→ Set Theory started from Back of Notebook

COA GATE

Topic	Level	No. of revisions
① Caching (Most important)	Moderate	
② Memory Interfacing	Easy and Tricky	
③ Secondary Memory	Easy	
imp ④ Machine Instruction & Addressing modes	Moderate	
imp ⑤ ALU, Data Path & Control Unit	Moderate	
⑥ I/O Interface	Moderate	
⑦ Pipelining.	Moderate	

→ Do more and more questions on each topic from book.

Caching (100% question in Gate)

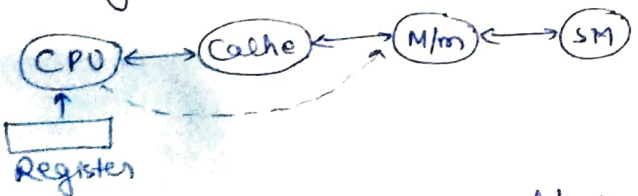
Introduction to Cache Memory

→ CPU is faster than main memory. So, if CPU tries to access data from main memory its speed get reduces because main m/m can't reply that fast.

→ So, we need some fast m/m. Register is the solution but its size is small and its also too costly.

→ Next solution is cache, it is slower than register but better than main m/m. Cache doesn't match the level with CPU in speed but its better than main m/m.

→ So, we have various level of memories.



Main m/m contains a part of process available in secondary m/m.
Cache contains some part of page.
Register contains some part of cache.
So, CPU directly starts access from register and go forward to main m/m if required.

→ But we are considering register here bcoz its take negligible time.

→ If require element present in cache called 'Cache hit'.

And time required to fetch data from cache is 'hit latency'.

→ If data doesn't found in cache called as 'cache miss' and time require to fetch data from main m/m called 'miss latency'.

→ If data doesn't present in main m/m called as page fault otherwise page hit.

→ Tag directory: It says the require page in present in tag or not.

→ In case of page fault we take page fault service means loading the page from secondary memory.

→ So, there is need to thought that which part of main m/m needs to load in cache. For this we have principle of locality. It is of two types —

① Spatial Locality

→ It says that if you possess a cache miss then its chances to find the element in proximity is very high. It means you can find the element around the previous miss.

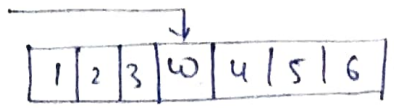
② Temporal Locality

→ It say that least recently used element is going to use again. So, we use LRU generally in page replacement algorithm.

→ Explaination of spatial locality.

Cont—

Say, we face a page fault because we don't find a word in main m/m. So we load the complete page from the secondary m/m



Say, w is the word that posses page fault. We load the complete page instead of loading only a single word bcoz of spatial locality means its highly probable that other 6 words can be demand soon.

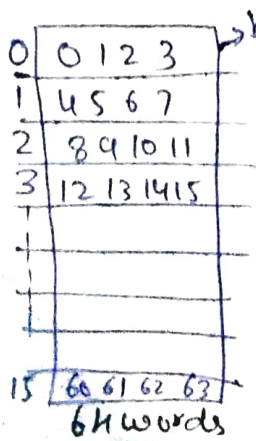
Introduction to Direct mapping

→ We are talking about main m/m and Cache mapping.

→ Paging is just a concept to manage the m/m more efficiently.

→ In OS we divide main m/m into → frame but here we divide it into blocks and Cache into lines i.e block size is same as line size.

→ We assume 1 word = 1 Byte.



Say block size = 4 words

No. of blocks = $\frac{64}{4}$

= 16 blocks

→ 16 blocks with 4 words in each block.

No. of lines = $\frac{16}{4}$ = 4 lines

4 lines with 4 words in each line.

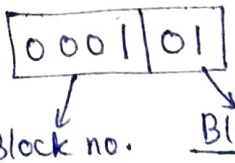
→ for 64 words we need 6 bit to represent.

→ So, CPU will generate 6-bit address.

Say CPU generate address as

$[000101]_2 \rightarrow (5)_{10}$

→ Since block size = 4 words So we devide the least significant (2^2) 2 bits as block offset.



So, it's task that which block from main m/m should be loaded in cache. for this we have various kind of mapping techniques.

① Direct mapping:

Cache

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

→ So main m/m to cache is a many-one kind of function.

Cache contains block numbers.

We have 16 blocks so we need 4 bits to represent each block.

	offset		line no.	
0 -	0	0	0	0
1 -	0	0	0	1
2 -	0	0	1	0
3 -	0	0	1	1
4 -	0	1	0	0
5 -	0	1	0	1
6 -	0	1	1	0
7 -	0	1	1	1
8 -	1	0	0	0
9 -	1	0	0	1
10 -	1	0	1	0
11 -	1	0	1	1
12 -	1	1	0	0
13 -	1	1	0	1
14 -	1	1	1	0
15 -	1	1	1	1

for line '0' in cache we choose those block that have least two digits as 00.

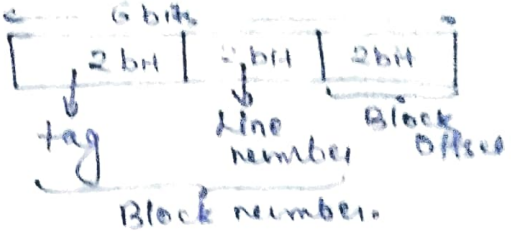
Similarly for various lines the last 2 least significant digits are

Line 1 → 01

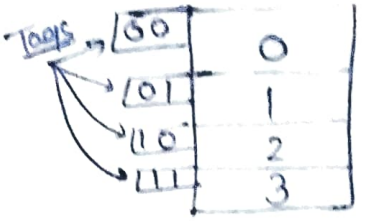
Line 2 → 10

Line 3 → 11

→ We take two bits only to take line no.



Ex:



0, 1, 2, 3 are line number and each line contain 4 words

In block offset we are free to put any of the four combinations.

① 0×4=0
0 + 1 + 2 + 3
(0, 1, 2, 3)

		2bits
00	00	00
00	00	01
00	00	10
00	00	11

5×4
20 + 21 + 22 + 23

② (20, 21, 22, 23)

		2bits
01	01	00
01	01	01
01	01	10
01	01	11
01		

10×4=40

③ (41 42 43 44)

10	10	00
10	10	01
10	10	10
10	10	11
10		

→ We give all four different value to tag but we can give same value to tag if we have sufficient no. of blocks to represent.

