CHAPTER ONE

1.1 Organization and Architecture

Computer architecture refers to those attributes of a system visible to a programmer or, put another way, those attributes that have a direct impact on the logical execution of a program. A term that is often used interchangeably with computer architecture is instruction set architecture (ISA) . The ISA defines instruction formats, instruction opcodes, registers, instruction and data memory; the effect of executed instructions on the registers and memory; and an algorithm for controlling instruction execution.

Computer organization refers to the operational units and their interconnections that realize the architectural specifications. Examples of architectural attributes include the instruction set, the number of bits used to represent various data types (e.g., numbers, characters), I/O mechanisms, and techniques for addressing memory.

1.2 Structure and Function

A hierarchical system is a set of interrelated subsystems; each subsystem may, in turn, contain lower level subsystems, until we reach some lowest level of elementary subsystem. At each level, the system consists of a set of components and their interrelationships. At each level, the designer is concerned with structure and function:

Structure: The way in which the components are interrelated.
Function: The operation of each individual component as part of the structure.

Function

Both the structure and functioning of a computer are, in essence, simple. In general terms, there are only four basic functions that a computer can perform:

1. Data processing
2. Data storage
3. Data movement.
4. Control

Structure

We now look in a general way at the internal structure of a computer. We begin with a traditional computer with a single processor that employs a microprogrammed control unit, then examine a typical multicore structure.

1. ***SIMPLE SINGLE-PROCESSOR COMPUTER***

   Figure 1.1 provides a hierarchical view of the internal structure of a traditional single-processor computer. There are four main structural components:

   - Central processing unit (CPU):
   - Main memory:
   - System interconnection:

However, for our purposes, the most interesting and in some ways the most complex component is the CPU. Its major structural components are as follows:

- Control unit:
- Arithmetic and logic unit (ALU)
- Registers: Provides storage internal to the CPU.
- CPU interconnection: Some mechanism that provides for communication among the control unit, ALU, and registers

Finally, there are several approaches to the implementation of the control unit; one common approach is a microprogrammed implementation. In essence, a microprogrammed control unit operates by executing microinstructions that define the functionality of the control unit. With this approach, the structure of the control unit can be depicted, as in Figure 1.1..

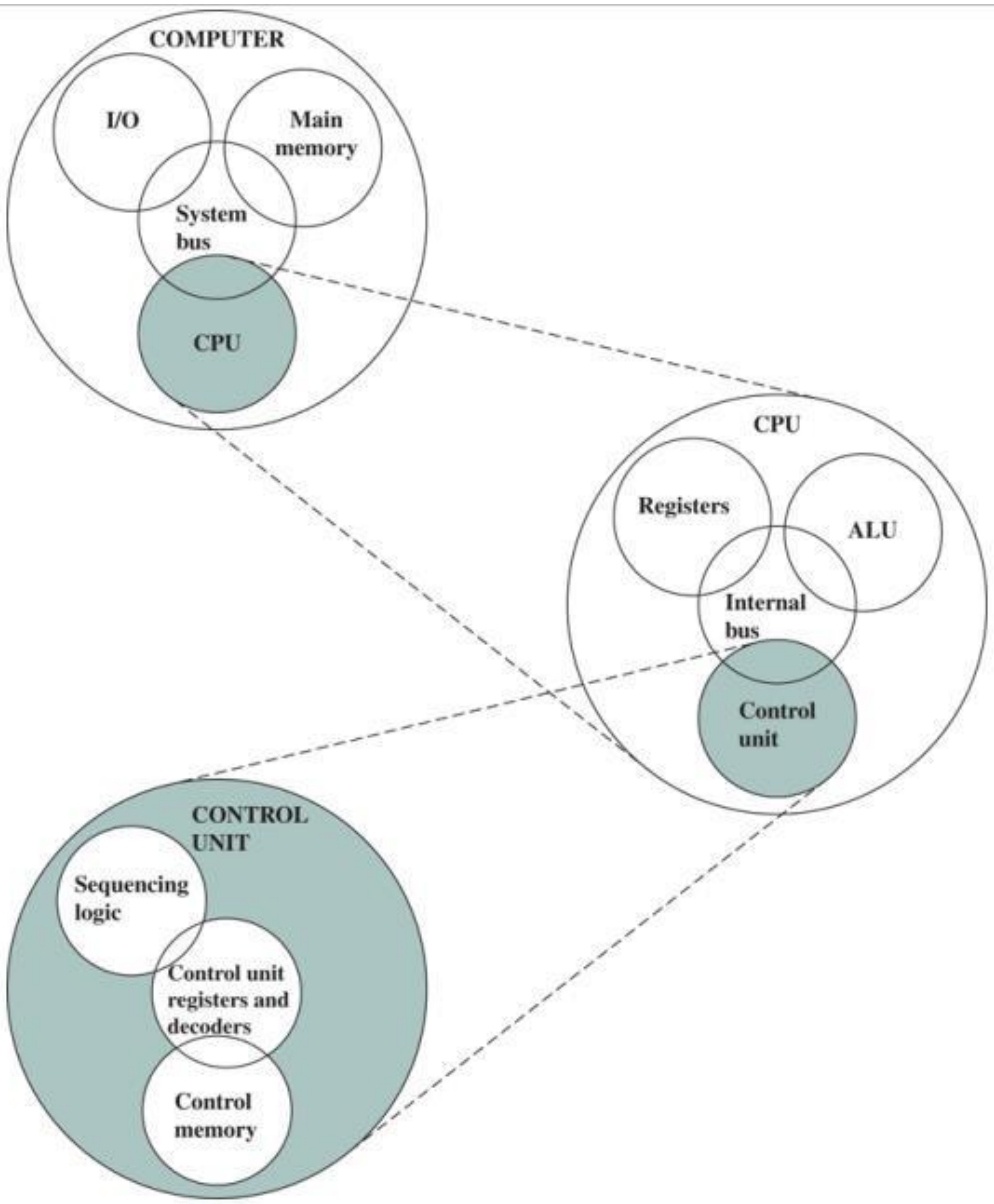


Figure 1.1 The Computer: Top-Level Structure

2. *MULTICORE COMPUTER STRUCTURE*

When these processors all reside on a single chip, the term multicore computer is used, and each processing unit (consisting of a control unit, ALU, registers, and perhaps cache) is called a core.

- Central processing unit (CPU):. It consists of an ALU, a control unit, and registers. In a system with a single processing unit, it is often simply referred to as a *processor*.
- Core: An individual processing unit on a processor chip. A core may be equivalent in functionality to a CPU on a single-CPU system.
- Processor: A physical piece of silicon containing one or more cores. The processor is the computer component that interprets and executes instructions. If a processor contains multiple cores, it is referred to as a multicore processor.

Figure 1.2 is a simplified view of the principal components of a typical multicore computer. A printed circuit board (PCB) is a rigid, flat board that holds and interconnects chips and other electronic components. The board is made of layers, typically two to ten, that interconnect components via copper pathways that are etched into the board. The main printed circuit board in a computer is called a system board or motherboard, while smaller ones that plug into the slots in the main board are called expansion boards.

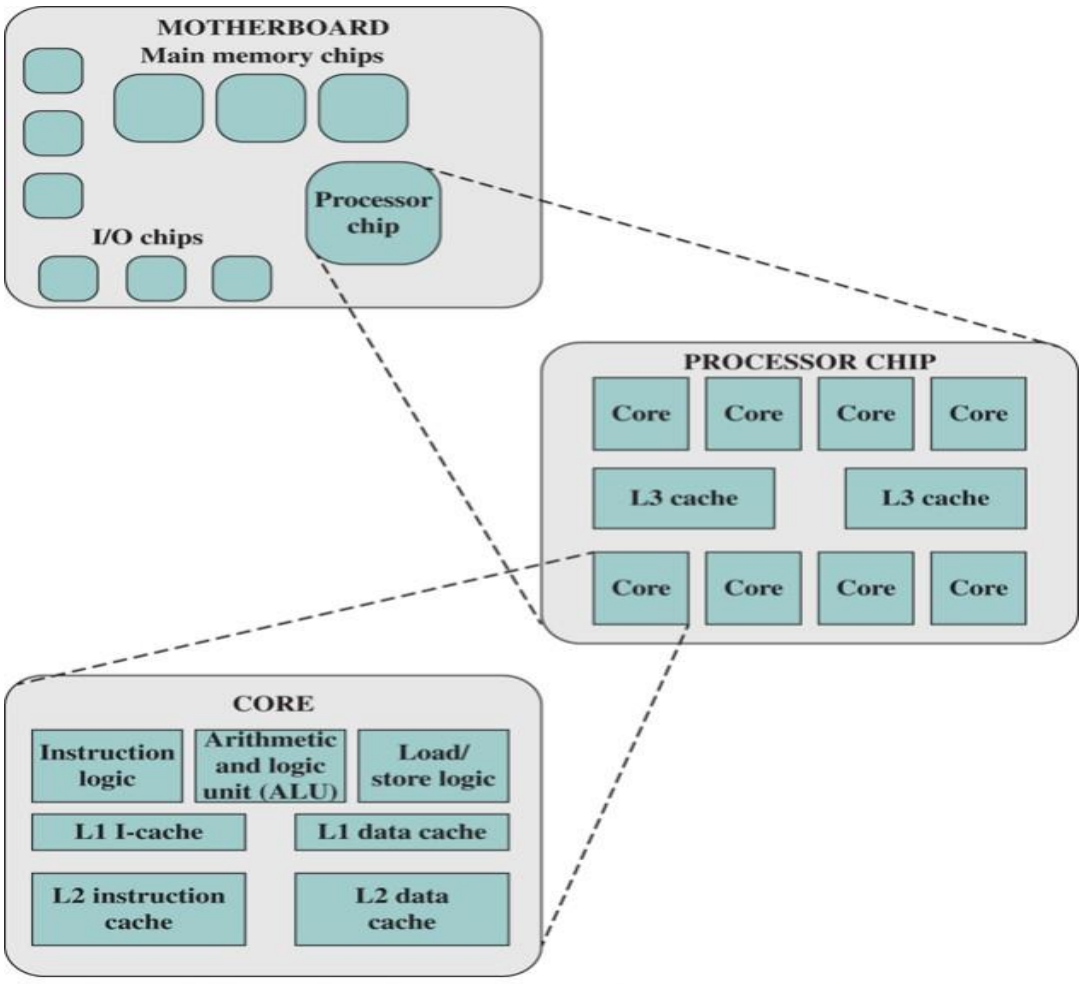


Figure 1.2 Simplified View of Major Elements of a Multicore Computer

The most prominent elements on the motherboard are the chips. A chip is a single piece of semiconducting material, typically silicon, upon which electronic circuits and logic gates are fabricated. The resulting product is referred to as an integrated circuit .

1.3 The Evolution of the Intel x86 Architecture

The current x86 offerings represent the results of decades of design effort on complex instruction set computers (CISCs). The x86 incorporates the sophisticated design principles once found only on mainframes and supercomputers and serves as an excellent example of CISC design. An alternative approach to processor design is the reduced instruction set computer (RISC) . The ARM architecture is used in a wide variety of embedded systems and is one of the most powerful and best-designed RISC based systems on the market. Table 1.1 shows that evolution

	(c) 1990s Processors			
	486TM SX	Pentium	Pentium Pro	Pentium II
Introduced	1991	1993	1995	1997
Clock speeds	16–33 MHz	60–166 MHz,	150–200 MHz	200–300 MHz
Bus width	32 bits	32 bits	64 bits	64 bits
Number of transistors	1.185 million	3.1 million	5.5 million	7.5 million
Feature size (μm)	1	0.8	0.6	0.35
Addressable memory	4 GB	4 GB	64 GB	64 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB
Cache	8 kB	8 kB	512 kB L1 and 1 MB L2	512 kB L2

	(d) Recent Processors				
	Pentium III	Pentium 4	Core 2 Duo	Core i7 EE 4960X	Core i9-7900X
Introduced	1999	2000	2006	2013	2017
Clock speeds	450–660 MHz	1.3–1.8 GHz	1.06–1.2 GHz	4 GHz	4.3 GHz
Bus width	64 bits	64 bits	64 bits	64 bits	64 bits
Number of transistors	9.5 million	42 million	167 million	1.86 billion	7.2 billion
Feature size (nm)	250	180	65	22	14
Addressable memory	64 GB	64 GB	64 GB	64 GB	128 GB
Virtual memory	64 TB	64 TB	64 TB	64 TB	64 TB
Cache	512 kB L2	256 kB L2	2 MB L2	1.5 MB L2/ 15 MB	14 MB L3

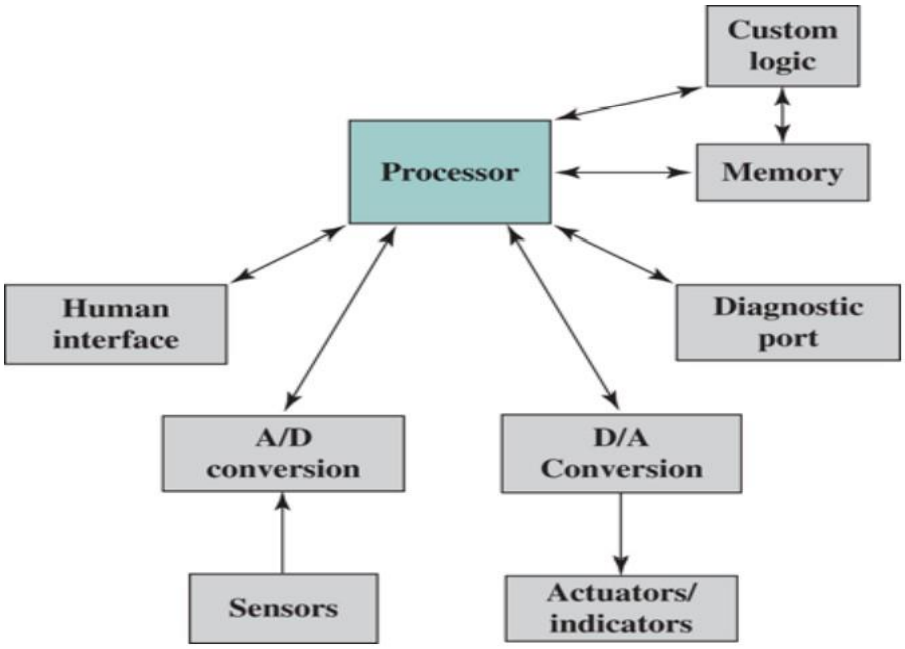
8080: This was an 8-bit machine, with an 8-bit data path to memory. 8080 was used in the first personal computer, the Altair.
8086: is the first appearance of the x86 architecture., 16-bit machine. In addition to a wider data path and larger registers. A variant of this processor, the 8088, was used in IBM's first personal computer, securing the success of Intel.
80386: Intel's first 32-bit machine, and a major overhaul of the product. With a 32-bit architecture. This was the first Intel processor to support multitasking, meaning it could run multiple programs at the same time.
80486: The 80486 introduced the use of much more sophisticated and powerful cache technology and sophisticated instruction pipelining. The 80486 also offered a built-in math coprocessor.
Pentium: With the Pentium, Intel introduced the use of superscalar techniques, which allow multiple instructions to execute in parallel.
Pentium Pro: The Pentium Pro continued the move into superscalar organization begun with the Pentium, with aggressive use of register renaming, branch prediction, data flow analysis, and speculative execution.
Pentium II: The Pentium II incorporated Intel MMX technology, which is designed specifically to process video, audio, and graphics data efficiently.
Pentium III: The Pentium III incorporates additional floating-point instructions: The Streaming SIMD Extensions (SSE) instruction set extension added 70 new instructions designed to increase performance.
Pentium 4: The Pentium 4 includes additional floating-point and other enhancements formultimedia.
Core: This is the first Intel x86 microprocessor with a dual core, referring to the implementation of two cores on a single chip.
Core 2: The Core 2 extends the Core architecture to 64 bits. The Core 2 Quad provides four cores on a single chip. More recent Core offerings have up to 10 cores per chip.

1.4 **Embedded Systems**

The term embedded system refers to the use of electronics and software within a product, computing systems.
Types of devices with embedded systems are almost too numerous to list. Examples include cell phones, digital cameras, video cameras, calculators, microwave ovens, home security systems, washing machines, lighting systems, thermostats, printers, various automotive systems (e.g., transmission control, cruise control, fuel injection, anti-lock brakes, and suspension systems), tennis rackets, toothbrushes, and numerous types of sensors and actuators in automated systems.

Often, embedded systems are tightly coupled to their environment. This can give rise to real-time constraints imposed by the need to interact with the environment. Constraints, such as required speeds of motion, required precision of measurement, and required time durations, dictate the timing of software operations. If multiple activities must be managed simultaneously, this imposes more complex real-time constraints.

The following figure shows in general terms an embedded system organization. In addition to the processor and memory, there are a number of elements that differ from the typical desktop or laptop computer



Possible Organization of an Embedded System

Microprocessors versus Microcontrollers

microprocessor chips included registers, an ALU, and some sort of control unit or instruction processing logic. Contemporary microprocessor chips, as shown in Figure 1.2, include multiple cores and a substantial amount of cache memory.

