DATE / / OBJECT

→ Types of Memory :-

RAM
Random Access Memory
temporary.

ROM
Readonly Memory
Parment

two types of RAM DRAM (Dynamic RAM), SRAM (Static RAM).

DRAM

→ slow
→ Consits of Capacitor
→ need to refresh every milisecend to prevent data loss
cheap.

SRAM.

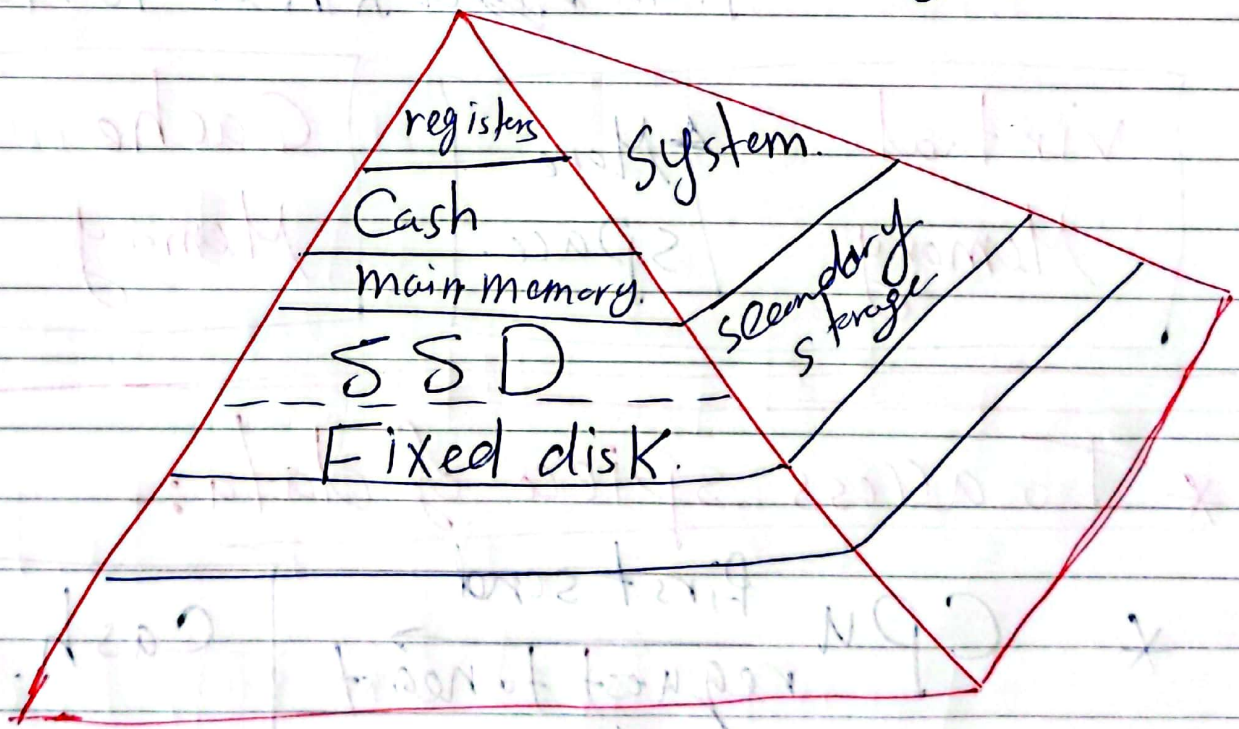
→ fast
→ smaller than DRAM
→ not need to refresh.
expensive.

* The Memory hierarchy :-

↳ To provide best performance at lowest Cost, Memory is organized by (hierarchial fashion.)

→ Small (fast storage).

→ large (slower storage).



DATE / / OBJECT

Memory, Registers Cash, Virtual Memory.

Registers → storage location available on the processor.

Virtual Memory used by hard drive.
it extends address space.
from ~~Hard~~ RAM to hard.

virtual Memory.	↑ More space.

