C h a p t e r **4 CACHE MEMORY**

➢ The complex subject of computer memory is made more manageable if we classify memory systems according to their key characteristics. The most important of these are listed in Table 4.1

An obvious **characteristic of memory** is its **capacity**. For internal memory, this is typically expressed in terms of bytes (1 byte = 8 bits) or words. Common word lengths are 8, 16, and 32 bits. External memory capacity is typically expressed in terms of bytes.

Table 4.1 Key Characteristics of Computer Memory Systems

Location	**Performance**
Internal (e.g. processor registers, main memory, cache)	Access time
External (e.g. optical disks, magnetic disks, tapes)	Cycle time
Capacity	Transfer rate
Number of words	**Physical Type**
Number of bytes	Semiconductor
Unit of Transfer	Magnetic
Word	Optical
Block	Magneto-optical
Access Method	**Physical Characteristics**
Sequential	Volatile/nonvolatile
Direct	Erasable/nonerasable
Random	**Organization**
Associative	Memory modules

A related concept is the **unit of transfer**. For internal memory, the unit of transfer is equal to the number of electrical lines into and out of the memory module. This may be equal to the word length, but is often larger, such as 64, 128, or 256 bits.

Word: The "natural" unit of organization of memory. The size of the word is typically equal to the number of bits used to represent an integer and to the instruction length.

Addressable units: In some systems, the addressable unit is the word. However, many systems allow addressing at the byte level. Another distinction among memory types is the **method of accessing** units of data. These include the following:

Sequential access: Memory is organized into units of data, called records. Access must be made in a specific linear sequence. Stored addressing information is used to separate records and assist in the retrieval process.

Direct access: As with sequential access, direct access involves a shared read–write mechanism.

Random access: Each addressable location in memory has a unique, physically wired-in addressing mechanism.
Associative: This is a random access type of memory that enables one to make a comparison of desired bit locations within a word for a specified match, and to do this for all words simultaneously.

- **Memory hierarchy**

A **memory hierarchy** a typical hierarchy is illustrated in Figure 4.1. As one goes down the hierarchy, the following occur:
a. Decreasing cost per bit
b. Increasing capacity
c. Increasing access time
d. Decreasing frequency of access of the memory by the processor

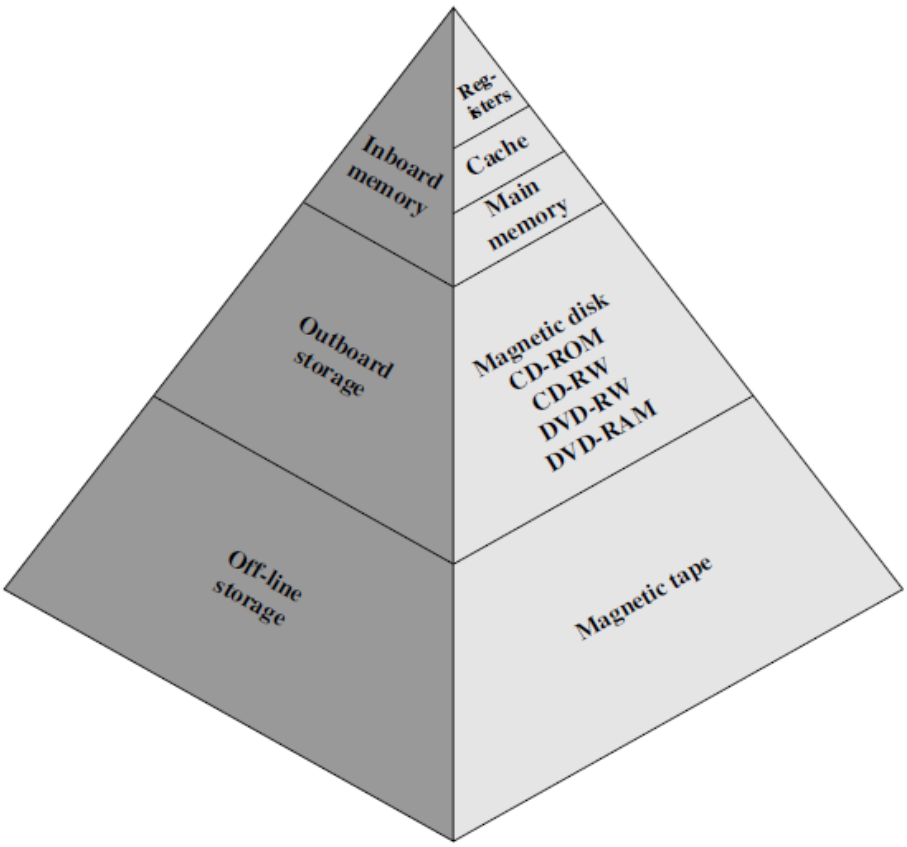
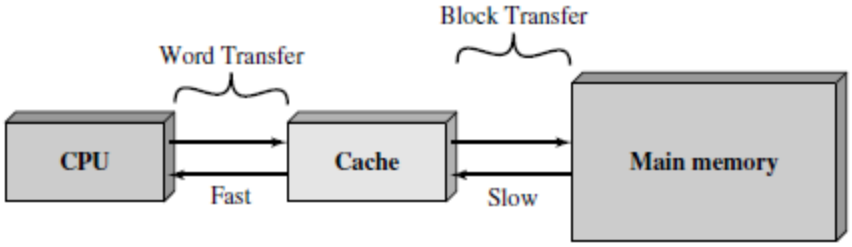


