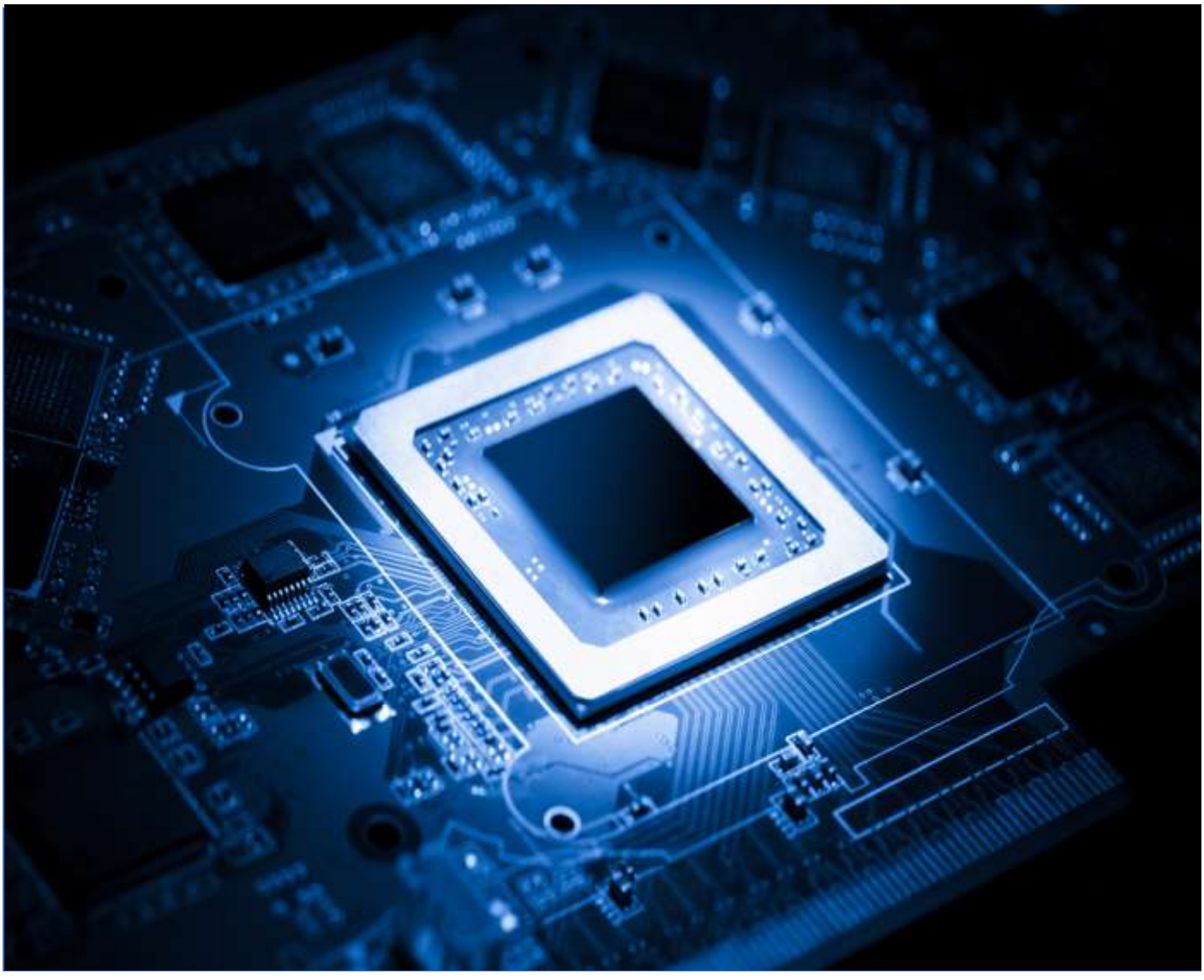
COMPUTER ORGANIZATION



Chapter Five

Memory System Design

1. BASIC CONCEPTS

In this section, we introduce a number of fundamental concepts that relate to the memory hierarchy of a computer.

❖ Memory Hierarchy

a typical memory hierarchy starts with a small, expensive, and relatively fast unit, called the **cache**, followed by a larger, less expensive, and relatively slow **main memory unit**.

Cache and main memory are built using solid-state semiconductor material. It is customary to call the fast memory level the **primary memory**.

The solid-state memory is followed by larger, less expensive, and far slower magnetic memories that consist typically of the (hard) disk and the tape. It is customary to call the disk the **secondary memory**, while the tape is conventionally called the **tertiary memory**.

The objective behind designing a memory hierarchy is to have a memory system that performs as if it consists entirely of the fastest unit and whose cost is dominated by the cost of the slowest unit.

