Logic Design and Computer Organization

IS234AT

Unit - III

Dr. Premananda B.S.

Modules

I. Synchronous Sequential Logic

II. Registers, Counters, and Memory

III. A Top-Level View of Computer Function and Interconnection, Cache Memory

IV. Input/ Output and Computer Arithmetic

V. Instruction Sets Characteristics and Functions, Processor Structure and Function, and Parallel Processing

Books

- Computer Organization and Architecture Designing for Performance, William Stallings, 10th Edition, Pearson, ISBN: 978-0134101613

Top-Level View of Computer Function and Interconnection

- Organization and Architecture
- Structure and Function
- Computer Components
- Computer Function
- Interconnection Structures
- Bus Interconnection

Cache Memory

- Computer Memory System Overview
- Cache Memory Principles
- Elements of Cache Design
- Pentium 4 Cache Organization

Computer Organization and Architecture

- **Computer architecture** refers to those attributes of a system visible to a programmer or, those attributes that have a direct impact on the logical execution of a program.
- The **instruction set architecture (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.
- Organizational attributes include those hardware details transparent to the programmer, such as control signals; interfaces between the computer and peripherals; and the memory technology used.

Computer Organization and Architecture

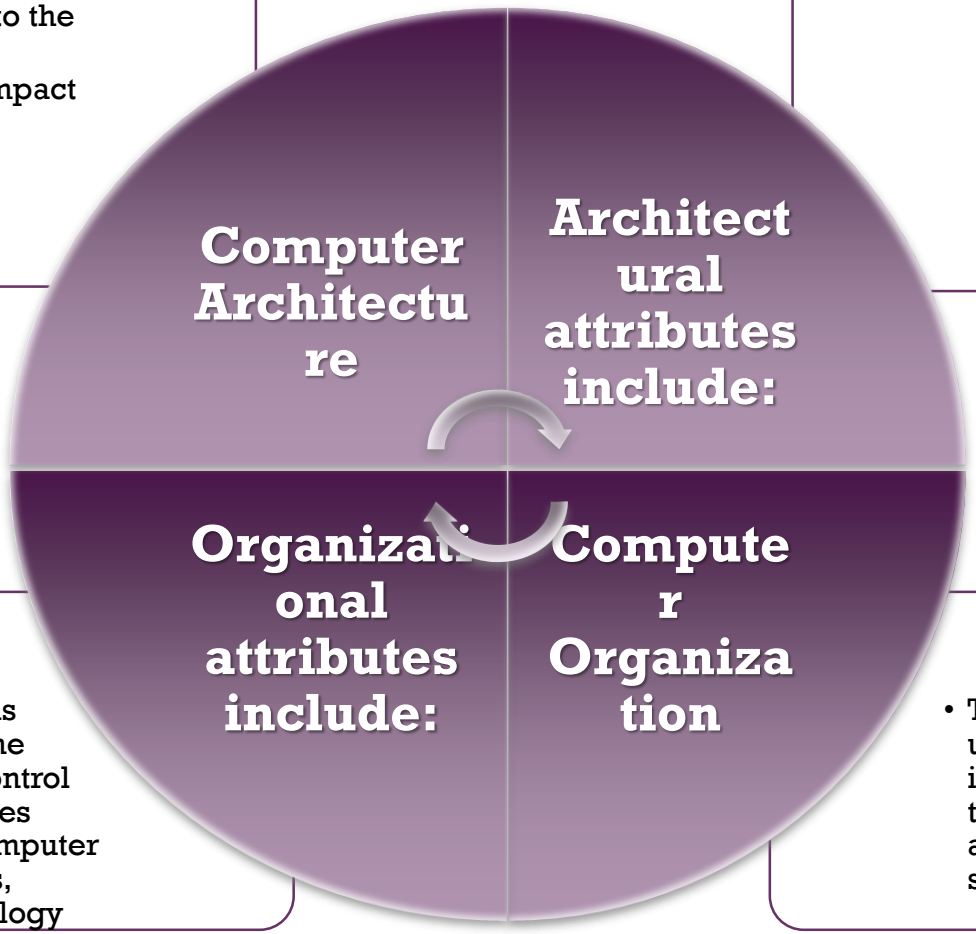
- Example: An architectural design issue is whether a computer will have a multiply instruction.
- An organizational issue is whether that instruction will be implemented by a special multiply unit or by a mechanism that makes repeated use of the add unit of the system.
- The organizational decision may be based on the:
  - frequency of use of the multiply instruction
  - The relative speed of the two approaches
  - cost and physical size of a special multiple-unit
- Many computer manufacturers offer a family of computer models, all with the same architecture but with differences in organization.
  - Different models have different price and performance characteristics.