Direct mapping Problems

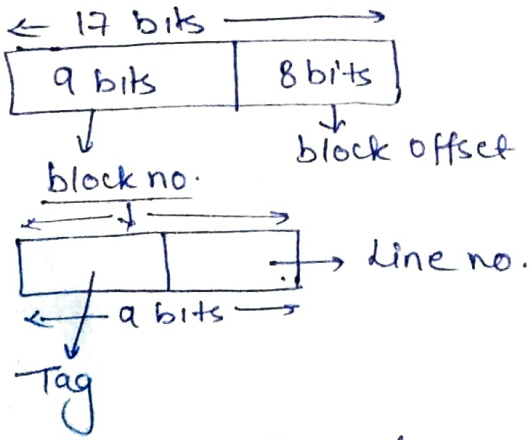
Problem 1 (Byte Addressable)

	MM size	Cache size	Block size	Tag Bits	Tag Directory size
①	128kB	16kB	256B	3	3X26
②	32GB	32kB	1kB	20	20X25
③	64MB	512kB	1kB	7	7X29
④	16GB	16MB	4kB	10	10X212
⑤	64MB	2^{16}B	Can't guess	10	Can't guess
⑥	2^{26}B	512kB	Can't guess	7	Can't guess

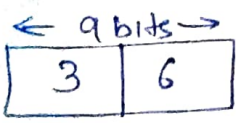
Tag directory size = (Tag bits) x no. of lines

Soln ① MM size = 2^{17} B
Cache — = 2^{14} B
Block — = 2^8 B

No. of blocks = $\frac{2^{17}}{2^8} = 2^9$ blocks



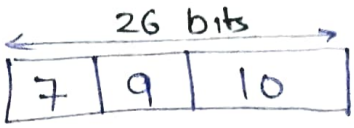
No. of lines = $\frac{2^{14}}{2^8} = 2^6$



2^3 blocks of main memory mapped with cache.

② Solution is same as ①

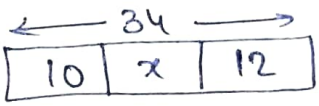
③



Block size = 2^{10} B
Cache size = 512 kB = 2^{19} B
No. of lines = 29 (9 bits)

MM size = 2^{26} B
= 64 MB

④



No. of blocks = $\frac{2^4 \times 2^{30}\ B}{2^{12}}$

$= 2^{34-12} = 2^{22}$

16 GB = 2^{34} B

So, $10 + x + 12 = 34$

$x = 12$.

Cache size 2^{24} B = 16MB

No. of lines = $\frac{2^{24}\ B}{2^{12}} = 2^{12}$

⑤ We can't find block size because we can guess many combination.

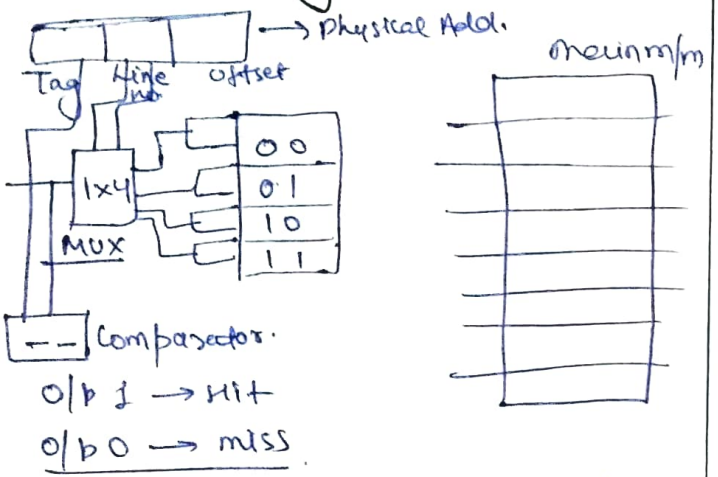
⑥ Cache size = 2^{19} B

(line no.) x (line offset) = 2^{19} B.

Ratio between main m/m and Cache is tag size.

→ Tag directory size is mostly asked question.

Direct mapping H/w implementation



We choose the line no and gets its tag bit as o/p through MUX. If MUX is o/p and tag bit in address is same then we get o/p '1' else '0'.

Time taken in above procedure is hit latency.

As the tag bits increases no. of MUX gets increases.

All the MUX will get the same line no.

~~for~~ k-bit tag we need k Multiplexers but only one Comparator (k-bit).

→ Since all the MUX are working in parallel so at a time only one MUX and only one comparator should work.

Hit latency = Comparator delay + MUX delay

Generally, MUX delay << Comparator delay

So we can neglect it.

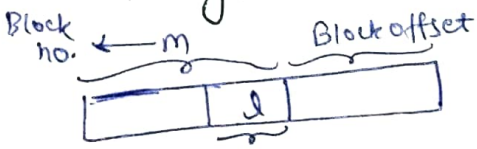
Ex: Main m/m size = 1GB
Cache — = 1MB
Comparator delay = 10ns
Hit latency = ?

Soln Tags = 2^{10} (10 bits) (k)
Delay = 10 x 10 = 100ns (k)
Hit latency = Delay (here).

Note:- $k = m \% n$

mth block of main m/m present in kth line of cache, where n is no. of lines in cache.

Disadvantage of direct mapping



No. of blocks = 2^m
No. of lines = 2^l

Say x is any m bits no. then

$x \% 2^l$ = line no.

→ It gives line no. in which x is present

Say a cache have 4 lines. and CPU generate block request as 5, 6, 4, 8, 9, 12, 15, 20. then find the line no. in which block get placed and also find the miss rate.

12%4=0
15%4=3
20%4=0

0	4 8 12 20
1	5 9
2	6
3	15

5%4 = 1
6%4 = 2
4%4 = 0
8%4 = 0
9%4 = 1

After filling the block no. 4 in line no. 0, we get the request of block no. 8 and then we replace the block no. 4 with block no. 8 even the line 3 was empty, but we can't use it. It is the drawback of direct mapping.

In worst case input should be like
4, 8, 12, 16, 20, 24 - - -

0	4 8 12 16 20 24 (Conflict miss)
1	
2	
3	

Even we have plenty of space available but we can't use because of many-one mapping.

We are missing the block due to conflict on same line with another block.

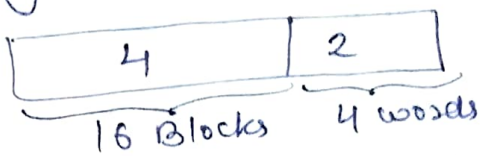
→ Particular block can go in particular line.

Associative mapping

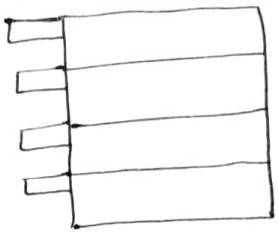
→ Solution to direct mapping.

→ A block can get inside any line under some conditions.

Say we have 6 bit PA.



All 16 blocks can map to any line of cache. so its many-many function.



→ Tag says that how many blocks can reside in a single line.

→ Here we can allow all of 16 blocks to use any of the line. So tag should be of 4 bits for 16 blocks.

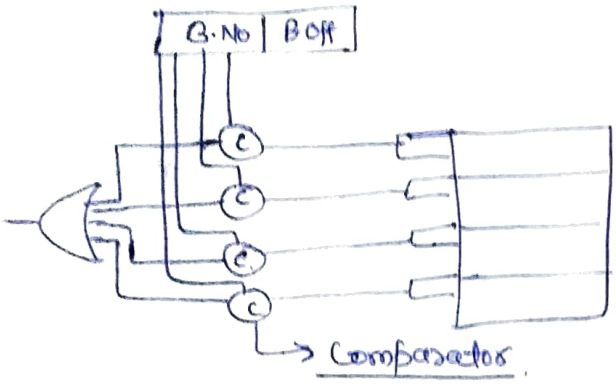
Block no = tag (Here)

→ There is 'no conflict miss'

Conflict miss. block 4 already line में था but 8 के आने से उसे replace किया तो अगर आगे चलके कभी block 4 की demand होगी तो हमें miss को face करना पड़ेगा क्योंकि यहाँ पर Conflict है कि कोई एक block ही रह सकता है।

→ Where should be search for a block?