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

Access: refers to the action that physically takes place during a *read* or *write* operation.

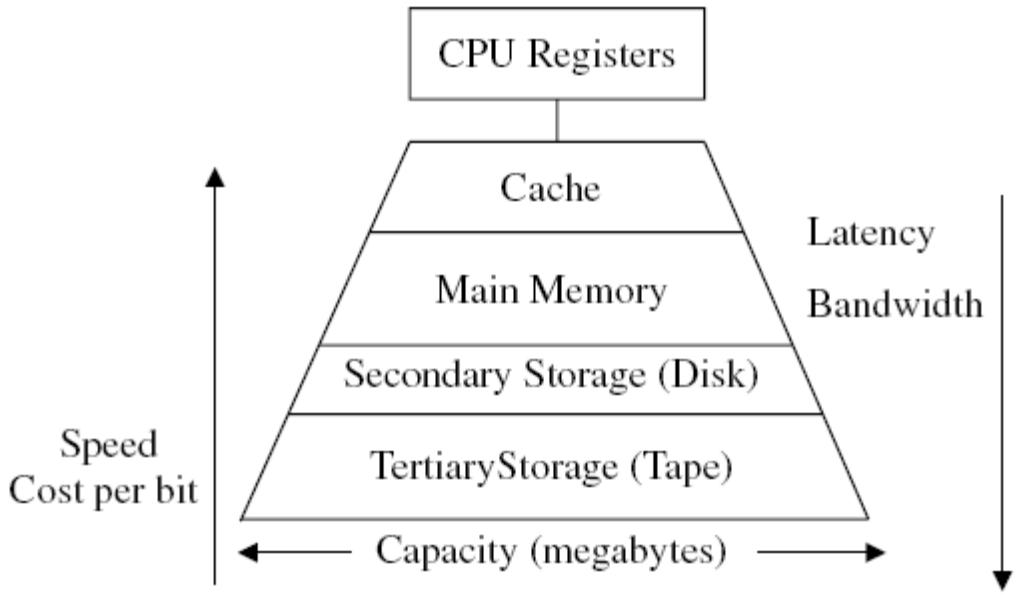
Capacity: The capacity of a memory level is usually measured in bytes.

Cycle time: The cycle time is defined as the time elapsed from the start of a read operation to the start of a subsequent read.

Latency: The latency is defined as the time interval between the request for information and the access to the first bit of that information.

Bandwidth: The bandwidth provides a measure of the number of bits per second that can be accessed.

Cost: The cost of a memory level is usually specified as dollars per megabytes.



Typical memory hierarchy

Random access refers to the fact that any access to any memory location takes the same fixed amount of time regardless of the actual memory location and/or the sequence of accesses that takes place.

Example: if a write operation to memory location 100 takes 15 ns and if this operation is followed by a read operation to memory location 3000, then the latter operation will also take 15 ns.

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.**

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 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.

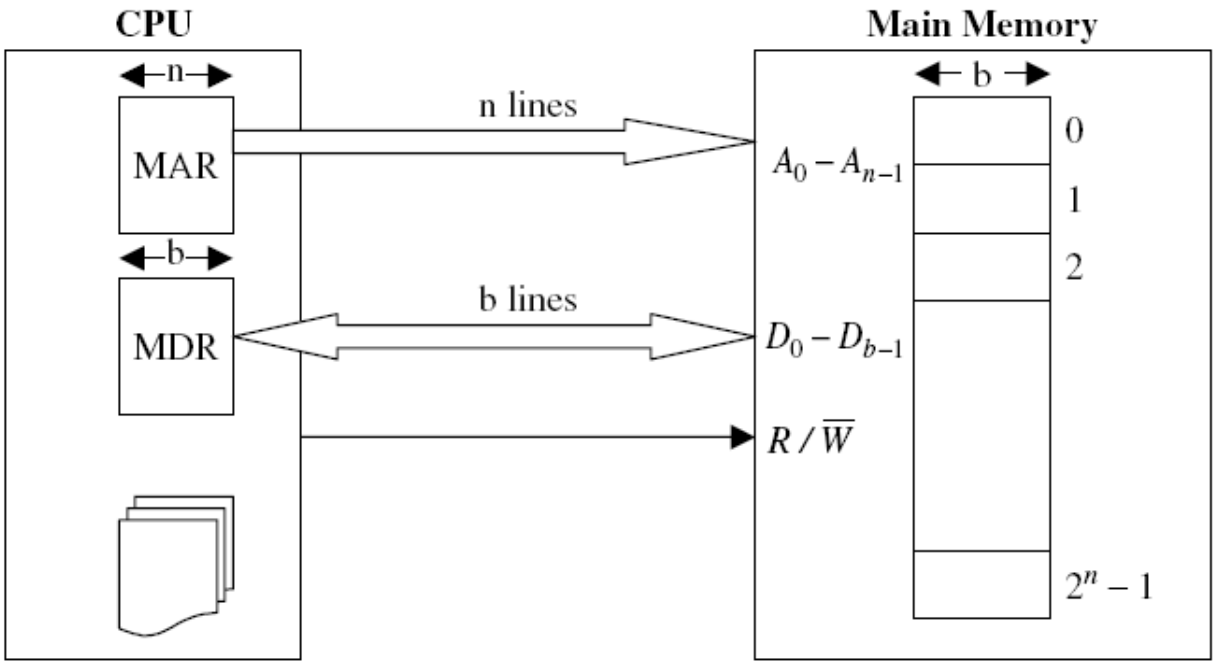
The probability of finding the requested item in the first level is called the *hit ratio*, *h1*. The probability of not finding (missing) the requested item in the first level of the memory hierarchy is called the *miss ratio*, $(1\text{-}h_1)$. When the requested item causes a "miss", it is sought in the next subsequent memory level.

