UNIVERSITY OF BABYLON
COLLEGE OF SCIENCE FOR WOMEN
COMPUTER DEPARTMENT

2024/ 2025
FIRST CLASS

Computer Skills

LECTURES

PREPARED BY:
Assist. Prof. Dr. Ahmed Mohammed Hussein

2024-2025

❖ LEARNING OBJECTIVES

After completion of this lecture, you should be able to:

- Describe the Memory Hierarchy.
- Describe the Cache Memory.
- Describe the Virtual Memory.

1. MEMORY HIERARCHY

Memory in a conventional digital computer is organized in a hierarchy as illustrated in Figure 1. At the top of the hierarchy are registers that are matched in speed to the CPU, but tend to be large and consume a significant amount of power. There are normally only a small number of registers in a processor, on the order of a few hundred or less. At the bottom of the hierarchy are secondary and off-line storage memories such as hard magnetic disks and magnetic tapes, in which the cost per stored bit is small in terms of money and electrical power, but the access time is very long when compared with registers. Between the registers and secondary storage are a number of other forms of memory that bridge the gap between the two.

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 slower level the **secondary 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. Among these parameters are ***the access type, capacity, cycle time, latency, bandwidth, and cost***. 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 ***cycle time*** is defined as the time elapsed from the

start of a read operation to the start of a subsequent read. 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. Figure 1 depicts a typical memory hierarchy. Table 1 provides typical values of the memory hierarchy parameters.

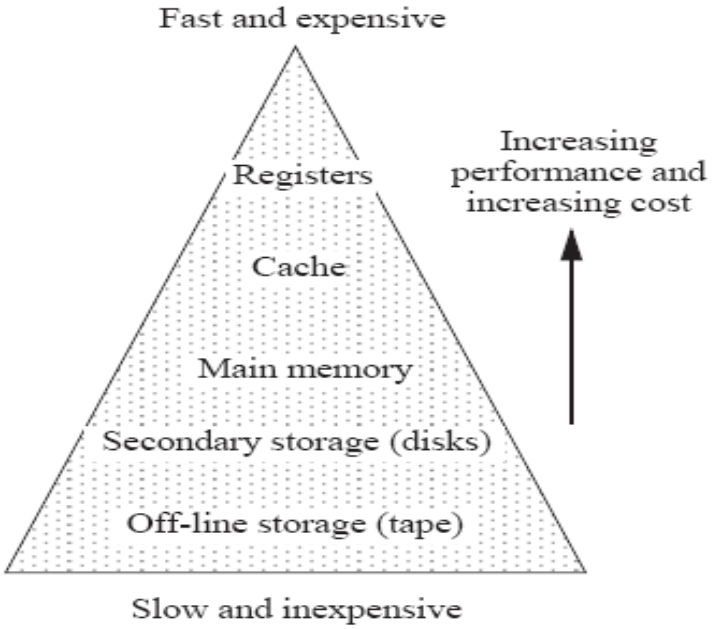


FIGURE 1.THE MEMORY HIERARCHY.

TABLE 1 MEMORY HIERARCHY PARAMETERS

	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

2. CACHE MEMORY

Memory access is generally *slow* when compared with the speed of the central processing unit (CPU), and so the memory poses تشكل a significant مأزق in computer performance. Processing speed is mostly limited by the speed of main memory. If the average memory access time can be reduced, this would reduce the total execution time of the program. In many systems it is not economical اقتصادي to use high speed memory devices for all of the internal memory. Instead, A small but fast **cache memory (**fast SRAM**)**, in which the contents of the most commonly accessed locations are maintained, can be placed between the main memory and the CPU. The idea behind using a cache 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**).

Cache memory is much smaller than the main memory. Usually implemented using SRAMs, which sits تقع between the processor and main memory in the memory hierarchy. The cache effectively isolates the processor from the slowness of the main memory, which is DRAM-based. The principle behind the cache memories is to prefetch جلب مسبق the data from the main memory before the processor needs them. If we are successful in predicting تخمين the data that the processor needs in the near future, we can preload تحميل مسبق the cache and supply those data from the faster cache. It turns out that predicting the processor future access requirements is not difficult owing to a phenomenon ظاهرة known as *locality of reference* that programs exhibit.

