Part4 : Cache Memory

1.1 Cache Memory

Principles Cache memory is designed to combine the memory access time of expensive, high-speed memory combined with the large memory size of less expensive, lower-speed memory. The concept is illustrated in Figure 4.1a.

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.

Figure 4.1b depicts the 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.

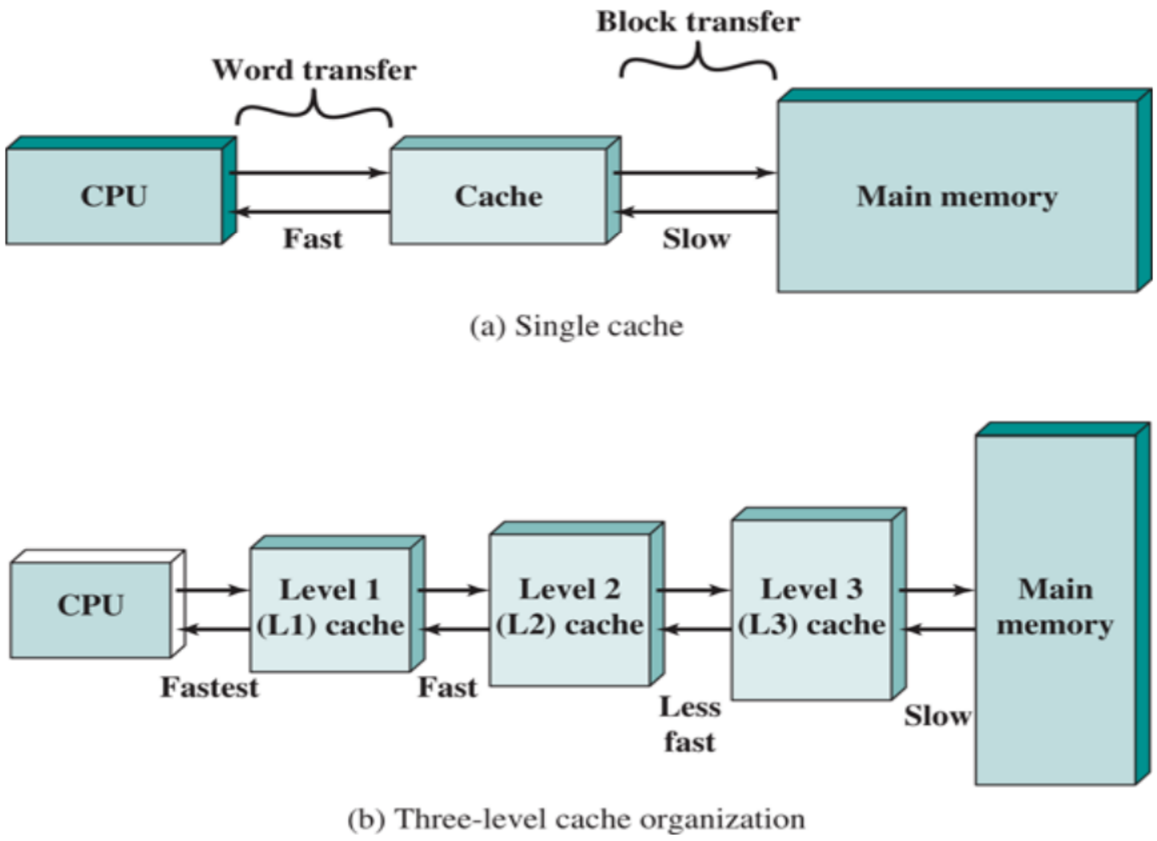


Figure 4.1 Cache and Main Memory

Figure 4.2 depicts the structure of a cache/main-memory system. Several terms are introduced:

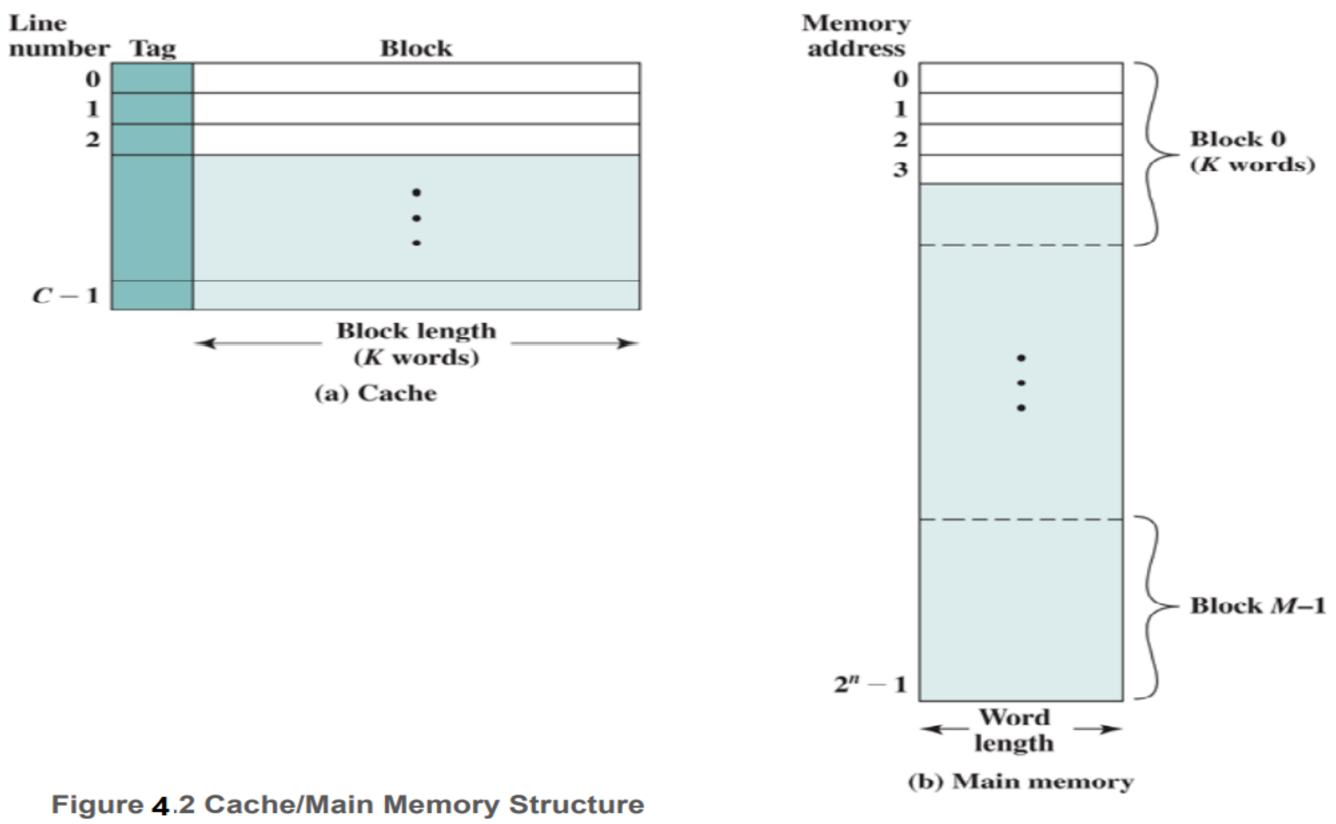


Figure 4.2 Cache/Main Memory Structure

- Block: The minimum unit of transfer between cache and main memory
- Frame: To distinguish between the data transferred and the chunk of physical memory, the term frame, or block frame, is sometimes used with reference to caches.
- Line: A portion of cache memory capable of holding one block, so-called because it is usually drawn as a horizontal object.
- Tag: A portion of a cache line that is used for addressing purposes, as explained subsequently

Main memory consists of up to 2^n addressable words, with each word having a unique n-bit address.This memory is considered to consist of a number of fixed-length blocks of K words each. That is, there are blocks in main memory. That is, there are $M=2^n/k$ blocks in main memory.The cache consists of M lines. Each line contains K words, plus a tag.
The term line size refers to the number of data bytes, or block size, contained in a line.

