main memory → RAM.

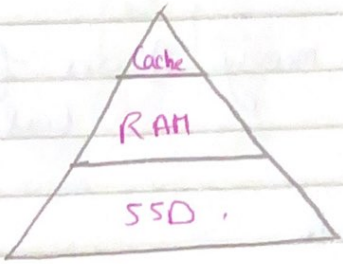
SENDOSS
FILA
JABFAR
DOUSS

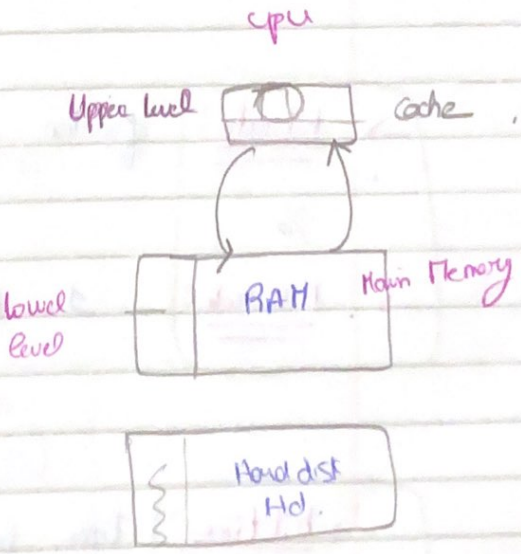
Chapter 5 : IBRAhim hamdan.

→ Principle of localites: states that the program access a relative small portion of their adress space at any instance in time.

- Temporal locality: instruction executed now will most probably be needed in future.
- Spacial locality: code work sequentially.

→ Memory hierarchy:





Sandeep

Jackson

→ **block:** is the minimum amount of information that can be transfered between 2 levels of memory

→ **hit:** If the data requested by the processor is found in Upper level its a hit

yes / No question

→ **Miss:** Not a hit.

Upper [ ] L1 cache.

Lower [ ] L-2 cache.

Upper [RAM]

Lower [HD.]

→ **hit rate:** fraction of memory Accesses found in upper level.

%

→ **Miss rate:** 1 - hitrate

→ **hit time:** time to access upper level of memory hierarchy.

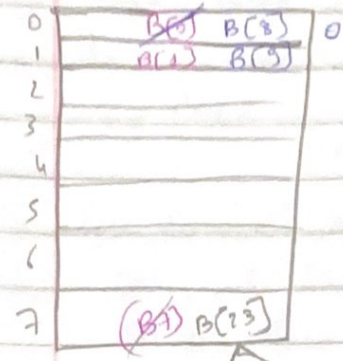
Time

→ **Miss penalty:** The time to replace a block in the upper level with the corresponding block from lower level + the time to deliver the block to the processor.

Ibrahim hamdan is the Jackal of the world.

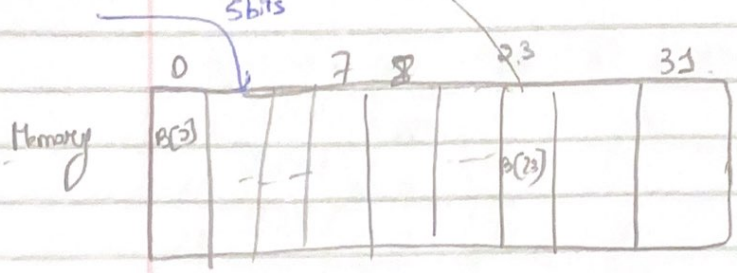
Cache

Direct Mapped cache



Block location in cache = Memory block Address % # of blocks in Cache.

= 0 % 8 = 0
1 % 8 = 1
8 % 8 = 0
9 % 8 = 1
23 % 8 = 7



TAG

bits	index cache
10	111

(23) in binary

TAG bits

Cache

	Valid	TAG	Data
Y	1	10	B[0]
N	0		
N	0		

Direct Mapped caches works by bringing a block from Memory & Applying the modulo operator on it (% # blocks include)

To

Direct Mapped cache:

Each block taken from memory goes into a specific place is cache according to the below location in cache

= Block address (Memory) % # of blocks in cache

Ex: given the below memory reference for a 1-word block cache - 8 - locations & a 32 word Memory. After these references.