A typical situation is shown in Figure 2. A simple computer without a cache memory is shown in the left side of the figure. This cache-less computer contains a CPU that has a clock speed of 400 MHz, but communicates over a 66 MHz bus to a main memory that supports a lower clock speed of 10 MHz. A few bus cycles are normally needed to synchronize the CPU with the bus, and thus the difference in speed between main memory and the CPU can be as large as a factor of ten or more. A cache memory can be positioned closer to the CPU as shown in the right side of Figure 2, so that the CPU sees fast accesses over a 400 MHz direct path to the cache.

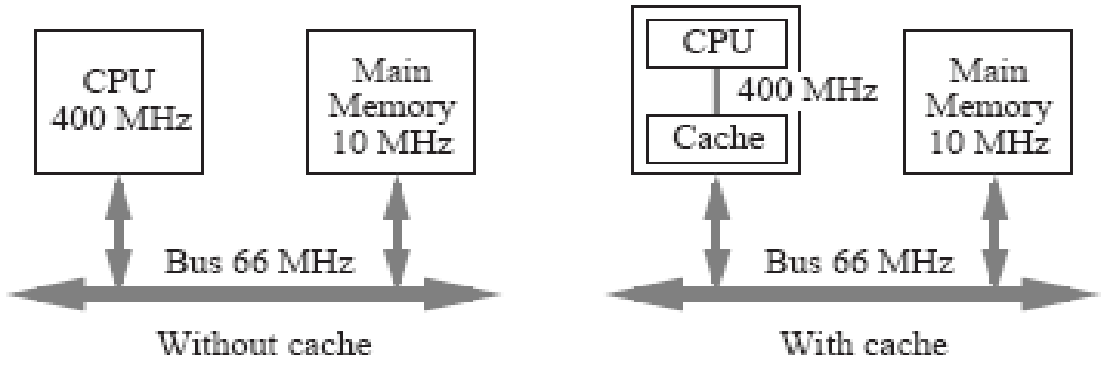
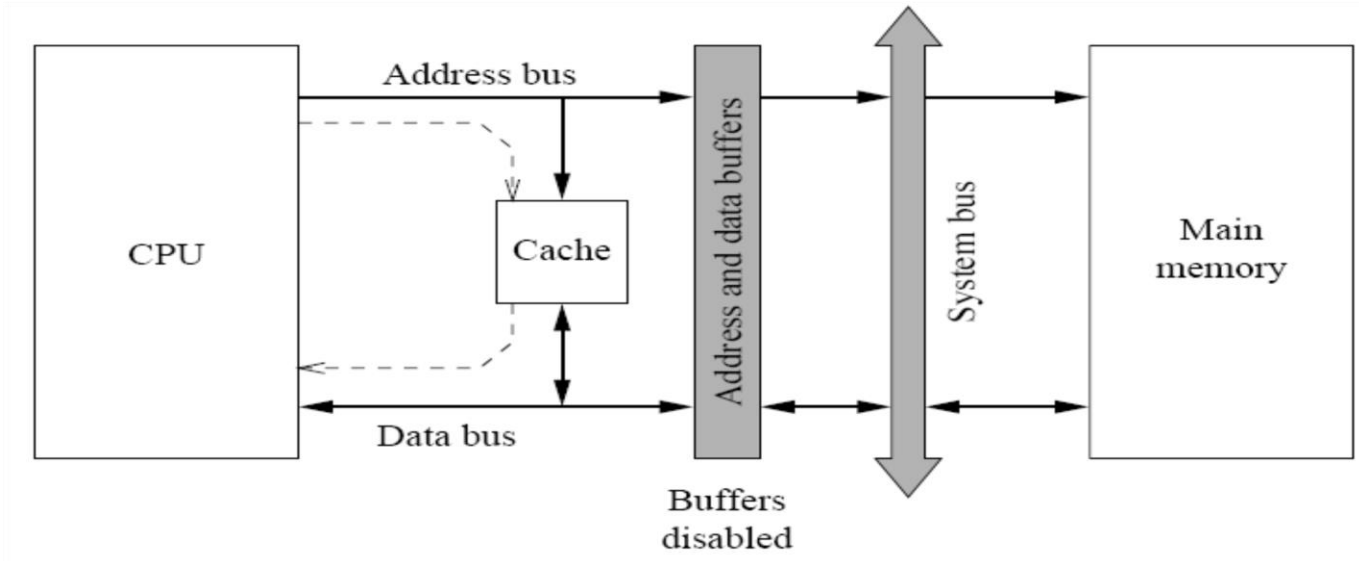


FIGURE 2.PLACEMENT OF CACHE IN A COMPUTER SYSTEM.