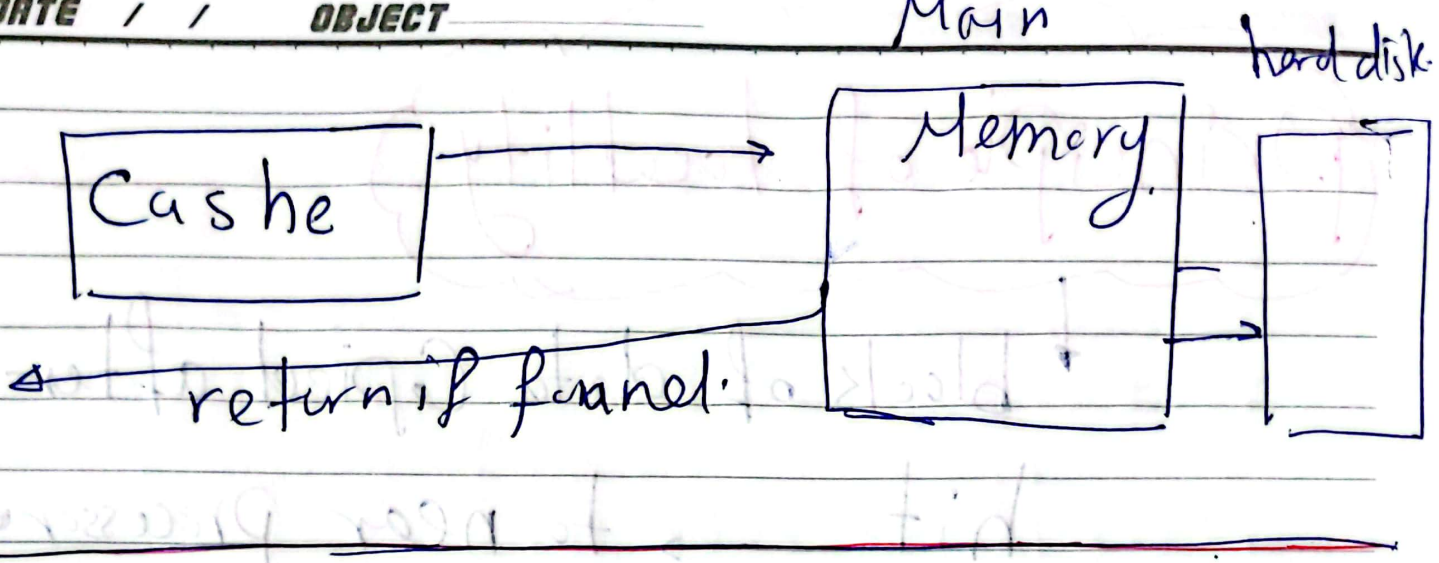
Cache Memory.	↑ More speed

* To access spiece of data:-

* CPU → (first send request to nearst Memory.) → Cash.

← send answer. If found or not.

DATE / / OBJECT



hit : when data is found at given Memory level.

miss : when data not found.

hit Rate : percentage of time data is found in Memory level.

Miss Rate : the percentage of time not found.

$$\text{Miss Rate} = 1 - \text{hit Rate}$$

Miss Penalty : → time required to process Miss, including time to replace block of Memory.

~~+ time to deliver to processor~~

DATE: / / OBJECT

Principle of locality

↓ block of data copied after hit → to near processor

→ nearby data element will be needed.

→ three forms of locality.

→ Temporal locality: Recently accessed data elements tend to access again. (locality to same things are clustered in time).

→ Spatial locality:

Access tend to cluster.

→ things that are refrenced close in time are close in space.

~~(nearby Memory address, nearby sectors on disk~~

→ Sequential locatly:-

Instructions tend to accessed sequentially.

Spatal locatly

A[1], A[2]
A[3],

temporal locatly

A[1] . . .
A[20000]
, A[2] ,
A[400000].

DATE / / OBJECT

Cashe Memory:

- speed up access by storing recently used data closer to Cpu instead of main memory

Main Memory accessed by address
Cashe accessed by Content

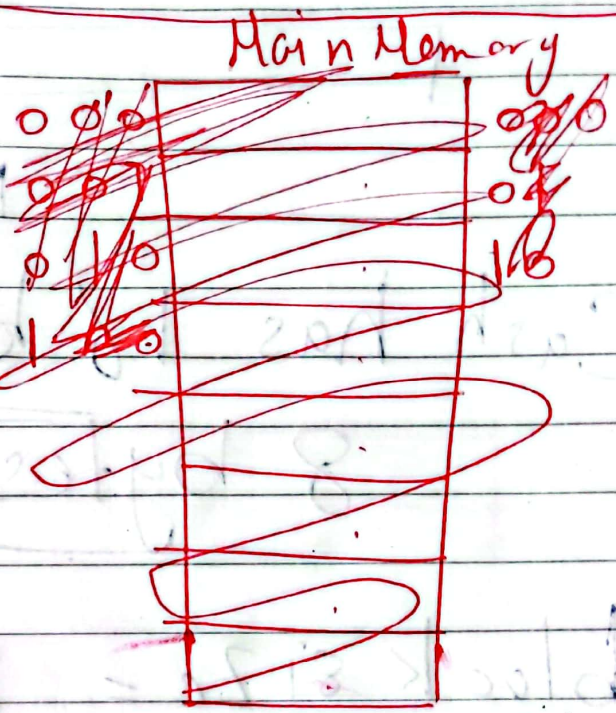
* indirect mapped Cashe Consist of N blocks of Cash, block X of main memory map to

$$Y = X \bmod N$$

7, 17, 27, 37.