Address in Cach		TAG	Hit or Miss
22	22%8 = 6 10110	10	Miss
26	26%8 = 2 11010	11	Miss
22	22%8 = 6 10110	10	Hit
26	26%8 = 2 11010	11	Hit
16	16%8 = 0 10000	10	Miss
3	3%8 = 3 00011	00	Miss
16	16%8 = 0 10000	10	Hit
18	18%8 = 2 10010	10	Miss
16	16%8 = 0 10000	10	Hit
26	26%8 = 2 11010	11	Miss

	V	TAG	Date
0	1	10	MEM[16]
1	0		
2	1	11 10 11	MEM[26] 16 18
3	1	00	MEM[3]
4	0		
5	0		
6	1	10	MEM[22]
7	0		

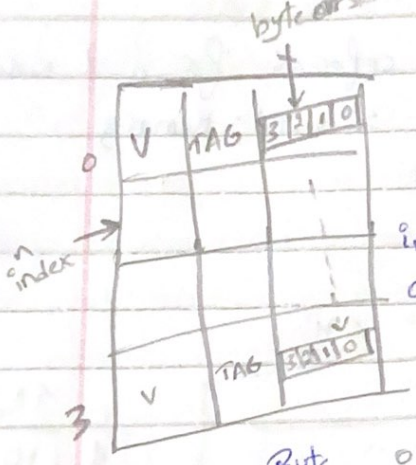
TAG is the remaining bits of Memory address After taking out bits related to cache addressing

Ex: Address → 8 bits
cache → 3 bits
TAG = 5 bits

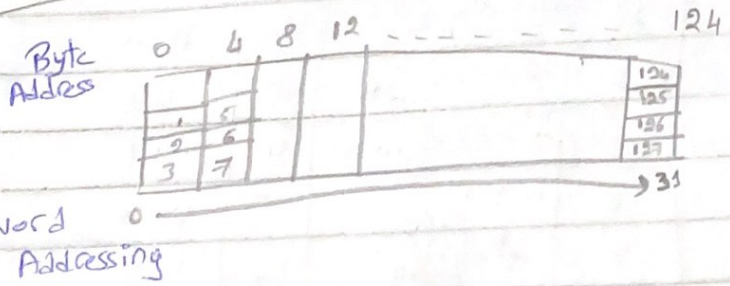
How many blocks did we replace?

We replaced 2 blocks ① MEM [26] by MEM [18]
② MEM [18] by MEM [26]

word Addressing ⟶ 1 word output
byte Addressing ⟶ 1 word
⟶ 1 byte

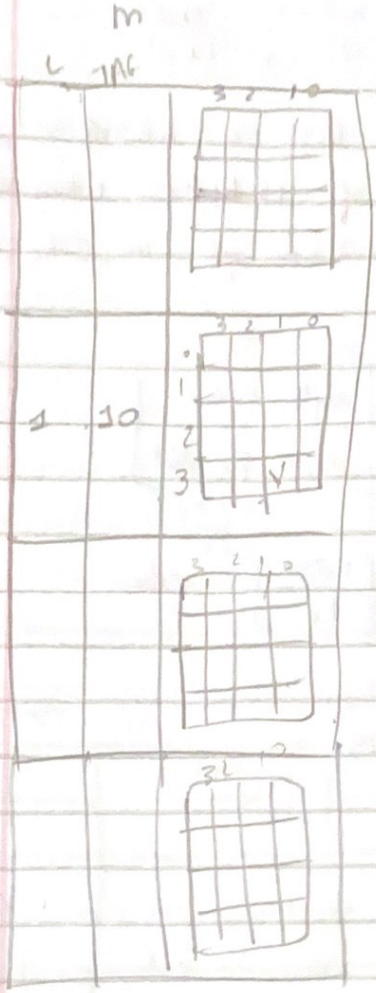


1 word ⟶ 4 bytes
in case of 32 bit word we have byte offset to be 2 bits.



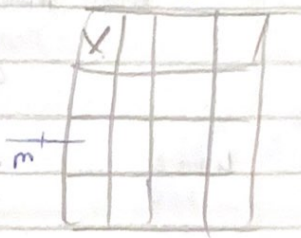
Given that we changed the Memory to byte Address & Kept it 32-words, what the mache of bits needed to Address the memory?

