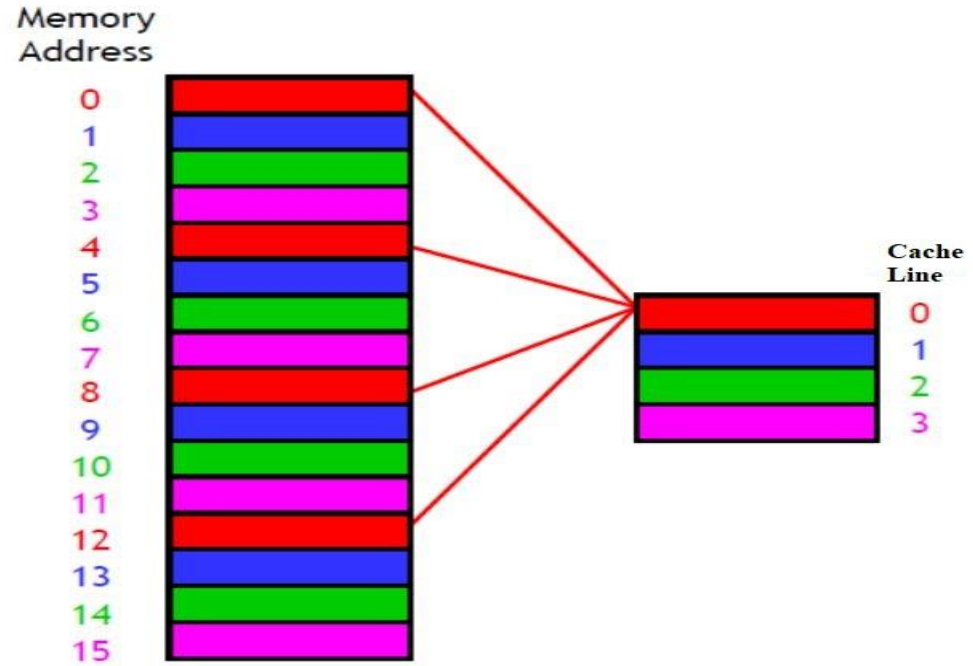
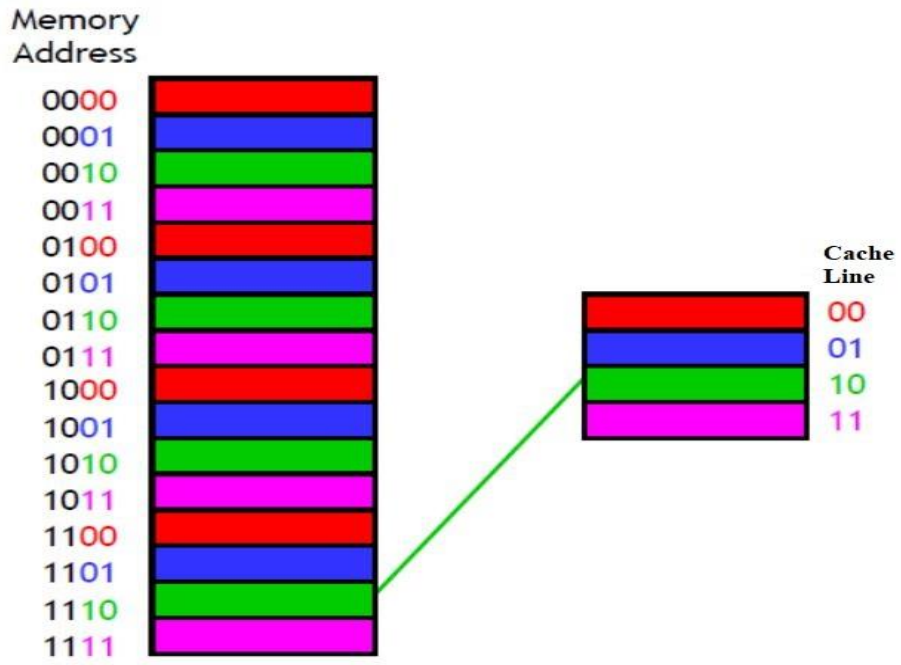
Direct Mapping

- An example: There is a 16 byte memory and a 4 byte cache (four 1-byte blocks, 2^2 blocks).
  - Here, memory addresses 0, 4, 8, and 12 are mapped into cache line 0 (block 0).
  - Similarly, addresses 2, 6, 10, and 14 are mapped into which block????



Direct Mapping

- Another point can be considered.
  - ➢To find out the right cache block, look at the least significant k bits of the memory address.
  - ➢For a 4 byte cache k=2 (2^2 blocks).
  - ➢So, address 14 (1110) would be mapped into cache block 2 (10).
  - ➢So, address 9 (1001) would be mapped into which cache block ??



