with

2) Explain working of RIP w/ example.
3) Draw tables, routes and how convergence occurs.

CO Lecture

set associative mapping

OFFset → which block out from (La we have pick

ory address is classified into 3 things

Tag	Index	OFFset

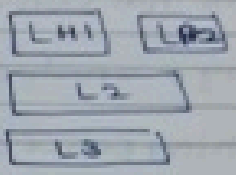
Tag → whether data reside in that cache levels not.

Hit rate and Miss rate?

↳ = 1 − Hit Rate

↓

particular rate where bit is are found in one level of cache.



* Direct Map, Full Map

L1 → in KB
L2 → 10 ~ 100 KB
L3 → 1 ~ 3 MB

* CPU ⟷ L1 (Fastest cache) ⟷ L2 (Fast cache) ⟷ L3 (Less fast cache) ⟷ Main memory (Slow)

Three-level cache organisation

① Pages are retrieved from secondary memory to main memory.

② From main memory blocks are taken to the CPU.

* Set Associative Mapping

↳ combination of direct and associative approaches.

① Cache consist a no. of sets.

② Each set contains a no. of lines.

③ A given block maps to any line in a given set.

④ Eg: 2 lines per set.

- 2 way associative mapping.
- A given block can be in one of 2 lines in only one set.

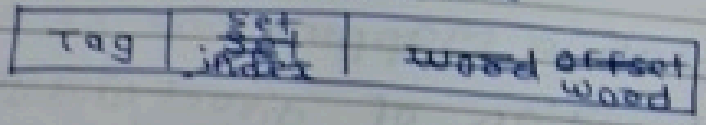
→

off chip cache
↳ this is not concept in current generation
cache → always onchip in recent generation

Data searching process:

```
if in L1 cache:
    break
elif in L2 cache:
    break
elif in L3 cache:
    break
else:
    go to main memory and get the data
```

* Memory address structure

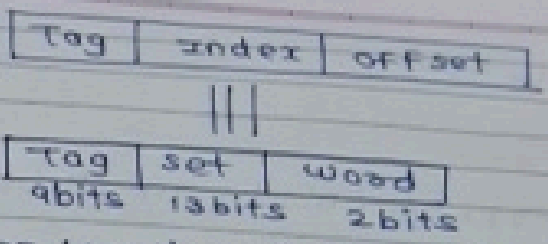


① Direct map
② Full map
③ Set associative → can be 1-way, 2-way, 4-way, 8-way set associati

Tag → denotes whether particular block is present in the one level cache or not.

→ if present, then find in cache

→ if not present, then find in next level.



① Address length = (s+w) bits.
② No. of address = 2^{s+w} bytes or words.
③ Block size = line size = 2^w words or bytes
④ No. of blocks in main memory = 2^s.
⑤ No. of lines in set = k.
⑥ No. of sets = v = 2^d
⑦ No. of lines in cache = m = kv = k * 2^d.
⑧ Size of cache = k * 2^{d+w} words or bytes
⑨ size of tag = (s-d) bits.

Q] A set associative cache consist of 64 lines or slots, divide into 4 lines set. Main memory contain 4K blocks of 128 words each. show the format of main memory address.

→ 16 lines | 16 lines | 16 lines | 16 lines. $k = \frac{64}{4}$

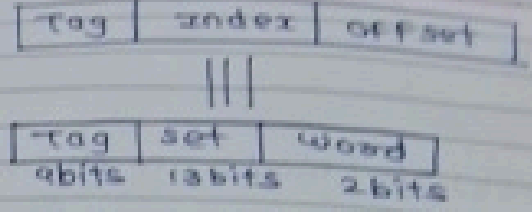
S1 | S2 | S3 | S4

Main memory = 4 X 128 Kb

4 X 1024 = 2^S

$\therefore S = 12$

Block size = 128 $w = 7$



① Address length = (s+w) bits.
② No. of address = 2^{s+w} bytes or words.
③ Block size = line size = 2^w words or bytes
④ No. of blocks in main memory = 2^s
⑤ No. of lines in set = k.
⑥ No. of sets = v = 2^d
⑦ No. of lines in cache = m = kv = $k * 2^d$.
⑧ Size of cache = $k * 2^{d+w}$ words or bytes
⑨ size of tag = (s-d) bits.

07. A set associative cache consist of 64 lines or slots, divide ifto 4 lines set. Main memory contain 4K blocks of 128 words each. Show the format of main memory address.

→ 16 lines 16 lines 16 lines 16 lines $k = \frac{64}{4}$
S1 S2 S3 S4

Main memory = 4 X 128 kb

4 X 1024 = 2^S

∴ S = 12

Block size = 128

$2^w = 128$

w = 7

→ Tag = 12 − 2 = 10 bits

* Direct mapped
  └ (Block address) % (Blocks in cache)

Blocks is a power of 2
Use low-order address bits.

* Full Map

Block characteristic — Tag
                     — valid [1 = present, 0 = not present]
                     — Dirty bit

Eg: 8-blocks, 1 word / block, direct mapped.

Intial state:

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

* Distribution

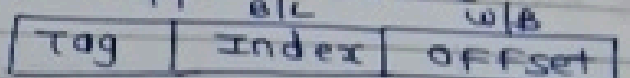
23/10/23
Monday

Co Lecture

* Principle of Inclusion.

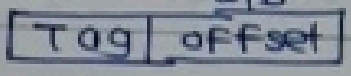
* Direct Mapped cache

→



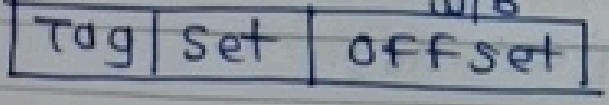
Block mod # of ... Miss rate is high

* Fully associative / Fully Mapped.



* set associative

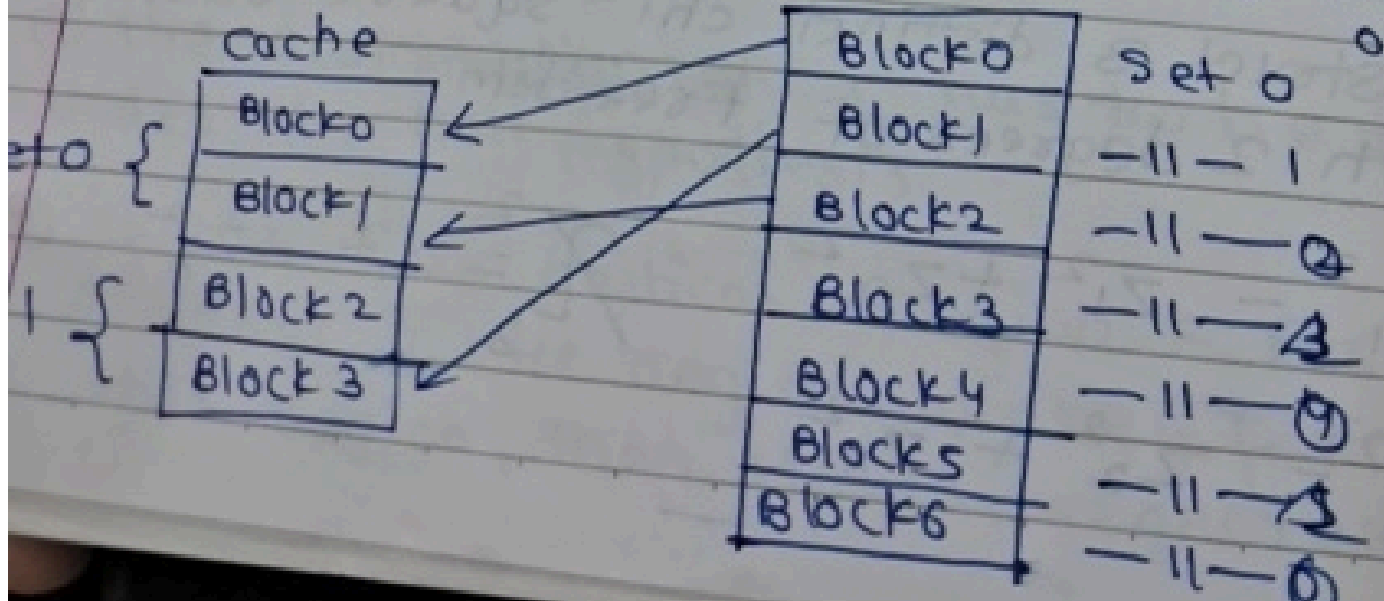
→



Block mod # no of sets.

k-set associative

So it can go in any on... of give... blo...



12 bit address

16 bytes per block ⇒ 4 LSB's used to determine the desired byte/word offset within the block

* Fully associative address scheme
- Any block from memory can be put in any cache block i.e [no mapping scheme]

↳ condition: cache block where it is going should be empty.

When all blocks fitted then to accomodate new data → either we can use FIFO or LIFO scheme to remove ~~it~~ already. ~~a block from~~ empty data from the block, and then place data there.

↓

[Here Dirty bit is also important]

↙ IF DF=1, then store the data to be evicted into main memory and then evict that.

↘ IF DF=0, directly evict it.

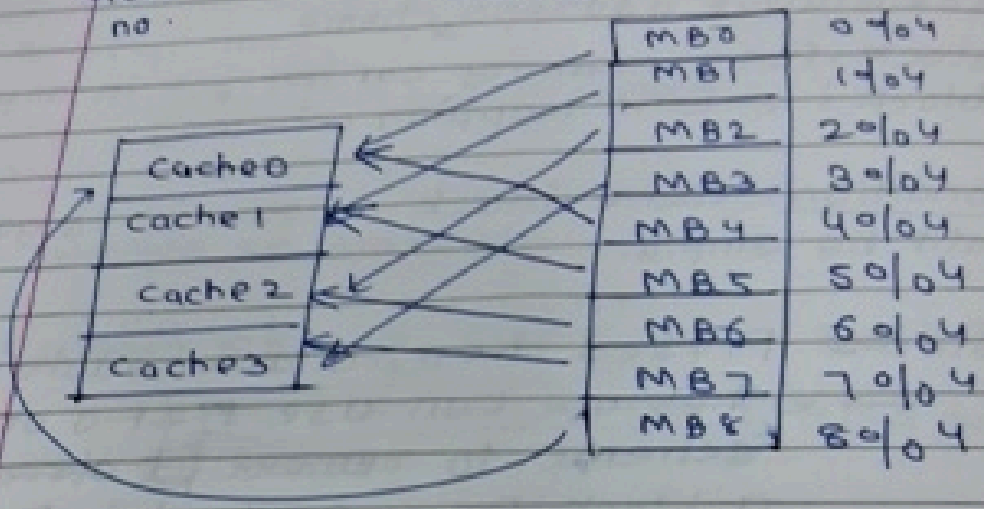
⊛ LFU and LRU policies are also imp.

* Also random is there Replacement.

⊛ Block placement v/s Block replacement

Dirty bit and Tag ⇒ used to determine whether a data is in cache or not.

* Direct mapping
↳ 0 mod 4
↓ ↳ no. of blocks in cache.
Block no.



* Direct mapping implementation
→

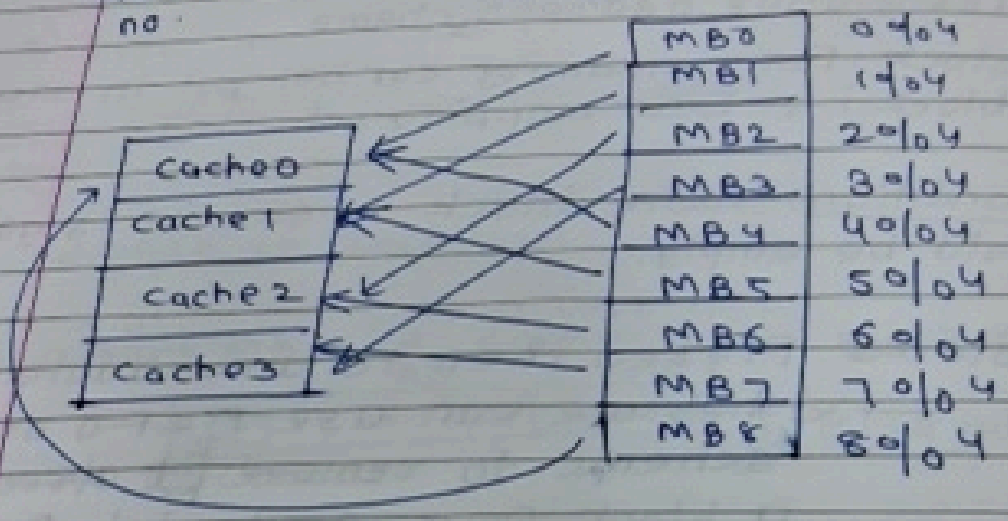
* set Associative Mapping

* Summary
→ ① Fully Associative
- Most Flexible
- Longest search time O(N)

② Direct Mapped cache → More eviction
- Least Flexible (more evictions) more
- shortest search time O(1). miss

→

* Direct mapping
↳ 0 mod 4
↓ ↳ no. of blocks in cache.
Block no.



* Direct mapping implementation
→

* set Associative Mapping

* Summary
→ ① Fully Associative
- Most Flexible
- Longest search time O(N)

② Direct Mapped cache → More evictions
- Least Flexible (more evictions) more
- shortest search time O(1). misses

→

③ K-way set associative

- Compromise
  - 1-way set associative
  - N-way set associative

- search time is O(K) [usually small enough to be done in parallel => O(1)].

e.g. this problem is written on pg no ③ at bottom

→ Here 16 block set will be formed. Main memory has 4K blocks.

↓

Each block has 128 words.

Block 0 — S0 — 4 lines — Block 63

Block 0 — S1 — 4 lines — Block 63

S2 — 4 lines

S3 — 4 lines

① Therefore 4 bits are needed to identify set no.

② Main memory has 2^{12} (4K blocks) = 4K blocks

So it generate 19 bits.

↓

Here set + tag length = 12 bits

no. of blocks in main memory

Block 0 — S15 — 4 lines — Block 63

Each block has 128 words

So we need 7 bits for it specifying words

TAG	SET	WORD
8 bits	4 bits	7 bits

Q3. consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a] How is a 16-bit memory address divided into tag, line no. and byte no?

