$

Q) Cache with 8K blocks and 2 words/block find tag, index & BO for a 32 bit address

Q) How many total bits are required for a direct mapped cache with 16KB of data and 4 word blocks assuming 32 bit address.



total bits = no: of blocks x size of each entry.

no: of blocks $= \frac{16\,KB}{4\,word} = \frac{16 \times 2^{10}}{4 \times 4}\,B$

$= 2^{10}$ blocks

index $= 10$

size of each entry = V + tag + data

4 word blocks = 4 × 4 B = 16 B

$= 2^4$

$\therefore$ IBO = 4

$\therefore$ tag = 32 − (10 + 4) = 18

no are row = 4 words (1 word = 32 bits)

$\therefore$ total bits = 4 × 32 = 128

= 1 + 128 + 18 = 147

$\therefore$ total bits = no of blocks * size of each

$= 147 * 2^{10}$

Q) same question as b4 with 64 kB of data & 8 word blocks assuming 30 bit address?

Handling cache miss.

Cache miss – A request for data from the cache that cannot be filled because the data is not present in cache

steps taken on an instr cache miss.

1. Send the original PC value (of the instr not present in cache) (PC−4) to memory
2. Instruct main mem to perform a read op for the requested instr & wait for the mem to complete its access. (wait is equivalent to stall).
3. Once main mem completes the access, write this instr to cache putting data from ~~dat~~ mem in the data portion of cache entry, writing the upper bits of addr (tag) into the tag field and turning valid bit on.
4. Restart instr exec at the 1st step, which will refetch the instr, this time finding it in cache.

Handling writes.

* on a store miss, we have to write data both to cache as well as main mem., else if we write only to cache - cache & mem are said to be inconsistent.

various schemes.

① write through

* simplest
* update both mem & cache at the same time.
* disadv: poor performance coz - writing data to memory - long time - ≈ 100 cc

② write buffer.

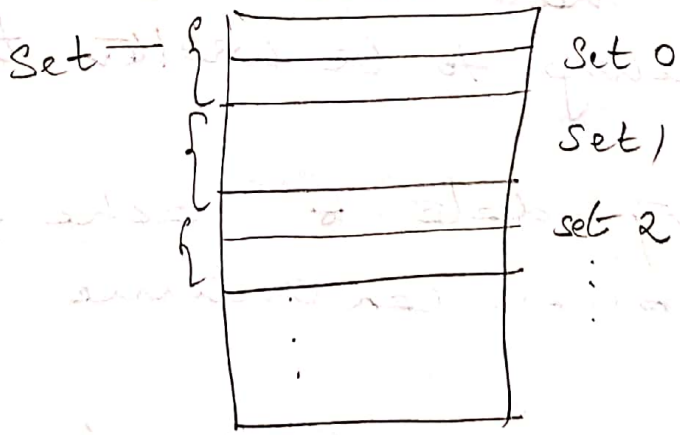
* a queue that stores data while it is waiting to be written to mem.
* after writing data to cache & buffer, processor can continue execution.
* when write to mem is complete, write buffer is cleared.
* If buffer is full when CPU reaches a write CPU must stall until there is an empty pos in buffer.

(3) write back

* handles writes by updating values only to the block in cache.
* modified block is written to mem only when it is replaced.
* good performance but complicated to implement compared to write through.

(II) Set Associative cache

* fixed no: of loc where each block can be placed.

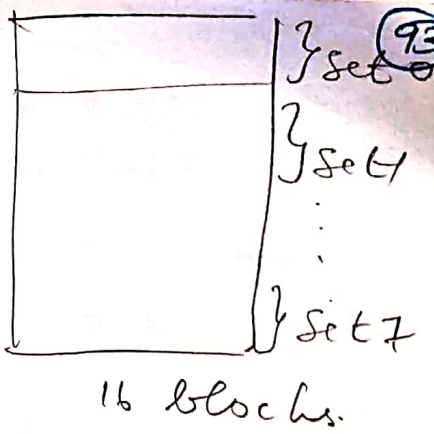


2-way set associative.
every set - 2 blocks

formula - block no: % no: of sets in cache

eg: cache with 16 blocks for a 2-way set associative

16 blocks — 8 sets.