→ So here we have many comparator connected to each tag.



So, we get the soln of conflict miss but it increases the hardware cost bcoz of many comparators

Hit latency → Time taken in comparator + Time taken by (OR) Gate

→ We can replace OR Gate with any other combinational ckt if required.

Ex → MM = 32GB, Block = 32kB
Tag = ?
Propagation delay of comparator = 10k ns
Prop. delay of OR gate = 10ns
Hit latency = ?
(k = no. of tag bits)

Soln $Tag = \frac{2^{35}}{2^{15}} = 2^{20}$
$= 20$ bits
$k = 20$

Hit latency = (10×20 + 10) ns
= 210 ns

→ Tag = no. of blocks.

→ All the comparators are used in parallel. So we consider it as only one comparator time.

→ Assume that all the lines are full and we want to place a new block then which block from which line will get replace. For this decision we have block replacement algorithms.

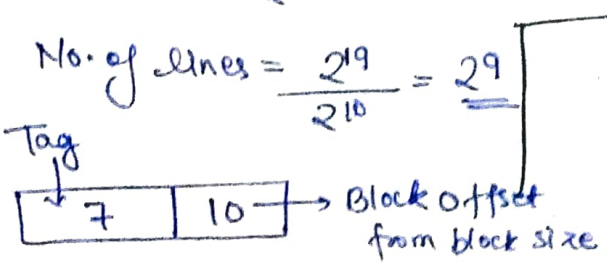
Ex – (Byte Addresable)

	MM Size	Cache Size	Block Size	Tag Size	Tag directory size
①	128kB	16kB	256B	9	$2^6 \times 9$
②	32GB	32kB	1kB	25	25×2^5
③	2^{17}B	512kB	1kB	7	7×2^9
④	16GB	Can't guess	4kB	22	$10 \times$ Can't guess
⑤	64MB	Can't guess	Can't guess	10	$10 \times$ Can't guess
⑥	Can't guess	512kB	Can't guess	7	$7 \times$ Can't guess

① Tag size = $\frac{2^{17}}{2^8}$ = 9 bits
T. Directory size = $\frac{2^{14}}{2^8} \times 9 = 2^6 \times 9$

② Tag size = $\frac{2^{35}}{2^{10}}$ = 25 bits
No. of lines = $\frac{2^{15}}{2^{10}}$ = 2^5

③ No. of lines = $\frac{2^{19}}{2^{10}}$ = 29



Similarly we can do ③, ⑤ and ⑥ question option.

→ No. of comparator required is equals to no. of lines in cache.

→ In ③ option size of main m/m is less than cache size. It doesn't make sense. So the question is logically not correctly framed.

→ To reduce no. of comparators we have set associative method because associative method was taking huge no. of comparators.

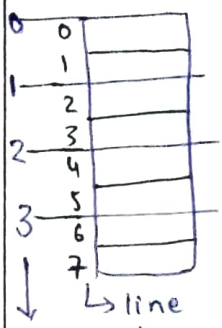
we devide cache into sets and we allow a block to sit in particular set but in a set it can sit anywhere. It reduce the comparison time.

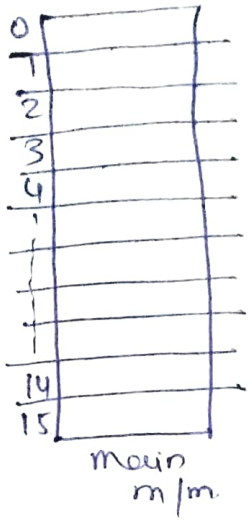
Set associative mapping

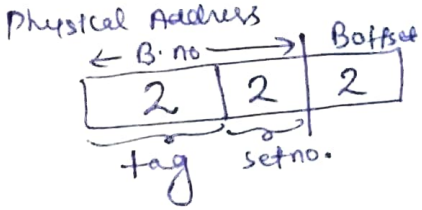
Ex → MM size = 64 B
Cache " = 32 B
Block " = 4 B
Set " = 2 ~~Blocks~~ lines
(2-way set associative bcoz 2 blocks are here).

No. of lines = $\frac{32\,B}{4\,B}$ = 8 lines

No. of sets = $\frac{8}{2}$ = 4 sets.

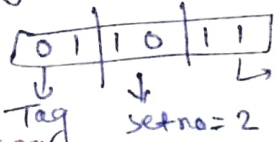






setno. = 2 bits bcoz of 4 sets

Tag = 2 bits.

Ex 

Tag = 2 bits bcoz (6-4) = 2
and PA = 2^6 = 64 block in m/m.

In a k-way set associative we need k comparators of size of tag.

→ Here we require two comparator of 2 bit size.

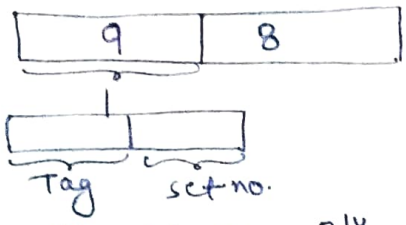
Problems on set associative mapping

MM size	Cache size	Block size	Tag bits	Set Associative (Given)
128kB	16kB	256 B	4	2-way
32 GB	32 kB	1 kB	22	4-way
8 MB	512 kB	1 kB	7	8-way
16 GB	64 MB	4 kB	10	4-way
64 MB	2^{18} B	can't guess	10	4-way
8 MB	512 kB	can't guess	7	8-way

Soln

① Block offset = 8 bits

Main m/m = 2^{17} B

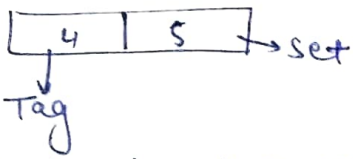


No. of lines = $\frac{2^{14}}{2^8} = 2^6$

In two way set associative

1 set contains 2 lines

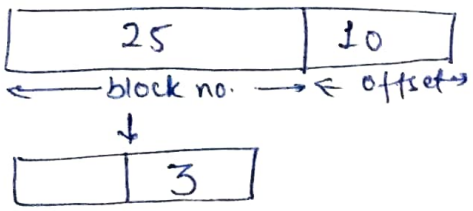
No. of set = $\frac{2^6}{2} = 2^5$ (5-bit)



No. of comparators required = 2.