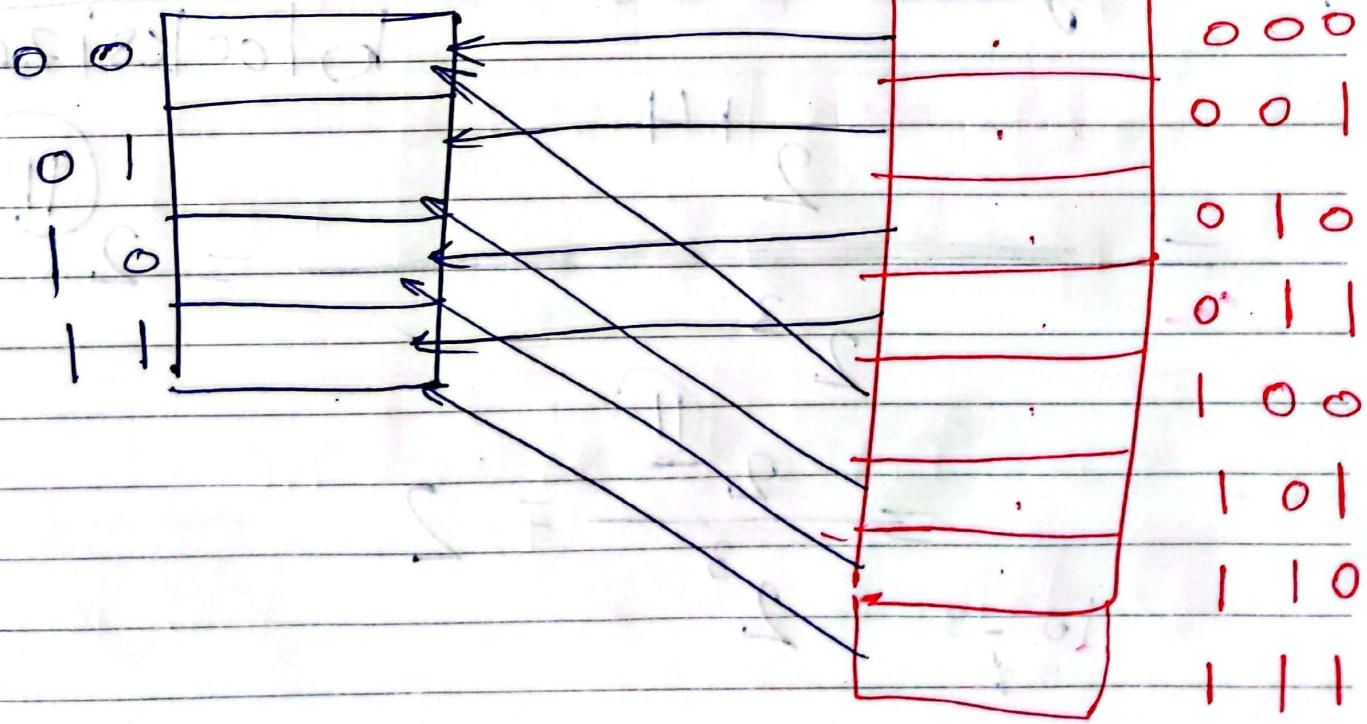
DATE / / OBJECT

→ If we want to map block 7 and we have 4 blocks of Cache.



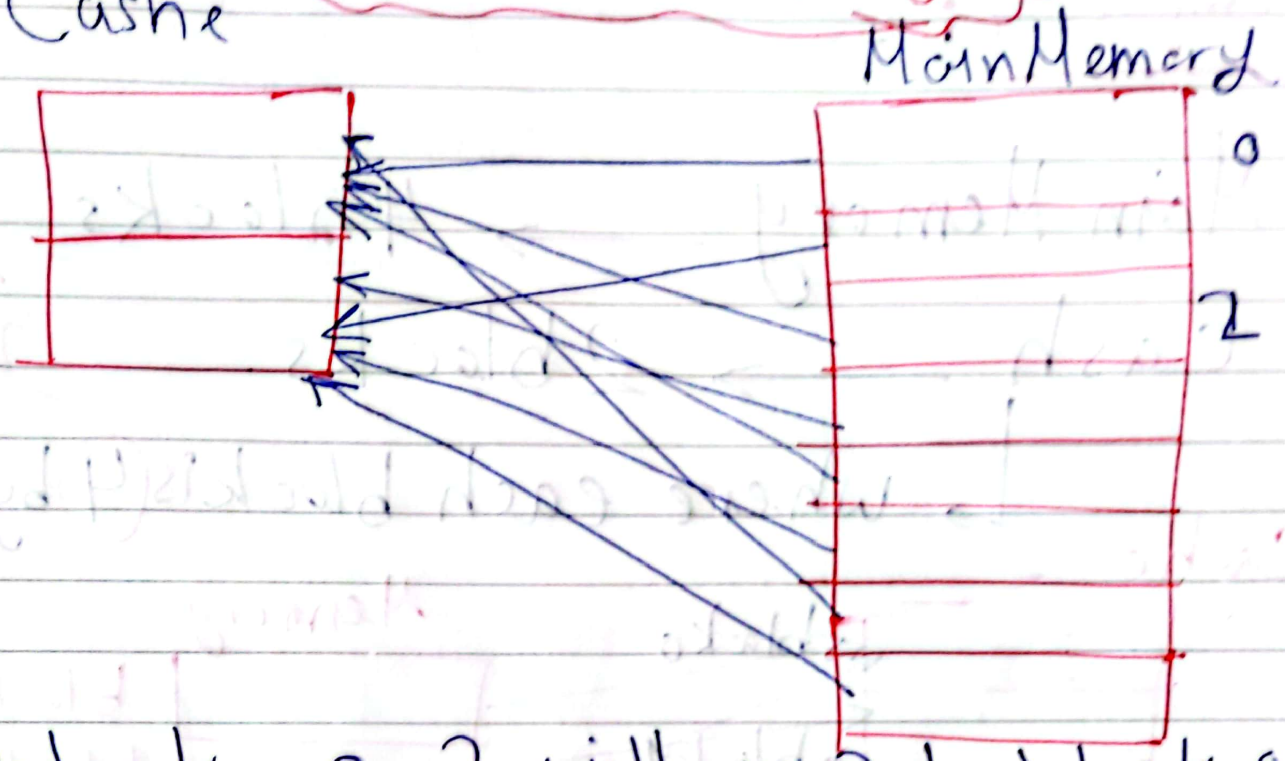
Cache.

Main Memory



DATE / / OBJECT

direct Mapping



block 0, 2 will map to block 0 in ~~cat~~ cashe. ⟶ the tag will differe between them.

→ the binary Main Memory address is partitioned into.

