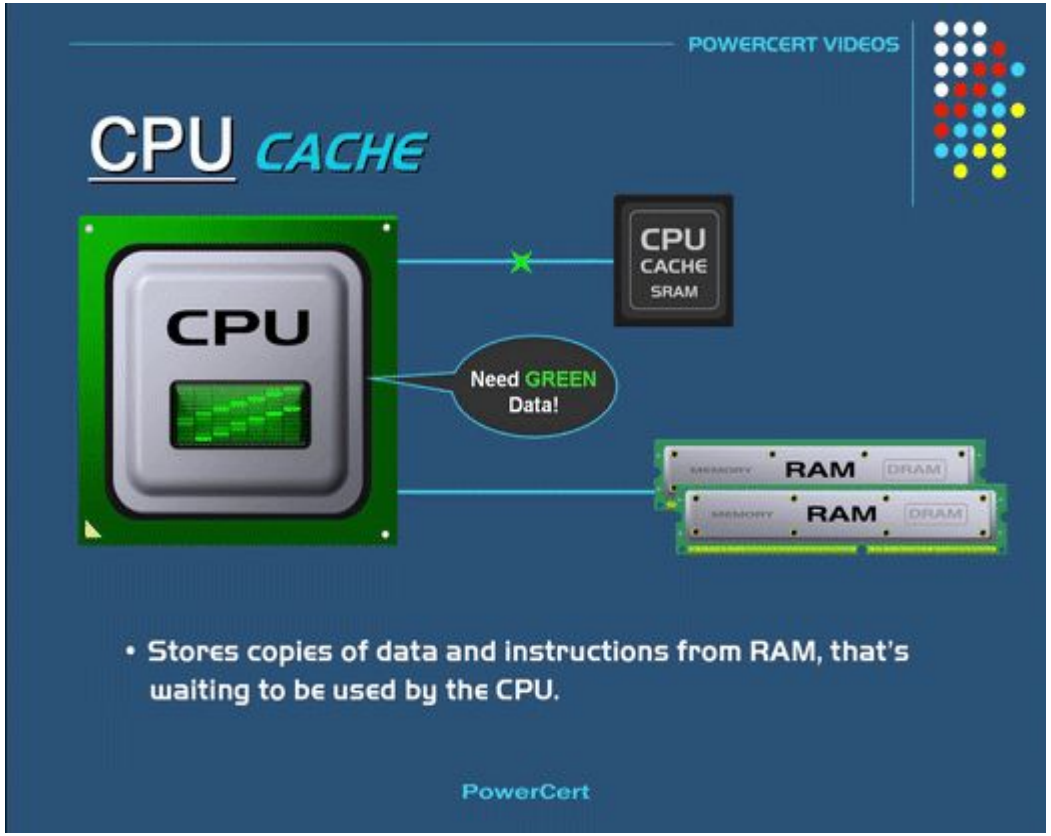
20CS2009
Computer Architecture
Lecture #5

Cache Memory

- is a special very high-speed memory used to speed up and synchronizing with high-speed CPU.
- Cache memory is costlier than main memory or disk memory but economical than CPU registers.
- Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU.



- It holds frequently requested data and instructions so that they are immediately available to the CPU when needed.
- Cache memory is used to reduce the average time to access data from the Main memory.

Locality of reference

Since size of cache memory is less as compared to main memory. So to check which part of main memory should be given priority and loaded in cache is decided based on locality of reference.

Types of Locality of reference

- **Spatial Locality of reference**

  This says that there is a chance that element will be present in the close proximity to the reference point and next time if again searched then more close proximity to the point of reference.

- **Temporal Locality of reference**

  In this Least recently used algorithm will be used. Whenever there is page fault occurs within a word, that will not only load word in main memory but complete page fault will be loaded because spatial locality of reference rule says that if you are referring any word next word will be referred in its register that's why we load complete page table so the complete block will be loaded.

Levels of memory:

Level 1 or Register

It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.

Level 2 or Cache memory

It is the fastest memory which has faster access time where data is temporarily stored for faster access.

Level 3 or Main Memory

It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.

Level 4 or Secondary Memory

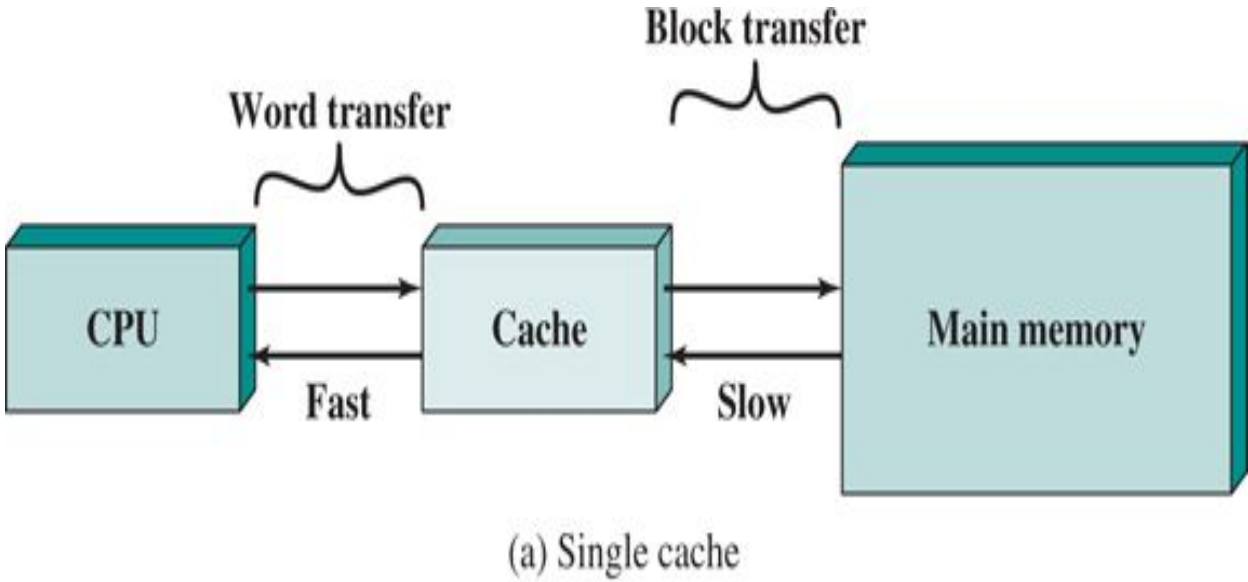
It is external memory which is not as fast as main memory but data stays permanently in this memory.

The **cache contains a copy of portions of the main memory**. When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache.

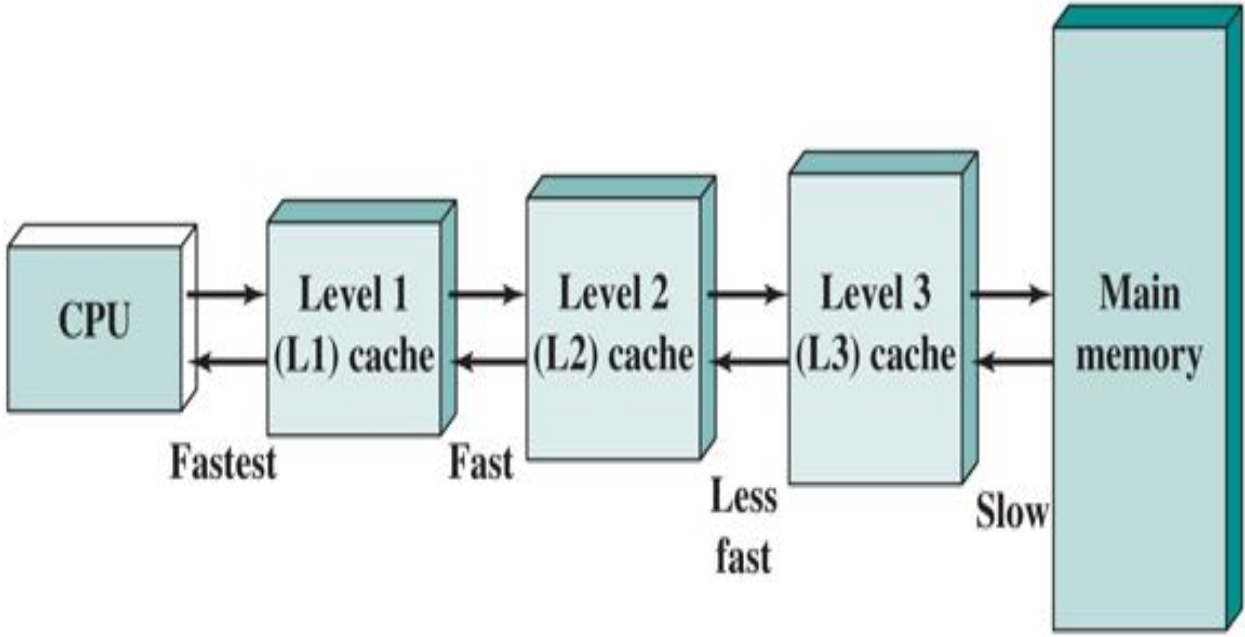
If so, the word is delivered to the processor.

If not, a **block of main memory**, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor.

