✱ Cache Memory

→ Locality of Reference:

During execution of a given program memory references tend to cluster.

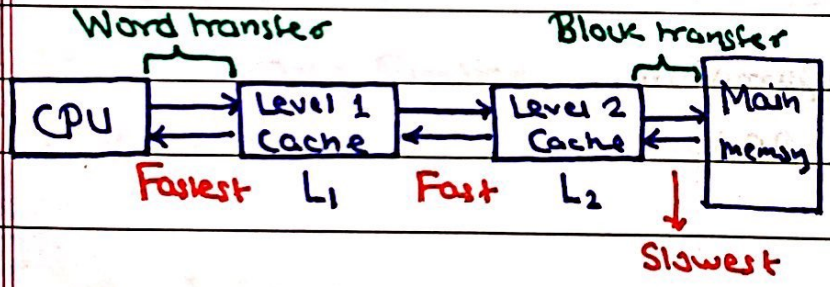
eg: Sum of elements of an array. (# contiguous memory locations)

→ Cache

① Small amount of fast memory

② Between CPU & Main memory

③ Maybe located on CPU chip



✱ Memory Hierarchy

→ Registers
CACHE (L_1, L_2)
Main Memory
Disk Cache
Disk
Optical
Tape

↓ cost per bit
↑ capacity
↑ access time
↓ freq of access time by the processor

Inboard
Outboard
Offline

* Cache Operation:

① CPU req contents of memory location.

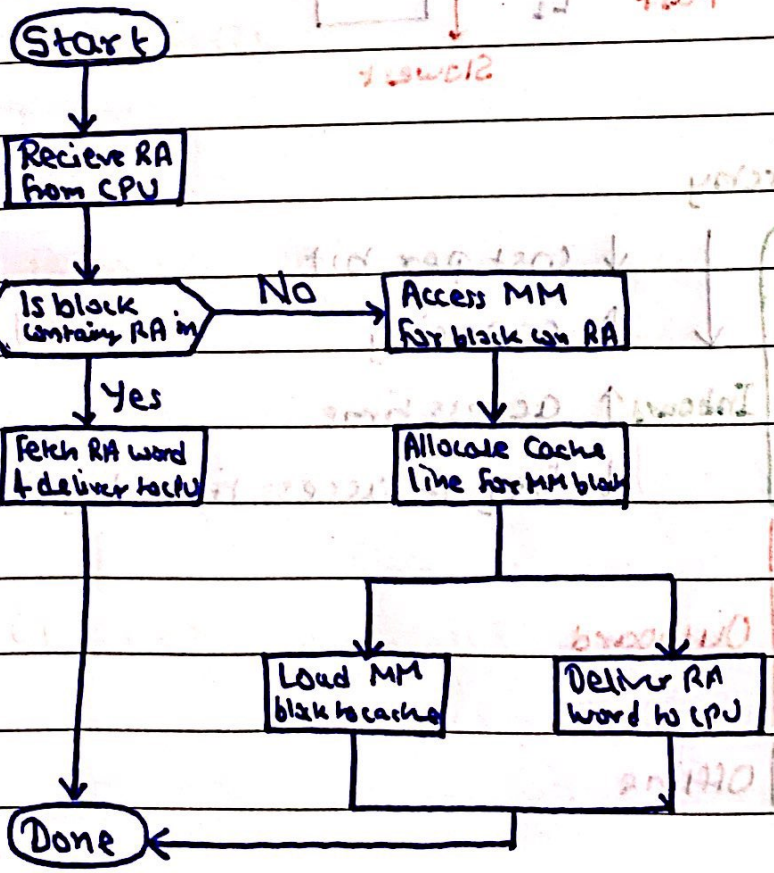
② Check Cache for this data
(If present get from Cache) [Cache Hit]

③ If not present [Cache Miss]
Read required BLOCK from main memory to cache
then deliver from Cache to CPU

