Cache memory

→ we know that cpu is fastev than main memory and the main memory is fastev than secondary memory

if cpu executes 1 instruction in 10^{-12} sec

but main memory speed is 10^{-9} sec then the speed of cpu falls down to speed of main memory

→ So we intvaduced registevs, they ave having small size compared to main memory but fastev than main memory But they ave vevy costly so we introduced cache memory which is cheapev to registevs and fastev than main memory.

cpu is having diffevent levels of memory

i.e Registevs as level one memory
cache as level two memory
main memory as level thvee memory
secondavy memovy as level fouv memovy.

we know the Concept of paging in that process is present in secondavy memovy and a pavt of it is present in main memovy and if cache is having some pavt of the process then cpu can fetch the instructions fvom the cache and execute the pvocess.

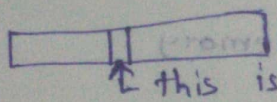
if the cpu gets what it need in cache then it is called cache hit and if not present then it is called cache miss then cpu must fetch from main memory. Now the time taken fov cpu to fetch from the main memovy and sevvice the cache (veplace with newly fetched instvuctions) and execute them is called the "miss latency"

if the part of the process which cpu is looking for is not present in main memory then it is called the "Page fault"

Like page table which shows what is present in page in cache also "Tag directory" is present which shows what are the elements present in cache.

As we know that the size of the cache small compared to main memory now the question is which part in the main memory will be stored in the cache.

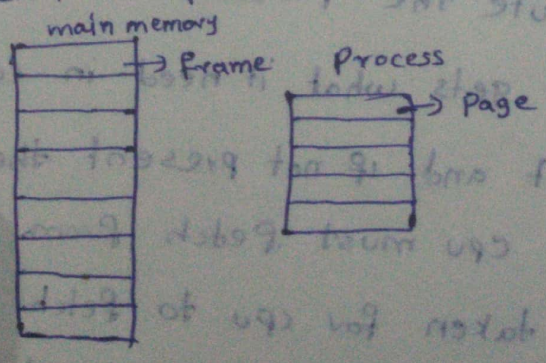
For this temporal locality says that whatever we need now in the page will be present in near to the recently used.



the entire page into cache instead of that bit because According to temporal locality the data needed in the future will be present near to the bit.

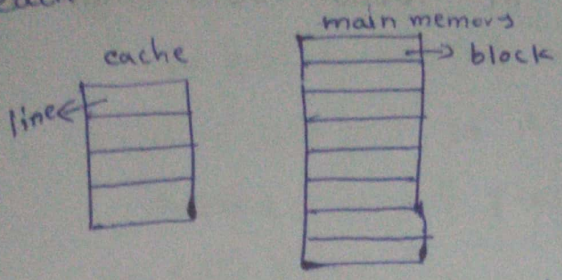
Direct mapping:

like in paging the main memory is divided into frames and process is divided into ~~frames~~ Pages and each page can be fit into one frame



so like wise the in cache main memory is divided into blocks and cache is divided into lines, and

each block can fit into one line



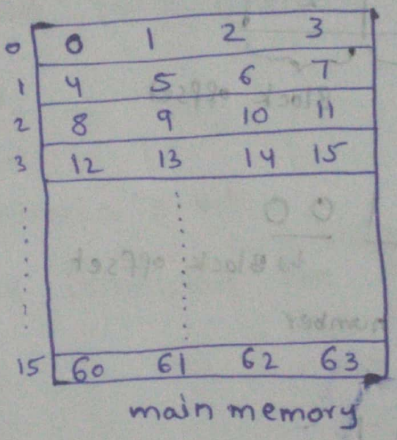
"word" is the unit that we use here instead of "Byte"

Let main memory size = 64 words, block size = 4 words

cache size = 16 words

no of blocks = $\frac{64}{4} = 16$

no of lines = $\frac{16}{4} = 4$



totally 64 words so $2^6 = 64$

we can represent each word with 6 bits

Suppose cpu wants ~~to~~ word '5'

000101 then 0001 [01]

↳ Block offset

first check block number i.e '1' [0001] 01

↳ Block number

in block '1' check '01' i.e '5' so this is how cpu gets what it wants.

if cpu wants word '10'

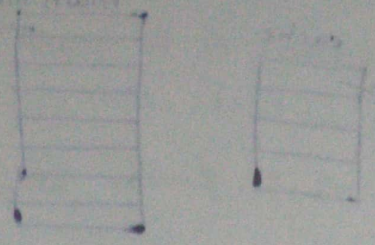
0010 [10]

block '2' ↳ element '3'

now we will see how the blocks are stored in the cache memory

we will fill the cache in round robin manner

0	0	4	8	12
1	1	5	9	13
2	2	6	10	14
3	3	7	11	15

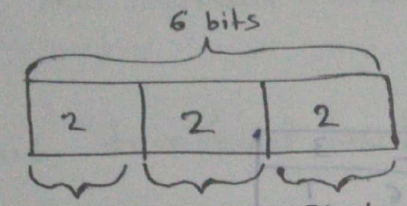


i.e the blocks 0, 4, 8, 12 should always goto line '0'

the blocks 1, 5, 9, 13 should always goto line '1'

the blocks 2, 6, 10, 14 should always goto line '2'

the blocks 3, 7, 11, 15 should always goto line '3'



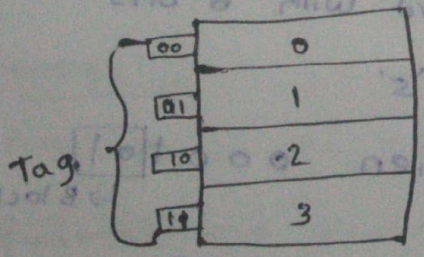
Ex: 0 0 0 1 0 0

↳ Block offset

↳ Block number

0 0 0 1

↳ line no



the msB two bits represents the tag number

if cpu wants 01 01 01 then first it cheks the cache memory if it is present or not

in cache it checks first line no i.e (01) and goes to it and checks tag no i.e (01) then goes to the block offset this is how the cpu checks in the cache.

10 01 11

cpu goes to line (01) and checks the tag

it is not (10) so it thinks that required block number is not present in cache memory

so cache miss

Direct mapping Problem:

Here Conversions are very important

if byte addressable then

128 KB

$= 2^{17}$ Bytes

so Physical address = 17 bits

if word addressable, 1w = 4 bytes

128 KB $\cong$ 32 K w

$= 2^{15}$ w

so physical address = 15 bits

So based upon the Conversion the no of bits representation changes.

Q)

	mm size	cache size	Blocksize	tag bits	tag directory size
1)	128 KB	16 KB	256 B	?	?
2)	32 GB	32 KB	1 KB	?	?
3)	?	512 KB	1 KB	7	?
4)	16 GB	?	4 KB	10	?
5)	64 MB	?	?	10	?
6)	?	512 KB	?	7	?

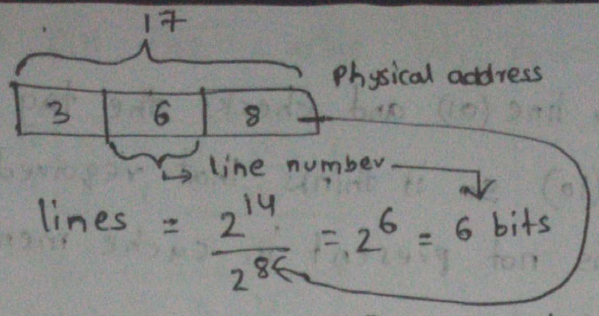
Assume that memory is byte addressable

Soln: (1) Given mm = 128 KB = 2^{17} B

Cache = 16 KB = 2^{14} B

Block size = 256 B = 2^{8} B

we know



no of lines $= \frac{2^{14}}{2^8} = 2^6 = 6$ bits

$17 = 8 + 6 - 2$

$\Rightarrow \boxed{x = 3}$

line (or) Block offset $= 2^8$ B $= 8$ bits

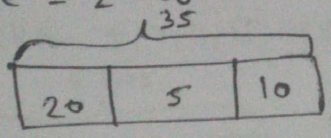
tag bits $= 3$ bits

tag directory size $= 3 \times 2^6$

(2) Given mm $= 2^{35}$ bytes

cache $= 2^{15}$ bytes

so



no of lines $= \frac{2^{15}}{2^{10}} = 2^5 = 5$ bits

So tag bits $= 20$ bits, tag directory $= 2^5 \times 20$

(3) Let mm $= x$ bits

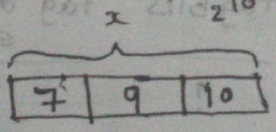
tag size $= 7$

cache $= 2^{19}$ B

no of lines $= \frac{2^{19}}{2^{10}} = 2^9$

tag directory $= 7 \times 2^9$

so



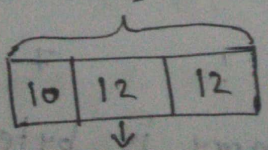
So $x = 26$ bits

(4) Given mm $= 2^{34}$ ~~Bytes~~ bytes

Block size $= 2^{12}$ bytes

cache $= x$

tag bits $= 10$

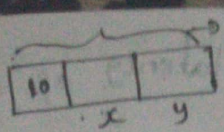


$x = 34 - 22 = 12$ bits

So cache size $= 2^{12} \cdot 2^{12} = 2^{24}$ bytes

(5) Given mm $= 2^{26}$ Bytes

tag bits $= 10$



$x + y = 16$

So cache size be 2^{16} Bytes

Block size, tag directory size cannot be guessed

6) same Block size, tag directory size cannot be guessed.