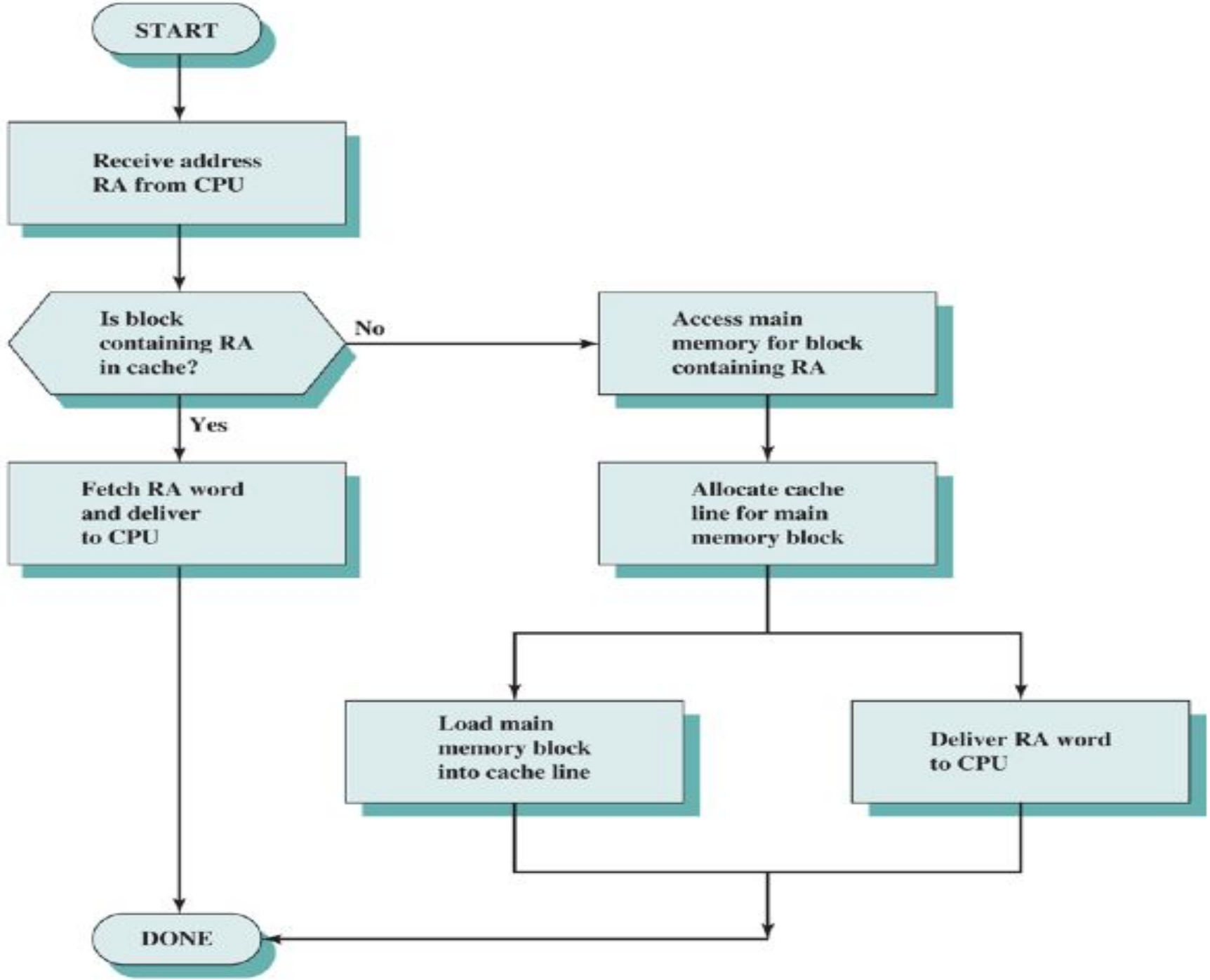
Use of multiple levels of cache. The **L2 cache** is slower and typically larger than the **L1 cache**, and the **L3 cache** is slower and typically larger than the L2 cache.



(a) Single cache



(b) Three-level cache organization

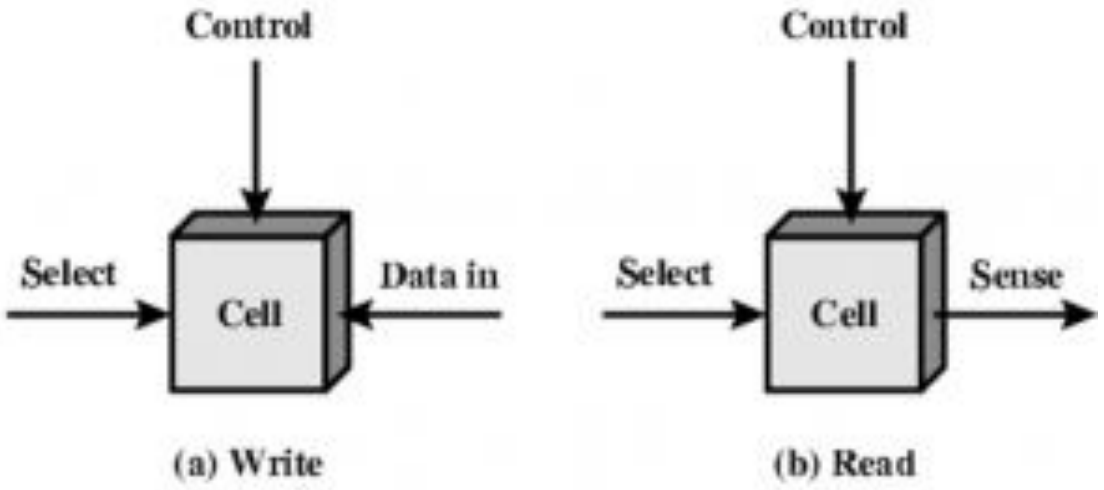

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

- **Definitions**
- Bit = Binary digIT = 0 or 1
- **Byte** = **a sequence of 8 bits** = 00000000, 00000001, ..., or 11111111
- **Word** = a **sequence of N bits** where **N = 16, 32, 64** depending on the computer
- **Measuring Amount of Data/Memory Capacity**
- 1 kilobyte = 1 KB = $\mathbf{2^{10}}$ **bytes** = 1024 bytes
- 1 Megabyte = 1 MB = $\mathbf{2^{20}}$ **bytes** = 1,048,576 bytes

  = 1 KB *1024 = 1MB = $\mathbf{2^{20}}$
- 1 Gigabyte = 1 GB = $\mathbf{2^{30}}$ **bytes** = 1,073,741,824 bytes

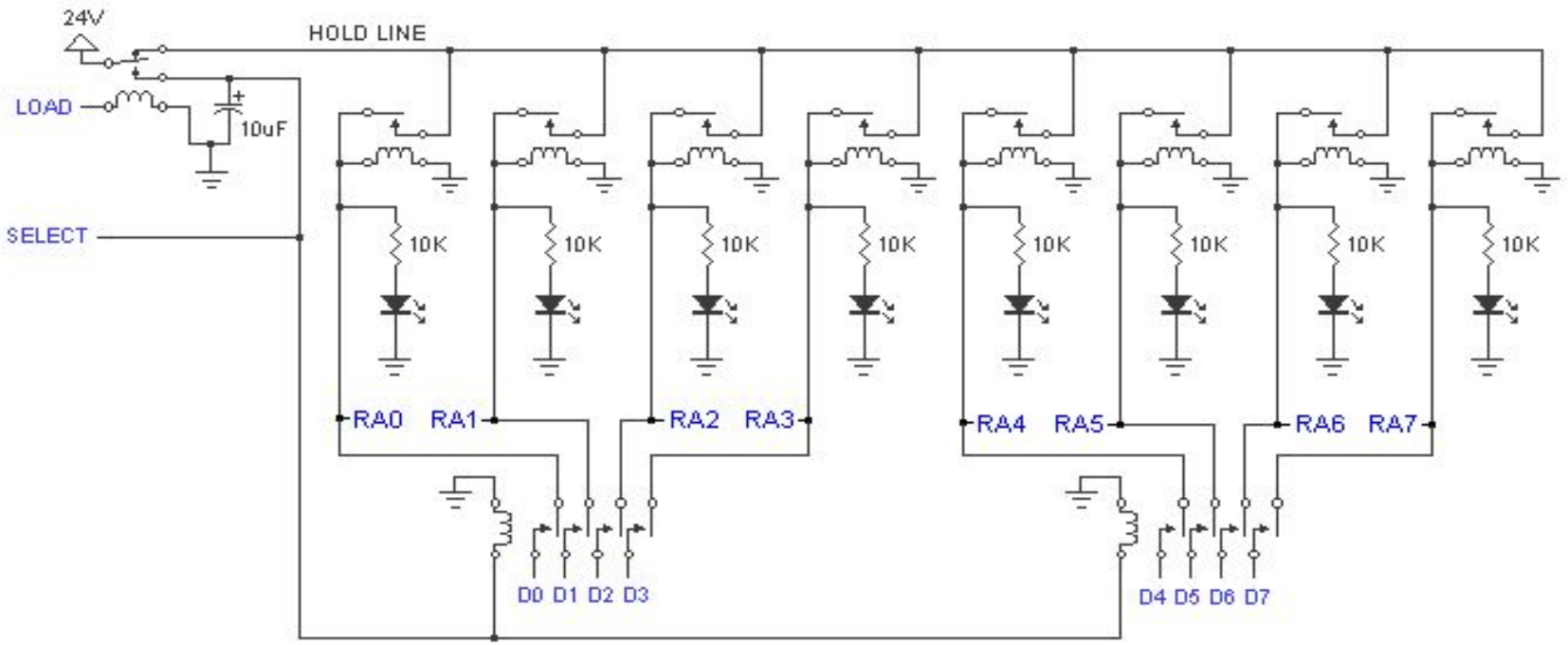
  = 1 MB *1024 = 1GB = $\mathbf{2^{30}}$

- Before going into cache memory mapping we need to know following things…
- What memory – Physical Memory & Logical Memory
- How, data is stored in a memory?
- What is there inside a memory unit…?
- What is memory cell?
- How many bits can be stored in a memory cell?
- In Main memory how many memory cell will be there?



Memory Cell

- A Memory cell can hold only one bit
- Bit = 0 (or) 1
- 8 bit = 1 Byte -> 00000001
- Following diagram shows 8 bit register



8-BIT REGISTER (REGISTER A SHOWN)

- **Cache miss**
  - It occurs when a cache doesn't have the CPU requested data in its memory.
- **Cache Hit**
  - It occurs when a cache have the CPU requested data in its memory.
- **Write-through**
  - Write is done synchronously both to the cache and to the backing store
- **Write-Back**
  - The data is updated only in the cache and updated into the memory at a later time.
  - Each Block in the cache needs a bit to indicate if the data present in the cache was modified(Dirty) or not modified(Clean). If it is clean there is no need to write it into the memory. It is designed to reduce write operation to a memory.

Logical Address Vs Physical Address

- In Simple way- An **address generated by CPU** is called Physical Address
- Logical addresses are also referred as virtual addresses
- **Logical Address Space** – set of logical addresses generated by a program

- Physical Address (**Real address**) is the actual memory address that denotes a memory cell
- **Logical address need to be mapped with physical address before use**.
- This mapping is handled by a hardware device called MMU