A microcontroller is a single chip that contains the processor, non-volatile memory for the program (ROM), volatile memory for input and output (RAM), a clock, and an I/O control unit. The processor portion of the microcontroller has a much lower silicon area than other microprocessors and much higher energy efficiency. See figure 1.3

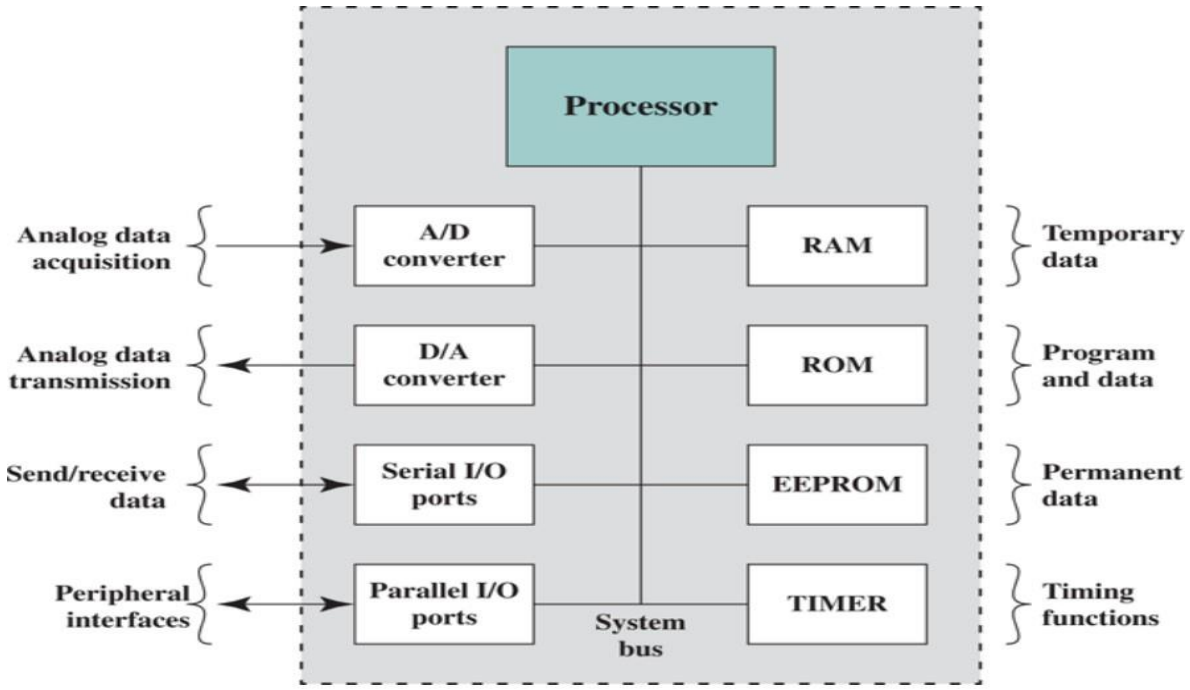


Figure 1.3 Typical Microcontroller Chip Elements

1.5 ARM Architecture

The ARM architecture refers to a processor architecture that has evolved from RISC design principles and is used in embedded systems. ARM is probably the most widely used embedded processor architecture and indeed the most widely used processor architecture of any kind in the world

ARM chips are

- high-speed processors that are known for their small die size
- and low power requirements.
- They are widely used in smartphones and other handheld devices
- ARM chips are the processors in Apple's popular iPod and iPhone devices, and are used in virtually all Android smartphones as well

Chapter Two
The Computer System

2.1 Computer Components

In the original case of customized hardware, the system accepts data and produces results (Figure 2.1a), the system accepts data and control signals and produces results. Thus, instead of rewiring the hardware for each new program

The entire program is actually a sequence of steps. At each step, some arithmetic or logical operation is performed on some data. For each step, a new set of control signals is needed. Let us provide a unique code for each possible set of control signals, and let us add to the general- purpose hardware a segment that can accept a code and generate control signals (Figure 2.1b).

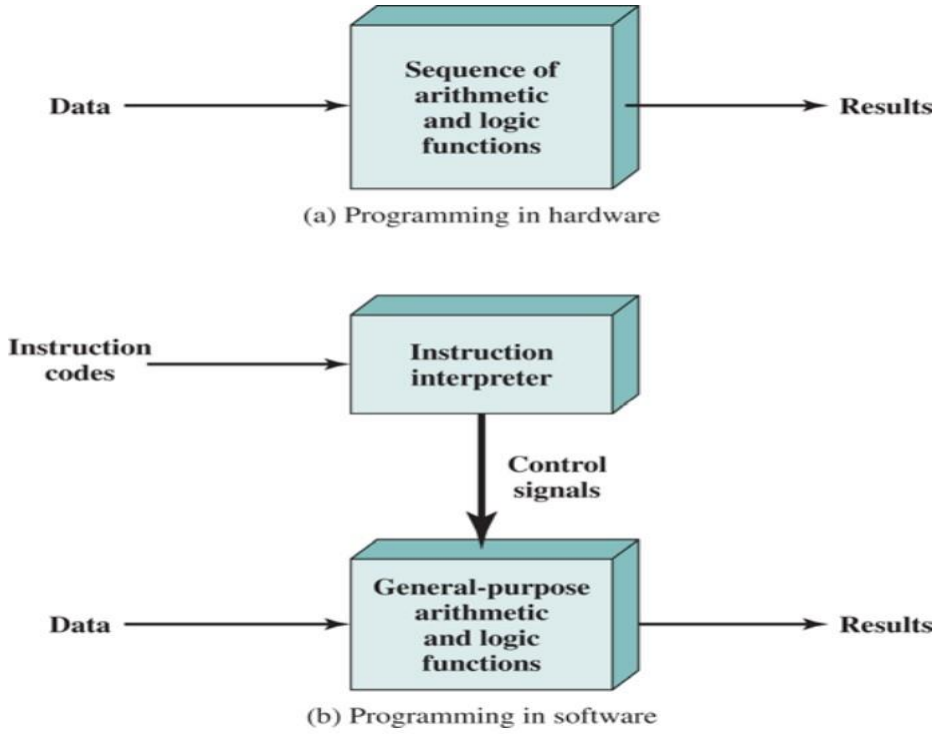


Figure 2.1 Hardware and Software Approaches

Figure 2.2 illustrates these top-level components and suggests the interactions among them. The CPU exchanges data with memory. For this purpose, it typically makes use of

- two internal (to the CPU) registers:
  1- a memory address register (MAR) , which specifies the address in memory for the next read or write,
  2- and a memory buffer register (MBR) , which contains the data to be written into memory or receives the data read from memory.
- Similarly, an I/O
  1- address register (I/O AR) specifies a particular I/O device.

2- An I/O buffer register (I/O BR) is used for the exchange of data between an I/O module and the CPU.

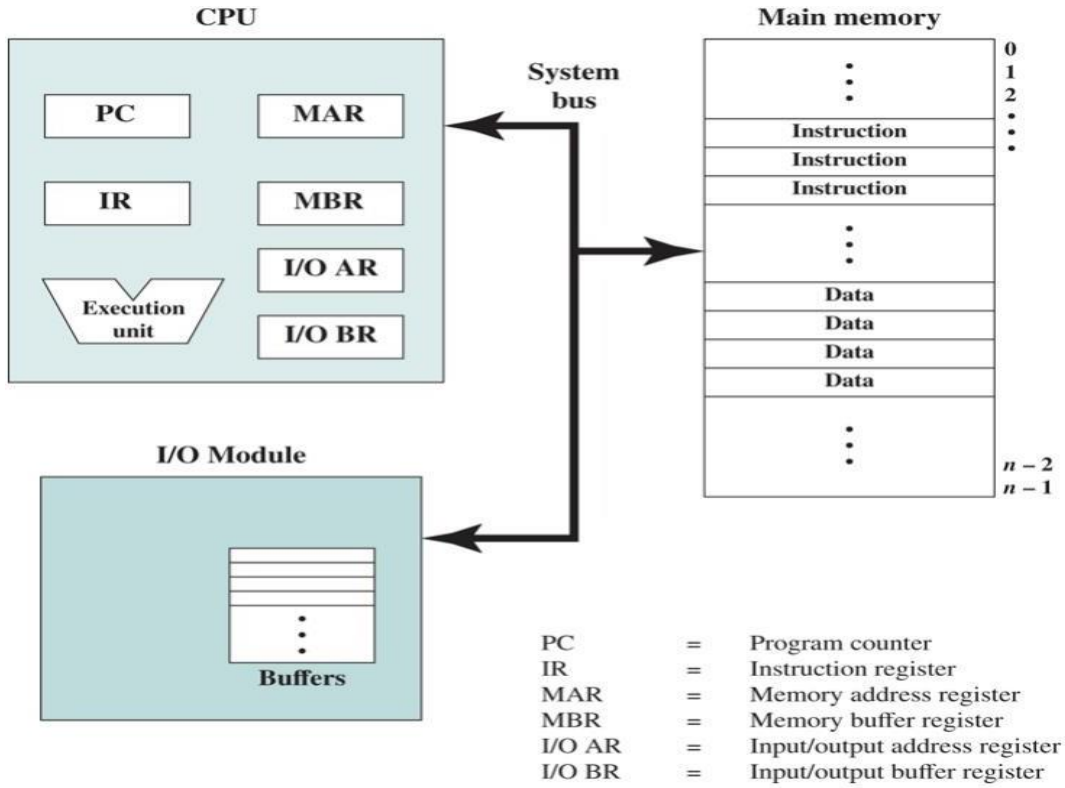


Figure 2.2 Computer Components: Top-Level View

2.2 Computer Function

The processing required for a single instruction is called an instruction cycle . Using the simplified two-step description given previously, the instruction cycle is depicted in Figure 2.3. The two steps are referred to as

- the fetch cycle and
- the execute cycle .

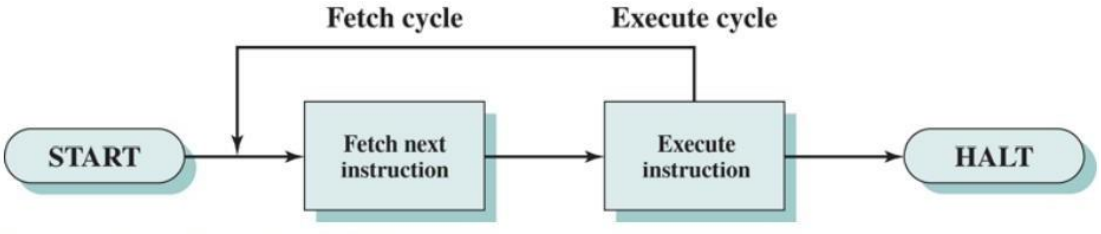


Figure 2.3 Basic Instruction Cycle

Instruction Fetch and Execute: At the beginning of each instruction cycle, the processor fetches an instruction from memory.

1- a register called the program counter (PC) holds the address of the instruction to be fetched next
2- the processor always increments the PC after each instruction fetch so that it will fetch the next instruction

3- The fetched instruction is loaded into a register in the processor known as the instruction register (IR).
4- . The processor interprets the instruction and performs the required action. In general, these actions fall into four categories.

Consider a simple example using a hypothetical machine that includes the characteristics listed in Figure 2.4. For And the organize memory using 16-bit words.

- The instruction format provides 4 bits for the opcode,, so 2^4=16 different opcodes
- and up to 2^{12}=4096 (4K) words of memory can be directly addressed

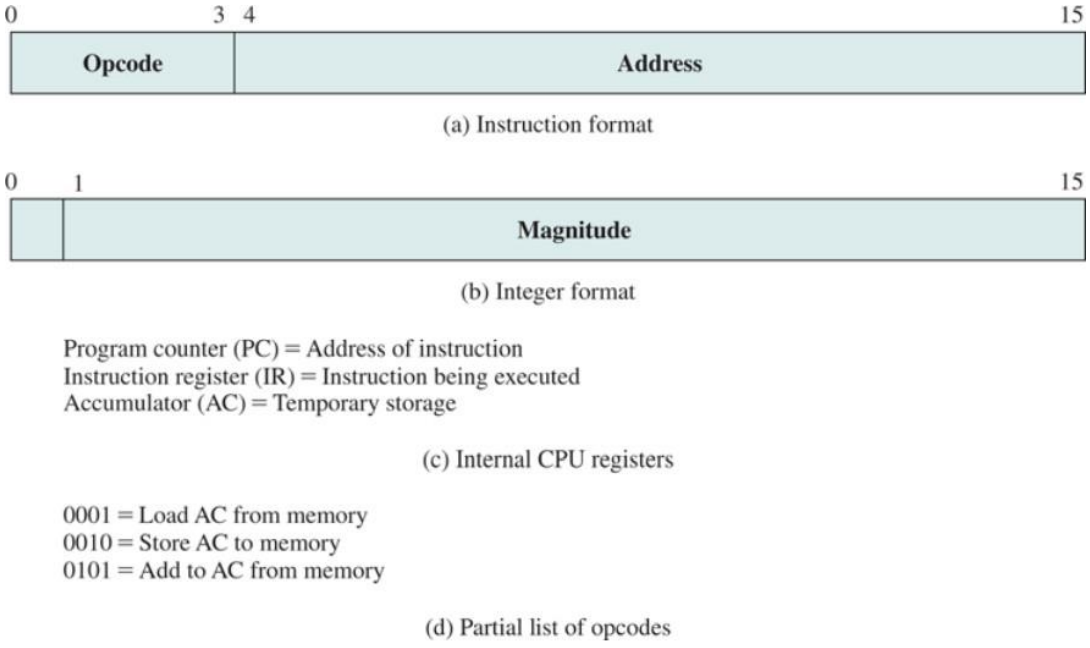


Figure 2.4 Characteristics of a Hypothetical Machine

A single instruction cycle with the following steps occurs:

1- Fetch the ADD instruction.
2- Read the contents of memory location A into the processor.
3- Read the contents of memory location B into the processor.
4- Add the two values.
5- Write the result from the processor to memory location A

In Figure 2.5 For any given instruction cycle, some states may be null and others may be visited more than once. The states can be described as follows:

- Instruction address calculation (iac):
- Instruction fetch (if):
- Instruction operation decoding (iod):
- Operand address calculation (oac):.
- Operand fetch (of):.
- Data operation (do):
- Operand store (os):

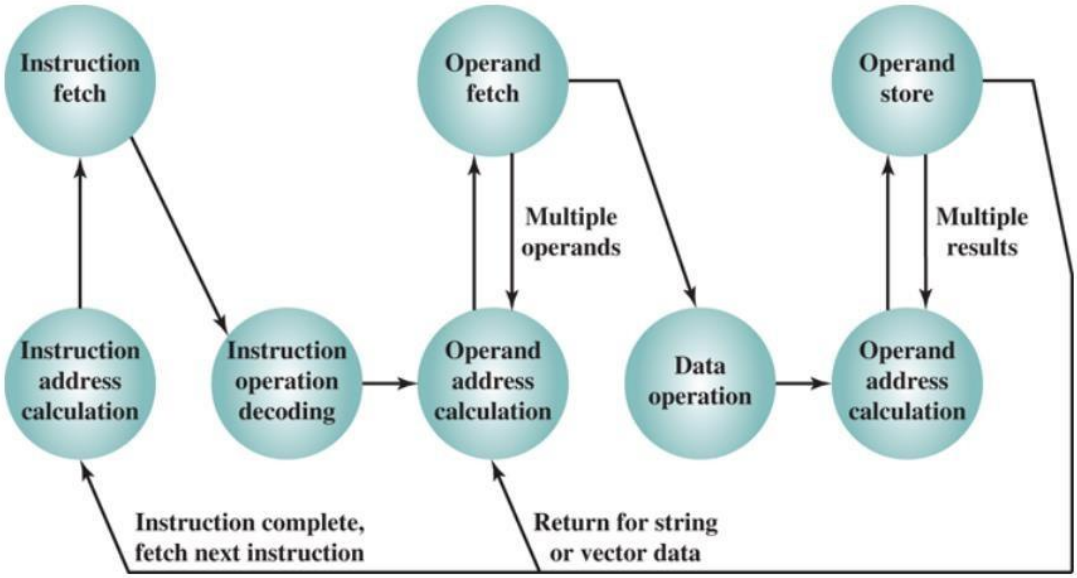


Figure 2.5 Instruction Cycle State Diagram

Interrupts

Virtually all computers provide a mechanism by which other modules (I/O, memory) may interrupt the normal processing of the processor. Table 2.1 lists the most common classes of

Table 2.1 Classes of Interrupts

Program	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, or reference outside a user's allowed memory space.
Timer	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
I/O	Generated by an I/O controller, to signal normal completion of an operation, request service from the processor, or to signal a variety of error conditions.
Hardware Failure	Generated by a failure such as power failure or memory parity error.

Let us try to clarify what is happening in *Figure 2.6* ,We have a user program that contains two *WRITE* commands. There is a segment of code at the beginning, then one *WRITE* command, then a second segment of code, then a second *WRITE* command, then a third and final segment of code. The *WRITE* command invokes the I/O program provided by the OS. Similarly, the I/O program consists of a segment of code, followed by an I/O command, followed by another segment of code. The I/O command invokes a hardware I/O operation.

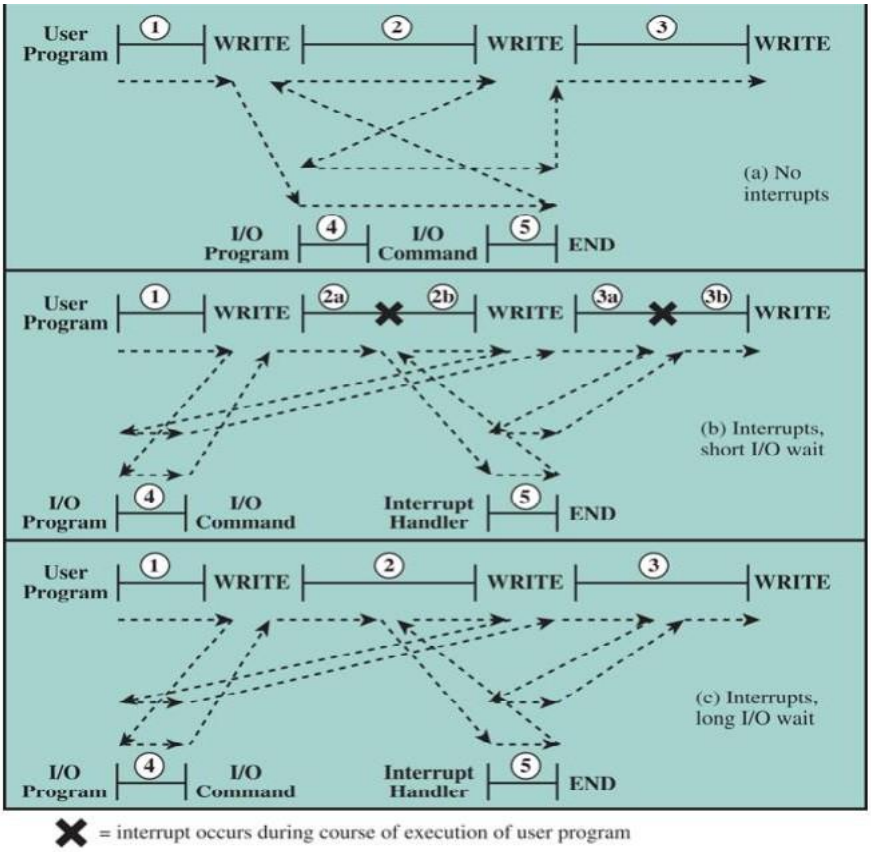


Figure 2.6 Program Flow of Control without and with Interrupts

When the interrupt processing is completed, execution resumes *(Figure 2.7*). To accommodate interrupts, an interrupt cycle is added to the instruction cycle, as shown in Figure 2.8:

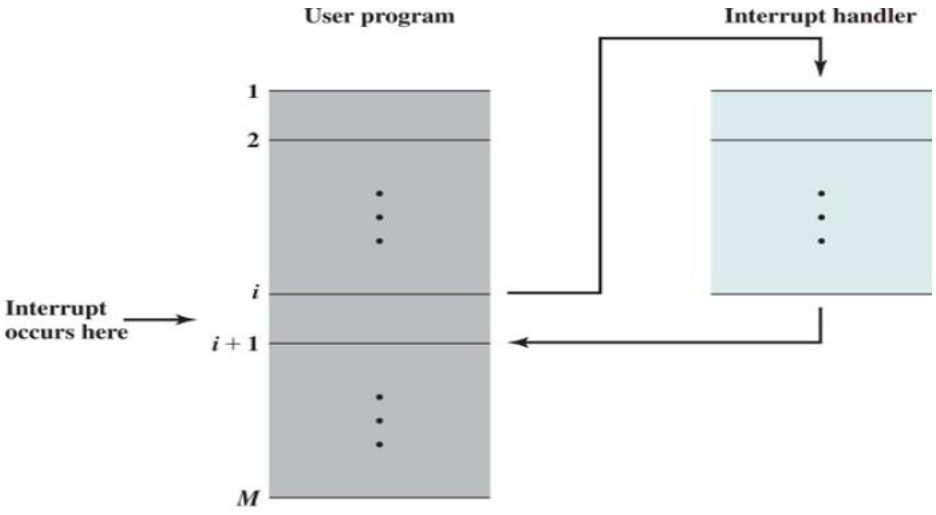


Figure 2.7 Transfer of Control via Interrupts

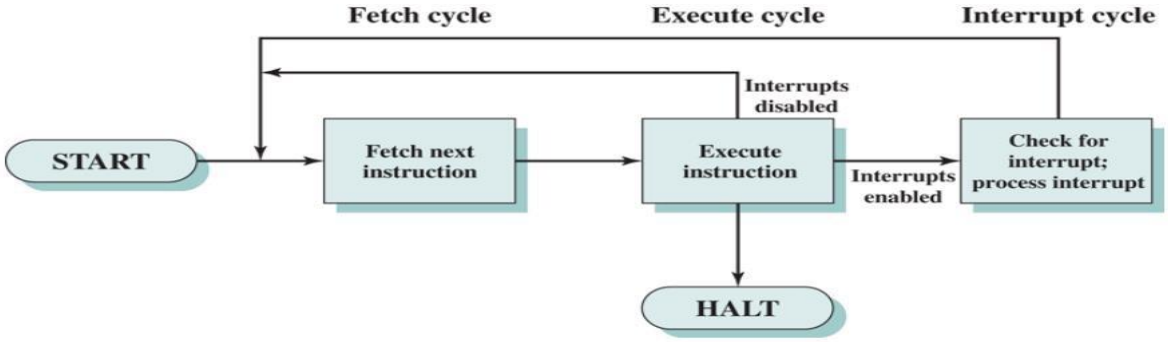
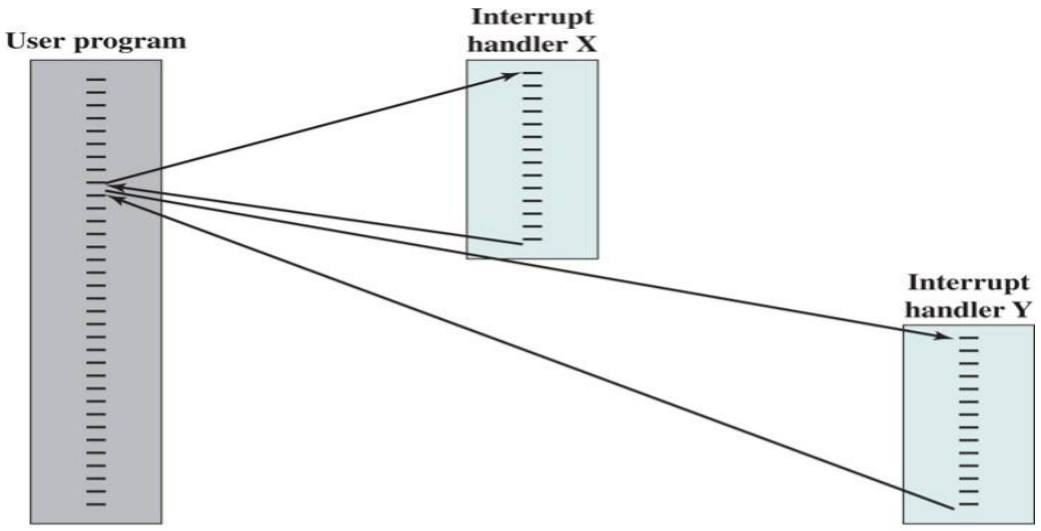


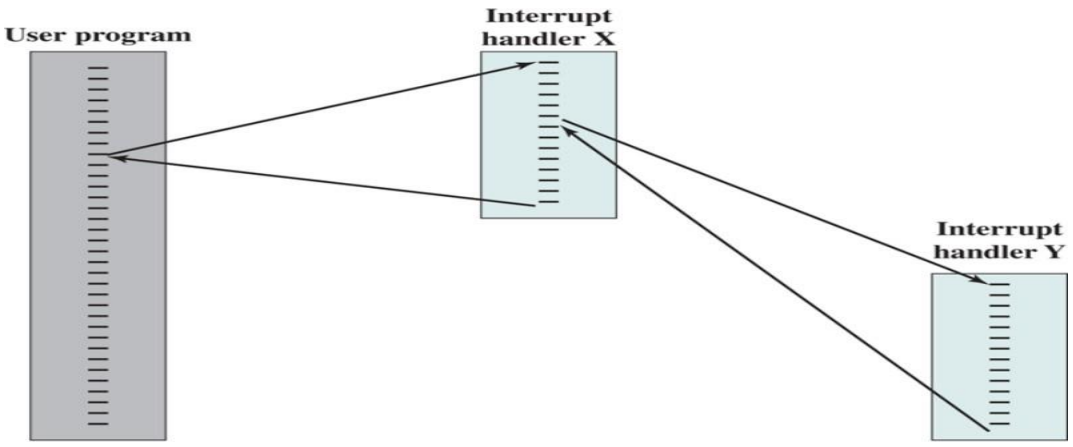
Figure 2.8 Instruction Cycle with Interrupts

MULTIPLE INTERRUPTS. Two approaches can be taken to dealing with multiple interrupts:

- The first is to disable interrupts while an interrupt is being processed. This approach is nice and simple, as interrupts are handled in strict sequential order (Figure 2.9a)
- A second approach is to define priorities for interrupts and to allow an interrupt of higher priority to cause a lower-priority interrupt handler to itself be interrupted (Figure 3.9b).



(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 2.9 Transfer of Control with Multiple Interrupts

Example: of this second approach, consider a system with three I/O devices: a printer, a disk, and a communications line, with increasing priorities of 2, 4, and 5, respectively. Figure 3.10 illustrates a possible sequence

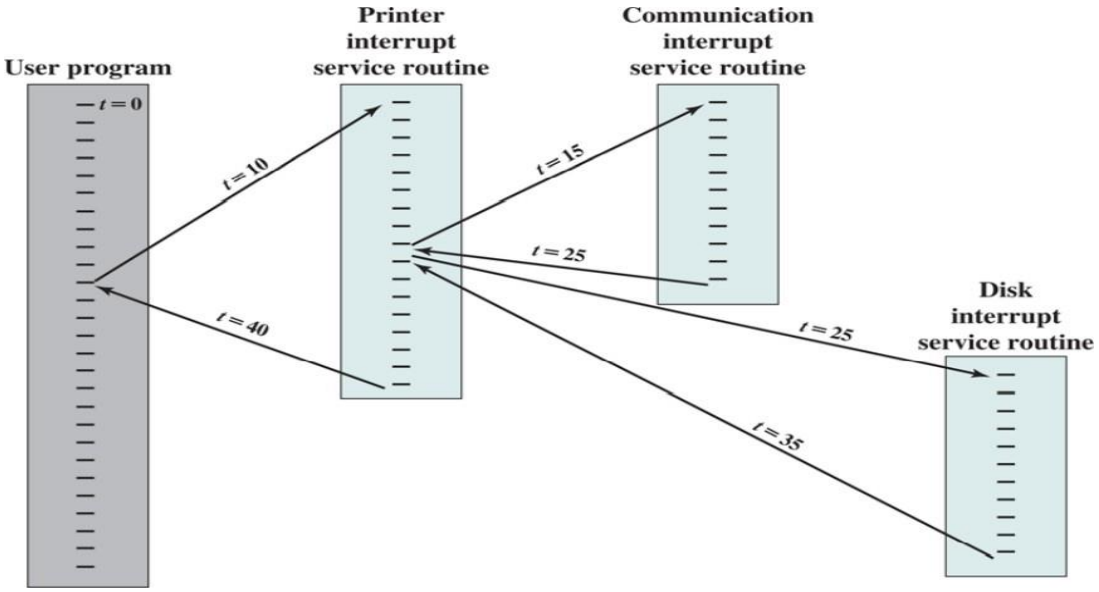


Figure 2.10 Example Time Sequence of Multiple Interrupts

2.3 Interconnection Structures

A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other. Thus, there must be paths for connecting the modules. The collection of paths connecting the various modules is called the interconnection structure. Figure 2.11 suggests the types of exchanges that are needed by indicating the major forms of input and output for each module type

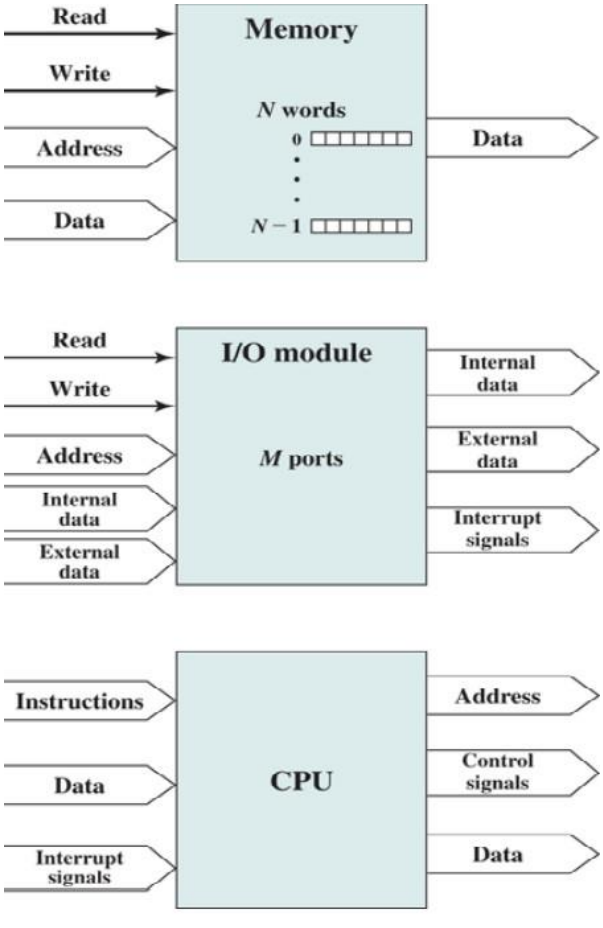


Figure 2.11 Computer Modules

The interconnection structure must support the following types of transfers:

- Memory to processor:
- Processor to memory:
- I/O to processor:
- Processor to I/O:
- I/O to or from memory:

Over the years, a number of interconnection structures have been tried. By far the most common are :
(1) the bus and various multiple-bus structures,
(2) point-to-point interconnection structures with packetized data transfer.

3.3.1 Bus Interconnection

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy. A bus that connects major computer components (processor, memory, I/O) is called a system bus .

Although there are many different bus designs, on any bus the lines can be classified into three functional groups (Figure 2.12): data, address, and control lines.

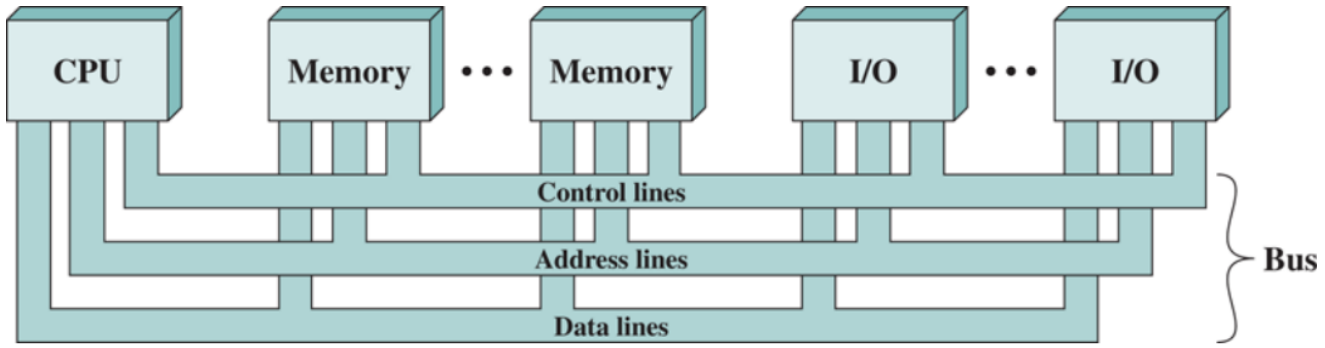


Figure 2.12 Bus Interconnection Scheme

The data lines =data bus . The data bus may consist of 32, 64, 128, or even more separate lines, the number of lines being referred to as the width of the data bus.
The address lines = data bus. For example, if the processor wishes to read a word (8, 16, or 32 bits) of data from memory, it puts the address of the desired word on the address lines. Clearly, the width of the address bus determines the maximum possible memory capacity of the system.
The control bus are used to control the access to and the use of the data and address lines. Typical control lines include:

- Memory write.
- Memory read
- I/O write.
- I/O read.
- Transfer ACK
- Bus request
- Bus grant

- Interrupt request
- Interrupt ACK
- Clock
- Reset

2.3.1 Point-to-Point Interconnect

contemporary systems increasingly rely on point-to-point interconnection rather than shared buses. The principal reason driving the change from bus to point-to-point interconnect was:

1- the electrical constraints encountered with increasing the frequency of wide synchronous buses.
2- with multiple processors and significant memory on a single chip, it was found that the use of a conventional shared bus on the same chip magnified the difficulties of increasing bus data rate and reducing bus latency to keep up with the processors.

example of the point-to-point interconnect approach: Intel's QuickPath Interconnect (QPI), which was introduced in 2008**.** The following are significant characteristics of QPI:

- Multiple direct connections
- Layered protocol architecture.
- Packetized data transfer

Figure 2.13 illustrates a typical use of QPI on a multicore computer. If core A in Figure 2.13 needs to access the memory controller in core D then

1- it sends its request through either cores B or C,
2- which must in turn forward that request on to the memory controller in core D.

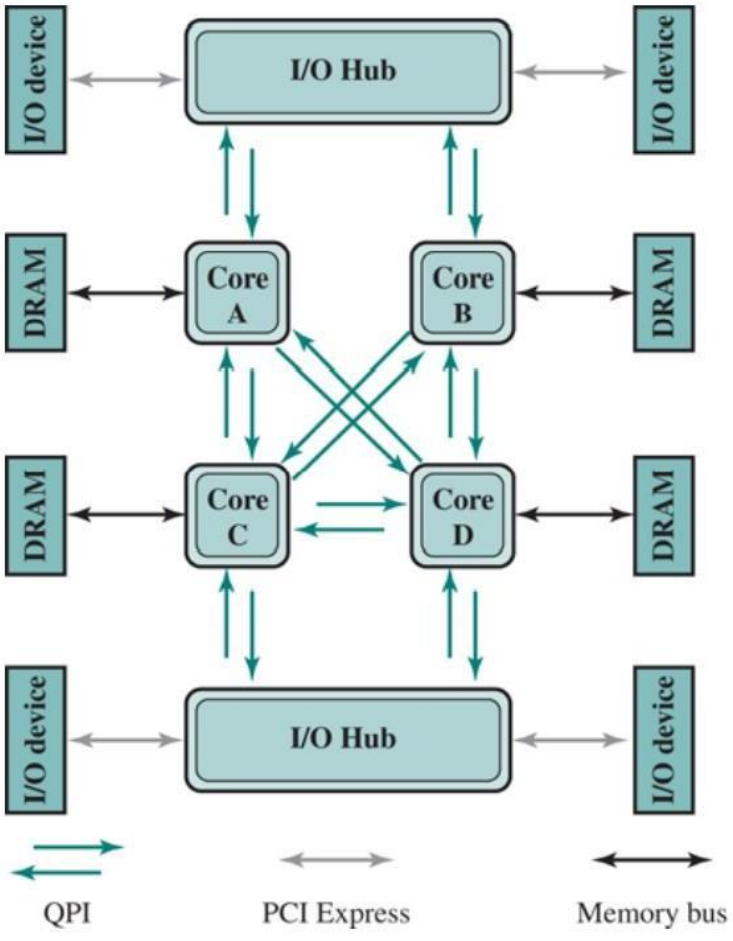


Figure 2.13 Multicore Configuration Using QPI

I/O hub (IOH) acts as a switch directing traffic to and from I/O devices.

QPI is defined as a four-layer protocol architecture, encompassing the following layers (Figure 3.14)

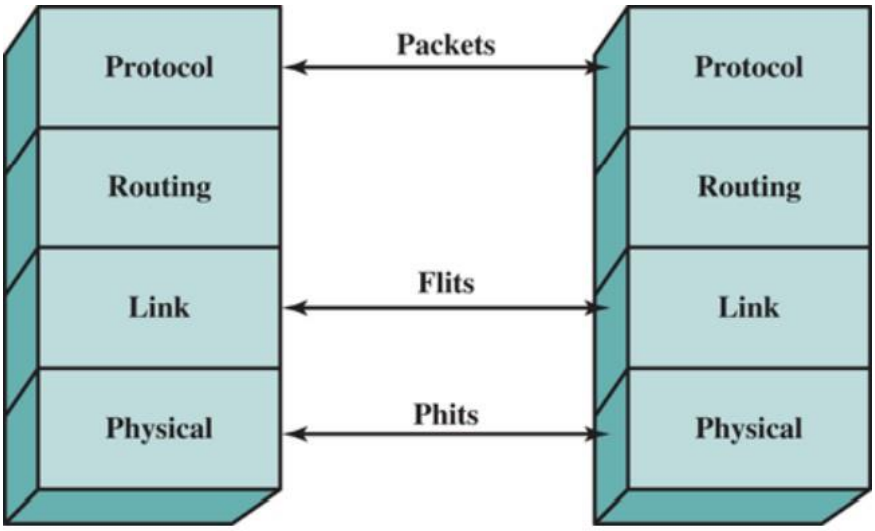


Figure 2.14 QPI Layers

- Physical:. The unit of transfer at the Physical layer is 20 bits, which is called a Phit (physical unit).
- Link: Responsible for reliable transmission and flow control. The Link layer's unit of transfer is an 80-bit Flit (flow control unit).
- Routing: Provides the framework for directing packets through the fabric.
- Protocol: The high-level set of rules for exchanging packets of data between devices.