128 bytes ⟶ 7 bits to Address.

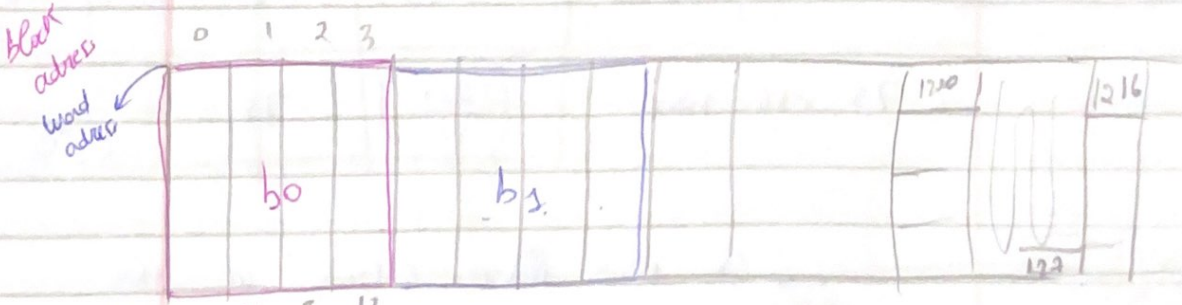


Multi word blocks.

byte offset



1 word
2 word
4
8 pixels of



32 bit → 4 byte → 2 bit
64 bits → 3 bit
128 → 4 bit

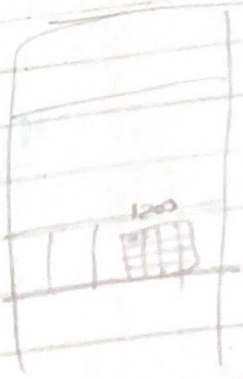
→ Consider a Cache with 64 blocks & a block size of 16 bytes. When does byte Address 1200 map opt in Cache. 32 bit word

Block Address in Memory = $\left\lfloor \frac{\text{byte Address}}{\text{bytes Per block}} \right\rfloor$

4 word block = $\left\lfloor \frac{1200}{16} \right\rfloor$ floor

= 75

75 % 64 = 11



75 × 16 = 1200 $\left(\frac{1205}{16}\right) = 75$

→ The block Address. Contains in this case 16 bytes it ranges from 1200 → 1215

To set the Address of first byte we multiply block Address x # of bytes in block

64 blocks

4 words

32 bits

→ 00000 | 00101 | 01 | 00

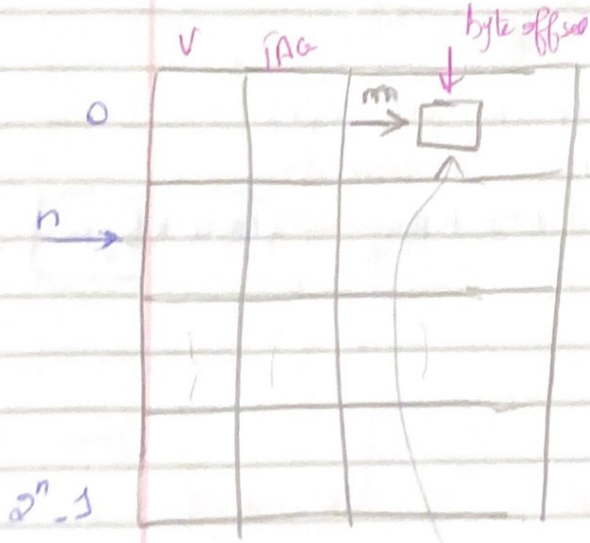
TAG — n — m — byte offset

$2^{14} = 2^{10} \times 2^4$

$= 16$ KB byte

$\frac{16\text{K bytes}}{16 \text{ byte}} = (1\text{K blocks})$

$2^4 \cdot 2^6 = (2^{10})$



32-bit

→ equation for calculating # of bits in Cache

TAG

~~$2^n (2^m \times 32 + (32 - m - n - \text{byte offset}))$~~

Valid

32 bit → 1

$2^n \left(\underbrace{2^m \times 32}_{\text{blocks}} + (\overbrace{32 - m - n - \text{byte offset}}^{\text{TAG}}) + 1 \right)$