Memory Mapping Techniques

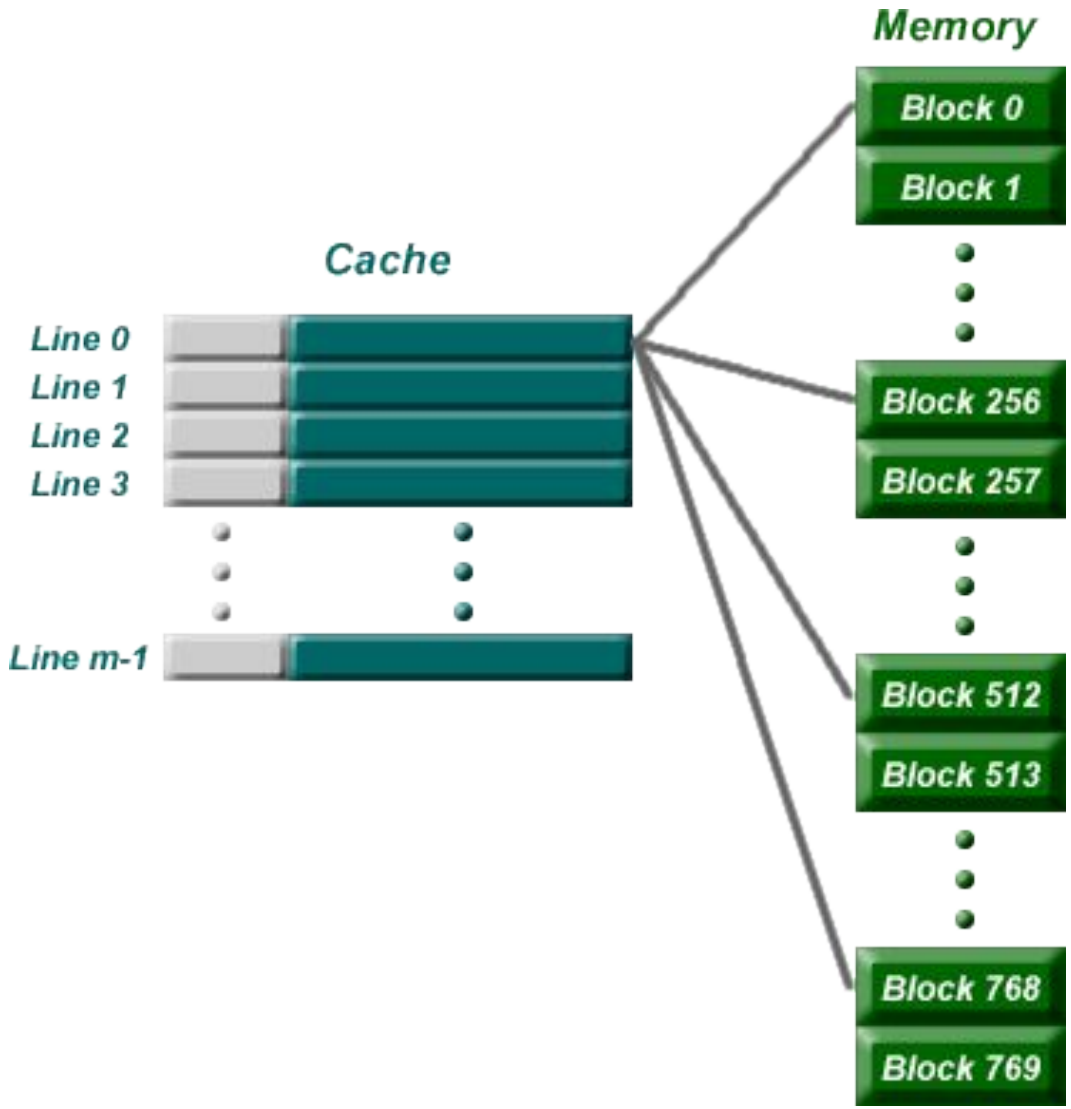
- **Direct Memory Mapping**
- **Associative Memory Mapping**
- **Set Associative Memory Mapping**

Note:

Block: The minimum unit of transfer between cache and main memory.

Line: A portion of cache memory capable of holding one block

Direct Memory Mapping



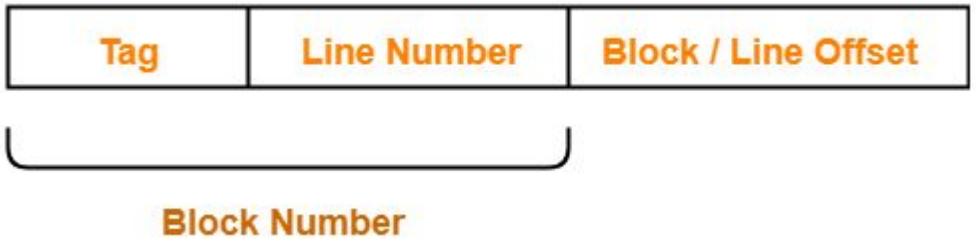
Direct Mapping-

In direct mapping,

- A particular block of main memory can map to only one particular line of the cache.
- The line number of cache to which a particular block can map is given by

Cache line number = (Main Memory Block Address) Modulo (Number of lines in Cache)

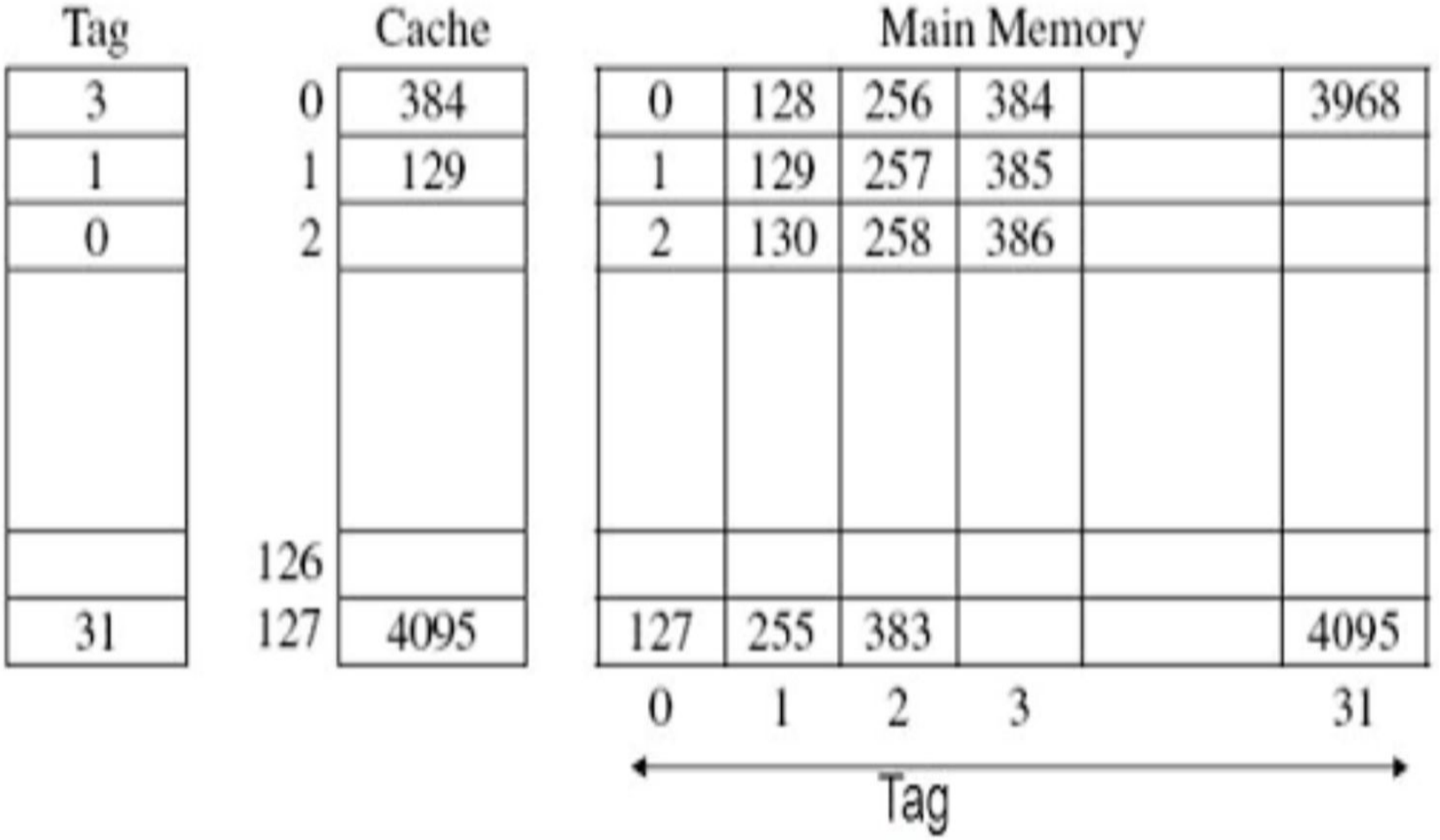
In direct mapping, th

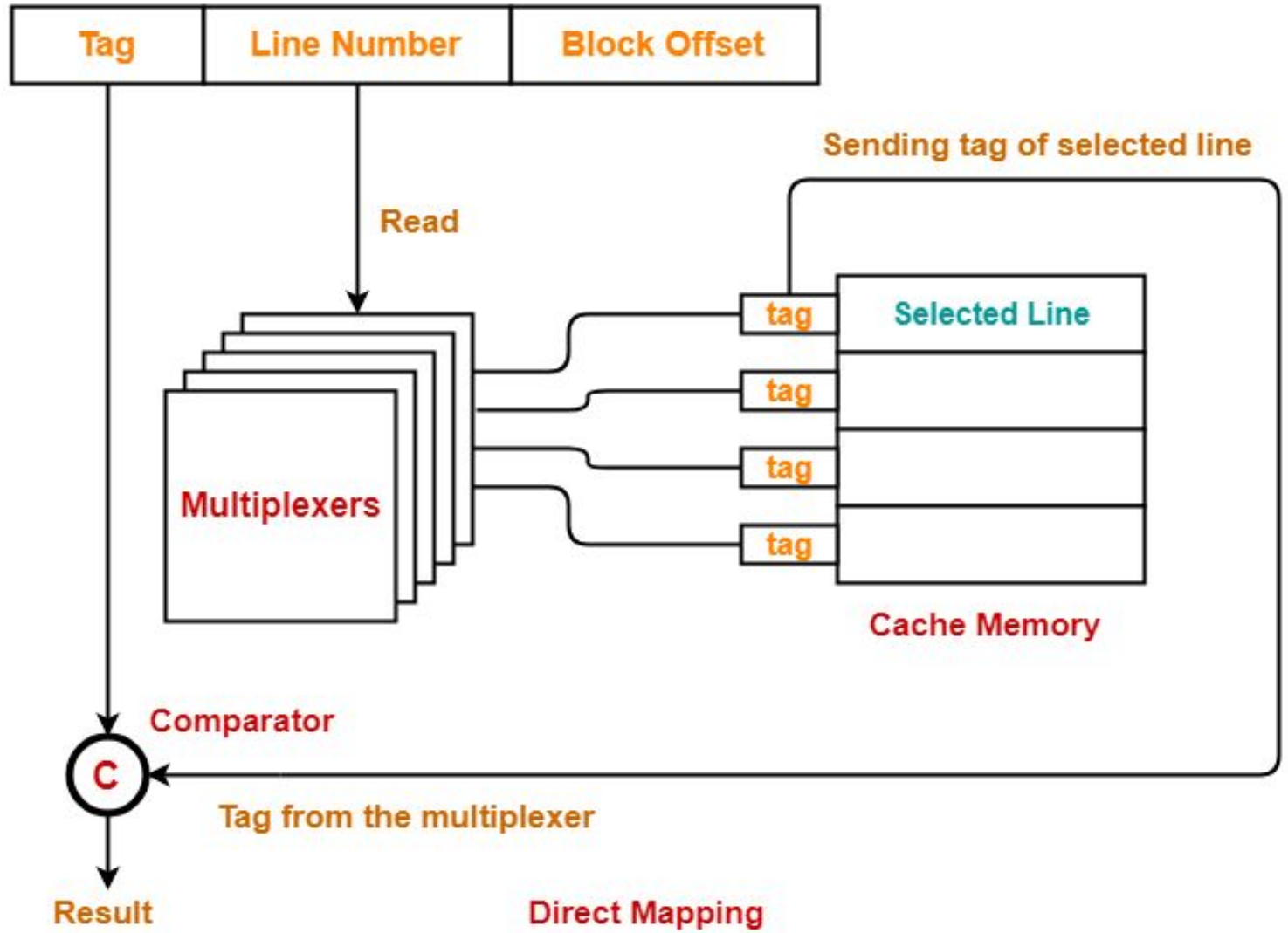


Working of direct mapped cache

- After CPU generates a memory request,
- The line number field of the address is used to access the particular line of the cache.
- The tag field of the CPU address is then compared with the tag of the line.
  - If the two tags match, a cache hit occurs and the desired word is found in the cache.
  - If the two tags do not match, a cache miss occurs.
- In case of a cache miss, the required word has to be brought from the main memory.
- It is then stored in the cache together with the new tag replacing the previous one.

