# MAPPING FUNCTIONS

- There are several possible methods for determining where memory blocks are placed in the cache.
- It is instructive to describe these methods using a specific small example.

- Consider a cache consisting of 128 blocks of 16 words each, for a total of 2048 (2K) words, and assume that the main memory is addressable by a 16-bit address.
- The main memory has 64K words, which we will view as 4K blocks of 16 words each.
- For simplicity, we have assumed that consecutive addresses refer to consecutive words.

# MAPPING FUNCTIONS

[Hamacher, Vranesic & Zaky, "Computer Organization" (5th Ed), McGraw Hill]

## Direct Mapping

- The simplest way to determine cache locations in which to store memory blocks is the direct-mapping technique.
- In this technique, block j of the main memory maps onto block j modulo 128 of the cache.

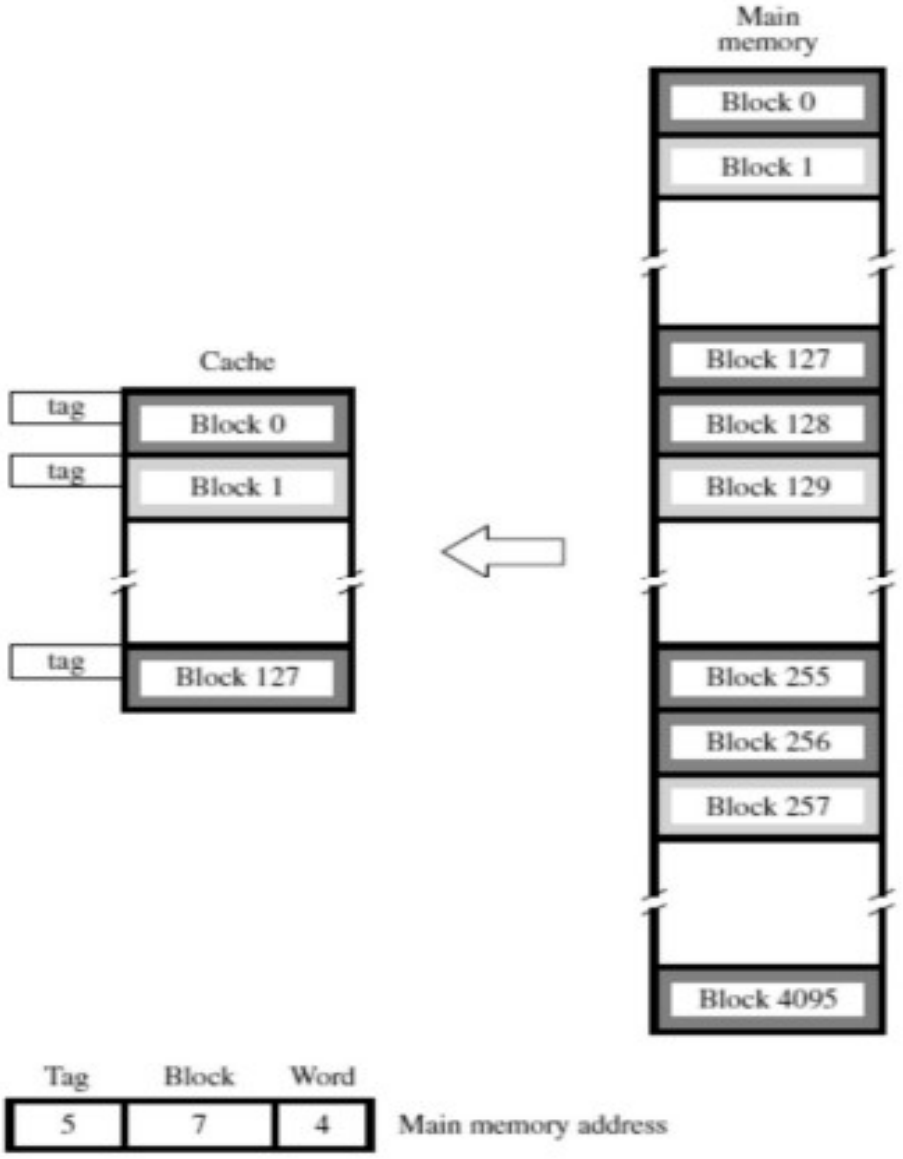



Figure 12: Direct-mapped cache

# MAPPING FUNCTIONS

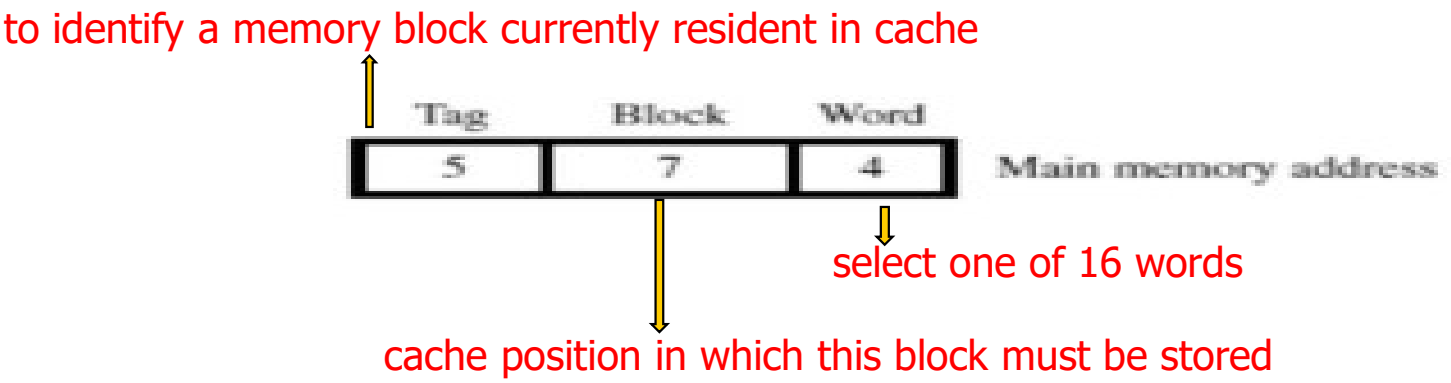
## Direct Mapping

- Thus, whenever one of the main memory blocks 0, 128, 256, . . . is loaded into the cache, it is stored in cache block 0.
- Blocks 1, 129, 257, . . . are stored in cache block 1, and so on.
- Since more than one memory block is mapped onto a given cache block position, contention may arise for that position even when the cache is not full.
- For example, instructions of a program may start in block 1 and continue in block 129, possibly after a branch. As this program is executed, both of these blocks must be transferred to the block-1 position in the cache.
- Contention is resolved by allowing the new block to overwrite the currently resident block.
- With direct mapping, the replacement algorithm is trivial.