Computer Architecture

Computer Organization

- Attributes of a system visible to the programmer
- Have a direct impact on the logical execution of a program

- Instruction set, number of bits used to represent various data types, I/O mechanisms, techniques for addressing memory



- Hardware details transparent to the programmer, control signals, interfaces between the computer and peripherals, memory technology used

- The operational units and their interconnections that realize the architectural specifications

Example:

- A particular architecture may span many years and encompass several different computer models, its organization changing with changing technology.

IBM System/370 architecture

- Introduced in 1970
- Included several models
- Could upgrade to a more expensive, faster model without having to abandon original software
- New models are introduced with improved technology but retain the same architecture so that the customer's software investment is protected
- Architecture has survived to this day as the architecture of IBM's mainframe product line

Top-Level View of Computer Function and Interconnection

- Organization and Architecture
- **Structure and Function**
- Computer Components
- Computer Function
- Interconnection Structures
- Bus Interconnection

Structure and Function

- Hierarchical system
  - Set of interrelated subsystems
- Hierarchical nature of complex systems is essential to both their design and their description
- Designers need to deal with a particular level of the system at a time:
  - The behavior at each level depends on an abstract characterization of the system at the next lower level.
  - Concerned with structure and function at each level

- Structure
  - The way in which components relate to each other
- Function
  - The operation of each individual components as part of the structure

Structure and Function

- At each level, the designer is concerned with structure and function.
- In terms of description, we have two choices:
  - Start at the bottom and building up to a complete description, or
  - Begin with a top view and decompose the system into its subparts.
- The top-down approach is the clearest and most effective.
- The computer system will be described from the top down.

- First, the major components of a computer are described with their structure and function and proceed to lower layers of the hierarchy.

Function

- Four basic functions that a computer can perform:
  - **Data processing**
    - Data may take a wide variety of forms and the range of processing requirements is broad
  - **Data storage**
    - Short-term
    - Long-term
  - **Data movement**
    - Input-output (I/O) - when data are received from or delivered to a device (peripheral) that is directly connected to the computer
    - Data communications – when data are moved over longer distances, to or from a remote device
  - **Control**
    - A control unit manages the computer's resources and manage the performance of its functional parts in response to instructions

Structure

- Figure 1.1 provides a hierarchical view of the internal structure of a traditional *simple single-processor computer*.
- Main structural components are:
  - CPU
  - Main memory
  - I/O
  - System interconnections