→ How many bits are required for a direct mapped Cache with 16 KB & (4 word ⇒ 16 byte block) 32 bit address (2^4)

byte offset = 4 ?

m = ?

$$\frac{16\ \text{KBytes}}{16\ \text{byte}} = 1\text{K block}$$

$2^{10} \Rightarrow n = 10$

$$2^{10}\left(2^7 \times 32 + (32 - 10 - 4 - 2) + 1\right)$$

$$= 2^{10} \times 147 = 157\ \text{KBits} = 18.4\ \text{Kbyte}$$

2^m → word (4 bytes)

Cache sizes:

32 bits → $2^n \left(32 \times 2^m + (32 - n - m - 2) + 1 \right)$

- 2^n: number of blocks
- 32×2^m: block size in bits
- $(32 - n - m - 2)$: TAG
- $+2$: byte offset
- $+1$: valid bit
- 1 for each block.

64 bits → $2^n \left((64) \times 2^m + ((64) - n - m - (3)) + 1 \right)$

- 64×2^m: data
- $64 - n - m$: address
- 3: 8 bytes in 64 bits
- $+1$: data

ex: 64 blocks → $n = 6$ (blocks)

2 words → $m = 1$ (word)

$m = 2^1 = 2$

32 bits data:

$= \frac{16\text{K bytes}}{4 \times 4 \text{ Bytes}}$

word: 4 → 2^2

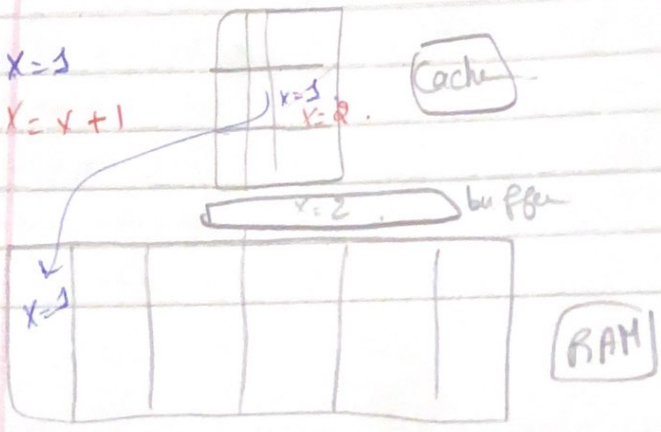
64 bits

$\frac{16\text{K Byte}}{4 \times 8 \text{ bytes}}$

- m → same
- n → ↓

→ One comparator is needed to compare TAG bits found in adress with the one pointed by the index in cache, if validdes is one then we have a bit.

→ Write Places?



→ Write through

Processor ask for x

① it reads from RAM

② Change the value, now we have inconsistency between RAM & Cache.

③ Once changed, it will also update RAM.
This means each write will take time to update RAM.

→ Why write through without buffer is not a good idea ??

ex: Suppose we have 10% store CPI without cache misses is 1.
Time to write to RAM is 100 cycles
What would be the new CPI with a write through Cache with No buffer

$$CPI = 1 + 10\% \times 100 \text{ cycles} = 11$$

→ Write buffer will not be beneficial if the rate the Processor generate writes (SW) is more than the rate, the buffer can empty its content in RAM.

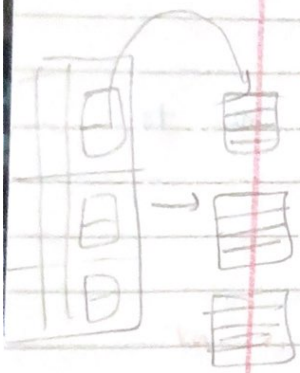
→ when buffer is full & we have on SW ⇒ The Processor will stall.

block size ↗ — Rate ↘ ? Pen ↗

↗ blocksize — miss Rate ↘ — Pen ↗

→ Write back? instead of writing to RAM write each SW, I keep the content changed in cache until a block will replace the current content.

a Dirty is used to indicate if a block has its content modified.



→ Assume you have a fixed cache size ex. 64 blocks & it is a good idea to increase block size.

+ if we increase block size this means ⇒ more words in cache block. based on sperial locality, this will ↘ decrease miss Rate however this will ↗ increase miss penalty.