Figure 4.3 illustrates the read operation. The processor generates the read address (RA) of a word to be read. If the word is contained in the cache (cache hit), it is delivered to the processor.
If a cache miss occurs, two things must be accomplished:

- the block containing the word must be loaded in to the cache,
- and the word must be delivered to the processor.

Note: One possible technique that is used to increase the bandwidth is *memory interleaving*. To achieve best results, we can assume that the block brought from the main memory to the cache, upon a cache miss.

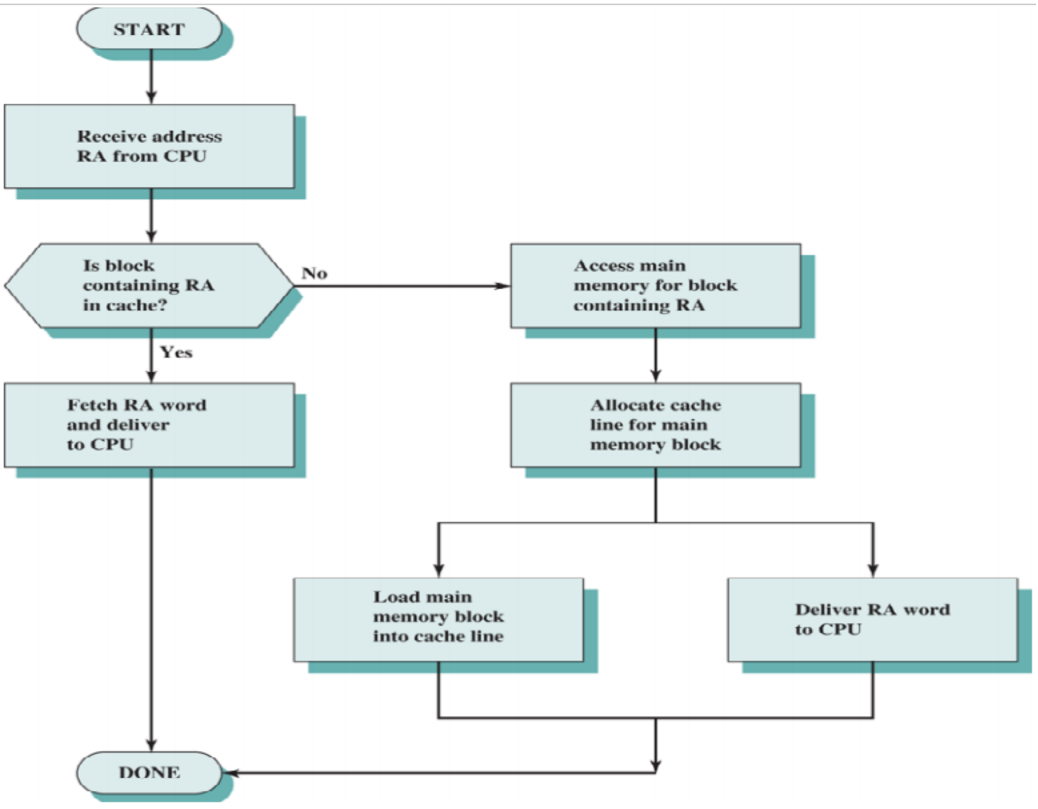

START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

Figure 4.3 Cache Read Operation

The organization shown in Figure 4.4, which is typical of contemporary cache organizations. In this organization:

- the cache connects to the processor via data, control, and address lines.
- The data and address lines also attach to data and address buffers, which attach to a system bus from which main memory is reached.
- When a cache hit occurs, the data and address buffers are disabled and communication is only between processor and cache, with no system bus traffic.
- When a cache miss occurs, the desired address is loaded onto the system bus and the data are returned through the data buffer to both the cache and the processor

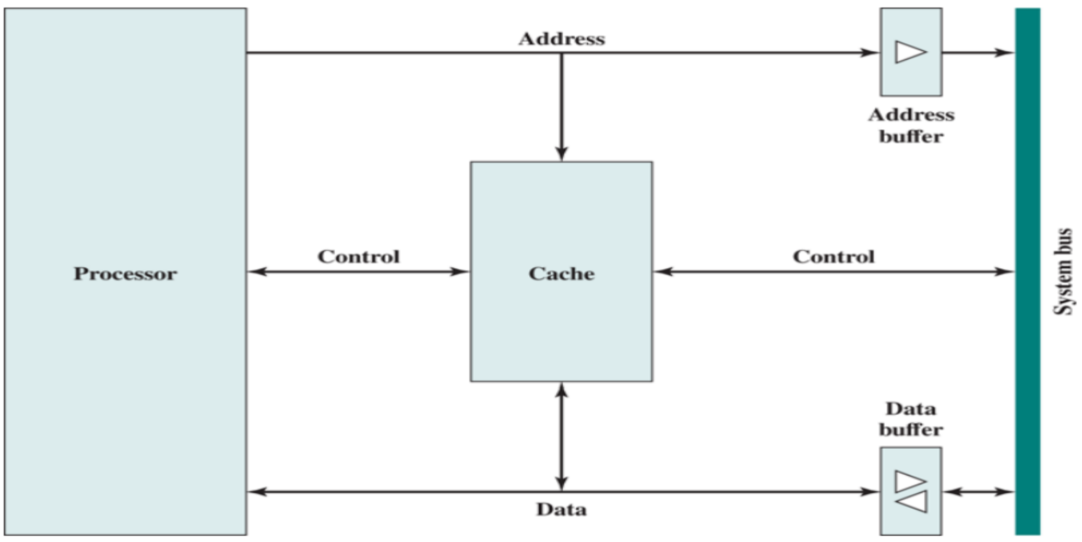


Figure 4.4 Typical Cache Organization

4.2 Elements of Cache Design

Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures.