How to Map Cache memory with Main memory in Direct Mapping




Tag
Line Number
Block Offset
Sending tag of selected line
Read
tag
Selected Line
tag
tag
tag
Multiplexers
Cache Memory
Comparator
C
Tag from the multiplexer
Result
Direct Mapping

Step-01:

- Each multiplexer reads the line number from the generated physical address using its select lines in parallel.
- To read the line number of L bits, number of select lines each multiplexer must have = L.

Step-02:

- After reading the line number, each multiplexer goes to the corresponding line in the cache memory using its input lines in parallel.
- Number of input lines each multiplexer must have = Number of lines in the cache memory

Step-03:

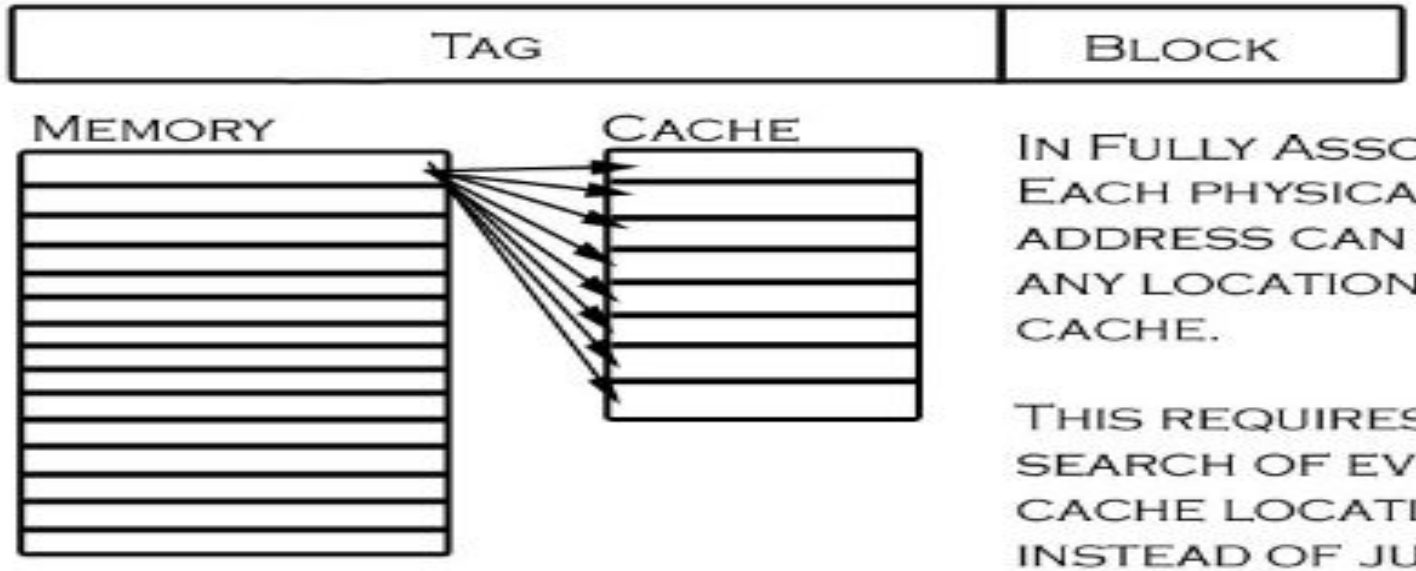
- Each multiplexer outputs the tag bit it has selected from that line to the comparator using its output line.
- Number of output line in each multiplexer = 1.

Step-04:

- Comparator compares the tag coming from the multiplexers with the tag of the generated address.
- Only one comparator is required for the comparison where Size of comparator = Number of bits in the tag
    - If the two tags match, a cache hit occurs otherwise a cache miss occurs.

Associative Memory Mapping

FULLY ASSOCIATIVE

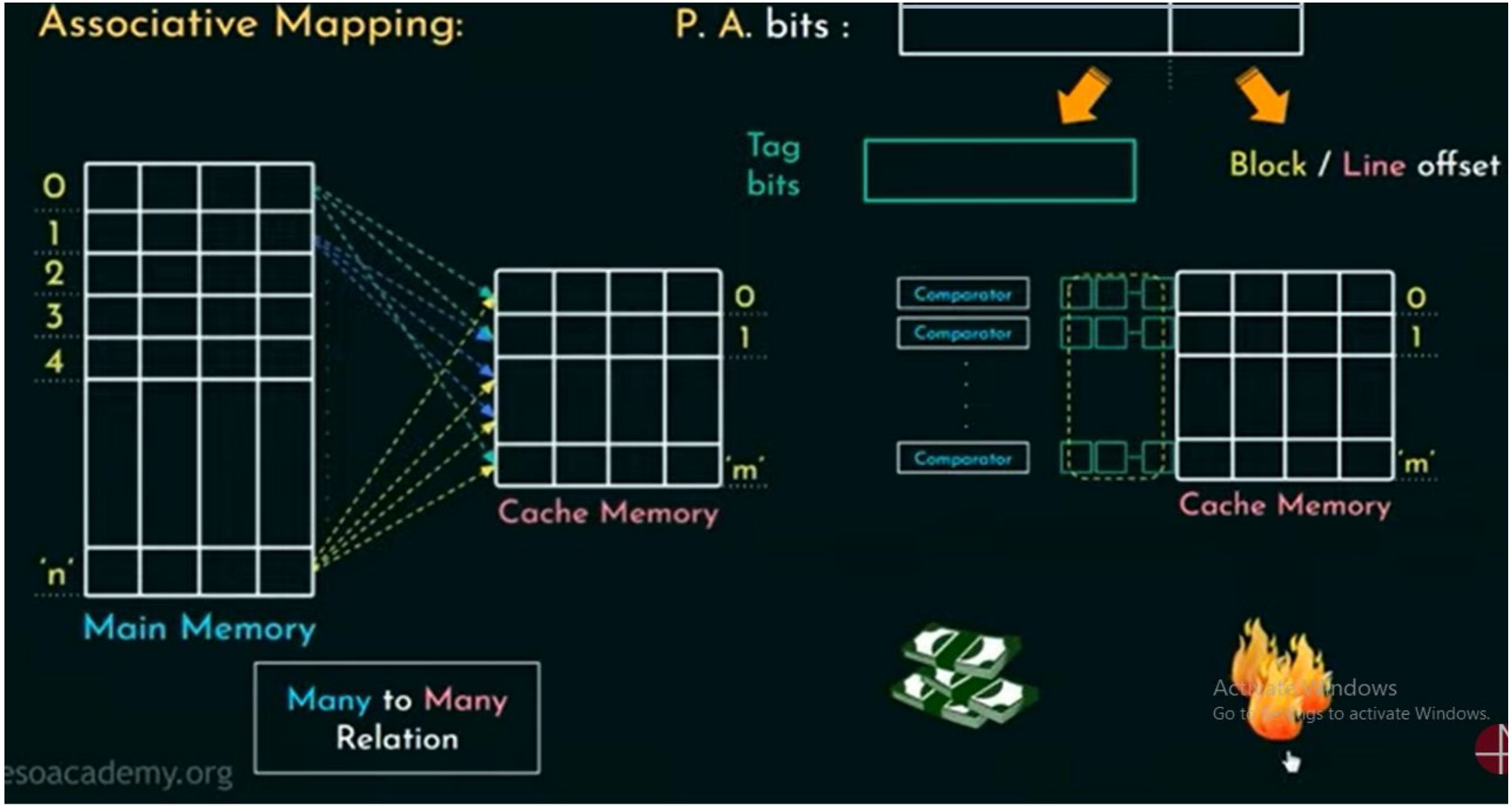


IN FULLY ASSOCIATIVE EACH PHYSICAL ADDRESS CAN MAP TO ANY LOCATION IN THE CACHE.