- Placement of a block in the cache is determined by its memory address.

# MAPPING FUNCTIONS

## Direct Mapping

- The memory address can be divided into three fields.
  - The low-order 4 bits select one of 16 words in a block.
  - When a new block enters the cache, the 7-bit cache block field determines the cache position in which this block must be stored.
  - The high-order 5 bits of the memory address of the block are stored in 5 tag bits associated with its location in the cache.
  - The tag bits identify which of the 32 main memory blocks mapped into this cache position is currently resident in the cache.



# MAPPING FUNCTIONS

Direct Mapping

- As execution proceeds, the 7-bit cache block field of each address generated by the processor points to a particular block location in the cache.
- The high-order 5 bits of the address are compared with the tag bits associated with that cache location.
- If they match, then the desired word is in that block of the cache.
- If there is no match, then the block containing the required word must first be read from the main memory and loaded into the cache.
- The direct-mapping technique is easy to implement, but it is not very flexible.

# MAPPING FUNCTIONS

[Hamacher, Vranesic & Zaky, "Computer Organization" (5th Ed), McGraw Hill]

## Associative Mapping

- Figure 13 shows the most flexible mapping method, in which a main memory block can be placed into any cache block position.
- In this case, 12 tag bits are required to identify a memory block when it is resident in the cache.
- The tag bits of an address received from the processor are compared to the tag bits of each block of the cache to see if the desired block is present. This is called the associative-mapping technique.