② Block offset = 10 bits (2^{10}B)

32 GB = 2^{35} B PA



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

In 4 way set associative

No. of set bits = $\frac{2^5}{2^2} = 2^3$ (3 bits)

Tag bits = 25 - 3 = 22

No. of comparators = 4

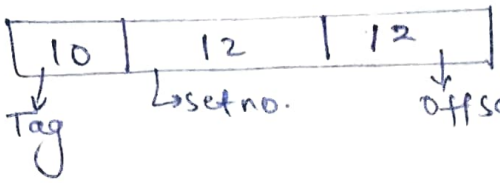
③ same as above ② and ①

Ans Tag bits = 7

Main m/m = 8 MB (2^{23}B)

2 mp

④ PA = 34 bits



No. of sets = 2^{12} = $\frac{\text{No. of lines}}{2^2}$

No. of lines = 2^{14}

Cache size = No. of lines * line size

= $2^{14} * 2^{12}$ B

= 2^{26} B (64 MB)

⑤ ~~PA = 2^{26} B~~, Tag bits = 10

wrong

we can't guess block offset.

Cache size depends on line size that is not given.

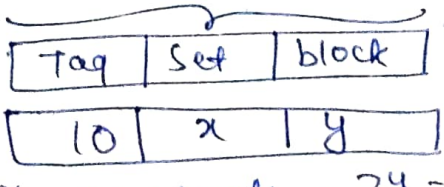
→ So, we can't guess bcoz there can be multiple combinations possible.

⑥ ~~wrong~~

we can't find Block offset without block size.

→ So we can't guess anything.

⑤



26 bits

2^x = no. of sets, 2^y = block size

x + y = 26 - 10 = 16

Cache size = No. of sets * No. of lines per set * line size

$= 2^x * 2^2 * 2^y$

↓ (4-way)

$= 2^{x+y+2} = 4 \cdot 2^{(x+y)}$

$= 4 \cdot 2^{16} = \underline{2^{18} B}$.

→ But we can't find block size.

⑥ Similar approach as ques. ⑤ we get

MM size = 8 MB

Gate 2012

Given, Cache size = 256 kB

4-way set associative used

Block size = 32 B, PA = 32 bits

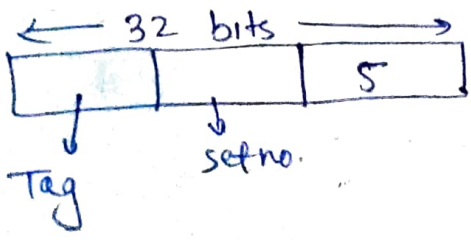
•) Each cache tag directory entry contains in addition to address tag, 2 valid bits, 1 modified bit and 1 replacement bit

To find:

① No. of tag bits.

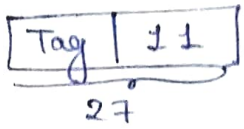
② Size of cache tag directory.

Soln

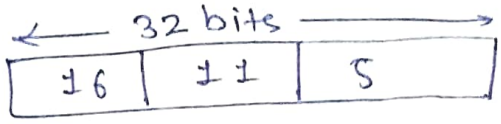


No. of lines $= \frac{2^{18}}{2^5} = 2^{13}$

No. of sets $= \frac{2^{13}}{4} = 2^{11}$



Tag = 16 bits.



Size of tag directory

$= (16+2+1+1) * \text{no. of lines}$

$= 20 * 2^{13}$

$= 160$ kB

Gate-2005 (Direct mapping)

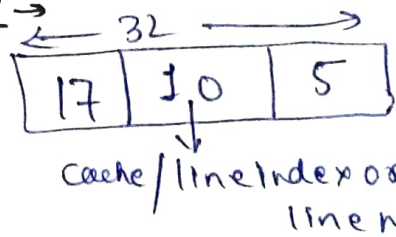
Given, Cache size = 32 kB

Block — = 32 B

PA = 32 bits.

Find no. of bits of cache indexing and tag.

Soln →



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

tag bits = 32 − 15

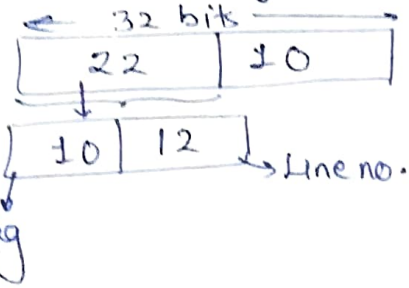
= 17 Ans//

Comparing all the mappings

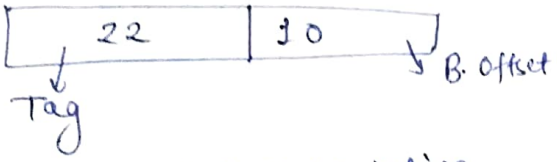
Ex → MM = 4GB = 2^{32} B
Cache = 1MB = 2^{22} B
Block = 1 KB = 2^{10} B

PA = 32 bits
No. of lines = $\frac{2^{22}}{2^{10}} = 2^{12}$

① Direct mapping

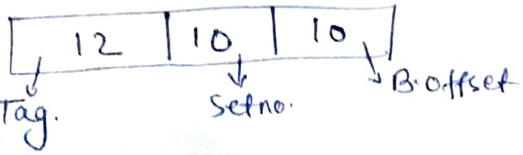


② Associative mapping



③ 4-way set associative

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



Direct mapping is k-way set associative with $k = 1$.

K-way set associative is fully associative with $k = N$ where, N is no. of lines.

→ k-way is a balance in between other two mapping.

	TAG	Comparators	Size of tag directory
Direct	10	1	10×2^{12}
Associate	22	2^{12}	22×2^{12}
k-way	12	4	12×2^{12}

Gate Questions

99) Main m/m = 2CM Blocks
Cache = 2C Blocks
2-way set associativity.

Soln No. of sets = $\frac{2C}{2} = C$ sets

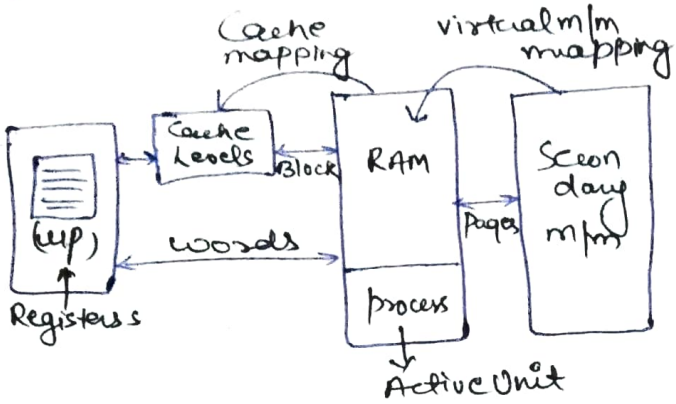
kth block will map to k mod C.

Note :- There are some videos with previous yr. gate. Do watch all the videos during solving practice questions bcoz we will definitely get a better solution.