→ main memory = 2^{16} bytes → 16 bit address.

Here each block has 8 bytes.

$\frac{2^{16}}{8} = 2^{13}$ blocks of main memory.

∴ Set length + tag length = 13 —①.

Here cache has 32 lines. Here

Tag	~~Offset~~ Index	~~Word~~ offset
8 bits	5 bits	3 bits

$32 = 2^5$

↓

are no. of lines in directly mapped.

b] Into what line would bytes with each of following will be stored?

→

① 0001 0001 | 0001 | 101 1

Tag = 17 | Offset = ~~5~~ 3 | word = 3.

~~this means~~ ~~17th block~~

Line 3

Q) consider a machine with a byte addressable main memory of 2^{16} bytes and block size of 8 bytes. Assume that a direct mapped cache consisting of 32 lines is used with this machine.

a) How is a 16-bit memory address divided into tag, line no and byte no?

→ main memory = 2^{16} bytes → 16 bit address.

Here each block has 8 bytes.

$\frac{2^{16}}{8} = 2^{13}$ blocks of main memory.

∴ Set length + tag length = 13 —①.

Here cache has 32 lines. Here

Tag	~~Offset~~ Index	~~Word~~ Offset
8 bits	5 bits	3 bits

$32 = 2^5$

↓

are no. of lines in directly mapped.

b) Into what line would bytes with each of following will be stored?

→

ⓐ 0001 0001 | 0001 | 1011

0001 0001 → Tag = 17

0001 → Offset = ~~5~~ 3

1011 → word = 3.

↓

Line 3

~~This means~~

No. of bytes stored in cache block = No. of bytes stored in main memory block.

c] Suppose the byte with address 0001 1010 0001 1010 is stored in cache what are the addresses of other bytes stored along with it.

d] How many total bytes of memory can be stored in the cache?

e] Why is the tag also stored in cache?

→ c] 0001 1010 0001 1010

Line 3 word no. 2

So word stored in this

0001 1010 0001 1000 (0th word)
0001 1001 (1st word)
0001 1010 (2nd word)
0001 1011 (3rd word)
0001 1100 (4th word)
0001 1101 (5th word)
0001 1110 (6th word)
0001 1111 (7th word)

d] 32 x 8 = 256 bytes.

32 lines

each line/block will store 8 bytes.

e] Because two items with two different memory addresses can be stored in the same place in the cache. The tag is used to distinguish between them.

—

lab3 | MSD Lecture |

Data Analytics Steps

ta → Data → Data preprocessation → choosing learning algorithm
llection

↓

Training model

↓

Evaluation of Model

↓

Predictions

Why do we need data processing?

Real world data is noisy, missing values, incomplete, etc.

Erroneous data, unordered, unstructured and unformatted.

se-processing steps:

$$t(x) = \frac{\bar{x} - \mu}{s_x/\sqrt{n}} \sim t_{n-1} \Rightarrow$$ for variance σ^2 unknown

Here $s_x^2 = \frac{1}{n-1}(x_i - \bar{x})^2$

—

28/10/22
Wednesday

CO Lecture

a) consider a memory system that uses a 32-bit address to address at byte level, plus a cache that uses a 64-byte line size.

a) Assume a direct mapped cache with a tag field in address of 20 bits. Show address format

→ Block size in main memory = Block size in cache $= 2^6 = 64$.

No. of blocks in main memory $= 2^{32-6} = 2^{26}$.

∴ Tag + Index = 26

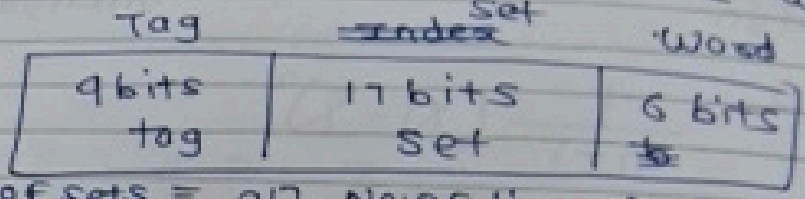
So:

Tag	Index	Offset.
20 bits	6 bits	6 bits

b]. Assume an associative cache. Show the address format and determine following parameters

a] 4 way set-associative cache with tag = 9 bits

L



No. of sets = 2^{17} No. of lines in cache = $2^{17} \times 2^2$

soln of b]. Tag = 26 bits Word = 6 bits.

each set has 4 lines.

c.]. consider a computer with following characteristics: total of 1MB main memory, word size = 1 byte, block size = 16 bytes [one block has 16 words] and cache size of 64 Kbytes.

a]. Give main memory address of F0010, 01234 and CABBE give all things for direct mapped cache.

→ 2^{20} → 1 Mb.

↳ so address length = 20 address lines.

20 - 4 = 16 so their are 16 blocks in main memory.

For representing words.

For representing lines

In Also cache, block size = 16 bytes — [two of bytes one byte]

Tag	Index	offset
4	12	4

As one block has 16 words in the cache as well so for 16 words we need this.

This should be also = 16 bits for blocks.

A → 10
B → 11
C → 12
D → 13
E → 14
F → 15

① For F0010

1111 0000 0000 0001 0000

Tag | Index | offset

② For 01234.

0000 0001 0010 0011 0100

Tag | Index | offset

③ For CABBE

1100 1010 1011 1011 1110

Tag | Index | offset

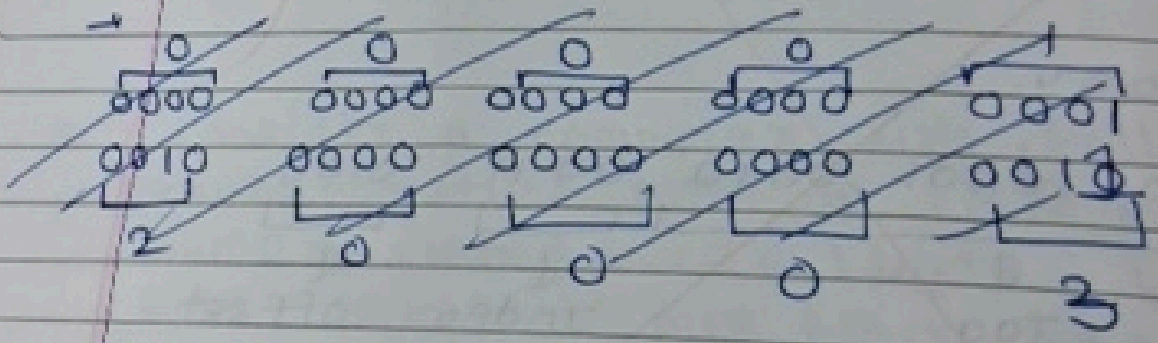
① ~~No of~~ block size in cache should always be equal to block size in main memory

$$\#\text{block in cache} = \frac{2^{16}}{2^4} = 2^{12}$$

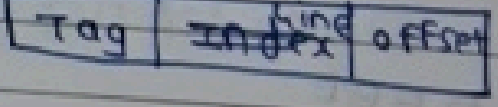
$$\#\text{ of block in MM} = \frac{2^{20}}{2^4} = 2^{16}$$

$$\text{Tag} = \frac{2^{16}}{2^{12}} = 2^4 \rightarrow \text{Tag}.$$

b) Give any two main memory address with different tags that map to same cache slot for a directed-mapped cache



In directly mapped:



↓

For this question for 2 addresses.

Tag ↓ Take different

Line ↓ should

Offset ↓ may or may not be same

Eg: 0FFFF and 3FFF1.

c) No. of blocks in main memory can be or cannot be equal to no. of blocks in cache (2) main memory

d) For the main memory addresses of F0010 and CABBE, give the corresponding tag and offset values

→ As this is fully associative, here their is no lines distinction.

F0010

1111 0000 0000 0001 0000

Tag | offset

CABBE

1100 1010 1011 1011 1110

Tag | offset.

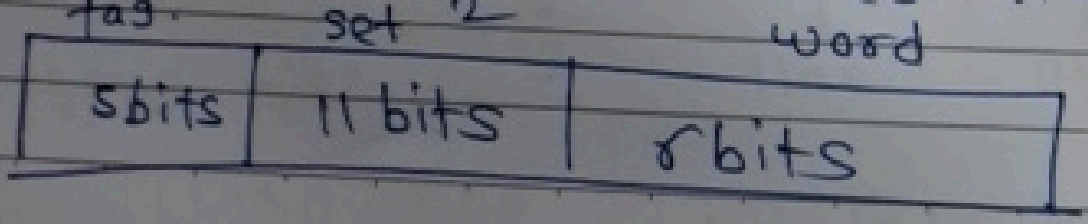
d) For the main memory addresses of F0010 and CABBE, give the corresponding tag, cache set and offset values for a two-way set associative

→ No. of blocks in cache $= 2^{12}$.

Each Set has 2 blocks.

∴ So total no. of sets $= \frac{2^{12}}{2^1} = 2^{11}$.

For representing 11 we need 11 bits set



↳ similar to what we do in page replacement policy in OS

* Replacement Algorithms.
  1. FIFO and LIFO
  2. LRU and LFU
  3. Random

↳ ① on miss, a new block must be brought in.
② this requires evicting a current block residing in the cache.

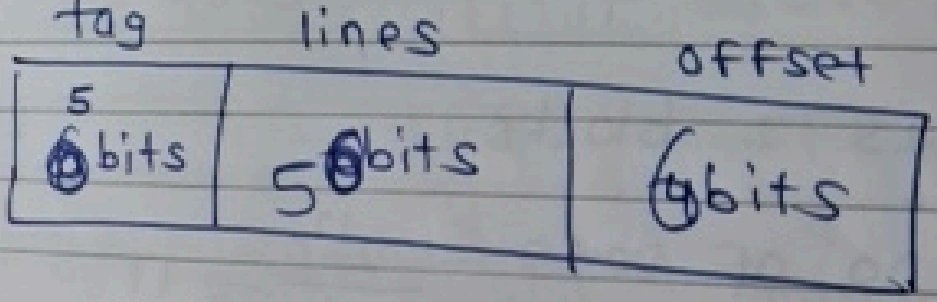
Page Fault in OS	≈	Block miss in cache

Q.] A computer system use 16-bit memory address [2^{16} addresses = 64KB memory] It has a 2KB cache in a direct-mapped fashion with 64 bytes per cache block. Give

a] Calculate the structure

↳ $2KB = 2^{12}$ ⟹ 12 address lines in cache.

$\frac{2^{12}}{2^{6}} = 2^{6}$ blocks in cache.



Memory Address structure.

(b) Initially cache is empty, for each of above address (in decimal) indicate hit as miss.

128, 144, 2176, 2180, 128, 2176

3 Cs → ① Cold or compulsory
② Capacity
③ Conflict
④ Coherence
} Types of cache misses.

① For $(128)_{10} = (0000\ 0000\ 1000\ 0000)_2$
tag | lines | offset

since initially cache was emty so it is a miss.
After this miss, tag field for cacheblock 00010 is set to 00000.

② For $(144)_{10} = (0000\ 0000\ 1001\ 0000)_2$
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
tag | Lines | offset.

Since tag field for cacheblock 00010 is 00000 before this access, this will be a cache hit.

③ For $(2176)_{10} = (0000\ 1000\ 1000\ 0000)_2$.
Tag | line | OFFSENT.

Here miss.

so line 00010 is not set to tag 00001

④ $(128)_{10} = (0000\ 0000\ 1000\ 0000)_2$

tag line offset

Here miss at 128 because at line 00010 tag was 00001 but it [due to 2176] the should what we want was 00000.

So miss And now line 00010 has tag 00000.

⑤ $(2176)_{10} = (0000\ 1000\ 1000\ 0000)_2$

tag Line offset

Here also miss

(As tag at block 00010 tag is 00000.

↓

So it is miss.

(And now it will get updated.

26/10/23
Thursday

AI Lecture

* → Horn Normal Form in FOL [First orders logic

Generalised modes ponen rule:

[Incomplete]

6|10|2|3

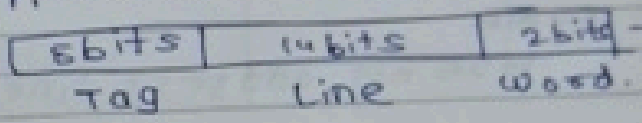
Tuesday

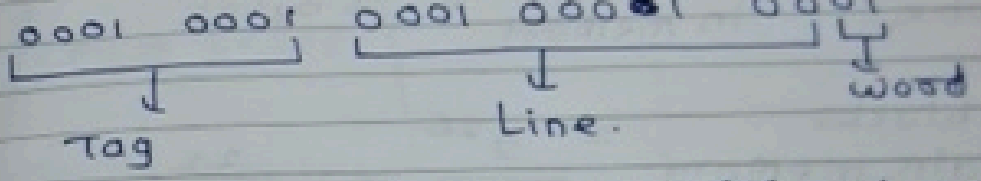
CO Lecture

2. A set-associative cache consists of 64 lines of 60 slots, divided into four-lines set. Main memory address contains 4K blocks of 128 words each. Show the format of main memory.

Q1] For hexadecimal main memory addresses 111111, 666666, BBBBBB, show following info in hexadecimal format:

a] Tag, line and word values for a direct-mapped cache using format of fig 4.10





Similarly you can do for 666666 and BBBBBB

Q2] A two way-set associative cache has lines of 16 bytes and a total size of 8KB the 64 MB main memory is byte addressable show format of main memory address.

→ Main memory = 64 MB
↳ So need 26 bit address.

Block size in cache = Block size in memory = 16 bytes
↳ So requires 4 line

∴ No. of blocks in main memory $= \frac{2^{26}}{2^4} = 2^{22}$

No. of of block in cache $= \frac{2^3 \times 2^{10}}{2^4} = 2^9$

13bit	9bit	4bit

↳ Wrong soln

main memory = 64MB
↳ 26 bit address line.

Block size in cache = Block size in memory = 16 bytes.

