CACHE MEMORY

Mechanism of Cache Memory

Cache memory owes its introduction to Wilkes back in 1965. At that time, Wilkes distinguished between two types of main memory: The conventional and the slave memory. In Wilkes terminology, a slave memory is a second level of unconventional high-speed memory, which nowadays corresponds to what is called cache memory (the term cache means a safe place for hiding or storing things).The idea behind using a cache as the first level of the memory hierarchy is to keep the information expected to be used more frequently by the CPU in the cache (a small high-speed memory that is near the CPU). The end result is that at any given time some active portion of the main memory is duplicated in the cache. Therefore, when the processor makes a request for a memory reference, the request is first sought in the cache. If the request corresponds to an element that is currently residing in the cache, we call that a cache hit. On the other hand, if the request corresponds to an element that is not currently in the cache, we call that a cache miss. A cache hit ratio, h, is defined as the probability of finding the requested element in the cache. A cache miss ratio (1 - h) is defined as the probability of not finding the requested element in the cache. In the case that the requested element is not found in the cache, then it has to be brought from a subsequent memory level in the memory hierarchy. Assuming that the element exists in the next memory level, that is, the main memory, then it has to be brought and placed in the cache. In expectation that the next requested element will be residing in the neighboring locality of the current requested element (spatial locality), then upon a cache miss what is actually brought to the main memory is a block of elements that contains the requested element. The advantage of transferring a block from the main memory to the cache will be most visible if it could be possible to transfer such a block using one main memory access time. Such a possibility could be achieved by increasing the rate at which information can be transferred between the main memory and the cache.

• A cache memory is a small, very fast memory that retains copies of recently used information from main memory. It operates transparently

to the programmer, automatically deciding which values. to keep and which to overwrite.

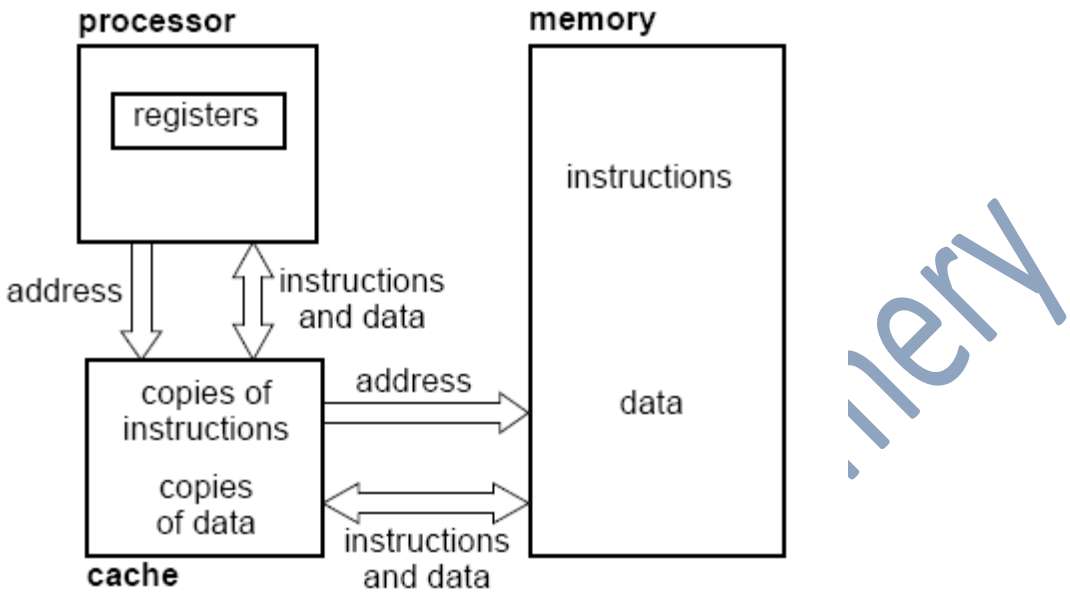

Fig 6.1 cache memory in computer system

• The processor operates at its high clock rate only when the memory items it requires are held in the cache. The overall system performance depends strongly on the proportion of the memory accesses which can be satisfied by the cache

• An access to an item which is in the cache: **hit**
An access to an item which is not in the cache: **miss**.
The proportion of all memory accesses that are satisfied by the cache: **hit rate**
The proportion of all memory accesses that are not satisfied by the cache: **miss rate**

• The miss rate of a well-designed cache: few %

During execution of a program, memory references by the processor, for both instructions and data, tend to cluster: once an area of the program is entered, there are repeated references to a small set of instructions (loop, subroutine) and data (components of a data structure, local variables or parameters on the stack).