→ Must watch all the videos.
→ Do some questions from standard books (10–15 Ques)

Memory Interfacing

Introduction

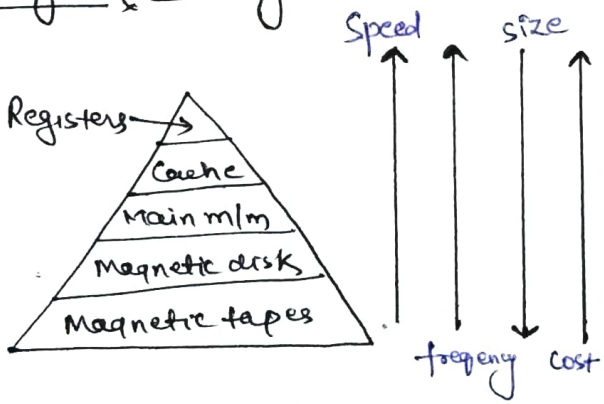


→ Speed of CPU measures in MIPS (Million Instruction per seconds)

→ Cache is a volatile m/m and very costly.

→ Registers are present inside CPU.

Memory Hierarchy



→ Higher the speed lower the access time.

Register implemented with flipflops
Cache implemented with static RAM. Main memory implemented with Dynamic RAM.

Cache, Main memory } Random Access
Magnetic Disk } Semi Random Access
Magnetic Tapes } Sequential acess

The purpose of main m/m is to bridge the speed mismatch between fastest processor to slow memory at reasonable cost.

→ SRAM faster than DRAM.

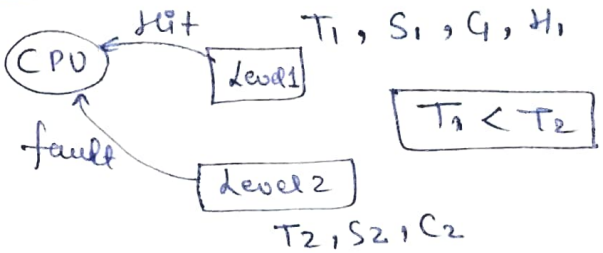
2-Level Memory

Information in ith level ⊂ (i+1)th level

→ If processor refers to ith level m/m is found then 'Hit' Otherwise 'Miss' or 'fault'.

→ There are two ways in which processor is connected to various levels of memory.

Case-1

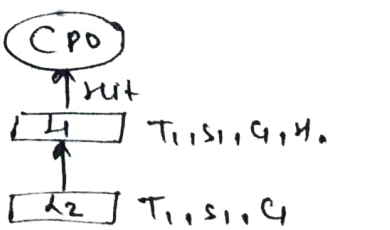


T → access time, S → size
C → cost per bit H → Hit ratio

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

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

Case-2



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

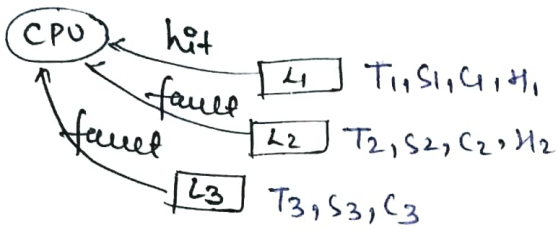
$C_{avg} = \frac{C_1 S_1 + C_2 S_2}{S_1 + S_2}$

→ In case of fault we have to go through both the levels.

→ Cost (C) measures in Rs/B, (Rupees per Byte) and size (S) in Bytes.

3-Level Memory

Case-1

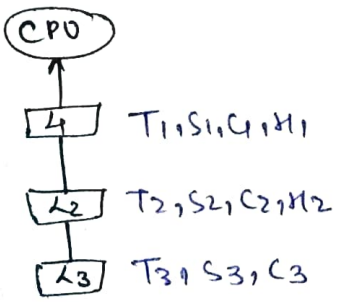


$(T_1 < T_2 < T_3)$

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

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

Case-2



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

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

In this case 1:

$T_1 \leq T_{avg} \leq T_3$

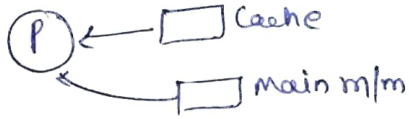
→ In both the cases H_3 for L_3 is always 1, in 3-level memory otherwise instruction will never be found.

→ In case 2:

$T_1 \leq T_{avg} \leq (T_1 + T_2 + T_3)$

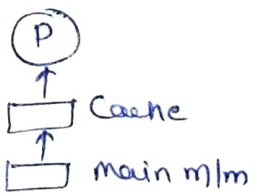
Ex: Cache and main m/m access time = 5 ns & 100 ns respectively
Cache hit rate = 0.8
Find avg. m/m access time?

Soln Case-1



$T_{avg} = 0.8 \times 5 + 0.2 \times 100$
$= 4 + 20$
$= 24$ ns

Case-2



$T_{avg} = 0.8 \times 5 + 0.2 (105)$
$= (4 + 21)$ ns
$= 25$ ns

Cache Replacement Algo.

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

→ Replacement policy are aimed to minimize miss penalty references.

Replacement Policy

- **Random:** No specific criteria to replace the block.
- **FIFO:** The block which entered first is the candidate for replacement.
- **LRU:** The block which is not reffered for longest time will get replaced.
- **LFU:** The block with fewer references is replaced.

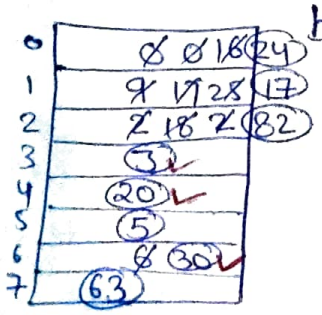
Ex: Direct mapped cache with 8 blocks (0-7). Blocks request in order,

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 block will not be in cache at the end of sequence.

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

Soln



block no. % 8 = line no.

→ 18 is not there in cache at the end of sequence.

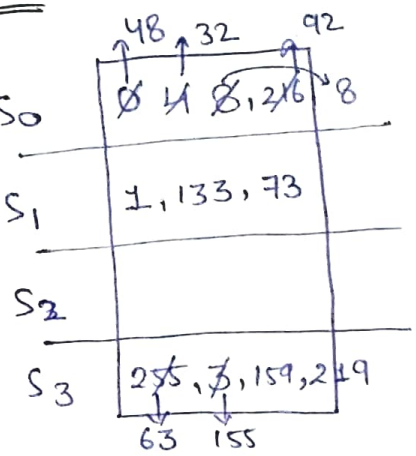
Ex → 4-way set associate mapping with 16 cache block. Blocks are requested in order.

(0, 255, 1, 4, 3, 8, 133, 159, 216, 219, 63, 8, 48, 32, 73, 92, 155).

Which one of the m/m block will not be in cache if LRU used?

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

Soln



Sno = (block no) mod (no. of sets)

↳ set no.

Set no = block no. mod 4

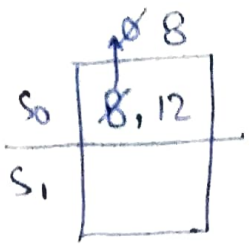
Ans → a), c) and d)

→ when 8 comes 2nd time in S0, we write it again bcoz we were following LRU. So, 2nd time 8 can't get replaced by 92.

Ex → Consider a small 2-way set associative mapping with total 4 blocks for choosing the block to be replaced use LRU. the number of cache misses for the sequence of block address

8, 12, 0, 12, 8 is _____?