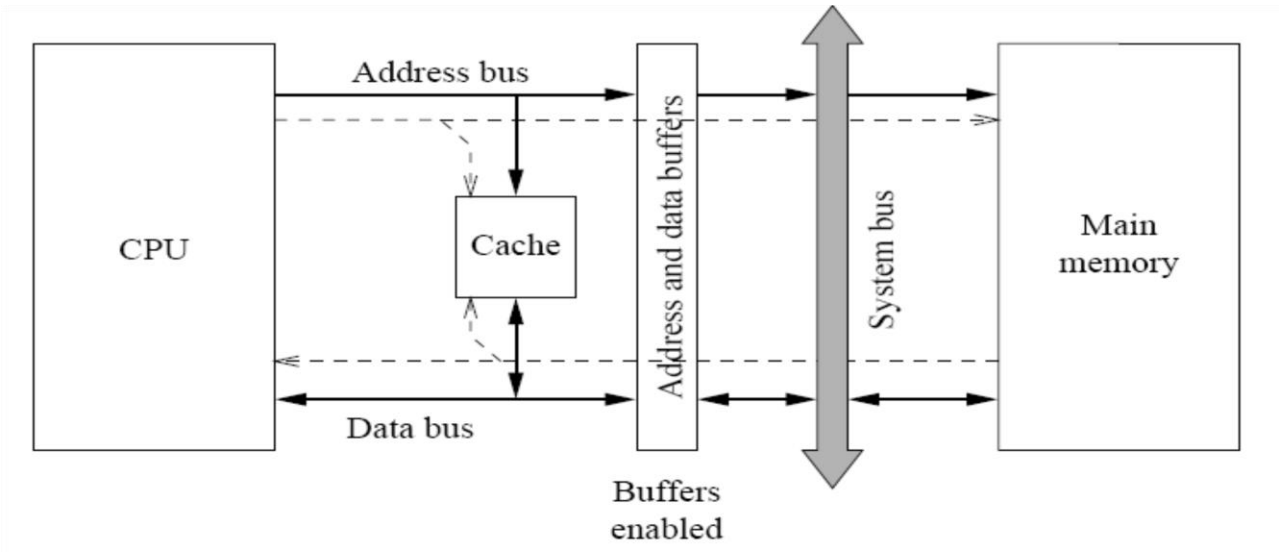
Performance of cache systems can be measured using the *hit rate* معدل النجاحات or *hit ratio*. When the processor makes a request for a memory reference, the request is first sought تبحث in the cache. If the request corresponds to an element that is currently residing in the cache, we call that a ***cache 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, hc, is defined as the probability احتمالية of finding the requested element in the cache. A cache miss ratio (1 - hc) is defined as the probability of not finding the requested element in the cache. The likelihood احتمال of the processor finding what it wants in cache a high as 95 percent.

Figure 3 shows how memory read operations are handled to include the cache memory. When the data needed by the processor are in the cache memory ("read hit"), the requested data are supplied by the cache (see Figure 3a).

The dashed line indicates the flow of information in the case of a read hit. In the case of a read miss, we have to read the data from the main memory. While supplying the data from the main memory, a copy is also placed in the cache as indicated by the bottom dashed lines in Figure 3b. Reading data on a cache miss takes more time than reading directly from the main memory as we have to first test to see if the data are in the cache and also consider the overhead involved in copying the data into the cache. This additional time is referred to as the *miss penalty*.



(a) Read hit



(a) Read miss

FIGURE 3.DETAILS OF CACHE READ OPERATION.

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 (a ***block*** as a minimum unit of data transfer). 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. The transferring of data is referred to as a mapping process.

Modern memory systems may have several levels of cache, referred to as Level 1 (L1), Level 2 (L2), and even, in some cases, Level 3 (L3). In most instances the L1 cache is implemented right on the CPU chip. Both the Intel Pentium and the IBM-Motorola PowerPC G3 processors have 32 Kbytes of L1 cache on the CPU chip. Processors will often have two separate L1 caches, one for instructions (**I-cache**) and one for data (**D-cache**). A level 2 (L2) high speed memory cache store up to 512 kilobytes of data.

