COA

Ch3-P2

Cache memory

By

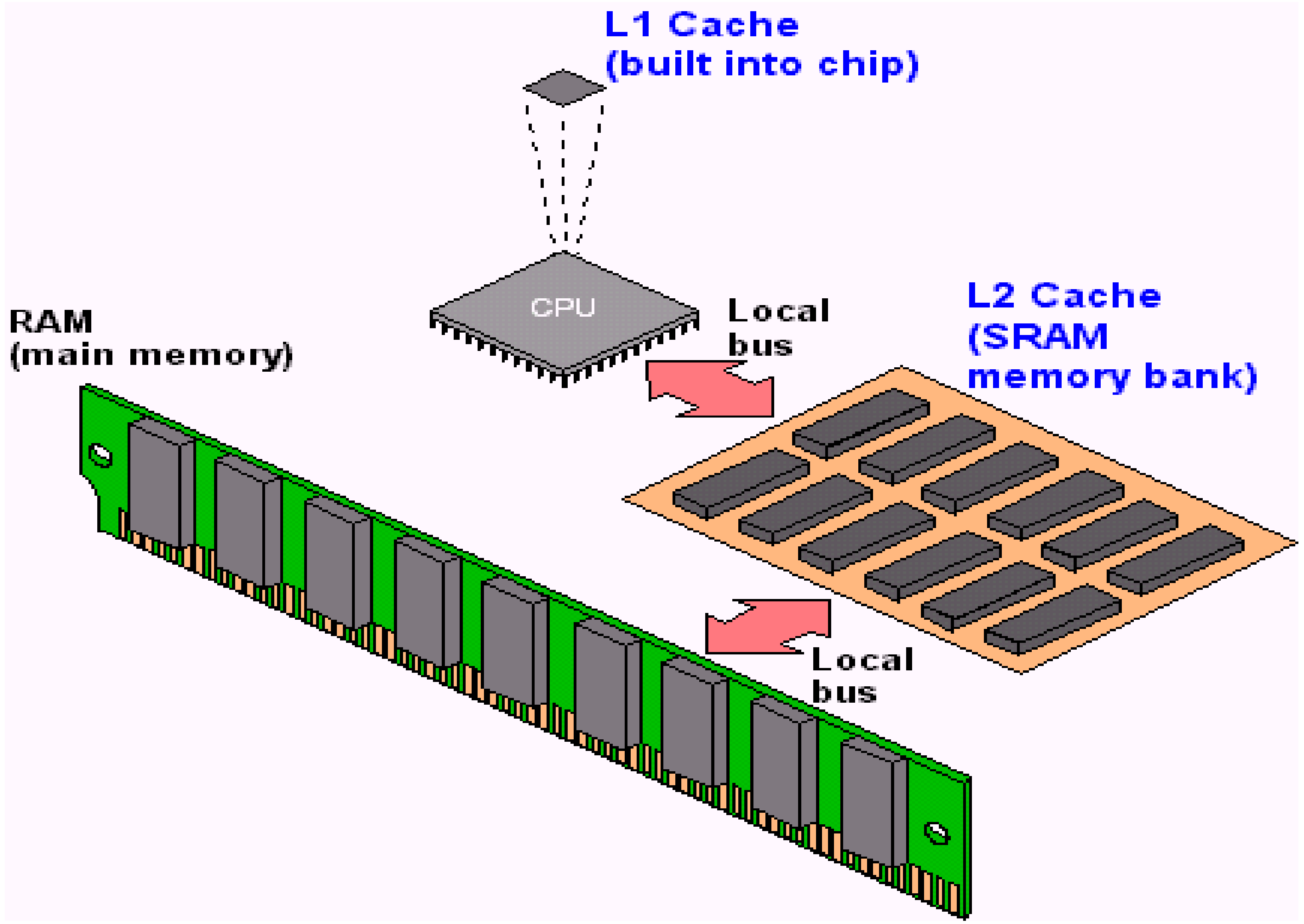
Dr.Raghad Samir Al Najim

2017@SPU

Introduction

- A **CPU cache** is a cache used by the central processing unit of computer to reduce the average time to access memory.

- The cache is a smaller, faster memory which <u>stores copies</u> of the data from the most frequently used main memory locations.

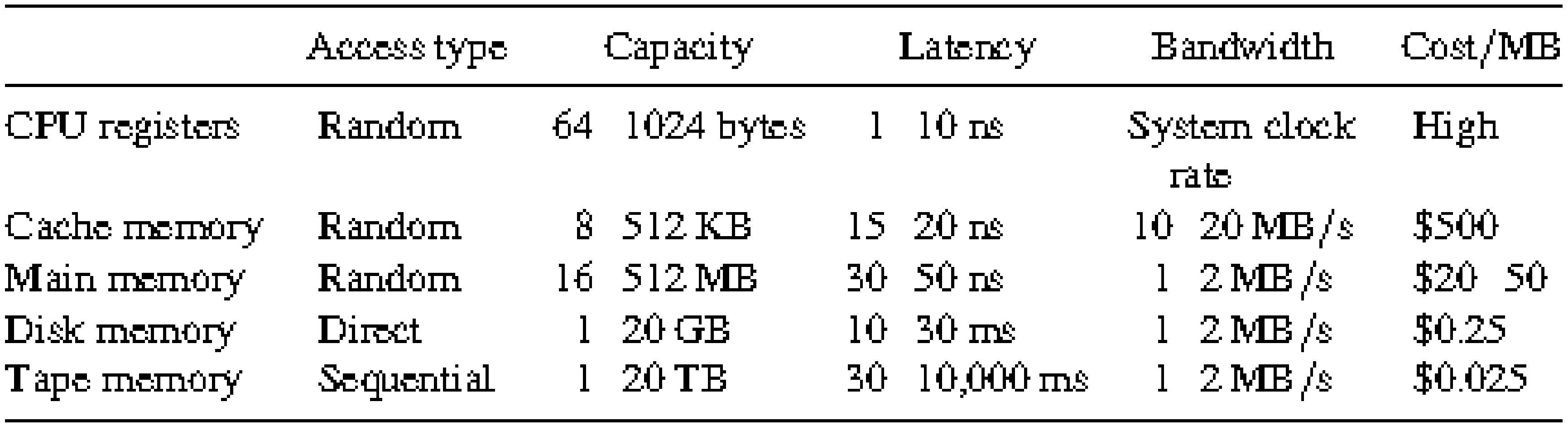


- L1 and L2 are levels of cache memory in a computer. If the computer processor can find the data it needs for its next operation in cache memory, it will save time compared to having to get it from random access memory.
- L1 is "level-1" cache memory, usually built onto the microprocessor chip itself. The L1 cache is a kind of memory and it is the place where the CPU first try to access.
- L2 caches may be built the same way as the L1 caches, into the CPU but sometimes it can also be located in another chip or it can be a completely separate chip.
- With some exceptions, L1 and L2 caches are considered SRAM (static RAM) while the memory of the computer is considered DRAM (Dynamic Ram).

- ❖ The size of cache memory can vary in its size.
- ❖ A typical personal computer’s level 2 (L2) cache is 256K or 512K. Level 1 (L1) cache is smaller, typically 8K or 16K.
- ❖ L1 cache resides(تستقر) on the processor, whereas L2 cache resides between the CPU and main memory.
- ❖ L1 cache is, therefore, faster than L2 cache.
- ❖ Some processors use another cache level named L3.

The memory hierarchy can be characterized by a number of parameters.