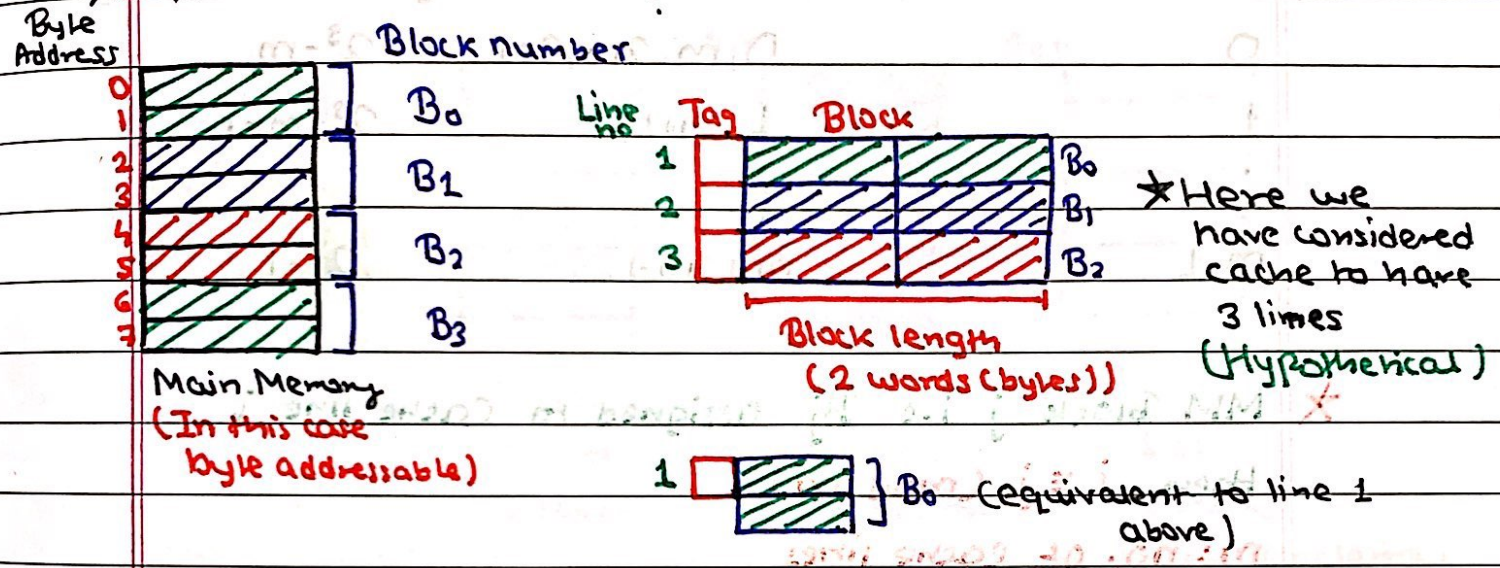
* Cache includes tags of blocks of main memory.
in each cache slot.

* Cache Read operation:

→ Read address (RA)



★ What is cache line?



★ Cache Design

1. Cache Size
2. Cache Mapping Function
3. Replacement Algo
4. Write Policy
5. Line Size
6. No. of caches (Levels)

① Direct Mapping

① Each Block of main memory maps to only one cache line

② Address is in 2 parts

Least sig w bits identify unique word (block size = 2^w)

Most sig s bits specify one memory block

Tag : s-r r : Cache line

★

s-r	r	w
8	12	4
Tag	Line	word

24 bit address

16 byte block (2^4)

No two blocks in same line have same tag field

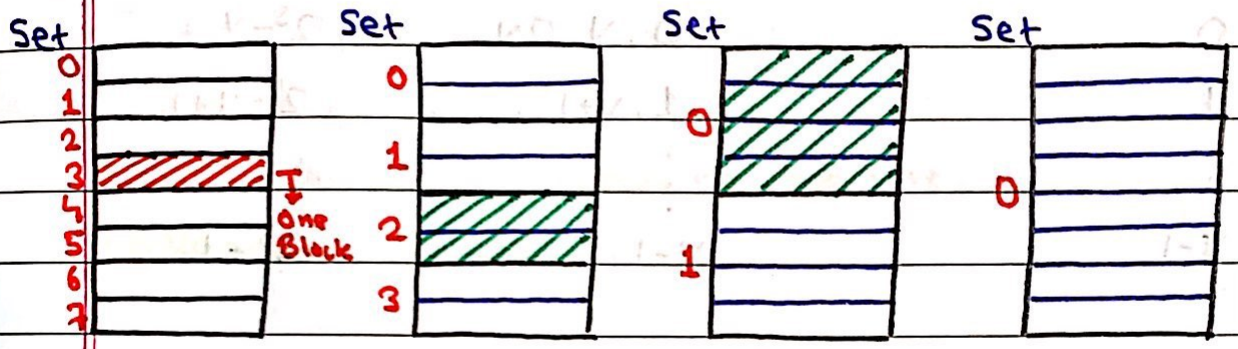
Cache line	MM blocks assigned
0	$0, m, 2m, 3m, \ldots, 2^s - m$
1	$1, m+1, \ldots, 2^s - m + 1$
⋮	⋮
m-1	$m-1, 2m-1, \ldots, 2^s - 1$