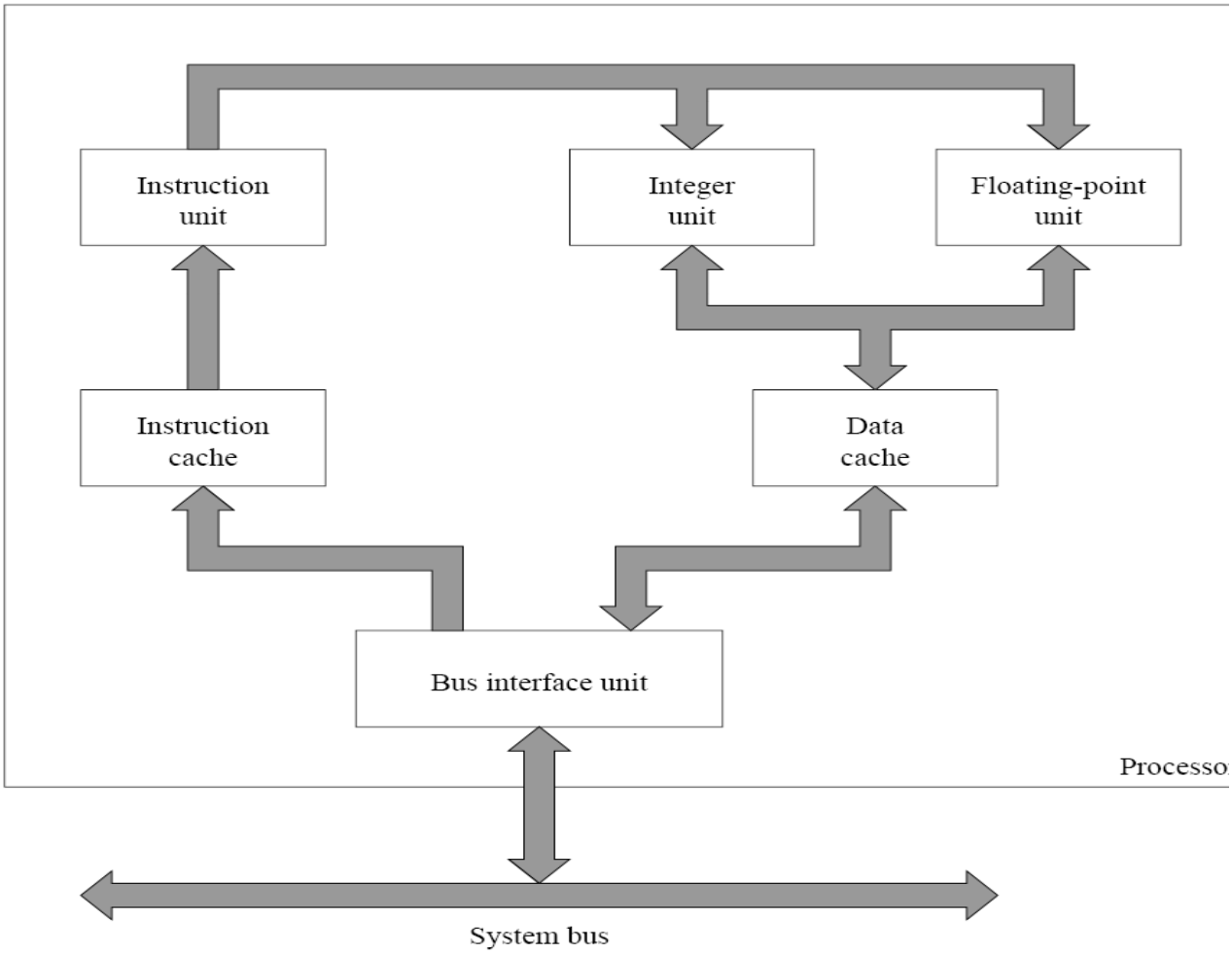


FIGURE 4. MOST CURRENT PROCESSORS USE SEPARATE CACHES FOR INSTRUCTIONS AND DATA WITH SEPARATE INSTRUCTION AND DATA BUSES.

2.1. 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).

2.1.1. 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 5 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 many-to-one mapping technique.