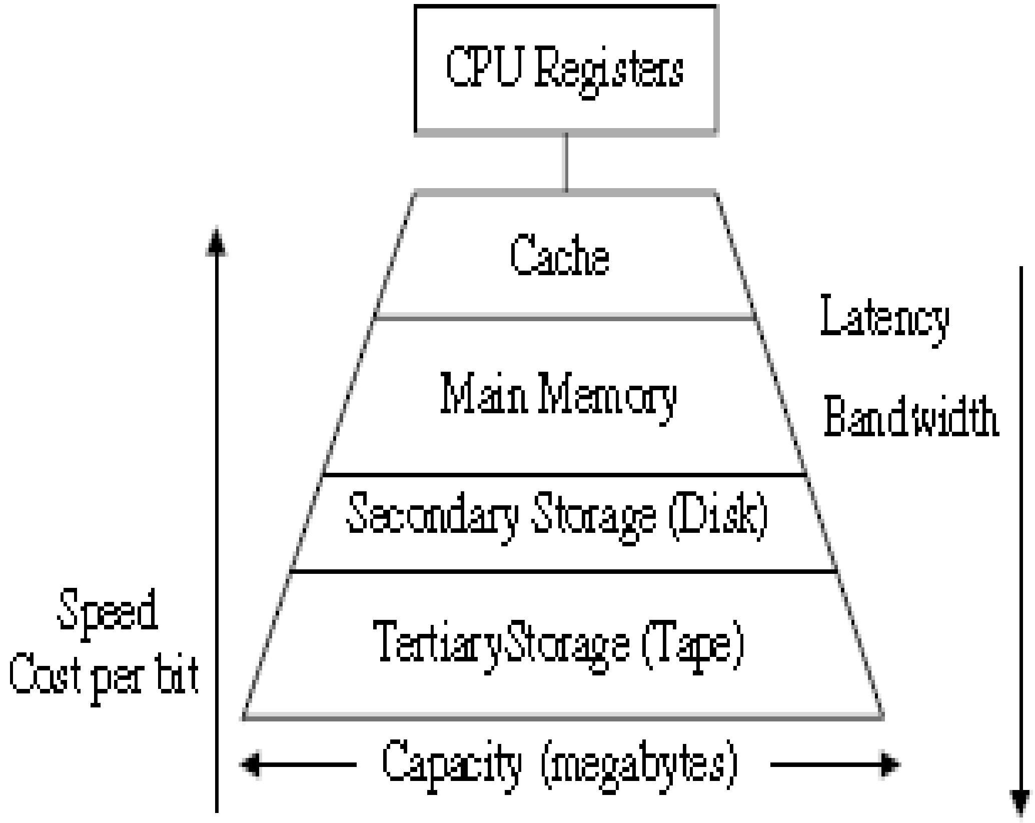
- The term access refers to the action that physically takes place during a read or write operation.
- The capacity of a memory level is usually measured in bytes.
- The latency is defined as the time interval between the request for information and the access to the first bit of that information.
- The bandwidth provides a measure of the number of bits per second that can be accessed.
- The cost of a memory level is usually specified as dollars per megabytes



	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64 1024 bytes	1 10 ns	System clock rate	High
Cache memory	Random	8 512 KB	15 20 ns	10 20 MB/s	$500
Main memory	Random	16 512 MB	30 50 ns	1 2 MB/s	$20 50
Disk memory	Direct	1 20 GB	10 30 ms	1 2 MB/s	$0.25
Tape memory	Sequential	1 20 TB	30 10,000 ms	1 2 MB/s	$0.025

The memory hierarchy can be characterized by a number of parameters.

- The term access refers to the action that physically takes place during a read or write operation.
- The capacity of a memory level is usually measured in bytes.
- The latency is defined as the time interval between the request for information & the access to the first bit of that information.
- The bandwidth provides a measure of the number of bits per second that can be accessed.
- The cost of a memory level is usually specified as dollars per megabytes



The effectiveness of a memory hierarchy depends on the principle of moving information into the fast memory infrequently and accessing it many times before replacing it with new information.
This principle is possible due to a phenomenon called locality of reference; that is, within a given period of time, programs tend to reference a relatively confined area of memory repeatedly.
There exist two forms of locality: spatial and temporal locality.
Spatial locality refers to the phenomenon that when a given address has been referenced, it is most likely that addresses near it will be referenced within a short period of time, for example, consecutive instructions in a straight line program.

Temporal locality, on the other hand, refers to the phenomenon that once a particular memory item has been referenced, it is most likely that it will be referenced next, for example, an instruction in a program loop.

The sequence of events that takes place when the processor makes a request for an item is as follows:.

First, the item is sought in the first memory level of the memory hierarchy. The probability of finding the requested item in the first level is called the hit(hit ratio), h1. The probability of not finding (missing) the requested item in the first level of the memory hierarchy is called the miss ratio, (1-h1). When the requested item causes a “miss,” it is sought in the next memory level.

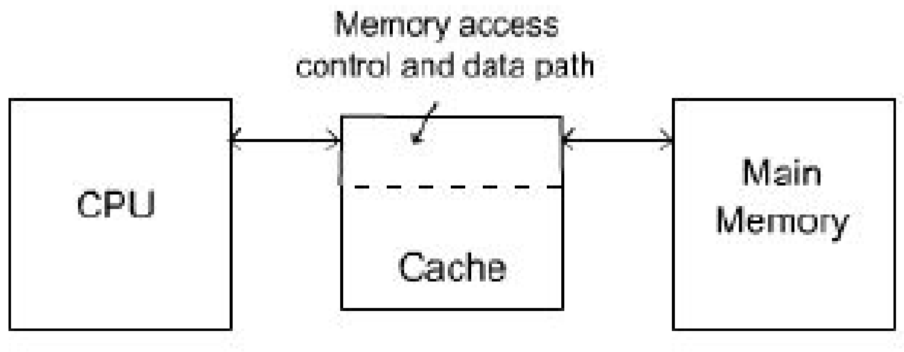
The probability of finding the requested item in the second memory level, the hit ratio of the second level, is h2. The miss ratio of the second memory level is (1 - h2). The process is repeated until the item is found.
when the requested item is found, it is brought and sent to the processor.

Cache Memory

Now, if it can be arranged to have the active segments of a program in a fast memory, then the tolal execution time can be reduced. It is the fact that CPU is a faster device and memory is a relatively slower device.
Memory access is the main bottleneck for the performance efficiency. If a faster memory device can be inserted between main memory and CPU, the efficiency can be increased. The faster memory that is inserted between CPU and Main Memory is termed as Cache memory.

To make this arrangement effective, the cache must be considerably faster than the main memory, and typically it is 5 to 10 time faster than the main memory.

It is the fact that CPU is a faster device and memory is a relatively slower device.



Cache memory between the CPU and the main memory

Some assumptions are made while designing the CPU - memory system:

1. **The CPU does not need to know explicitly about the existence of the cache.**
2. **The CPU simply makes Read and Write request. The nature of these two operations are same whether cache is present or not.**
3. **The address generated by the CPU always refer to location of main memory.**
4. **The memory access control circuitry(Memory management unit) determines whether or not the requested word currently exists in the cache.**

CPU Read request

When a Read request is received from the CPU, the contents of a block of memory words containing the location specified are transferred into the cache. When any of the locations in this block is referenced by the program, its contents are read directly from the cache.

The cache memory can store a number of such blocks at any given time.

The correspondence between the Main Memory Blocks and those in the cache is specified by means of a mapping function.

Note that:

we present cache-mapping function taking into consideration the interface between two successive levels in the memory hierarchy:

primary level & secondary level.

Cache & Main memory

If the focus is on the interface between the cache and main memory, then the cache represents the primary level, while the main memory represents the secondary level.