Cache Addresses

- For reads to and writes from main memory, a hardware memory management unit (MMU) translates each virtual address into a physical address in main memory.
- When virtual addresses are used, the system designer may choose to place the cache between the processor and the MMU or between the MMU and main memory (Figure 4.5).
- A logical cache : also known as a virtual cache, stores data using virtual addresses. The processor accesses the cache directly, without going through the MMU.
- A physical cache :stores data using main memory physical addresses

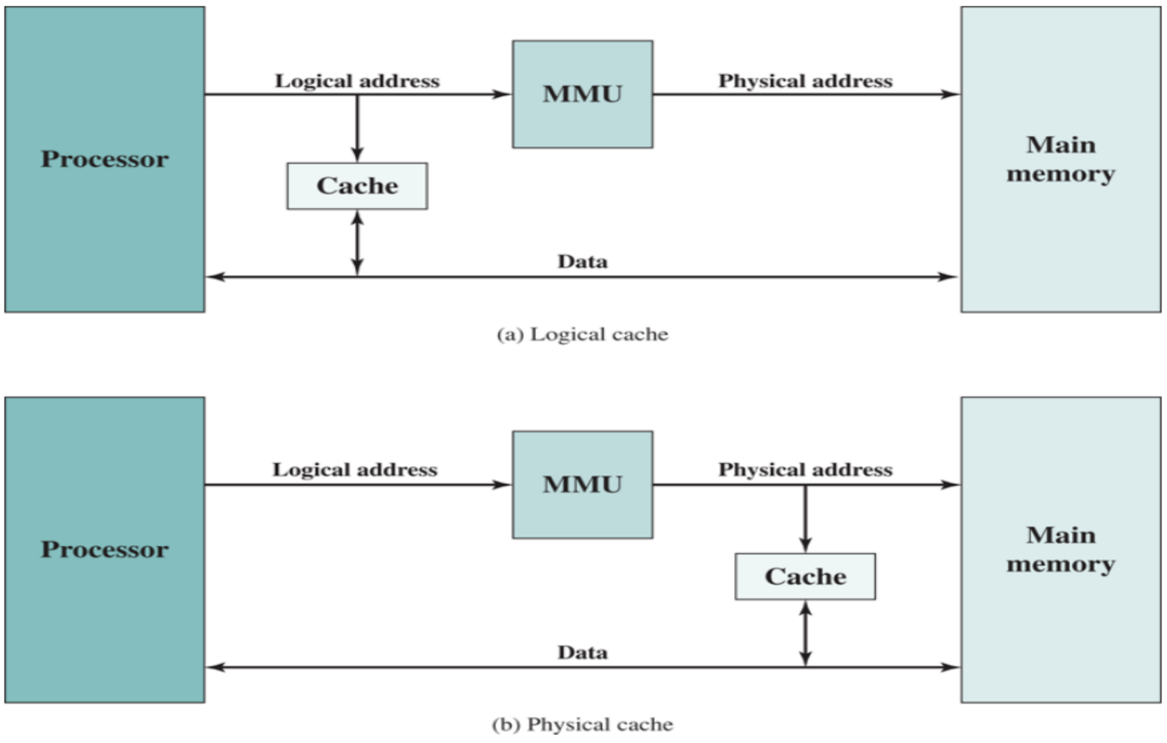


Figure 4.5 Logical and Physical Caches

Cache Size

We would like the size of the cache to be small enough so that the overall average cost per bit is close to that of main memory alone and large enough so that the overall average access time is close to that of the cache alone. There are several other motivations for minimizing cache size. The larger the cache, the larger the number of gates involved in addressing the cache. Table 4.1 lists the cache sizes of some current and past processors.

Table 4 .1 Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128 to 256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256 to 512 KB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 kB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 kB/32 kB	4 MB	—
Itanium 2	PC/server	2002	32 kB	256 KB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB

[a] Two values separated by a slash refer to instruction and data caches.
[b] Both caches are instruction only; no data caches.

Logical Cache Organization(Mapping Function)

Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines. Further, a means is needed for determining which main memory block currently occupies a cache line . The choice of the mapping function dictates how the cache is logically organized. Three techniques can be used: direct, associative, and set-associative.

Example For all three cases, the example includes the following elements:

- The cache can hold 64 KBytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as 16K $= 2^{14}$ lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address ($2^{24} = 16M$). Thus, for mapping purposes, we can consider main memory to consist of 4M blocks of 4 bytes each.

1-Direct Mapping:

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. The mapping is expressed as: *i = j modulo m*

i = cache line number

j = main memory block number

m = number of lines in the cache

Figure 4.6a shows the mapping for the first *m* blocks of main memory. Each block of main memory maps into one unique line of the cache. The next *m* blocks of main memory map into the cache in the same fashion; that is, block B_m of main memory maps into line L_0 of cache, block B_{m+1} maps into line L_1, and so on.

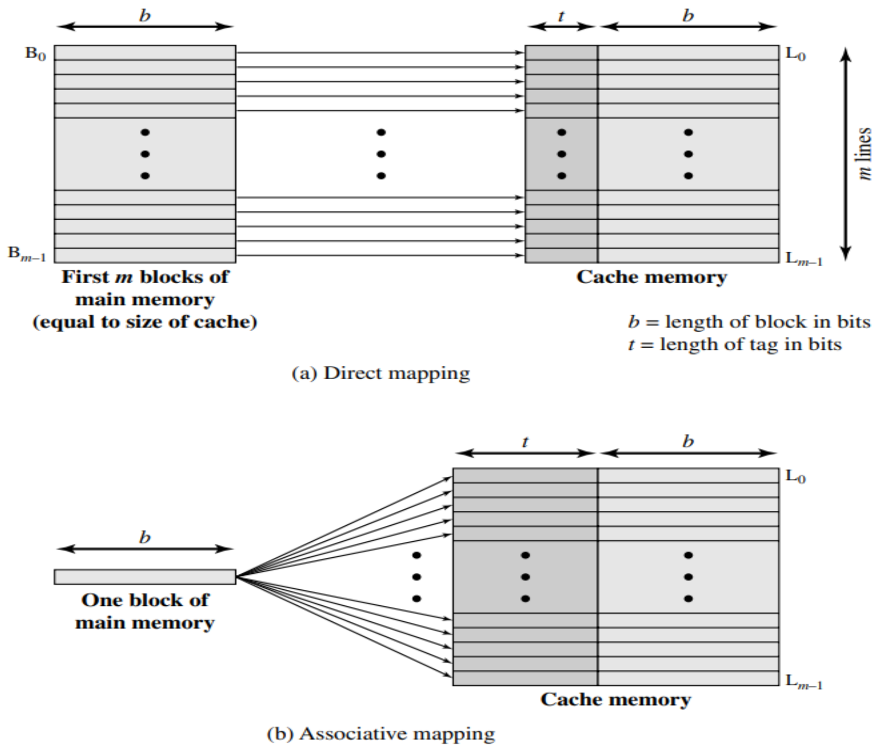


Figure 4.6 Mapping from Main Memory to Cache: Direct and Associative

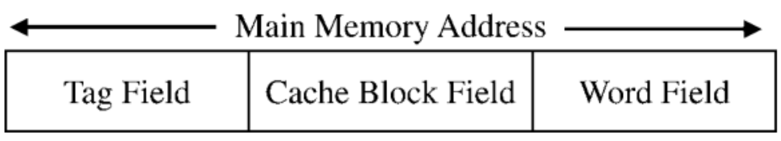
1. Word field = $\log_2$ B, where B is the size of the block in words.
2. Block field = $\log_2$ N, where N is the size of the cache in blocks.
3. Tag field = $\log_2$ (M/N), where M is the size of the main memory in blocks.
4. The number of bits in the main memory address = $\log_2$ (B x M)

Example : 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. The following table 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.