Chapter 3
The Memory Hierarchy

3.1 Principle Of Locality

locality of reference: The principle reflects the observation that during the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

A distinction is made in the literature between two forms of locality:

1. Temporal locality: Refers to the tendency of a program to reference in the near future those units of memory referenced in the recent past. *For example*: when an iteration loop is executed, the processor executes the same set of instructions repeatedly.
2. Spatial locality: Refers to the tendency of a program to reference units of memory whose addresses are near one another. That is, if a unit of memory x is referenced at time t, it is likely that units in the range through will be referenced in the near future

EXAMPLE: For cache memory:

- temporal locality is traditionally exploited by keeping recently used instruction and data values in cache memory and by exploiting a cache hierarchy.
- Spatial locality is generally exploited by using larger cache blocks and by incorporating prefetching mechanisms (fetching items of anticipated use) into the cache control logic.

Typically, each instruction execution involves fetching the instruction from memory and, during execution, accessing one or more data operands from one or more regions of memory. Thus, there is a *spatial dual locality* :

- ✓ data spatial locality
- ✓ and instruction spatial locality.

And, of course, temporal locality exhibits this same *temporal dual behavior*:

- ✓ data temporal locality
- ✓ and instruction temporal locality.

Figure(3.1),show that, when an instruction is fetched from a unit of memory, it is likely that in the near future, additional instructions will be fetched from that same memory unit;

and when a data location is accessed, it is likely that in the near future, additional instructions will be fetched from that same memory unit

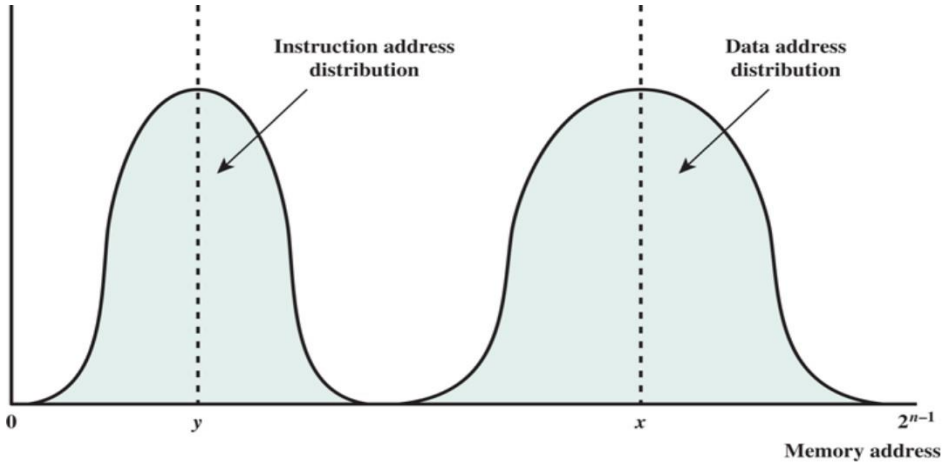


Figure 3.1 Idealized Spatial Locality Behavior: Probability Distribution for Next Memory Access (most recent data memory access at location x; most recent instruction fetch at location y)

3.2 Characteristics Of Memory Systems

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 : Location

- Internal: internal memory (e.g., processor registers, cache, main memory)
- External: External memory consists of peripheral storage devices, such as disk and tape, that are accessible to the processor via I/O controllers

Capacity

- Number of words : Common word lengths are 8, 16, and 32 bits.
- Number of bytes : External memory capacity is typically expressed in terms of bytes

Unit of Transfer(Word ,Block): 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, (1byte = 8bits)but is often larger, such as 64, 128, or 256 bits.

- Word: For example, the CRAY C90 has a 64-bit word length .The Intel x86 architecture has a wide variety of instruction lengths, expressed as multiples of bytes, and a word size of 32 bits.
- Addressable units: many systems allow addressing at the byte level. In any case, the relationship between the length in bits A of an address and the number N of addressable units is : $2^A=N$

Access Method

- Sequential : Memory is organized into units of data, called records. Access must be made in a specific linear sequence. (shared read–write mechanism)

- Direct : Access is accomplished by direct access to reach a general vicinity plus sequential searching, counting, or waiting to reach the final location(shared read–write mechanism)
- Random : Any location can be selected at random and directly addressed and accessed.
- 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

Performance

- Access time : : For random-access memory, this is the time it takes to perform a read or write operation. For non-random-access memory, access time is the time it takes to position the read–write mechanism at the desired location.
- Memory cycle time: This concept is primarily applied to random-access memory and consists of the access time plus any additional time required before a second access can commence
- Transfer rate : This is the rate at which data can be transferred into or out of a memory unit , it is equal to 1/(cycle time)

Physical Type (Semiconductor , Magnetic ,Optical ,Magneto-optical)

Physical Characteristics (Volatile/nonvolatile , Erasable/nonerasable ,Organization ,Memory module)

3.3 The Memory Hierarchy

A variety of technologies are used to implement memory systems, and across this spectrum of technologies, the following relationships hold:

- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access time

The way out of this dilemma is not to rely on a single memory component or technology, but to employ a memory hierarchy. A typical hierarchy is illustrated in Figure 3.2. 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

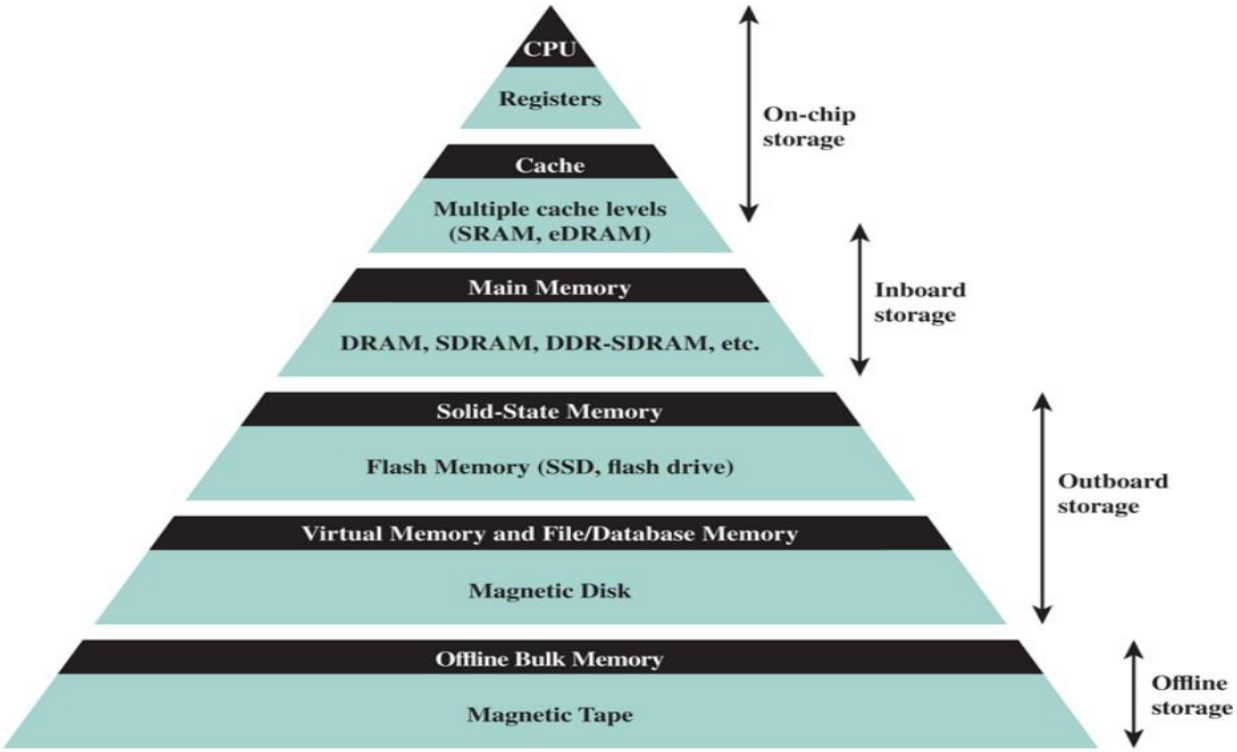


Figure 3.2 The Memory Hierarchy

Let us label the memory at level i of the memory hierarchy such that is M_i closer to the processor than M_{i+1} If C_i , T_i and R_i and S_i are respectively the cost per byte, average access time, average data transfer rate, and total memory size at level i, then the following relationships typically hold between level I and i+1:

$$\begin{aligned} C_i &> C_{i+1} \\ T_i &< T_{i+1} \\ R_i &> R_{i+1} \\ S_i &< S_{i+1} \end{aligned}$$

EXAMPLE

Suppose that the processor has access to two levels of memory. Level 1 contains X words and has an access time of $0.01\mu s$; level 2 contains $1000 \times X$ words and has an access time of $0.1\mu s$.

- Assume that if a word to be accessed is in level 1, then the processor accesses it directly.
- If it is in level 2, then the word is first transferred to level 1 and then accessed by the processor.

In our example, suppose that 95% of the memory accesses are found in Level 1. Then the average time to access a word can be expressed as

$$(0.95)(0.01\mu s) + (0.05)(0.01\mu s + 0.1\mu s) = 0.0095 + 0.0055 = 0.015\mu s$$

The average access time is much closer to $0.01\mu s$ than to $0.1\mu s$, as desired.

Let level 2 memory contain all program instructions and data. Currently used clusters can be temporarily placed in level 1. From time to time, one of the clusters in level 1 will have to be swapped back to level 2 to make room for a new cluster coming in to level 1. On average, however, most references will be to instructions and data contained in level 1.

In practice, the dynamic movement of chunks of data between levels during program , shown in Figure 3.3, with an indication of the size of the chunks of data exchanged between levels.

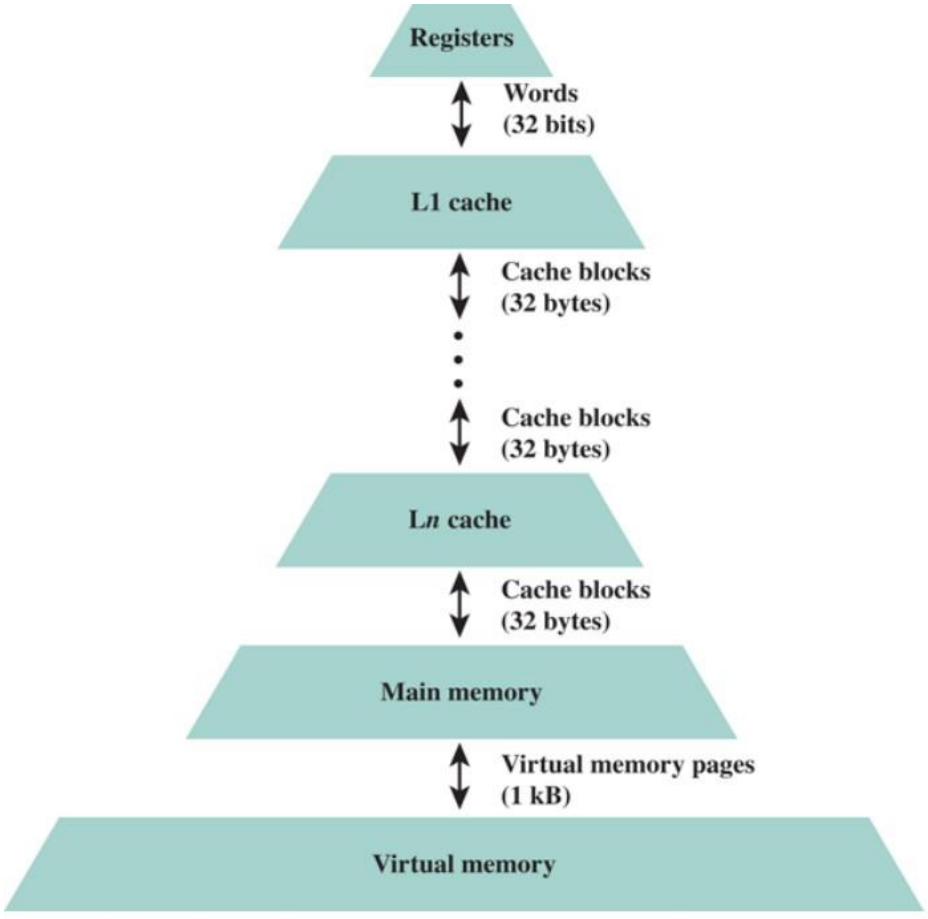


Figure 3.3 Exploiting Locality in the Memory Hierarchy (with typical transfer size)

3.3.1 Typical Members of the Memory Hierarchy

Table 3.1 lists some characteristics of key elements of the memory hierarchy:

- The fastest, smallest, and most expensive type of memory consists of the *registers internal to the processo*r.
- Next will be typically multiple layers of cache. Level 1 cache (L1 cache), closest to the processor registers, is almost always divided into an *instruction cache* and

a *data cache*. This split is also common for L2 caches. L3 cache and some have an L4 cache.

- *Main memory* is the principal internal memory system of the computer. Each location in main memory has a unique address. Main memory is visible to the programmer, whereas cache memory is not.
- External, nonvolatile memory is also referred to as secondary memory or auxiliary memory. These are used to store program and data files and are usually visible to the programmer only in terms of files and records, as opposed to
- Table 3.1

Memory level	Typical technology	Unit of transfer with next larger level (typical size)	Managed by
Registers	CMOS	Word (32 bits)	Compiler
Cache	Static RAM (SRAM); Embedded dynamic RAM (eDRAM)	Cache block (32 bytes)	Processor hardware
Main memory	DRAM	Virtual memory page (1 kB)	Operating system (OS)
Secondary memory	Magnetic disk	Disk sector (512 bytes)	OS/user
Offline bulk memory	Magnetic tape		OS/User

3.3.2 The IBM z13 Memory Hierarchy

Figure 3.4 illustrates the memory hierarchy for the IBM z13 mainframe computer ,It consists of the following levels:

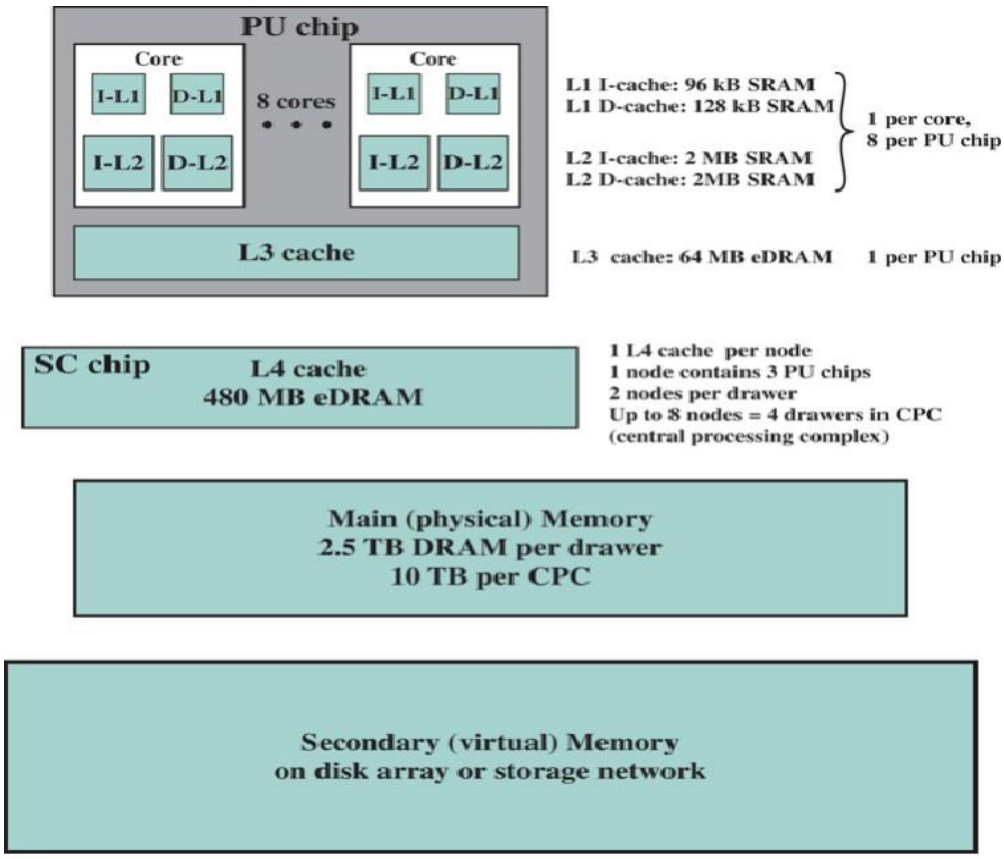


Figure 3.4 IBM z13 Memory Hierarchy

3.3.3 Design Principles for a Memory Hierarchy

Three principles guide the design of a memory hierarchy :

1. **Locality:** Locality is the principle that makes effective use of a memory hierarchy possible.
2. **Inclusion:** This principle dictates that all information items are originally stored in level M_n, where *n* is the level most remote from the processor. During the processing, subsets of M_n are copied into M_{n-1}. similarity, subsets of M_{n-1} are copied into M_{n-2}, and so on. This is expressed concisely as $M_i \subseteq M_{i+1}$. Thus, this is in contrast to our simple example of **Figure 4.1**, where Bob moved a folder from the file cabinet to his desk. With the memory hierarchy, units of data are copied rather than moved, so that the data unit that is moved to M_i remains in M_{i+1}. Thus, if a word is found in M_i, then copies of the same word also exist in all subsequent layers $M_{i+1}, M_{i+2}, \ldots, M_n$.
3. **Coherence:** Copies of the same data unit in adjacent memory levels must be consistent. If a word is modified in the cache, copies of that word must be updated immediately or eventually at all higher levels.

This leads to two requirements:

Vertical coherence: If one core makes a change to a cache block of data at L2, that update must be returned to L3 before another L2 retrieves that block.

Horizontal coherence: If two L2 caches that share the same L3 cache have copies of the same block of data, then if the block in one L2 cache is updated, the other L2 cache must be alerted that its copy is obsolete.

Chapter 4
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.