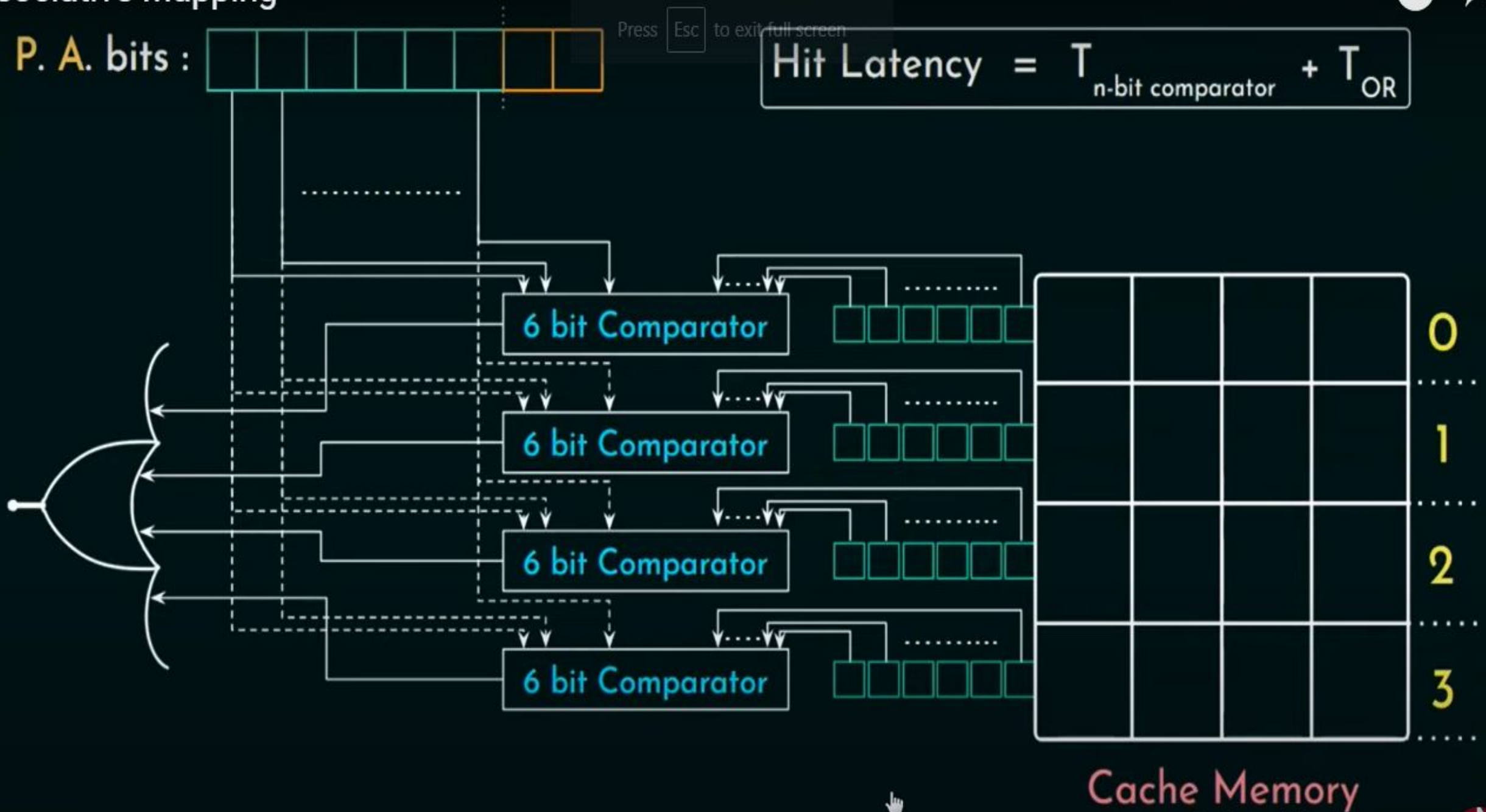
THIS REQUIRES A SEARCH OF EVERY CACHE LOCATION, INSTEAD OF JUST ONE FOR DIRECT MAPPED.

NO REPEAT OF TAGS, UNLESS THE BYTE IS WITHIN THE SAME DATA BLOCK (BUT THAT DATA BLOCK WILL STILL ONLY APPEAR ONCE IN THE CACHE).

Associative Memory Mapping

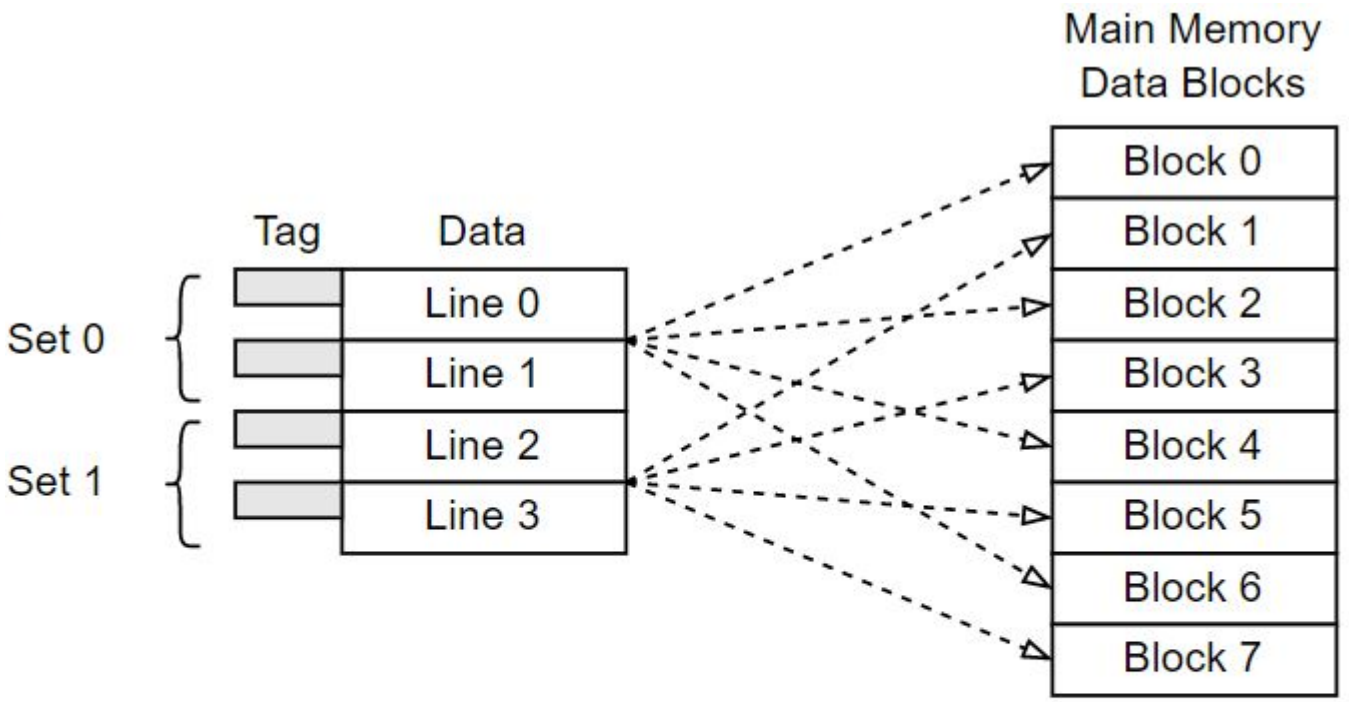


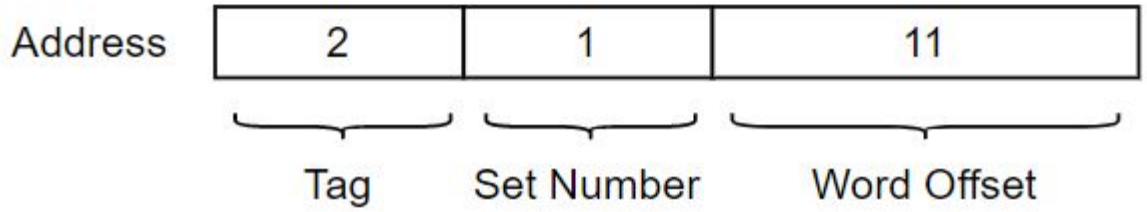
Associative Mapping



Set- Associative Memory Mapping

Main Memory = 16KB
Cache Size = 4KB
Block Size = 2KB
Number of Cache Lines = 4
Number of Main Memory Blocks = 8





In the above cache system, the cache is divide into 2 sets. Each set consists of 2 cache lines. Each block of main memory is directly mapped to a set in the cache. For example, Block 0, Block 2, Block 4, and Block 6 can only occupy the cache line that belongs to Set 0. Similarly, Block 1, Block 3, Block 5, and Block 7 can only occupy the cache line that belongs to Set 1.

Set Number of a Main Memory Block = (Block Number) % (Number of Set in Cache)

It is the exact formula that we use in direct cache mapping. The only difference is instead of mapping each block to a single line, we are mapping each block to a set of lines.

Now we will discuss how CPU handles block requests in set associative cache.

1. Suppose initially the cache is empty.
2. The CPU requests Block 0. Block 0 is not in the cache. CPU brings Block 0 from main memory to cache. Block 0 occupies Set 0, Line 0.
3. The CPU requests Block 4. Block 4 is not in the cache. CPU brings Block 4 from main memory to cache. Block 4 occupies Set 0, Line 1.
4. After that CPU requests Block 2. Block 2 is not in the cache. Block 2 is mapped to Set 0, but both lines of Set 0 are already occupied. In this case, the CPU will choose one of the lines in Set 0 and replace it with Block 2. We generally use LRU (Least Recently Used) algorithm to choose which line to replace.
5. Note that Set 1 is empty but we cannot use lines of another set to store Block 2 since it is direct-mapped to Set 0.

Addressing in Set Associative Mapping

In set associative mapping, the physical memory address is divided into 3 parts

- **Word Offset**

The word offset is used to identify which word of the block we want to access. In the above example, we assumed 1 word = 1 byte. The block size is 2 KB. Therefore, we will need 11 bits to uniquely identify each word.

- **Set Number**

The set number is used to identify which block in the main memory is mapped to which set in the cache. In the above example, there are only 2 sets. Therefore, we need only 1 bit to identify the set.

- **Tag**

As we saw in the above example, a block can occupy any line in its designated set. Tag is used to uniquely identify each block in a set. Each block in a set has a unique tag number. But two blocks belonging to different sets may have the same tag number. In the above example, the number of blocks mapped to a set is 4. Thus, we need 2 bits to uniquely identify each block.

We can calculate the number of tag bits by simply subtracting word offset and set number bits from main memory bits.

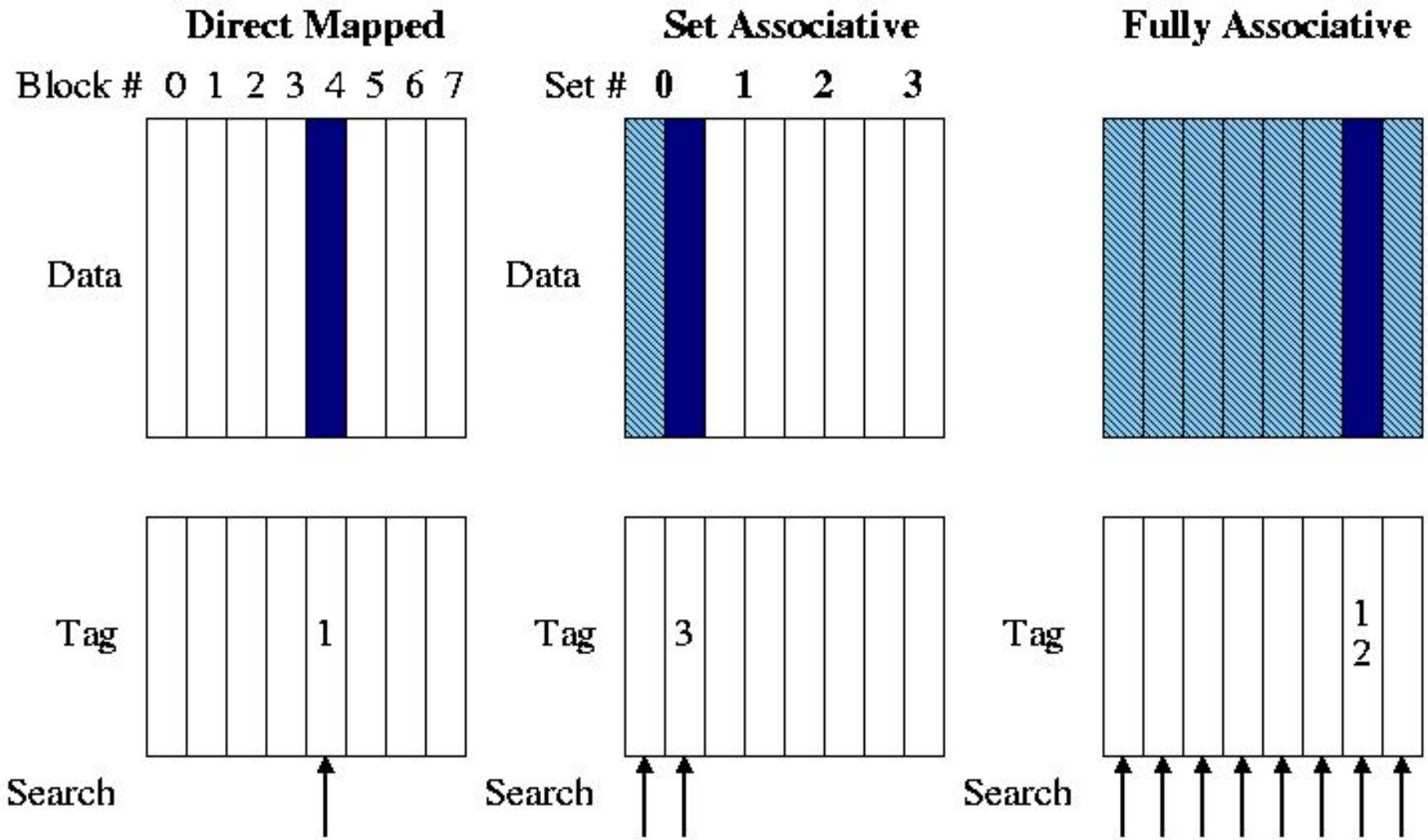
Tag Number = 14 – 1 (Set Number) – 11 (Word Offset) = 2 bits