Direct mapping Hardware implementation :

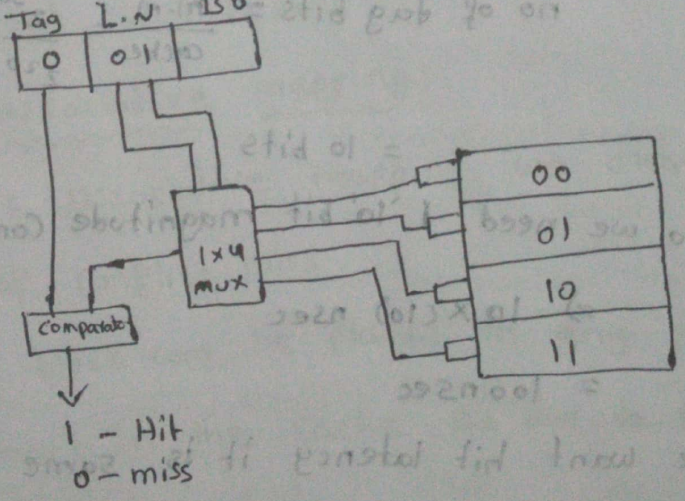
the time taken to check if required data is present in the cache is known as hit latency

the time taken to service the cache with required data is known as miss latency.

now we will see how to find that required data is present in the cache or not

suppose physical address is generated by CPU and no of tag bits = 1

now a multiplexer is connected to the tag bits and selection lines are given from line number



if the line no is given as selection lines then the multiplexer will process the o/p with is a tag number now the tag number of the cache is compared with tag no generated by CPU

then if both are same then it is a hit and if not there is a miss

So the time taken = latency of mux + latency of comparator

if the tag bits are 'two' then we need two mux's

if there are k tag bits then we need

k multiplexers and

1 k bit comparator

even if k mux's are present all will work in parallel so the latency of all the mux's will be taken as single mux latency.

a) Given

main memory = 1 GB = 2^{30} Bytes

Cache memory = 1mB = 2^{20} Bytes

Comparator latency = 10 kns

then comparator delay = ?

we know that

$$\text{no of tag bits} = \frac{m.m}{\text{cache}} = \frac{2^{30}}{2^{20}} = 2^{10}$$

$\therefore$ = 10 bits

so we need 1 '10' bit magnitude comparator

$\Rightarrow$ 10 x (10) nsec

= 100 nsec

if we want hit latency it is same as the comparator delay so hit latency = 100 nsec

if multiplexer delay is ignored then we also ignore it when we solve the problems.

** in direct mapping only one comparator is used.

*

Disadvantage of direct mapping:

Let n be the line no of the cache

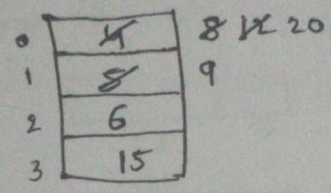
Also there are 4 linespresent in the cache

if m is the block number present in the main memory then

the line number at which the block number will be present is

$n = m \bmod 4$ (or) $n = m \,\%\, 4$

Let 5, 6, 4, 8, 9, 12, 15, 20 are the block numbers then in cache



even though line '3' is free for a long time the block numbers 8, 12 did not go there because at starting we discussed that 0, 4, 8, 12 should goto only line '0' so this is the disadvantage of direct mapping.

This is called Conflict miss

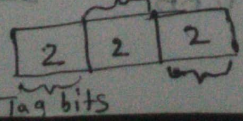
Introduction to associative mapping:

in this associative mapping we cleared the disadvantage of conflict miss

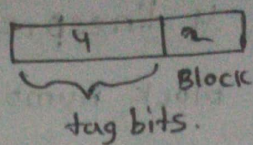
i.e any block can be placed in any line which is available in the cache so due to this the Conflict miss donot occur

but tag bits need to be increased because in case of direct mapping only few are able to fill in one line so we need less bits to represent what is present but now any block can place any where so more no of tag bits are required

in direct mapping:

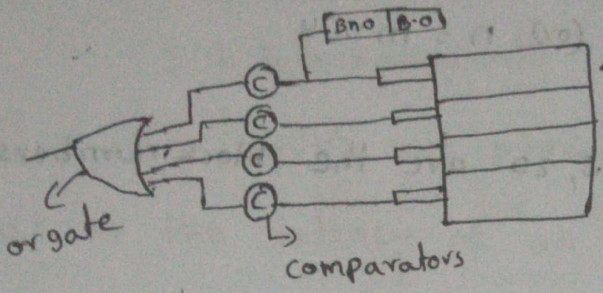


in associative mapping:

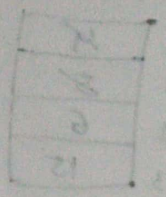


Due to the increase in tag bits the load on the comparators increases so latency will increase.

for this we use move comparators.



with increase in no of lines the no of comparators also increase



② Given main memory = 32 GB

Block size = 32 kB

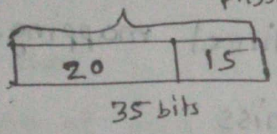
Tag bits = K

Delay in Comparator = 10 × K ns

Delay in or gate = 10 ns

then latency = ?

we know



so total no of tags = 20 bits

Delay in Comparator = 10 × 20 = 200 ns

total latency = 200 + 10 = 210 ns.

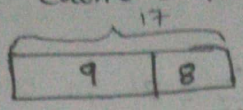
Problems on associative mapping:

②

	Main memory	cache size	Block size	Tag size	Tag directory size
①	128 KB	16 KB	256 B	9 bits	?
②	32 GB	32 KB	1 KB	?	?
③	—	512 KB	1 KB	17	—
④	16 GB	—	4 KB	—	—
⑤	64 MB	—	—	10	—
⑥	—	512 KB	—	7	—

① Given mm = 128 KB = 2^{17} B

Cache = 16 kB = 2^{14} B



no of lines = $\frac{2^{14}}{2^8} = 2^6$

So tag bits = 9

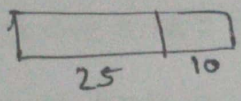
Tag directory size = 9 × 64

② Given mm = 2^{35} B

Cache = 2^{15} B

So Tag bits = 2^{20} = 20 bits

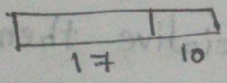
no of lines = $\frac{2^{15}}{2^{10}} = 2^5$



Tag directory size = 25×2^5

③ Given Cache = 512 kB = 2^{19} B

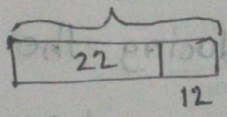
no of lines = $\frac{2^{19}}{2^{10}} = 2^9$



so main memory = 2^{27} = 128 mB

tag directory = 17×2^9

④ Given main memory = 2^{34} B



so no of bits in tag = 22

Cache size, tag directory size cannot be guessed.

⑤ Cache size, tag directory size cannot be guessed

Set Associative mapping:

in set Associative mapping we divide the no of lines in the cache into sets

if set size = 2 lines then it is called two way set associative

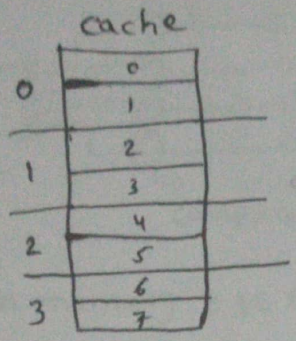
Let Main memory = 64 B

Cache size = 32 B

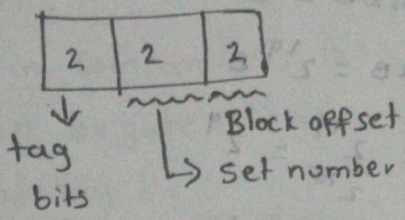
Block size = 4 bytes
set size = 2 lines
2 way set associative

total number of lines = $\frac{C \cdot S}{B \cdot S}$ = $\frac{32}{4} = 8$

sets = $\frac{\text{no of lines}}{\text{set size}} = \frac{8}{2} = 4$



in set Associative mapping



if cache is k-way set Associative then we need k-comparators.

the main intention of introducing the set associative mapping is to decrease the number of comparators.