Temporal locality (locality in time): If an item is referenced, it will tend to be referenced again soon**(e.g., loops, reuse)**.
Spatial locality (locality in space): If an item is referenced, items whose addresses are close by will tend to be referenced soon(e.g., straightline code, array access).

Separate Data and Instruction Caches

• The previous figure(6.1) shows an architecture with a unified instruction and data cache.
• It is common also to split the cache into one dedicated to instructions and one dedicated to data.

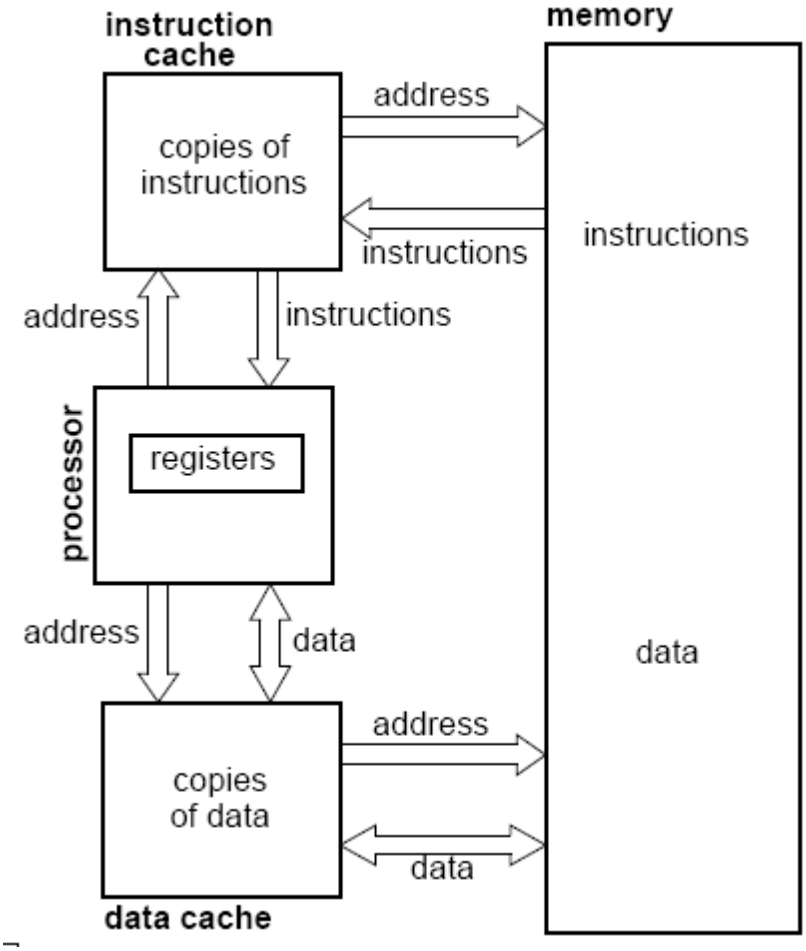

Fig 6.2 separate data and instruction cache memory

Separate Data and Instruction Caches

• Advantages of unified caches:
- they are able to better balance the load between instruction and data fetches depending on the dynamics of the program execution;

- design and implementation are cheaper.
• Advantages of split caches (Harvard Architectures)
- competition for the cache between instruction processing and execution units is eliminated instruction fetch can proceed in parallel with memory access from the execution unit.

Cache Memory Organization

There are three main different organization techniques used for cache memory. The three techniques are discussed below. These techniques differ in two main aspects:

1. The criterion used to place, in the cache, an incoming block from the main memory.
2. The criterion used to replace a cache block by an incoming block (on cache full).

Direct Mapping

This is the simplest among the three techniques. Its simplicity stems from the fact that it places an incoming main memory block into a specific fixed cache block location. The placement is done based on a fixed relation between the incoming block number, i, the cache block number, j, and the number of cache blocks, N:

$j = i \bmod N$

Example 1 Consider, for example, the case of a main memory consisting of 4K blocks, a cache memory consisting of 128 blocks, and a block size of 16 words.