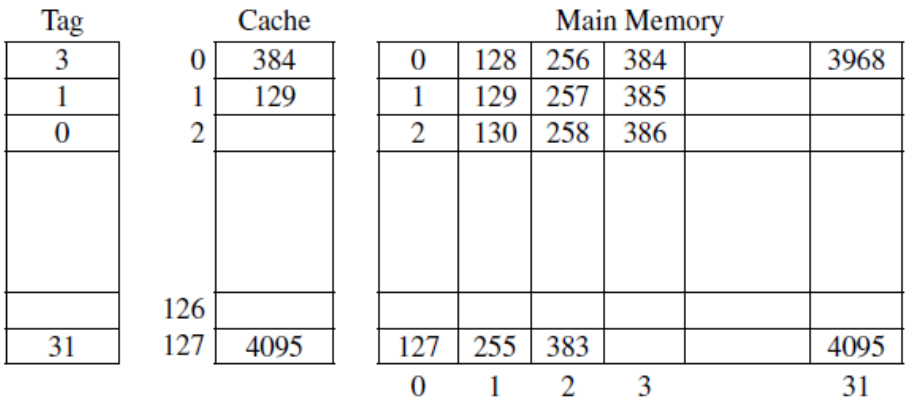


FIGURE 5. MAPPING MAIN MEMORY BLOCKS TO CACHE BLOCKS

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 غير كفوّة 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 address issued by the processor interprets by dividing the address into three fields as shown in Figure 6. The lengths, in bits, of each of the fields in Figure 6 are:

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.

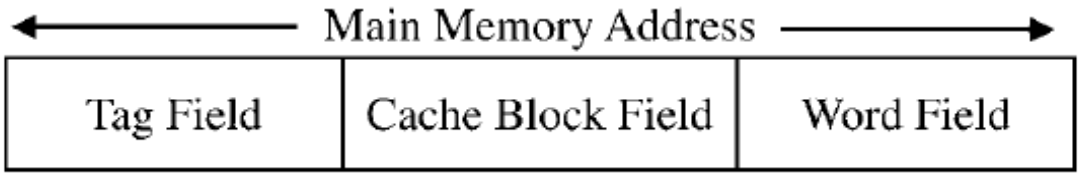


FIGURE 6.DIRECT-MAPPED ADDRESS FIELDS

EXAMPLE 2: Compute the above four parameters for Example 1.

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.

2.1.2. 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 address issued by the processor is interpreted by dividing it into two fields as shown in Figure 7. The length, in bits, of each of the fields in Figure 7 are given by:

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 = $\log_2 (B \times M)$

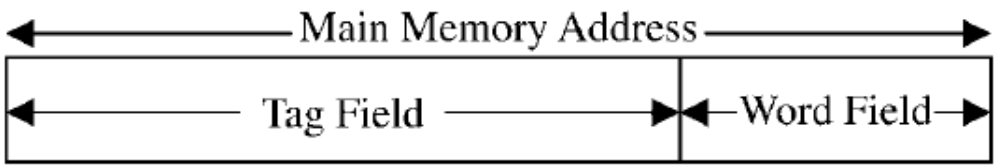


FIGURE 7. ASSOCIATIVE-MAPPED ADDRESS FIELDS

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.

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

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

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

2.1.3. 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 address issued by the processor is interprets by dividing it into three fields as shown in Figure 8. The length, in bits, of each of the fields of Figure 8 is given by

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 × M)

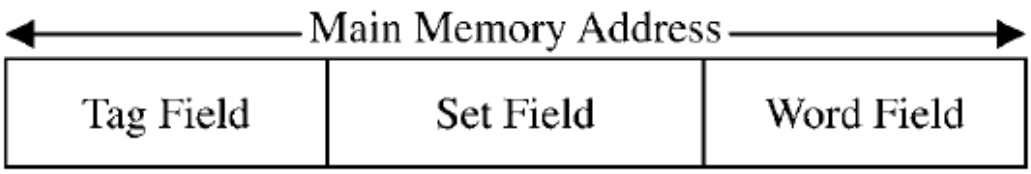


FIGURE 8.SET-ASSOCIATIVE-MAPPED ADDRESS FIELDS

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 = log2 16 = log2 2^4 = 4 bits

2. Set field = $\log_2$ 32 = 5 bits

3. Tag field = $\log_2$ (4 × 2^{10}/32) = 7 bits

3. VIRTUAL MEMORY

When you write programs, you are not really concerned with the amount of memory available on your system to run the program. What if your program requires more memory to run than is available on your machine? This is not a theoretical question, in spite of the amount of memory available on current machines. Even on a single-user system, not all the memory is available for your program. The operating system takes quite a big chunk كمية كبيرة of it. If you consider a time-shared multiprogrammed system, the problem becomes even more serious. Virtual memory was proposed to deal with this problem.