The probability of finding the requested item in the second memory level, the hit ratio of the second level, is h_2. The miss ratio of the second memory level is $(1\text{-}h_2)$. The process is repeated until the item is found. Upon finding the requested item, it is brought and sent to the processor.

2. MAIN MEMORY

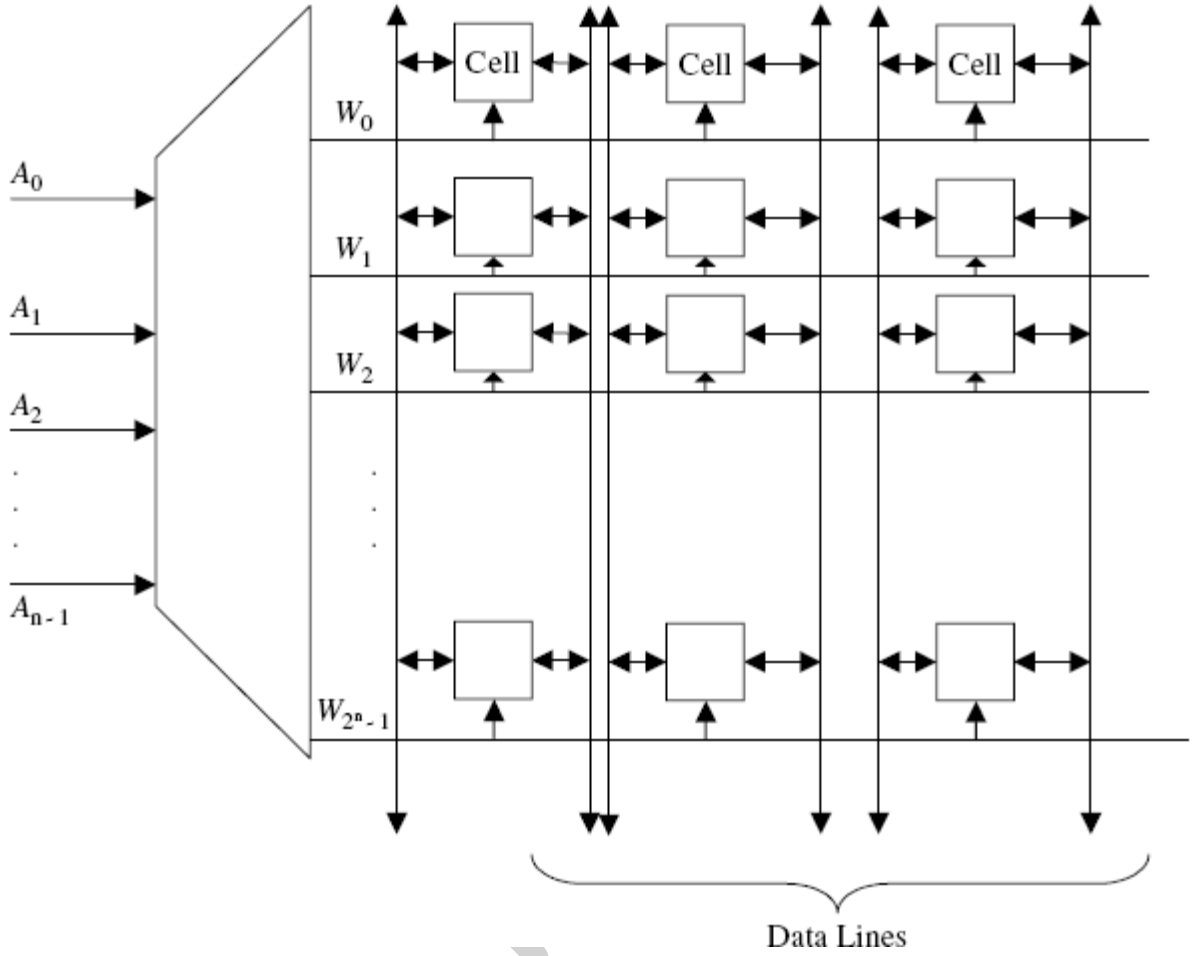
The main memory provides the main storage for a computer.