Figure 6.3 shows the division of the main memory and the cache according to the direct-mapped cache technique. As the figure shows, there are a total of 32 main memory blocks that map to a given cache block. For example, main memory blocks 0, 128, 256, 384, . . . ,3968 map to cache block 0. We therefore call the direct-mapping technique a

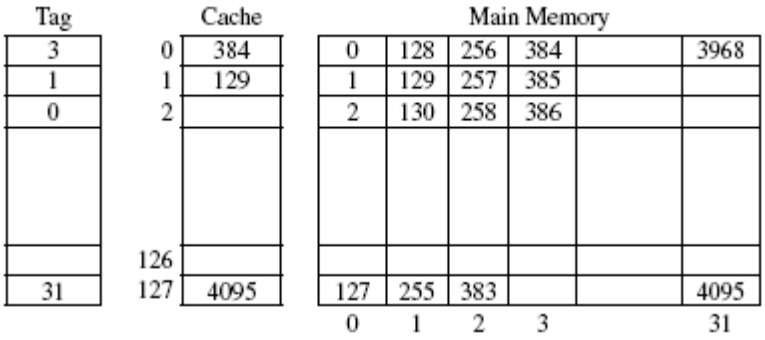

Fig 6.3.Mapping main memory blocks to cache blocks

many-to-one mapping technique. The main advantage of the direct-mapping technique is its simplicity in determining where to place an incoming main memory block in the cache. Its main disadvantage is the inefficient use of the cache. This is because according to this technique, a number of main memory blocks may compete for a given cache block even if there exist other empty cache blocks. This disadvantage should lead to achieving a low cache hit ratio. According to the direct-mapping technique the MMU interprets the address issued by the processor by dividing the address into three fields as shown in Figure 6.4. The lengths, in bits, of each of the fields are:

1. Word field = log2 B, where B is the size of the block in words.
2. Block field = log2 N, where N is the size of the cache in blocks.
3. Tag field = log2 (M/N), where M is the size of the main memory in blocks.
4. The number of bits in the main memory address = log2 (B * M)

It should be noted that the total number of bits as computed by the first three equations should add up to the length of the main memory address. This can be used as a check for the correctness of your computation.

Exampl2:Compute the above four parameters for Example1

$$\text{Word field} = \log_2 B = \log_2 16 = \log_2 2^4 = 4 \text{ bits}$$
$$\text{Block field} = \log_2 N = \log_2 128 = \log_2 2^7 = 7 \text{ bits}$$
$$\text{Tag field} = \log_2(M/N) = \log_2(2^2 \times 2^{10}/2^7) = 5 \text{ bits}$$

$$\text{The number of bits in the main memory address} = \log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16 \text{ bits.}$$

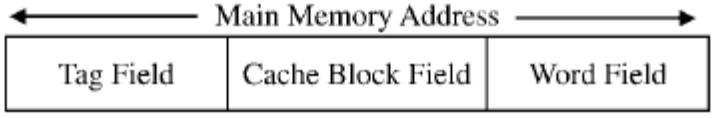

Fig 6.4 Direct-mapped address fields

Having shown the division of the main memory address, we can now proceed to explain the protocol used by the MMU to satisfy a request made by the processor for accessing a given element. We illustrate the protocol using the parameters given in the example :

The steps of the protocol are:

1. Use the Block field to determine the cache block that should contain the element requested by the processor. The Block field is used directly to determine the cache block sought, hence the name of the technique: direct-mapping.
2. Check the corresponding Tag memory to see whether there is a match between its content and that of the Tag field. A match between the two indicates that the targeted cache block determined in step 1 is currently holding the main memory element requested by the processor, that is, a cache hit.
3. Among the elements contained in the cache block, the targeted element can be selected using the Word field.
4. If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory, deposited in the cache, and the targeted element is made available to the processor. The cache Tag memory and the cache block memory have to be updated accordingly.