Problems on set Associative mapping:

Q)	main memory	cache size	Block size	Tag bits	Tag directory size	set associative
①	128 KB	16 KB	256 B	—	—	2 way
②	32 GB	32 KB	1 KB	—	—	4 way
③	—	512 KB	1 KB	7	—	8 way
④	16 GB	—	4 KB	10	—	4 way
⑤	64 MB	—	—	10	—	4 way
⑥	—	512 KB	—	7	—	8 way

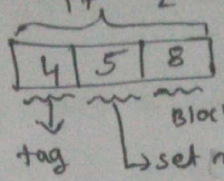
① Given main memory = 128 KB = 2^{17} B

Cache size = 16 KB = 2^{14} B

no of lines = $\frac{2^{14}}{2^8} = 2^6$

it is 2 way set associative

so no of sets = $\frac{2^6}{2} = 2^5$



Tag directory size = 4×2^6 bits

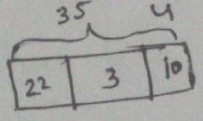
② Given main memory = 2^{35} B

Cache size = 2^{15} B

no of lines = $\frac{2^{15}}{2^{10}} = 2^5$

no of sets = $\frac{2^5}{4} = 2^3$

so



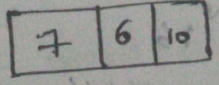
So tag directory size = $2^5 \times 22$

③ Given cache memory = 2^{19} B

no of lines = $\frac{2^{19}}{2^{10}} = 2^9$

it is 8 way set associative

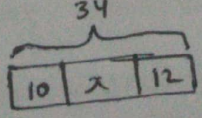
So no of sets = $\frac{2^9}{2^3} = 2^6$



so mainmemory = 23 bits

Tag directory size = $2^9 \times 7$

④ Given main memory = 16 GB = 2^{34} B



so no of sets = 12

no of lines = $12 \times 4 = 2^{14}$

Cache size = no of lines x block size

$= 2^{14} \times 2^{12} = 2^{26}$

Tag directory size = $2^{14} \times 10$

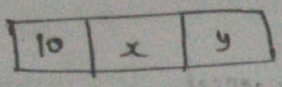
(5) we know that set Associative mapping

Cache size = no of sets x $\frac{\text{no of lines}}{\text{set}}$ x line size

so

Cache size, Tag directory size cannot be guessed.

but



so $26 = 10 + x + y$

so $x + y = 16$

cache size = $2^x \times 4 \times 2^y$

$= 2^{x+y+2}$

$= 2^{16+2} = 2^{18}$ Bytes

Cache size = 256 KB

(6) we know

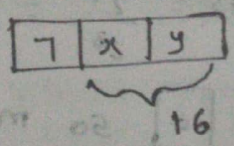
Cache size = sets x $\frac{\text{no of lines}}{\text{set}}$ x line size

$= 2^x \times 8 \times 2^y$

$2^{19} = 2^{x+y+3}$

$\Rightarrow x + y = 16$

so



Physical address size = $7 + 16 = 23$ bits $= 2^{23}$

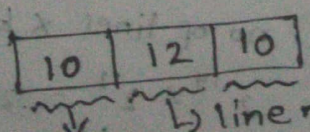
Comparing all the mappings:

Given main memory = 4 GB

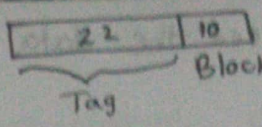
Cache memory = 4 MB

Block size = 1 KB

Direct:



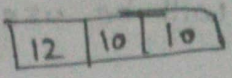
associative :



4 way set associative :

Block offset = 10

no of sets = $\frac{2^{12}}{4} = 2^{10}$



So

for K-way set associative

K=1 it is direct mapping

K=N it is associative mapping.

	Tag Comparators	Tag directory size
Direct	10	1	10×2^{12}
Associative	22	2^{12}	22×2^{12}
set Associative	12	4	12×2^{12}

Gate 99:

→ main memory has "2cn" blocks and cache has "2c" blocks and "2 way" set associative is used then k^{th} block of main memory maps to ___ set

Soln: we know that if it is a 2 way set mapping then there will be 'c' block sets will be present so K^{th} element will go in "K mod c" set

Gate 06:

→ more than one word are put in one block to exploit ____ in a program

Soln: if we want element into cache then we would load the surrounding bits also this is called locality of refevence.

it is of two types. 1) spatial locality and 2) temporial locality.

spatial locality says that if we need some data

now then in future we will need the data closest to what we picked

Temporial locality says that the data that we need now will be again needed in future also

so spatial locality will be used.

Gate 95:

→ Cache size = 4K words

block size = 64 words

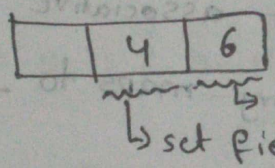
set size = 4 blocks

the no of bits present in "set" and "word" field of main memory address are

soln: no of lines = $\frac{2^{12}}{2^6} = 2^6$

no of sets = $\frac{2^6}{4} = 2^4$

so



Gate 07:

→ 4 way set associative

cache lines = 128

line size = 64 words

Physical addvess = 20

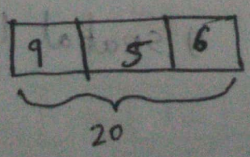
tag, set, word feilds are

soln: no of lines = 128

no of sets = $\frac{128}{4} = 2^5$

line size = 2^6

so



Gate 13

→ K way set associative cache contains 'v' sets the main memory block numbered 'j' must be mapped to any of the cache lines from

a) (j mod v) * k to (j mod v) * k + (k-1)

b) (j mod v) to (j mod v) + (k-1)

c) (j mod v) to (j mod v) + (v-1)

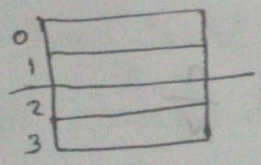
d) (j mod k) * v to (j mod k) * v + (v-1)

Soln: let k = 2

v = 2

we know

set number = j mod v



if we want to goto line 'k' then

(j mod v) * k

we know no of lines = k

so (j mod v) * k to (j mod v) * k + (k-1)

Gate 14

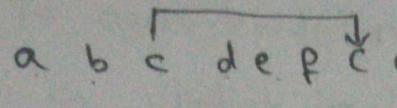
→ An access sequence of cache block address is of length 'N' and containing 'n' unique block addresses the number of unique addresses between two consecutive addresses access to the same block address is bounded above by 'k' what is the miss ratio if the access sequence is passed through a cache of associativity A ≥ k, exercising least recently used replacement policy

a) n/N b) $\frac{1}{N}$ c) $\frac{1}{A}$ d) $\frac{k}{n}$

Soln: Let the access sequence be

a b c d e f c which have 'n' unique block numbers

now, no of block numbers which are unique between two consecutive repeative numbers be 'k'



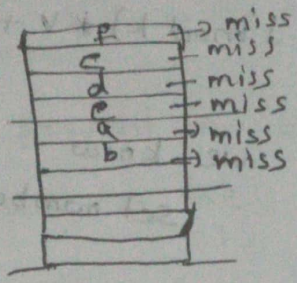
$k = 3$

$N = 8$

$n = 6$

Given Associativity $\geq k$

So



So miss rate $= \frac{6}{8} = \frac{n}{N}$

Gate 90:

→ Blocks in cache = 128

4 way set Associative

main memory contains 2^{14} blocks

Block size is 256 eight bit words

(i) How many bits are required for addressing MM

(ii) How many bits are needed to represent the TAG set and word fields.