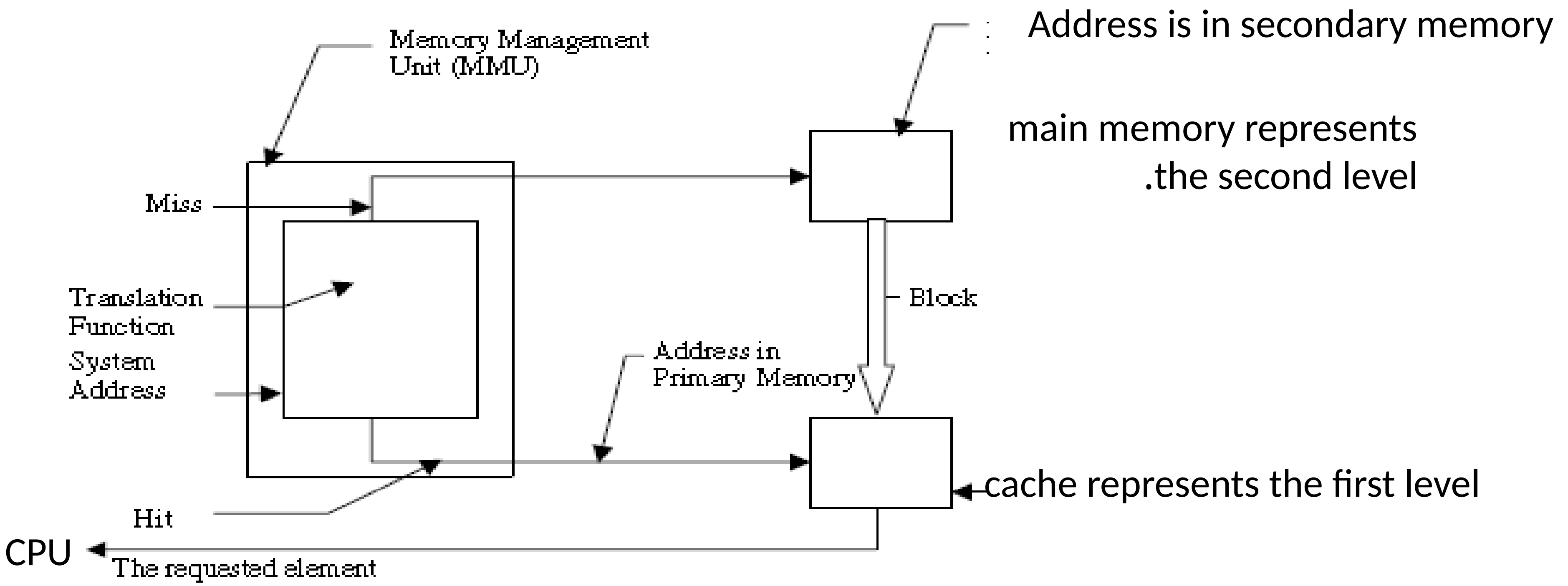
The same principles apply to the interface between any two memory levels in the hierarchy.

Memory Management Unit (MMU).

MMU: It should be noted that a request for accessing a memory element is made by the processor through the address of the requested element. The addressing by the processor may correspond to that of an element that exists currently in the cache (cache hit); otherwise, it may correspond to an element that is currently in the main memory.

Therefore, address translation has to be made in order to determine where to find the requested element. This is one of the main functions performed by the memory management unit (MMU).

if the element is not currently in the cache, then it will be brought (as part of a block) from the main memory and placed in the cache and the element requested is made available to the processor.


Memory Management Unit (MMU)
Address is in secondary memory
main memory represents .the second level
Miss
Translation Function
System Address
Block
Address in Primary Memory
cache represents the first level
Hit
CPU
The requested element

What makes Cache special

"special"? Cache is not accessed by address; it is *accessed by content.*

For this reason, cache is sometimes called content addressable memory or *CAM*.

Under most cache mapping schemes, the cache entries must be checked or searched to see if the value being requested is stored in cache. To simplify this process of locating the desired data, various cache mapping algorithms are used.

For cache to be functional, it must store useful data. However, this data becomes useless if the CPU can't find it. When accessing data or instructions, the CPU first generates a main memory address. If the data has been copied to cache, the address of the data in cache is not the same as the main memory address.

Cache Mapping Schemes

The CPU uses a specific mapping scheme that “converts” the main memory address into a cache location. This address conversion is done by giving special significance to the bits in the main memory address.

We first divide the bits into distinct groups we call fields. Depending on the mapping scheme, we may have two or three fields. How we use these fields depends on the particular mapping scheme being used.

Before we discuss these mapping schemes, it is important to understand how data is copied into cache. Main memory and cache are both divided into the same size blocks.

When a memory address is generated, cache is searched first to see if the required word exists there. When the requested word is not found in cache, the entire main memory block in which the word resides is loaded into cache.

- Memory locations(words) can be grouped into block.
- **Memory capacity usually measured in bits:**
  - **Total no. of memory locations(words) * size of memory word**
  - **Capacity= no. of blocks* no. of words in block. *no of bits in word**

- Ex: If a memory word is 8 bit and the size of a block is 8 words.
- What is the capacity of the main memory, if the total number of blocks in the memory is 128.
  - ✓ 128 blocks * 8 words * 8 bits = $2^7 * 2^3 * 2^3 = 2^{13}$ = 1K * 8 = 8 Kbit
- How many blocks in the main memory if the memory capacity is 32 Kbit.

  - Total number of words = 32 Kbit / 8 bit = $2^{15} / 2^3 = 2^{12}$ words
  - Total number of blocks = 2^{12} word / 8 words = $2^{12} / 2^3 = 2^9$ = 512 blocks

So, how do we use fields in the main memory address? One field of the main memory address points us to a location in cache in which the data resides if it is resident in cache (this is called a cache hit), or where it is to be placed if it is not resident (which is called a cache miss).
The cache block referenced is then checked to see if it is valid. This is done by a valid bit with each cache block.
A valid bit of 0 means the cache block is not valid (we have a cache miss) and we must access main memory.
A valid bit of 1 means it is valid (we may have a cache hit but we need to **complete** one more step before we know for sure).

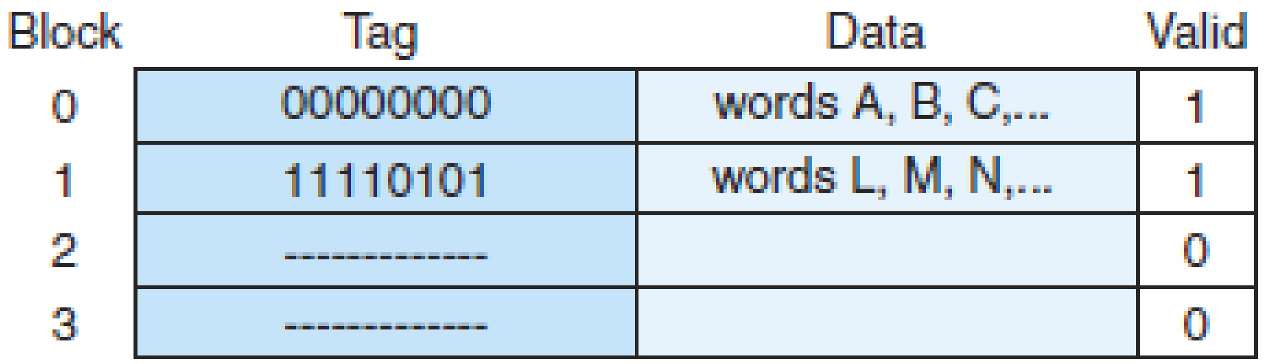
We then compare the tag in the cache block to the tag field of our address. If the tags are the same, then we have found the desired cache block (we have a cache hit). At this point we need to locate the desired word in the block; this can be done using a different portion of the main memory address called the word field.

Cache Dynamics

When a memory *read (or fetch)* is issued by the CPU:
If the line with that memory address is in the cache (this is called a cache *hit*), the data is read from the cache to the MDR.
If the line with that memory address is not in the cache (this is called a *miss*), the cache is updated by replacing one of its active lines by the line with that memory address, and then the data is read from the cache to the MDR.

When a memory *write* is issued by the CPU:
If the line with that memory address is in the cache, the data is written from the MDR to the cache, and the line is marked "invalid" (since it no longer is an image of the corresponding memory line).
If the line with that memory address is not in the cache, the cache is updated by replacing one of its active lines by the line with that memory address. The data is then written from the MDR to the cache and the line is marked "invalid."

Cache Memory Organization



Block	Tag	Data	Valid
0	00000000	words A, B, C,...	1
1	11110101	words L, M, N,...	1
2	-------------		0
3	-------------		0

A Closer Look at Cache

In this figure, there are two valid cache blocks. Block 0 contains multiple words from main memory, identified using the tag “00000000”. Block 1 contains words identified using tag “11110101”. The other two cache blocks are not valid.