No of blocks in main memory $= \frac{2^{26}}{2^4} = 2^{22}$

Set tag =

— " — in cache $= \frac{2^{13}}{2^4} = 2^9 = 512$

Each set has 2 blocks.
↳ ∴ 256 set
↳ To represent this we nee
8 lines.

tag	set	word offset
14 bits	8 bits	4 bits

→

Similar to → Dirty read, Dirty write problem in the DBMS

→ Shared memory problem in OS

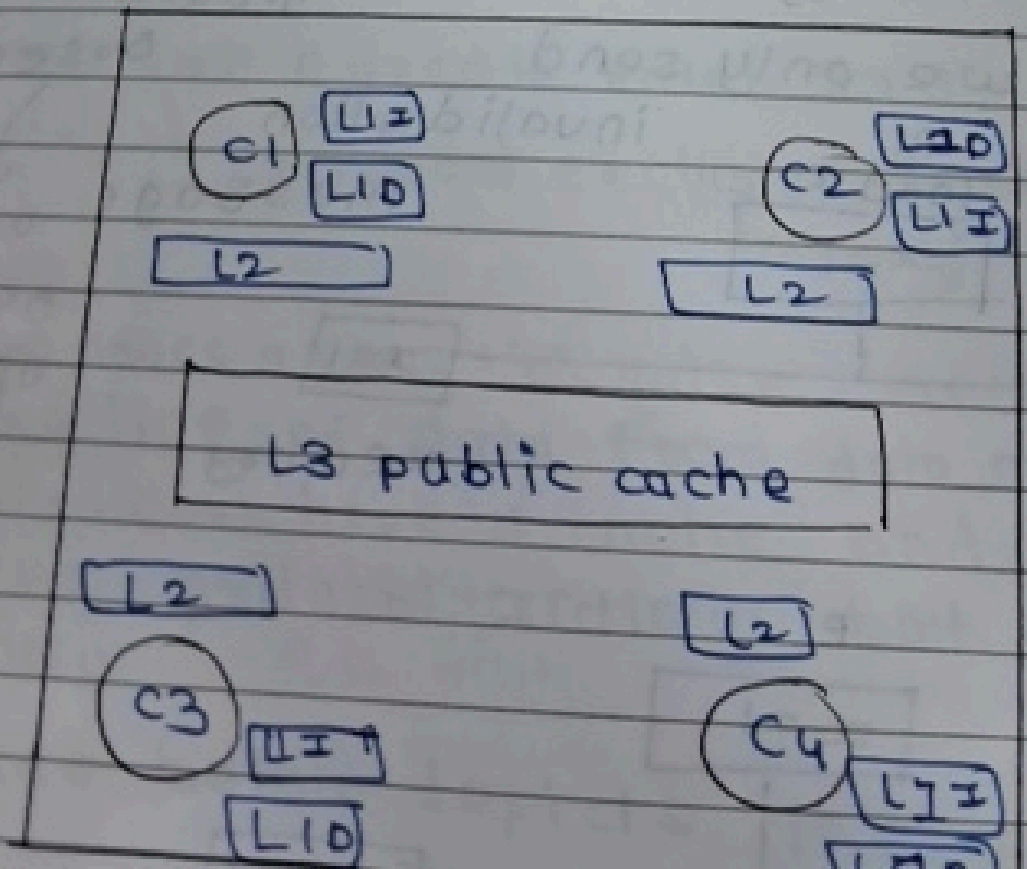
cache coherence

Protocol to solve this: Cache coherence protocol

Time step	Event	CPU A's cache	CPU B's cache	Memory
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A writes 1 to X	1	0	1

L stale copy

Main memory



{L1, L2}

{L3} p

* Cache coherence protocol → Hardware based → Snooping protocol, Directory protocol
→ Software based

Necessary conditions to keep memory system coherent:

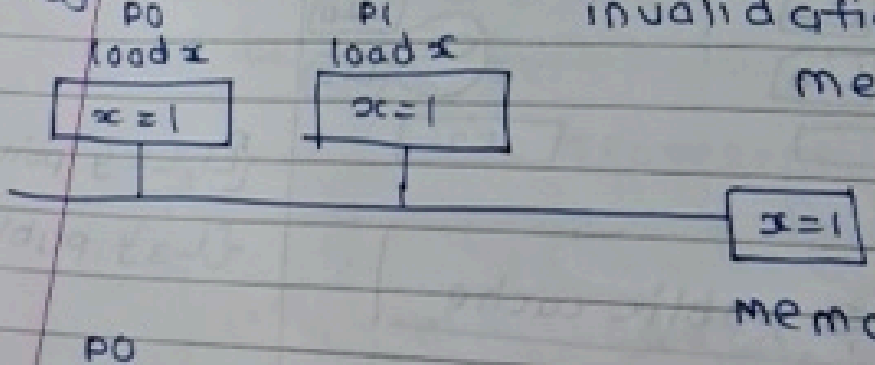
① Read: P to X not followed by a write P to X and write by another processor.

② Read: P to X followed by write: P to X and no write by another processor.

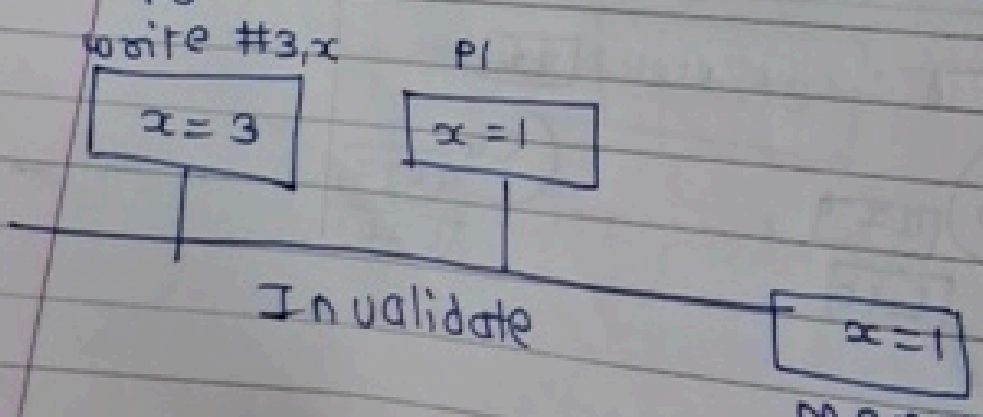
③ [Incomplete]

Snoopy Bus [Incomplete] [means checking] → Invalidation Based (only we give invalidation msg)
→ Updation Based

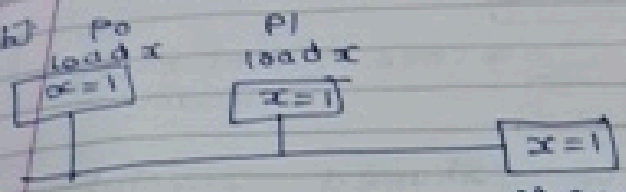
a] Here we only send invalidation message.

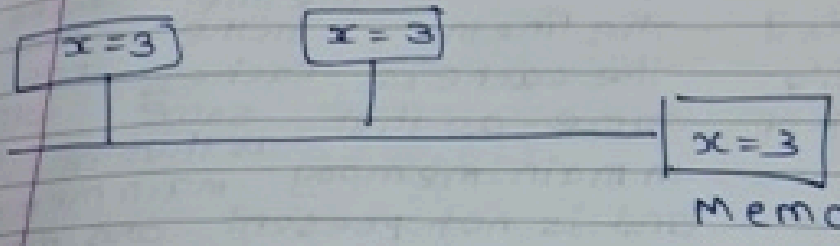


we also rectify / update the data.



Here we update also





This points are not in book

↳ Need to study this.

* Variants of snoopy BUS
  - MSI → Modify shared invalid
  - MESI → Modify exclusive shared invalid.
  - MOESI → Modify own exclusive shared invalid
  - MERSI → ~~Modify~~
  - MOSI
  - MESIF [used in intel I7].
    - ↳ Moward exclusive shared invalid foward.

a) Two processors ~~X~~ acquire access to the same line of data from data memory. Processors have a cache and use the MESI protocol. Intially both cache are empty.

Figure below depicts the consequence

Laptops
L1 d-cache 32kB 8way
L1 i-cache 32kB 8way
L2 unified cache 256kB 8 way

sololab
Monday

CO Lecture

L2 unified cache (8MB) 16 way

* MESI protocol

* core i7 intel cache diagram

→ Modified
line in this cache is modified and available only in this block of cache

Exclusive
the line in the cache is same as that in main memory and is not present in any other cache

Shared
the line in the cache is same as that in main memory and may be present in another cache

Invalid
the line in cache does not contain valid data

data in shared memory is only for reading.

↳ to modify it we need the permission of the cache controller.

↓
then it will modify and broadcast the messag

protocols → invalidation based → Mostly used i nowadays
→ update based [consumes lot of ti and resources]

Invalid → line in the cache does not contai valid data.
↓
we cannot use this for processing → as it was recently updated by some core

imp

Valid flag	Dirty Flag	Tag	State

* If we want to access modified data which is private property of the cache, then first convert it to shared state and then next process takes place.

private properties of cache

	Modified	Exclusive	Shared	Invalid
Valid cache	Yes	Yes	Yes	No
Memory copy	out of date	valid	valid	-
copies exist in other cache	No	No	Maybe	Maybe
A write to this line...	does not go to bus	does not go to bus	goes to bus and updates cache	goes directly to bus

3] Difference between consistency and coherency

↓ Deals with multiple blocks of memory

↓ Deals with single bloc of memo

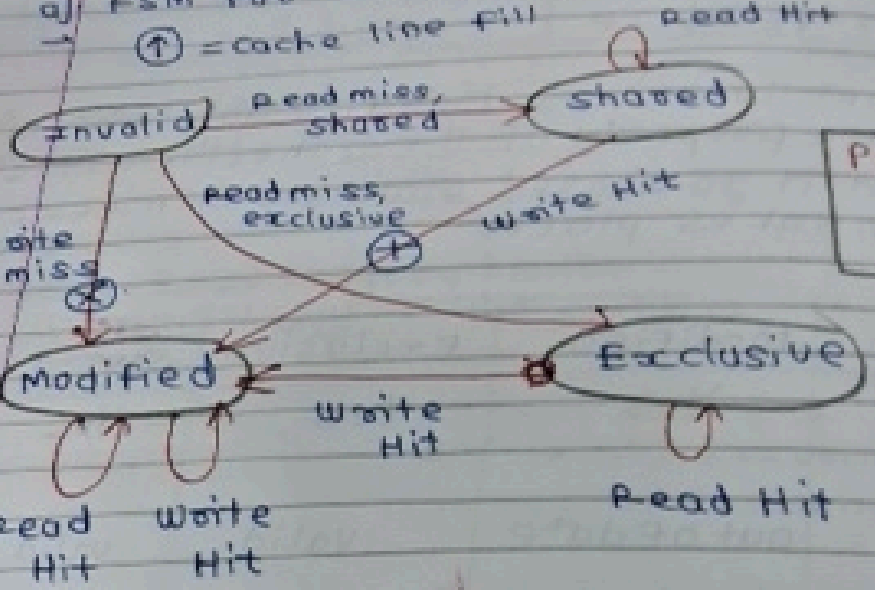
→

Ⓧ = seek permission from cache controller

⊕ Invalidate transaction

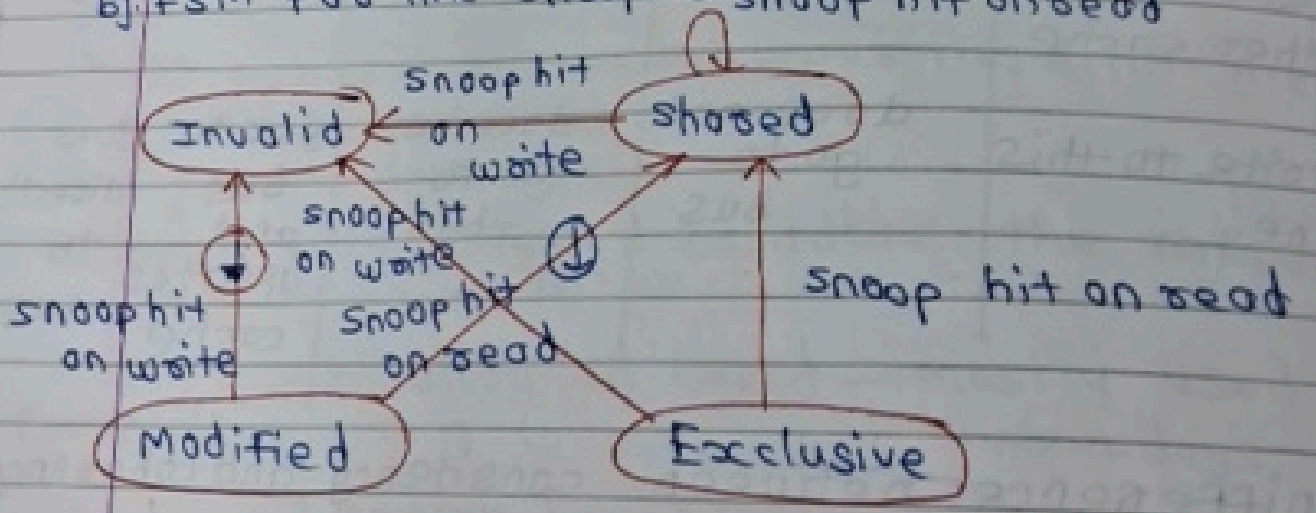
a] FSM for Line in cache at initiating processor

↑ = cache line fill



Processor generated events

b] FSM for line snooping



↓ = Dirty line copyback

↓

means before modifying the block write it to the main memory

Bus generated events

a] A snapshot of state associated with 2 caches on 2 seperate cores in a centralized shared memory system in shown below. In this system, cache coherency is maintained with an MSI snooping protocol. You can assume direct mapped cache.

P0:

	Tag	Data Word 1	Data Word 2	Data Word 3	Data Word 4	Coherency State
Block 0	1000	10	20	30	40	M
Block 1	4000	500	600	700	800	S
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Block N	300	2	4	6	8	S

b] If P0 wants to write Block 0, what happens to its coherency state?