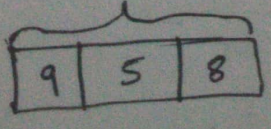
Soln: no of lines = 128

no of sets $= \frac{128}{4} = 2^5$

Block size $= 2^8$

main memory size = no of blocks × block size

$= 2^{14} \times 2^8 = 2^{22}$



Gate 05:

→ Direct mapped cache

cache size = 32 kB

Block size = 32 B

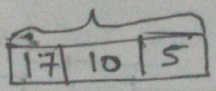
P.A = 32 bits

the no of bits needed for "cache indexing" and tag bits are respectively

Soln:

$$\text{no of lines} = \frac{32\ kB}{32\ B} = 2^{10}$$

Block size = 2^5



Gate 06:

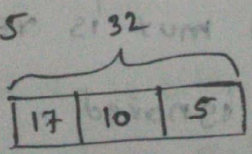
→ Consider two cache organisations the first one is 32 kB 2 way set associative with 32 byte block size. the second one is of same size but directly mapped. the size of an address is 32 bits in both cases. A 2 to 1 multiplexer has latency of 0.6 ns while a k bit comparator has latency of (k/10) nsec. the hit latency of the set associative organisation is h_1, while that of direct mapped is h_2.

Soln: Given in direct mapping

$$\text{no of lines} = \frac{32\ kB}{32} = 2^{10}$$

P.A = 32

Block size = 2^5



we know each multiplexer can read only 1 bit so for 10 bits

$2^{10} \times 1$ mux is needed

but latency for 2×1 is given not for $2^{10} \times 1$

So we ignore it

now in direct mapping latency is only due to

K bit magnitude Comparator

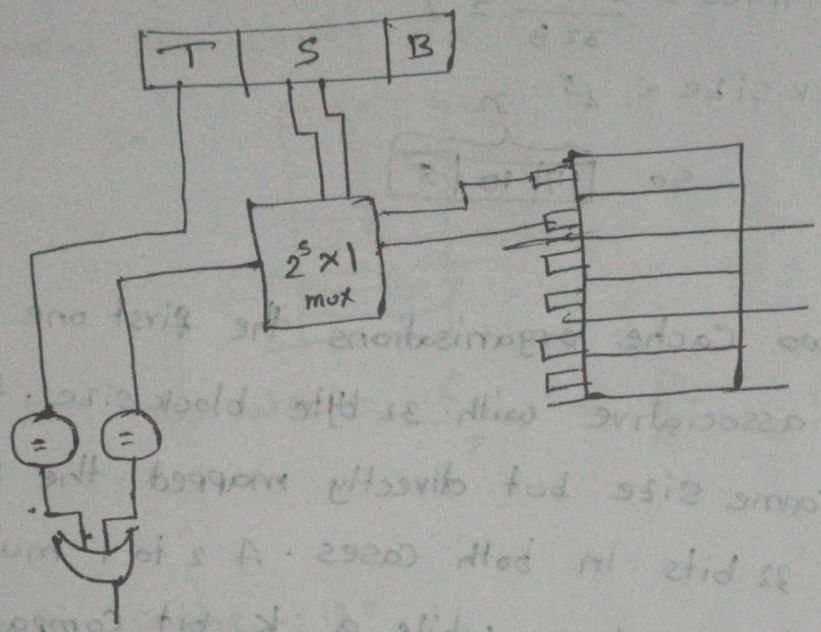
so k depends on no of tag bits

so $k = 17$

so magnitude Comparator latency $= \frac{k}{10} = 1.7$ nsec

so in direct mapping latency $= 1.7$ nsec

in set Associative mapping



so Given $B = 5$

$P.A = 32$

$S = \frac{32-10}{2} = 2^9$

so $\boxed{S = 9}$

$T = 18$

totally $2^9 \times 1$ mux is required but it is not given so it is ignored

2 K bit magnitude Comparators are required but they are Connected in parallel so one comparator time is enough

so magnitude Comparator latency $= \frac{K}{10} = \frac{18}{10}$

$= 1.8$ nsec

now or gate is implemented using 2x1 mux so

latency = 1.8 + 0.6 = 2.4ns

Gate 11

→ Direct mapping

cache size = 8 KB

Block size = 32 bytes

physical Address = 32 bits

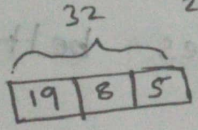
the cache Controller maintains tag information for each cache block Compraising of the following

1 valid bit, 1 modified bit and as many bits needed to identify the memory block mapped in the cache

what is the total size of the memory needed at the Cache Controller to store meta-data (tags) for the cache

a) 4864 bits b) 6144 bits c) 6656 bits d) 5376 bits

Soln: no of lines = $\frac{2^{13}}{2^5} = 2^8$



so

19 + 1 valid bit + 1 modified bit

= 21 bits

tag directory size = 21×2^8

= 5376 bits

Gate 12

→ Cache Size = 256 KB

4 way set associative

Block size = 32 B, P.A = 32 bits

Each Cache tag directory entry Contains in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit

Q) the no of bits in the tag field of an address is

a) 11 b) 14 c) 16 d) 27

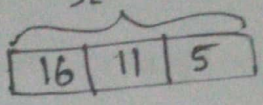
Q) the size of the cache tag directory is

a) 160 k bits b) 136 k bits c) 40 k bits d) 32 k bits

Soln:

$$\text{Given no of lines} = \frac{256\,KB}{32\,B} = \frac{2^{18}}{2^5} = 2^{13}$$

$$\text{no of sets} = \frac{2^{13}}{32} = \frac{2^{13}}{4} = 2^{11}$$



So tag field has 16 bits

Cache tag directory = (16 + 2 + 1 + 1) no of lines

$= 20 \times 2^{13}$

$= 160$ K Bits

Gate 14

⇒ in designing a computer cache system, the cache block (or cache line) size is an important parameter which of the following is correct.

a) A smaller block size implies better spatial locality

b) A smaller block size implies a smaller cache tag and hence lower cache tag overhead

c) A smaller block size implies a larger cache tag and hence lower cache hit time

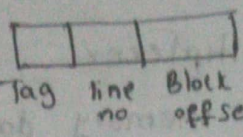
d) A smaller block size incurs a lower cache miss penality

Soln: option a:

According to spatial locality taking more data into cache helps us to reduce the miss latency but if block size is low then more data cannot be in the cache so it is not true

option b:

direct mapping:

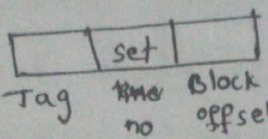


if block offset decreases

$$\text{no of lines} = \frac{\text{cache size}}{\text{Block offset}} = \text{increases}$$

which donot effect tag size

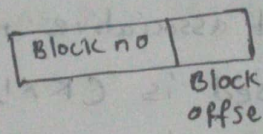
in set associative mapping:



if Block offset decreases no of sets increases

which donot effect tag size

in associative mapping:



there is no tags present here seperately

so option b is false.

option c:

lower block size will increase cache hit time

so it is false

option d:

lower block size means we can replace the miss lines easily into cache so lower miss penalty

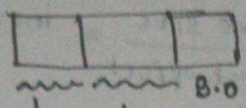
so it is true.

Gate 14

→ if the associativity of a processor cache is doubled while keeping the capacity and block size unchanged which one of the following is guaranteed not to be effected.

a) width of a tag Comparator
b) width of set index decoder
c) width of way selection multiplexer
d) width of processor to main memory data bus

Soln:



↓ to double the associativity

as the B.O is Constant

half the set size so '-1'

Tag must increase

So as tag size increase width of tag comparator increase

as no of sets decreases the width of set index decoder decrease

as we know in set associative mapping no of multiplexers used is (K x 1)

since k is doubled

no of multiplexers (2K x 1)

in option 'd' as the word length donot change the the width of processor to main memory databus will not change.

Gate 14

→ 4 way set Associativity

Cache size = 16 KB

Block size = 8 words

P.A.S = 4 GB

Tag bits

Soln: if any information is given in words the convert all the other information into words

i.e Cache size = 4 Kw

Block size = 8 w

P.A.S = 1 G w

so

no of lines = $\frac{2^{12}}{2^3} = 2^9$ Lines no of sets = $\frac{2^9}{4} = 2^7$

30

20	7	3

Memory interfacing

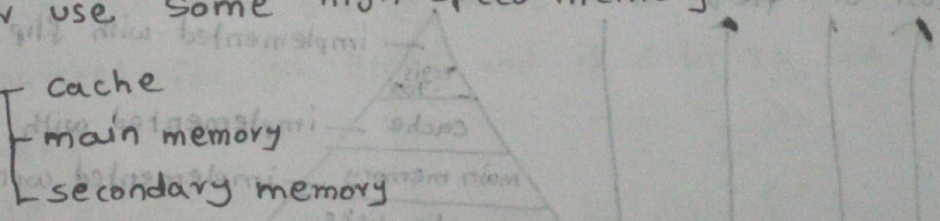
Introduction:

it comes under the computer organisation, the main difference between organisation and architecture is in organisation we study of how different parts are connected in computer whereas in architecture we study the effective use of the connected parts.