Sol^n



Set no = k % 2 → block no.

8 ↓ Miss | 12 ↓ Miss | 0 ↓ Miss | 12 ↓ Hit | 8 ↓ Miss

There are '4' misses.

→ In the begging there were nothing in cache. So loading of first 8 and 12 was the miss bcoz we load in from main m/m.

Q. Consider a 2-way set associative mapping consisteny of 2^c memory blocks and 2c cache blocks the, cache location for the memory block k is.

Sol^n Cache size = 2c blocks
MM —— = 2^c blocks
Set size = 2 blocks.

Set no = cache location = k mod c

Sets = $\frac{2c}{2} = c$

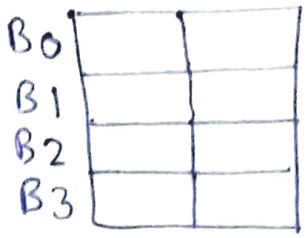
Ans → k mod c

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

a) what is the hit ratio for the statement

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

Sol^n Cache size = 32 words
block —— = 8 words,
No. of blocks = 4 blocks



Cache have 4 blocks of size 8 words each. Each element take 4 words. so each block can hold 2 array elements

In the statement

A[I] = A[I] + 2
↓ for writing (hit) | → for reading (miss)

A[I] refered two times first for reading and second for writing. Since cache is empty so when element choose for reading its a miss and then when it refferes for writing its a hit bcoz after a miss it will get loaded to cache.

→ Not complete solution bcoz of doubt.
→ watch the video in doconloaded video as 'Array and cache ex-1'

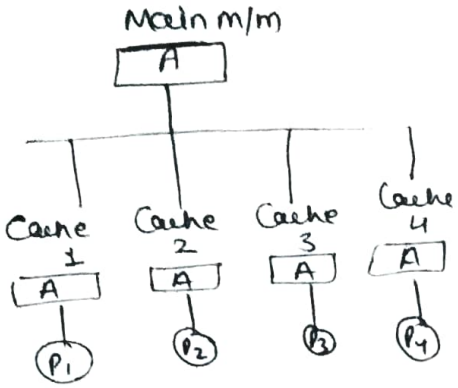
So on calling A[0] it will fill two elements in block as
A0[0] | A[1] and since each elements occur two times so it will occur as

A0	A0	A1	A1
Miss	Hit		

because as A0 comes all the 3 other will get a hit. similarly 50 pairs formed by 100 elements.

Cache Coherence Problem

Multiple copies of same data can exist in different caches at the same time, and if processors are allowed to update their own copies freely an unconsistent view of m/m can results



Each processor is running a job and all the jobs are accessing the common variable (A) at the same time. It may arise inconsistency while updating data.

Methods to avoid Cache coherence Problems (4 methods)

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

→ write through means update the value in main memory just after updating value in cache.

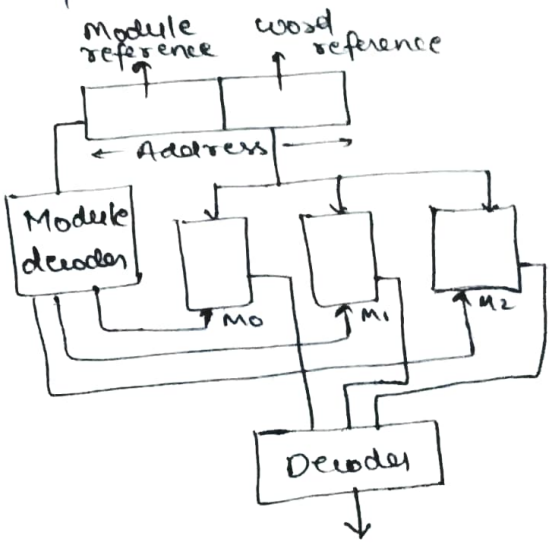
→ write back means update the value in main memory at the end when you either replace the block or flush out the memory.

→ write update means update the value of variable in all the cache at the same time.

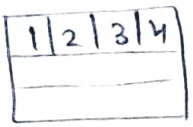
→ write invalidate means after update the value tells the other processors that the variable copy is invalidate.

Memory Interleaving

→ Reduces avg access time and improves data transfer rate.



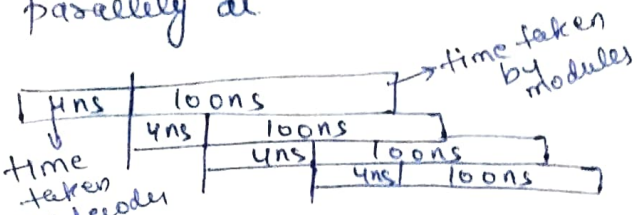
Consider the cache that have 4 words like, and searching each word take 100ns.



Assume you want to access 4th words then first you hit 1 it take 100ns. similarly upto 4 you will hit 2,3 then 4 it will take 4x100 ns = 400ns.

But if all modules are different as.

then, some time will take for modules decoder and some time will take by modules but everything will occur parallelly as



$(4 \times 4) + 100ns = 116ns$ only

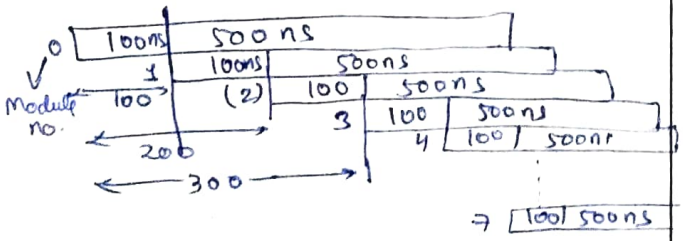
Imp

Gate: 2014 COA (1.48 PYQ)

Soln → Total 8 modules

time taken by modules = 500ns

time taken by decoder = 100ns



We are generating request for next modules after 100ns so,

In 100ns → 1 request

$$1s = \frac{1}{100 \times 10^{-9}}$$

$$1ms = \frac{10^{-3}}{100 \times 10^{-9}} = \frac{10^{-3}}{10^{-7}}$$

$$= 10000 \text{ request}$$

→ Solution of Gate 2006 is given but you can't understand anything from it. So try to do it by self.

Gate 2006: CPU has a cache of 64 bytes and main m/m have k-banks [See ques. in PYQ]

Soln

Each of size 2 bytes. All bytes are allotted sequentially. Total 48 bytes. All blocks accessed parallelly

→ 80ns to read one block.

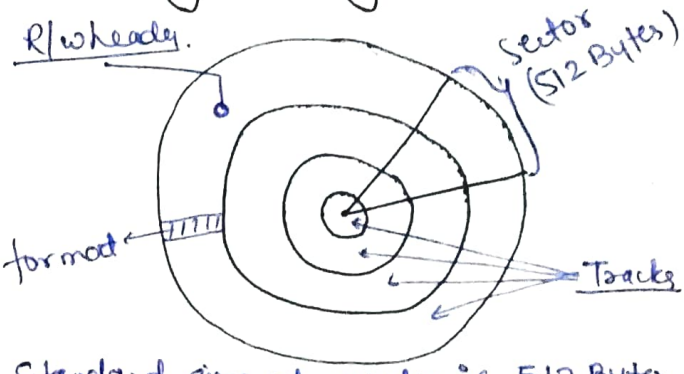
→ Decoder takes $\frac{k}{2}$ ns (12ns).