Two CPU registers are used to interface the CPU to the main memory. These are the memory address register (MAR) and the memory data register (MDR).



A typical CPU and main memory interface

It is possible to visualize a typical internal main memory structure as consisting of rows and columns of basic cells. Each cell is capable of storing one bit of information.



A conceptual internal organization of a memory chip

In this figure above, cells belonging to a given row can be assumed to form the bits of a given memory word.

Address lines $A_{n-1}A_{n-2}$ . . . A_1A_0 are used as inputs to the address decoder in order to generate the word select lines W_{2^n-1} . . . W_1W_0.

A given word select line is common to all memory cells in the same row. At any given time, the address decoder activates only one word select line while deactivating the remaining lines.

A word select line is used to enable all cells in a row for read or write.

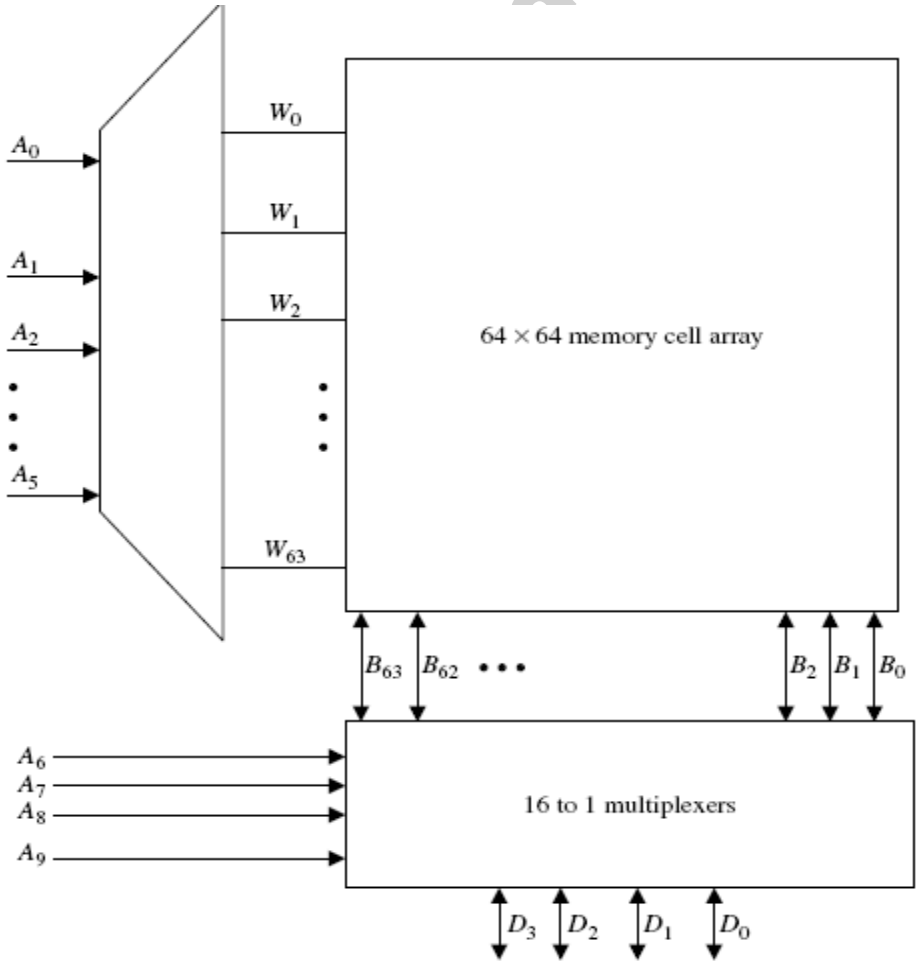
Data (bit) lines are used to input or output the contents of cells.

Each memory cell is connected to two data lines. A given data line is common to all cells in a given column.

Example: a 1K×4 memory chip. The memory array should be organized as 1K rows of cells, each consisting of four cells. The chip will then have to have 10 pins for the address and four pins for the data.

However, this may not lead to the best utilization of the chip area.