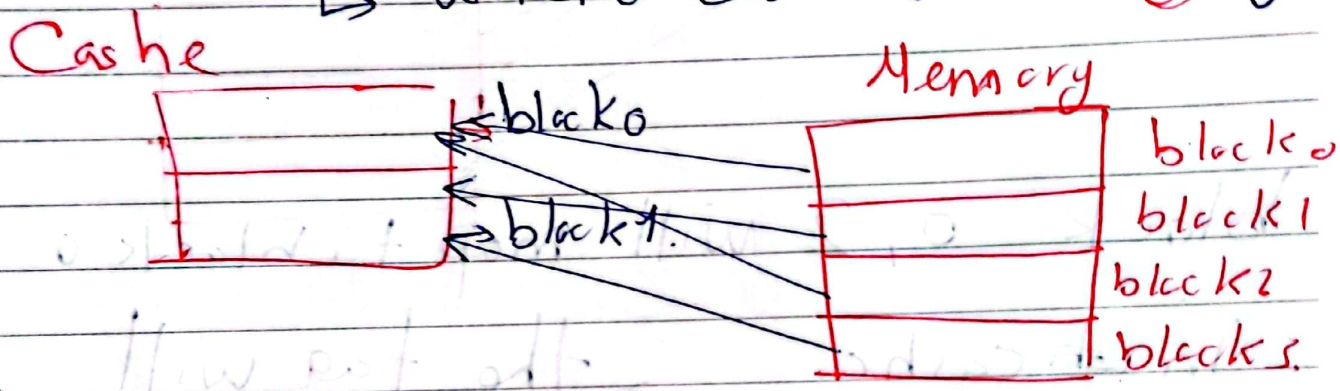
Tag	block	offset
	select unique block of Cash.	identify address with specific blocks

Example 1:-

Main Memory ⟶ 4 blocks

Cash ⟶ 2 blocks.

↳ where each block is (4) bytes. 2^2



(1) * each block of Cashe is

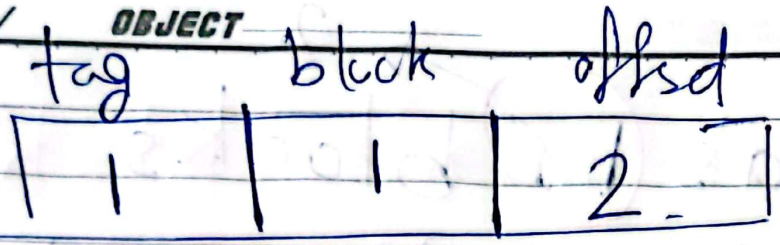
4 bytes.

$4 = 2^{(2)}$ ⟶ offset

(2) two blocks in Cashe.

$2 \text{ block} = 2^{(1)}$ ⟶ block.

DATE / / OBJECT

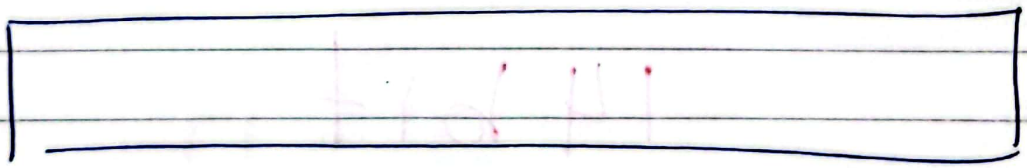


4 bit

Example 2 :-

Memory Contain 2^{14} bytes.
Cash has 16 block.
each block has 8 bytes.

(1) Memory Contain $2^{(14)}$ bytes
(14) ↓ bit address



14 bit

DATE / / OBJECT

Cash has 16 blocks, 2^4
each block has 8 bytes

→

* the number of Memory blocks

$$= \frac{\text{total Memory}}{\text{size of one block in cashe}}$$

$$= \frac{2^{14}}{2^3} = 2^{11}$$

7	4	3
tag	block	offset

14 bit

DATE / / OBJECT

sheet 1 (direct Mapping)

I * Main Memory = 2^{20} bytes.

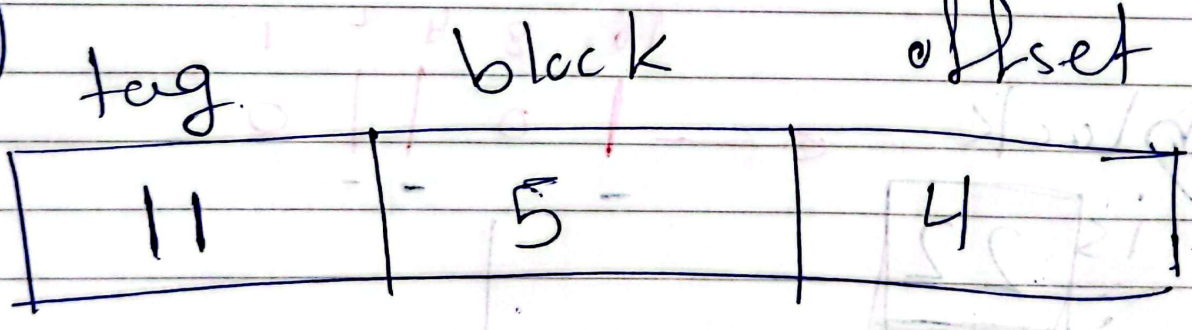
* Cashe has 32 block.

* each block Contain 16 bytes.

a) blocks of Main Memory.