Virtual memory was developed to eliminate the physical memory size restriction mentioned before. There are some similarities between the cache memory and virtual memory. Just as with the cache memory, we would like to use the relatively small main memory and create the illusion خداع (to the programmer) of a much larger memory, as shown in Figure 9. The programmer is concerned only with the virtual address space. Programs use virtual addresses and when these programs are run, their virtual addresses are mapped to physical addresses at run time.

The illusion of larger address space is realized by using much slower disk storage. Virtual memory can be implemented by devising an appropriate mapping function between the virtual and physical address spaces. As a result of this similarity between cache and virtual memories, both memory system designs are based on the same underlying principles. The success of the virtual memory in providing larger virtual address space also depends on the locality we mentioned before.

Before the virtual memory technique was proposed, one had to resort to a technique known as overlaying in order to run programs that did not fit into the physical memory. With only a tiny amount of memory available (by current standards) on the earlier machines, only a simple program could fit into the memory. In this technique, the programmer divides the program into several chunks, each of which can fit in the memory. These chunks are known as ***overlays*** *تراكبات*. The whole program (i.e., all **overlays**) resides in the disk. The programmer is responsible for explicitly managing the overlays. Typically, when an overlay in the memory is finished, it will bring in the next overlay that is required for program execution. Needless to sayوغني عن القول, this is not something a programmer would like to doلا يفضله المبرمج. Virtual memory automates the management of overlays without requiring any help from the programmer.

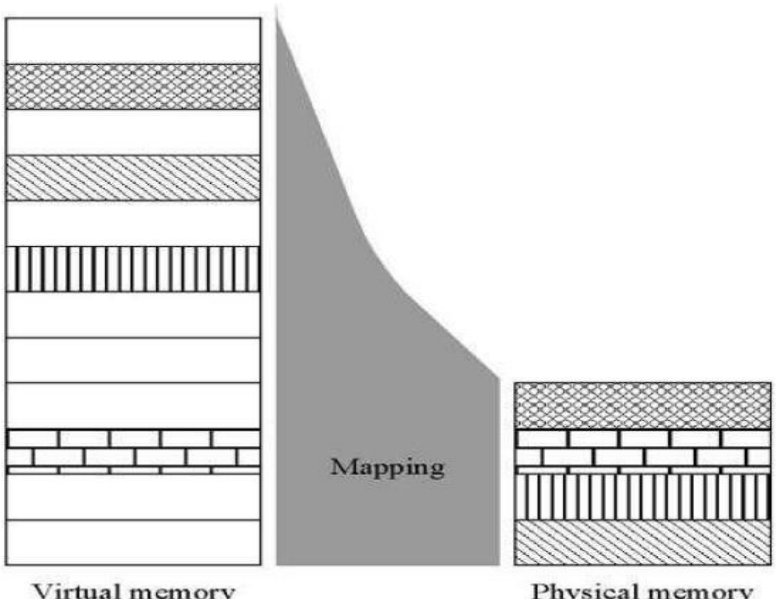


FIGURE 9.VIRTUAL MEMORY CREATES THE ILLUSION OF A MUCH LARGER MEMORY TO THE PROGRAMMER THAN WHAT IS PHYSICALLY AVAILABLE IN THE SYSTEM.

Many of the concepts we use here are similar to the concepts used in cache systems. Caches use a small amount of fast memory but to a program it appears as a large amount of fast memory. As mentioned before, virtual memory also provides a similar illusion. As a result of this similarity, the principles involved are the same. The details, however, are quite different مختلفة تماما because the motivations الدوافع for cache memory and virtual memory are different. We use cache memory to improve performance whereas virtual memory is necessary to run programs in a small amount of memory.

Virtual memory implements a mapping function between a much larger virtual address space and the physical memory address space. For example, in the PowerPC a 48-bit virtual address is translated into a 32-bit memory address. On the other hand, the Pentium uses 32-bit addresses for both virtual and physical memory addresses.