⇒ increase block size will lead to less locations in cache to be searched & higher competition between blocks to contend for ⇒ block replacement will be high this making miss rate higher.

write allocate — تتعب بل تجيح cache/RAM.

No write — تشيل مه فوق تحطو كت.

→ Write Allocate?

Consider a miss in a write through cache

in this case ⇒ fetch the blocks from RAM, Allocate it in cache & perform the write.

No write allocate the processor write an entire block of data in memory but not in cache.

* This is useful when O.S needs to zeros a page in memory.

⇒ Measuring Cache Performance technique

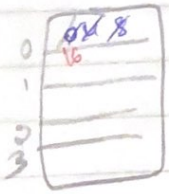
* Reduce miss Rate : reduce the Probability that two different memory blocks contend to the same cache location.

* Reduce miss penalty by using multi level cache

CPU time is divided into clock cycles that the CPU spends executing the program & clock cycles that CPU spends waiting for the memory system.

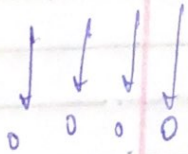
Cache Performance.

- miss Rate: Cache design reduce number of blocks contending for the same location



- miss penalty: Multi level cache.

0, 4, 8, 16, 0, 4, 8, 16.



% 4 module

→ CPU Time = CPU Execution cycles + Memory stall cycles

CPU Execution cycles → CPI

Memory stall cycles → Time waiting for cache to retrieve Data from Memory

→ Memory stall cycles = Read stall + Write stall

Read stall = $\frac{\text{Reads}}{\text{Program}}$ × miss Rate × miss penalty

Same

⇒ Perfect cache is cache with no Misses

Same

Write stall = $\frac{\text{Writes}}{\text{Program}}$ × miss Rate × miss penalty + Write buffer stalls

(negligable) → 0

D. cache → Data memory stalls = $\frac{\text{Memory accesses (W/SW)}}{\text{Prog}}$ × miss rate × miss penalty

I. cache → Instruction Memory stall = $\frac{\text{Instruction}}{\text{Prog}}$ × misses/instruction × miss Penalty

Assume that the miss Rate of an instruction Cache is (2%) & miss rate of data cache is (4%).

I cache 2% D-cache 4%

If a processor has a CPI of (2) Without Memory stalls & miss penalty is 100 cycles.

CPI = 2

Determine how much faster a processor would run with a perfect cache that never misses, Assume the frequency of All loads & stores is 36% (load/store).

→ Instruction cache stalls = $\frac{All}{I} \times 2\% \times 100$ cycles

= 2I cycles

→ Data cache stall = $\frac{lw/sw}{I} \times 0.36 \times 0.04 \times 100$.

= 1,44I cycles

→ CPI Normal with cache = $\frac{\text{perfect}}{2} + \frac{\text{CPI}}{\frac{2I}{I}} + \frac{\text{CPI}}{\frac{1,44I}{I}}$.

= 5,44

% faster = $\frac{5,44}{2}$ = 2,72 Time

* Perfect → (ما في miss)

⚠ → If we speed up the Processor

CPI 2 → 1

→ What will be the effect on the overall system

CPI with cache = 1 + 2 + 1,44
= 4,44

slower than Perfect Cache.

% slower = $\frac{4,44}{1}$ = 4,44.

CPI
5,44 > 4,44

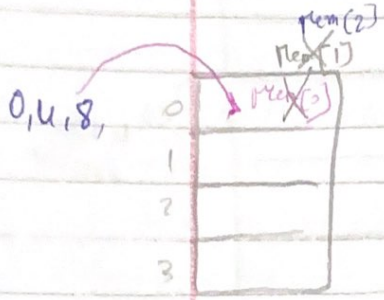
بس
قلل ال%
يعني ع تانية
وقت ال stalls
قلل

AMAT : Average Memory Access Time?

→ AMAT = time for a hit + Miss rate × Miss Penalty.