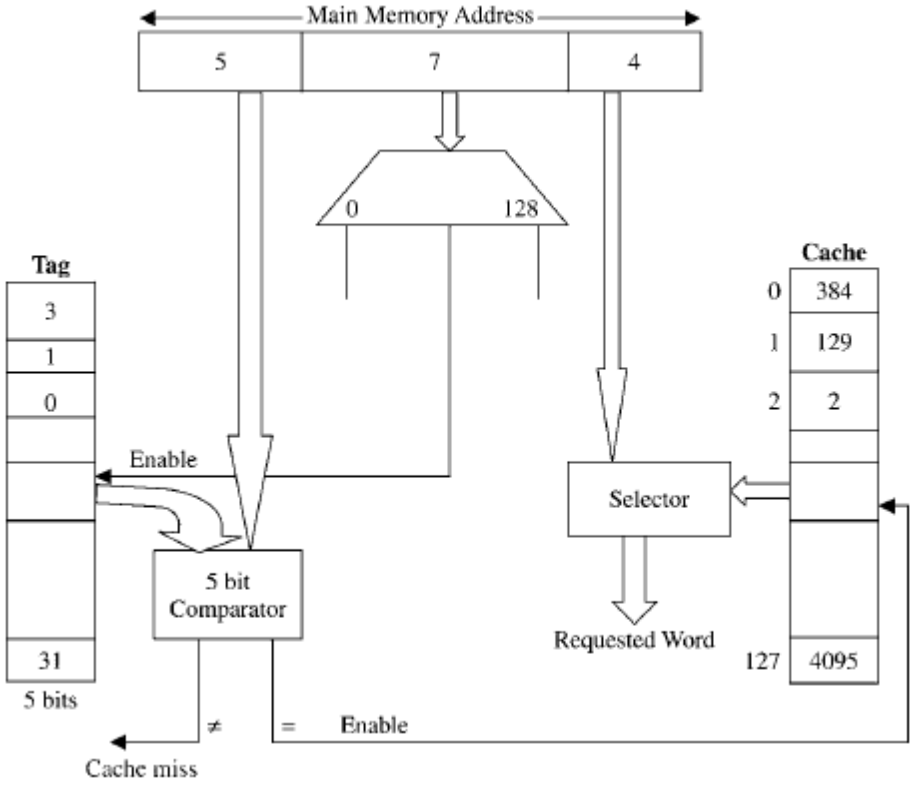

Fig 6.5 Direct-mapped address translation

The direct-mapping technique answers not only the placement of the incoming main memory block in the cache question, but it also answers the replacement question. Upon encountering a totally filled cache while a new main memory block has to be
brought, the replacement is trivial and determined by the equation
$j = i \bmod N$.
The main advantages of the direct-mapping technique is its simplicity measured in terms of the direct determination of the cache block; no search is needed. It is also simple in terms of the replacement mechanism. The main disadvantage of the technique
is its expected poor utilization of the cache memory. This is represented in terms of the possibility of targeting a given cache block, which requires frequent replacement of blocks while the rest of the cache is not used. Consider, for example, the sequence of requests made by the processor for elements held in the main
memory blocks 1, 33, 65, 97, 129, and 161. Consider also that the cache size is 32 blocks. It is clear that all the above blocks map to cache block

number 1. Therefore, these blocks will compete for that same cache block despite the fact that the remaining 31 cache blocks are not used. The expected poor utilization of the cache by the direct mapping technique is mainly due to the restriction on the placement of the incoming main memory blocks in the cache (the many-to-one property). If such a restriction is relaxed, that is, if we make it possible for an incoming main memory block to be placed
in any empty (available) cache block, then the resulting technique would be so flexible that efficient utilization of the cache would be possible. Such a flexible technique, called the Associative Mapping technique, is explained next. Fully Associative Mapping According to this technique, an incoming main memory block can be placed in any available cache block. Therefore, the address issued by the processor need only have two fields. These are the Tag and Word
fields. The first uniquely identifies the block while residing in the cache. The second field identifies the element within the block that is requested by the processor. The MMU interprets the address issued by the processor by dividing it into two fields as shown in Figure 6.6. The length, in bits, of each of the fields are given by:

1. Word field = log2 B, where B is the size of the block in words
2. Tag field = log2 M, where M is the size of the main memory in blocks
3. The number of bits in the main memory address = log2 (B* M)

It should be noted that the total number of bits as computed by the first two equations should add up to the length of the main memory address. This can be used as a check for the correctness of your computation.

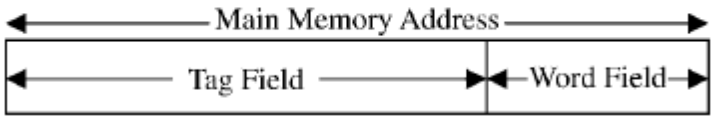

Fig 6.6 Associative-mapped address fields

Example 3 Compute the above three parameters for a memory system having the following specification: size of the main memory is 4K blocks, size of the cache is 128 blocks, and the block size is 16 words. Assume that the system uses associative
mapping.

$$\text{Word field} = \log_2 B = \log_2 16 = \log_2 2^4 = 4 \text{ bits}$$
$$\text{Tag field} = \log_2 M = \log_2 2^7 \times 2^{10} = 12 \text{ bits}$$

The number of bits in the main memory address $= \log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ bits.