The size of each field depends on the physical characteristics of main memory and cache. The word field - identifies a word from a specific block; therefore, it must contain the appropriate number of bits to do this. This is also true of the block field - it must select a unique block of cache. The tag field is whatever is left over.
When a block of main memory is copied to cache, this tag is stored with the block & identifies this block.

Cache Memory Organization

There are **three main different organization of Mapping 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).

Three organizations of Mapping techniques

Direct mapping

Fully Associative Mapping

Set-Associative Mapping

Direct Mapping

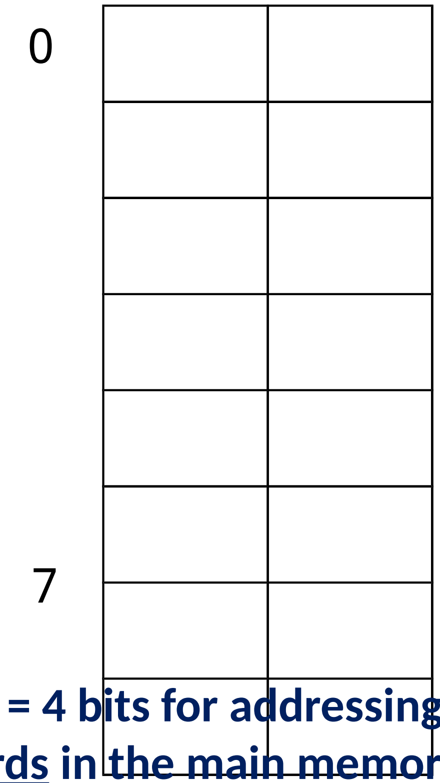
This is the simplest among the three techniques. Its simplicity stems from the fact that it places an incoming main memory block into a specific fixed cache block location.

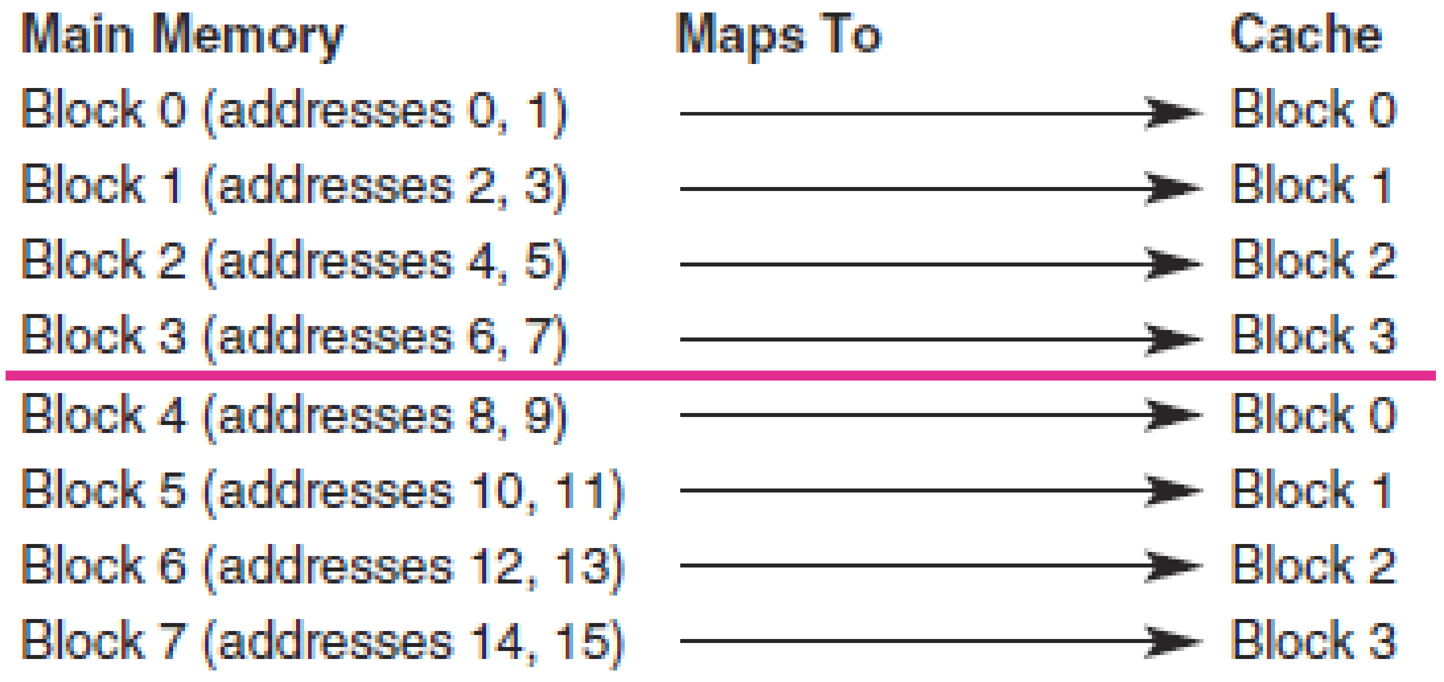
The placement is done based on a fixed relation between the incoming block number, i,
the cache block number, j,
and the number of cache blocks, N:

$$j = i \bmod N \quad ??$$

Suppose we have a system using direct mapping with 16 words of main memory divided into 8 blocks (so each block has 2 words).
Assume the cache is 4 blocks in size (for a total of 8 words). Table below shows how the main memory blocks map to cache.

Main memory





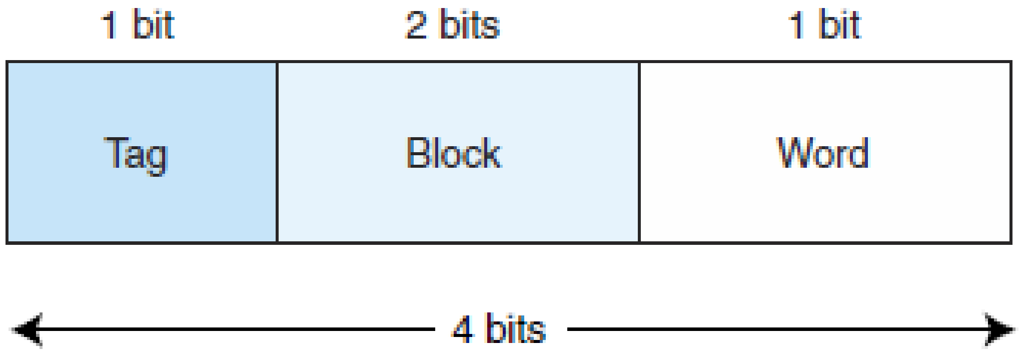
Main Memory	Maps To	Cache
Block 0 (addresses 0, 1)	$\longrightarrow$	Block 0
Block 1 (addresses 2, 3)	$\longrightarrow$	Block 1
Block 2 (addresses 4, 5)	$\longrightarrow$	Block 2
Block 3 (addresses 6, 7)	$\longrightarrow$	Block 3
Block 4 (addresses 8, 9)	$\longrightarrow$	Block 0
Block 5 (addresses 10, 11)	$\longrightarrow$	Block 1
Block 6 (addresses 12, 13)	$\longrightarrow$	Block 2
Block 7 (addresses 14, 15)	$\longrightarrow$	Block 3