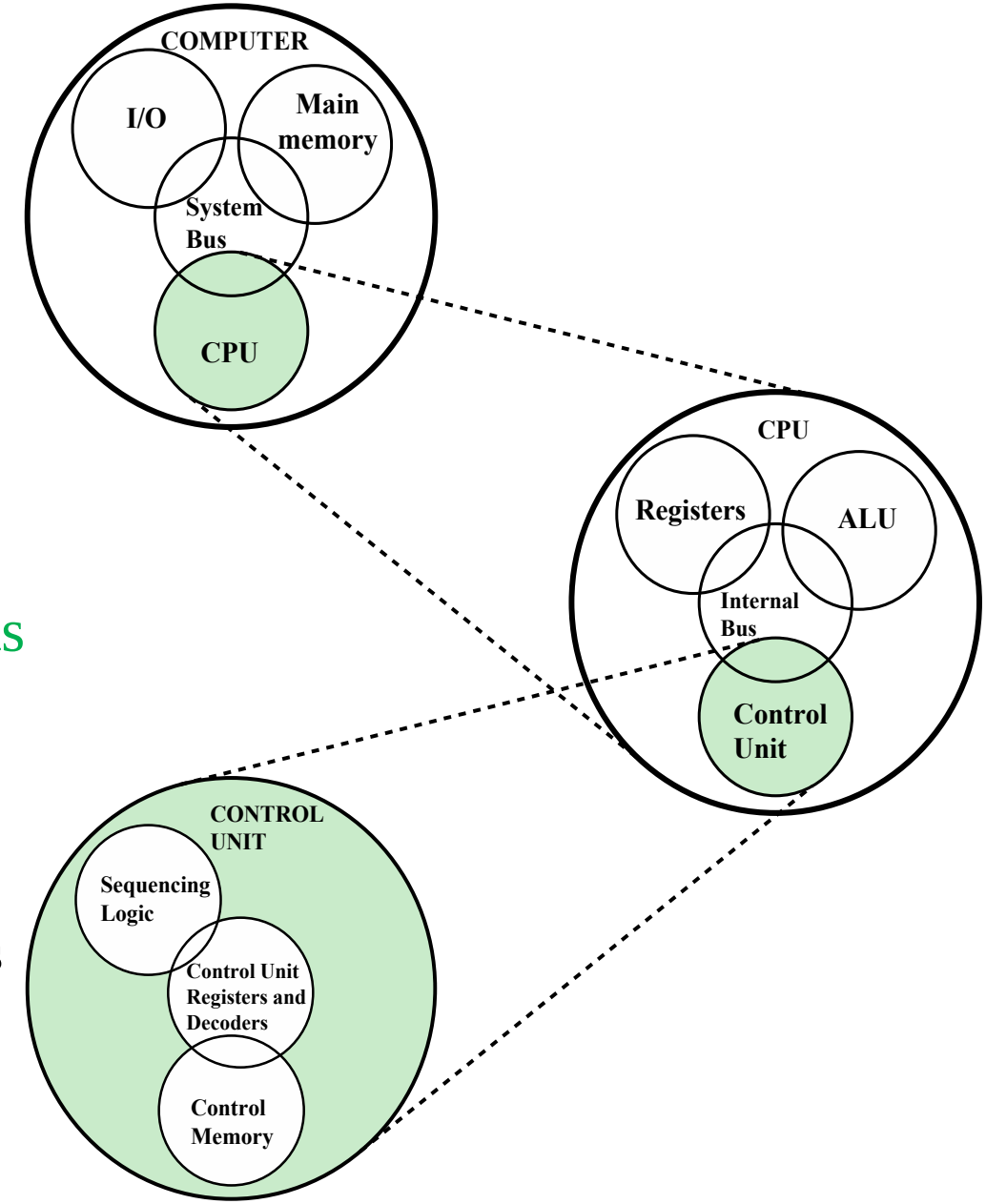
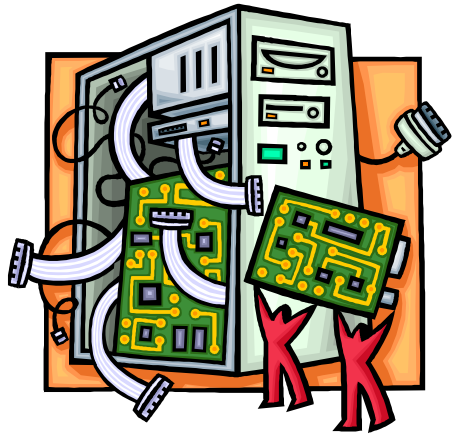


Figure 1.1 A Top-Down View of a Computer

Structural components of the Computer

- CPU – controls the operation of the computer and performs its data processing functions
- Main Memory – stores data
- I/O – moves data between the computer and its external environment
- System Interconnection – some mechanism that provides for communication among CPU, main memory, and I/O
  - Example: A system bus, consisting of a number of conducting wires to which all the other components attach.



Central Processing Unit (CPU)

Major structural components:

- Control Unit
  - Controls the operation of the CPU and hence the computer
- Arithmetic and Logic Unit (ALU)
  - Performs the computer's data processing function
- Registers
  - Provide storage internal to the CPU
- CPU Interconnection
  - Some mechanism that provides for communication among the control unit, ALU, and registers

Multicore Computer Structure

- Present-day computers have multiple processors.
- When these processors all reside on a single chip, the term *multicore computer* is used.
- Each processing unit (consisting of a control unit, ALU, registers, and cache) is called a *core*.

Multicore Computer Structure

- The central processing unit (CPU)
  - A portion of the computer that fetches and executes instructions
  - Consists of an ALU, a control unit, and registers
  - Referred to as a processor in a system with a single processing unit
- Core
  - An individual processing unit on a processor chip
  - May be equivalent in functionality to a CPU on a single-CPU system
  - Specialized processing units are also referred to as cores
- Processor
  - A physical piece of silicon containing one or more cores
  - Computer component that interprets and executes instructions
  - Referred to as a *multicore processor* if it contains multiple cores

Cache Memory

- The modern computers use multiple layers of memory, called *cache memory*, between the processor and main memory.
- Cache memory is smaller and faster than main memory
- Used to speed up memory access by placing in the cache data from main memory, to be used in the near future
- A performance improvement may be obtained by using multiple levels of cache, with level 1 (L1) closest to the core and additional levels (L2, L3, etc.) progressively farther from the core
- The level n is smaller and faster than level $n + 1$.