Generally cpu speed is calculated is "MIPS" (Million instructions per second)

But we cannot give input to cpu at that speed so we better use some high speed memory

memory
- cache
- main memory
- secondary memory



the speed of the secondary memory is very low

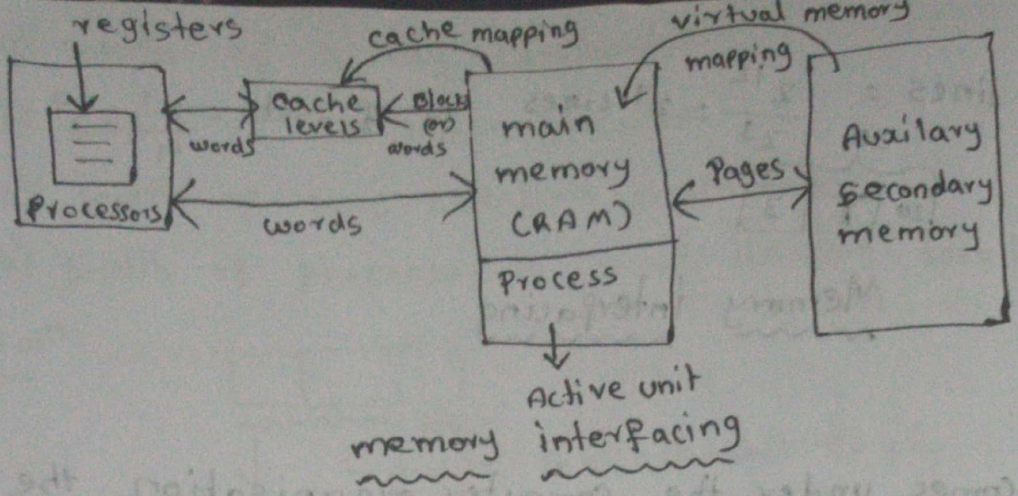
so in terms of speed

Cache > main memory > secondary memory

(volatile)	(volatile)	(non volatile)
very costly	costly	cheap

when we want to load program into cpu then we transfer it from secondary memory to main memory and main memory to cpu (or) main memory to cache and cache to cpu

we donot load program directly within the secondary memory because the speed of the secondary memory is very low.

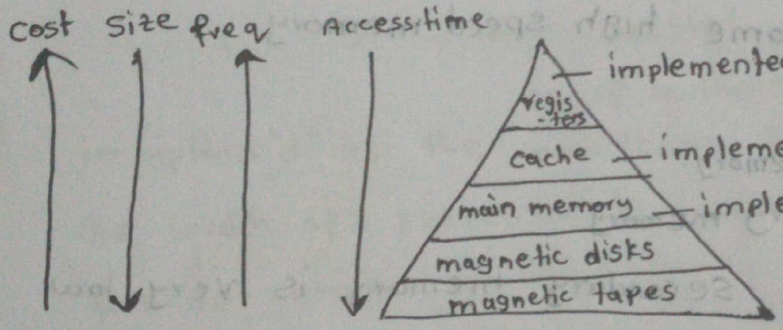


→ Cache levels, main memory have random access i.e we can goto a bit anywhere anytime in cache, main memory

→ in case of magnetic disks we got semi random access i.e upto some part we get random access and after that we got sequential access

→ magnetic tapes got sequential access.

memory Hierarchy:



Access time:

for registers Access time is less, which means they can be accessed very fast compared to cache, main memory etc

frequency:

Generally CPU uses register, cache more often than the others because they are fast

size:

Registers, cache are not having much sizes because they are very cost to implement

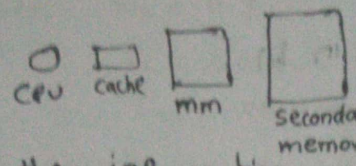
cost:

As we know registers are having very high cost.

the purpose of memory hierarchy is to bridge the speed mismatch between fastest processor to slow memory at reasonable cost.

2-level memory:

information in i^{th} is subset of $(i+1)^{th}$ level

i.e 

all the information present in computer is present in second memory and some part of it is present in main memory

some part of information in main memory is present in cache

some part of information present in cache is present in CPU

→ if processor refers to i^{th} level memory is found then "Hit" otherwise "miss (or) fault"

i.e if processor is looking for data in cache then if it found then Hit else fail (or) miss

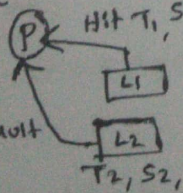
* if CPU searches for 100 times and if x times we found data then

$$\text{Hit rate} = \frac{x}{100}$$

$$\text{miss rate} = \frac{100-x}{100} = 1 - \frac{x}{100}$$

→ there are two ways in which the processor is connected to various levels of memory

Case-1:



L_1 is Level 1 memory

L_2 is Level 2 memory

Also access time of L_1 < access time of L_2

Let T_1 = access time of L_1

S_1 = size of L_1

C_1 = cost per bit of L_1 (or) cost per byte of L_1

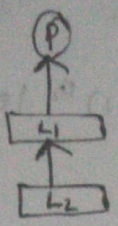
H_1 = Hit rate of L_1

now $T_{avg} = H_1T_1 + (1-H_1)T_2$

Also

$$C_{avg} = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

Case-2:



$$T_{avg} = H_1 T_1 + \overbrace{(1-H_1)(T_1+T_2)}^{\text{if fail in } L_1}$$

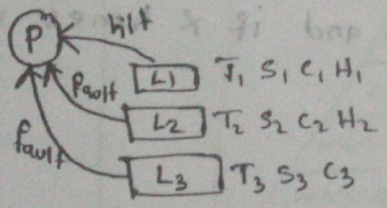
$H_1 T_1$ → if we found in L_1

$$C_{avg} = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$$

if in the question if type of connection is not mentioned i.e case 1 (or) case 2 then by default go take case-2

3-level memory:

Case-1:

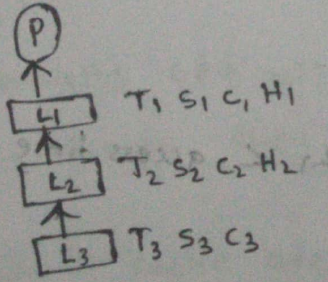


$T_1 < T_2 < T_3$

$$T_{avg} = H_1 T_1 + (1-H_1) H_2 T_2 + (1-H_1)(1-H_2) T_3$$

$$C_{avg} = \frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3}$$

Case-2:



$$T_{avg} = H_1 T_1 + (1-H_1) H_2 (T_1+T_2) + (1-H_1)(1-H_2)(T_1+T_2+T_3)$$

$$C_{avg} = \frac{C_1 S_1 + C_2 S_2 + C_3 S_3}{S_1 + S_2 + S_3}$$

→ the average memory access time for a machine with a cache hit rate of 80% where the cache access time is 5 nsec and main memory access time is 100 ns is

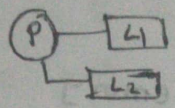
Soln: Given $H_c = 0.8$

$T_c = 5$ nsec

$T_m = 100$ nsec

but case(1) (or) case(2) is not given so we do both the cases

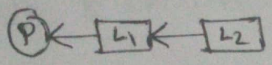
case-1:



$T_{avg} = 0.8(5) + 0.2(100)$

$= 24$ nsec

case-2:



$T_{avg} = 0.8(5) + 0.2(100+5)$

$= 25$ nsec

Cache replacement Algorithms:

→ Replacement policy is required for Associative mapping and set associative mapping but not for direct mapping

→ Replacement policies are aimed to minimize miss penalty for future references

Various replacement policies are:

Random:

no specific criteria to replace the block.

FIFO:

the block that entered first is the candidate for replacement

LRU:

the block which is not referred for the longest time is the candidate for replacement.

LFU:

the block with fewer refevences is replaced.

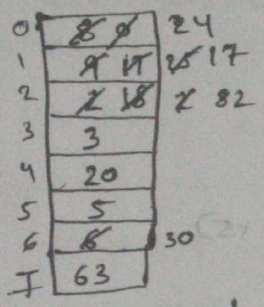
→ Consider a direct mapped cache with 8 cache blocks (0-7) if the memory blocks requests are in the order 3, 5, 2, 8, 0, 6, 3, 9, 16, 20, 17, 25, 18, 30 24, 2, 63, 5, 82, 17, 24 which one of the memory blocks are not present in cache at the end of the sequence

a) 3 b) 18 c) 20 d) 30

Soln: $i = j \bmod 8$

where j is block no in main memory

i is line no in cache



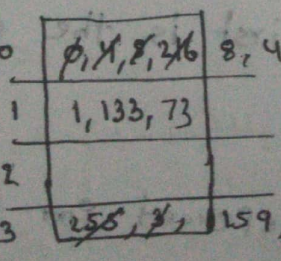
So 18 is not present.