Figure 4.1 The Memory Hierarchy

Figure 4.1 the fastest, smallest, and most expensive type of memory consists of the registers internal to the processor. Typically, a processor will contain a few dozen such registers, although some machines contain hundreds of registers. Skipping down two levels, main memory is the principal internal memory system of the computer. Each location in main memory has a unique address. Main memory is usually extended with a higher-speed, smaller cache. The cache is not usually visible to the programmer or, indeed, to the processor

- **Cache memory Principles**

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories. The concept is illustrated in Figure 4.3a. There is a relatively large and slow main memory together with a smaller, faster cache memory. The cache contains a copy of portions of main memory



(a) Single cache

- **Typical Cache Organization**

Figure 4.6, 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.

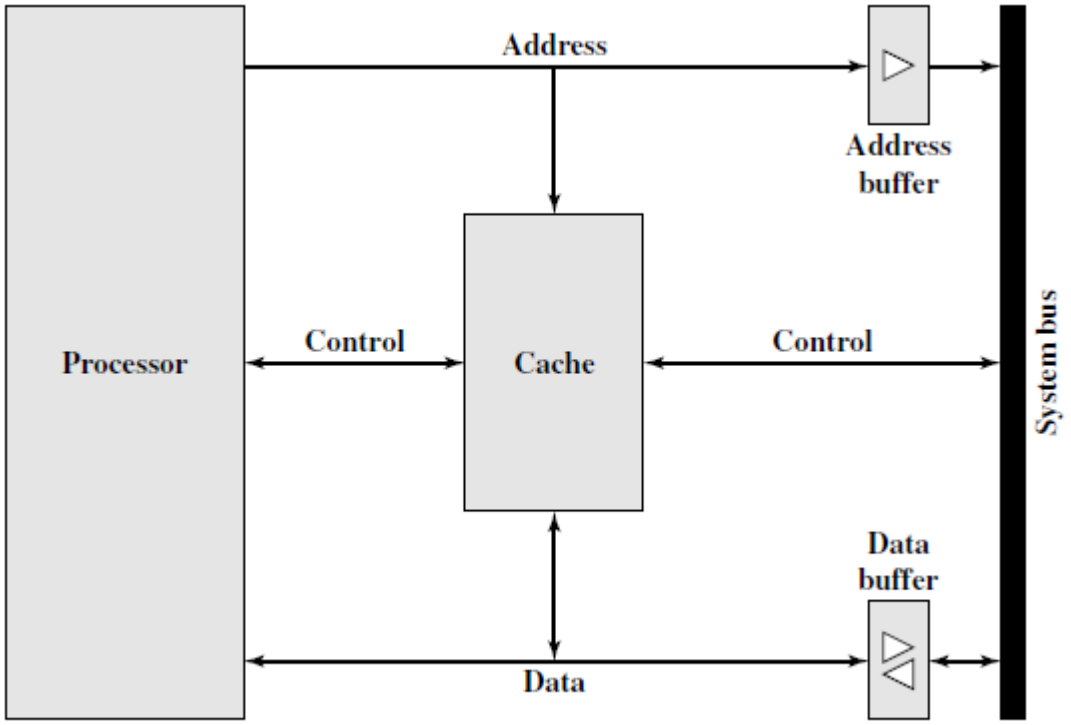


Figure 4.6 Typical Cache Organization

- **Elements of cache design**

Table 4.2 Elements of Cache Design

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	Write once
Mapping Function	**Line Size**
Direct	**Number of caches**
Associative	Single or two level
Set Associative	Unified or split
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Cache Address

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**.