↳ remains in modified ~~as it want to write to its own~~

b]

P1:

	Tag	DW1	DW2	DW3	DW4	Coherency state
Block 0	1000	10	10	10	10	I
Block 1	8000	500	600	700	800	S
⋮	⋮	⋮	⋮	⋮	⋮	⋮
Block N	3000	2	4	6	8	S

6) If P1 write to block1, is block 1 on P0 invalidated? Why or why not?

—

30/01/23
Monday

DBMS Lecture

* Dependancy Preservation

↓

If we take union of decomposition's FD's then we should exactly get the original FD's set.

$\therefore (F_1 \cup F_2 \cup F_3 \cup F_4)^+ = (F)^+$

$F_1 \cup F_2 \cup F_3 \cup F_4$ ↓ FD's of decomposition set

↳ But very lengthy proc

* Testing for dependancy preservation

To check if a dependancy $\alpha \rightarrow \beta$ is preserved in a decomposition of R into $R_1, R_2 \ldots R_n$ we apply the following test (with attribute closure done with respect to F).

result = α

→ Testing for BCNF

↳

→ Testing De Composition for BCNF
↳ Not able to understand.

—

30/01/23 CO Lecture

continuing question of pg no: 62 and 64

b] the blocks will not get invalidated as the tags are different.

↳ If tags were same then it would have got invalidated

c] If P1 brings in Block M for reading, and no other cache has a copy, what state is it cached in?

↳ It will be in shared state in MSI protocol. If it was in MESI protocol, then it would have been exclusive state.

10.2] Action sequence form:

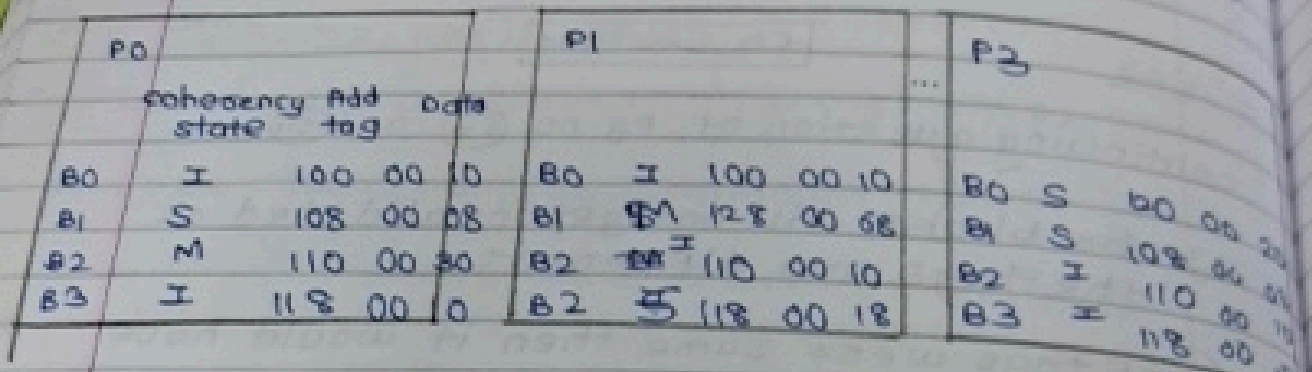
P#: <op> <add> [<value>] w/R → write operation; R → Read operation

→

Each iteration here is independent

Here <address> denotes memory address
<value> indicates the value.

One diagram: this uses MSI protocol



Here memory diagram is remaining.

a] [10] <S,2> P0: read 120 ← this address

b] P0: write 120 ← 80

c] P3: write 120 ←

→ Soln for a Here addr tag 120 is residing in P3
↳ so we don't need to bring it from main memory.
↓
In P3 120 is present in S state.
↓
so no need to change its state, as it is already available for reading.

But now when we are bringing it for reading, we must first ~~replace~~ remove some block from P0 to make space for this coming block.

1st strategy:	2nd strategy:	3rd strategy:
Replace the block with Invalid state	Replace the block with M state	Replace the block with S state.

↓ we follow this strategy

↓ If their are many then replace the first invalid Block.

So now: B0 I 100 00 10

↓ Replaced by

B0 S 120 00 20

→ sol^n for [Here we assume initial state and all^n are independ...]

④. P0: Write data ~~~~ ~~addr~~ 80 to addr 120.

↳ So tag 120 is present in P3 in shared [S] state.

↓

First get it in the P0 state and that too in modified state.

For modify state seek permission from cache controll...

But now when we are bringing it for reading, we must first replace / remove some block from P0 to make space for this coming block.

1st strategy: replace the block with invalid state

2nd strategy: replace the block with M state

3rd strategy: replace the block with S state.

↓ we follow this strategy

↓ If there are many then replace the first invalid Block.

So now: B0 I 100 00 10

↓ Replaced by

B0 S 120 00 20

Sol^n for [Here we assume initial state and addr are independent]

b]. P0: Write data ~~addr~~ 80 to addr 120.

↳ So tag 120 is present in P3 in shared [S] state.

↓

First get it in the P0 state and that too in modified state.

For modify state seek permission from cache control

Here also we remove first invalid block from P0 and then bring this new block in modified state.

↓

And now send the invalidate message for that tag

↓

So wherever tag 120 is present [except P0] mark them to invalid [I] state.

soln for c]. P3: write 80 to addr 120.

↳ Already 120 is present in P3 tag

⇓

So directly convert it to modify state and write 80 to addr of tag 120 location

⇓

No need of seeking permission from cache controllers.

d]. P1: read 110

e]. P0: write 108 ← 48

f]. P0: write 130 ← 78

g]. P3: write 130 ← 78.

→

Scenario for d] Read addr tag no from P1

↳ but addr tag no is not present in P1

↓

Now search in other caches.

↓

It is also

It is present in local cache P1 but in invalid state.

↓

So now search in other caches

Here it is present in P0 in M state

P3 in I state

✓

we will not use this as this is in I state.

↓

Now read from P0 ~~state~~ cache

↓

First convert P0 no addr tag into shared state, bring it into local cache [P1] and replace it with P1 no addr tag which is in I state.

e]. P0: write 48 to addr tag 108.

↳ It is present in local cache in S state.

↓

Seek it ~~in~~ the permission of cache controller to change it to M state.

↳ Modify it bring it and replace it with some invalid block and then broadcast

f) P0: write 130 ← 78

↳ Here 130 addr tag is not present in any the cache.

⇓

So seek it from the main memory in 's' state and side back recently ~~used~~ modified block in P0 back to memory.

⇓

then seek permission from cache controller to convert it to M state and write 78 to that location

↓

Here no need of ~~modification~~ invalid broadcasting ~~state~~ as we seeked this from the memory

g) P3: write 78 to addr tag 130 location

↳ Here 130 addr tag is not present in any of cache

⇓

So seek it from the main memory in 's' state and side back the invalid state block back to memory. first

then seek permission from cache controller to convert it to M state and write to that location

↓

Here no need of ~~modification~~ invalid broadcasting as we sooked from the memory

* Average Access Time

→ AMAT = Hit Time + Miss rate × Miss penalty —①

time required for hit

e.g. CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%.

↓

when data is not available in upper hierarchy, it then refers to PLB and then goes to next cache hierarcy.!

In this process whatever clock cycles are wasted, it is miss penalty.

AMAT = 1 + 0.05 × 20 = 2ns.

∴ 2 cycles per instruction

→

* Modified CPU time

→

$$\text{CPU time} = \left(\begin{matrix}\text{CPU execution} \\ \text{clock cycles}\end{matrix} + \begin{matrix}\text{Memory stall} \\ \text{clock cycles}\end{matrix}\right) \times$$

* Modified average access time

= Hit time of L1 + Miss rate of L1 × (Hit time for L2 + Miss rate of L2 + Miss penalty L2)

:-

30/10/23
Monday

Robotics Lecture

ESE → many numericals

↳ based on ~~tra~~ jectory

↳ [this part I have missed]

* Automation and Robotic Economics

→ (1) Basic components of robotic industrial

a] controller
b] Manipulator
c] Power supply
d] End effector

variance is known

* z-test
  - used when variance is known

9/11/2023
Wednesday

CO Lecture

* DRAM Refreshing techniques
  1. RAS only refresh
  2. Hidden refresh
  3. Distribute refresh.

1. RAS means it will refresh cells after a particular period. All cells in row get refreshed at a time. Also known as Burst refresh
2. once if we tried a particular location, then RS is made high. the reading data sight is not disturbed, but other rows are refreshed at same time.
3. RAM is not available for read or write during refresh. So first refresh, then wait then again refresh. In wait, read or write can be possible

→

* A chip present on DDR (double data rate), on one side is BANK0 and second is BANK1 on each RAM, there are 8 chips that are simultaneously working.

* DIMM (Dual inline memory Module)

* ① channel ② DIMM ③ RANK ④ CHIP ⑤ BANK ⑥ Row/ column.

imp terminology

RAM → Has 8 banks → Each bank has 2 chips → Each chip has 8 banks

Each bank has 64 rows and 8 columns

→ ① channel is socket where DDR gets inserted.

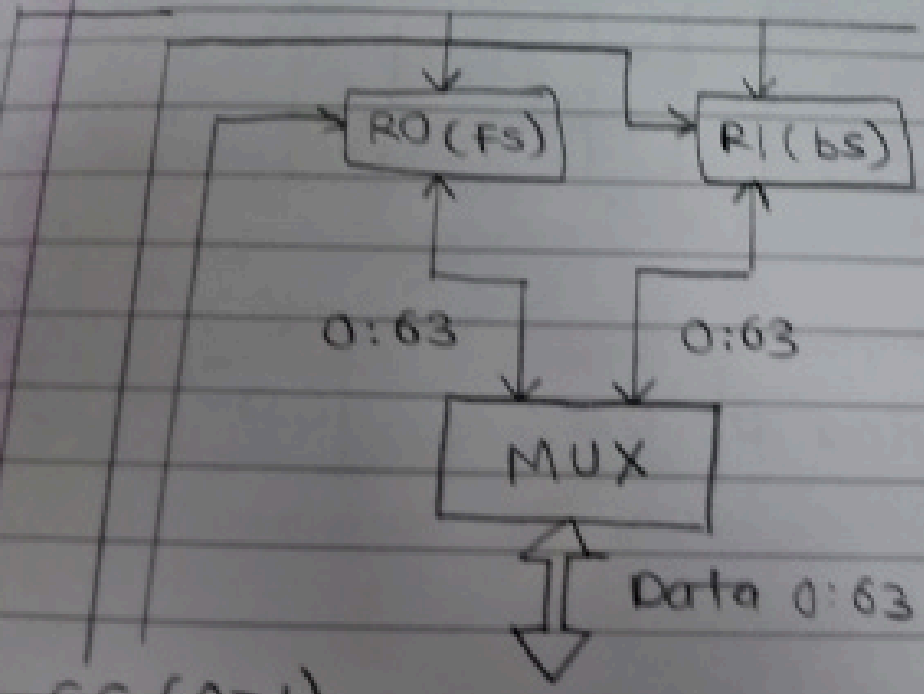
② DIMM is a module.

③ RANK 0 and RANK 1 are two sides and on each side chips are present.

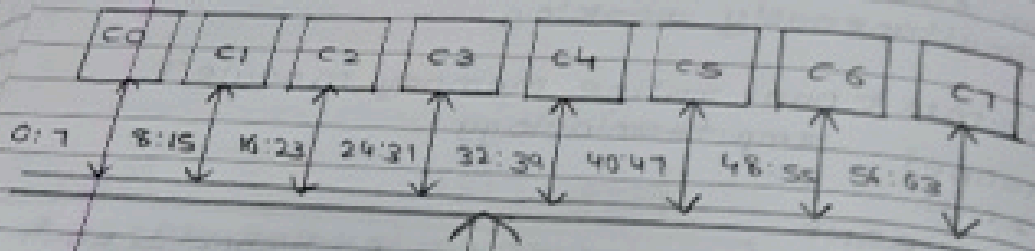
④ on each side [RANK] 8 chips are present.

⑤ Each chip consists of 8 banks.

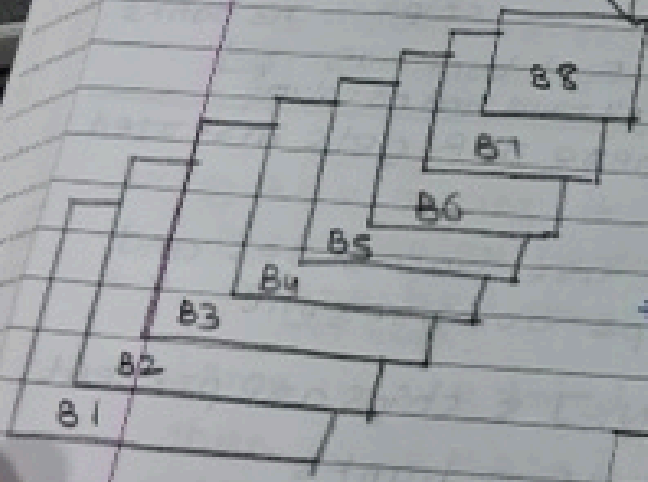
⑥ In bank there are row and column addresses.



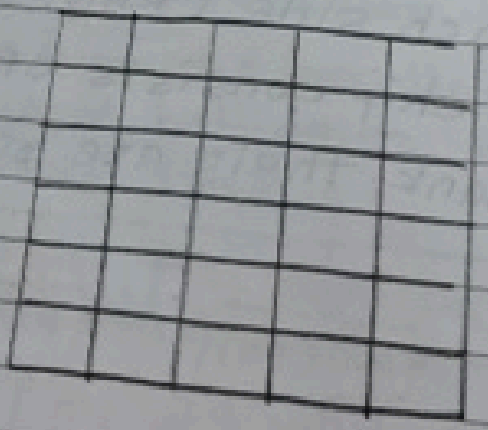
* 8 chips on a rank

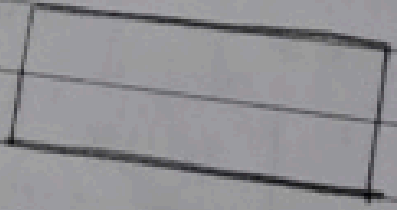


Data 0:63



* Rows and columns in DRAM.