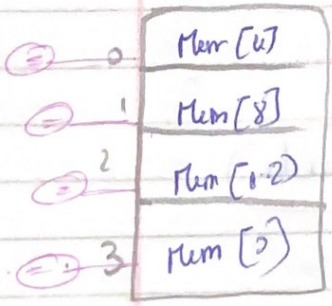
Direct Mapped Cache:

- Every block has a specific location
- Very fast in Terms of addressing
- miss Rate ↗↗



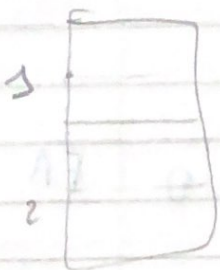
→ Block address in Cache = Block Adress in Mem. % # of blocks in Cache

⇒ instead of every block going into a specific location why not All blocks can go in Any location.



0, 4, 8, 12.
4, 8, 12, 0.

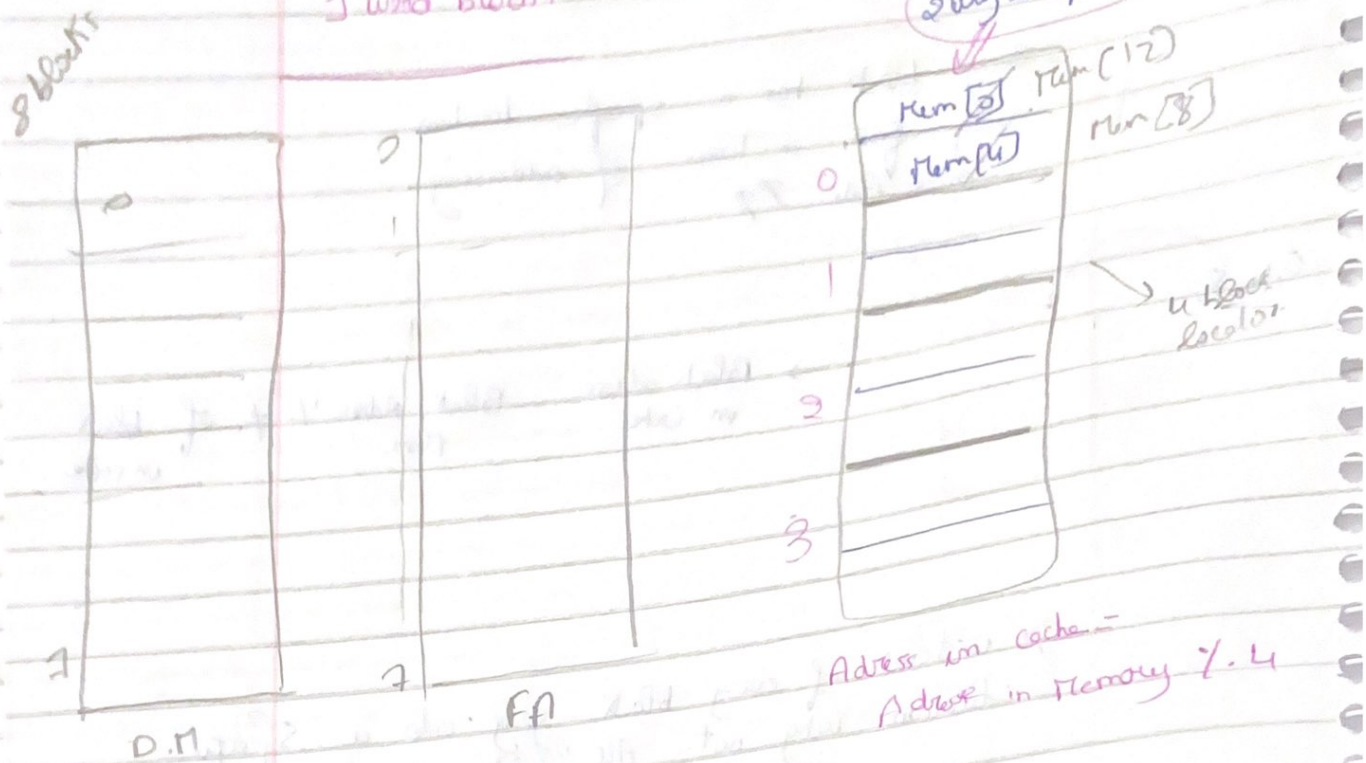
Full Associative.



8 way — fully
0 ⟷ direct Mapped

1 word block?

0, 4, 12, 8.



Adress in cache :-
Adress in Memory % 4