Major Elements of a Multicore Computer

- Most computers, including embedded computers in smartphones and tablets, plus personal computers, laptops, and workstations, are housed on a motherboard.
- A printed circuit board (PCB) is a flat board that holds and interconnects chips and other electronic components.
- The main PCB in a computer is called a system or motherboard, while smaller ones that plug into the slots in the main board are called expansion boards.
- A chip, upon which electronic circuits and logic gates are fabricated, the resulting product is referred to as an *integrated circuit* .
- The motherboard contains a slot or socket for the processor chip, which contains multiple individual cores, in what is known as a multicore processor .
- Slots for memory chips, I/O controller chips, and other computer components.
- A motherboard connects a few individual chip components, with each chip containing up to hundreds of millions of transistors.
- Figure 1.2 shows a view of the principal components of a multicore computer.

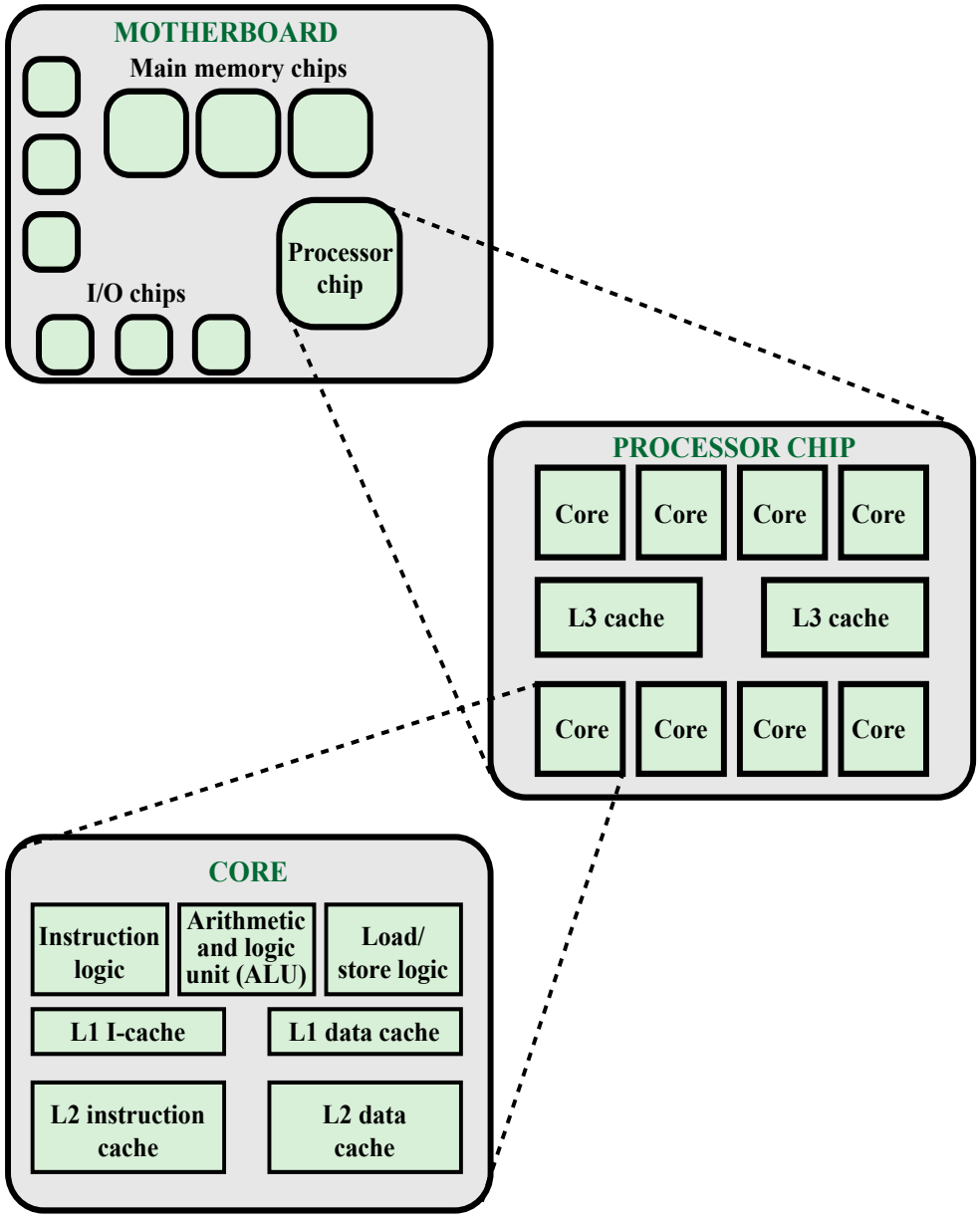


Figure 1.2 Simplified View of Major Elements of a Multicore Computer

Major Elements of a Multicore Computer

- The functional elements of a core are:
  - **Instruction logic:** This includes the tasks involved in fetching instructions and decoding each instruction to determine the instruction operation and the memory locations of any operands.
  - **ALU:** Performs the operation specified by an instruction.
  - **Load/store logic:** Manages the transfer of data to and from main memory via cache.
- The core also contains an L1 cache, split between an instruction cache (I-cache) that is used for the transfer of instructions to and from main memory, and an L1 data cache, for the transfer of operands and results.
- Processor chips also include an L2 cache as part of the core.
  - This cache can be split between instruction and data caches.