Table 4.2 Mapping main memory blocks to cache blocks

Tag		Cache	Main Memory					
3	0	384	0	128	256	384		3968
1	1	129	1	129	257	385		
0	2		2	130	258	386		
	126							
31	127	4095	127	255	383			4095
			0	1	2	3		31

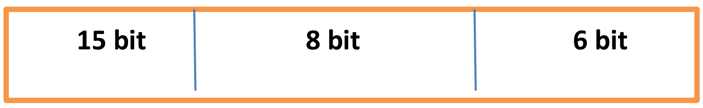
1. Word field $=\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits
2. Block field $= \log_2 N = \log_2 128 = \log_2 2^7 = 7$
3. bits Tag field $= \log_2 (M/N) = \log_2 (2^2 \text{ X} 2^{10}/2^7) = 5$ bits
4. The number of bits in the main memory address $= \log_2 (B \text{ X } M) = \log_2 (2^4 \text{ X } 2^{12}) = 16$ bits.

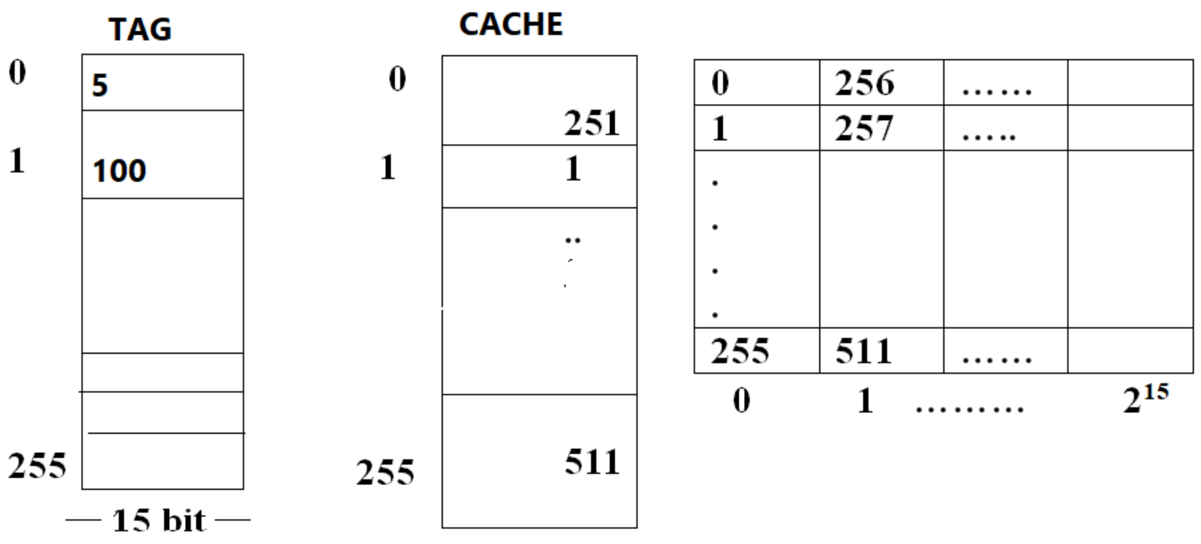


Direct-mapped address fields

<u>Example:</u> **Word field B= 64 word**
Block field = 256 block
Main memory= 8 M block

<u>Answer:</u> **Word field B= $\log_2 64 = \log_2 2^6 = 6$**
Block filed= $\log_2 256 = \log_2 2^8 = 8$
bits Tag field = $\log_2 (M/N) = \log_2 (2^3 \text{ x } 2^{20}/2^8) = \log_2(2^{23}/2^8)$=15 bits
bits in the main memory address = $\log_2 (B \text{ X } M) = \log_2 (2^6 \text{ x } 2^{23}) = 29$





The following Figure 4.7 illustrates the general mechanism.:

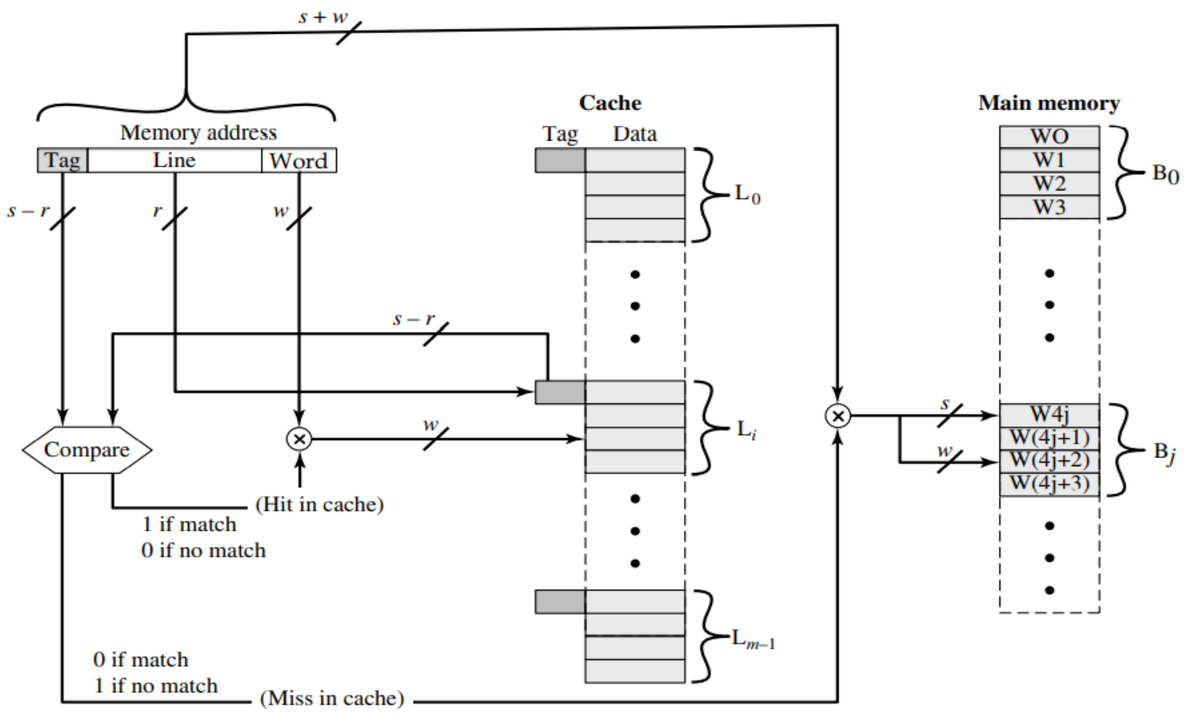


Figure 4..7 Direct-Mapping Cache Organization

Example

Figure 4.8 shows our example system using direct mapping.[5] In the example, $m = 16K = 2^{14}$ and $i = j \text{ modulo } 2^{14}$. The mapping becomes