= $\frac{\text{total Main Memory}}{\text{block Contain of Cash}} = \frac{2^{20}}{2^4} = 2^{16}$ block

b)



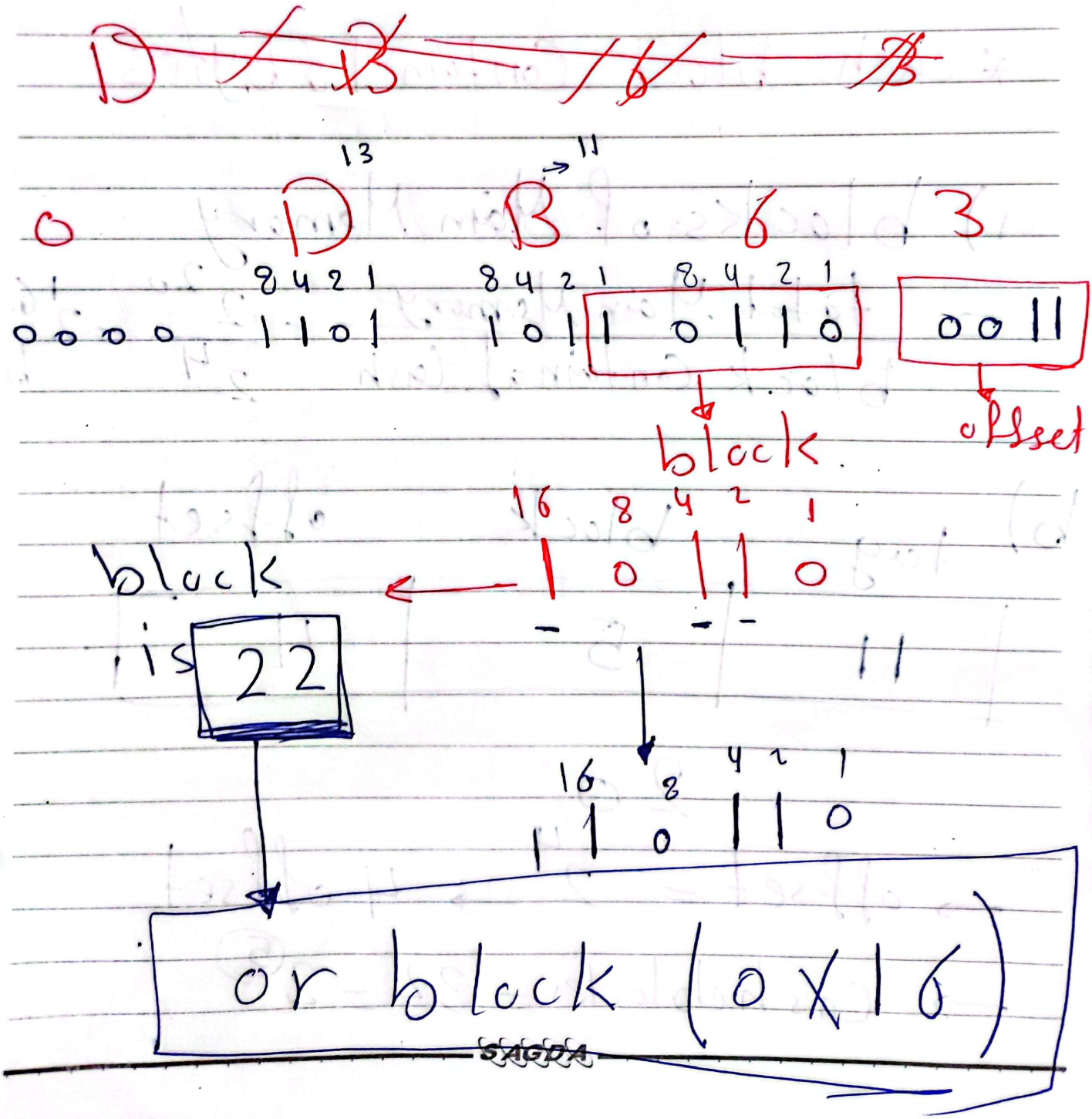
11 | 2^9 | 2^4

→ offset = 2^4 → 4 offset

→ Cashe blocks = 32 = 2^5

DATE / / OBJECT

c) To which Cashe block will the Memory address 0x0DB63 map.



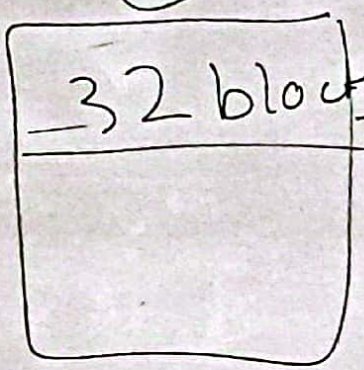
0110454g637

Report (2, 3, 5, 6)

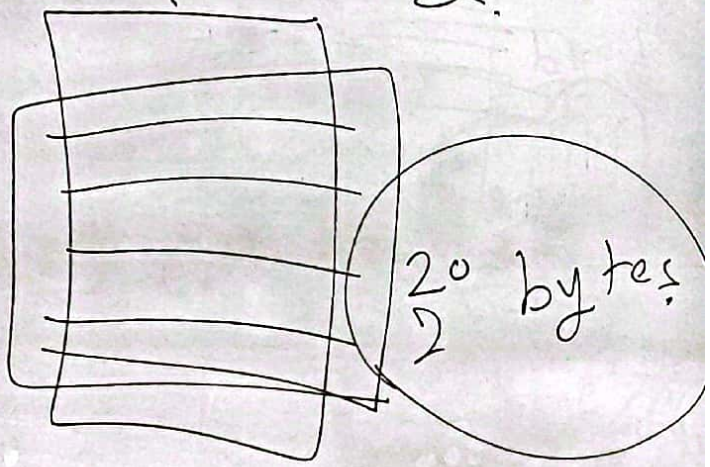
next tuesday

A4

Cashe.