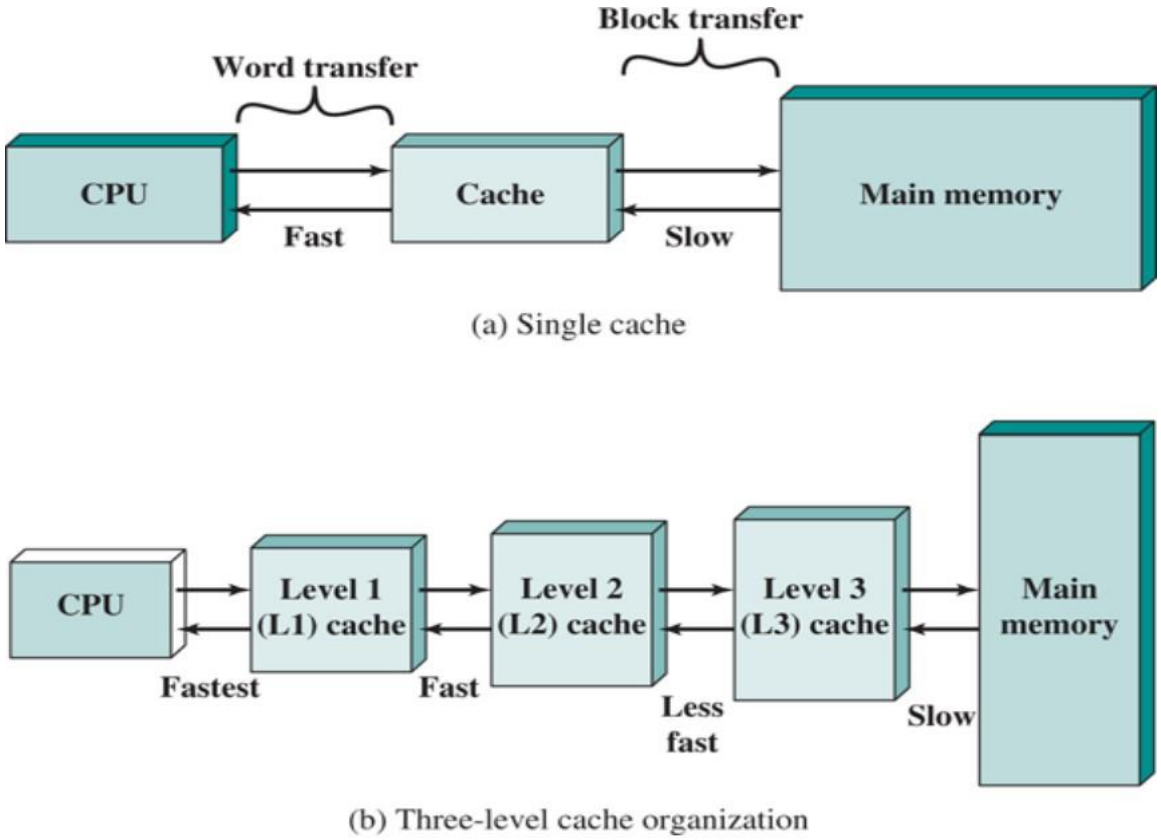


Figure 4.1 Cache and Main Memory

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

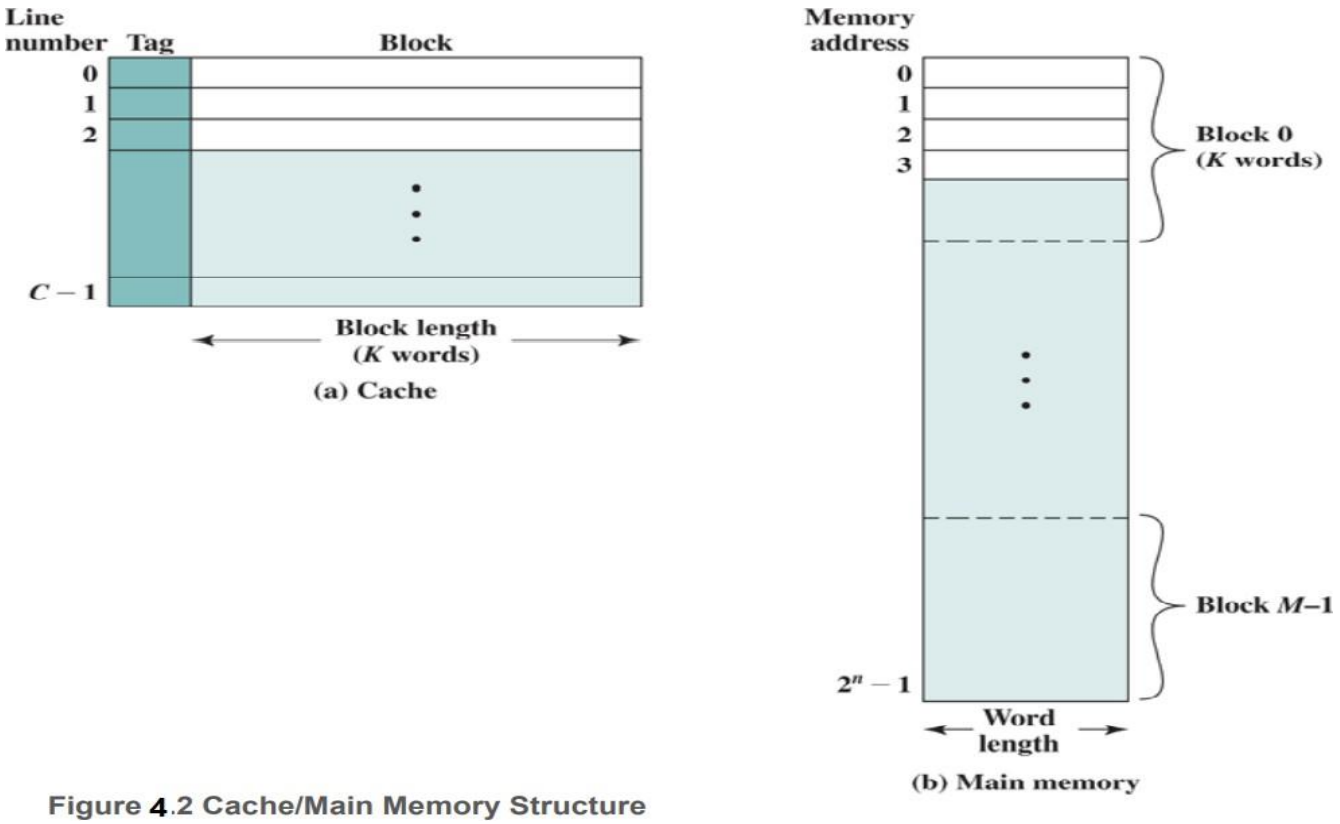


Figure 4.2 Cache/Main Memory Structure

- 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.

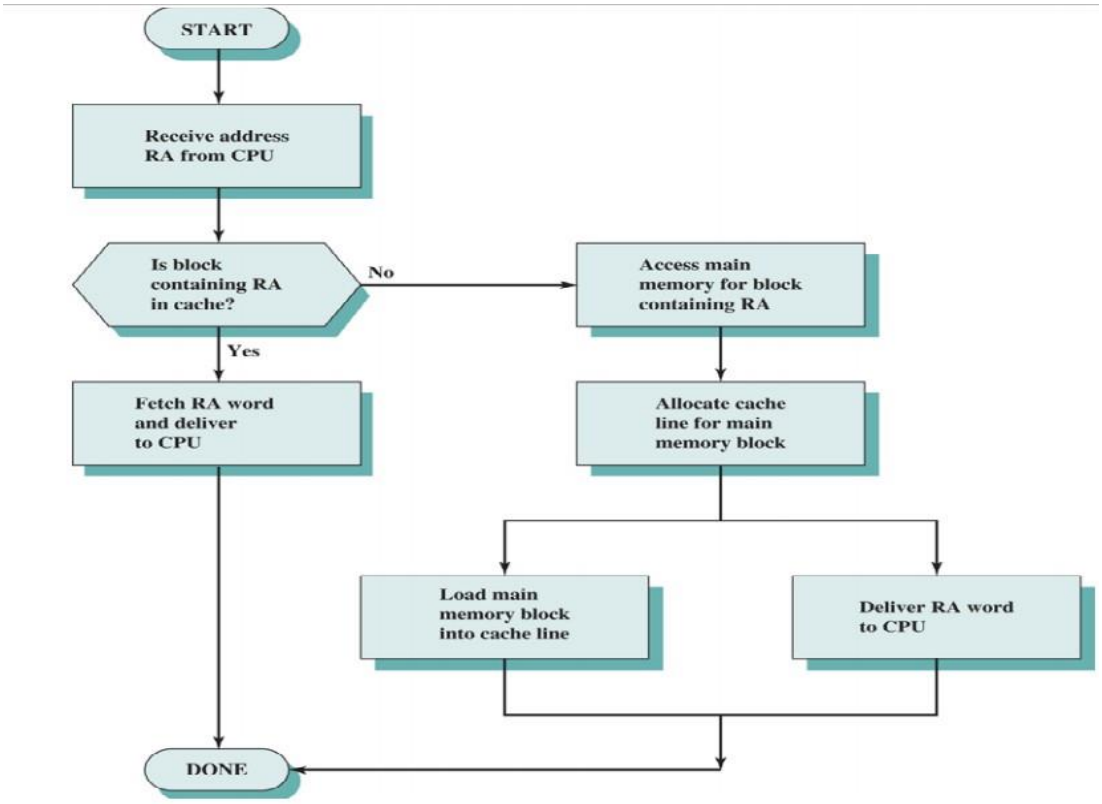


Figure 4.3 Cache Read Operation

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

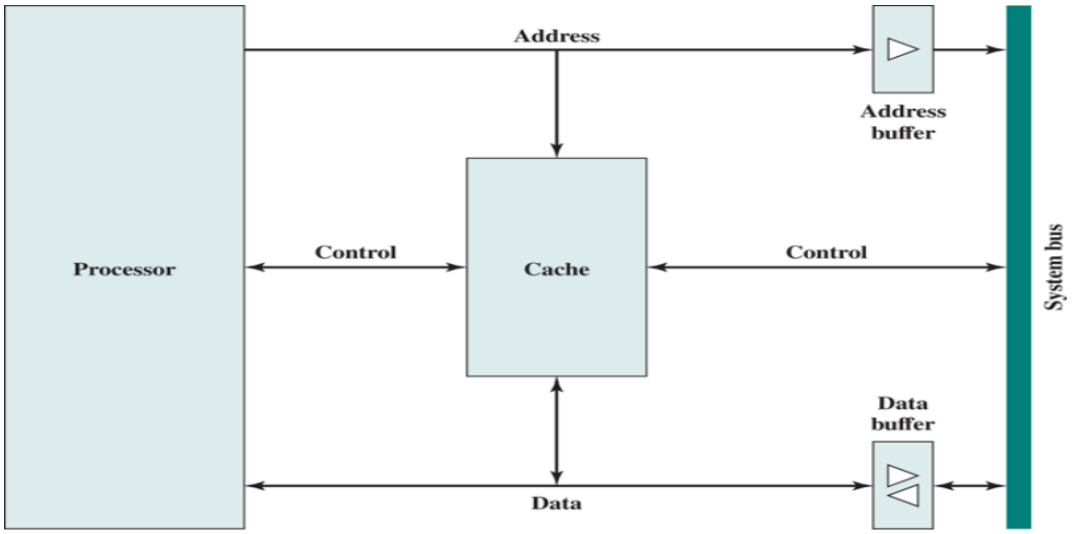


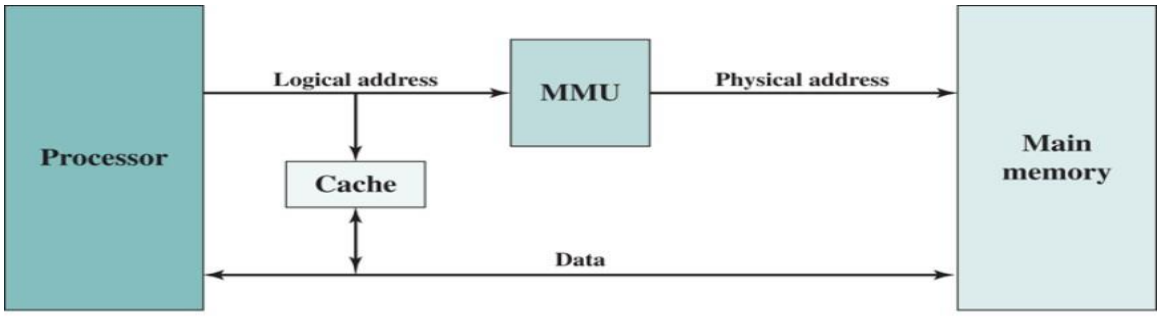
Figure 4.4 Typical Cache Organization

4.2 Elements of Cache Design

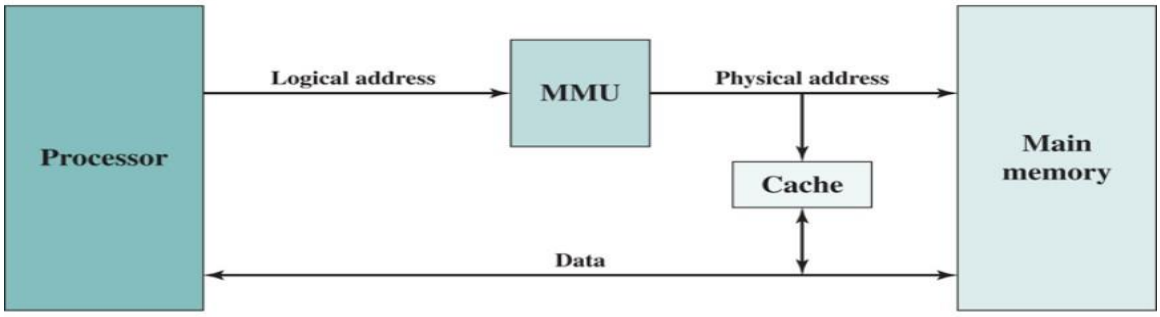
Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures.

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



(a) Logical cache



(b) Physical cache

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:

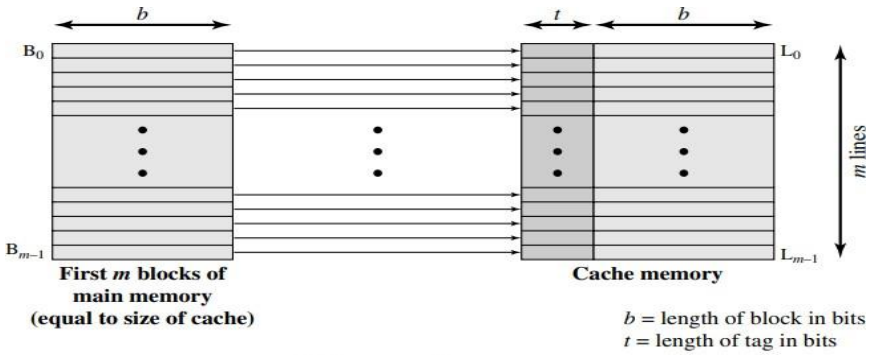
- The cache can hold 64 KBytes.
- Data are transferred between main memory and the cache in blocks of 4 bytes each. This means that the cache is organized as 16K = 2^{14} lines of 4 bytes each.
- The main memory consists of 16 Mbytes, with each byte directly addressable by a 24-bit address (2^{24} =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

1- Direct Mapping:

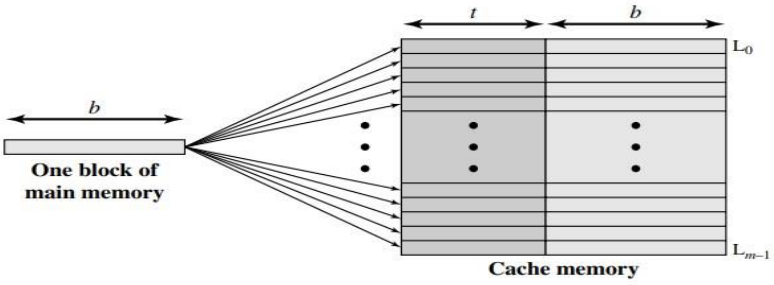
The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. The mapping is expressed as: $i = j \text{ modulo } m$

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.



(a) Direct mapping



(b) Associative mapping

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)

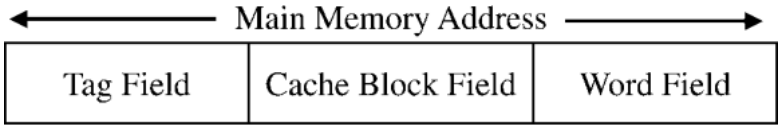
Example : Consider, for example, the case of a main memory consisting of 4K blocks, a cache memory consisting of 128 blocks, and a block size of 16 words. The following table shows the division of the main memory and the cache according to the direct-mapped cache technique. As the figure shows, there 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
3	0	384
1	1	129
0	2	
	126	
31	127	4095

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

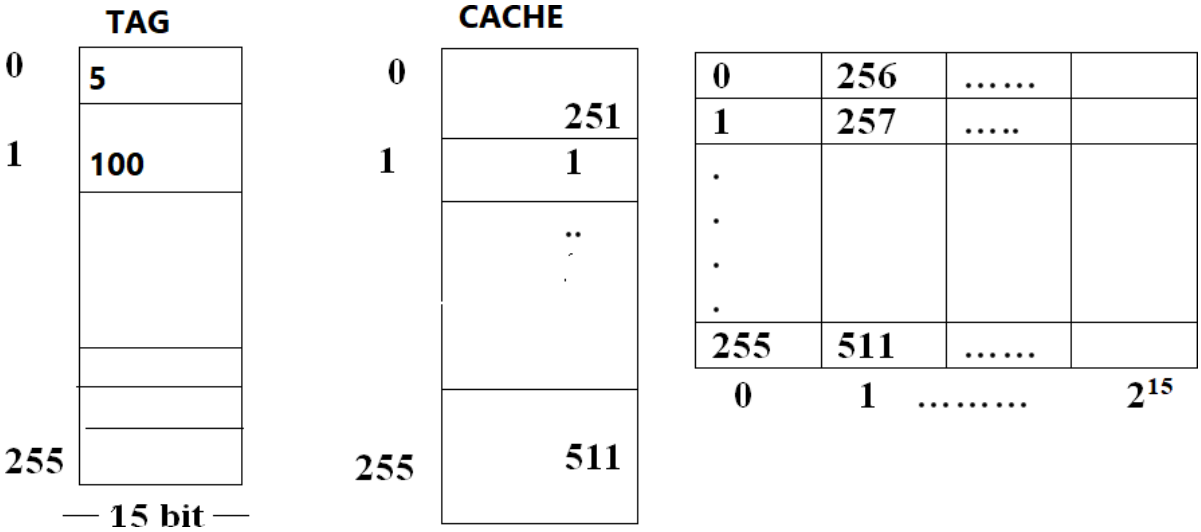
1. Word field =$\log_2$ B = $\log_2$ 16 = $\log_2$ 2^4 = 4 bits
2. Block field = $\log_2$ N = $\log_2$ 128 = $\log_2$ 2^7 = 7
3. bits Tag field = $\log_2$ (M/N) = $\log_2$ (2^2 X2^{10}/2^7) = 5 bits
4. The number of bits in the main memory address = $\log_2$ (B X M) = $\log_2$ (2^4 X 2^{12}) = 16 bits.