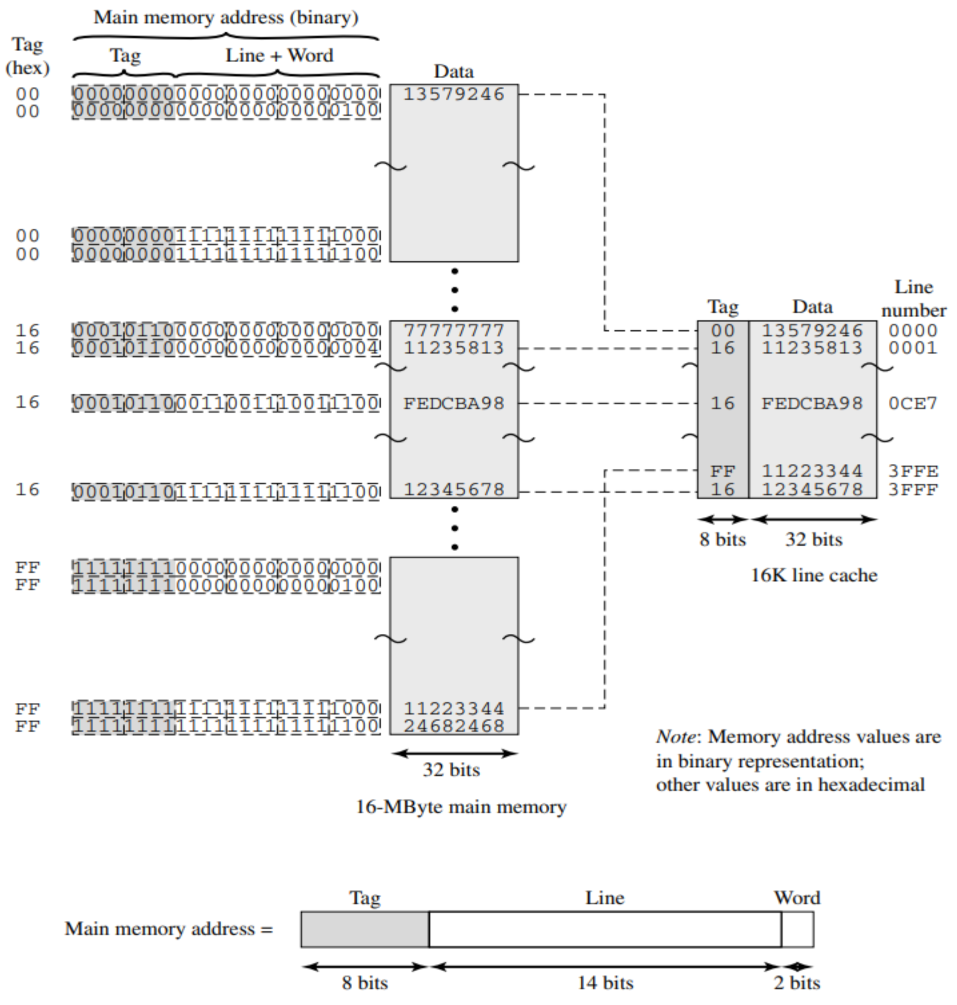


Figure 4.8 Direct Mapping Example

2- Fully ASSOCIATIVE MAPPING

Associative mapping overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of the cache(**Figure 4.6b**) .The Tag field uniquely identifies a block of main memory. To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match. Figure 4.9 illustrates the logic

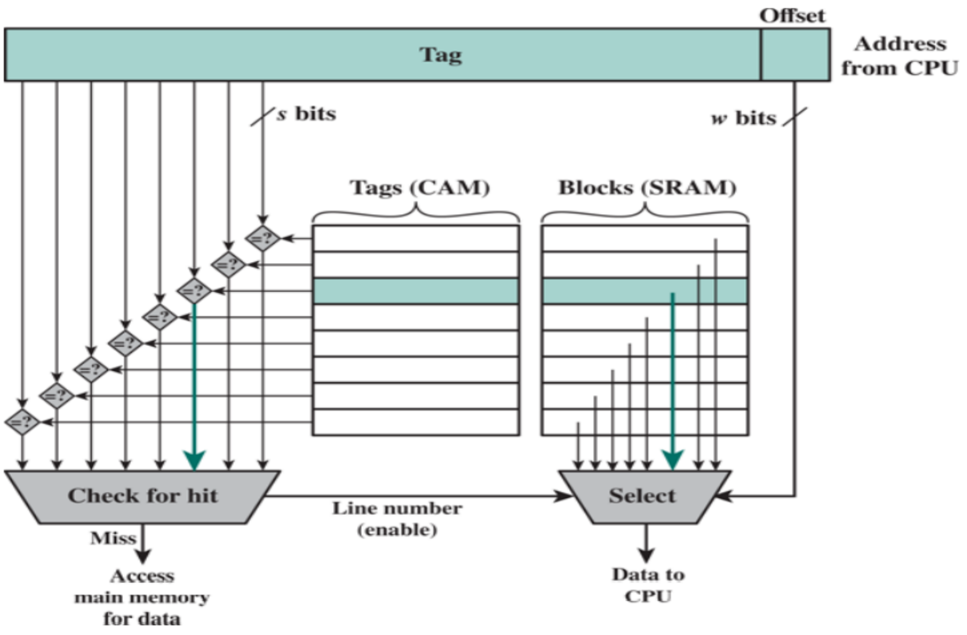
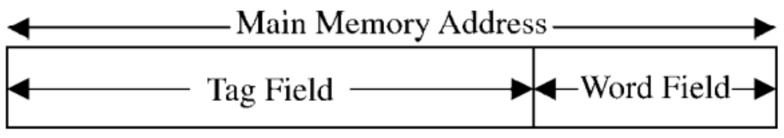


Figure 4.9 Fully Associative Cache Organization

Example : 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.

1. Word field =$\log_2$ B =$\log_2$ 16 =$\log_2 2^4$ = 4 bits
2. Tag field =$\log_2$ M =$\log_2 2^2 \times 2^{10}$ = 12 bits
3. The number of bits in the main memory address =$\log_2$ (B x M) =$\log_2 (2^4 \times 2^{12})$ = 16 bits.



Associative-mapped address fields

Example: Word field B= 64 ,Main memory= 8 M

Answer: Word field B= $\log_2 64 = \log_2 2^6 = 6$
bits Tag field = $\log_2$ (M) = $\log_2(2^{23})$=23 bits
bits in the main memory address = $\log_2$ (B X M) = $\log_2 (2^6 \times 2^{23})$ =29

23 bit	**6 bit**

Example

Figure 4.11 shows our example using associative mapping. A main memory address consists of a 22-bit tag and a 2-bit byte number. The 22-bit tag must be stored with the 32-bit block of data for each line in the cache.