* MM Block j i.e Bj assigned to Cache line i then $i \equiv j \pmod{m}$
m: no. of cache lines

1. Address length = (s+w) bits (Addressable units 2^{s+w} words or bytes)
2. Size of tag = (s-r) bits
3. Block size (MM) = line size (Cache) = [redacted] = 2^w words
4. No. of lines in cache = $m = 2^r$
5. No. of blocks in MM = 2^s

* Simple
* Inexpensive
* Fixed location for a given block (MM)
If program accesses 2 memory blocks that map to same line repeatedly, cache misses are very high (i.e memory blocks with index same value modulo m)

② Associative (Set Associative)



1-way	2-way	4-way	8-way
8 sets	4 sets	2 sets	1 set
1 block each	2 blocks each	4 blocks each	8 blocks
(Direct mapped)			(fully associative)

ⓐ Fully Associative:

① MM block can load into into any line of cache

② Memory address as tag & word

③ Tag uniquely identifies the block of memory
(Every line tag is examined for match)

④ Catch searching gets expensive

ⓑ Set Associative

(K lines/blocks per set ⇒ k-way set associative mapping)

① Cache divided into v sets

② Each set has k lines

③ No of lines in cache $m = v \times k$

④ Bj goes to set i
when $i \equiv j \pmod{v}$

Set no.	MM blocks held
0	$0, v, 2v, \ldots, 2^s - v$
1	$1, v+1, \ldots, 2^s - v + 1$
:	:
$v-1$	$v-1, \ldots, 2^s - 1$

tag	set	word
$s-d$	d	w

1. Address length = $(s+w)$ bits
2. Addressable units = 2^{s+w} words
3. Block size = line size = 2^w words / bytes
4. No. of blocks in MM = 2^s
5. No. of lines / set = k
6. No. of sets = $v = 2^d$
7. No. of lines in cache = $kv = k \times 2^d$
8. Size of tag = $(s-d)$ bits

classmate
Date
Page

★ Replacement Algo:

① Direct mapping:

① No choice

② Each block only maps to one line

③ Replace that line

→ Write Policy:

① Musn't overwrite a cache block unless MM is updated

② Multiple CPU's may have individual caches

③ I/O may address MM directly.

① Write through

→ All writes go to MM as well as cache

→ Lots of traffic

→ Slows down write

→ Remember bogus writes through caches

(Multiple CPUs can monitor MM traffic to keep local cache updated)

② Write back

→ Updates initially made in cache only

→ Update bit for cache is set when update occurs.

→ If block to be replaced write to MM only if update bit set.

(I/O must access main memory through cache)

② Associative mapping

→ Least Recently Used (LRU) (which one block is LRU)

→ FIFO (replace the block which in cache for longest)

→ Least frequently used (replace block with fewest hits)

→ Random

* LRU

① Consider 4 line set in cache.

→ Control bits:

Tag bits

* 2-bit counter for each line (to track LRU block)

d_bit : Dirty Bit

f_bit : for occupied bit

② Initially reset all the counters (d_bit & f_bit)

③ Cache hit occurs:

→ set counter value to 0 of this cache line.

→ for other counters if value less than the referenced line increment the counter value.

else donot change counter value.

Cache miss:

① Set is not full

set counter value of cache line to 0

set f-bit to 1.

increment counter of other lines whose f-bit is 1

② Set is ~~not~~ exactly full

→ The line with ↑ counter value is removed, write back it d-bit is 1.