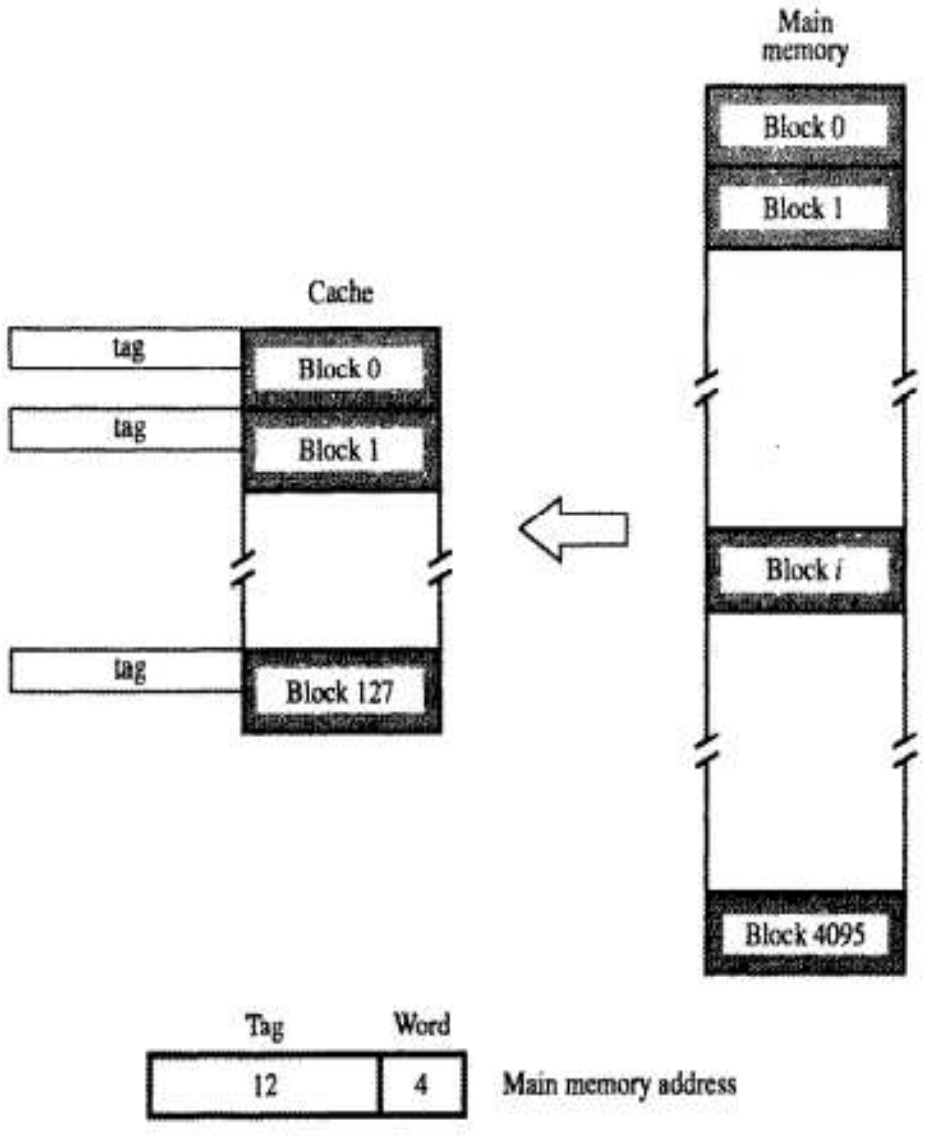



Figure 13 Associative-mapped cache.

# MAPPING FUNCTIONS

Associative Mapping

- It gives complete freedom in choosing the cache location in which to place the memory block, resulting in a more efficient use of the space in the cache.
- When a new block is brought into the cache, it replaces (ejects) an existing block only if the cache is full.
- In this case, we need an algorithm to select the block to be replaced. Many replacement algorithms are possible.
- The complexity of an associative cache is higher than that of a direct-mapped cache, because of the need to search all 128 tag patterns to determine whether a given block is in the cache. A search of this kind is called an associative search.
- To avoid a long delay, the tags must be searched in parallel.