Motherboard with Two Intel Quad-Core Xeon Processors

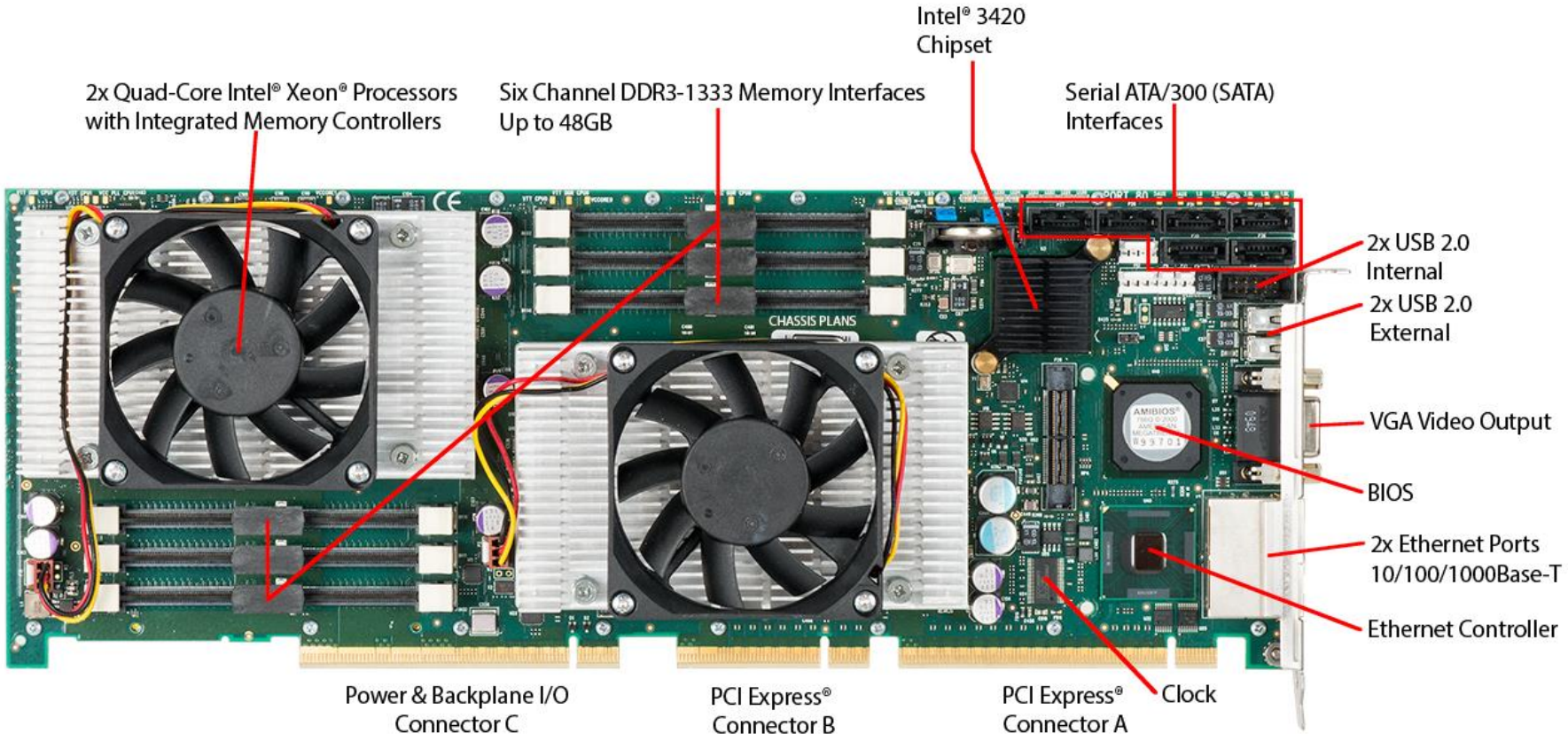