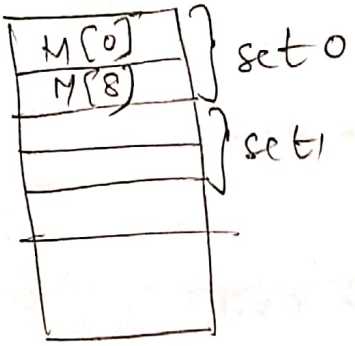


if main mem with 64 blocks.

cache addrs = block addr % no: of sets

M[0]
M[8]
M[16]
M[56]

% 8 = 0 → will go to set 0.



if M[0] comes first, then M[8], no need to replace M[0], it will go to the next slot in set 0.

only if another addrs, say M[16] comes again, we need to go for replacement.

[For replacement — any replacement policy. eg: LRU least recently used]

Four-way set associative — 4 blocks/set.

8 way — 8 blocks/set.

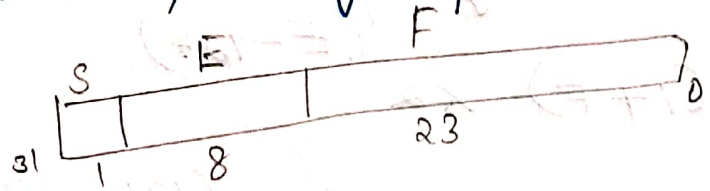
:

fully set associative — all blocks in one set.

→ any mm block addrs can go to any block in cache.

Floating point representation.

IEEE 754 single precision format. (32 bits)



(bias = 127)

Q) −0.75

in binary :- 0.11

scientific notation : -0.11×2^{0}

normalised scia not : -1.1×2^{-1}

gen representation for a single precision no: ?s :

$$(-1)^{S} \times (1+F) \times 2^{(E-\text{bias})}$$

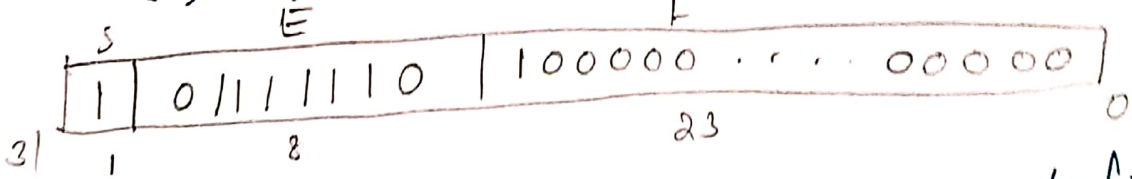
(E − bias) = actual exp

$(E-(\text{bias}) = -1)$

$\therefore E = 126$

$126 - 127$

$(-1)^{1} \times (1+.10000..00) \times 2$



- single precision binary representation

Q) 25.4

binary : 11001.01100

scient not : 11001.01100×2^{0}

normalized not :

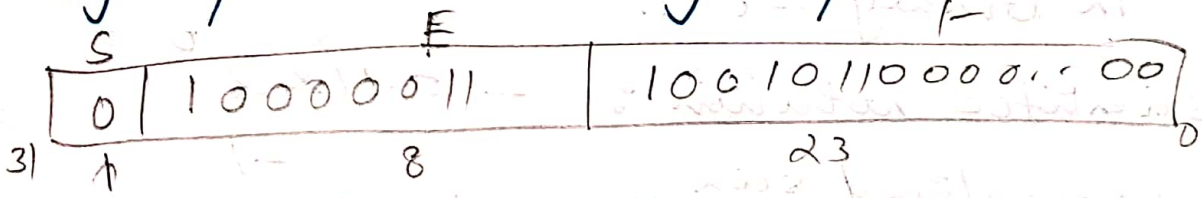
1.1001011×2^{4}

gen. rep: for single precision

$(-1)^{S} \times (1+F) \times 2^{(E-127)}$

$(-1)^{0} \times (1 + 0.100\ 1011) \times 2^{(131-127)}$

single precision binary repns:



Double precision: (64 bits)

(bias - 1023)