Direct-mapped address fields

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

Answer: **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 x 2^{20}/2^8) = $\log_2$(2^{23}/2^8)=15 bits
bits in the main memory address = $\log_2$ (B X M) = $\log_2$ (2^6 x 2^{23}) =29



The following Figure 4.7 illustrates the general mechanism.:

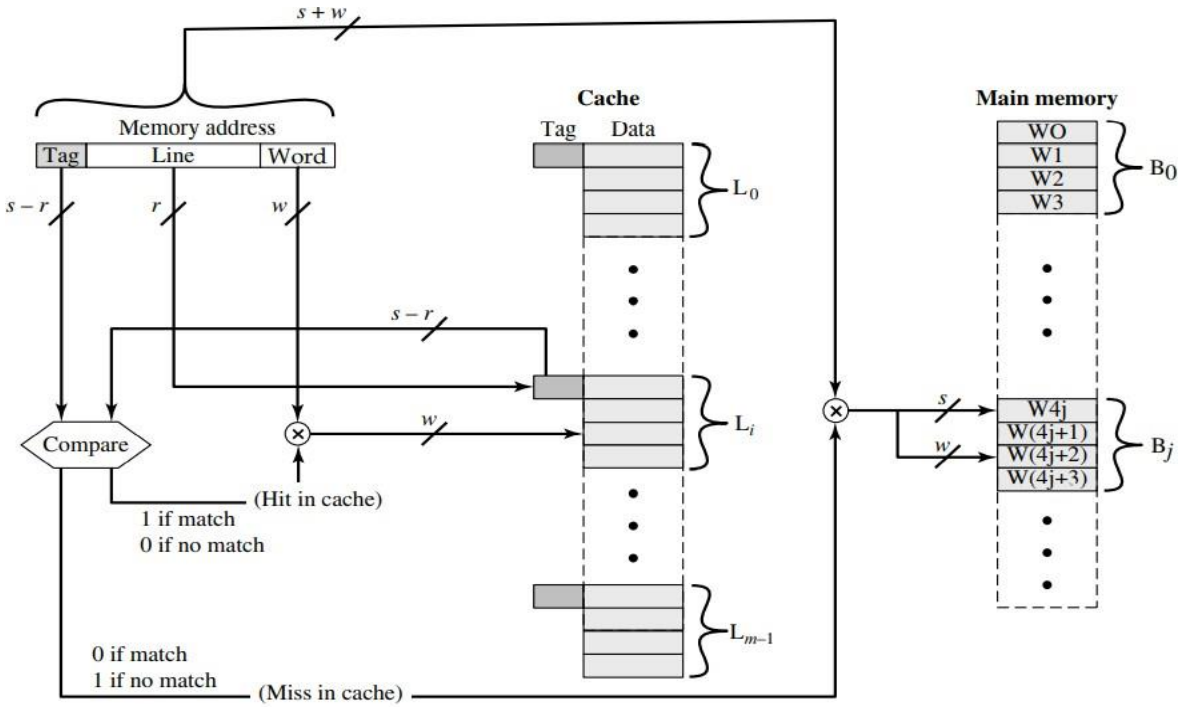
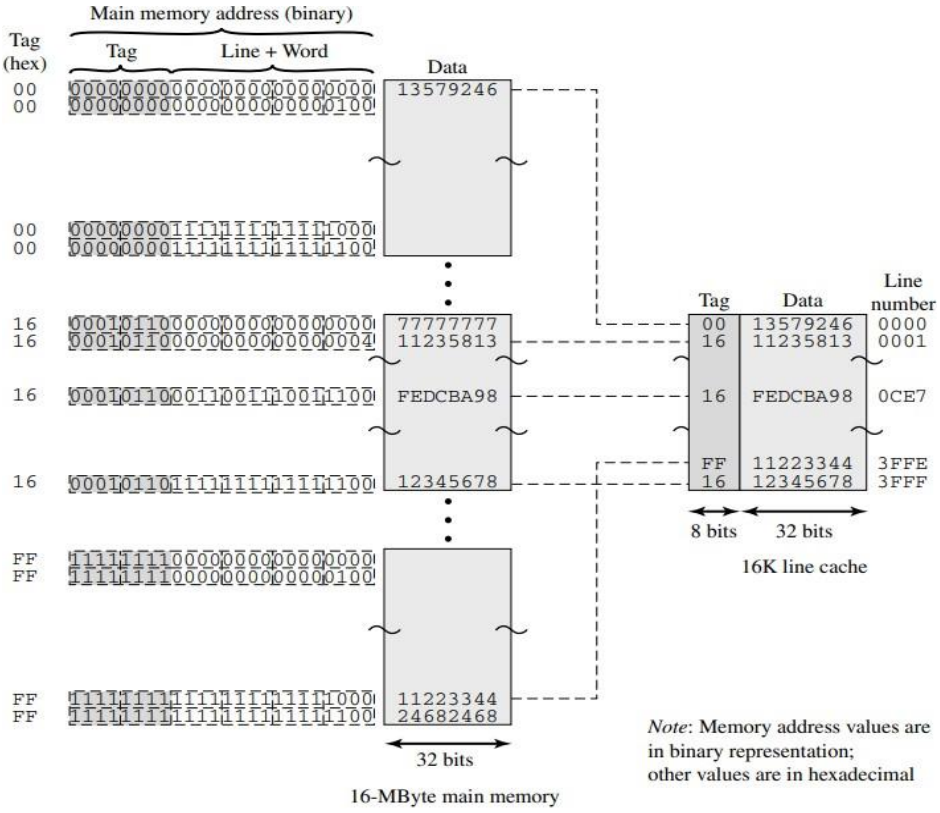


Figure 4.7 Direct-Mapping Cache Organization

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



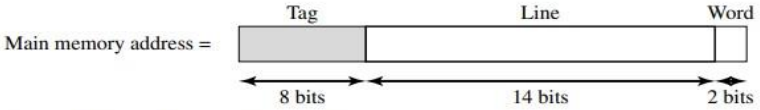


Figure 4.8 Direct Mapping Example

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

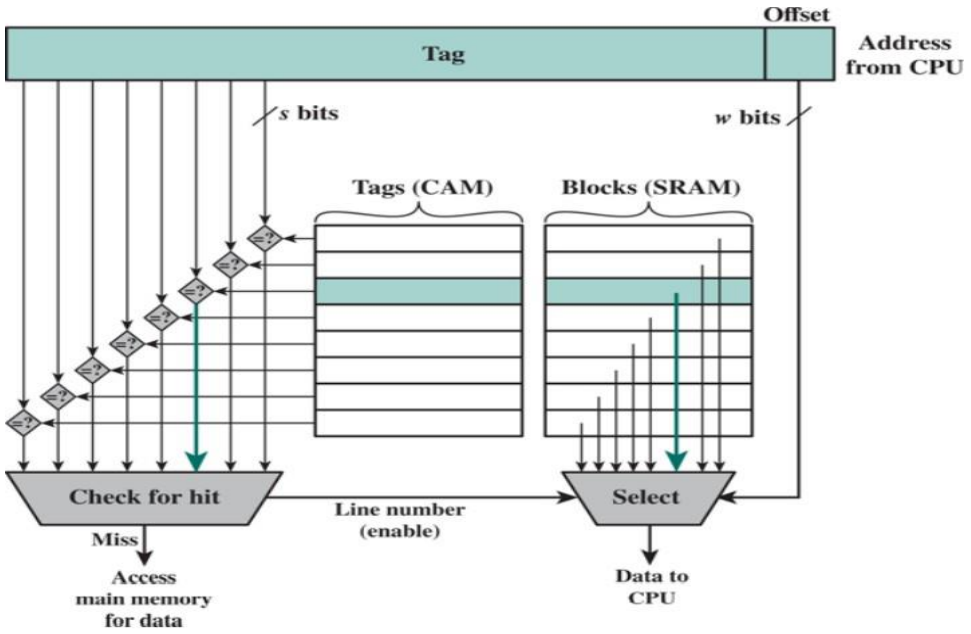
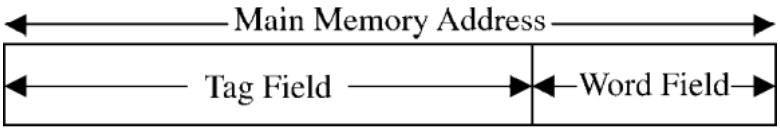


Figure 4.9 Fully Associative Cache Organization

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 $=\log_2 (B \times M) = \log_2 (2^4 \times 2^{12}) = 16$ 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 = $\log_2 (B \times M) = \log_2 (2^6 \times 2^{23}) = 29$

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.

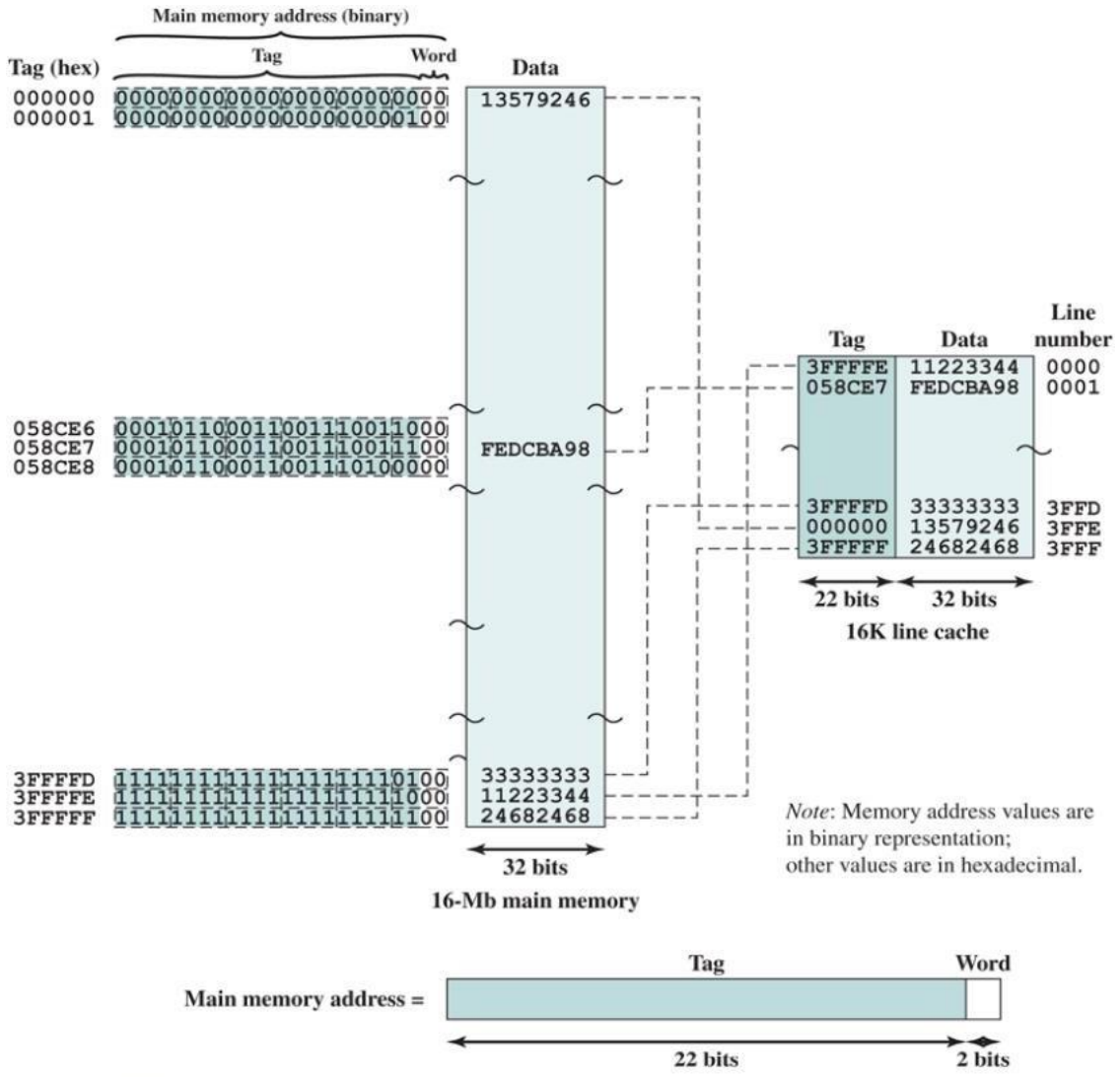


Figure 4.11 Associative Mapping Example

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

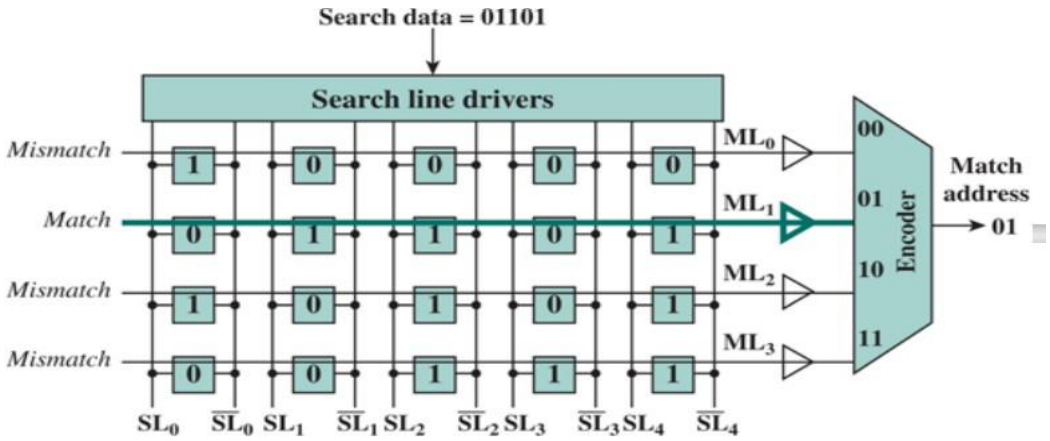


Figure 4.12 Simplified CAM circuitry

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.