0-7

- while getting 64-bit data from DRAM, 8 bit is taken from each chip.
- Here the activated bank no. for each chip is same.
- Also row and coln no. of activated bank in each chip is same.
- For 8 bits from each bank, 8 column of that bank are read.

e.g. Assume DDR with storage = 1GB. It consist of 1 RANK, 8 chip, 8 banks per chip. rows are 14, find no. of columns

→ 1GB = 2^{30} bytes. one location → 1 byte

So 2^{30} locations.

$8 \times 8 \times 14 \times 2^{x} = 2^{30}$

- 8 → No. of chips
- 8 → banks per chip
- 14 → 2^{14} → no. of ~~rows~~ ~~columns~~ rows in one ~~chip~~ bank
- 2^{x} → No. of columns in one bank.

$x = 30 - 14 - 6$

$\boxed{x = 10}$ ∴ So 10 columns

$r: 2^{32}$ bytes

Q.2] Storage capacity = 4GB.
2 banks, 8 chips per bank, 8 banks per chip.
column = 10, find rows?

$$2^1 \times 2^3 \times 2^3 \times 2^{10} \times 2^x = 2^{32}$$

$$\therefore x = 32 - 10 - 3 - 3 - 1$$
$$= 15$$

$$\therefore x = 15.$$

* Error correction.
(Imp)

One relation

$$2^{k-1} \geq M + k$$

This inequality gives the no. of bits...
[Incomplete]

M = data length
K = syndrome length.

1) If syndrome is all 0's => No error.
2) If syndrome contains one and only one bit set to 1, then an error is occured in one of the 4 check bits. No correction is needed.

chip.

(a) If syndrome contains more than one bit set to 1, then the numerical value of the syndrome indicates the position of the data bit in error. This data bit is inverted.

The check bits are calculated as follows:

'⊕' → denote exclusive-OR.

$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7.$

$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7.$

$C4 = D2 \oplus D3 \oplus D4 \oplus D8.$

$C8 = \oplus D5 \oplus D6 \oplus D7 \oplus D8.$

Each check bit operates on every data bit whose position number contains a 1 in the same [Incomplete].

e.g. Data = 0011 1001

D8 D1

→ $C1 = D1 \oplus D2 \oplus D4 + D5 \oplus D7$

$= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0$

$= 1$

$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$

(a) If syndrome contains more than one bit set to 1, then the numerical value of the syndrome indicates the position of the data bit in error. This data bit is inverted.

The check bits are calculated as follows:

'⊕' → denote exclusive-or.

$C1 = D1 \oplus D2 \oplus D4 \oplus D5 \oplus D7.$

$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7.$

$C4 = D2 \oplus D3 \oplus D4 \oplus D8.$

$C8 = \oplus D5 \oplus D6 \oplus D7 \oplus D8.$

Each check bit operates on every data bit whose position number contains a 1 in the same [Incomplete].

(a.i) Data = 0011 1001

D8 D1

→ $C1 = D1 \oplus D2 \oplus D4 + D5 \oplus D7$

$= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0$

$= 1$

$C2 = D1 \oplus D3 \oplus D4 \oplus D6 \oplus D7$

$= 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1.$

$c_4 = D_2 \oplus D_3 \oplus D_4 \oplus D_8$

$= 0 \oplus 0 \oplus 1 \oplus 0 = 1.$

$c_8 = D_5 \oplus D_6 \oplus D_7 \oplus 8$

$= 1 \oplus 1 \oplus 0 \oplus 0 = 0.$

→ Suppose now D_3 sustains an error and changes from 0 to 1, so recalculated check bits.

$c_4 = 0 \quad c_1 = 1$

$c_8 = 0 \quad c_2 = 0$

And now

c_8	c_4	c_2	c_1
0	0	0	1
⊕ 0	1	1	1
0	1	1	0

↳ this means D_3 is erroneous [by diagram given by sir].

SGPA of 2nd semester, SGPA of 3rd semester, CGPA of 4th semester

predict their CGPA of 4th semester
Design 3 layer neural network

—

CO Lecture

slide3
Thursday

* Hamming Parity code → Hamming bits are added at power of 2.

Assuming 4 bits

1 1 0 1
D3 D2 D1 D0

D3	D2	D1	P4	D0	P2	P1
1	1	0	P4	1	P2	P1.

→ O/P = 100 x 5.

① For P1 → start from P1 and pick one and skip one.

D3	D1	D0	P1
1	0	1	P1

② Take P1 such that no. of ones are even.

→ 1010.

similarly go for P2, P4, P8...

② then for P2
start from P2 and go like pickup 2, skip 2
also pick P2 such that it is completely has even no. of 1's

D3 D2 D0, P2
1 1 1 1

④ then for P4:
start from P4 and go like pickup 4, skip 4.

D3 D2 D1 P4
1 1 0 0

↳ keep no. of 1's even.

Ans 1 1 0 0 1 1 0

If D3 bit is getting corrupted due to some reason, so it got changed from 1 to 0.

↳ org: D3 D2 D1 P4 D0 P2 P1
1 1 0 0 1 1 0

erroneous: D3 D2 D1 P4 D0 P2 P1
0 1 0 0 1 1 0

1st check

0 0 1 0 ⇒ Failed the condition of even no. of ones.

if D3 bit is getting corrupted due to some reason, so it got changed from 1 to 0.

0101 Now on find P_4', P_2' and P_1' we get

$P_4' = 1$, $P_2' = 0$ and $P_1' = 1$

→ take XOR of old and new bits

```
  010
⊕ 101
-------
  111   => so 7th bit is an error.
```

↓ i.e D3 is an error.

↓ so flip it.

1101 → so D3 bit is corrected.

* CRC → cyclic Redundancy check

→

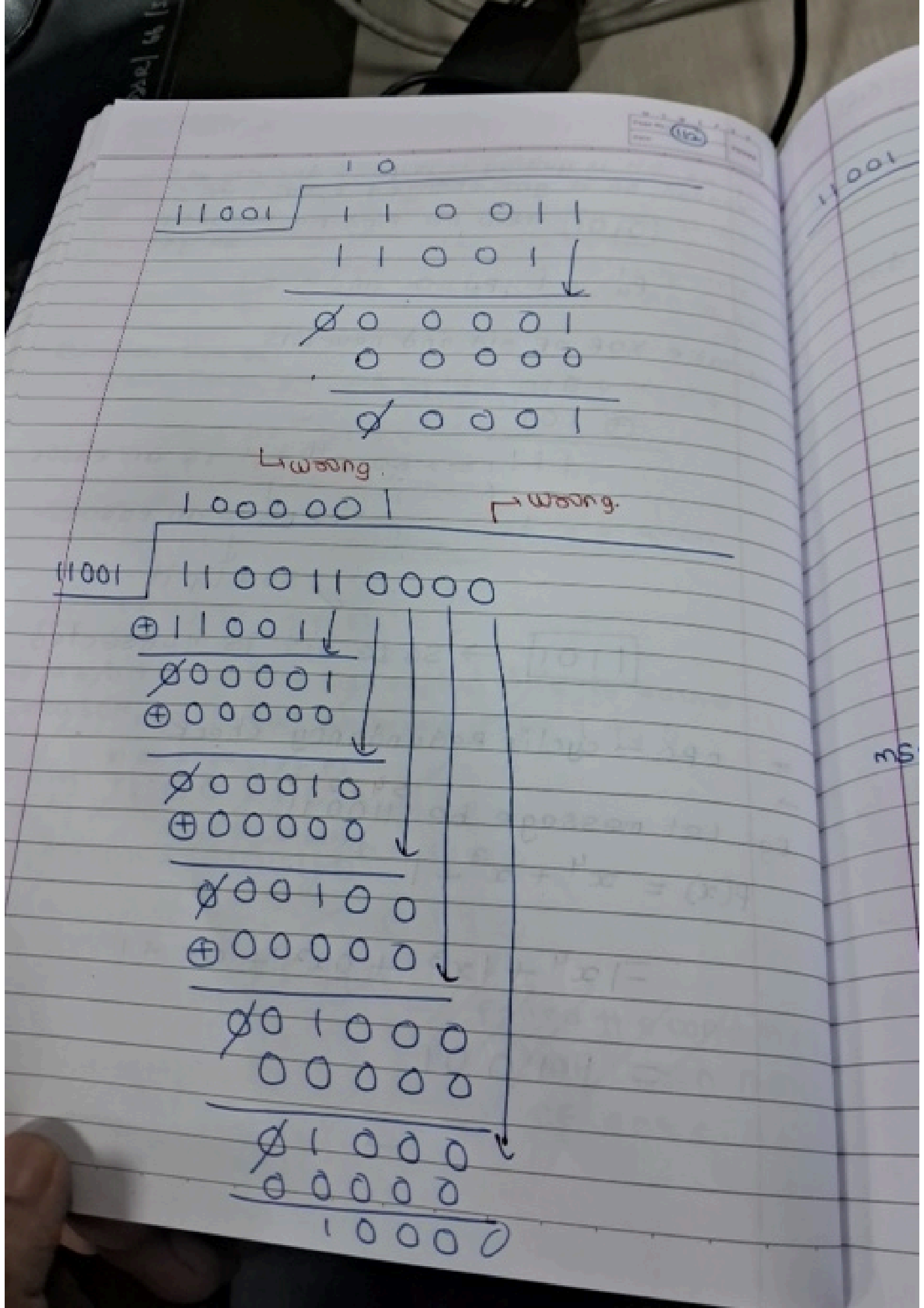
Eg: Let message be $1100\,11$ (bit positions 5 4 3 2 1 0).

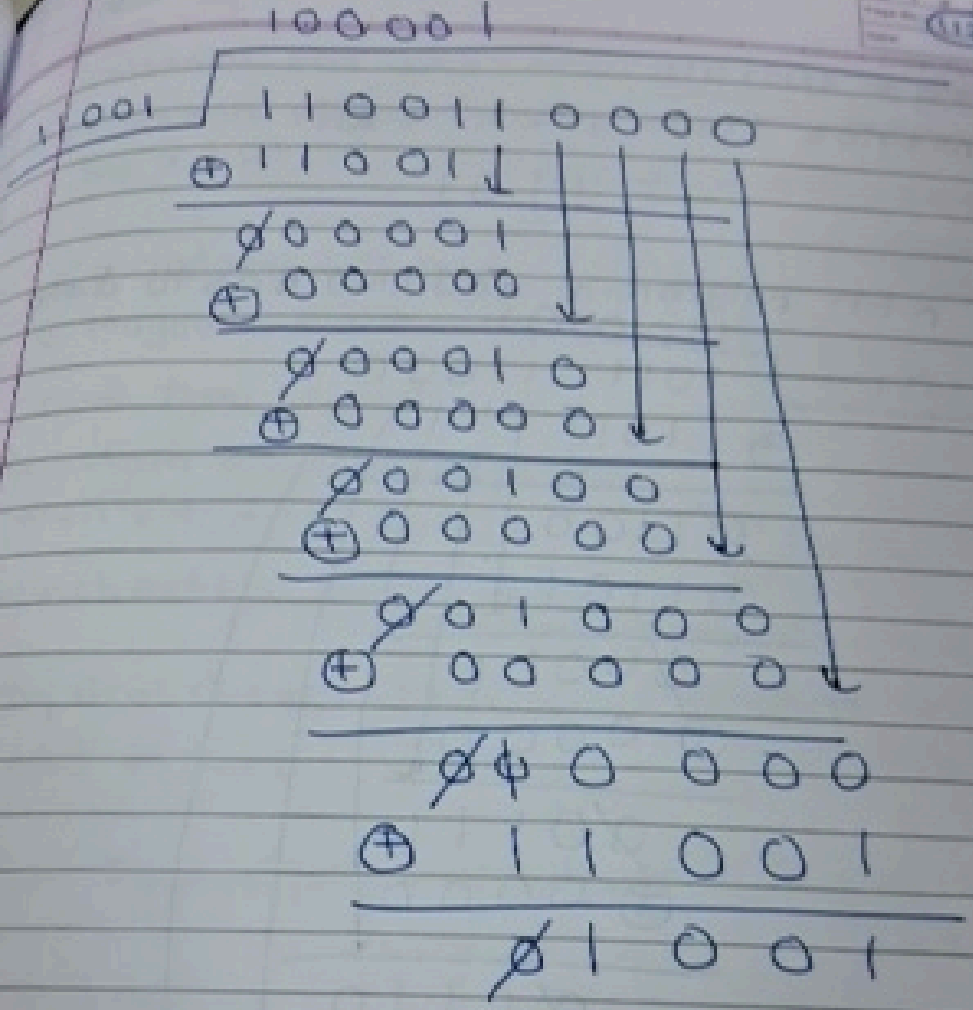
$P(x) = x^4 + x^3 + 1$

$= 1x^4 + 1x^3 + 0x^2 + 0x + 1$

$= 11001$

→


10
11001 | 11 0 0 11
11001
00 0 0 0 1
0 0 0 0 0
0 0 0 0 1
wrong
100000 1
wrong.
11001 | 11 00 11 0000
11001
00001
00000
00010
00000
00100
00000
01000
00000
01000
00000
10000