Having shown the division of the main memory address, we can now proceed to explain the protocol used by the MMU to satisfy a request made by the processor for accessing a given element. We illustrate the protocol using the parameters given in the example presented above (see Fig. 6.7).

The steps of the protocol are:

1. Use the Tag field to search in the Tag memory for a match with any of the tags stored.
2. A match in the tag memory indicates that the corresponding targeted cache block determined in step 1 is currently holding the main memory element requested by the processor, that is, a cache hit.

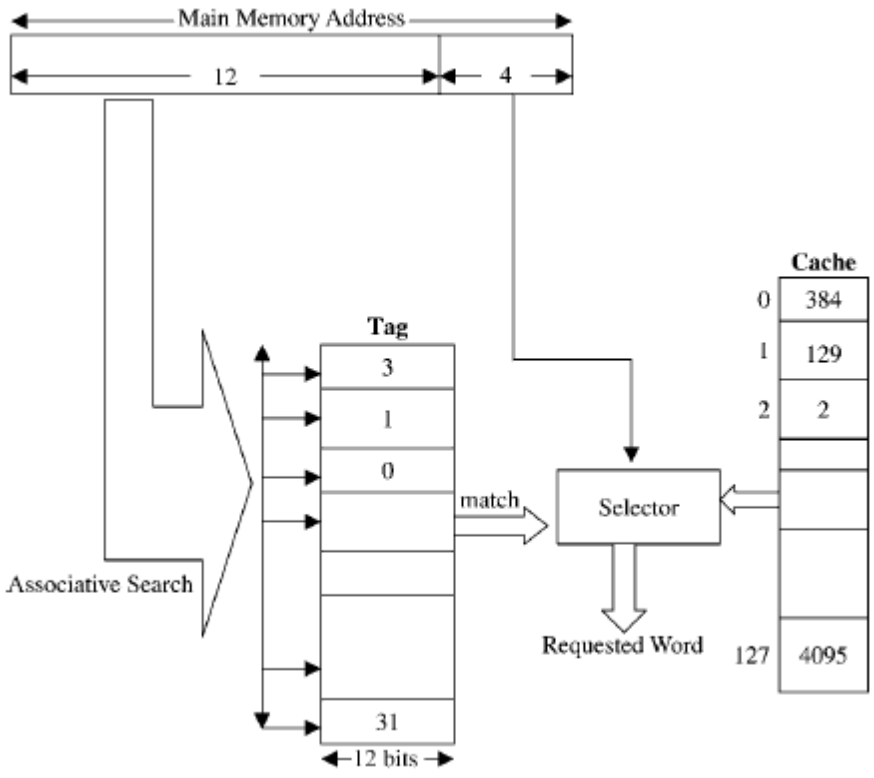

Fig 6.7 Associative-mapped address translation

3. Among the elements contained in the cache block, the targeted element can be selected using the Word field.
4. If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory,

deposited in the first available cache block, and the targeted element (word) is made available to the processor. The cache Tag memory and the cache block memory have to be updated accordingly.

It should be noted that the search made in step 1 above requires matching the tag field of the address with each and every entry in the tag memory. Such a search, if done sequentially, could lead to a long delay. Therefore, the tags are stored in an associative (content addressable) memory. This allows the entire contents of the tag memory to be searched in parallel (associatively), hence the name, associative mapping.

It should be noted that, regardless of the cache organization used, a mechanism is needed to ensure that any accessed cache block contains valid information. The validity of the information in a cache block can be checked via the use of a single bit for each cache block, called the valid bit. The valid bit of a cache block should be updated in such a way that if valid bit = 1, then the corresponding cache block carries valid information; otherwise, the information in the cache block is invalid. When a computer system is powered up, all valid bits are made equal to 0, indicating that they carry invalid information. As blocks are brought to the cache, their statuses are changed accordingly to indicate the validity of the information contained.