→ Consider a 4 way set associative mapping with 16 cache blocks the memory block requests are in the order (0, 255, 1, 4, 3, 8, 133, 159, 216, 219, 63, 8, 48, 32, 73, 92, 155) which one of the following memory block will not be in cache if LRU is used

a) 3, b) 8, c) 129 d) 216

Soln: Given 4 way set associative

So each set has 4 lines



in set associative mapping the blocks will go into sets ranging from (0 to (k-1))

now the block number will go to set no

$i = j \bmod k$

k-set associative

j → is main memory block number

i = set no which j will go

So $i = j \bmod 4$

Given LRU replacement policy is given

in set 0, 3 4 lines are filled so now we must replace them using 'LRU'

So 255 is replaced for 63

8 is replaced because 8 is already present in the cache and since we are using 'LRU' if we leave like it in future we may think that 216 is ~~least~~ recently used but '8' is reecently used

so we replace '8' with '8' for clarity

So only '8' is present in the cache.

→ Consider a small 2-way set Associative mapping with a total of 4 blocks, for choosing the block to be replace use LRU scheme the number of cache misses for the following block addresses 8, 12, 0, 12, 8 is ____

Soln: Given

0	8, 12	0, 12, 8
1		

8 - miss
12 - miss
0 - miss
12 - hit
8 - miss

So hit rate = $\frac{1}{5} \times 100 = 20\%$

miss rate = $\frac{4}{5} \times 100 = 80\%$

→ Consider a 2 way set associative mapping consisting of 2^c memory blocks and 2c cache blocks, the cache location for the memory block 'k' is

a) k mod 2c b) k mod 2^c c) k mod c d) 2^c mod k

Soln: Cache Size = 2c blocks set size = 2 blocks

main memory = 2^c blocks

no of sets = $\frac{\text{cache size}}{\text{set size}} = \frac{2c}{2} = c$

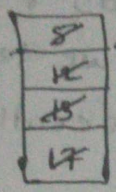
So k mod c

⇒ Consider the cache has 4 blocks for the memory refevences 5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 13, 43, 61, 19 what is the hit ratio of following cache replacement policy.

1) FIFO 2) LRU 3) Direct mapping 4) 2 way set Associa

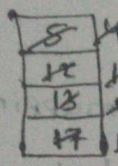
Soln:

FIFO



Hit ratio = $\frac{5}{15} \times 100 = 33.33$

LRU



(or)

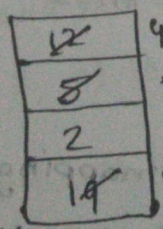
5, 12, 13, 17, 4, 12, 13, 17, 2, 13, 19, 43, 13, 19

5 - m
12 - m
13 - m
17 - m
4 - m
43 - m
13 - h
19 - h

12 - hit
13 - hit
17 - hit
2 - m
13 - h
61 - m

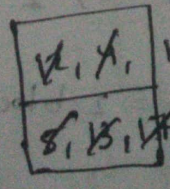
Hit ratio = $\frac{6}{15} \times 100 = \frac{2}{5} \times 100$

Direct mapping:



So Hit ratio = $\frac{1}{15} \times 100$

2 way set associative (LRU)



So hit ratio = $\frac{5}{15} \times 100$

→ A hierarchal memory system has the following specifications 20 mB main storage with access time of 350 ns and 256 B cache with access time 50 nsec word size is 4B, page size is 8 words what will be the hit ratio if the page address trac of a program has the pattern 0,1,2,3,0,1,2,4 and following LRU page replacement technique.

Soln: word size = 4B
Page size = 8 w
now we will convert words to bytes
so page size = 8 × 4B = 32 B

$$\text{no of cache pages} = \frac{\text{cache size}}{\text{page size}} = \frac{256}{32} = 8$$

LRU is used

0, 1, 2, 3, 0, 1, 2, 4

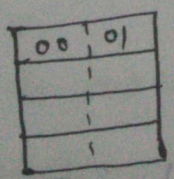
So hit ratio = $\frac{3}{8} \times 100$

→ Consider an array has 100 elements and each element occupies 4 words A 32 bit word cache is used and divided into a block of 8 words then what is the hit ratio of

```
for (i=0; i<10; i++)
    for (j=0; j<10; j++)
        A[i][j] = A[i][j] + 10;
```

Soln: we can store the array in two orders
Row major order, coloumn major order

[00, 01, 02, ...] [00, 10, 20, ...]

Given 

in cache each block can hold two elements

i.e if we call 00 then

if we call 00 then along with 00, 01 will also be called into cache as each block can store two elements

read 00 00 write, read 01 01 write

so read 00 - miss
write 00 - hit
read 01 - hit
write 01 - hit

So in row major order hit rate = $\frac{3}{4} \times 100 = \frac{3}{4} \times 100$

So in case of Coloumn major order

m	H	m	H
00	00	01	01
read	write	read	write

So hit rate = $\frac{2}{4} \times 100$

→ Consider an array A[100] and each element occupies 4 words, A 32 word cache is used and divided into 8 word blocks

what is the hit ratio for the statement.

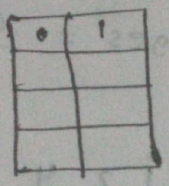
```
for ( i=0; i<100; i++)
    A[i] = A[i] +10
```

Soln: Given

Cache size = 32 w

Block size = 8 w

Blocks = $\frac{32}{8} = 4$

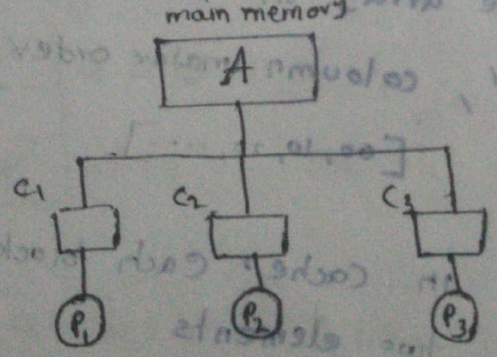


0	0	1	1
read	write	read	write
m	H	H	H

So hit ratio = $\frac{3}{4} \times 100$

Cache coherence problem:

multiple copies of same data can exist in different caches simultaniously and if processors are allowed to update their own copies freely an inconsist view of memory can result



if 3 processors P_1, P_2, P_3 are having cache memories C_1, C_2, C_3 then if all the processors having common main memory then if all of them want to operate

on same variable 'A'

Let A = 5

P_1 wants to increment A so it loads A into cache and increment it to '6'

P_2 wants to decrement 'A' but before P_1 writes the result of 'A' in main memory P_2 loads A into its cache and decrement it to '4'

so Actually $A = 5 \rightarrow \underset{P_1}{6} \leftrightarrow \underset{P_2}{5}$

But $A = 5 \rightarrow \underset{P_1}{6}$

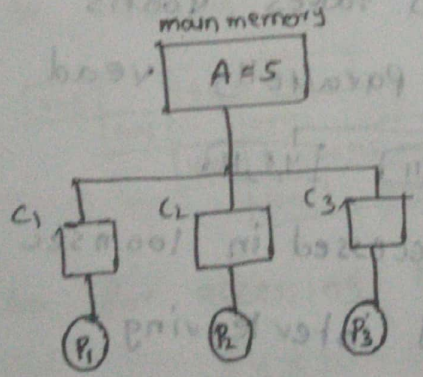
$= 5 \rightarrow \underset{P_2}{4}$

So inconsistency occurs

this is called cache coherence problem.

there are four methods to avoid cache coherence problem

1) write update - write through
2) write update - write back
3) write invalidate - write through
4) write invalidate - write back.