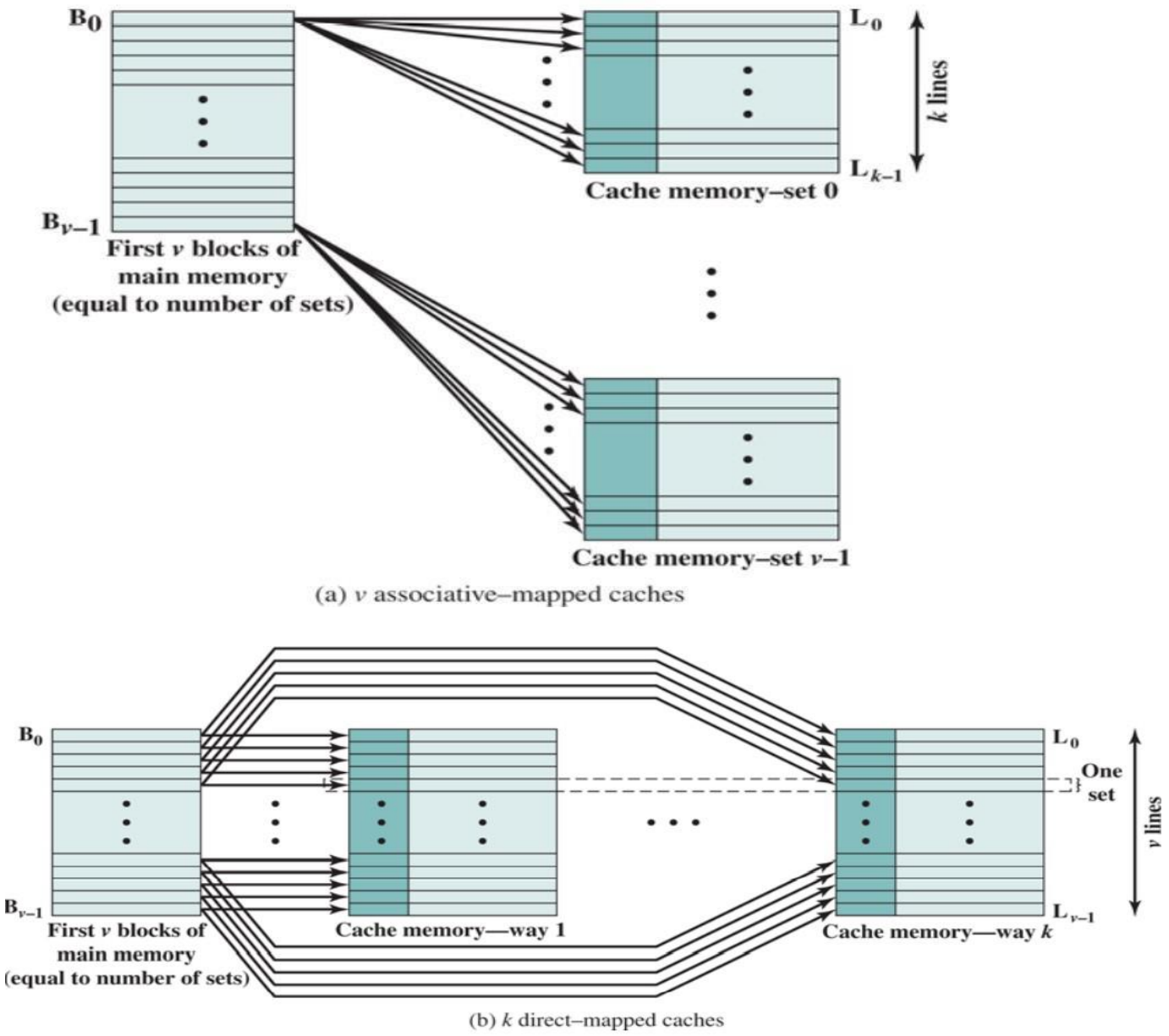


Figure 4.13 Mapping from Main Memory to Cache: *k*-Way Set Associative

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

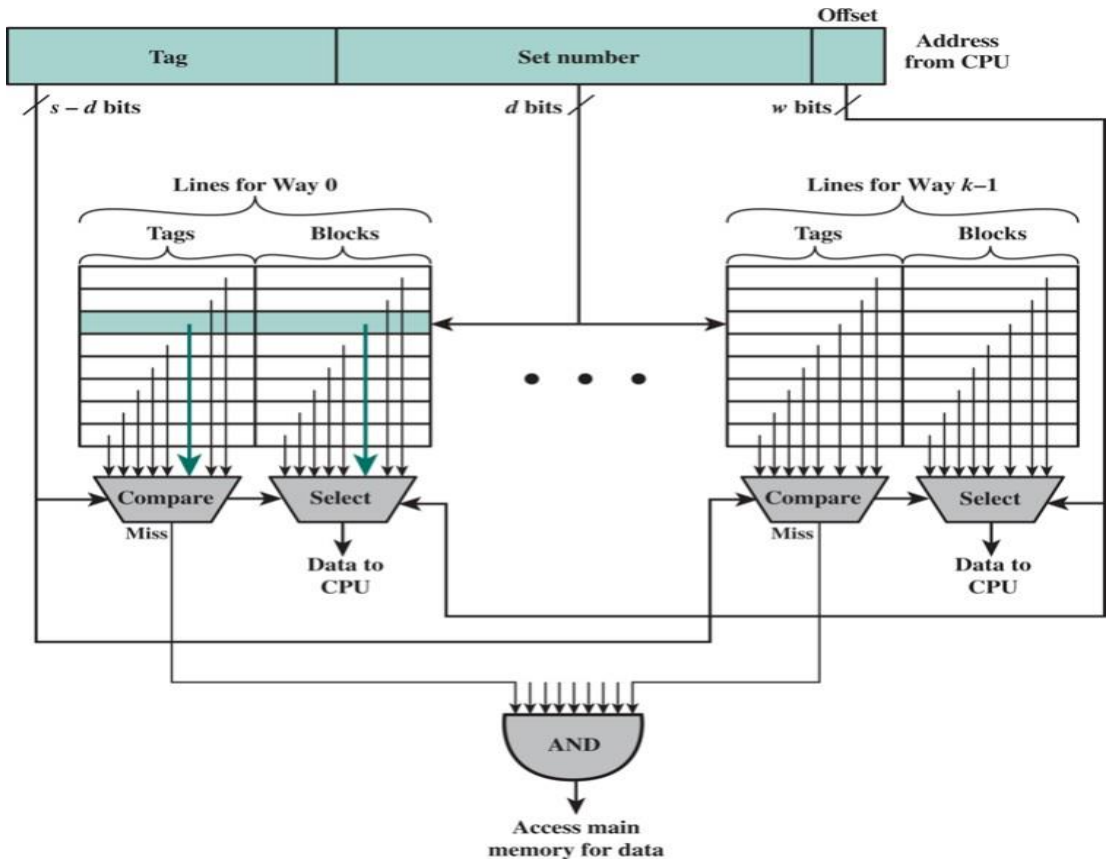


Figure 4.14 *k*-Way Set Associative Cache Organization

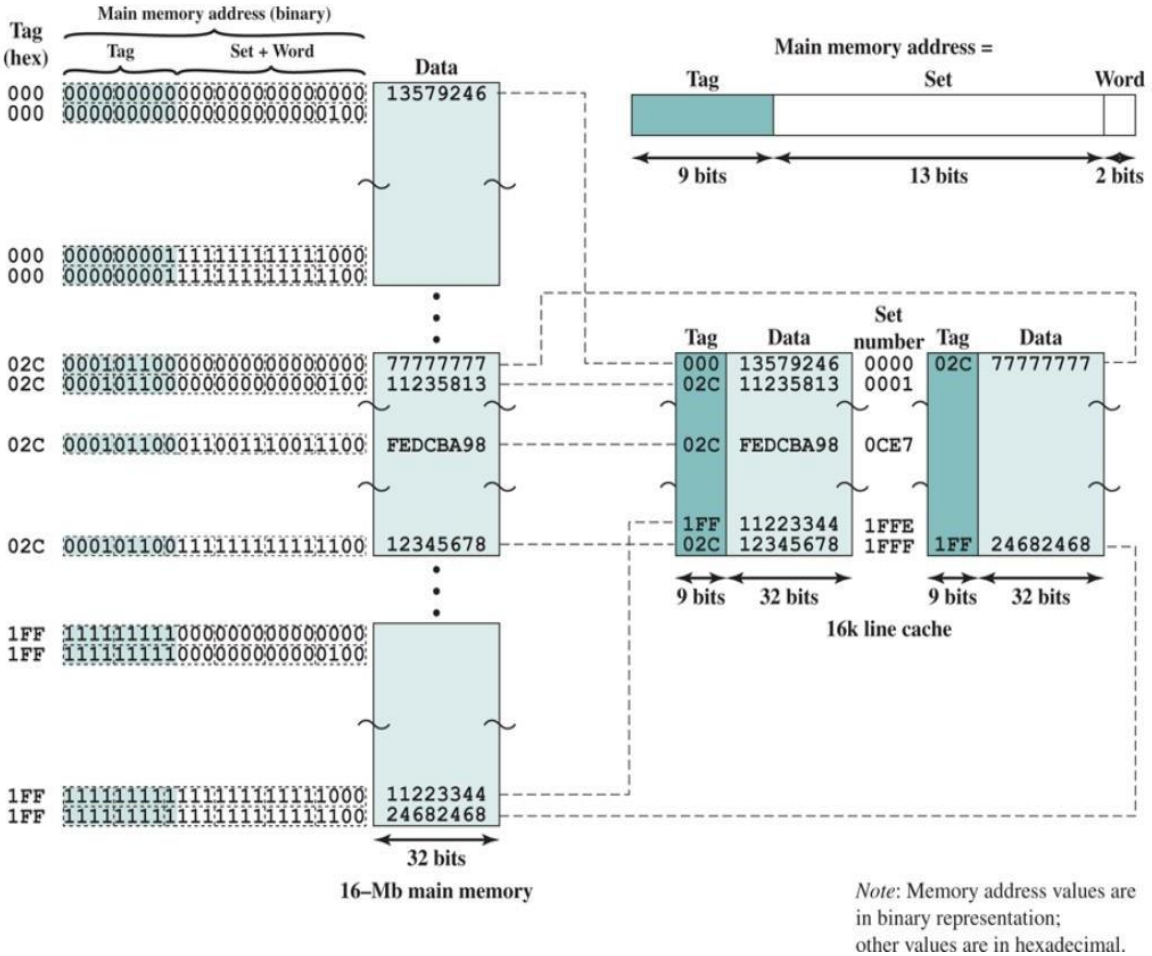


Figure 4.15 Two-Way Set-Associative Mapping Example

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

PART5: Internal Memory

5.1 Semiconductor Main Memory

In earlier computers, the most common form of random access storage for computer main memory employed an array of doughnut-shaped ferromagnetic loops referred to as cores. Hence, main memory was often referred to as core, a term that persists to this day.Today, the use of semiconductor chips for main memory is almost universal. Key aspects of this technology are explored in this section**.**

Organization

The basic element of a semiconductor memory is the memory cell. Although a variety of electronic technologies are used, all semiconductor memory cells share certain properties:

- They exhibit two stable (or semistable) states, which can be used to represent binary 1 and 0.
- They are capable of being written into (at least once), to set the state.
- They are capable of being read to sense the state.

Figure 5.1 depicts the operation of a memory cell**.** Most commonly, the cell has three functional terminals capable of carrying an electrical signal. The select terminal, as the name suggests, selects a memory cell for a read or write operation. The control terminal indicates read or write. For writing, the other terminal provides an electrical signal that sets the state of the cell to 1 or 0. For reading, that terminal is used for output of the cell's state.

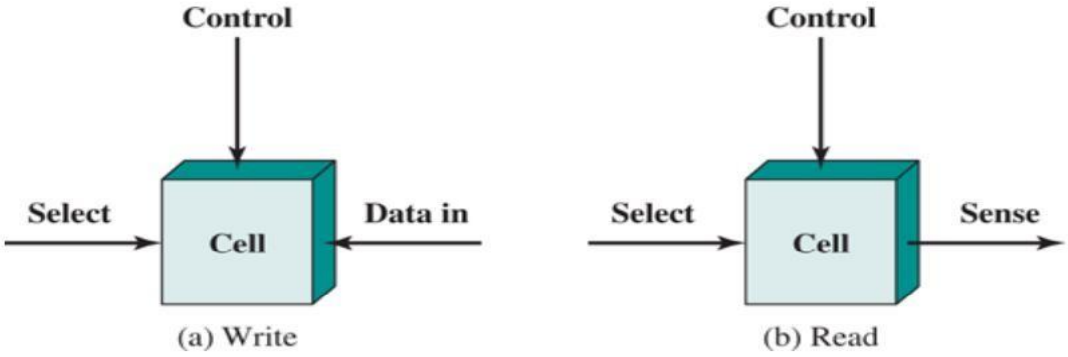


Figure 5.1 Memory Cell Operation

5.1.1 DRAM and SRAM

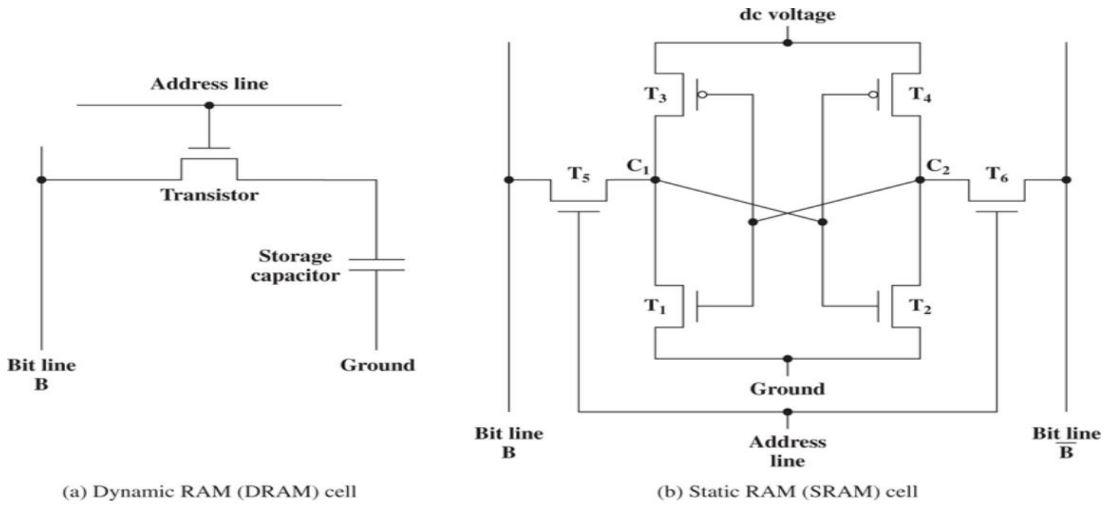
All of the memory types that we will explore in this chapter are random access. That is, individual words of memory are directly accessed through wired-in addressing logic.

DYNAMIC RAM : A dynamic RAM (DRAM) is made with cells that store data as charge on capacitors. The presence or absence of charge in a capacitor is interpreted as a binary 1 or 0. Because capacitors have a natural tendency to discharge, dynamic RAMs require periodic charge refreshing to maintain data storage. The term dynamic refers to this tendency of the stored charge to leak away, even with power continuously applied**.**See figure 5.2a

- For the write operation, a voltage signal is applied to the bit line; a high voltage represents 1, and a low voltage represents 0. A signal is then applied to the address line, allowing a charge to be transferred to the capacitor.
- For the read operation, when the address line is selected, the transistor turns on and the charge stored on the capacitor is fed out onto a bit line and to a sense amplifier. The sense amplifier compares the capacitor voltage to a reference value and determines if the cell contains a logic 1 or a logic 0. The readout from the cell discharges the capacitor, which must be restored to complete the operation.

STATIC RAM In contrast, a static RAM (SRAM) is a digital device that uses the same logic elements used in the processor. In a SRAM, binary values are stored using traditional flip-flop logic-gate configurations (see Chapter 12 for a description of flip-flops). A static RAM will hold its data as long as power is supplied to it.
Figure 5.2b is a typical SRAM structure for an individual cell. Four transistors (T1 , T2 , T3 , T4) are cross connected in an arrangement that produces a stable logic state.

- In logic state 1, point C1 is high and point C2 is low; in this state,T1 and T4 are off and T2 and T3 are on.
- In logic state 0, C1 point is low and C2 point is high; in this state, T1 and T4 are on and T2 and T3 are off. Both states are stable as long as the direct current (dc) voltage is applied. Unlike the DRAM, no refresh is needed to retain data



(a) Dynamic RAM (DRAM) cell (b) Static RAM (SRAM) cell

Figure 5.2 Typical Memory Cell Structures

5.1.2 Types of ROM

A read-only memory (ROM) contains a permanent pattern of data that cannot be changed. A ROM is nonvolatile; that is, no power source is required to maintain the bit values in memory. While it is possible to read a ROM, it is not possible to write new data into it. A ROM is created like any other integrated circuit chip,and can classify to:

PROM: When only a small number of ROMs with a particular memory content is needed, a less expensive alternative is the programmable ROM (PROM) . PROM is nonvolatile and may be written into only once. For the PROM, the writing process is performed electrically and may be performed by a supplier or customer at a time later than the original chip fabrication. PROMs provide flexibility and convenience.

EPROM :The optically erasable programmable read-only memory (EPROM) is read and written electrically, as with PROM. However, before a write operation, all the storage cells must be erased to the same initial state by exposure of the packaged chip to ultraviolet radiation. Erasure is performed by shining an intense ultraviolet light through a window that is designed into the memory chip. For comparable amounts of storage, the EPROM is more expensive than PROM, but it has the advantage of the multiple update capability.

EEPROM :A more attractive form of read-mostly memory is electrically erasable programmable read-only memory (EEPROM). This is a read-mostly memory that can be written into at any time without erasing prior contents; only the byte or bytes addressed are updated. EEPROM is more expensive than EPROM and also is less dense, supporting fewer bits per chip.

5.1.3 Chip Logic

Figure 5.3 shows a typical organization of a 16-Mbit DRAM. In this case, 4 bits are read or written at a time. Logically:

- the memory array is organized as four square arrays of 2048 by 2048 elements.
- Address lines :A total of $\log_2 W$ lines are needed. In our example,
  1. 11 address lines are needed to select one of 2048 rows. These 11 lines are fed into a row decoder, which has 11 lines of input and 2048 lines for output. The logic of the decoder activates a single one of the 2048 outputs depending on the bit pattern on the 11 input lines(2^{11}=2048) .
  2. An additional 11 address lines select one of 2048 columns
- Four data lines are used for the input and output of 4 bits to and from a data buffer.
  1. On input (write), the bit driver of each bit line is activated for a 1 or 0 according to the value of the corresponding data line.
  2. On output (read), the value of each bit line is passed through a sense amplifier and presented to the data lines. The row line selects which row of cells is used for reading or writing
- only 11 address lines (A0–A10), half the number you would expect for a array. This is done to save on the number of pins. The 22 required address lines are passed through select logic external to the chip and multiplexed onto the 11 address lines.
  1. First, 11 address signals are passed to the chip to define the row address of the array,
  2. and then the other 11 address signals are presented for the column address.
  3. These signals are accompanied by row address select(RAS) and column address select(CAS) signals to provide timing to the chip
  4. The write enable(WE) and output enable(OE) pins determine whether a write or read operation is performed.

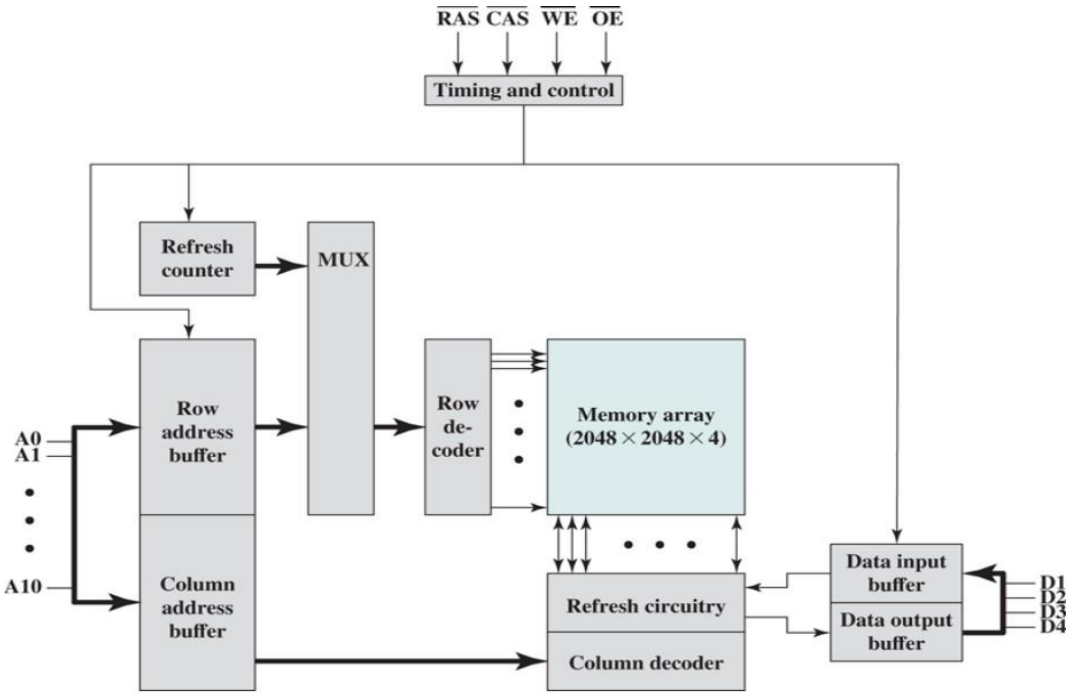


Figure 5.3 Typical 16-Mbit DRAM ($4M \times 4$)

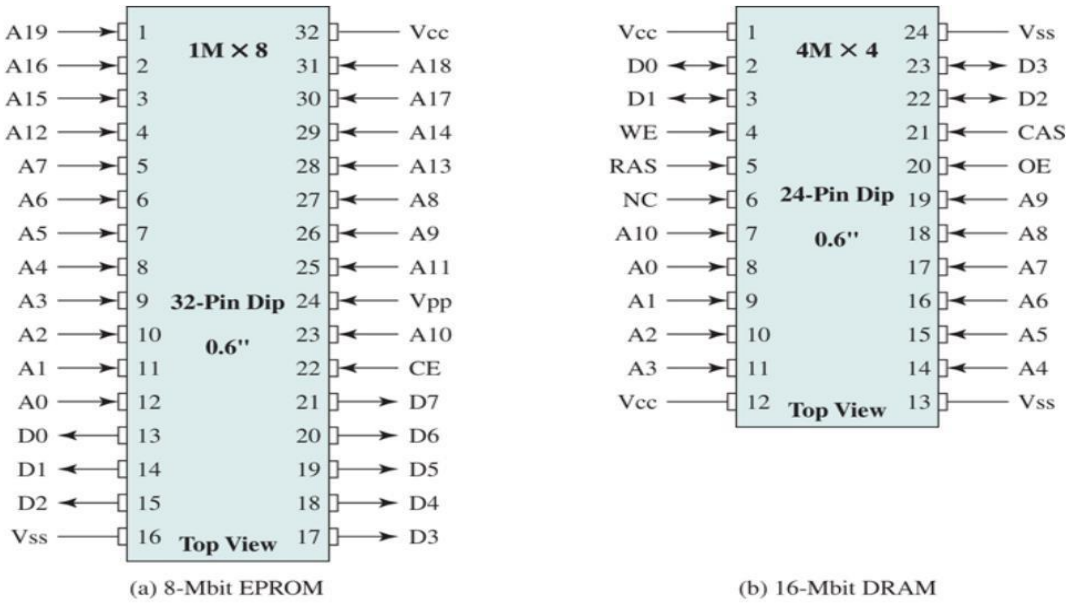
5.1.4 Chip Packaging

Figure 5.4a shows an example EPROM package, which is an 8-Mbit chip organized. The package includes 32 pins. The pins support the following signal lines:

- The address of the word being accessed. For 1M words, a total of 2^{20} pins are needed (A0–A19).
- The data to be read out, consisting of 8 lines (D0–D7).
- The power supply to the chip (V_{CC}) .
- A ground pin . A chip enable (CE) pin. Because there may be more than one memory chip, each of which is connected to the same address bus, the CE pin is used to indicate whether or not the address is valid for this chip..
- A program voltage(V_{pp})that is supplied during programming (write operations).