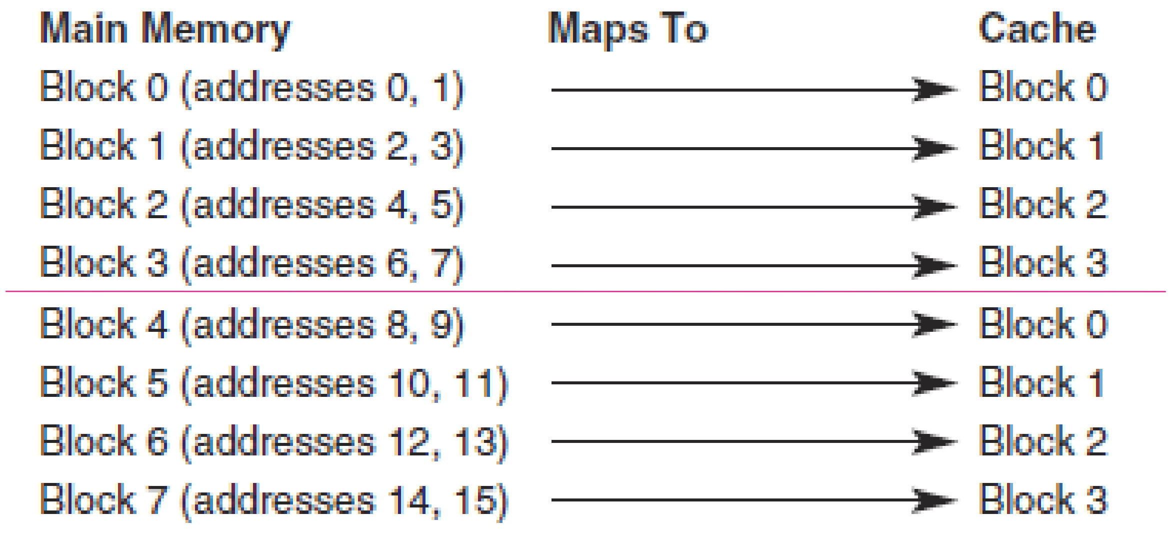
$2^3 * 2^1 = 4$ bits for addressing
16 words in the main memory

For this example:

- A main memory address has 4 bits (because there are 16 words in main memory).
- This 4-bit main memory address is divided into three fields:

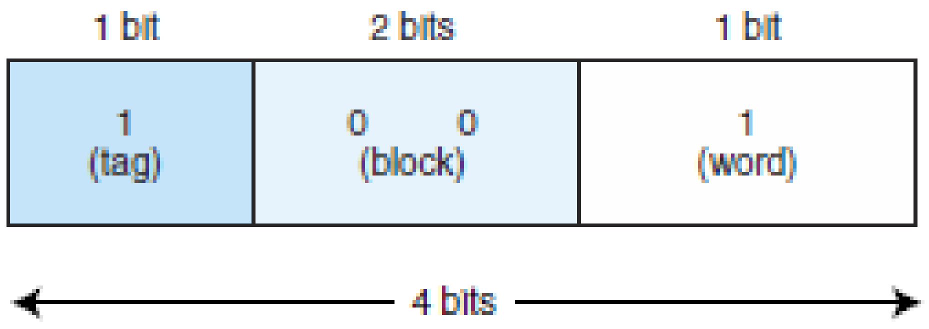
The **word field** is 1 bit (we need only 1 bit to differentiate between the two words in a block);
 the **block field** is 2 bits (we have 4 blocks in cache and need 2 bits to uniquely identify each block); and the **tag field** has 1 bit (this is all that is left over).
The main memory address is divided into the fields shown in Figure below:

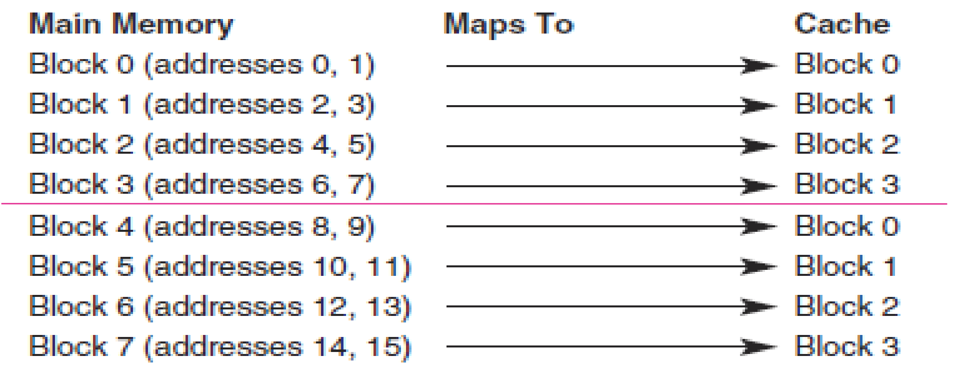




Main Memory	Maps To	Cache
Block 0 (addresses 0, 1)	→	Block 0
Block 1 (addresses 2, 3)	→	Block 1
Block 2 (addresses 4, 5)	→	Block 2
Block 3 (addresses 6, 7)	→	Block 3
Block 4 (addresses 8, 9)	→	Block 0
Block 5 (addresses 10, 11)	→	Block 1
Block 6 (addresses 12, 13)	→	Block 2
Block 7 (addresses 14, 15)	→	Block 3

Suppose we generate the main memory address 9. We can see from the mapping listing above that address 9 is in main memory block 4 and should map to cache block 0 (which means the contents of main memory block 4 should be copied into cache block 0).





Main Memory	Maps To	Cache
Block 0 (addresses 0, 1)	→	Block 0
Block 1 (addresses 2, 3)	→	Block 1
Block 2 (addresses 4, 5)	→	Block 2
Block 3 (addresses 6, 7)	→	Block 3
Block 4 (addresses 8, 9)	→	Block 0
Block 5 (addresses 10, 11)	→	Block 1
Block 6 (addresses 12, 13)	→	Block 2
Block 7 (addresses 14, 15)	→	Block 3

When the CPU generates this address, it first takes the block field bits 00 and uses these to search in cache blocks.

00 indicates that cache block 0 should be checked.

If the cache block is valid, it then compares the tag field value of 1 to the tag associated with cache block 0.

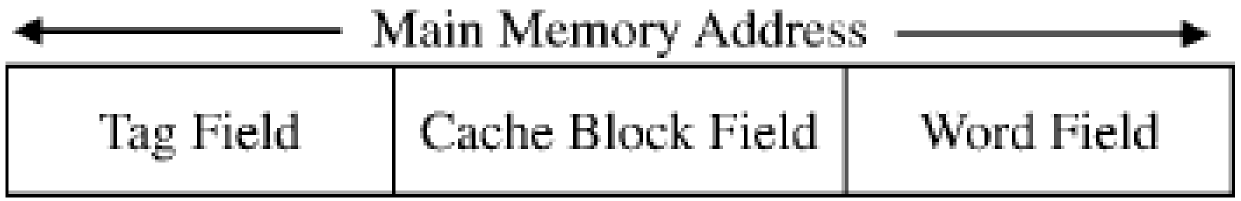
If the cache tag is 1, then block 4 currently resides in cache block 0.

If the tag is 0, then block 0 from main memory is located in block 0 of cache.

(To see this, compare
main memory address 9 = 1001 , which is in block 4, to
main memory address 1 = 0001, which is in block 0.

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

According to the direct-mapping technique the MMU interprets the address issued by the processor by dividing the address into three fields