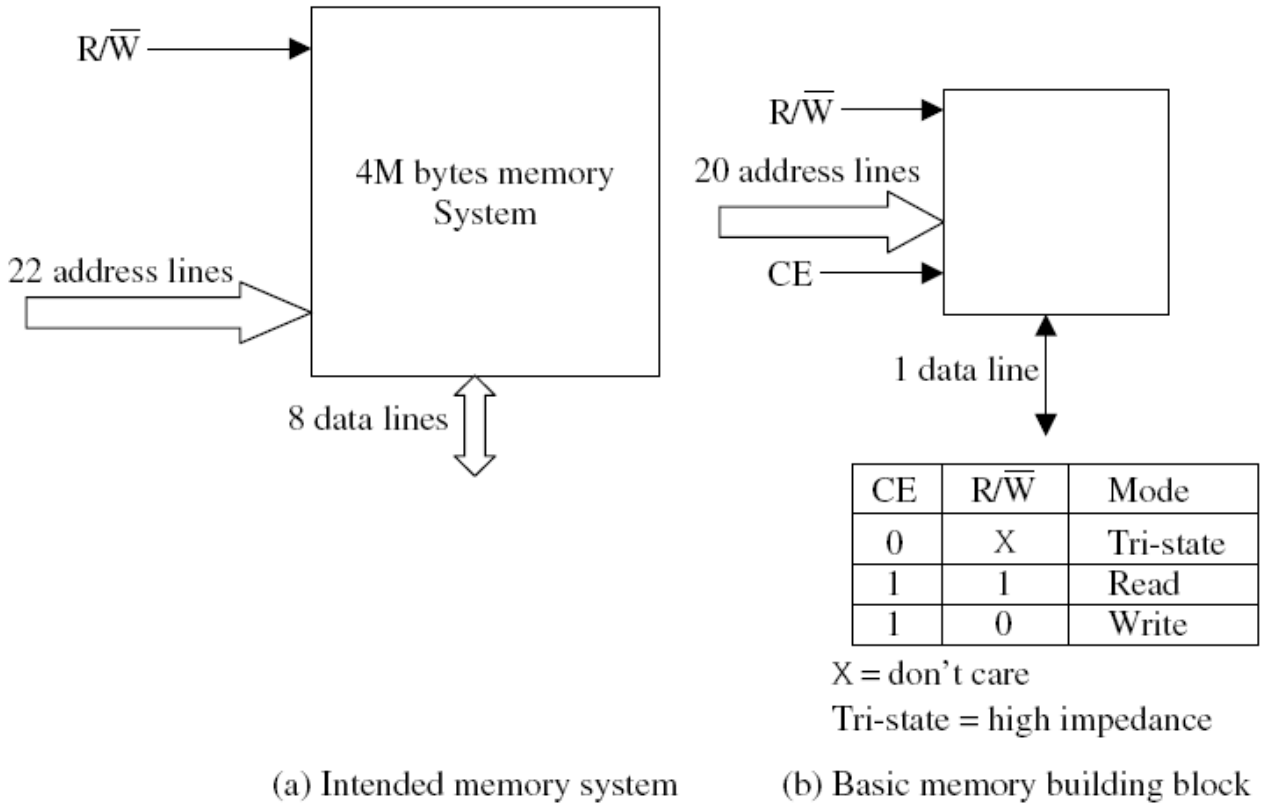
Another possible organization of the memory cell array is as a 64×64, that is, to organize the array in the form of 64 rows, each consisting of 64 cells. In this case, six address lines (forming what is called the row address) will be needed in order to select one of the 64 rows. The remaining four address lines (called the column address) will be used to select the appropriate 4 bits among the available 64 bits constituting a row.



Efficient internal organization of a 1K_4 memory chip

Example the design of a 4M bytes main memory subsystem using 1M bit chip. The number of required chips is 32 chips. It should be noted that the number of address lines required for the 4M system is 22, while the number of data lines is 8.

Figure below shows a block diagram for both the intended memory subsystem and the basic building block to be used to construct such a subsystem.



CE	R/W	Mode
0	X	Tri-state
1	1	Read
1	0	Write

Block diagram of a required memory system and its basic building block

3. VIRTUAL MEMORY

A virtual memory system attempts to optimize the use of the main memory (the higher speed portion) with the hard disk (the lower speed portion).

In effect, virtual memory is a technique for using the secondary storage to extend the apparent limited size of the physical memory beyond its actual physical size.

It is usually the case that the available physical memory space will not be enough to host all the parts of a given active program. Those parts of the program that are currently active are brought to the main memory while those parts that are not active will be stored on the magnetic disk.