Let A=5

if P_1 takes A into its cache and decrement it and immediatly update it in main memory then it is called write through

if P_1 takes A=5 and decrement it and the new value is kept in cache and if it wants to clear the cache then it update the value in main memory then

it is called write back

if P_1, P_2, P_3 are accessing 'A'

P_1 changed A = 5 to '4' and then it immediatly updated the value of A in P_2, P_3 then it is called write update

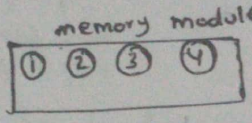
if P_1, P_2, P_3 are accessing A

P_1 changed A = 5 to '4' and then it warned P_2, P_3 to not to use value of 'A' then it is called write invalidate.

memory interleaving:

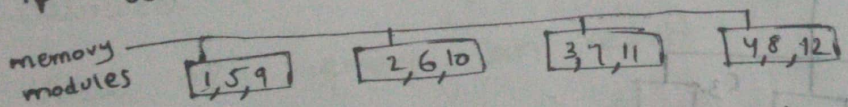
we know when a program is being run then it CPU asks for more than one word at a time for execution

So it means that



1, 2, 3, 4 are the words present in a program accessing one after the other is waste of time i.e access time for each word is 100ns then for accessing all the 4 words takes 400ns

if all the words are parallelly read then

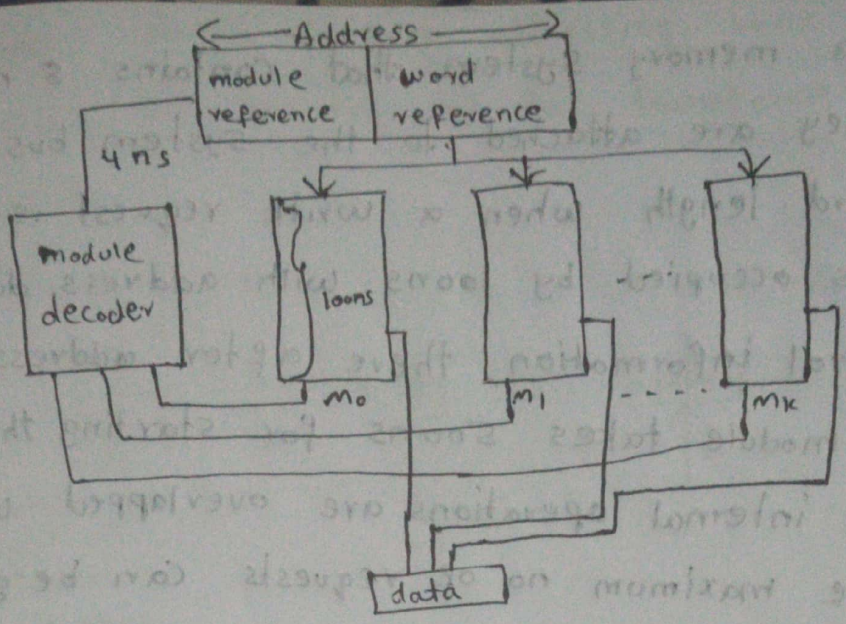


1, 2, 3, 4 can be accessed in 100 nsec so this is the concept of memory interleaving.

memory interleaving will reduce the average access time and improves data transfer rate.

the accessing time for all these four words will be reduced if we access them parallely

So due to memory interleaving the access time of memory decreases.

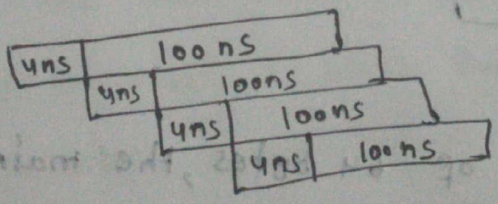


Suppose cpu generated an address then it is divided into module reference i.e a decodev will decode this reference and says the required data is present in which module ($m_0, m_1, \cdots m_k$)

for decodev it takes 4ns to decode the module no word reference will find the selected word in the particular module

if access time of each module is 100ns

then to find a word



so for accessing k words

$$(4 \times K + 100\ ns)$$

If memory interleaving is not used then

$(K \times 100\ ns)$ will be the access time for k words.

This is how memory interleaving will reduce the average access time and improves data transfer rate

→ Consider a memory system that contains 8 memory modules they are attached to the system bus which is one word length when a write request is made the bus is occupied by 100ns with address data and control information, there after addressed memory module takes 500ns for starting the work if modules internal operations are overlapped what will be the maximum no. of requests can be generated in 1ms time.

Soln: we know

the decoder takes 100 ns

memory module takes 500 ns

So

$$1ms = K \times (100ns) + 500\,ns$$

$$10^{-3} = K \times 10^{-7} + 5 \times 10^{-7}$$

$$10^{-3} = 10^{-7}(K+5)$$

$$K+5 = 10^{4}$$

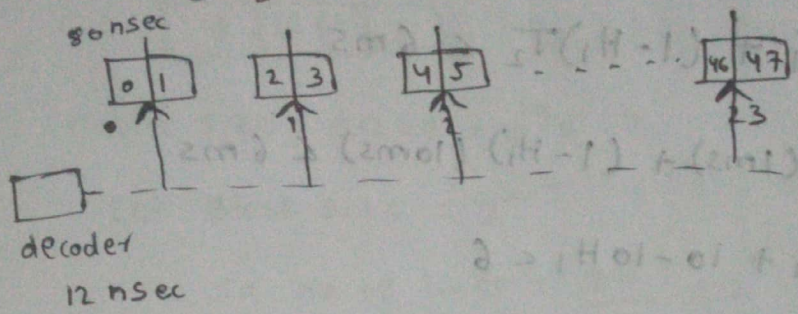
$$\boxed{K \cong 10^{4}}$$

Gate 06:

→ A cpu has a cache of 64 bytes, the main memory has k banks each bank has can store c-bytes of data. the consecutive 'c' byte chunks are mapped to consecutive banks with wrap around manner all k banks can be accessed in parallel. However the two access for the same bank have to be serialized (one after the other). Let "k" = 24 and c = 2. Each iteration requires decoding the bank numbers to be accessed in parallel and this takes

$\frac{K}{2}$ nsec, latency of each addressing bank is 80 ns
How much time is required for transfering initial block of cache.

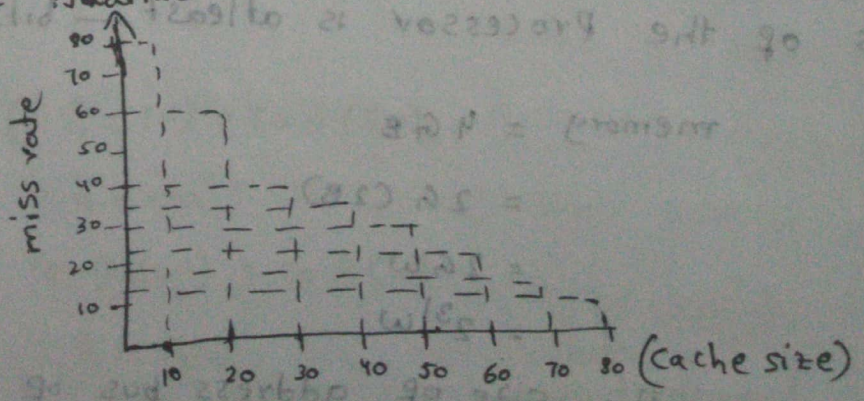
Soln: Given K = 24
C = 2



for 24 banks 48 bytes of data is present
but to transfer 64 bytes we need two iterations because we only transfer integral no of times as time for one iteration is given
so time for one iteration = 12 + 80 nsec
= 92 nsec
for two iterations 2×92 = 184 nsec.

Gate 16

→ A file system uses an in memory cache to cache disk blocks. the miss rate of the cache is shown in fig the latency to read a block from cache is 1ms and to read a block from disk is 10ms Assume that cost of checking weather a block exists in the cache is negligible Available cache sizes are in multiples of 10 mB



A smallest cache size required to ensure an average read latency of less than 6ms is —— mB?

Soln: we should get miss rate in order to find the Cache size

we know

Read latency < 6 ms

$H_1 T_1 + (1-H_1) T_2 < 6$ ms

$H_1 (1\text{ms}) + (1-H_1)(10\text{ms}) < 6$ ms

$H_1 + 10 - 10H_1 < 6$

$-9H_1 < -4$

$H_1 > \frac{4}{9}$

$-H_1 < -\frac{4}{9}$

$1 - H_1 < 1 - \frac{4}{9}$

miss rate

$m < \frac{5}{9} \Rightarrow \boxed{m < 0.57}$

So neavest value is 40% which has 30 MB as cache size.

Gate 16

→ A Processor can support a maximum memory of 4 GB, where the memory is word addressable (a word consists of two bytes) the size of address bus of the Processor is atleast ___ bits.

Soln: memory = 4 GB

= 2 G (2B)

= 2 G W

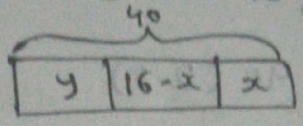
= 2^{31} W

So the size of address bus of the Processor is atleast 31 bits.

Gate 16

→ the width of the physical address on a machine is 40 bits. the width of tag field in a 512 KB 8 way set associative cache is ___ bits.