1- Word field = $\log_2 B$, B is the size of the block in words

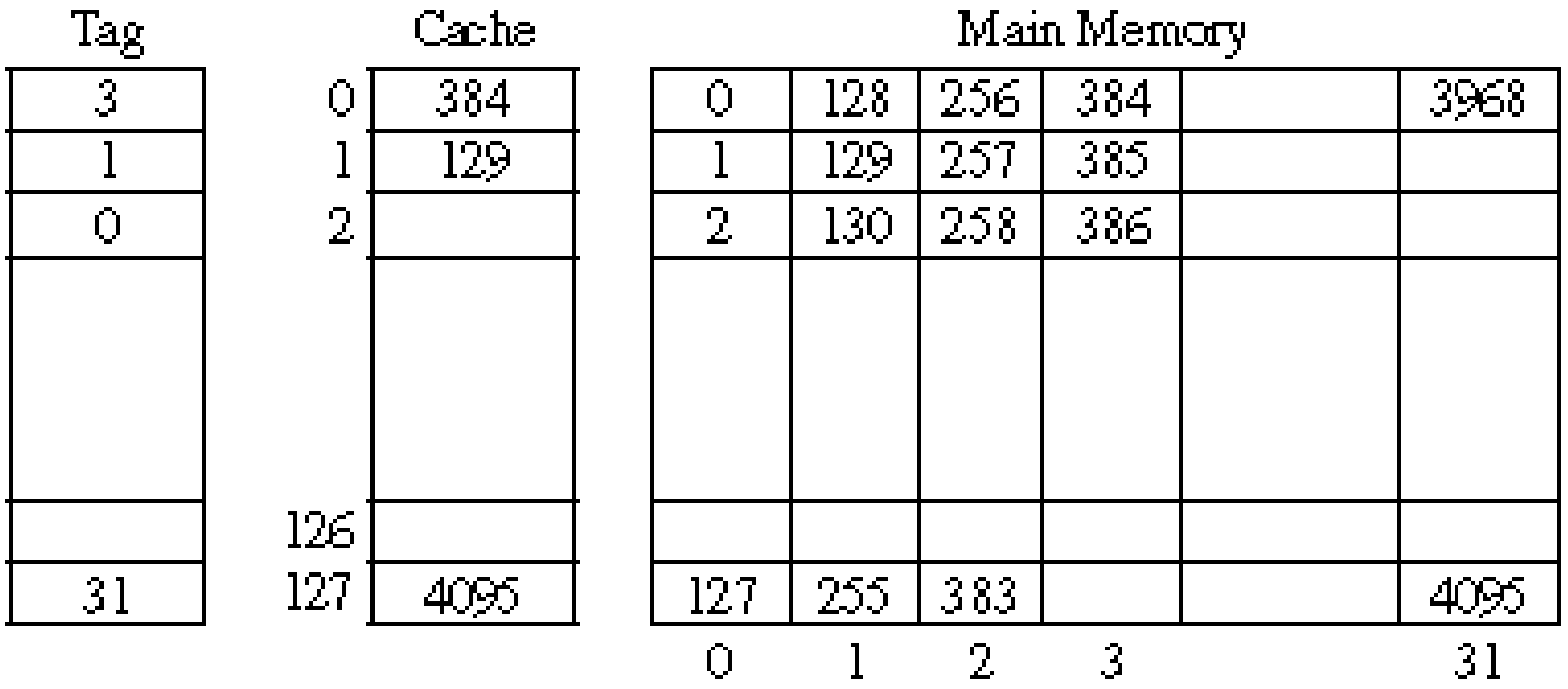
2- Block field = $\log_2 N$, N is the size in the cache in blocks (lines)

3- Tag field = $\log_2 (M/N)$, M is the size of the main memory in blocks

4- the number of bits in the main memory address = $\log_2 (B \times M)$

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 many-to-one mapping technique.

4k=4096
4096*16=65536=16 bit address of main memory



The MMU protocol steps 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 requested element 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 (يوضع)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.

Compute the above four parameters for our Example

1- Word field = $\log_2 B = \log_2(16) = \log_2(2^4) = 4$ bits

2- Block field =Line fields = $\log_2 N = \log_2(128) = \log_2(2^7) = 7$ bits

3- Tag field = $\log_2(M/N) = \log_2(2^2 \times 2^{10} / 2^7) = \log_2(2^5) = 5$ bits

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

Here, the "Word" field selects one from among the 16 addressable words in a line.
The block "Line" field defines the cache line where this memory line should reside.
The "Tag" field of the address is then compared with that cache line's 5-bit tag to determine whether there is a hit or a miss.

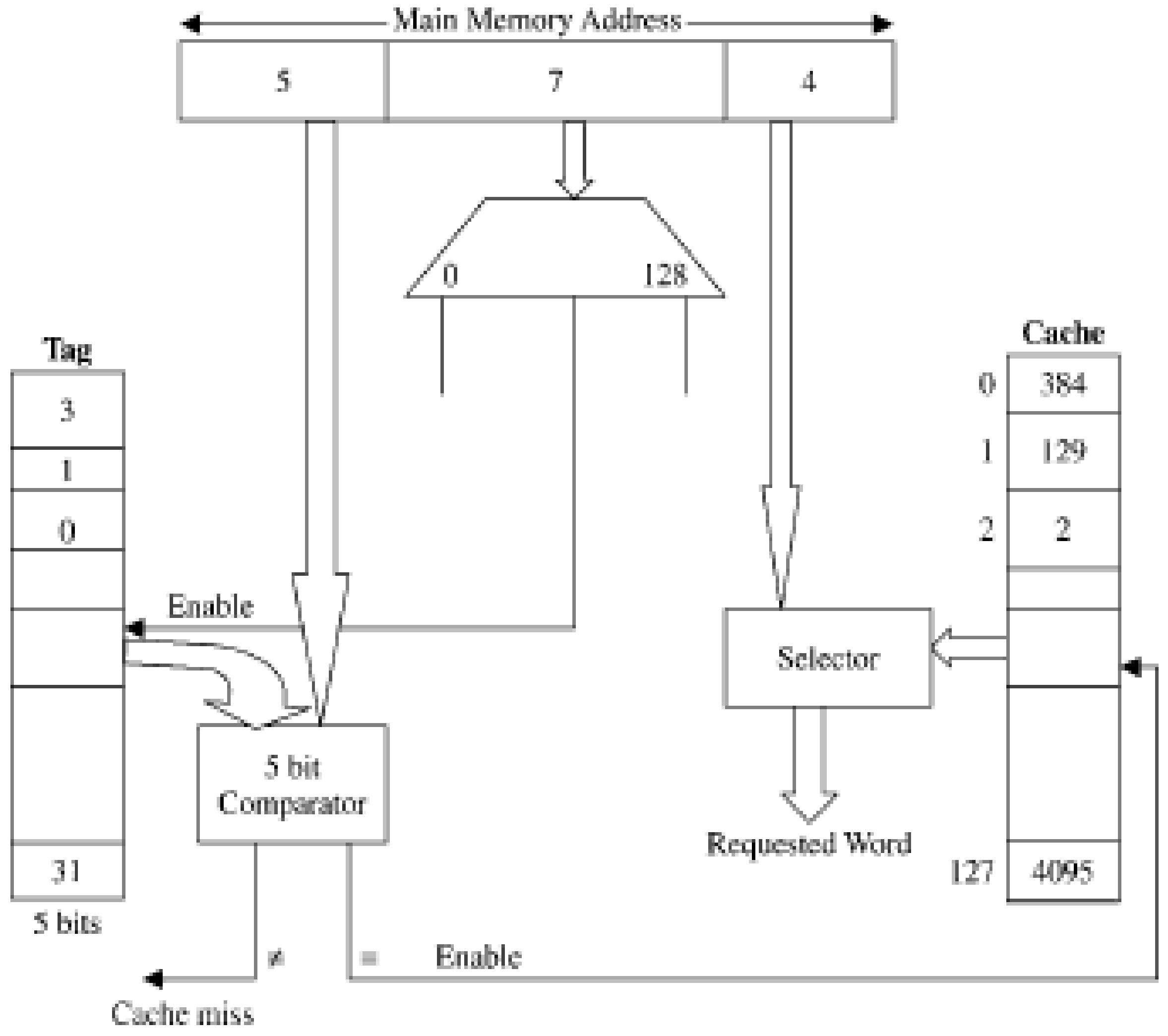
If there's a miss, we need to swap out the memory line that occupies that position in the cache and replace it with the desired memory line.

E.g., Suppose we want to read or write a word at the address 357A, whose 16 bits are 00110 1010111 1010.
This translates to Tag = 6, line = 87, and Word = 10 (all in decimal).

If line 87 in the cache has the same tag (6), then memory address 357A is in the cache.

Otherwise, a miss has occurred and the contents of cache line 87 must be replaced by the memory line 001101010111 = 855 before the read or write is executed.