Note

The least significant bits of the main memory address are used for word offset and most significant bits are used for tag number. The bits in the middle are used for set number.

K Way Set Associative Cache

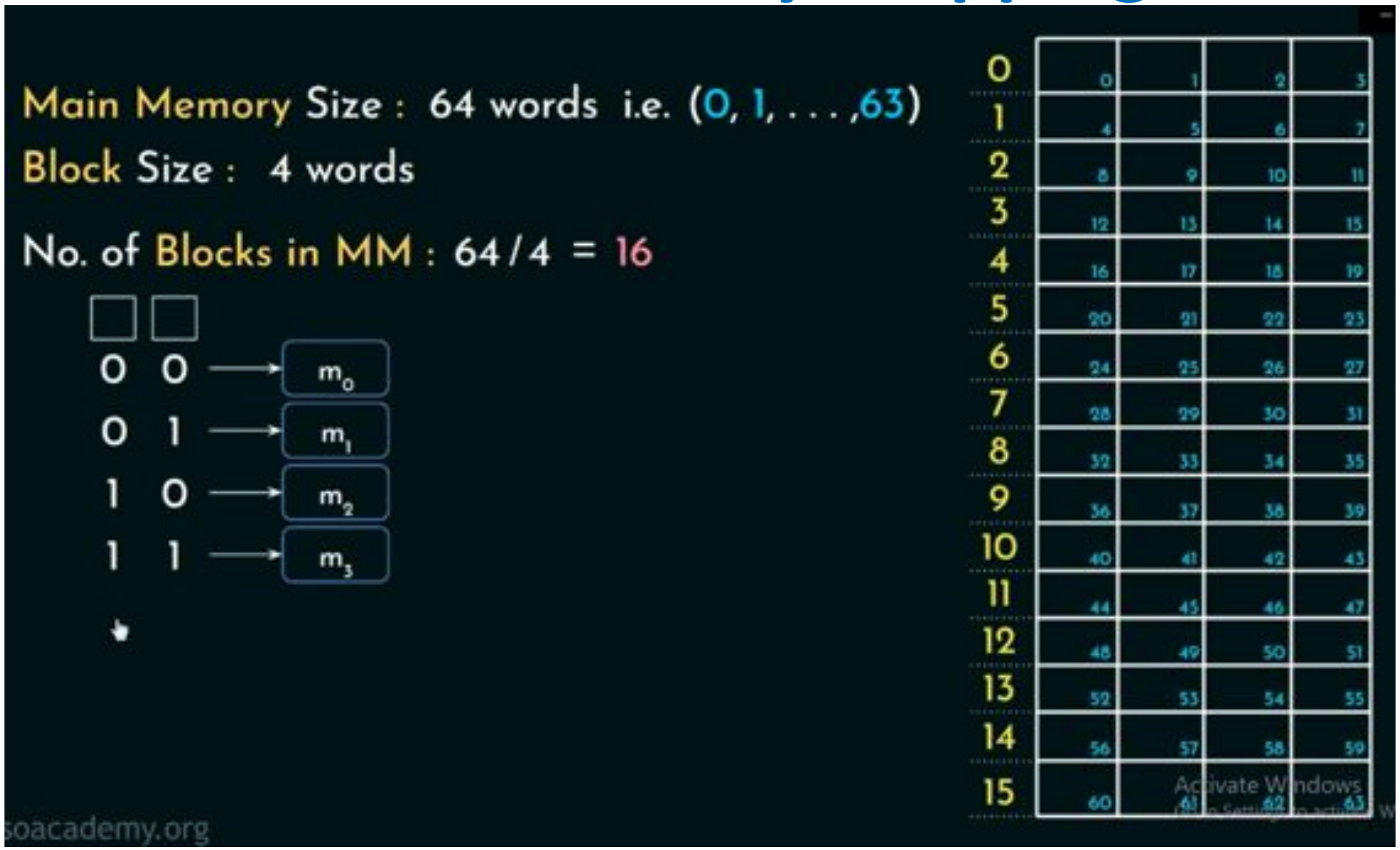
K way set associative cache means that each set in the cache consists of K-lines.

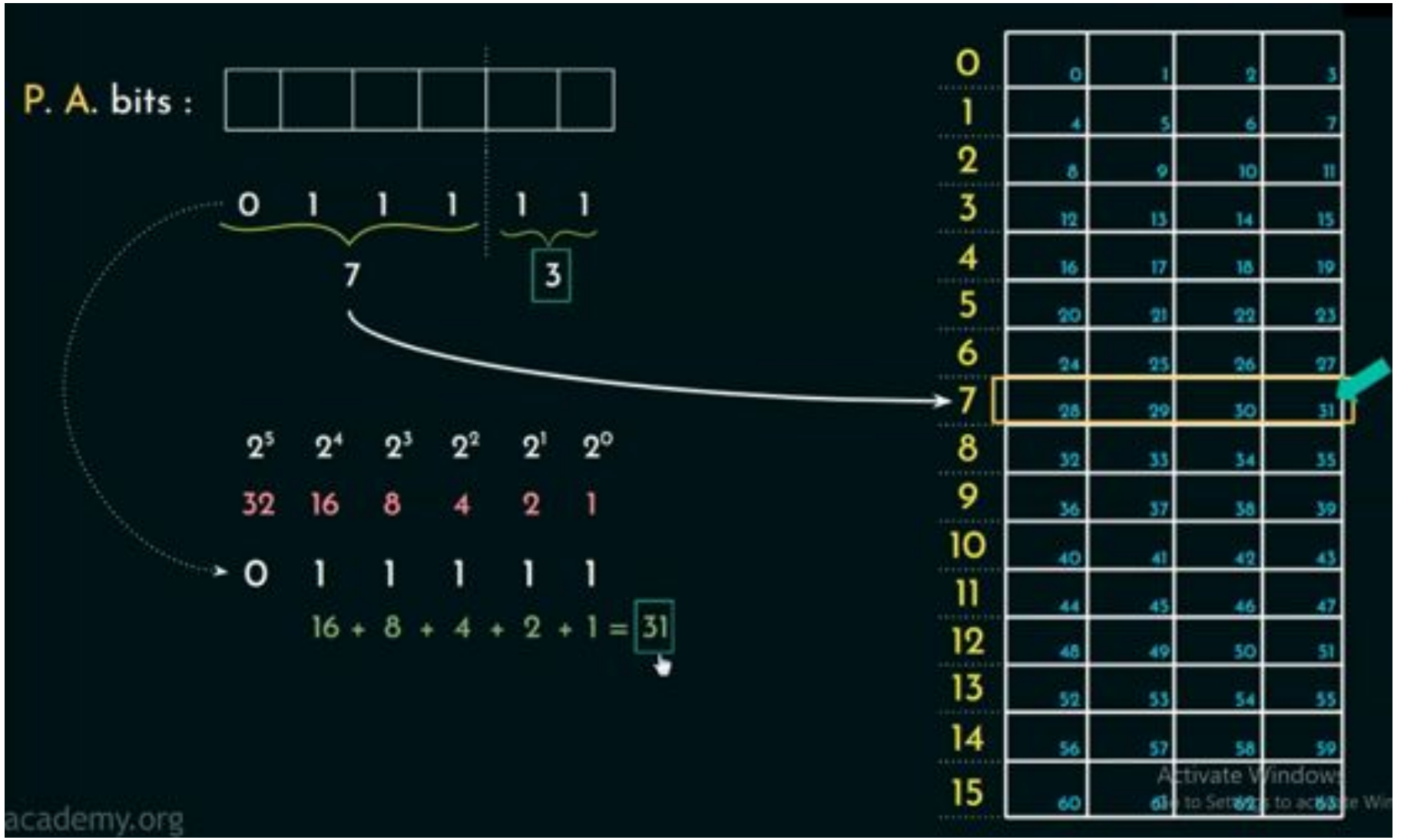

Direct Mapped
Block # 0 1 2 3 4 5 6 7
Data
Tag
1
Search
Set Associative
Set # 0 1 2 3
Data
Tag
3
Search
Fully Associative
Tag
1
2
Search

S.No	Direct-mapping	Associative Mapping	Set-Associative Mapping
1.	Needs only one comparison because of using direct formula to get the effective cache address.	Needs comparison with all tag bits, i.e., the cache control logic must examine every block's tag for a match at the same time in order to determine that a block is in the cache/not.	Needs comparisons equal to number of blocks per set as the set can contain more than 1 blocks.
2.	Main Memory Address is divided into 3 fields : TAG, **BLOCK** & WORD. The BLOCK & WORD together make an index. The least significant TAG bits identify a unique word within a block of main memory, the BLOCK bits specify one of the blocks and the Tag bits are the most significant bits.	Main Memory Address is divided into 1 fields : TAG & WORD.	Main Memory Address is divided into 3 fields : TAG, **SET** & WORD.
3.	There is one possible location in the cache organization for each block from main memory because we have a fixed formula.	The mapping of the main memory block can be done with any of the cache block.	The mapping of the main memory block can be done with a particular cache block of any direct-mapped cache.
4.	Search time is less here because there is one possible location in the cache organization for each block from main memory.	Search time is more as the cache control logic examines every block's tag for a match.	Search time increases with number of blocks per set.