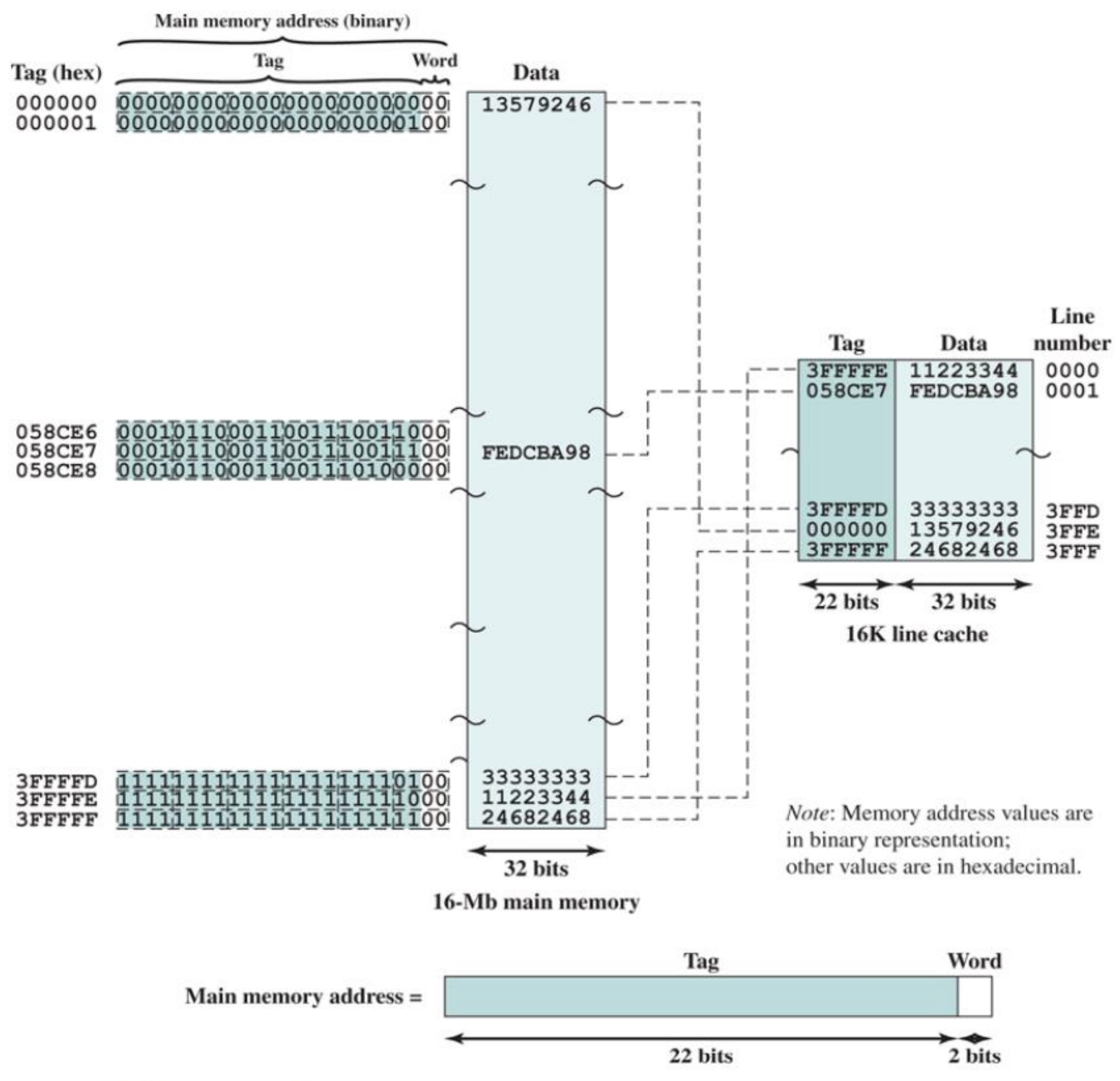


Figure 4.11 Associative Mapping Example

NOTE: CONTENT-ADDRESSABLE MEMORY(CAM): A CAM is designed such that when a bit string is supplied, the CAM searches its entire memory in parallel for a match. If the content is found, the CAM returns the address where the match is found and, in some architectures, also returns the associated data word. This process takes only one clock cycle. Figure 4.12 is a simplified illustration of the search function of a small CAM with four horizontal words, each word containing five bits, or cells. CAM cells contain both storage and comparison circuitry

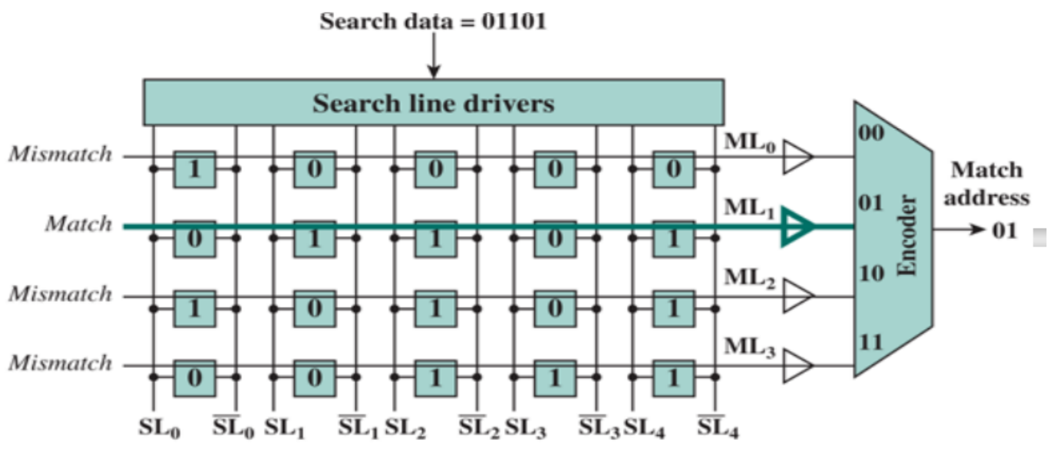

Search data = 01101
Search line drivers
Mismatch
Match
Mismatch
Mismatch
ML_0
ML_1
ML_2
ML_3
00
01
10
11
Encoder
Match address
01
SL_0 $\overline{SL}_0$ SL_1 $\overline{SL}_1$ SL_2 $\overline{SL}_2$ SL_3 $\overline{SL}_3$ SL_4 $\overline{SL}_4$