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.

Mapping Function 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 organized. Three techniques can be used: direct, associative, and set associative

For the associative and set associative techniques, a replacement algorithm is needed. To achieve high speed, such an algorithm must be implemented in hardware. Another possibility is first-in-**first-out (FIFO):** Replace that block in the set that has been in the cache longest. FIFO is easily implemented as a round-robin or circular buffer technique.

Still another possibility is **least frequently used (LFU):** Replace that block in the set that has experienced the fewest references. LFU could be implemented by associating a counter with each line. A technique not based on usage (i.e., not LRU, LFU, FIFO, or some variant) is to pick a line at random from among the candidate lines. Simulation studies have shown that random replacement provides only slightly inferior performance to an algorithm based on usage

The simplest technique is called **write through**. Using this technique, all write operations are made to main memory as well as to the cache, ensuring that main memory is always valid. An alternative technique, known as **write back**, minimizes memory writes. With write back, updates are made only in the cache

Line size
Another design element is the line size. When a block of data is retrieved and placed in the cache, not only the desired word but also some number of adjacent words is retrieved.

Number of caches
When caches were originally introduced, the typical system had a single cache. More recently, the use of multiple caches has become the norm. Two aspects of this design issue concern the number of levels of caches and the use of unified versus split caches.

- **Fully Associative Mapping**

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 4.8b). In this case, the cache control logic interprets a memory address simply as a Tag and a Word field. 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 4.11 illustrates the logic. Note that no field in the address corresponds to the line number, so that the number of lines in the cache is not determined by the address format. To summarize,

- Address length = $(s + w)$ bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^w words or bytes
- Number of blocks in main memory = $\frac{2^{s+w}}{2^w} = 2^s$
- Number of lines in cache = undetermined
- Size of tag = s bits