Problems

Direct Memory Mapping




P. A. bits :
0 1 1 1 1 1
7
3
2^5 2^4 2^3 2^2 2^1 2^0
32 16 8 4 2 1
0 1 1 1 1 1
16 + 8 + 4 + 2 + 1 = 31
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
60
Activate Windows
academy.org

Cache Size : 16 words

Line Size : 4 words

No. of Lines in Cache : 16 / 4 = 4 i.e. 0, 1, 2, 3

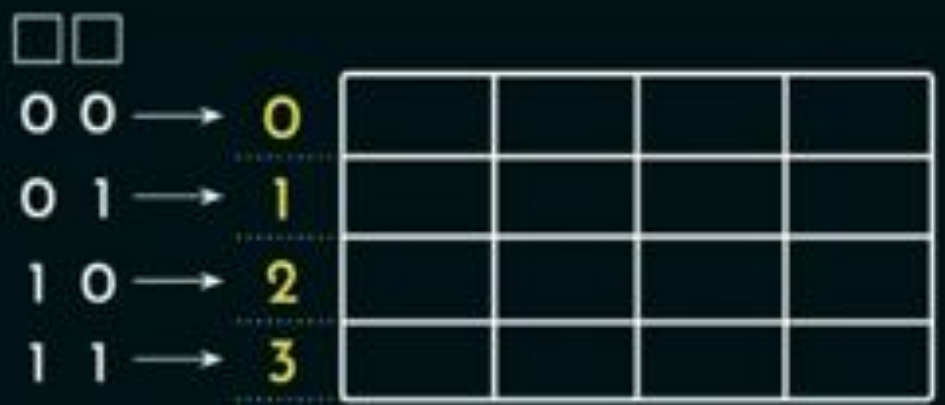
Block Size : 4 words

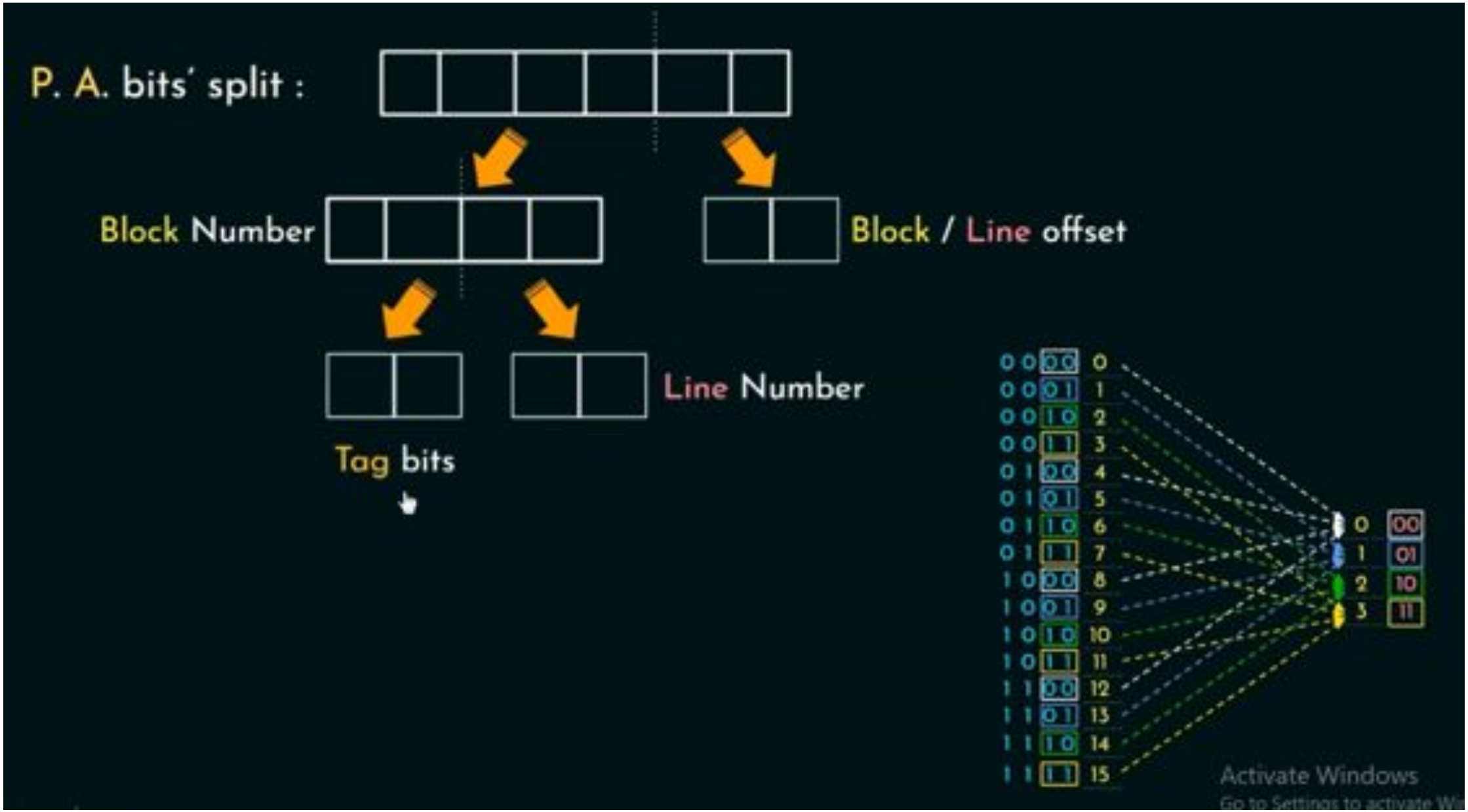
Block Size = Line Size

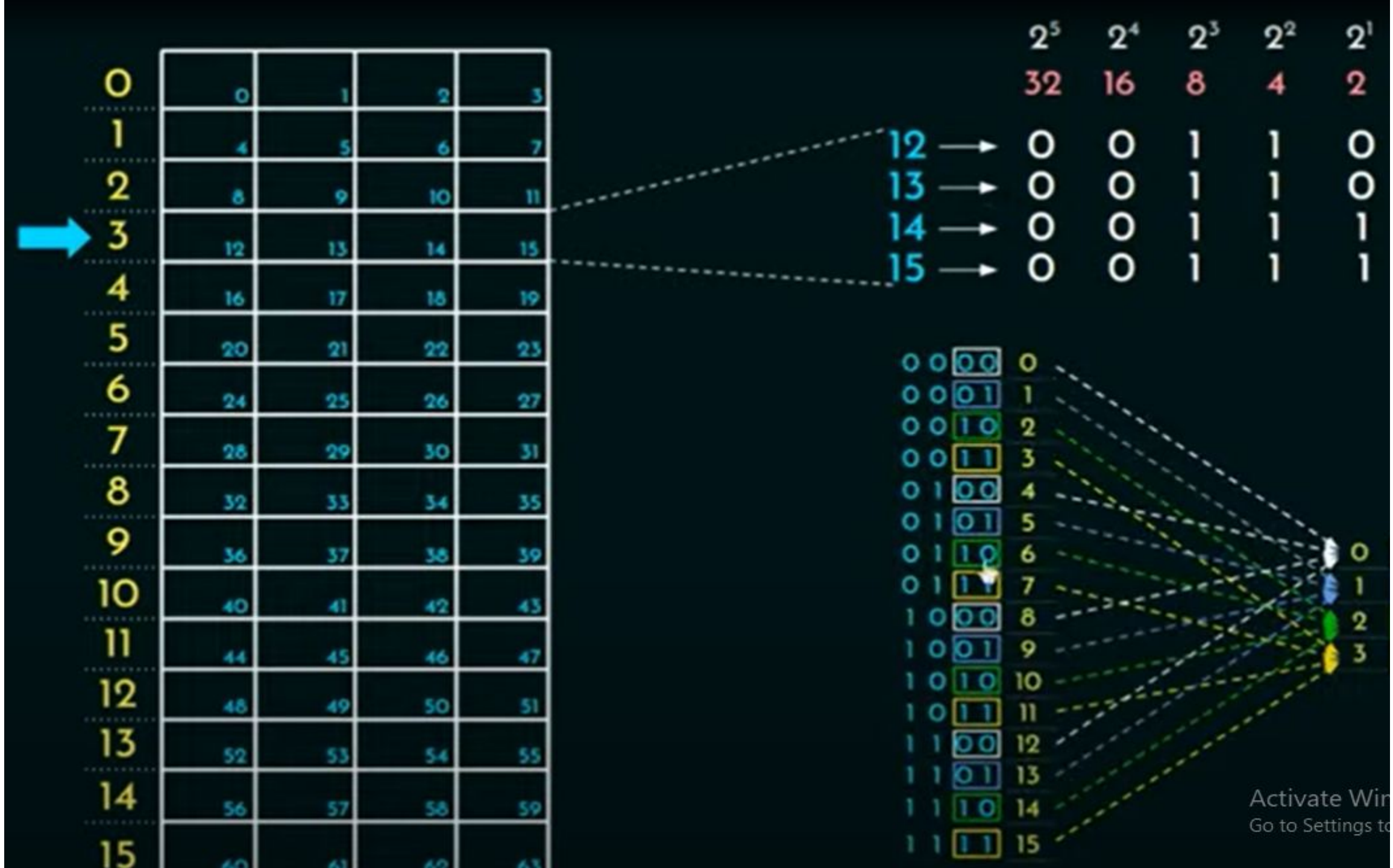
4 Lines

$\log_2 4 = \log_2 2^2$

= 2 bits




P. A. bits' split :
Block Number
Block / Line offset
Line Number
Tag bits
0 0 00 0
0 0 01 1
0 0 10 2
0 0 11 3
0 1 00 4
0 1 01 5
0 1 10 6
0 1 11 7
1 0 00 8
1 0 01 9
1 0 10 10
1 0 11 11
1 1 00 12
1 1 01 13
1 1 10 14
1 1 11 15
0 00
1 01
2 10
3 11
Activate Windows


0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
24 25 26 27
28 29 30 31
32 33 34 35
36 37 38 39
40 41 42 43
44 45 46 47
48 49 50 51
52 53 54 55
56 57 58 59
2^5 2^4 2^3 2^2 2^1
32 16 8 4 2
12 → 0 0 1 1 0
13 → 0 0 1 1 0
14 → 0 0 1 1 1
15 → 0 0 1 1 1
0 0 0 0 0
0 0 0 1 1
0 0 1 0 2
0 0 1 1 3
0 1 0 0 4
0 1 0 1 5
0 1 1 0 6
0 1 1 1 7
1 0 0 0 8
1 0 0 1 9
1 0 1 0 10
1 0 1 1 11
1 1 0 0 12
1 1 0 1 13
1 1 1 0 14
1 1 1 1 15
0
1
2
3
Activate Win
Go to Settings to

Given:

Main Memory size

Cache Size

Block size

Note: Block size = cache line size

Main Memory(MM)

- **No of Blocks in MM (Z)= MM size/Block size**
- **Block number bits = log Z**

Cache Memory(CM)

- **No of cache lines in CM (C)= CM size/Block size**
- **Cache line number bits = log C**

No of Tag bits(n)

n = MM size / Cache size

Assignment Problem Discussion

1. Consider a direct mapped cache of size 16 KB with a block size of 256 bytes. The size of the main memory is 128 KB.
Find the following
a)Number of bits in the tag
b)Tag directory size

No of bit to represent Main Memory – No of bits to represent Cache Memory

2.Consider a direct mapped cache of size 512 KB with block size 1 KB. There are 7 bits in the tag. Find

a) Size of main memory

b)Tag directory size

Given:

Cache Memory = 512 KB = 2^9 x 2^{10} B = 2^{19} B

Block Size = 1KB = 2^{10} B

Tag bits = 7 bits

Solution:

No of Cache lines= Cache size / Block size = 2^{19} / 2^{10} = 2^9 lines

a)Size of main memory = 2^{26} B = 2^6 x 2^{20} B = 64 MB

b) Tag directory size = No of Cache lines x Tag bits = 2^9 x 7 = 512 x7 = 3584 bits

Note:

Once we know the cache size , we can get the number of bits to represent cache memory. By adding tag bits to it , we will get no of bits used to represent Main memory.