Total time = 12 + 80 = 92 because decoder activates all the block at a time.

Ans $2 \times 92 = 184$

Secondary Memory

R/W head.



Standard size of sector is 512 Bytes.

Total Capacity of hard disk equals to

$$(\text{No. of tracks}) \times \frac{\text{(No. of sectors) per Track}}{\text{No. of sectors}} \times \text{Bytes}$$

$$= \text{Bytes}$$

Some space get used in formatting so, we can't use the complete space for storage purpose.

Usable storage = (Total size) − (format size)

Time taken by R/w header to move from one track to another track is called 'seek time'

Time taken by header to reach particular sector in the track called 'rotational delay'.

(T_{avg}) access = $T_{seek} + T_{rd} + T_{trans}$

T_{trans} → time spend in target sector

If T_{seek} not given assume as '0',

for T_{rd} use $\left(\frac{1}{2} \text{ Rotation time}\right)$

2mp

Example

(1) A disk pack have 16 surfaces, 128 tracks / surface, 256 sectors per track and 512 Bytes / sector.

a) Find Capacity of disk pack?

Each disk have two surfaces and we can store data on both surfaces.

$$16 \times 128 \times \frac{T}{Surface} \times 256 \frac{sec}{T} \times \frac{512B}{sec}$$

$$\Rightarrow 16 \times 128 \times 256 \times 512$$

$$= 2^{28} = \underline{256\,MB}$$

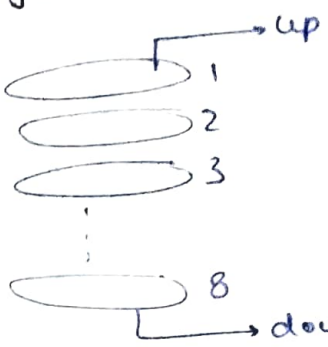
b) Bits required to address the sectors.

→ No. of sectors = $2^4 \times 2^7 \times 2^8$

$= 2^{19}$

So, 19 bits are required.

Note → In previous question we have 8 disk so, that we can get 16 surfaces as



In case if question says that we are not considering the boundary surfaces means (up of disk 1) and (down of disk 8) then we will add a new disk to introduce two more surfaces. In that case we get 16 surfaces from 9 disks.

c) If format overhead is 32 Bytes / sector. What is the formatted disk space?

Wasted space = $2^{19} \times 2^5$

$= 2^{24}$ Bytes

$= \underline{16\,MB}$

formatted disk space

$= (256 - 16)$ MB

$= \underline{240\,MB}$

d) Let the diameter of innermost track is 21 cm. What is the max. recording density.

→ Each track have same no. of sectors and each sector have same size so, outer track have low recording density bcoz

m/m divided in High area as Compare to Innermost track.

Max recording density → Inner most track.

Min. recording density → Outer most track.

Track capacity = 256×512
$= 2^9 \times 2^8$
$= 2^{17}$ Bytes.

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

Max density = $\frac{128\ KB}{66}$
$= 1.9$ KB/cm.

(e) The disk is rotating at 3600 Rpm. What is the data transfer rate?

for each surface we can read one track and for 16 parallel surface we can read 16 surface at a time.

→ 3600 Rotation → 1 m

1 Rotation = $\frac{1}{60}$ sec.

and it means we read the complete track once in $\frac{1}{60}$ sec.

So, $\frac{1}{60}$ sec → read 128 KB

→ 1 sec → 128×60 KB

→ (Data Rate)total = $128 \times 60 \times 16$
$= 120$ MBPS

→ Bcoz there was nothing is given. So we go with 16 R/w heads for 16 surfaces.

(f) Using one read/write head, the disk is rotating at 6000 Rpm. What is data transfer rate?

6000 Rotations → 60 sec

1 Rotation → $\frac{1}{100}$ sec

$\frac{1}{100}$ sec → 128 kB

1 sec → 128×100 kB

(Data Rate) ≅ 128×100 kB
≅ 13 MBPS.

(g) If rotation speed is 3000 rpm, what is average access time with seek time of 11.5 m sec?

Taccesstime = Tseek + Trotation delay + T_{trans}^{0} (Data transfer time)

$= 11.5 + \frac{1}{2}(RT)$
$= (11.5 + 10)$ ms
$= 21.5$ ms.

(h) Find the avg. access time for transfering 512 Bytes of data with following specification.

Avg seek time = 5 msec
disk rotation = 6000 rpm
Data rate = 40 KB/sec
Controller overhead = 0.1 msec

Cont—

$(T_{access})_{avg} = T_{seek} + T_{rd} + T_{trans} + 0.1$

$\Rightarrow 5 + 0.1 + T_{rd} + T_{trans}$

6000 → 60 sec

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

$\frac{1}{2}R = \frac{1}{200}$ sec $= 0.5 \times 10^{-2} = 5$ ms

T_{rd}

$T_{trans} = \frac{512}{40 \times 1024} = \frac{12.8}{1024}$ ms

$= 12.5$ ms

$(T_{avg}) = (5 + 0.1 + 5 + 12.5)$ ms

$= 22.6$ ms

Q A certain moving arm disk storage with one head has following specification.

No of tracks / surface = 200

disk rotation speed = 2400 rpm

track capacity = 62,500 bits

avg. latency = p msec

data transfer rate = q bits/sec

Find the value of p and q?

Soln $p = \frac{1}{2}$ (Rotation time)

2400 → 60 sec

$1R = \frac{1}{40}$ sec

$\frac{1}{2}R = \frac{1}{80}$ sec

$\boxed{p = 12.5}$

(2 × 12.5 ms) → 1R → 1 Track

(2 × 12.5) → 62500 bits

25 ms → 62500 bits

$1s = \frac{62500}{25 \times 10^{-3}}$

$= 2.5 \times 10^6$ bits/sec

$\boxed{q = 2500000}$

Q: Gate: A HD has 63 sectors per track, 10 platters ——

i) <400, 16, 29>

ii) <2039>.

Note: Do it by self first to

Machine Instructions and Addressing Modes

(12)

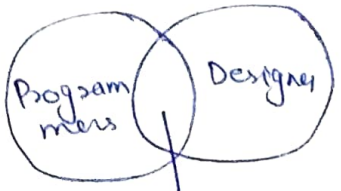
Introduction

Embedded System ⇔ (µP)

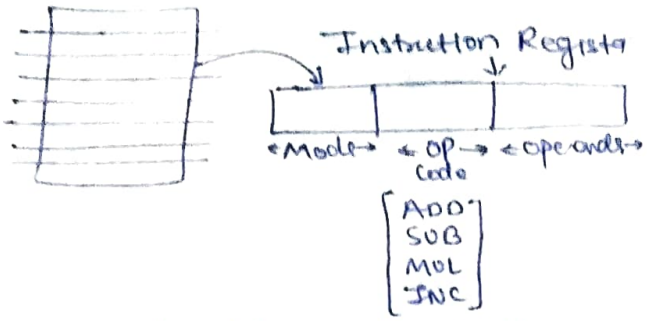
Computer System

Main m/m, registers, ALU and control unit. We select all to design a computer architecture according to cost selection might vary.