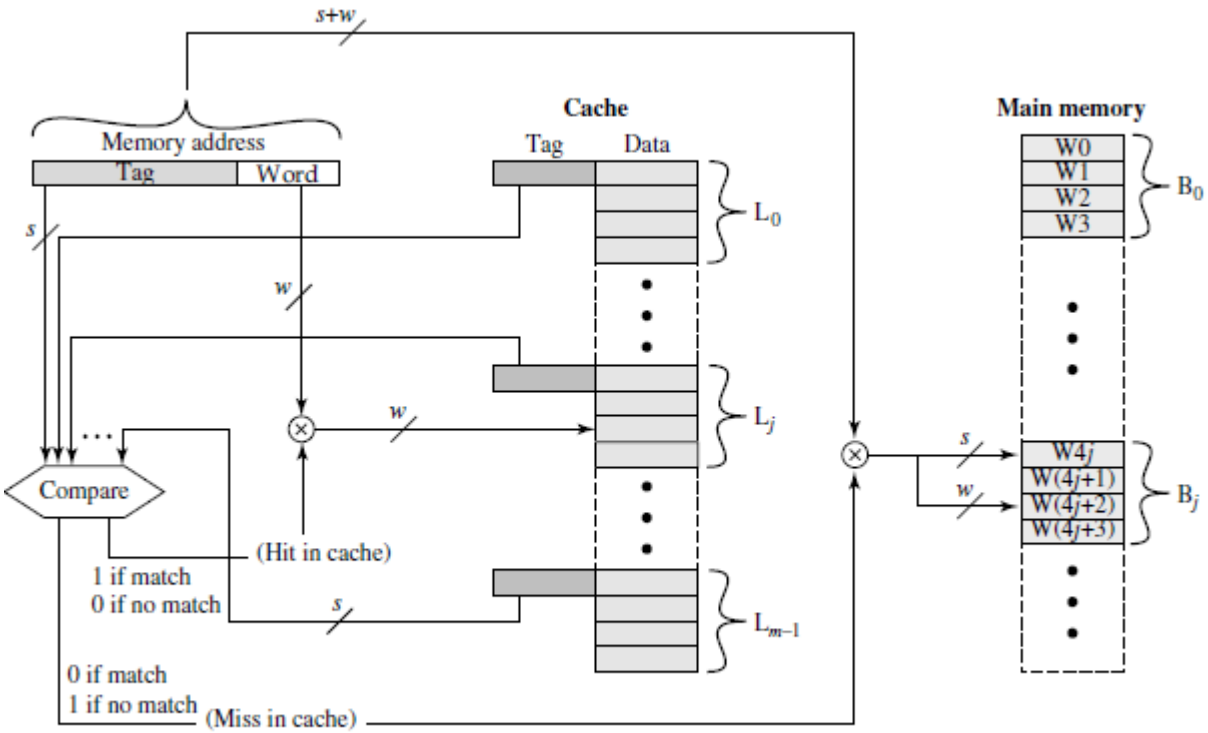


Figure 4.11 Fully Associative Cache Organization

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 where
i =cache line number
j =main memory block number
m =number of lines in the cache

The mapping function is easily implemented using the main memory address. Figure 4.9 illustrates the general mechanism. For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory; in most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of $s - r$ bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m = 2^r$ lines of the cache. To summarize,

- Address length $= (s + w)$ bits
- Number of addressable units $= 2^{s+w}$ words or bytes
- Block size $=$ line size $= 2^w$ words or bytes
- Number of blocks in main memory $= \dfrac{2^{s+w}}{2^w} = 2^s$
- Number of lines in cache $= m = 2^r$
- Size of cache $= 2^{r+w}$ words or bytes
- Size of tag $= (s - r)$ bits