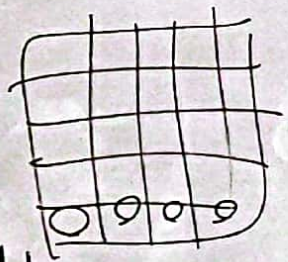


Main Memory.

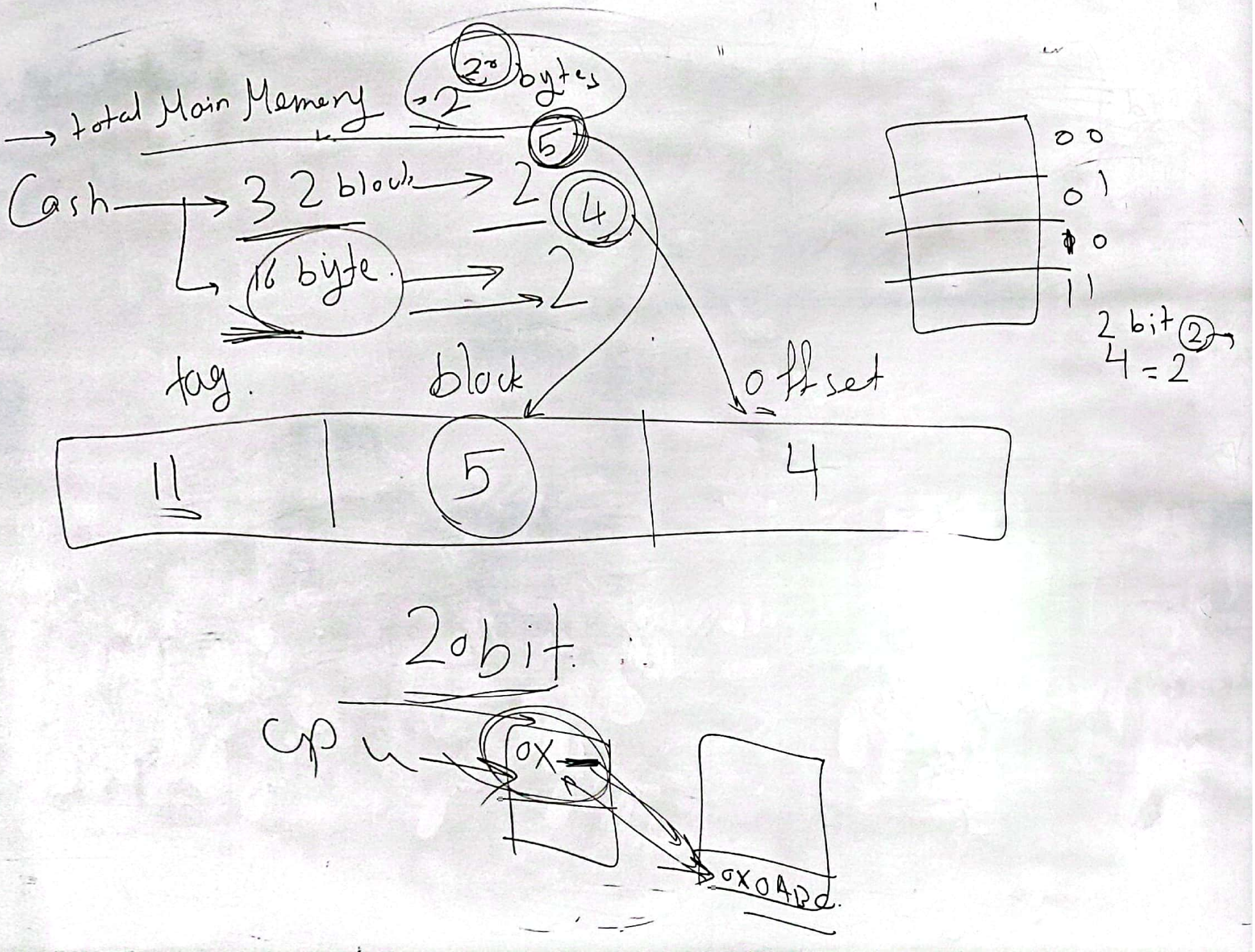


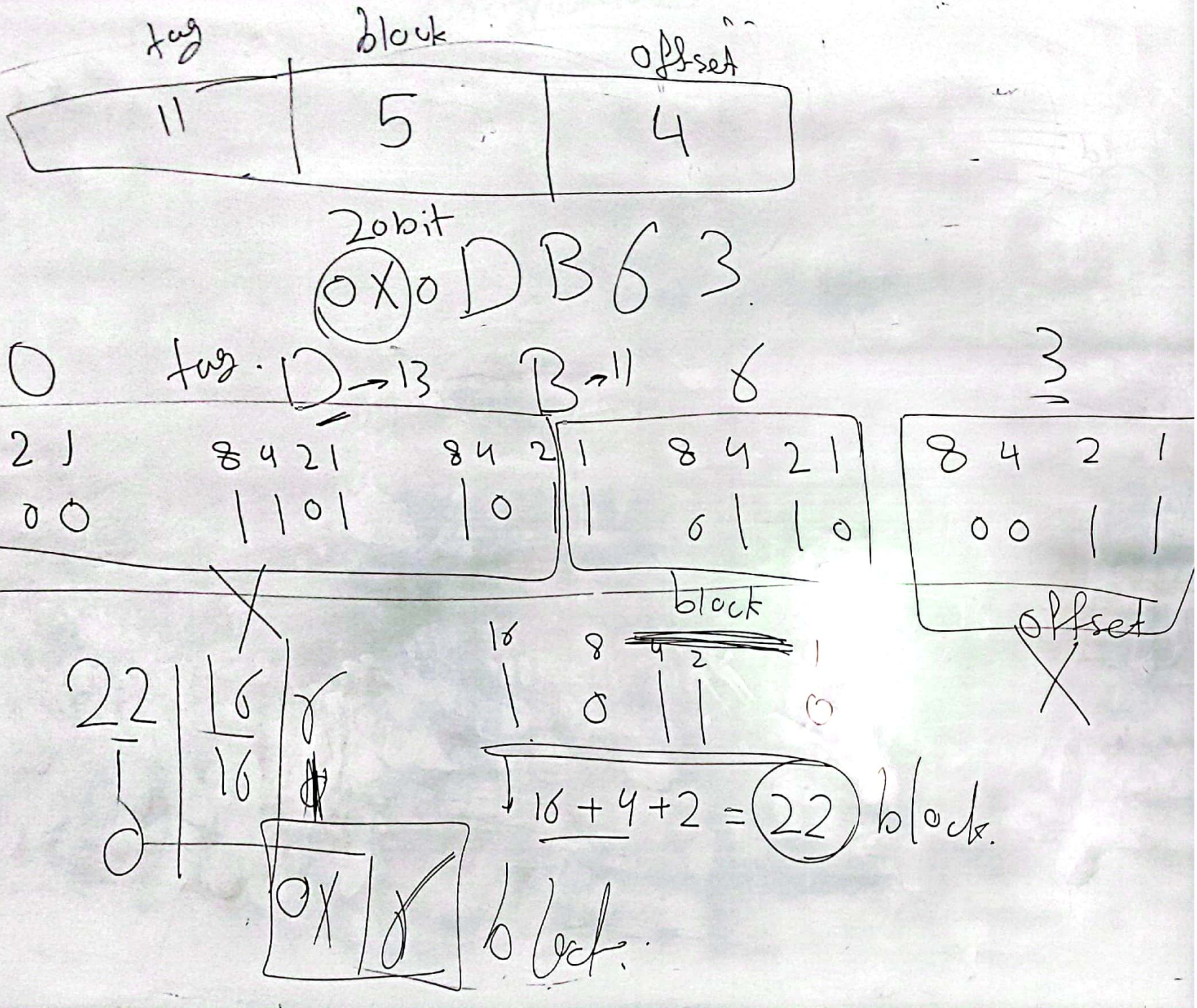
→ total Main Memory = 2^{20} bytes.

Cashe → 32 block
→ 16 bytes



no. of blocks = $\frac{\text{total size of Main Memory}}{\text{Size one of block in Cashe.}} = \frac{2^{20}}{16} = 2^{16}$ block


total Main Memory
2^{20} bytes
Cash
32 block
16 byte
2
2
5
4
00
01
10
11
2 bit
4 = 2
tag
block
offset
11
5
4
2 bit
CPU
0X
0X0ABC


tag
block
offset
11
5
4
20bit
0X0DB63
0
tag
D → 13
B → 11
6
3
8 4 2 1
0 0 0 0
8 4 2 1
1 1 0 1
8 4 2 1
1 0
8 4 2 1
1 0 1 0
8 4 2 1
0 0 1 1
block
offset
16 8 4 2 1
1 0 1 1 0
16 + 4 + 2 = 22 block.
22
16
6
0X
6
block.

* total Main Memory = 2^{32} byte.

Cache → 1024 block → 2^{10}

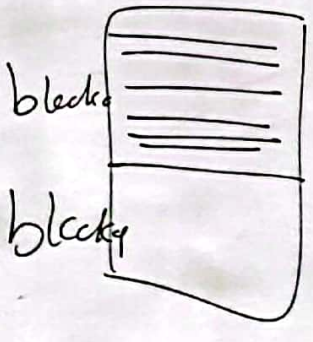
↳ 32 byte → 2^{5}.