Direct Mapping

- One question arise here; How to know whether the required data is already stored in the cache or not?
  - By applying the modulus technique, it is possible to know a required data would be stored in which cache block.
  - But other memory addresses could be mapped into that same cache block. For example, cache block 2 can contain data from memory addresses 2, 6, 10, and 14. How to distinguish between them?
  - To solve this, it is required to add **tag bits** to the cache. The rest of the memory address bits are considered as tag bits, that allows to distinguish between different memory addresses that map into the same cache block.

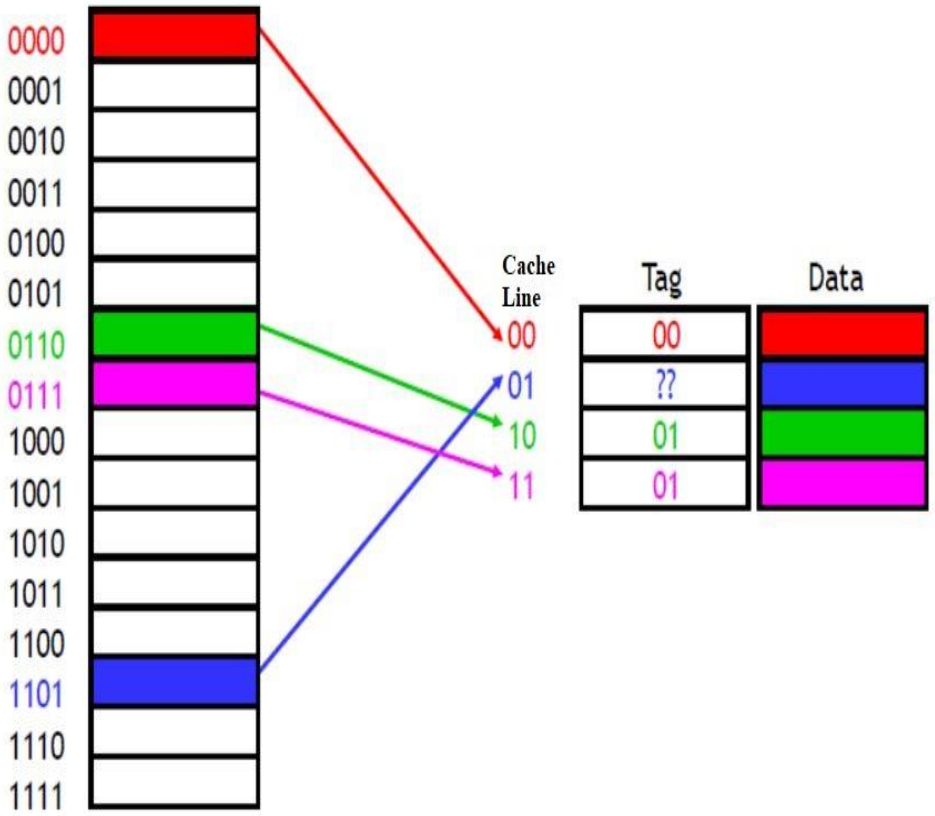
Direct Mapping

- Now it is clear, exactly which memory address is mapped into which cache block by concatenating the cache block tag with the cache line.

tag	cache line		memory address
00	00	=	0000
11	01	=	1101

…….

…….



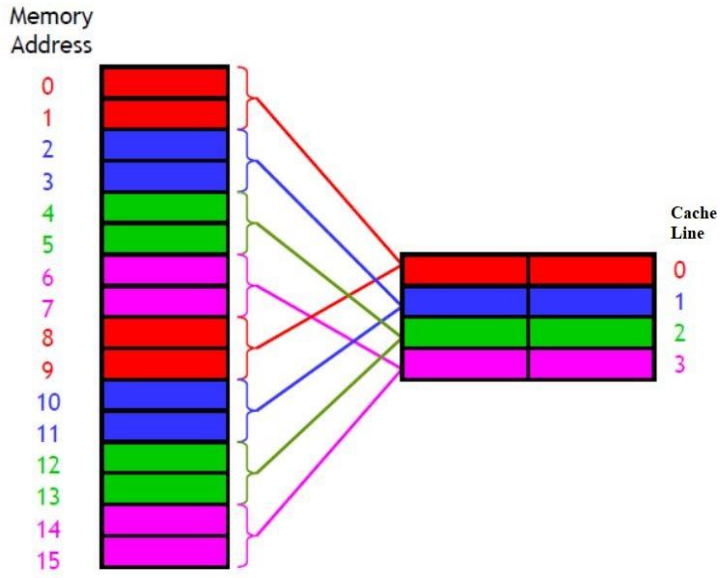
Direct Mapping

- **Question**: For a byte-addressable machine with 16-bit addresses with a cache with the following characteristics:
  - ✓It is direct-mapped
  - ✓Each block holds one byte
  - ✓The cache index is the four least significant bits

  So, How many blocks does the cache hold?