→ new block is transferred to this line

→ reset d-bit & counter

→ Other counter values ↑ by 1

★ External Memory:

→ Performance:

① Access Time: (Time req between presenting address & getting valid

② Memory Cycle Time: Access + Recovery

③ Transfer Rate: (Rate at which data can be moved)

*Recover: memory to recover before next access

→ Physical Types

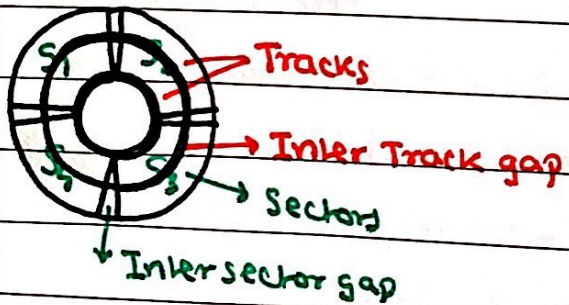
① Semiconductor : RAM

② Magnetic : Disk & Tape

③ Optical : CD & DVD

★ Disk Data Layout

a) Constant Angular Velocity



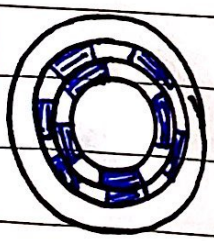
★ Each track has same width as head.

★ Each track same no. of bits

★ ∴ bit density ↑ from outer ring to inner (track)

(Hence ∃ wastage of space)

b) Multiple zone recording



★ Each zone fixed bits per track (Complex circuitry)

* Head
  1. Fixed (one head per track)
  2. Movable (one head for complete disk side)

* Removable / fixed

→ Single / double sided

→ Single / multiple platter

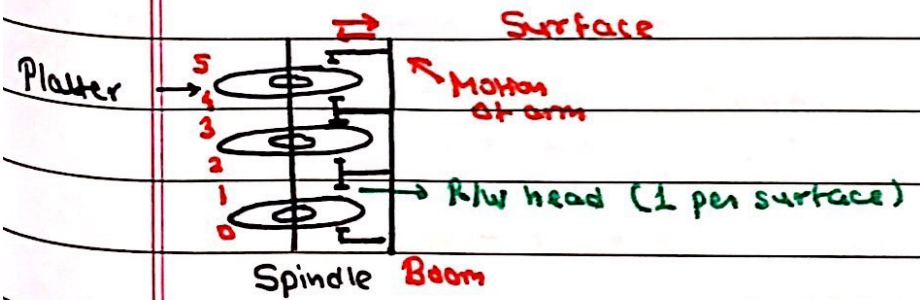
→ Head Mech
  - Contact (Floppy)
  - Fixed gap

* Multiple Platter

→ One head per side

→ Heads joined & aligned

→ Aligned tracks on each platter form a cylinder



* Disk addressing:
  1. Sector no.
  2. Track no.
  3. Surface no.

Date
Page

✱ Speed

① Seek time (T_s)

→ Moving head to correct track

② (Rotational) Latency ($\frac{1}{2r}$)

→ Moving to correct sector

Access Time = Seek time + Latency

③ Transfer Rate (T)

✱ Transfer Time T (Transfer Rate)

$T = b/rN$

b: No. of bytes to be transfered

N: No. of bytes on the track

r: rotational speed (revolution per second)

$\therefore$ Total time $= T_s + 1/2r + T$

$= T_s + 1/2r + b/rN$

* Performance Issues:

→ Execution Time

① Elapsed Time = CPU time + wait time (I/O, other programs)

② CPU time = user CPU time + system CPU time

* Our focus: user CPU time (time spent in executing the lines of code that are in our program)

* For a machine X:

① Performance $_X$ = 1/Execution time $_X$

* $\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Cycles}}{\text{Program}} \times \frac{\text{Seconds}}{\text{cycles}}$

* Clock freq = cycles/seconds (Clock rate)

* Cycle time = 1/clock freq

* Performance equation:

$\frac{\text{Seconds}}{\text{Program}} = \frac{\text{Cycles}}{\text{Program}} \times \frac{\text{Seconds}}{\text{cycle}}$

∴ CPU execution time = CPU clock cycles x Clock cycle time for program

* CPI = cycles / instruction

* MIPS = milions of instructions / second

★ Performance Equation 2:

CPU execution time = Instruction count × avg. CPI × Clock cycle time

→ Cache

① Hit rate : Fraction of memory access that are hits

② Miss rate = 1 − Hit rate

③ Hit time : time to determine access is a hit + time to access & deliver the data to CPU

④ Miss penalty: time to determine access is a miss + time to replace the block in upper level of cache + time to deliver data to CPU.

EXTRA time

★ Avg. memory access time = Hit time + (miss rate × miss penalty)

Memory stall cycles = memory accesses × miss rate × mispenalty

CPU time = (execution cycles + memory stall cycles) × cycle time