Figure 4.12 Simplified CAM circuitry

3-SET-ASSOCIATIVE MAPPING

Set-associative mapping is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages. In this case, the cache consists of number sets, each of which consists of a number of lines. The relationships are:

$$m = v \times k$$
$$i = j \text{ modulo } v$$

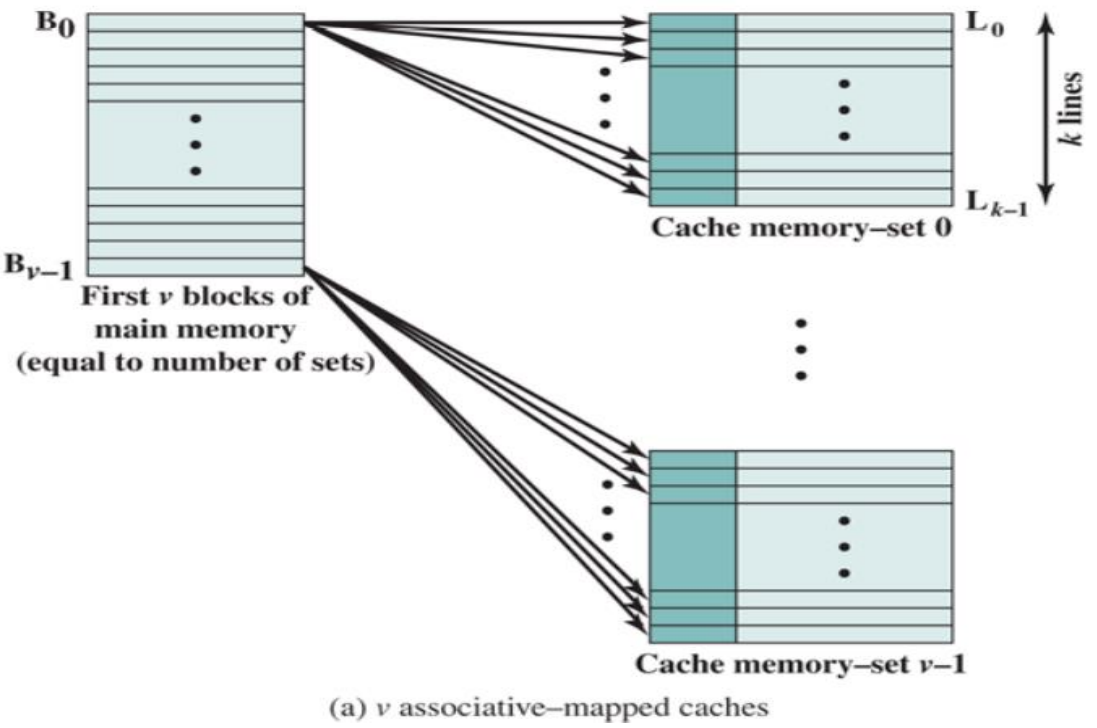
i = cache set number
j = main memory block number
m = number of lines in the cache
v = number of sets
k = number of lines in each se

Figure 4.13a illustrates this mapping for the first v blocks of main memory. As with associative mapping,

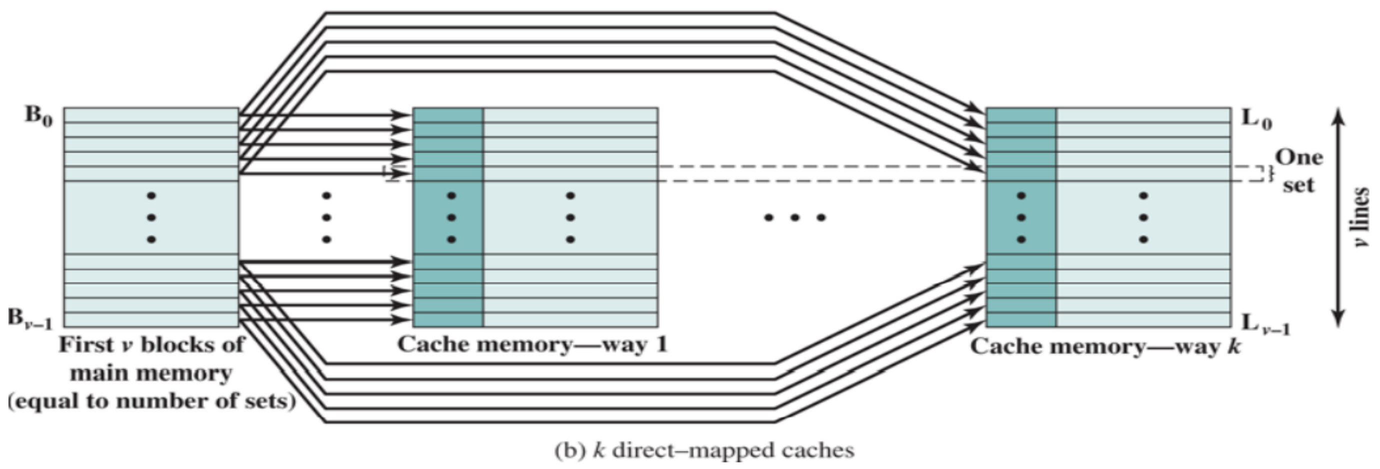
- each word maps into multiple cache lines.
- For set-associative mapping, each word maps into all the cache lines in a specific set, so that main memory block maps into set 0, and so on.
- Thus, the set-associative cache can be physically implemented as v associative caches, typically implemented as **v CAM** memories.

It is also possible to implement the set-associative cache as **k** direct mapping caches, as shown in Figure 4.13b.

- Each direct-mapped cache is referred to as a way, consisting of v lines.
- The first v lines of main memory are direct mapped into the v lines of each way; the next group of v lines of main memory are similarly mapped, and so on.



B_0
B_{v-1}
First v blocks of main memory (equal to number of sets)
L_0
L_{k-1}
k lines
Cache memory–set 0
Cache memory–set v–1
(a) v associative–mapped caches



B_0
B_{v-1}
First v blocks of main memory (equal to number of sets)
Cache memory—way 1
Cache memory—way k
L_0
L_{v-1}
One set
v lines
(b) k direct–mapped caches

Figure 4.13 Mapping from Main Memory to Cache: *k*-Way Set Associative

For set-associative mapping, the cache control logic interprets a memory address as three fields: Tag, Set, and Word. The d set bits specify one of sets. The s bits of the Tag and Set fields specify one of the blocks of main memory. Figure 4.14 illustrates the cache control logic.

Example :Figure 4.15 shows our example using two-way set-associative mapping with two lines in each set. The 13-bit set number identifies a unique set of two lines within the cache. It also gives the number of the block in main memory, modulo 2^{13} . Any of those blocks can be loaded into either of the two lines in the set. Note that no two blocks that map into the same cache set have the same tag number. For a read operation, the 13-bit set number is used to determine which set of two lines is to be examined. Both lines in the set are examined for a match with the tag number of the address to be accessed