If the segment of the program containing the word requested by the processor is not in the main memory at the time of the request, then such segment will have to be brought from the disk to the main memory.

The principles employed in the virtual memory design are the same as those employed in the cache memory. *The most relevant principle is that of keeping active segments in the high-speed main memory and moving inactive segments back to the hard disk*.

Movement of data between the disk and the main memory takes the form of **pages**.

A page is a collection of memory words, which can be moved from the disk to the MM when the processor requests accessing a word on that page.

A typical size of a page in modern computers ranges from 2K to 16K bytes.

The operating system is responsible for such movement of data and programs.

The address issued by the processor in order to access a given word does not correspond to the physical memory space. Therefore, such address is called *a virtual (logical) address*.

The memory management unit (MMU) is responsible for the translation of virtual addresses to their corresponding physical addresses.

Three address translation techniques can be identified. These are *direct-mapping*, *associative-mapping*, and *set-associative-mapping*.

In all these techniques, information about the main memory locations and the corresponding virtual pages are kept in a table called the *page table*.

The **page table** is stored in the main memory. Other information kept in the page table includes a bit indicating

- the validity of a page.
- modification of a page.
- the authority for accessing a page.

The *valid bit* is set if the corresponding page is actually loaded into the main memory.

Valid bits for all pages are reset when the computer is first powered on. The other control bit that is kept in the page table is the *dirty bit*. It is set if the corresponding page has been altered while residing in the main memory. If while residing in the main memory a given page has not been altered, then its dirty bit will be reset.

4. READ-ONLY MEMORY

A *volatile* storage is defined as one that loses its contents when power is turned off. Random access as well as cache memories are examples of volatile memories.

Nonvolatile memory storages are those that retain the stored information if power is turned off.

Computer system boot subroutines, microcode control, and video game cartridges are a few examples of computer software that require the use of nonvolatile storage.

Read-only memory (ROM) can also be used to realize combinational logic functions.

Mask-programmed ROMs are primarily used to store machine microcode.

Programmable ROM (PROM) instead of transistors, are placed at the intersection of word and bit lines. The user can program the PROM by selectively blowing up the appropriate fuses.

Erasable PROM (EPROM), is reprogrammable; that is, it allows stored data to be erased and new data to be stored.

Flash EPROMs (FEPROMs) have emerged as strong contenders to EPROMs. This is because FEPROMs are more compact, faster, and removable compared to EPROMs.

Erasure of the EPROM can be done electrically and, moreover, selectively; that is, only the contents of selective cells can be erased, leaving the other cells' contents untouched.

ROM type	Cost	Programmability	Typical applications
Mask-programmed ROM	Truly inexpensive	Once at manufacture	Microcode
PROM	Inexpensive	Once on site	Prototyping
EPROM	Moderate	Many times	Prototyping
FEPROM	Expensive	Many times	VCR & TVs
EEPROM	Truly expensive	Many times	VCR & TVs

5. CACHE MEMORY

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.

cache hit If the request corresponds to an element that is currently residing in the cache.

cache miss if the request corresponds to an element that is not currently in the cache.

A *cache hit ratio*, h_c, is defined as the probability of finding the requested element in the cache. A cache miss ratio (1-h_c) 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.

$$\text{Hit-ratio} = \frac{no.of\ hit}{total\ CPU\ reference\ to\ memory}$$

$$= \frac{hit}{miss+hit}$$

Example a computer with cache memory access time of 100 ns a main memory access time is 1000 ns, and hit-ratio of 0.9, compute the average access time.

Hit-ratio = $\frac{9}{10}$ hit=9 & miss= 10

Total time = 9×100 + 1×1000 + 1×1000 = 2000

Average access time = $\frac{total\ time}{10} = \frac{2000}{10}$ = 200 ns

❖ Cache Memory Organization

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

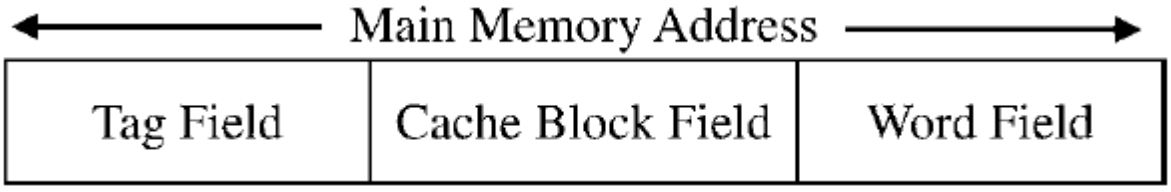
1. The criterion used to place, in the cache, an incoming block from the main memory.