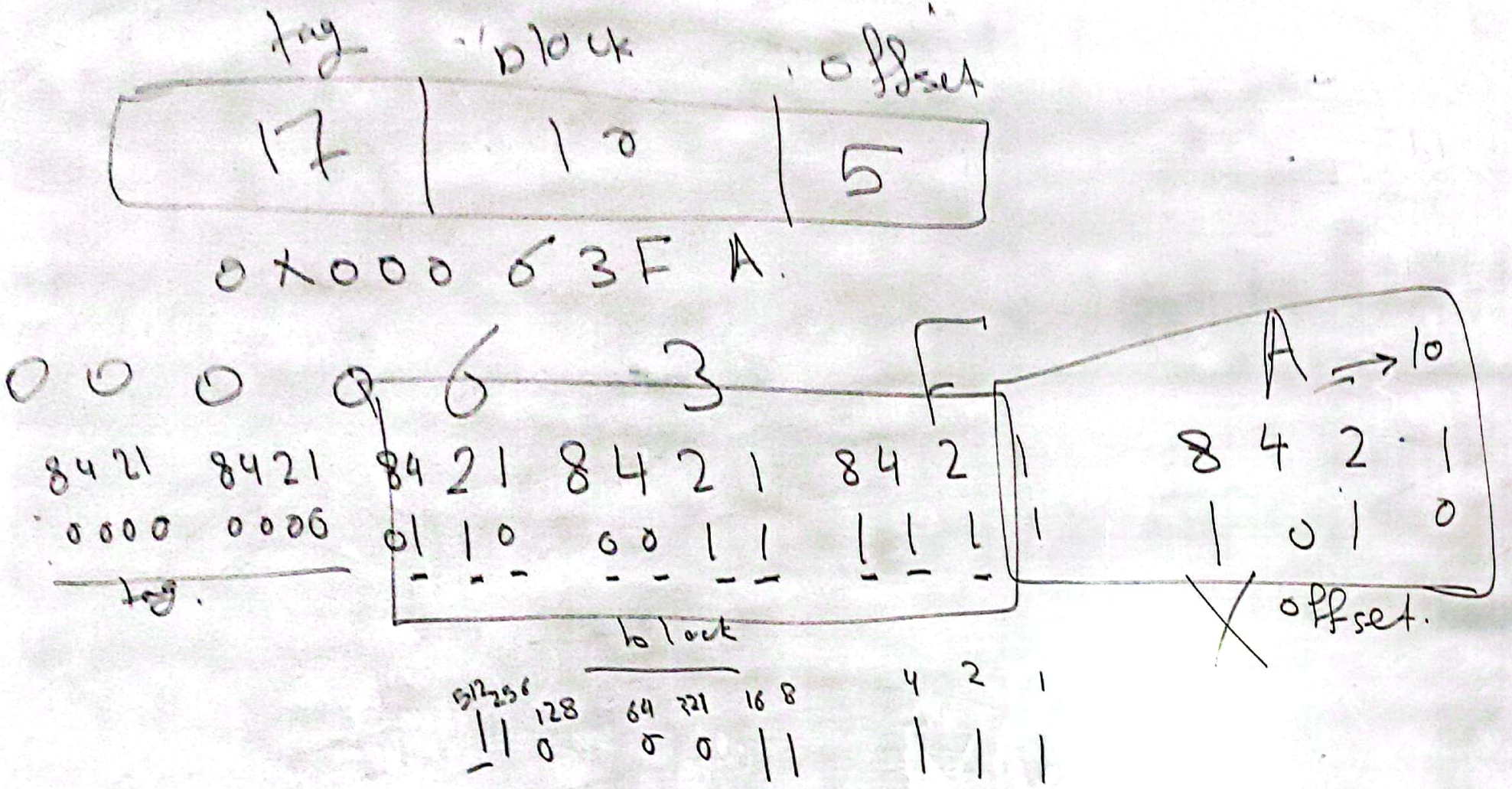
no. of blocks = $\frac{2^{32}}{32 \rightarrow 2^5} = 2^{27}$ block.

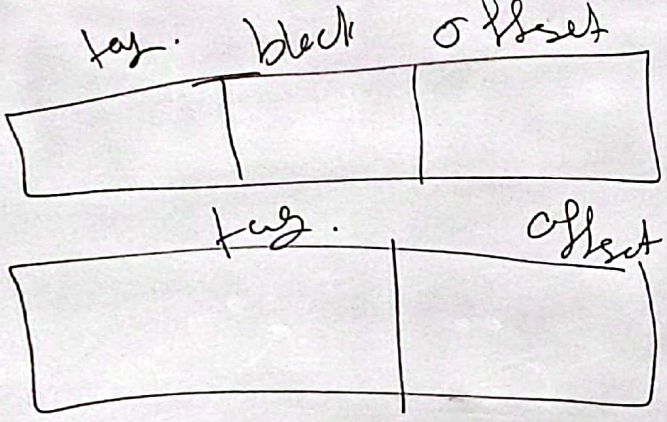
tag	block	offset
17	10	5

32 bit

32 − (10 + 5) = 17




tag
block
offset
17
10
5
0x0006 3FA.
0 0 0 0 6 3 F
A → 10
8421 8421 8421 8421 8421
0000 0006
0 1 1 0 0 0 1 1 1 1 1 1
8 4 2 1
1 0 1 0
tag.
block
offset.
512 256 128 64 32 16 8 4 2 1
1 1 0 0 0 1 1 1 1 1
512+256+16+8+4+2+1 = 799.

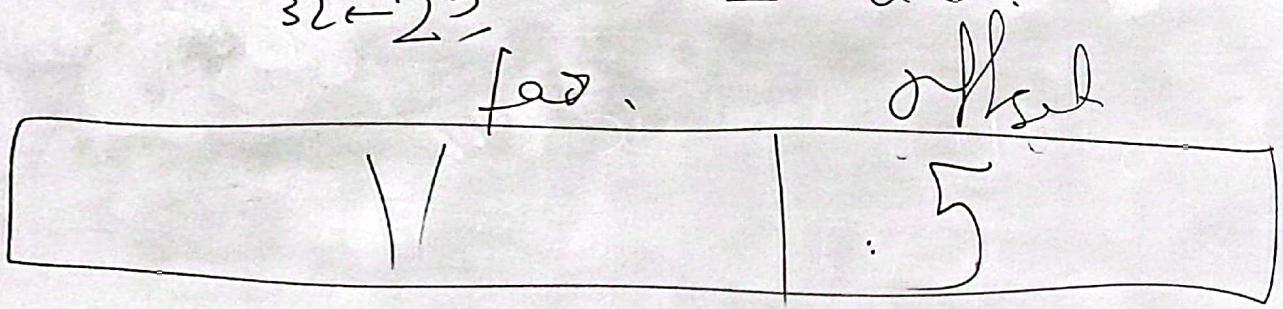


→ total Main Memory = 2^{16} bytes

Cache → 64 block

32 byte → 2^5

n.o of blocks = $\frac{2^{16}}{32 \times 2^5} = 2^{11}$ block.



16 bit