msg = 110011 | 1001

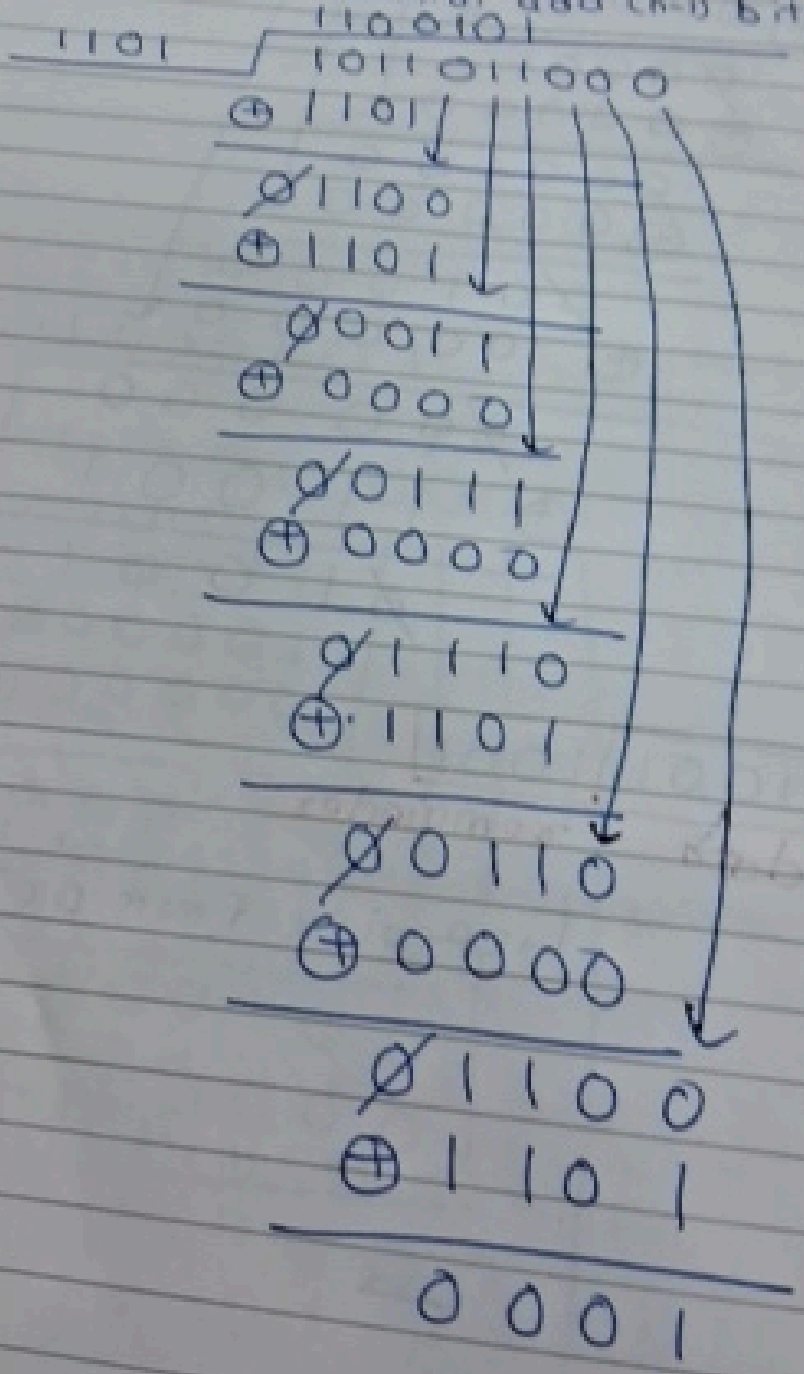
data remainder

[Revise from DC].

→

e.g. data
msg = 1011011
generator $f(x) = x^3 + x^2 + 1$
= 1101

→ For n bit polynomial add (n-1) bits to data



- Criteria for generator polynomial
- Generator polynomial should not get divided by x but not divided by $(x+1)$

- Secondary storage → Read about info
  ① Magnetic Read and write mechanisms.
- Disk Data Layout
  ① constant Angular Velocity
  ② Multiple zones.
- winchester Disk Format

- Physical characteristics of Disk systems
  Head Motion Platters
  Disk Portability Head Mechanism
  Sides
- Disk classification
- Typical Hard Disk Drive Parameters
- Timing of a Disk I/O transfer.
- Disk Performance ~~systems~~ parameters.

04/11/23
Saturday

CO Lecture

* RAID → Redundancy Assay of Independent Disc
↳ not respects to RAID 3, RAID 4, RAID 5
[width interleving parity]

a) using one parit bit for strips
↳ so first strip from all 4 disc'its parity is calculated, and parity is kept at 5th hard-disk in first ~~section~~ location.

* RAID 4 [Block level parity]
→ (1) The strip size is very large
(2) 1st parity for each block in same index in each hard-disk is calculated seperately
(3) After it we have 4 parity bits.
(4) this 4 parity bits are stored in 5th hard-disk with same index as strip or block.

* RAID 5 [Block level distributed parity]
→ (1) Like RAID4, parity strip accross all disk.
(2) Round Robin allocation for parity strip.
(3) Avoid RAID 4 battleneck at parity disk.
Commonly used in network services.
(4) ~~RAID 6~~.

* RAID 6 [Dual Redundancy].
→

2) two parity calculations.
3) store different blocks on different disk
4) user requirements of n disk requires n+2 disks.
5) High data availability.

* RAID comparision → [CODD Table 6.4].
* Comparision of SSD and Hard Disk drives.
* SSD architecture * Practical issues of SSD that are not in HHD.
* MBR and GPT disk ← Guided Partion Table
  ↓
  Master Boot Record.
  ↳ Its Boot process
  ① System intialation with firmware system BIOS [Basic I/P O/P system].
  ② BIOS looks for boot loader on MBR,
  ③ then executes it.
  ④ Boot loader reads the partitiontable

FAT, NEFTS → file system in windows

Ext4, Ext3 → Ubuntu file systems

⑤ Conventional Windows/DOS MBR searc for one primary partition.

⑥ Load the OS.

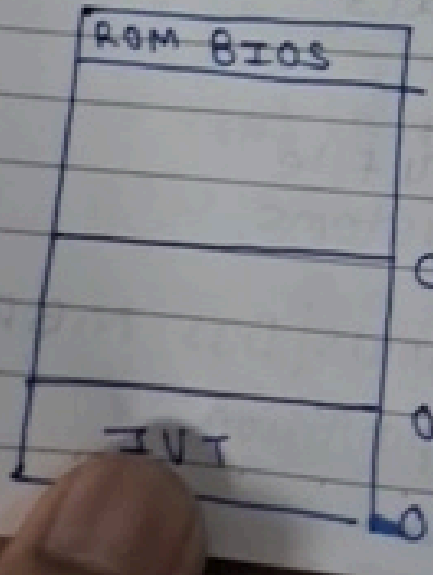
ⓐ As we power on CS = FFFFH and IP = 0000H
ⓑ Rom BIOS present at FF000H. Once we power on the reading on Rom bios starts
ⓒ while reading the Rom bios, it is made to read first sector of HDD.
ⓓ Very first sector of HDD = boot sector.
↓
loaded from
07C00H.

[CS: 0000H,
IP: 7C00H]

ⓔ In this the very first sector 512 bytes their is small boot program.
↳ Boot program finds out on HDD where is the active partition.

ⓖ Active partion means that OS is need to be loaded.

* MBR
→ ⓘ. The MBR that is located on first sector of hard-disk.



000 - 445 → 000H - 1BDH →
↳ Incomplete
MBR structure

code Area ↓ Master partition Table → Boot record signature

000 - 445	000H - 1BDH	446 bytes code area.
446 - 509	1BEH - 1F0H	64 bytes master partition table
510 - 511	1FEH - 1FFH	2 bytes boot record signature.

partion table breakdown:

446 - 461	1BE - 1CD	16 Bytes	#1
462 - 477	1CE - 1DD	16 Bytes	#2
478 - 493	1DE - 1ED	16 bytes	#3.
494 - 509	1EE - 1FD	16 bytes	#4

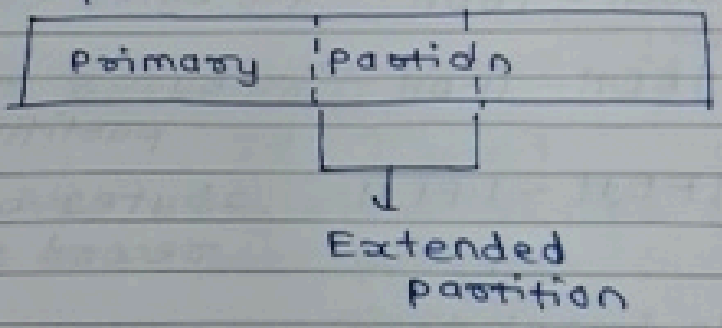
→ 0 1B Boot Indicator [80H]
1 2 3 3 Bytes starting sector [CHS]
4 1 Bytes Partion type
5-7 3 Bytes Ending sector [CHS]
8-11 4 Bytes starting sector [LBA]

12 - is 4 bytes indicate partion size (in sectors)

→ i) If it is MBR hard disk, atmost 4 primary partions possible.

ii) If more partions are needed then one of primary partion is used as extended partition

iii) the part is :



iv) In extended partition their may be linked list structure of extended partitions.

↳ If anyone of all link is destroyed, then we may lose all extended partions.

* LBA → 32 bytes

∴ 2^{32} max sectors can be pointed by LBA.

↳ Each sector of 2^9 bytes.

↳ ∴ $2^{32} \times 2^9 = 2^{41}$ Hence = $2^{40} \times 2^1$

Hence the entire capacity 2^{TB} = [1 TB = 2^{40}

FAT = file allocation table.

In MBR disk more than 2TB, the MBR cannot access if it is going more than 2TB.

Partion types → 0B 32bit FAT
- 0-5 type [extended partion with CHS]
- 0-F type [extended partion with LBA]
- 0x07 type [for windows partion]

0C 32 bit FAT using INT 13. as extension.

More tyes:
1. 0E using 16 bit FAT using INT13 extension.
2. 82 linux swap partion
3. 83 linux native file system.
4. EE GPT protective MBR
5. EFI/UEFI system partion.

* GPT [GUID partion table].
1. Intially intel came with IDR, that will remove limitation of 2TB disc.
2. The propersable model is Extensible firmware interfaces [EFI].
3. EFI IDR is accepted by all industries.

↳ Now it is UEFI
[Unified extensible Firmware...].

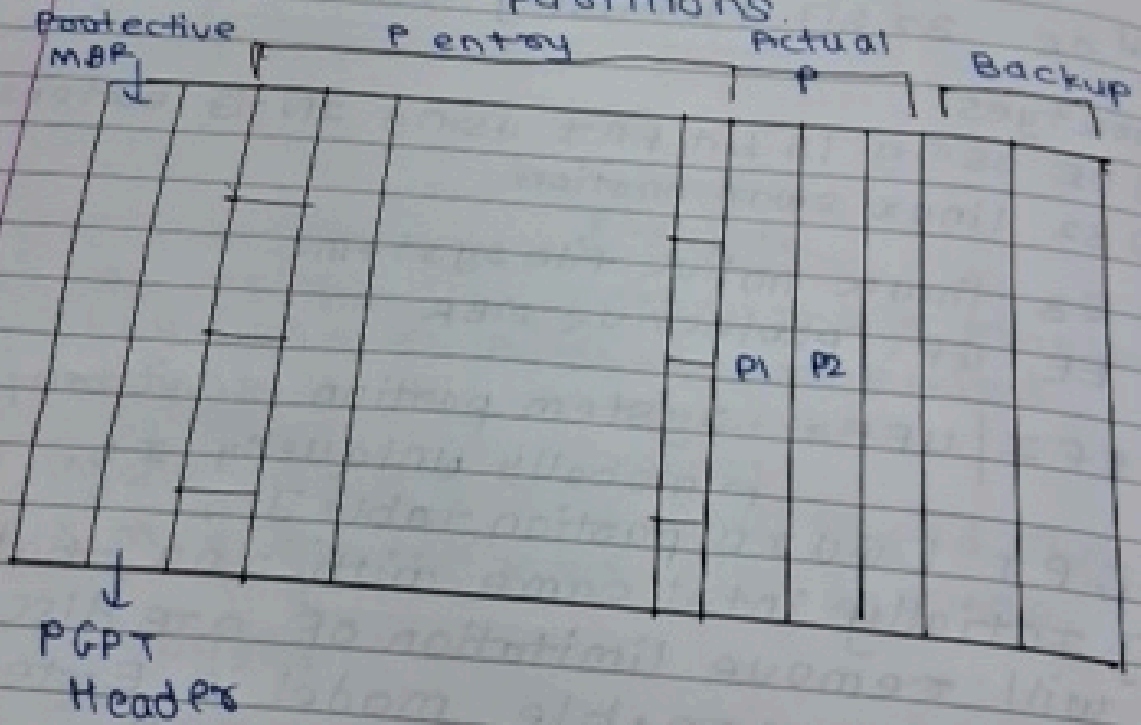
GPT → default partition style for X86 based systems.

↓

uses EFI to boot drives.

Protective MBR	GPT Header	Partition Table	Partion Area	Back up Area

- uses primary and ~~partion~~ back up partition table to improve integrity
- Support upto 128 partitions.



→ 1. First sector is protective MBR.
2. Next sector is GPT header.
3. Next few sectors are LBA's.
4. Next starts actual partion area.
5. At end, few are backup area.

MBR GPT

backup of primary GPT header present in secondary GPT header.

1) LBA 2 - 33 in upper part is copied to LBA 2' - 33' in lower part near secondary GPT for back up.

2) at max 28 partitions and each sector store information about 4 partions i.e LBA 2 - 33

$\frac{128}{4}$ = 32 sectors are required.

3) $\frac{512}{4}$ = 128 bytes [storage for each partions info].

5) After LBA - 33, actual partition starts.

* virtual Memory organisation

→ process ↔ Cache

Cache ↔ main memory

Cache → contains Blocks

One block contain multiple words

One page contain multiple block

One foame contain multiple pages

~~cache~~ main memory ↔ secondary memory.

Main memory to secondary memory we don't have the logic of write through.

↓

As access time is much higher.

MBR GPT

backup of primary GPT header present in secondary GPT header.

② LBA 2 – 33 in upper part is copied to LBA 2' – 33' in lower part near secondary GPT for backup.

③ It at max 28 partitions and each sector store information about 4 partions i.e LBA 2 – 33

$\frac{128}{4}$ = 32 sectors are required.