Direct Mapping

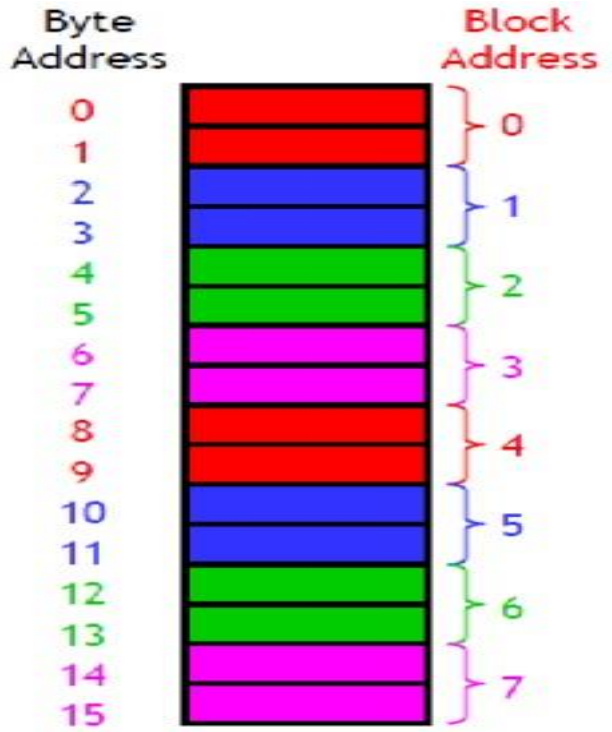
- With one byte cache blocks, the cache miss rate is high. One-byte cache blocks don't take advantage of **spatial locality**, which predicts that an access to one address will be followed by an access to a nearby address.
- To solve this, larger cache blocks can be used.
- Here, in a two byte cache blocks, two bytes will be loaded into the cache at a time.
- If memory address 12 is read from the memory, the data in addresses 12 and 13 will be loaded in the cache block 2.



Direct Mapping

- Now, we have to calculate the memory **block address** to map into the cache. For example, memory addresses 12 and 13 both correspond to block address 6 as follows:

$12 / 2^1 = 6$ and $13 / 2^1 = 6$, as the cache has two byte blocks and each block has 2^1 byte



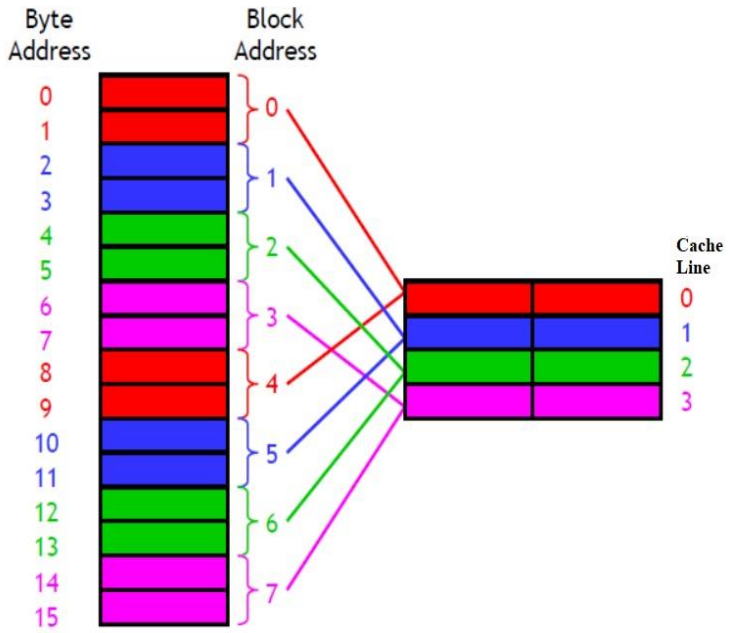
Direct Mapping

- Once the memory block address is determined, now it can be mapped into cache as before.
- For example, memory blocks 0 and 4 would be mapped into cache line/block 0 since,

  0 modulus 4=0 and

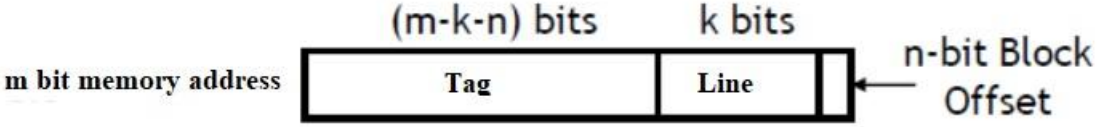
  4 modulus 4 =0

- To access one byte of data in memory, the entire *block would be mapped* into the cache, taking the advantage of spatial locality. That means, to access memory address 6, the entire block 3 (both addresses 6 and 7) would be mapped into the cache block 3 (6 in first block, and 7 in the second block).