Generate control signal according to instruction called instruction decoding.



Machine Instr. discussed by both before designing.



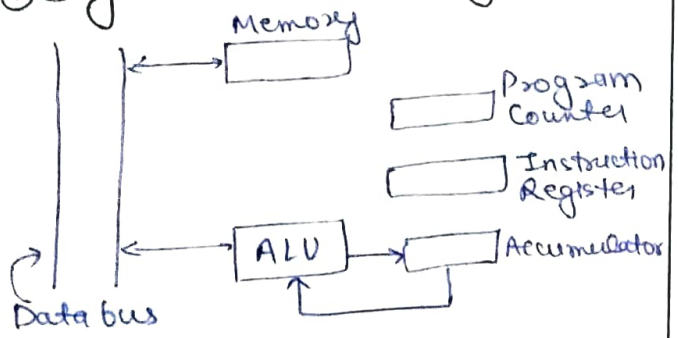
→ Mode defines about the operands.

Computer Organisations based on registers

Types of CPU Organisations -

1) Single accumulator organization.
2) General register organization
3) Stack Organization.

① Single accumulator organization



→ we use single address instruction, means we will have either 0 instruction or 1 instruction.

Ex → IHC A; is a 0 address register bcoz it has no need of m/m address.

ADD X; → means add x to value of accumulator so, its a 1 address register.

m[x] → operand present in m/m 'm' at address 'X'.

ADD X = m[x] + Accumulator

→ And finally value get stored in Accumulator.

② General register organisation.

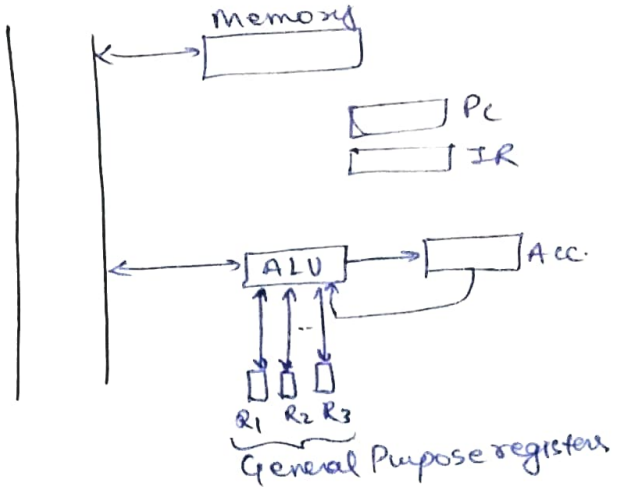
→ We have lot of registers for general purpose.

Ex- ADD R1, R2, R3;
R3 ← R1 + R2

→ No. of instruction for executing program is very less, as compare to single accumulator.

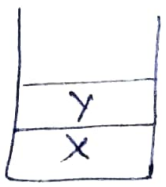
→ It is costly and faster.

→ We can have 2 address or 3 address instruction.



③ Stack Organization

→ Only zero instruction possible except Push as POP.



Push X
Push Y
Add

→ We just give opcode Add without giving any operand.

→ No. of operands in instruction decide the size of instruction.

Operands ↑ ↔ Size ↑

Addressing Mode

opcode	operand	mode

mode defines that how to treat operand.
operand can be variable and constant both but generally refers variable.
opcode is operation code to perform operation.
To reduce size of instruct size we keep the address in register despite of m/m.
→ we can manage pointers, loops, counters etc.

Implied Add. Mode (Implicit Mode)

→ Used for 0 or 1 addres instr.
→ operand specified implicitly in the instruction.

Ex → INC A, ADD (Addl in stack organization)
CLC A, CL A, (complement)
↳ Clear carry

Immediate Addressing Mode

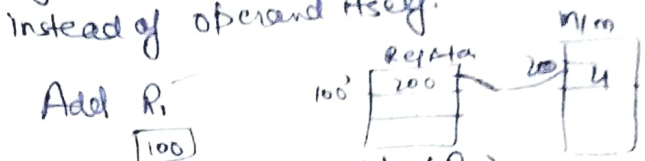
→ No effective addres require
→ operand provided as constant.
→ We use constant.
Ex → Add R1, #3 → data
→ We are not using variable here.
→ Constant size equals to length of operand.

Register Add. Mode

→ operand present in register.
→ Register no. written in Instruction
→ Data present inside the register.
→ Reduce the size of operand and instr
→ High speed access.

④ Register Indirect Mode

Register contains address of operand instead of operand itself.



→ Effective addres required.
→ Generally registers are less as compare to m/m, so give reference of m/m using registers to reduce the bits in operand.

⑤ Auto Increment or decrement

→ Special case of register indirect add. mode.
→ for continous sequential access.
→ Register are collection of flip flops and we can use them as counters.

⑥ Direct addressing mode

Ex ADD 500;
→ Absolute addressing mode.
→ Actual addres given
→ Use to access variables.
→ Direct m/m addres given.
→ Disadv → ① Large instruction size
② if addres feild is fixed size then variable value size is fixed.

⑦ Relative addressing mode

→ Use in program control instr like, looping, branching etc.
EA = Offset + Program counter
→ Beware of the value of Program counter.

Base register addressing mode

→ Used in 'program relocation'

→ Useful in swap in/out of process.

→ After swapping process add. might get changed.

EA = Base Reg. Value + Displacement

↓

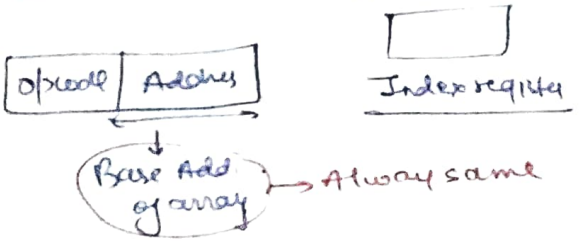
starting Add.

(8) Indexed Add. Mode

→ Used to implement array efficiently.

→ Multiple registers required to implement.

→ Any element can be accessed without changing instruⁿ

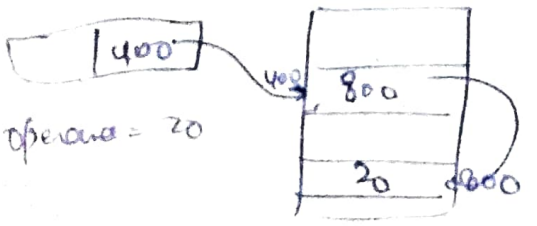


EA = Base Add + Index Register

(9) Indirect Add. Modes

→ Use to implement pointers and passing parameters.

→ 2 m/m access required.



EA = AC + M[M[X]]