④ $\frac{512}{4}$ = 128 bytes [storage for each partions info].

⑤ After LBA – 33, actual partition starts.

* Virtual Memory organisation

→ process ↔ Cache

Cache ↔ main memory

Cache contains Blocks

One block contain multiple words

One page contain multiple block

one frame contain multiple page

~~Cache~~ main memory ↔ secondary memory.

In main memory to secondary memory we don't have the logic of write through.

↓

As access time is much higher.

* Virtual Memory address translation
  ↳ Page Fault, virtual address, physical address
① The unit that decides/moves from secondary memory to Main memory or vice versa is called the page.

* TLB - inner core i7 TLB
  [web.cs.wpi.edu]

* TLB - Translational lookaside buffer.
  ↳ special type of n-way associative cache

MMU → Memory Management Unit
  ↳ mapping of logical address to

—

6/11/23
Monday

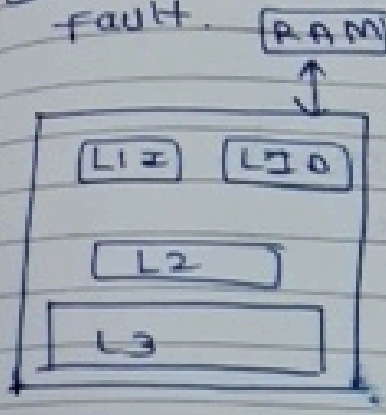
CO Lecture

* Virtual Memory
→ Use main memory as a "cache" for secondary (disk) storage.
  ↳ Managed by CPU hardware and OS.

programs share main memory.
↳ each gets a private virtual address space holding its frequently used code and data

CPU and OS translates virtual addresses to physical addresses.

① VM block is called a page.
② VM translation "miss" is called a page fault.

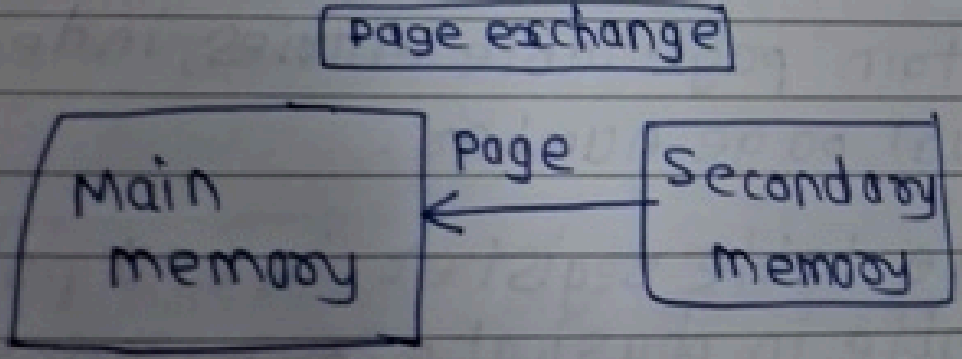


TLB → PTE → Page table entry

page = Fixed sized unit which moves from secondary to primary memory.

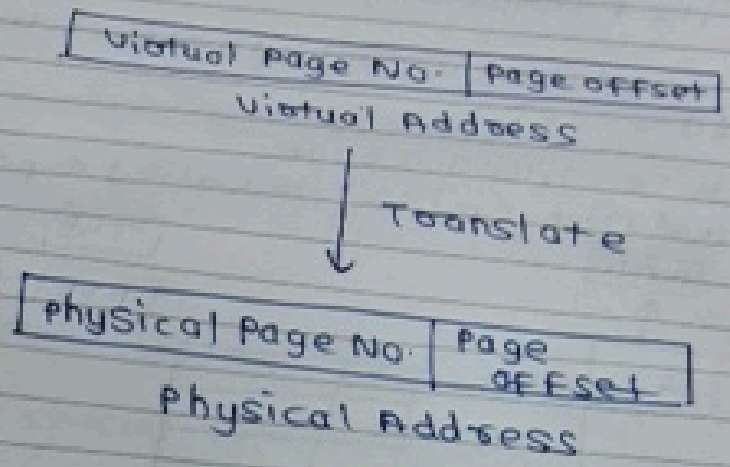
RAM consists of some ~~blocks~~ whose entries are in TLB.
page

TLB → consist info about blocks which resides in the main memory.



for this scenario, main memory act as the cache.

* Address Translation
  - Fixed - size pages [Eg: 4K]



* Page Fault
  - On page fault, the page must be fetched from disk.
    - Takes millions of clock cycles.
    - Handled by OS.

* Page Table
  - Resides in the main memory
  - Stores following info:
    1. Contain page table entries, indexed by virtual page number.
    2. Page table register in CPU points to page table in physical memory [RAM]

page is present in main memory:
① PTE stores physical page no.
② Also stores other status bits like dirty, valid, etc.

* Translating using a Page Table

* Mapping pages to storage. * Replacement and writes.

① LRU replacement policy

Here we have various bits
a]. Reference bit in PTE set to 1 on access to page.
b]. Periodically cleared to 0 by OS.
c].

[Incomplete].

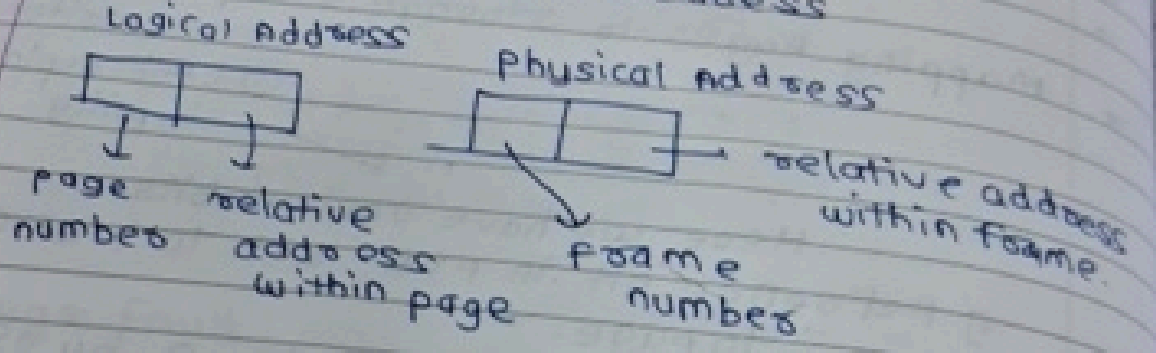
* Fast Translation using SLB

Address translation would appear to require extra memory references.
a]. one to access the PTE.
b]. Then the actual memory access.

But access to page tables has good locality.
a]. So use a fast cache of PTE's within the ~~buffer~~ CPU.
b]. Called a TLB [Translational look-aside Buffer]

Go Back N
selective reply

* TLB Misses
→
* TLB and Cache Interaction
* use of swapping * Dynamic Partitioning
* Allocation of Free Frames
* Logical and Physical Address
→



* Virtual Memory and Demand Paging
→ X86-based processor makes use of this.

① Each page of a process is brought in only when it is needed.
principle of locality.
↳ Find first in local storage, if present use it else trigger the page fault if it is not present in local storage.

ncept of thrashing

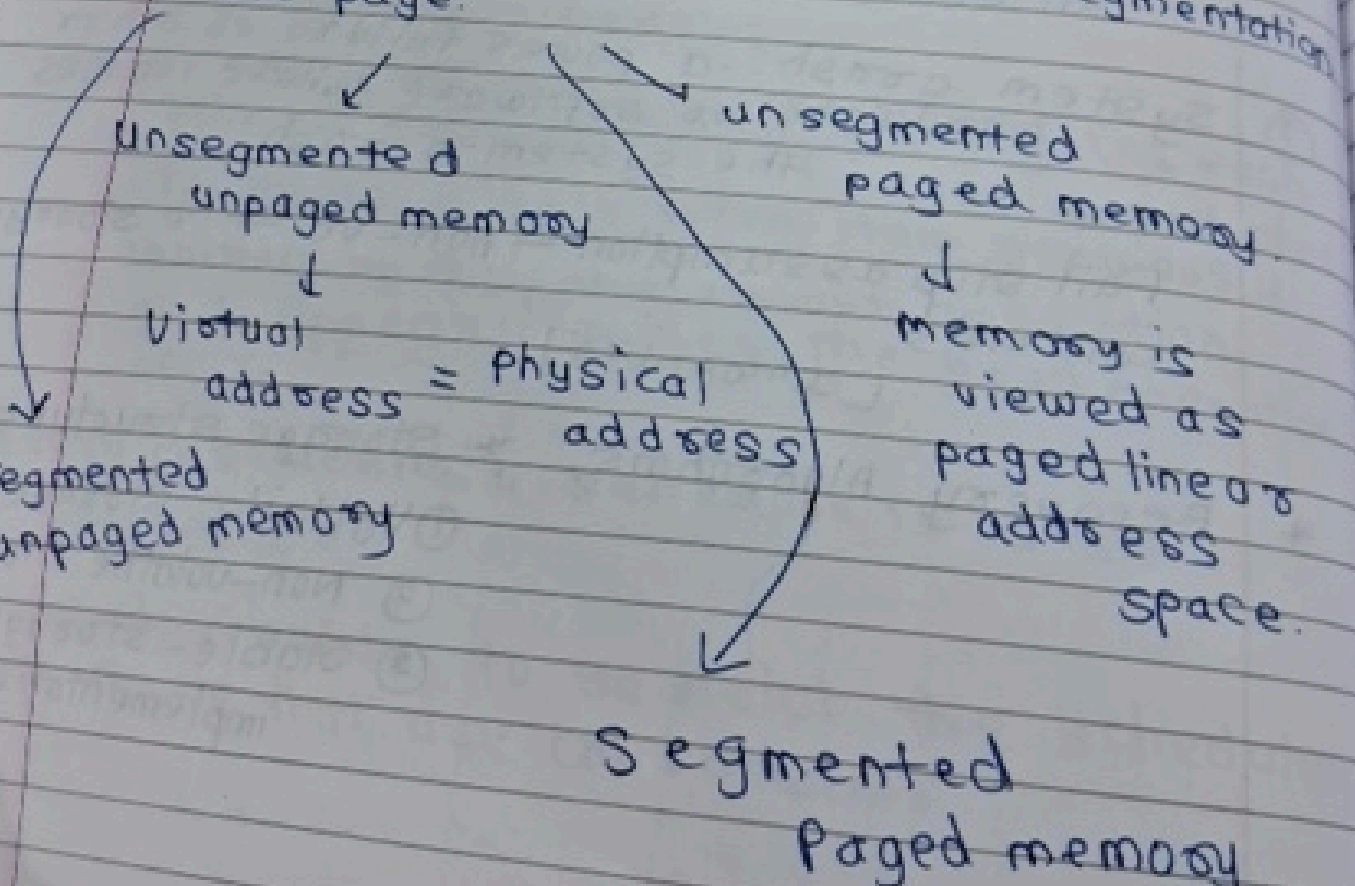
1KB = 2^{10} Bytes
1MB = 2^{20} Bytes
1GB = 2^{30} Bytes
1TB = 2^{40} Bytes

11/12/23
Monday

CO Lecture

- TLB and cache operations
- 1) multiple programms can be executed simultaneously
  2) Any program/software that requires more memory than available in main memory can also be serviced.

- Segmentation
  1) visible to programmers.
  2) Here programmer can think of various segments in address space.
  3) Includes hardware for both segmentation and page.



i) Each virtual address
   - 16 bit segment reference
   - 32 bit offset

ii) Unsegmented virtual memory = 2^{32} = 4GB
    segmented virtual memory = 2^{46} = 64TB

iii) physical address space employs a 32 bit address for a max of 4GB.

iv) virtual address space is divided into two parts:
   a) one-half is global, shared by all processes.
   b) the remainder is local and is distinct for each process.

* segment Protection
- Associated with each segment are two forms
  - Privilege level
  - Access attribute

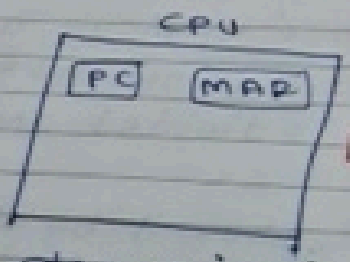
* Intel x86 Memory Management Formats.

* Address Translation Mechanisms.

—— X Memory Management Ends X ——

—— x Pipelining starts x ——

* Instruction cycle state Diagram
  ↳
* Data flow, fetch cycle
  ↳



[Incomplete Diagram]

① The stages in instruction processing include fetch the Instruction [IF], decode the instruction [IO], calculate operand address, fetch the operand, execute instruction [IE], write result [WR]

Q.1] Total students = 170.
Total answerbooks = 500 → Each answer sheet has 5 question.
Avg 5 min/ans book question
↳ ~~500 x 5~~ = ~~2500 mins~~
For one question → 5 min
For ans book → 25 min one.
For ~~one~~ all papers → 25 x 500
= 12500 mins.

This was case when only individual examines is checking the papers.

	Ex1	Ex2	Ex3	Ex4	Ex5
t1	A1-1				
t2	A2-1	A1-2	A1-3		
t3	A3-1	A2-2	A2-3	A1-4	

- If 5 examinees are working in pipelined way i.e 1st examiner checks only 1st que and give to next examiner.

① Firstly at starting, after 25 min A1 will be completely asseged.

② Then after every 5 min, next Answer book is completely assesed.

Total time = 25 + 5 * 499 = 2520

↓

First question will take 25 mins

Pipeline → allows to improve overall throughput.