No of bits to rep MM = Tag bits + No of bits to rep CM

Where **No of bits to rep CM = no of bits rep cache lines + no of bits to rep block size**

3 .Consider a direct mapped cache with block size 4 KB. The size of main memory is 16 GB and there are 10 bits in the tag. Find a) Size of cache memory b) Tag directory size

Given:

Main Memory = 16 GB = 2^4 x 2^{30} B = 2^{34} B

Block Size = 4KB = 2^{12} B

Tag bits = 10 bits

a)Size of Cache memory = 2^{26} B = 2^6 x 2^{20} B = 64 MB

No of bits to rep Cache lines = no of bit to rep MM – (Tag bits + No of bits to rep bits Block size)

= 34 – (10 + 12) = 34 – 22 = 12 bits

∴ Cache size = No of cache line x line size = **2^{12} x 4KB = 2^{12} x 2^{12} = 2^4 x 2^{20} = 16MB**

b) Tag directory size = No of Cache lines x Tag bits = 2^{12} x 10 = 40960 bits

= 5120 bytes

Thus, size of tag directory = 5120 bytes

Example 1

MM Size: 256 MB
Cache Size: 512 KB

- No. of tag bits?

Sol. Tag bits: Identifies the MM block residing in the Cache Line.

Block Size = Line Size

$\log_2$ (MM Size : Cache Size)

MM Size = 256 MB
$= 2^8 \times 2^{20}$ B
$= 2^{28}$ B

Cache Size = 512 KB
$= 2^9 \times 2^{10}$ B
$= 2^{19}$ B

$\therefore$ No. of Tag bits : $\log_2 (2^{28} / 2^{19}) = \text{Log}_2 2^{(28-19)}$
$= \text{Log}_2 2^9 = 9$

Example 2

MM Size: 4 GB
Cache Size: 1 MB
Block Size: 4 KB

1. P.A. bits' split?
2. Tag directory size?

Note :
No. of lines in cache x No of tag bits

Sol. MM Size = 4 GB = $2^2 \times 2^{30}$ B = $2^{(2+30)}$ B = 2^{32} B

∴ No. of P.A. bits = $\log_2 2^{32}$ = 32

Block Size = 4 KB = $2^2 \times 2^{10}$ B = 2^{12} B

No. of Blocks in MM = $2^{32} / 2^{12}$ = 2^{20}

∴ Block number bits = $\log_2 2^{20}$ = 20

∴ Block offset = $\log_2 2^{12}$ = 12

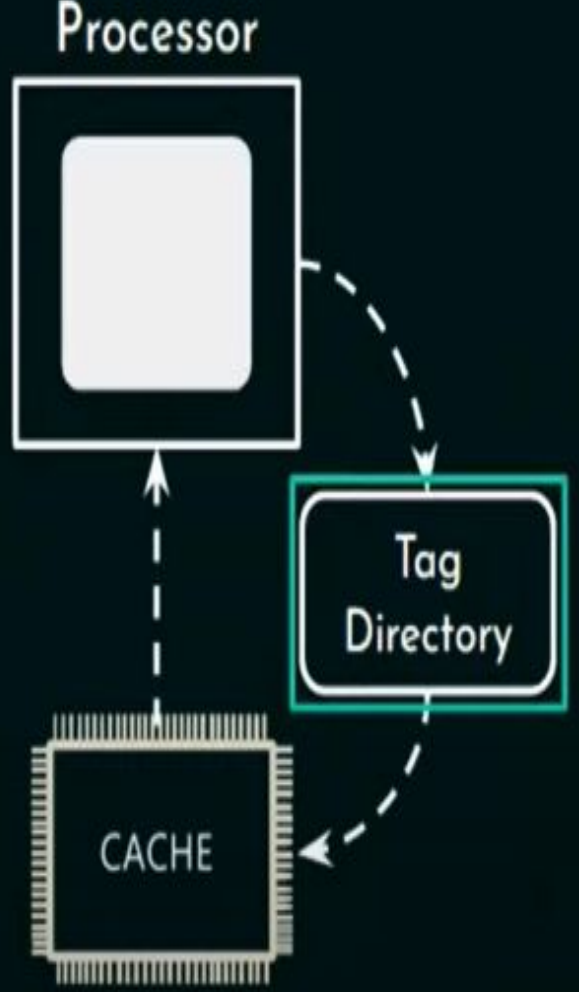
Cache Size = 1 MB = 1×2^{20} B = 2^{20} B

No. of Lines in Cache = $2^{20} / 2^{12}$ = 2^8

∴ Line number bits = $\log_2 2^8$ = 8

No. of Tag bits : P.A. bits - (Line no. bits + offset) = 32 - (8+12)

Sol.



Tag directory:

- Keeps primarily the record of Tag bits, cache line-wise.
- No. of entries = No. of Cache lines.

No. of Lines in Cache = 2^8

No. of Tag bits = 12

Tag directory size = 2^8 x 12 bits

Example 3

Byte-addressable MM Size: 16 GB

Block Size: 16 KB

No. of Tag bits: 10

- Cache size?

Sol. MM Size = 16 GB = $2^4 \times 2^{30}$ B = 2^{34} B

∴ No. of P.A. bits = $\log_2 2^{34}$ = 34

Block Size = 16 KB = $2^4 \times 2^{10}$ B = 2^{14} B

∴ Block offset = $\log_2 2^{14}$ = 14

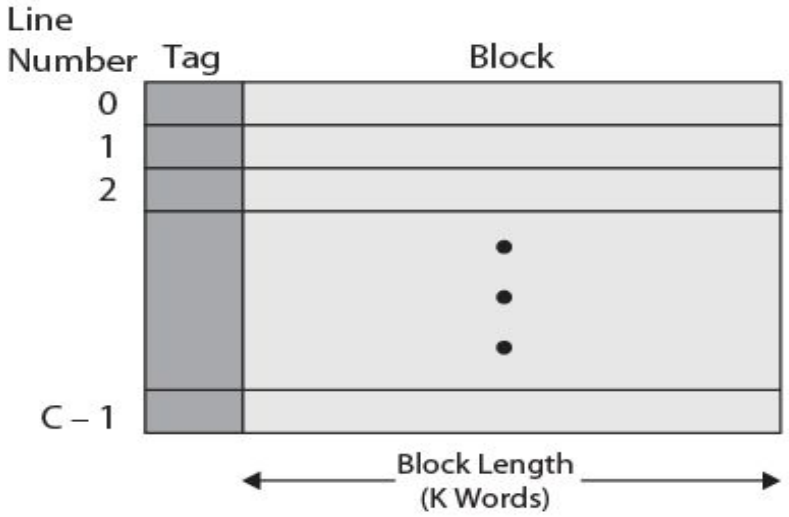
No. of Line number bits : P.A. bits - (Tag bits + offset) = 34 - (10+14) = 10

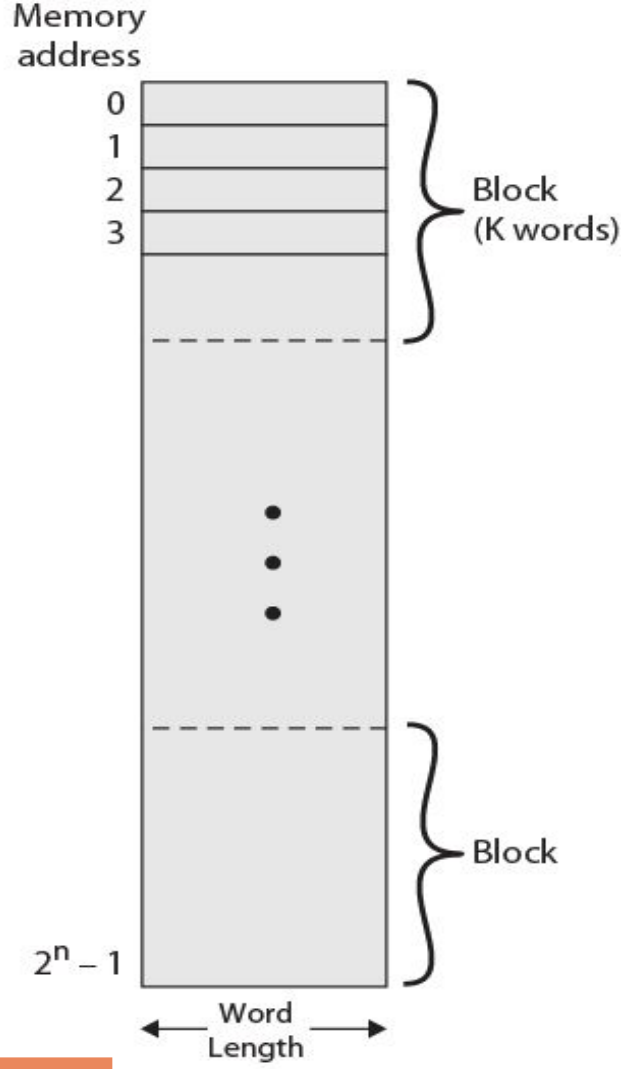
∴ No. of Cache Lines = 2^{10}

Line Size = 2^{14} B

Cache Size = $2^{10} \times 2^{14}$ B = $2^{(10+14)}$ B = 2^{24} B = $2^4 \times 2^{20}$ B = 16 MB

Cache/Main Memory Structure





(b) Main memory

Main Memory	2^n addressable words
Each word	Unique n – bit address
Mapping Purpose Considering	Number of fixed length blocks - K words - each

Cache Memory Consist	M blocks (M= 2^n/K) blocks
Blocks called Cache Lines	Each Line => K words