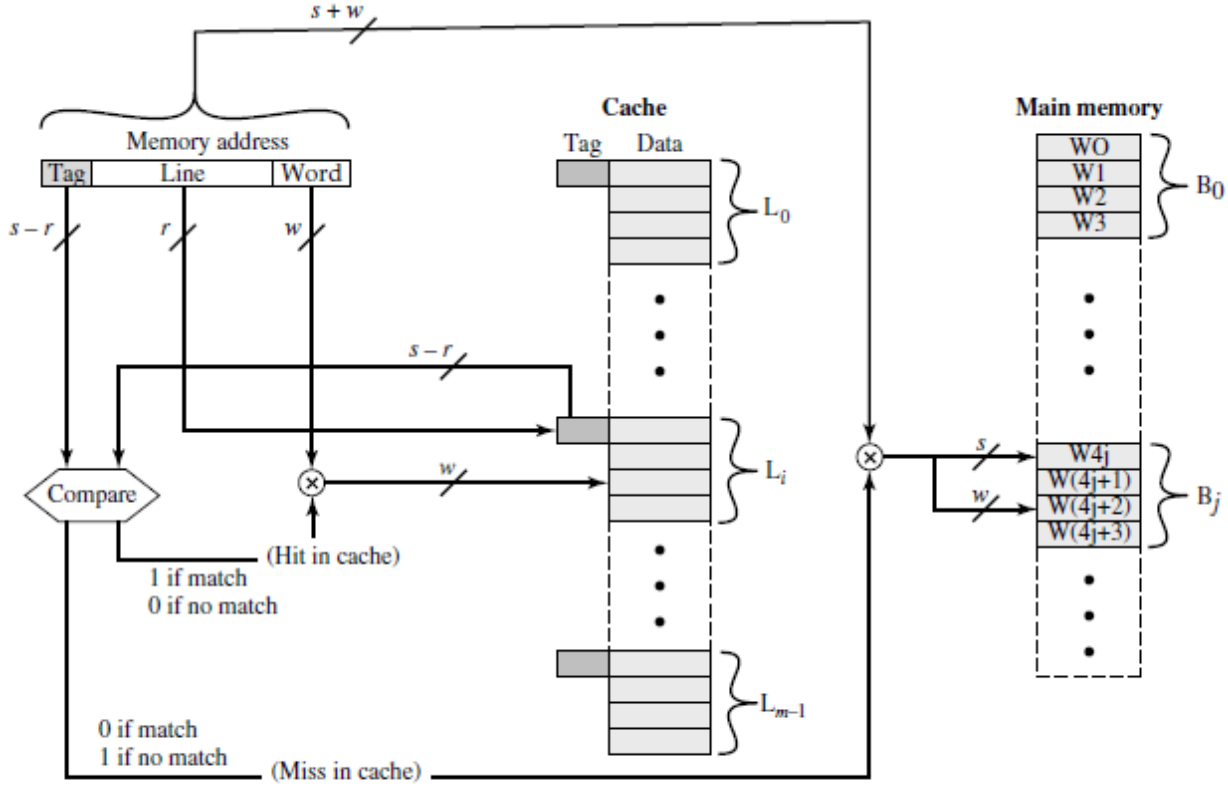


Figure 4.9 Direct-Mapping Cache Organization

➢ **Pentium 4 block diagram**

Figure 4.18 provides a simplified view of the Pentium 4 organization, highlighting the placement of the three caches. The processor core consists of four major components:

• **Fetch/decode unit:** Fetches program instructions in order from the L2 cache, decodes these into a series of micro-operations, and stores the results in the L1 instruction cache.

• **Out-of-order execution logic:** Schedules execution of the micro-operations subject to data dependencies and resource availability; thus, micro-operations may be scheduled for execution in a different order than they were fetched from the instruction stream. As time permits, this unit schedules speculative execution of micro-operations that may be required in the future.

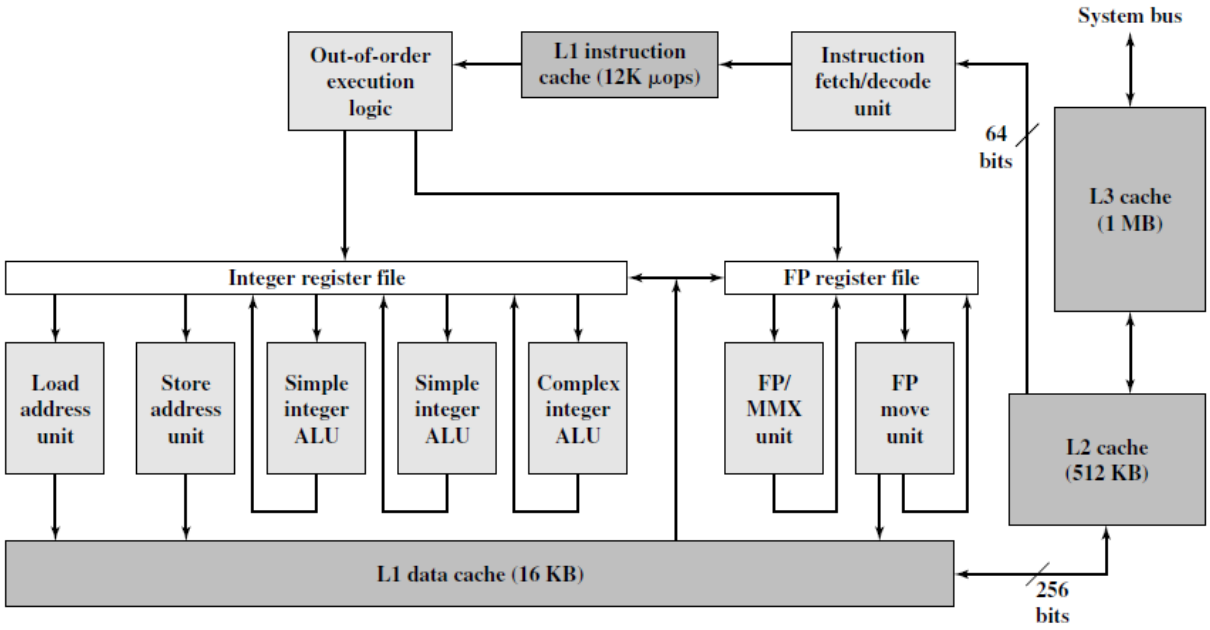


Figure 4.18 Pentium 4 Block Diagram

• **Execution units:** These units execute micro-operations, fetching the required data from the L1 data cache and temporarily storing results in registers.

• **Memory subsystem:** This unit includes the L2 and L3 caches and the system bus, which is used to access main memory when the L1 and L2 caches have a cache miss and to access the system I/O resources.