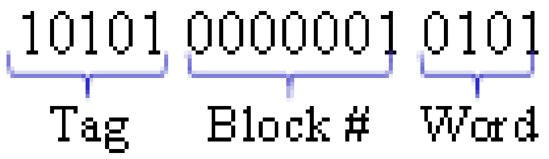
Direct mapping is the most efficient cache mapping scheme, but it is also the least effective in its utilization of the cache - that is, it may leave some cache lines unused.



Summery of direct: Use tag to see if a desired word is in cache

– It there is no match, the block containing the required word must first be read from the memory

Ex: address is A815



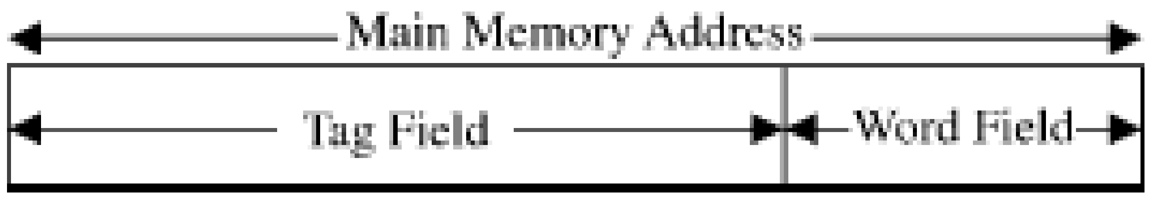
$$\underbrace{10101}_{\text{Tag}}\ \underbrace{0000001}_{\text{Block \#}}\ \underbrace{0101}_{\text{Word}}$$

a. Check if cache has tag 10101 for block 1
match -> hit; different -> miss, load the corresponding block

b. Access word 5 of the block

Fully Associative Mapping

The MMU interprets the address issued by the processor by dividing it into two fields as



1- Word field = $\log_2 B$, B is the size of the block in words

2- Tag field = $\log_2 (M)$, M is the size of the main memory in blocks

3- the number of bits in the main memory address = $\log_2 (BxM)$

we can now proceed to explain the protocol used by the MMU to satisfy a request, the protocol steps 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.

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.

Return to our example:

1- Word field = $\log_2 B = \log_2 (16) = \log_2 (2^4) = 4$ bits

2- Tag field = $\log_2 (M) = \log_2 (2^2 \times 2^{10}) = \log_2 (2^{12}) = 12$ bits

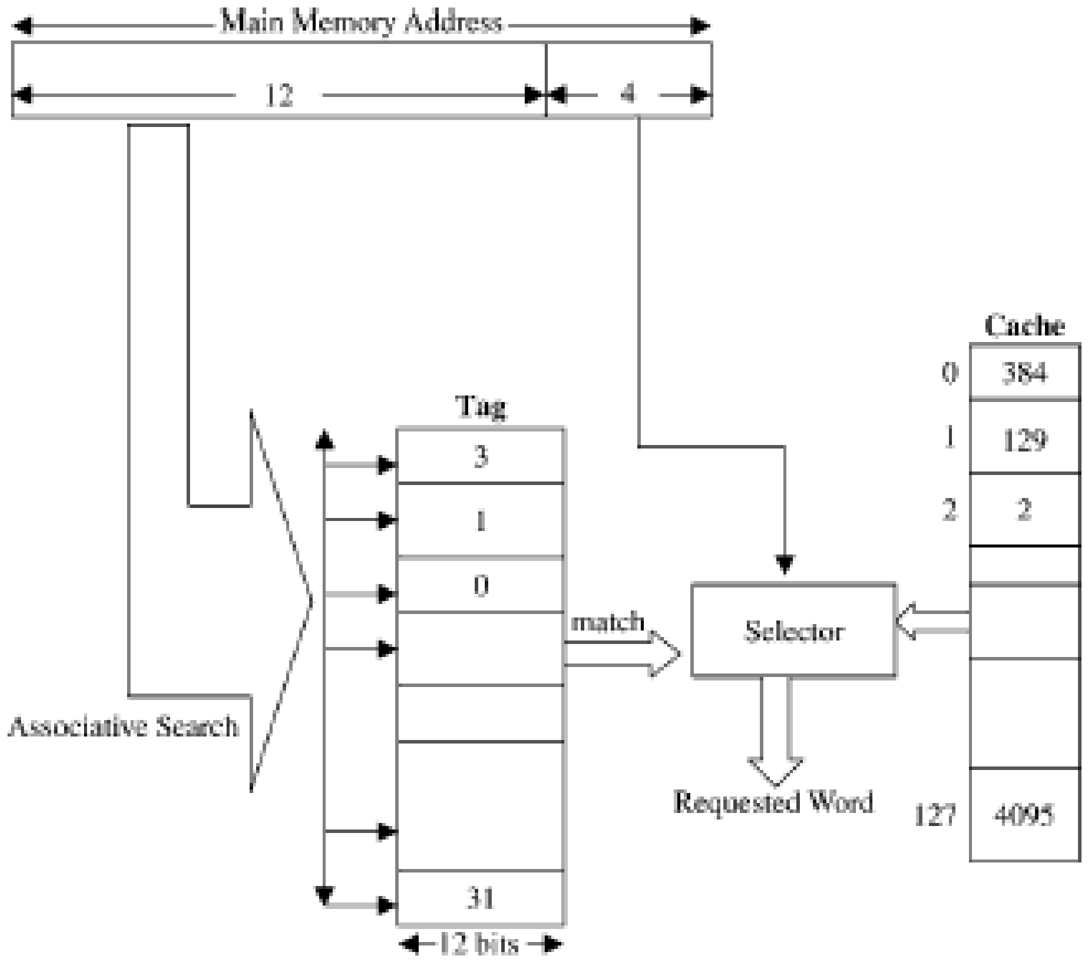
3- the number of bits in the main memory address = $\log_2 (B \times M) = = \log_2 (2^4 \times 2^{12}) = \log_2 (2^{16}) = 16$ bits

Here, the "Tag" field identifies one of the $2^{12} = 4096$ memory lines; all the cache tags are searched to find out whether or not the Tag field matches one of the cache tags. If so, we have a hit, and if not there's a miss and we need to replace one of the cache lines by this line before reading or writing into the cache. (The "Word" field again selects one from among 16 addressable words (bytes) within the line.)

For example, suppose again that we want to read or write a word at the address 357A, whose 16 bits are 0011010101111010. Under associative mapping, this translates to Tag = 855 and Word = 10 (in decimal).

So we search all of the 128 cache tags to see if any one of them will match with 855. If not, there's a miss and we need to replace one of the cache lines with line 855 from memory before completing the read or write.

The search of all 128 tags in the cache is time-consuming. However, the cache is fully utilized since none of its lines will be unused prior to a miss (recall that direct mapping may detect a miss even though the cache is not completely full of active lines).



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 following:

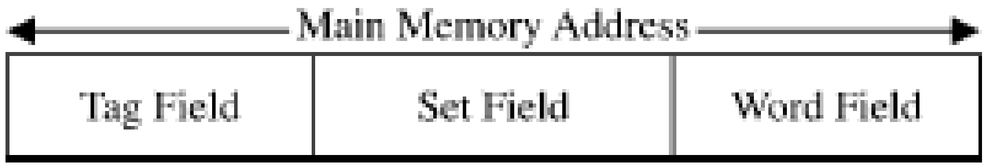
1- Word field = $\log_2 B$, B is the size of the block in words

2- Set field = $\log_2 S$, S is the number of sets in the cache,

S= N/Bs, where N is the number of cache blocks and Bs is the number of blocks per set.

3- Tag field = $\log_2 (M/S)$, M is the size of the main memory in blocks

4- the number of bits in the main memory address = $\log_2 (BxM)$



Return to our example:
Assume that the system uses set-associative mapping with four blocks per set.