Formula = $T + (n-1)\left(\frac{I}{K}\right)$ —①

For time using pipeline

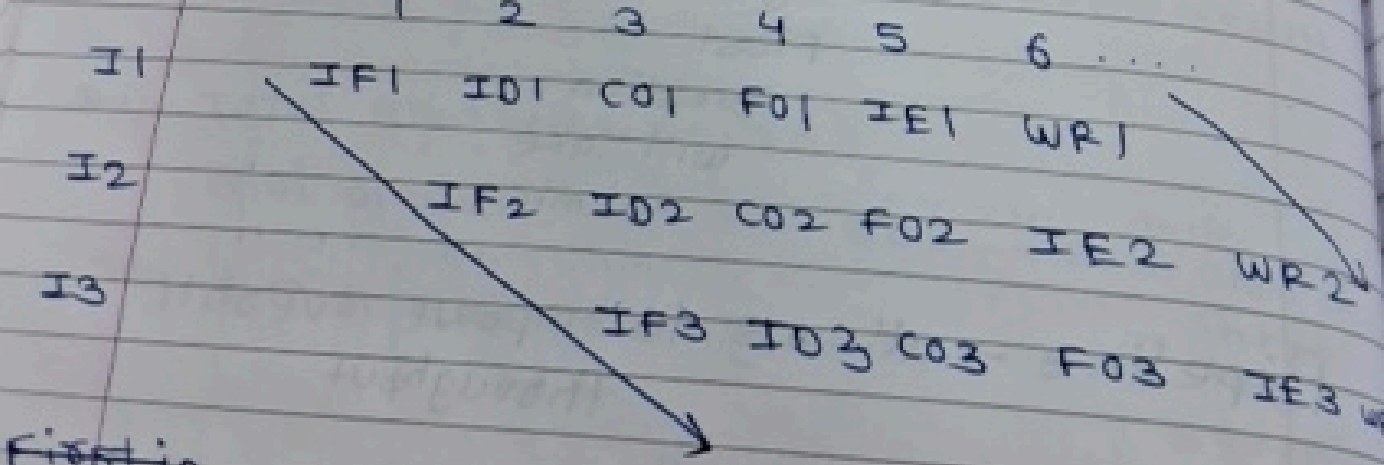
Time required = nT — ②
without using pipeline

* Speed up = $\frac{T_{old}}{T_{new}} = \frac{nT}{T + (n-1)\left(\frac{T}{k}\right)}$

$= \frac{n}{1 + \frac{n-1}{k}}$

$= \frac{nk}{n+k-1}$

* Pipelining for the instructions



~~First~~ in completion of 1st instruction in 6 clock cycles. After that one clock cycle each another instruction will be completed.

for executing 5 instructions → need 10 clock cycles

↳ But all this is in ideal scenario

But there are many dependancies:

a] Flow dependance

↳ A statement S2 is flow-dependant on statement S1 if an execution path exists from S1 to S2 and if at least one o/p of S1 is feed as an i/p to S2.

b]. Antidependance

↳ statement S2 is antidependant on statement S1 if S2 follows S1 in program order and if output of S2 overlaps the input to S1.

c]. o/p dependance

d]. I/O dependance

e]. Unknown dependance

Two statements are o/p dependance if they produce (write) the same o/p variable.

Read and write are I/O statements. I/O dependance occurs not because the same variable is involved but because the same file is referenced by both I/O statements.

RRR → read after read
this is not hazard

Hazards:

① RAW → read after write
② WAR → write after read
③ WAW → write after write

* Data dependance in programs:

S1: Load R1, A
S2: Add R2, R1
S3: Move R1, R3
S4: Store B, R1

① S1 and S2 flow dependant.
② S2 and S3 are o/p dependant.
③ S2 and S3 are anti-dependant

—

clu123
Monday

Robo Lecture

* Robot languages and programming
↳ WAVE and AL └ Their are multiple generations here

Question can come on this in ESE

Arm language

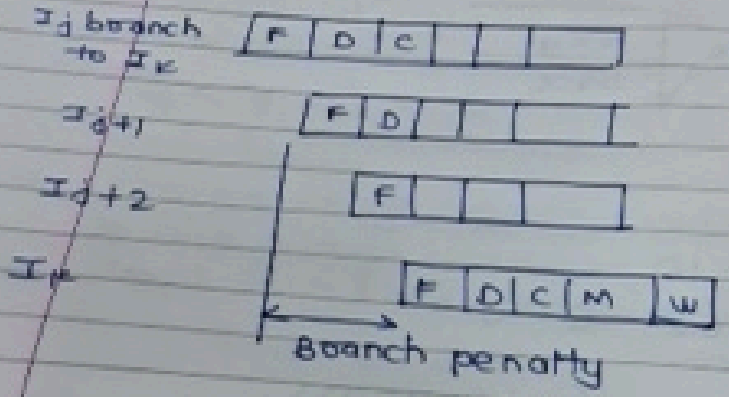
① VAL
↓
① textual robot language
② user-Friendly
③ provides arm movement in joint, world and tool rotates, gripping and speed control

② AML ③ MCL ④ RAIL ⑤ AUTOPASS

8/11/23
Wednesday

CO Lecture 6

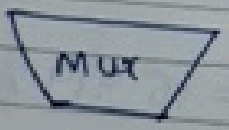
* Branch predictions * unconditional branches



* Decode stage of the pipeline

* Instruction address generator:

[Incomplet diagram]



* Branch delay slot

↳

Add R7, R8, R9
Branch if [R3 == 0] Target
I_{j+1}
:
:
Target: I_k

→ means Finding Target address

If we calculate this in encode stage then branch penalty is more than that as compared to calculate in decode stage

a] original sequence of instructions containing a conditional branch instruction.

```
Branch if R3==0  Target
Add  R7, R8, R9
I j+1
.
.
Target: I k
```

b] Placing the add instruction in the branch delay slot whose it is always executed.

↳ Delayed branching technique

* Branch delay slot

↳ Here place a instruction which is independent of whether branch is taken or not.

↓

It cannot be I_{j+1}.

↳ becoz it depends on branch condition satisfaction.

And the execution of this instruction is compulsory

↳ while doing this ensure that data dependancies are retained (preserved)

⇓

If no such independent instruction exist, then we insert 'NOP' in it.

↳ [No operation]

* control hazard / Branch hazard.
- 1] occurs when pipeline makes wrong decision on branch prediction.
2] Dealing with branches:
a] Multiple streams
b] Prefetch branch target
↳ Prefetch both data and instructions.
* c] Branch prediction
* d] Delayed branch

Core → Latency based [CPU]
↘ throughput based [GPU] → CUDA core
↘ tensor core

* Multiple streams
↳
* Branch prediction
* Loop Buffer.

* Branch prediction

Approach-1: Predict never taken

Approach-2: Predict always taken

Approach-3: Predict by Opcode

↓

These are static approaches

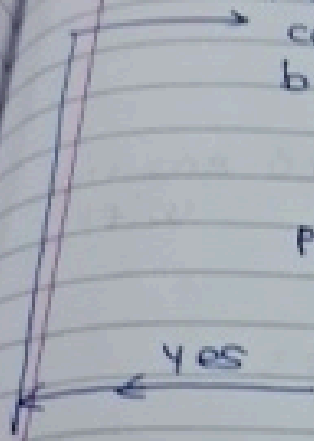
↓

Do not depend on execut history.

Approach 4: Taken/not taken Switch
Approach 5: Branch history table

These approaches are dynamic and depend on execution history

Branch prediction flow chart



[Incomplete diagram]

* Stall on Branch.
  - can come in ESE.
* Risc-V pipelined Datapath.

Unit of Pipelining Ends

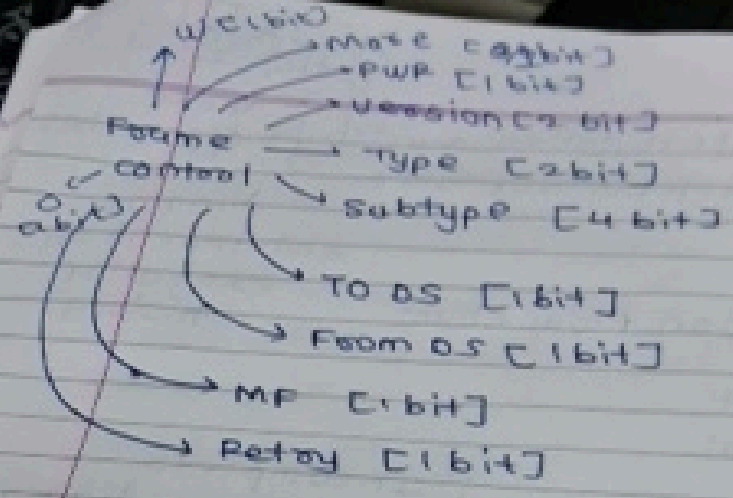
CN Lecture

IEEE 802.11 WiFi Frame Format

Frame	Duration	Address 1	Address 2	Address 3

Sequence	Address 4	Data	checksum

tinye



Q.] difference between 802.11 and 802.11 format.

Ethernet wifi

* ↳ ARP protocol in wired access
① Switch don't have MAC id.
② Routes has MAC id.

ARP protocol → so lets suppose we want to send data to a host whose IP we know, but we don't know MAC of him. So for finding his MAC we need ARP protocol.

↳ So for this it sends

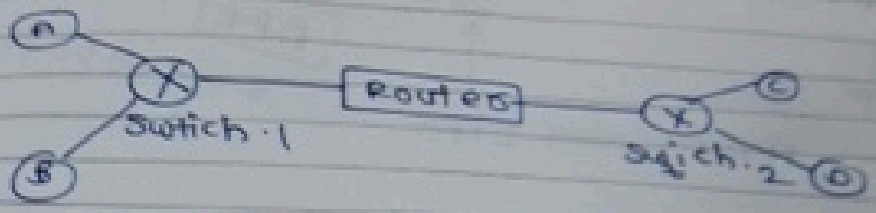
ere host
anbe any
in subnet

* DNS resolver v/s ARP → Here we want host in same subnet

Domain Name ⟷ IP mapping

Mac Address ⟷ IP address mapping

ARP querying is done by switch and ARP request is sent by host to Switch.



If A wants to send a packet to D but it doesn't know the D's MAC.

So:

① A sends doing ARP request to switch.1
② Switch.1 then does ARP, and sends discovering request to ALL.
③ Router also recieves discovering request so it also now does ARP.
④ Now switch 2 also recieve discovering request from Router, so it again does the ARP.
⑤ And when switch 2 get is it do backtrack.

Link Layer Switches - Filtering and forwarding

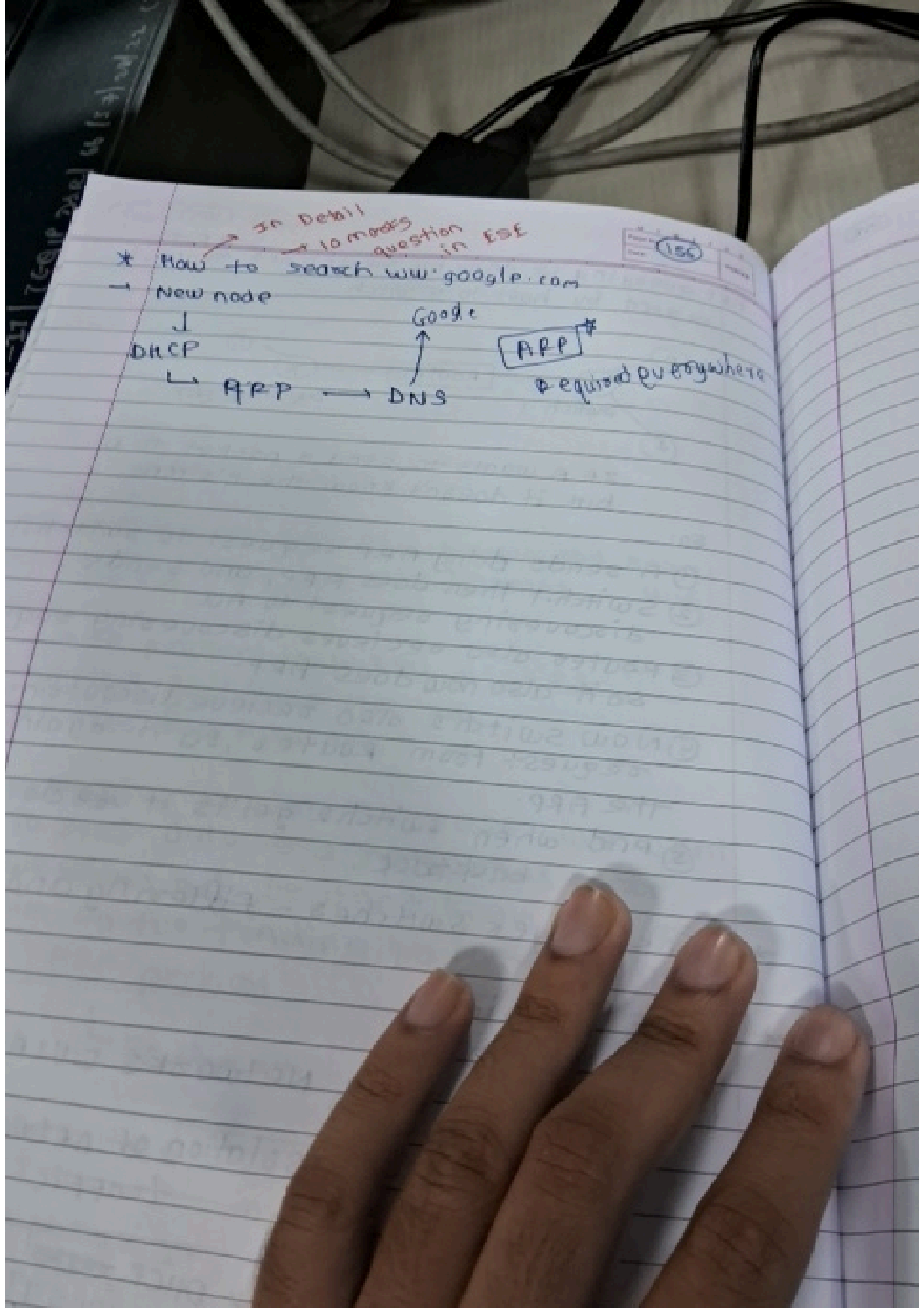
self learning

Virtual local area Networks (VLAN's).

↓ Concept is related to the proxy server

↳ Isolation of networks traffic

* DHCP uses UDP.


In Detail
10 marks question in ESE
* How to search www.google.com
→ New node
DHCP
APP → DNS
Google
APP
Required everywhere