# MAPPING FUNCTIONS

## Set-Associative Mapping

- Another approach is to use a combination of the direct- and associative-mapping techniques.
- The blocks of the cache are grouped into sets, and the mapping allows a block of the main memory to reside in any block of a specific set.
- Hence, the contention problem of the direct method is eased by having a few choices for block placement.
- At the same time, the hardware cost is reduced by decreasing the size of the associative search.

# MAPPING FUNCTIONS

[Hamacher, Vranesic & Zaky, "Computer Organization" (5th Ed), McGraw Hill]

## Set-Associative Mapping

- An example of the set-associative-mapping technique is shown in Figure 14 for a cache with two blocks per set.
- In this case, memory blocks 0, 64, 128, . . . , 4032 map into cache set 0, and they can occupy either of the two block positions within this set.

[Block j of the main memory maps onto block j modulo 64]

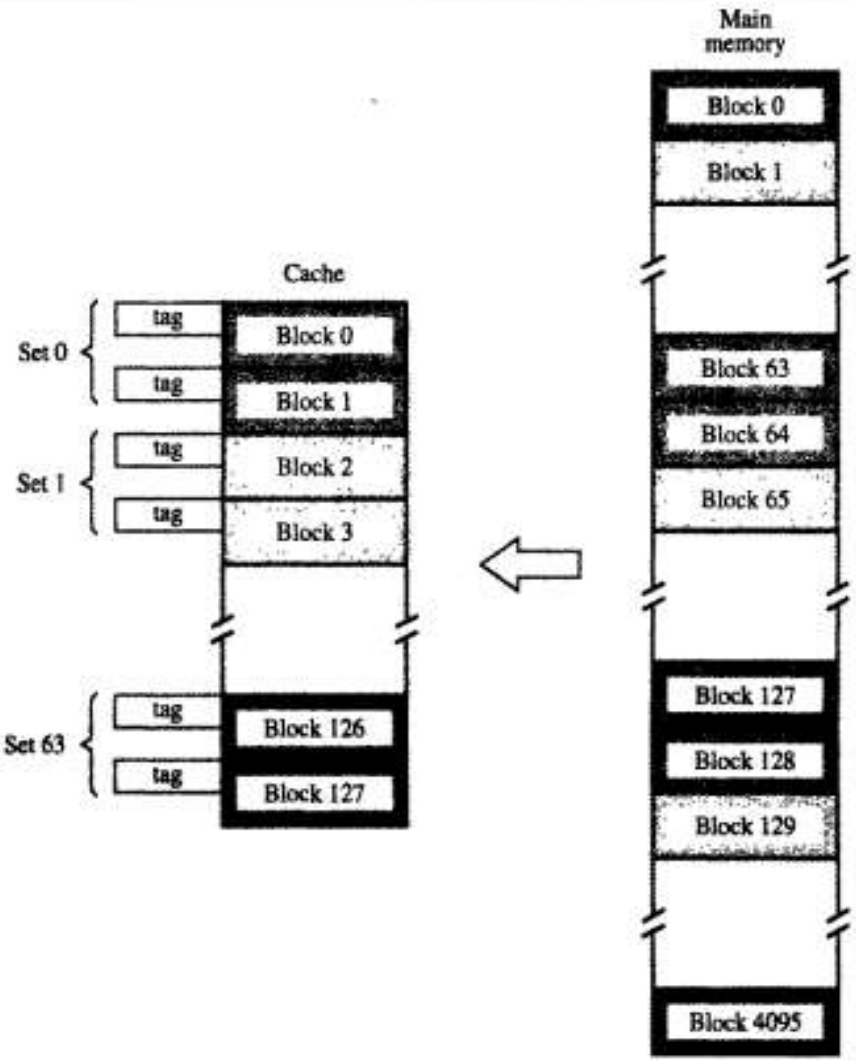



Figure 14 Set-associative-mapped cache with two blocks per set.