Soln:



Cache size = 512 KB = 2^{19} B

Let Block size = 2^x

So no of lines = $\frac{2^{19}}{2^x} = 2^{19-x}$

no of sets = $\frac{2^{19-x}}{8} = 2^{19-x-3} = 2^{16-x}$

So

$y + 16 - x + x = 40$

$\boxed{y = 14}$

Gate-14:

→ the memory access time is 1nsec for a read operation with a hit in cache, 5ns for a read operation for a miss in cache, 2ns for a write operation with a hit in cache and 10nsec for a write operation with a miss in cache. execution of a sequence of instructions involves 100 instruction fetch operations, 60 memory read operations and 40 memory write operations. the cache hit ratio is 0.9 the average memory access time (in ns) in executing the sequence of instructions is ____

Soln:

$T_R = (0.9)(1n) + (0.1)(5n)$

$= 1.4$ nsec

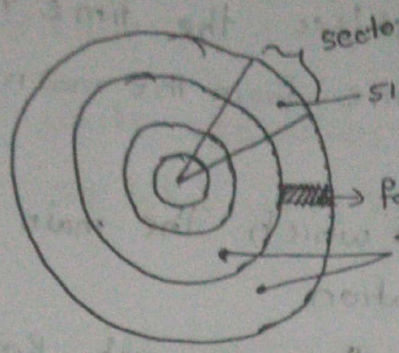
$T_W = (0.9)(2n) + (0.1)(10n)$

$= 2.8$ nsec

Given 100 instruction fetch, 60 instruction read

So instruction fetch is also instruction read

Secondary memory



→ Almost all the hard disks have sector size as 512 Bytes

→ Total size of disk = no of tracks × $\frac{\text{sectors}}{\text{tracks}}$ × $\frac{\text{no of Bytes}}{\text{sectors}}$

→ format is used to save sector number, track number etc

→ Read write head is used to read or write information to the hard disk.

So for reading or writing we need to move the read/write head from one sector to other sector(or) from one track to another track.

* the time required to move read/write head from one track to another track is called seek time
* the time taken to wait for sector to come under the read write head is called rotational delay

T_{avg} = Seek time + rotational delay + Time taken to transfer data

$= T_S + T_r + T_{transfer}$

if T_S is not given take it '0'

if T_r is not given take $\frac{1}{2}$ (rotation time)

→ Consider a disk pack with the following specifications 16 surfaces, 128 tracks/surface, 256 sectors/track and 512 bytes/sector

a) what is the capacity of disk pack

b) the no of bits required to address the sector

Soln: $16 \text{ surf} \times 128 \frac{\text{tracks}}{\text{surf}} \times 256 \frac{\text{sectors}}{\text{track}} \times 512 \frac{\text{bytes}}{\text{sectors}}$

$= 2^4 \times 2^9 \times 2^7 \times 2^9$

a) $= 256 \text{ mB}$

b) $16 \text{ surf} \times 128 \frac{\text{tracks}}{\text{surf}} \times 256 \frac{\text{sectors}}{\text{track}}$

$= 2^4 \times 2^7 \times 2^8 \text{ sectors}$

$= 2^{19}$

So 19 bits are required.

c) in the above disk pack the format overhead 32 bytes/sector then what is the formatted disk space

Soln: no of sectors $= 2^{19}$

32 bytes/sector

So $2^{19} \times 32 = 2^{24} \text{ B}$

$= 16 \text{ mB}$

Total usable space $= 256 - 16 \text{ mB}$

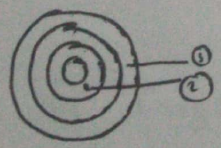
$= 240 \text{ mB}$

d) in the above disk space the format overhead is 64 bytes/sect. How much amount of memory is wasted due to formatting.

Soln: Formatting overhead $= 2^{19} \times 64 = 2^{25} \text{ B} = 32 \text{ mB}$

e) Let the diameter of the inner most track is 21cm what is the maximum recording density.

Soln:



in both ①, ② no of bytes present are same but circumference of '1' is greater than '2'

due to density difference in ①, ② no of bytes are same density of ② > density of ①

Track capacity $= 256 \text{ sectors/track} \times 512 \frac{\text{bytes}}{\text{sector}}$

$= 2^{17} \text{ bytes/track}$

Circumference $= 2\pi r = \pi d = \frac{22}{7} \times 21 = 66 \text{ cm}$

So for 66 cm $- 2^{17}$ bytes

$\Rightarrow \frac{2^{17}}{66} = \frac{128 \text{ KB}}{66} = 1.9 \text{ KB/cm}$

$- 1 \text{ cm} \rightarrow ?$

F) let the diameter of the innermost track is 21cm with 2 KB/cm what is the capacity of the track.

Soln: track capacity = circumference x density

$= 66\ cm \times 2\ \frac{KB}{cm}$

$= 132\ KB$

G) the disk is rotating at 3600 RPM what is the data transfer rate

Soln: there are 16 surfaces so all the surfaces can be read at a time

So

3600 Rotations – 60 sec

1 rotation → x

$x = \frac{1}{60} s$

we know track capacity = 128 KB

so after one rotation we can read 128 KB data

$\frac{1}{60} s \rightarrow 128\ KB$

$1 sec \rightarrow 128 \times 60$

total data rate = $16 \times 128 \times 60$

$= 120\ mbps$

H) using one read/write head disk is rotating at 6000 rpm what is the data transfer rate

Soln:

6000 R → 60 sec

$1 sec \rightarrow \frac{1}{1000} R$

$\frac{1}{100} s - 128\ KB$

$1 s - 128\ KB \times 100$

$1 s - 13\ mbps$

I) if the disk system has rotational speed of 3000 rpm what is the average access time with a seek time of 4.5 msec

Soln: we know that

$T_{avg} = S_k + \frac{1}{2}(RT) + \text{transfer rate}$

since transfer rate is not given take it as '0'

now rotating time - 3000 rpm

3000R - 60sec

1R - $\frac{1}{50}$ sec

$\frac{1}{2}R - \frac{1}{100}$ sec = 10 m sec

so $T_{avg} = 11.5 + 10$

$= 21.5$ msec

J) what is the average access time for transfering 512 bytes of data with following specs

Average seek time = 5 m sec

disk rotation = 6000 rpm

Data rate = 40 kB/sec

Controller overhead = 0.1 msec

Soln: $T_{avg} = S.T + \frac{1}{2}(RT) + D.T + C.O$

6000 R - 60sec

1R - $\frac{1}{100}$ sec

$\frac{1}{2}R$ - 5 msec

D.T → 40 Kb - 1sec

512 b →

$\frac{512}{40 \times 1024} = 12.5$ m sec

so $T_{avg} = 5m + 5m + 12.5m + 0.1m$

$= 22.6$ msec

→ A certain moving arm disk storage with one head has the following specifications

Number of tracks/surface = 200

disk rotation speed = 2400 rpm

tracks storage capacity = 62,500 bits

average latency = p msec

data transfer rate = a bits/sec

what is the value of P and Q

Soln:

2400 R – 60 sec

1R → $\frac{1}{40}$ sec

$\frac{1}{2}$R → 12.5msec

so P = 12.5msec

25msec – 1rot – 62,500 bits

1sec – $\frac{62500}{25} \times 1000 = 2.5 \times 10^6$ bps

Gate

→ A hard disk has 63 sectors/track, 10 platters each with 2 recording surfaces and 1000 cyclinders. the address of the sector is given as

< c, h, s > where c - cylinder number, h - surface number, s - sector number then the 0th sector is addressed as < 0, 0, 0 > 1st sector is addressed as < 0, 0, 1 > and so on

Q) the address < 400, 16, 29 > corrosponds to sector number

a) 5050 35 b) 505036 c) 505037 d) 505038

Soln: if a numbering is starting from '0' then if x is where we are which means we should travel x numbers to reach our place

0 1 2 3 if we are '3' in order to reach '3' we should cross 3 numbers

So in question

1 cyclinder has 10 platters, 2 surfaces, 63 sectors, no of tracks per surface is not given so take 1 track per surface

So, < 400, 16, 29 > = 400 (10 × 2 × 63) + 16 (63) + 29

= 505037

2) the address of 2039 sector is

a) $\langle 0, 15, 25 \rangle$ b) $\langle 0, 16, 30 \rangle$ c) $\langle 0, 16, 31 \rangle$, d) $\langle 0, 17, 31 \rangle$

Sol^n: ~~xxxxx~~

by verification

(c) = $1000 + 16(63) + 31$

$= 2039$

So option 'c'