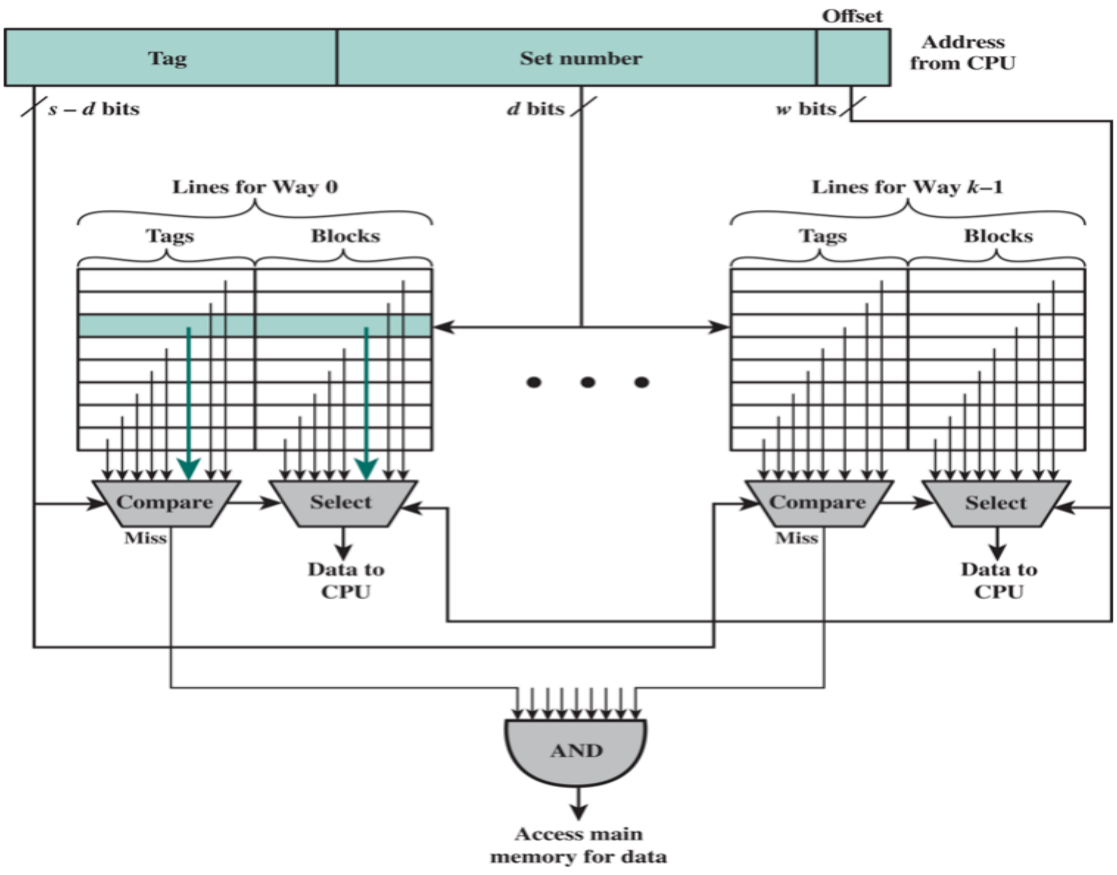


Figure 4.14 *k*-Way Set Associative Cache Organization

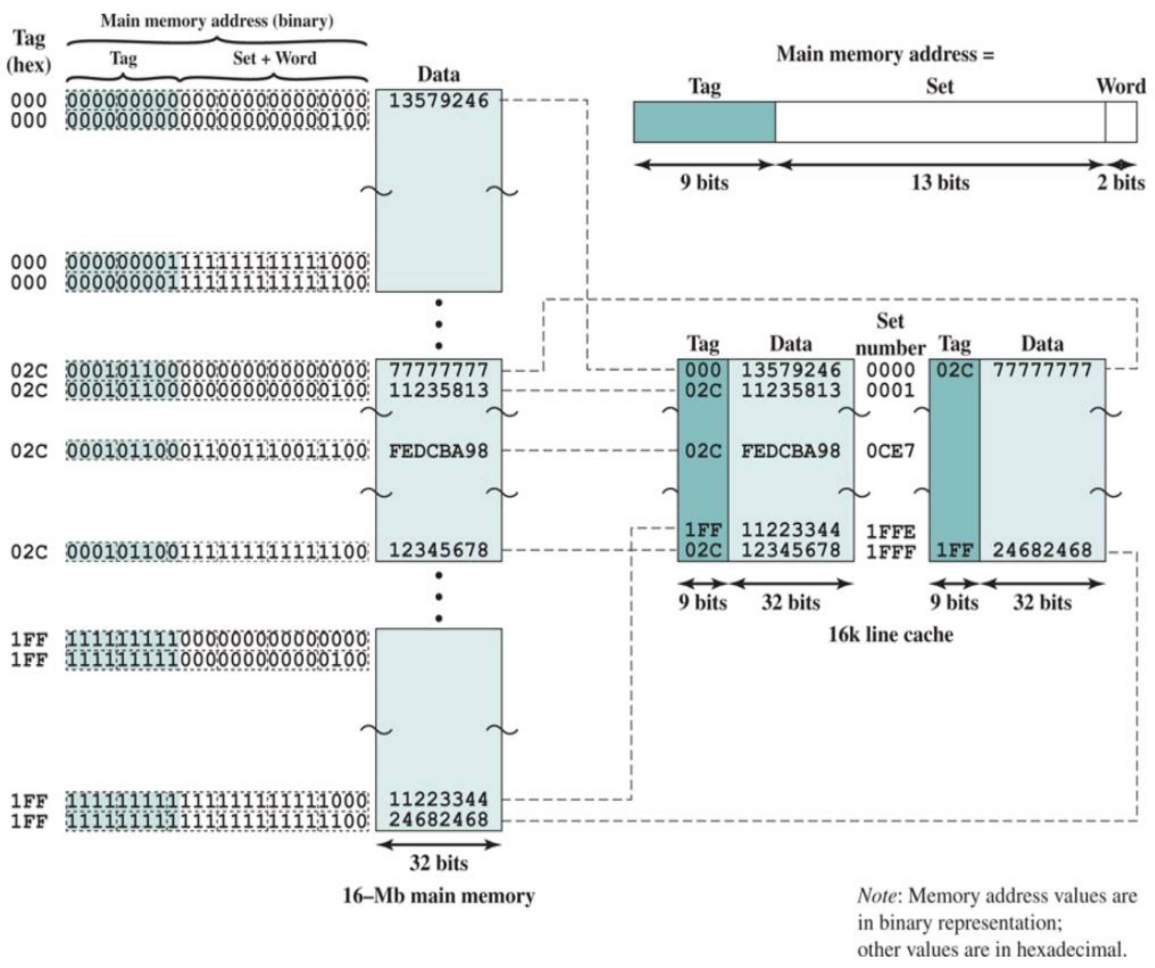

Main memory address (binary)
Tag (hex)
Tag
Set + Word
Data
000 000000000000000000000000 13579246
000 000000000000000000000100
000 000000000111111111111000
000 000000000111111111111100
02C 000101100000000000000000 77777777
02C 000101100000000000000100 11235813
02C 000101100011001110011100 FEDCBA98
02C 000101100111111111111100 12345678
1FF 111111111000000000000000
1FF 111111111000000000000100
1FF 111111111111111111111000 11223344
1FF 111111111111111111111100 24682468
32 bits
16–Mb main memory
Main memory address =
Tag
Set
Word
9 bits
13 bits
2 bits
Tag
Data
Set number
Tag
Data
000 13579246 0000 02C 77777777
02C 11235813 0001
02C FEDCBA98 0CE7
1FF 11223344 1FFE
02C 12345678 1FFF 1FF 24682468
9 bits
32 bits
9 bits
32 bits
16k line cache
Note: Memory address values are in binary representation; other values are in hexadecimal.

Figure 4.15 Two-Way Set-Associative Mapping Exampl

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.

$$S = \frac{128}{4} = 32 \text{ sets.}$$

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