# MAPPING FUNCTIONS

[Hamacher, Vranesic & Zaky, "Computer Organization" (5th Ed), McGraw Hill]

## Set-Associative Mapping

- Having 64 sets means that the 6-bit set field ($2^6$=64 sets), of the address determines which set of the cache might contain the desired block.
- The tag field of the address must then be associatively compared to the tags of the two blocks of the set to check if the desired block is present.
- This two-way associative search is simple to implement.

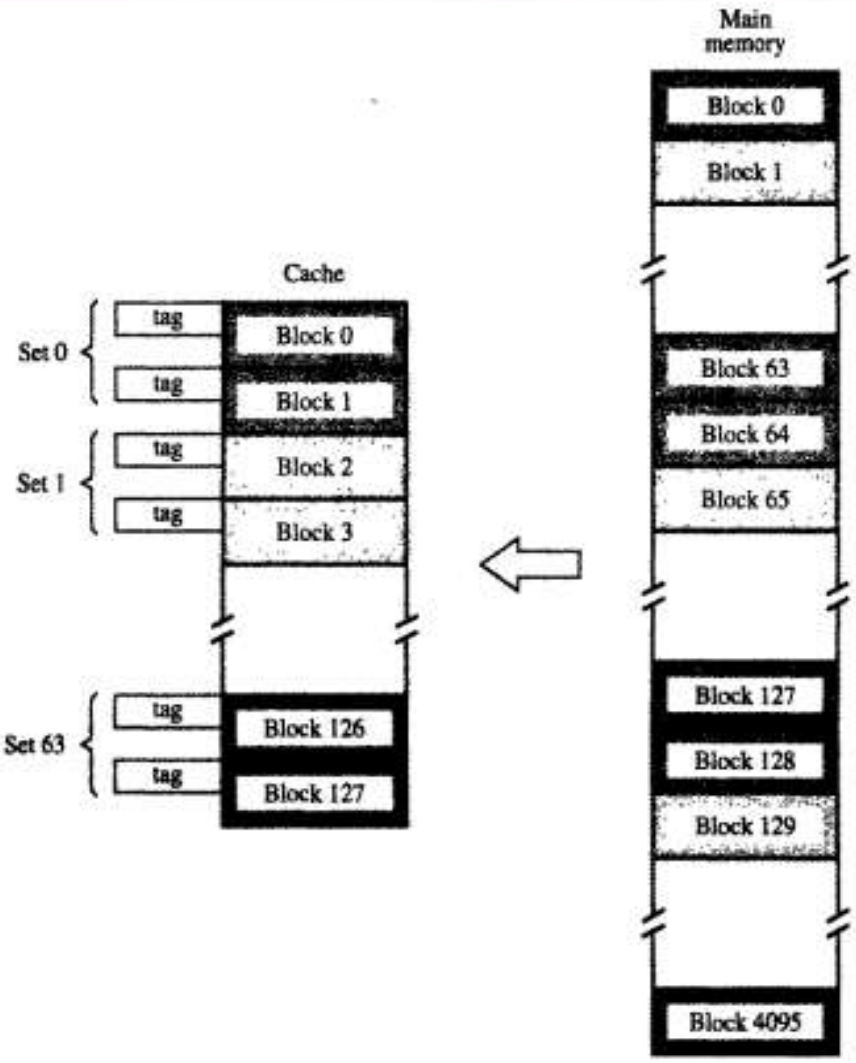



Figure 14 Set-associative-mapped cache with two blocks per set.

# MAPPING FUNCTIONS

Set-Associative Mapping

- The number of blocks per set is a parameter that can be selected to suit the requirements of a particular computer.
- For the main memory and cache sizes in Figure 14, four blocks per set can be accommodated by a 5-bit set field($2^5$=32 sets), eight blocks per set by a 4-bit set field ($2^4$=16 sets) and so on.
- The extreme condition of 128 blocks per set requires no set bits (set=0) and corresponds to the fully- associative technique, with 12 tag bits.
- The other extreme of one block per set is the direct mapping method.
- A cache that has k blocks per set is referred to as a k-way set-associative cache.

# MAPPING FUNCTIONS

## Set-Associative Mapping

- When power is first turned on, the cache contains no valid data.
- A control bit, called the valid bit, must be provided for each cache block to indicate whether the data in that block are valid.
- This bit should not be confused with the modified, or dirty, bit.
- The valid bits of all cache blocks are set to 0 when power is initially applied to the system.
- Some valid bits may also be set to 0 when new programs or data are loaded from the disk into the main memory.