First: S= N/Bs = 128/4 = 32 sets

1- Word field = $\log_2 B = \log_2 (16) = \log_2 (2^4)$= 4 bits

2- set field= $\log_2 (32) = \log_2 (2^5)$= 5 bits

3- Tag field= $\log_2 (2^2 \times 2^{10} / 32)$ =7 bits

3- the number of bits in the main memory address = $\log_2$ (BxM) = $= \log_2 (2^4 \times 2^{12}) = \log_2 (2^{16})$= 16 bits the protocol used by the MMU to satisfy a request made by the processor for accessing a given element. the steps of this protocol is:

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

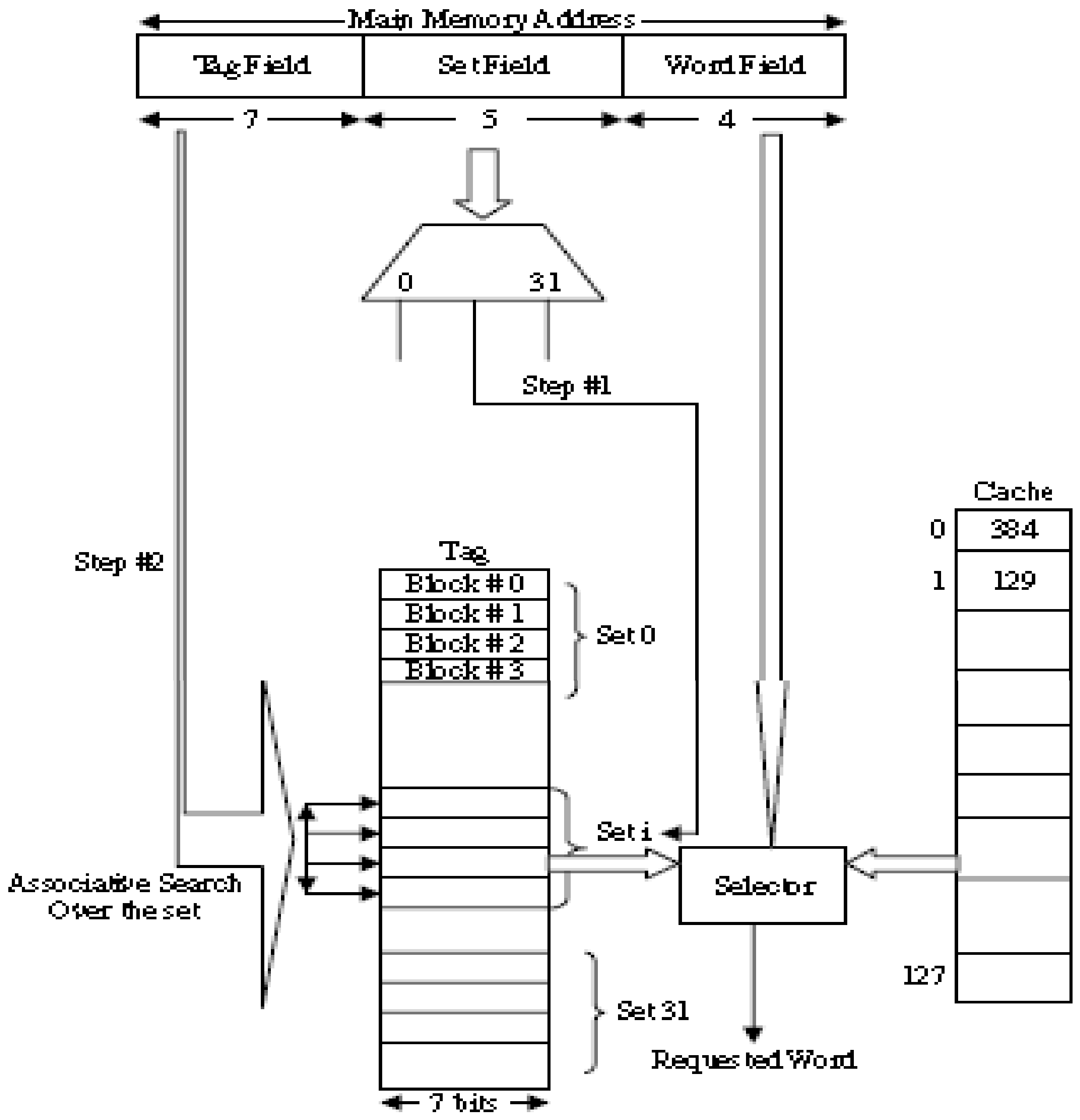
2. Use the Tag field to find a match with any of the (four) blocks in the determined set. A match in the tag memory indicates that the specified set determined in step 1 is currently holding the targeted block, that is, a cache hit.

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.

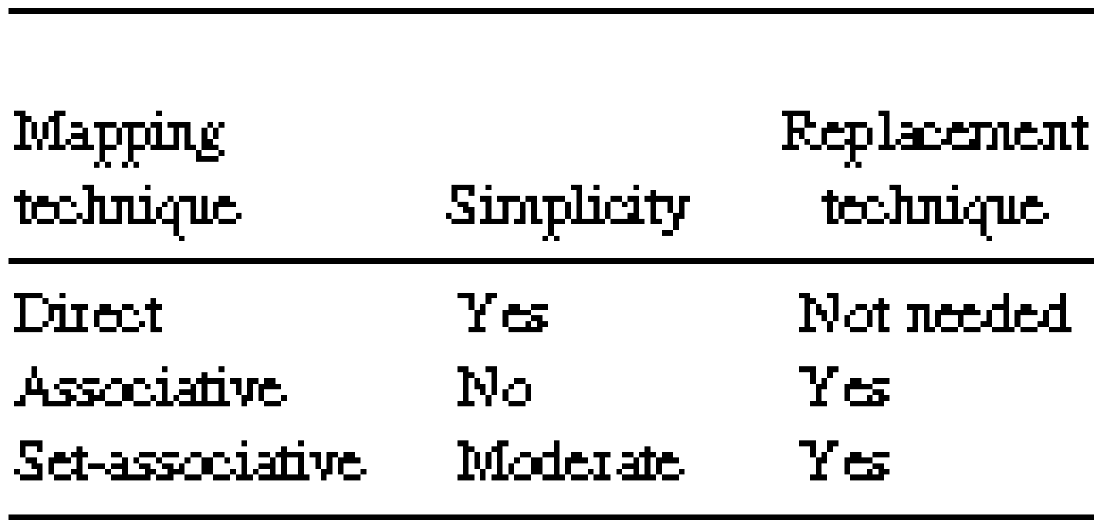
4. If in step 2, no match is found, then this indicates a cache miss. Therefore, the required block has to be brought from the main memory, deposited in the specified set first, and the targeted element (word) is made available to the processor. The cache Tag memory and the cache block memory have to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

E.g., Again suppose we want to read or write a word at the memory address 357A, whose 16 bits are 0011010101111010.

Under set-associative mapping, this translates to Tag = 26, Set = 23, and Word = 10 (all in decimal).
So we search tags in cache set 23 to see if either one matches tag 26. If so, we have a hit. Otherwise, one of these blocks must be replaced by the memory line being addressed (good old line 855) before the read or write can be executed.



Summery of the three techniques



Mapping technique	Simplicity	Replacement technique
Direct	Yes	Not needed
Associative	No	Yes
Set-associative	Moderate	Yes

Effective Access Time and Hit Ratio

The performance of a hierarchical memory is measured by its effective access time (EAT), or the average time per access. EAT is a weighted average that uses the hit ratio and the relative access times of the successive levels of the hierarchy. The formula for calculating effective access time for a two-level memory is given by:

$$\text{EAT} = H \times \text{Access}_C + (1 - H) \times \text{Access}_{MM}$$

where H = cache hit rate,

AccessC = cache access time,

AccessMM = main memory access time.

For example, suppose the cache access time is 10ns, main memory access time is 200ns, and the cache hit rate is 99%. The average time for the processor to access an item in this two-level memory would then be:

$$\text{EAT} = H \times \text{Access}_C + (1 - H) \times \text{Access}_{MM}$$

$$\text{EAT} = \underbrace{0.99(10\text{ns})}_{\text{cache hit}} + \underbrace{0.01(200\text{ns})}_{\text{cache miss}} = 9.9\text{ns} + 2\text{ns} = 11\text{ns}$$

This formula can be extended to apply to three- or even four-level memories.