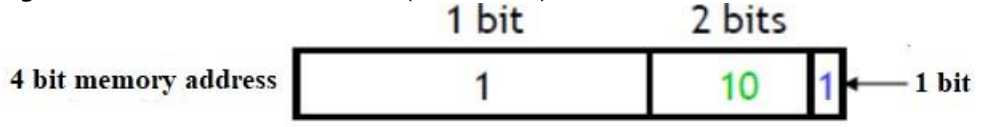


Direct Mapping

- Suppose, there is a cache with 2^k blocks, each containing 2^n bytes.
- So, to determine where a byte of data is in this cache by looking at its memory address. Hence, in a m bit memory address,
  - The *k* bits of the address will select one of the 2^k cache lines/blocks.
  - The lowest *n* bits are now a block offset that decides which of the 2^n bytes in the cache block will store the data.
  - The rest bits are used as tag bits (*m-k-n*).



  - The cache, for example used, has 2^2 blocks with 2^1 bytes in each block. Thus, the data in memory address 13 (1101) would be stored in byte 1 of cache line/block 2.

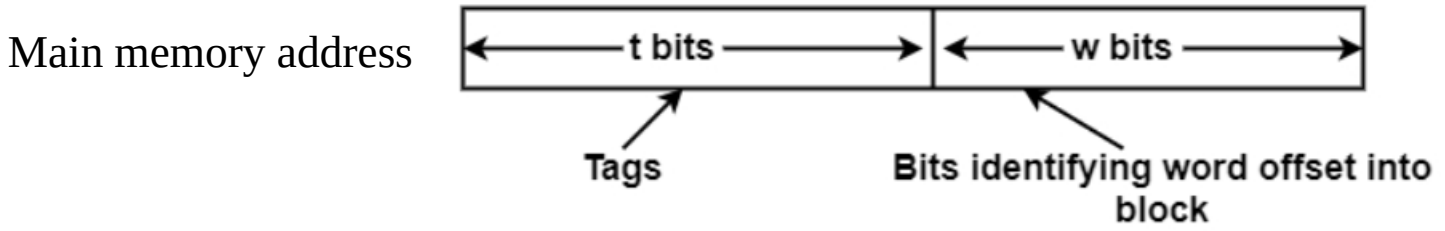


Associative Mapping

- The main **disadvantage** of direct mapping is that there is a fixed cache location for each memory address.
- So, if it happens that a program requires to access data from two different memory addresses repeatedly that map into the same cache line/block. For example, both memory addresses 2 and 6 are mapped into the same cache line/block 2. So, what happens if a program uses addresses 2, 6, 2, 6, 2, …repeatedly?
- Then, the memory blocks will be swapped into the cache. The hit rate will be low. This is known as **thrashing**.
- The associative mapping can overcome this disadvantage; thus increases the cache hit rate. **In associative mapping**, data in a memory address can be stored in any cache line/block, instead of forcing each memory address into one particular line/block.
  - When data is fetched from memory, it can be placed in any unused block of the cache.
  - There will never have a conflict between two or more memory addresses which map into the same cache block.

Associative Mapping

- In this mapping, the memory address is interpreted as a **tag** and **word** bits.
  - Tag uniquely identifies a block of main memory.
  - To determine whether a block is in the cache, the tag is pulled from the memory address and a search is performed through all of the lines of the cache to see if the block is present.



See example 2b

Associative Mapping

- This method of searching for a block within a cache appears like it might be a slow process, but it is not.
- **Each line** of the cache has its **compare circuitry**, which can quickly analyze whether or not the block is contained at that line. With all of the lines performing this **comparison process** in **parallel**, the correct line is identified **quickly**.
- This method is also known as **fully associative mapping**.
- The principal **disadvantage** of associative mapping is the **complex circuitry** required to examine the tags of all cache lines in parallel.

Set Associative Mapping

- This mapping is a combination of the strengths/advantages of the both the direct mapping and associative mapping while reducing their advantages.
  - In this case, the cache consists of a number of sets, each of the sets consists of a number of cache line.
  - This mapping is expressed as

    $m = v \times k$

    $i = j \text{ modulo } v$

    Where,

    i= cache set number

    J= main memory block number

    m= number of lines in the cache

    v= number of sets

    k= number of lines in each set

Set Associative Mapping

- In this way, a block B_j can be mapped into any lines in cache set j. That means a memory block can be mapped into **any cache lines** in a **specific set**.

See Example 2c