2. The criterion used to replace a cache block by an incoming block (on cache full).

(A) 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.

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 below. The lengths, in bits, of each of the fields:



Direct-mapped address fields

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 = $log_2 (B \times M)$

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 a main memory consisting of 4K blocks, a cache memory consisting of 128 blocks, and a block size of 16 words.

Word field = $\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits

Block field = $\log_2 N = \log_2 128 = \log_2 2^7 = 7$ bits

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

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

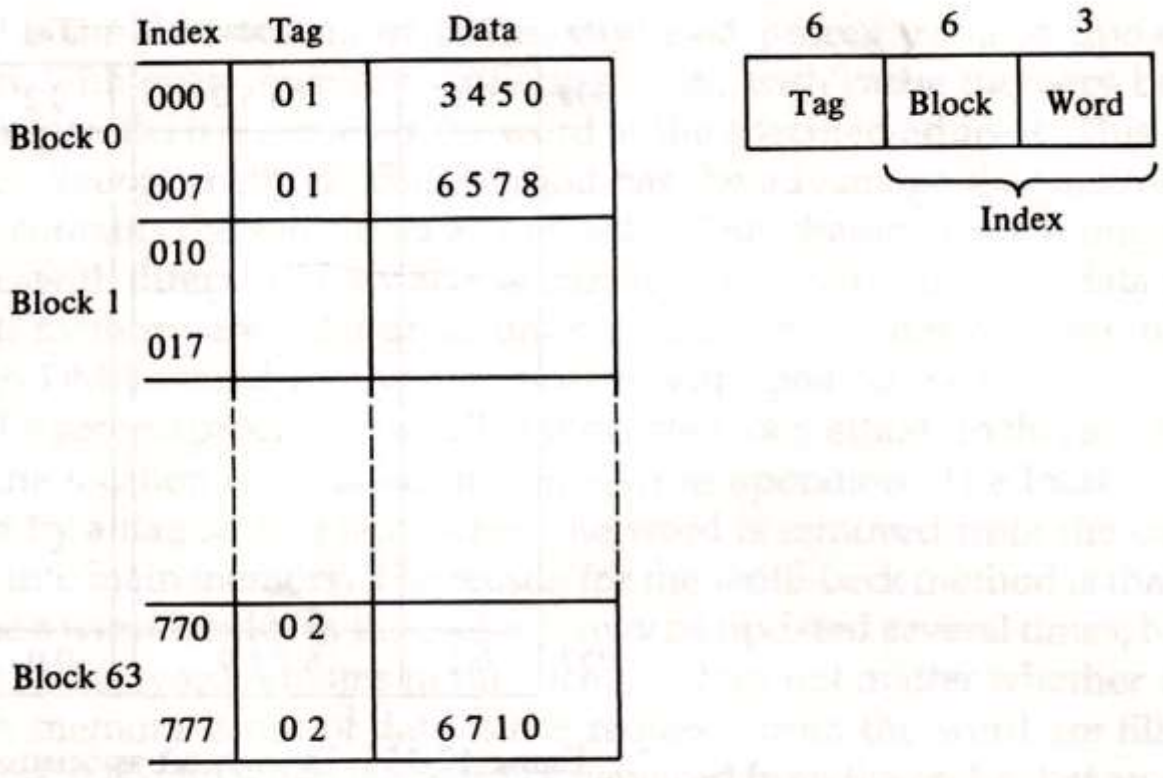
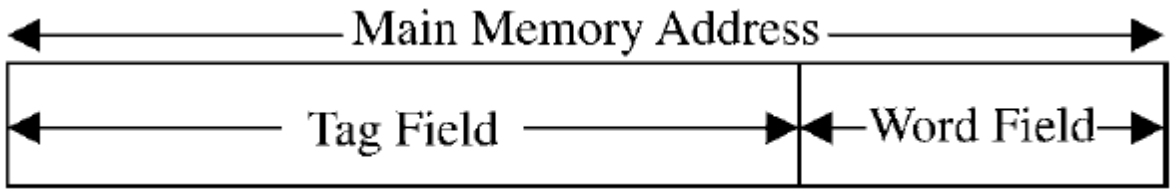


Figure 12-14 Direct mapping cache with block size of 8 words.