A typical DRAM pin configuration is shown in Figure 5.4b , for a 16-Mbit chip organized as 4M x 4. Because a RAM can be updated, the data pins are input/output.

- The write enable (WE) and output enable (OE) pins indicate whether this is a write or read operation.
- Because the DRAM is accessed by row and column, and the address is multiplexed, only 11 address pins are needed to specify the 4M row/column combinations: $2^{11} \times 2^{11} = 2^{22} = 4M$
- The row address select (RAS) and column address select (CAS) pins.
- Finally, the no connect (NC) pin is provided so that there are an even number of pins.



(a) 8-Mbit EPROM (b) 16-Mbit DRAM

Figure 5.4 Typical Memory Package Pins and Signals

5.2 Advanced DRAM Organization

In recent years, a number of enhancements to the basic DRAM architecture have been explored, and some of these are now on the market. The schemes that currently dominate the market are SDRAM, DDR-DRAM, and RDRAM**.**

5.2.1 SDRAM(Synchronous DRAM)

Synchronous dynamic random access memory (SDRAM) is dynamic random access memory (DRAM) with an interface synchronous with the system bus carrying data between the CPU and the memory controller hub. SDRAM has a rapidly responding synchronous interface, which is in sync with the system bus. SDRAM waits for the clock signal before it responds to control inputs.

- The speed of SDRAM is rated in MHz rather than in nanoseconds (ns).
- This makes it easier to compare the bus speed and the RAM chip speed.
- You can convert the RAM clock speed to nanoseconds by dividing the chip speed into 1 billion ns (which is one second). For example, an 83 MHz RAM would be equivalent to 12 ns.
- The SDRAM performs best when it is transferring large blocks of data sequentially, such as for applications like word processing, spreadsheets, and multimedia.

Table 5.1 DRAM Pin Assignments

A0 to A13	Address inputs
CLK	Clock input
CKE	Clock enable
$\overline{CS}$	Chip select
$\overline{RAS}$	Row address strobe
$\overline{CAS}$	Column address strobe
$\overline{WE}$	Write enable
DQ0 to DQ7	Data input/output
DQM	Data mask

In source-synchronous SDR interfaces, one edge of the clock, typically the rising edge, transfers the data.As shown in Figure 5.5,For the SDRAM operation, the burst length is 4 and the latency is 2. The burst read command is initiated by having and low while holding and high at the rising edge of the clock. The address inputs determine the starting column address for the burst, and the mode register sets the type of burst (sequential or interleave) and the burst length (1, 2, 4, 8, full page). The delay from the start of the command to when the data from the first cell appears on the outputs is equal to the value of the latency that is set in the mode register.

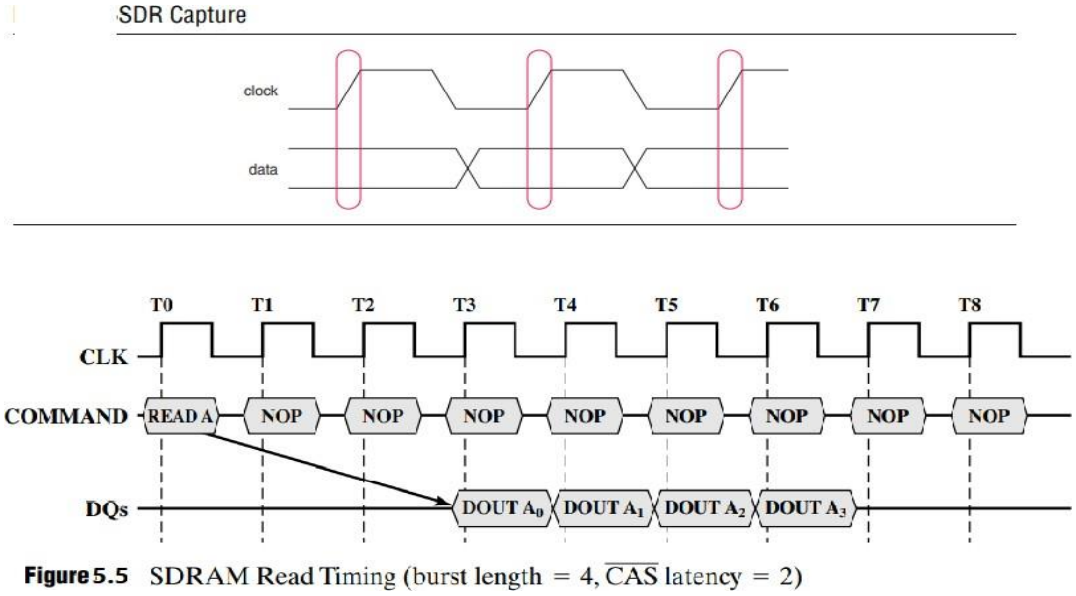


Figure 5.5 SDRAM Read Timing (burst length = 4, $\overline{\text{CAS}}$ latency = 2)

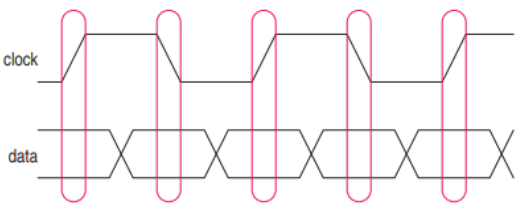
5.2.2 DDR SDRAM

SDRAM is limited by the fact that it can only send data to the processor once per bus clock cycle.A new version of SDRAM, referred to as double-data-rate SDRAM can send data twice per clock cycle, once on the rising edge of the clock pulse and once on the falling edge**.** DDR achieves higher data rates in three ways.

1. First, the data transfer is synchronized to both the rising and falling edge of the clock, rather than just the rising edge.
2. Second, DDR uses higher clock rate on the bus to increase the transfer rate.
3. Third, a buffering scheme is used, as explained subsequently.

In source-synchronous DDR interfaces, data is transferred on both edges of the clock, as shown below in Figure 5.6

Figure 5. DDR Capture



here are three significant characteristics differentiating SDRAM and DDR:

1. The main difference is the amount of data transmitted with each cycle, not the speed.
2. SDRAM sends signals once per clock cycle. DDR transfers data twice per clock cycle. (Both SDRAM and DDR use the same frequencies.)
3. SDRAM uses one edge of the clock. DDR uses both edges of the clock.

DDR SDRAM, also retroactively called DDR1 SDRAM, has been superseded by DDR2 SDRAM, DDR3 SDRAM, DDR4 SDRAM and DDR5 SDRAM. None of its successors are forward or backward compatible with DDR1 SDRAM, meaning DDR2, DDR3, DDR4 and DDR5 memory modules will not work in DDR1-equipped motherboards, and vice versa. JEDEC has thus far defined four generations of the DDR technology (Table 5.2).

Table 5.2 DDR Characteristics

	DDR1	DDR2	DDR3	DDR4
Prefetch buffer (bits)	2	4	8	8
Voltage level (V)	2.5	1.8	1.5	1.2
Front side bus data rates (Mbps)	200—400	400—1066	800—2133	2133—4266

In a prefetch buffer architecture, when a memory access occurs to a row the buffer grabs a set of adjacent data words on the row and reads them out ("bursts" them) in rapid-fire sequence on the IO pins, without the need for individual column address requests. This assumes the CPU wants adjacent datawords in memory, which in practice is very often the case. For instance, when a 64 bit CPU accesses a 16-bit-wide DRAM chip, it will need 4 adjacent 16 bit datawords to make up the full 64 bits. A 4n prefetch buffer would accomplish this exactly *("n" refers to the IO width of the memory chip; it is multiplied by the burst depth "4" to give the size in bits of the full burst sequence). An 8n prefetch buffer on a 8 bit wide DRAM would also accomplish a 64 bit transfer*..

NOTE: With data being transferred 64 bits at a time,
DDR SDRAM gives a max transfer rate (in bytes/s) = (memory bus clock rate=100MHz) × 2 (for dual rate) × 64 (number of bits transferred) / 8 (number of bits/byte)= 1600 MB/s.

5.3 Flash Memory

Another form of semiconductor memory is flash memory. Flash memory is used both for internal memory and external memory applications**.**

flash memory is intermediate between EPROM and EEPROM in both cost and functionality. Like EEPROM, flash memory uses an electrical erasing technology. An entire flash memory can be erased in one or a few seconds, which is much faster than EPROM. In addition, it is possible to erase just blocks of memory, rather than an entire chip. Flash memory gets its name because the microchip is organized so that a section of memory cells are erased in a single action or ―flash.‖

Figure 5.7 illustrates the basic operation of a flash memory. For comparison:

- Figure 5.7a depicts the operation of a transistor. Transistors exploit the properties of semiconductors so that a small voltage applied to the gate can be used to control the flow of a large current between the source and the drain.
- (Figure 5.7b),In a flash memory cell, a second gate—called a floating gate, is added to the transistor. Initially, the floating gate does not interfere with the operation of the transistor. In this state, the cell is deemed to represent binary 1.
- Applying a large voltage across the floating gate causes electrons to enter tunnel through it and become trapped on the floating gate, where they remain even if the power is disconnected (Figure 5.7c). In this state, the cell is deemed to represent binary 0.

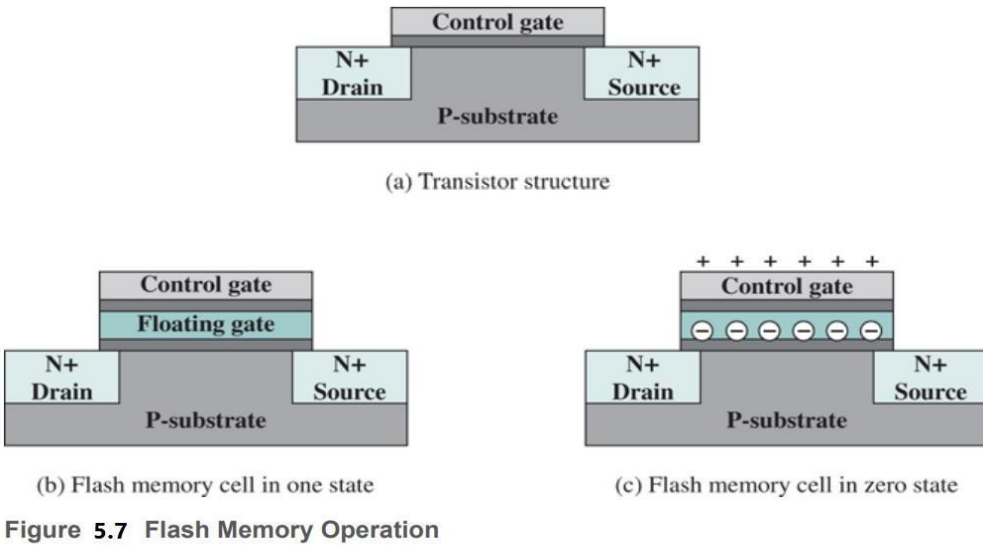
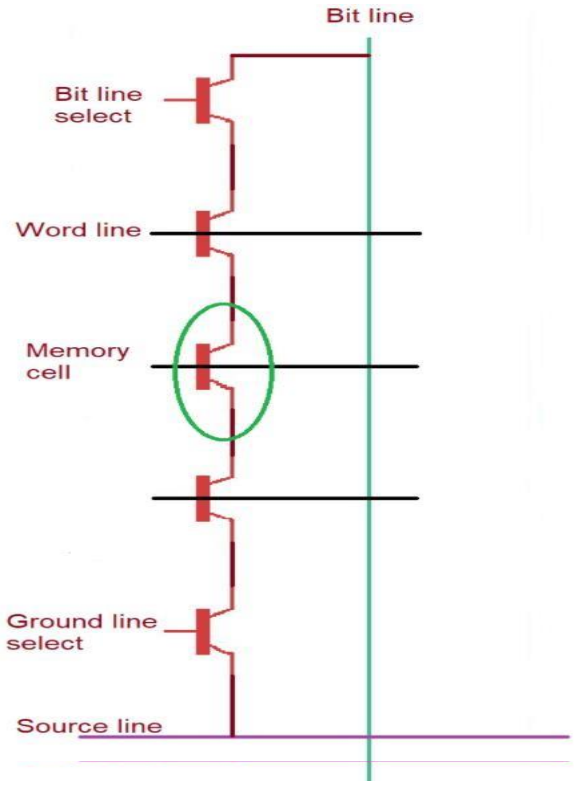


Figure 5.7 Flash Memory Operation

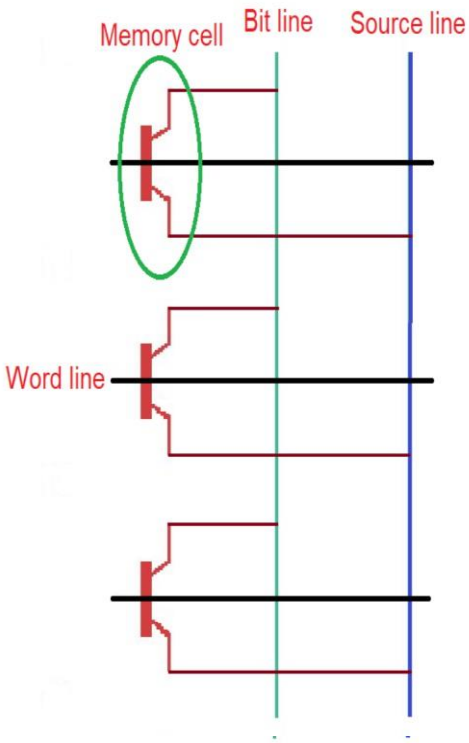
NOR and NAND Flash Memory

Flash memory architecture based on floating gate technology ,and There are two distinctive types of flash memory, designated as NOR and NAND (Figure 5.8).

- In NOR flash memory, the basic unit of access is a bit, referred to as a memory cell, every memory cell is connected to the floating gate. Cells in NOR flash are connected in parallel to the bit lines so that each cell can be read/write/erased individually. If any memory cell of the device is turned on by the corresponding word line, the bit line goes low. This is similar in function to a NOR logic gate**.** NOR memory is used for storing code and execution
- NAND flash memory is organized in transistor arrays with 16 or 32 transistors in series. The bit line goes low only if all the transistors in the corresponding word lines are turned on(several memory cells are connected in parallel.). This is similar in function to a NAND logic gate. NAND memory is used for data storage

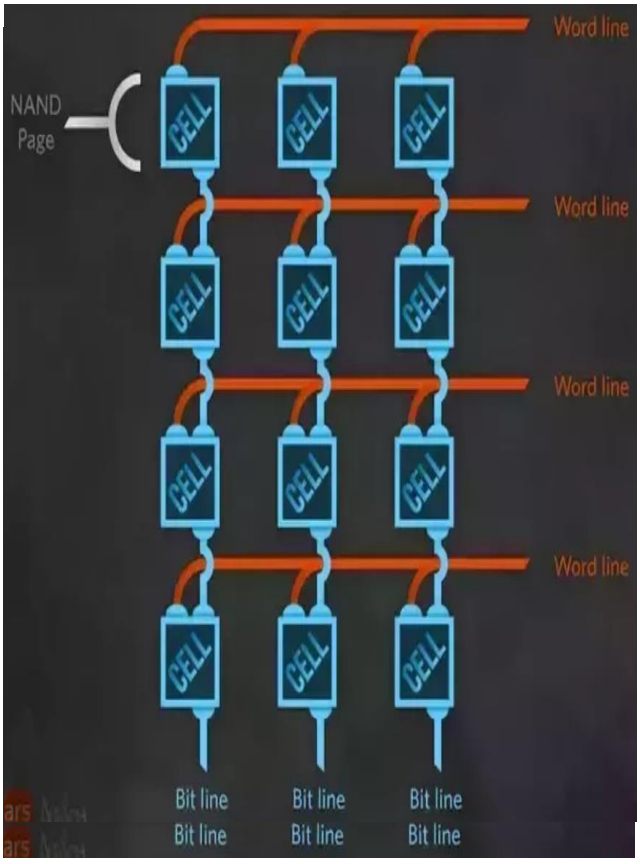


(b)NAND structure



(a)NOR structure

NAND



NOR

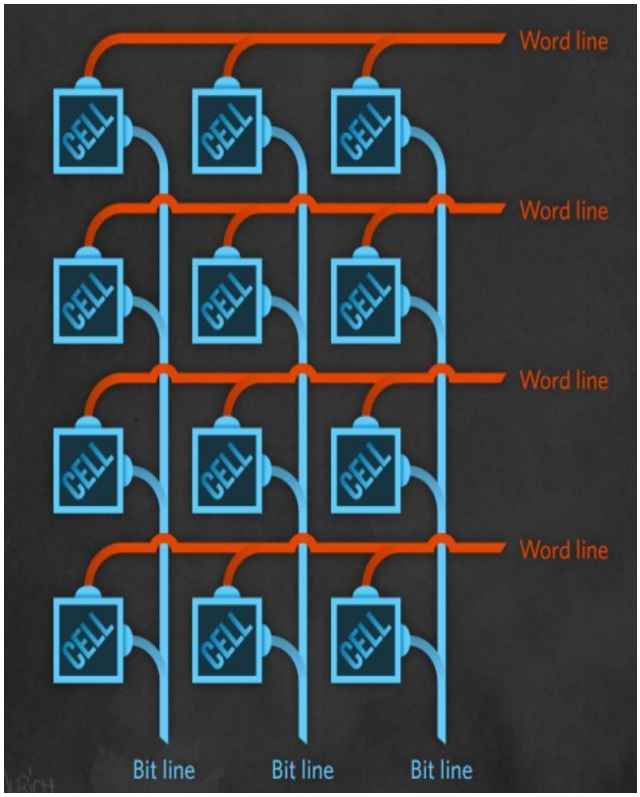


Figure 5.9 Flash Memory Structure