The main advantage of the associative-mapping technique is the efficient use of the cache. This stems from the fact that there exists no restriction on where to place incoming main memory blocks. Any unoccupied cache block can potentially be used to receive those incoming main memory blocks. The main disadvantage of the technique, however, is the hardware overhead required to perform the associative search conducted in order to find a match between the tag field and the tag memory as discussed above.

A compromise between the simple but inefficient direct cache organization and the involved but efficient associative cache organization can be achieved by conducting the search over a limited set of cache blocks while knowing ahead of time where in the cache an incoming main memory block is to be placed. This is the basis for the set-associative mapping technique explained next.

Set-Associative Mapping

In the set-associative mapping technique, the cache is divided into a number of sets. Each set consists of a number of blocks. A given main memory block maps to a specific cache set based on the equation:
s = i mod S,
where S is the number of sets in the cache, i is the main memory block number, and s is the specific cache set to which block i maps. However, an incoming block maps to any block in the assigned cache set.
Therefore, the address issued by the processor is divided into three distinct fields. These are the Tag, Set, and Word fields. The Set field is used to uniquely identify the specific cache set that ideally should hold the targeted block. The Tag field uniquely identifies the targeted block within the determined set. The Word field identifies the element (word) within the block that is requested by the processor. According to the set-associative mapping technique, the MMU interprets the address issued by the processor by dividing it into three fields as shown in Figure 6.8. The length, in bits, of each of the fields is given by

1. Word field = log2 B, where B is the size of the block in words
2. Set field = log2 S, where S is the number of sets in the cache
3. Tag field = log2 (M/S), where M is the size of the main memory in blocks.
S = N/Bs, where N is the number of cache blocks and Bs is the number of blocks per set
4. The number of bits in the main memory address = log2 (B * M)
It should be noted that the total number of bits as computed by the first three equations should add up to the length of the main memory address. This can be used as a check for the correctness of your computation.

Example 4 Compute the above three parameters (Word, Set, and Tag) for a memory system having the following specification: size of the main memory is 4K blocks, size of the cache is 128 blocks, and the block size is 16 words. Assume that the system uses set-associative mapping with four blocks per set.
S =128/4= 32 sets:

1. Word field $= \log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits
2. Set field $= \log_2 32 = 5$ bits
3. Tag field $= \log_2 (4 \times 2^{10}/32) = 7$ bits

The number of bits in the main memory address $= \log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ bits.

Having shown the division of the main memory address, we can now proceed to explain the protocol used by the MMU to satisfy a request made by the processor for accessing a given element. We illustrate the protocol using the parameters given in the example presented above (see Fig. 6.9). The steps of the protocol are:

1. Use the Set field (5 bits) to determine (directly) the specified set (1 of the 32 sets).

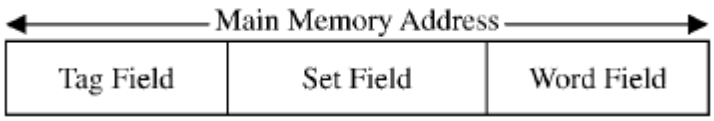

Fig 6.8 Set-associative-mapped address fields

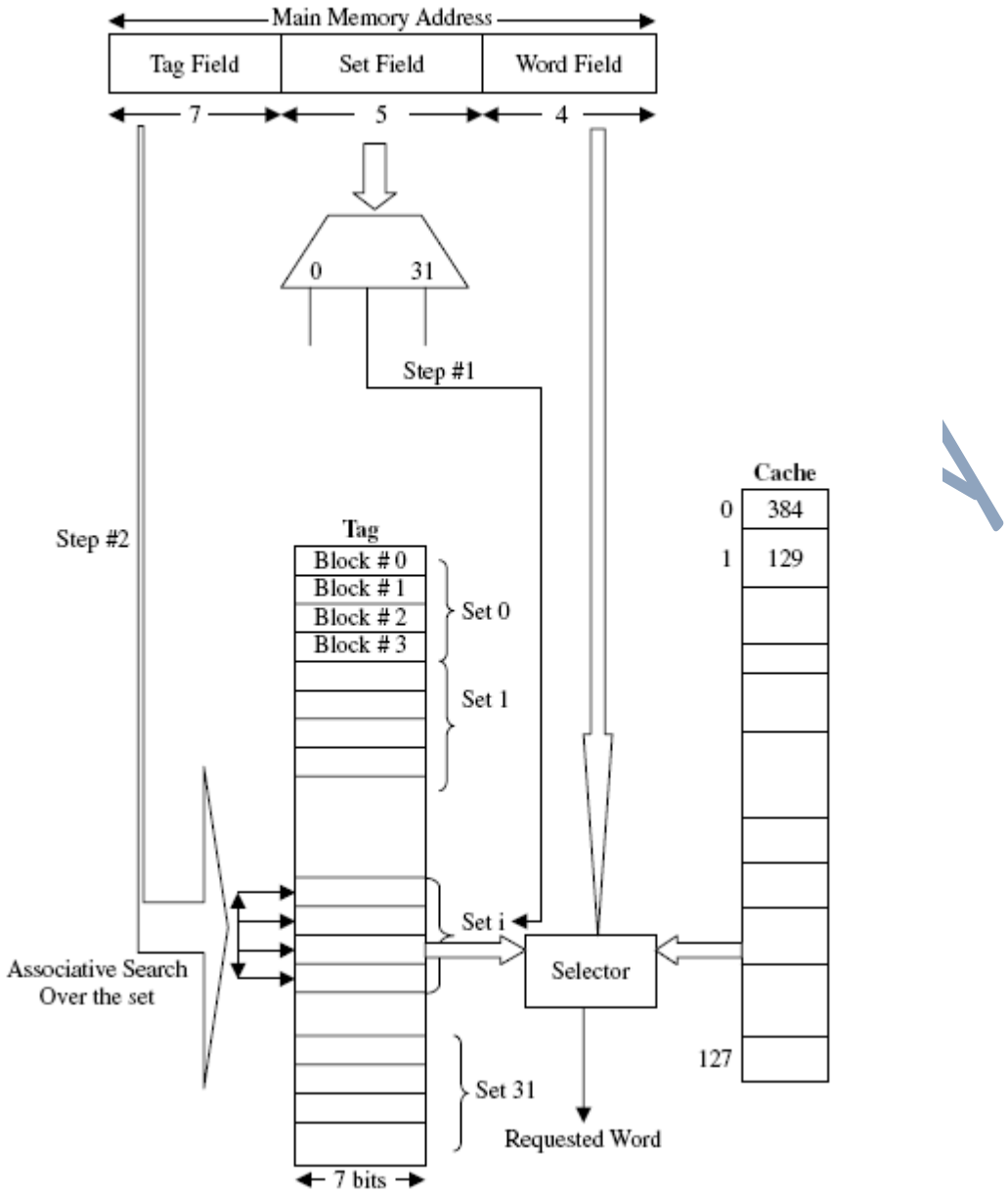

Fig 6.9 Set-associative-mapped address translation

2. Use the Tag field to find a match with any of the (four) blocks in the determined set. A match in the tag memory indicates that the specified set determined in step 1 is currently holding the targeted block, that is, a cache hit.

3. Among the 16 words (elements) contained in hit cache block, the requested word is selected using a selector with the help of the Word field.

4. If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory, deposited in the specified set first, and the targeted element (word) is made available to the processor. The cache Tag memory and the cache block memory have to be updated accordingly. It should be noted that

the search made in step 2 above requires matching the tag field of the address with each and every entry in the tag memory for the specified set. Such a search is performed in parallel (associatively) over the set, hence the name, set-associative mapping. The hardware overhead required to performing the associative search within a set in order to find a match between the tag field and the tag memory is not as complex as that used in the case of the fully associative technique.

Some Cache Architectures

Intel 80486
- a single on-chip cache of 8 Kbytes
- line size: 16 bytes
- 4-way set associative organization

Pentium
- two on-chip caches, for data and instructions.
- each cache: 8 Kbytes
- line size: 32 bytes
- 2-way set associative organization

Write Strategies

The problem: How to keep cache content and the content of main memory consistent without losing too much performance?

Problems arise when a write is issued to a memory address, and the content of the respective address is potentially changed.

• *Write-through*

All write operations are passed to main memory; if the addressed location is currently hold in the cache, the cache is updated so that it is coherent
with the main memory. For writes, the processor always slows down to main memory speed.

• *Write-back(Copy-back)*

Write operations update only the cache memory which is not kept coherent with main memory; cache lines have to remember if they have been updated; if such a line is replaced from the cache,

its content has to be copied back to memory. good performance (usually several writes are performed on a cache line before it is replaced and has to be copied into main memory), complex hardware.

Cache coherence problems are very complex and difficult to solve in multiprocessor systems.