(B) 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 below:



Associative-mapped address fields

1. Word field = $log_2 B$, where B is the size of the block in words

2. Tag field= $log_2 M$, where M is the size of the main memory in blocks

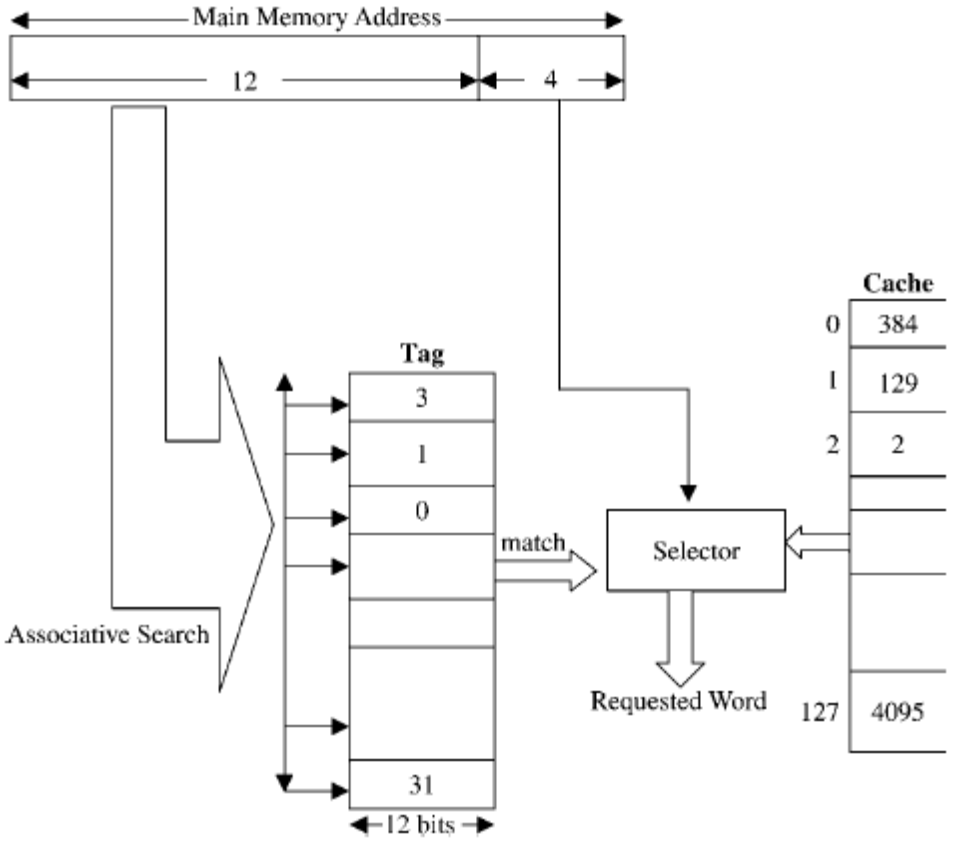
3. The number of bits in the main memory address = $log_2 (B \times M)$

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.

Word field = $\log_2 B = \log_2 16 = \log_2 2^4 = 4$ bits

Tag field = $\log_2 M = \log_2 2^7 \times 2^{10} = 12$ bits

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



Associative-mapped address translation

(C) Set-Associative Mapping

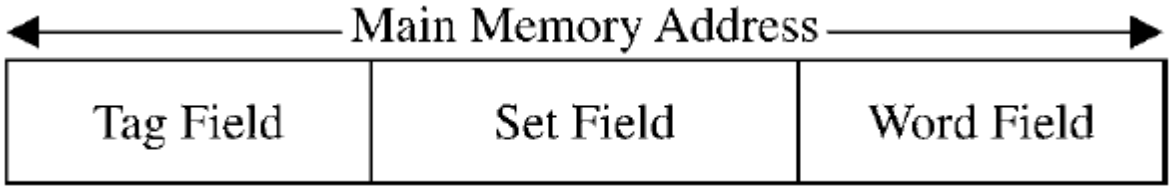
In the set-associative mapping technique, the cache is divided into a number of sets.

Each set consists of a number of blocks.

A given main memory block maps to a specific cache set based on the equation $s = i \bmod S$, 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.

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 below:



Set-associative-mapped address fields

1. Word field = $log_2 B$, where B is the size of the block in words

2. Set field = $log_2 S$, where S is the number of sets in the cache

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

$S = N/B_s$, where N is the number of cache blocks and B_s is the number of blocks per set

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

Example 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

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

H.W a cache memory size is 4096 word, and MM size is 256Kbyte. If each block has 16 words.

a- Give the size of cache memory required if use

1)Associative mapping 2)Direct mapping 3)Two-Set Associative mapping

b- Give the address format for MM if you use direct mapping.

c- How many no. of blocks in cache and MM.

H.W a digital computer has memory unit of 64kx16 and the cache memory of 1k word. The cache uses direct mapping with a block size of 4 word.

a- How many bits are there in the tag, index, block and word fields of the address format.

b- How many bits are there in each word of cache, and how are they divided into functions.

c- How many blocks can the cache accommodate.