# MAPPING FUNCTIONS

## Set-Associative Mapping

- Data transferred from the disk to the main memory using the DMA mechanism are usually loaded directly into the main memory, bypassing the cache.
- As program execution proceeds, the valid bit of a given cache block is set to 1 when a memory block is loaded into that location from the main memory.
- The processor fetches data from a cache block only if its valid bit is equal to 1.
- The use of the valid bit in this manner ensures that the processor will not fetch stale data from the cache.

# MAPPING FUNCTIONS

## Set-Associative Mapping

- A similar precaution is needed in a system that uses the write-back protocol.
- Under this protocol, new data written into the cache are not written to the memory at the same time.
- Hence, data in the memory do not always reflect the changes that may have been made in the cached copy.
- It is important to ensure that such stale data in the memory are not transferred. One solution is to flush the cache, by forcing all dirty blocks to be written back to the memory before the DMA transfer takes place.

# MAPPING FUNCTIONS

## Set-Associative Mapping

- The operating system can do this by issuing a command to the cache before initiating the DMA operation that transfers the data to the disk.
- Flushing the cache does not affect performance greatly, because such disk transfers do not occur often.
- This need to ensure that two different entities (the processor and the DMA subsystems) use same copies of the data is referred to as a cache-coherence problem.

# REPLACEMENT ALGORITHMS

- In a direct-mapped cache, the position of each block is predetermined by its address; hence, the replacement strategy is trivial.
- In associative and set-associative caches there exists some flexibility.
- When a new block is to be brought into the cache and all the positions that it may occupy are full, the cache controller must decide which of the old blocks to overwrite.
- This is an important issue, because the decision can be a strong determining factor in system performance.

# REPLACEMENT ALGORITHMS

- The objective is to keep blocks in the cache that are likely to be referenced in the near future.
- But, it is not easy to determine which blocks are about to be referenced.
- The property of locality of reference in programs gives a clue to a reasonable strategy.
- Because program execution usually stays in localized areas for reasonable periods of time, there is a high probability that the blocks that have been referenced recently will be referenced again soon.
- Therefore, when a block is to be overwritten, it is sensible to overwrite the one that has gone the longest time without being referenced. This block is called the least recently used (LRU) block, and the technique is called the LRU replacement algorithm.

# REPLACEMENT ALGORITHMS

- To use the LRU algorithm, the cache controller must track references to all blocks as computation proceeds.
- Suppose it is required to track the LRU block of a four-block set in a set-associative cache.
- A 2-bit counter can be used for each block.
- When a hit occurs, the counter of the block that is referenced is set to 0.
- Counters with values originally lower than the referenced one are incremented by one, and all others remain unchanged.

# REPLACEMENT ALGORITHMS

- When a miss occurs and the set is not full, the counter associated with the new block loaded from the main memory is set to 0, and the values of all other counters are increased by one.
- When a miss occurs and the set is full, the block with the counter value 3 is removed, the new block is put in its place, and its counter is set to 0.
- The other three block counters are incremented by one. It can be easily verified that the counter values of occupied blocks are always distinct.

# REPLACEMENT ALGORITHMS

- The LRU algorithm has been used extensively.
- Although it performs well for many access patterns, it can lead to poor performance in some cases.
- For example, it produces disappointing results when accesses are made to sequential elements of an array that is slightly too large to fit into the cache.
- Performance of the LRU algorithm can be improved by introducing a small amount of randomness in deciding which block to replace.

# REPLACEMENT ALGORITHMS

- Other replacement algorithms are also used in practice.
- An intuitively reasonable rule would be to remove the "oldest" block from a full set when a new block must be brought in.
- However, because this algorithm does not take into account the recent pattern of access to blocks in the cache, it is generally not as effective as the LRU algorithm in choosing the best blocks to remove.
- The simplest algorithm is to randomly choose the block to be overwritten.
- Interestingly enough, this simple algorithm has been found to be quite effective in practice.