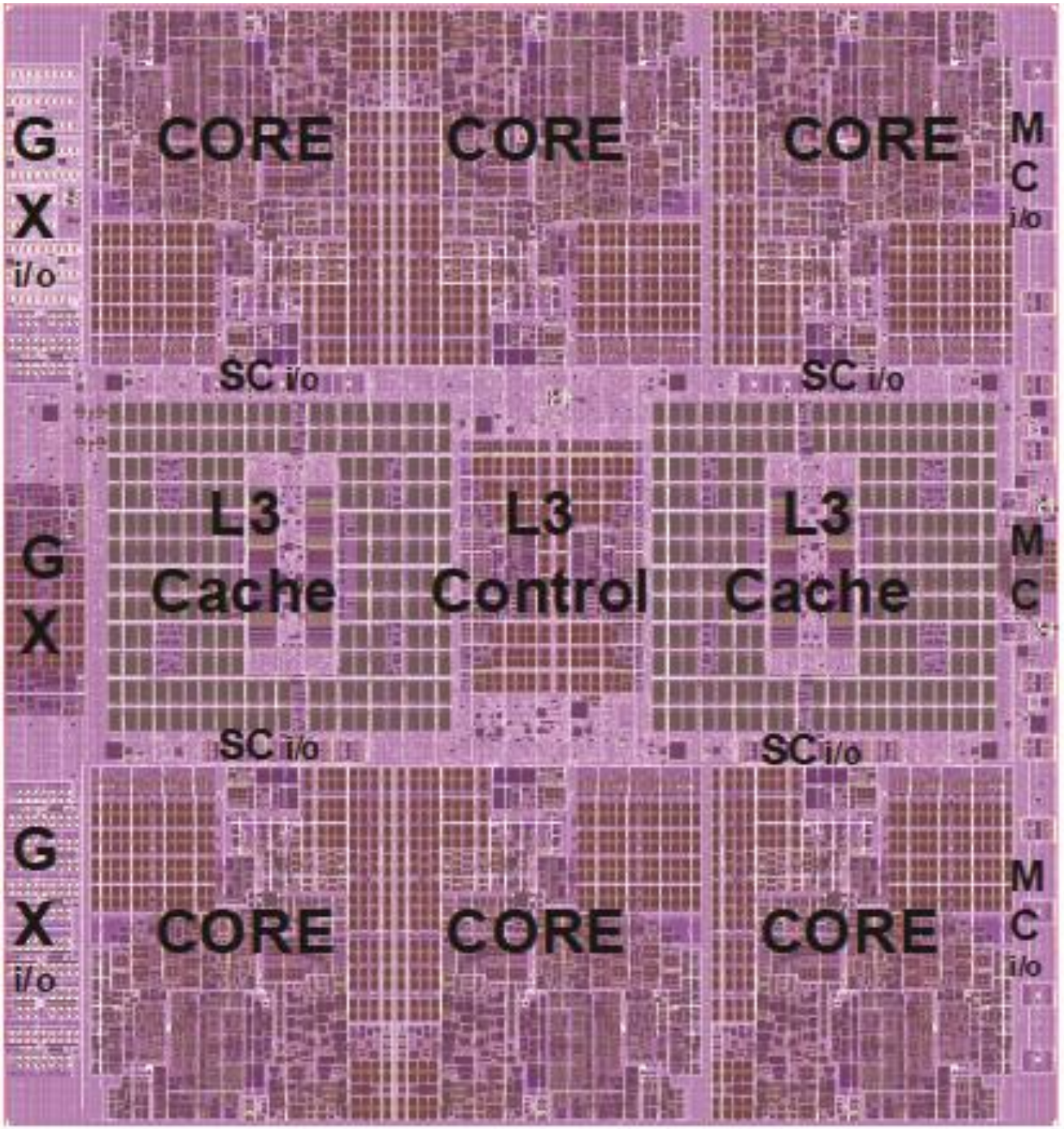


Figure. The motherboard for a computer built around two Intel Quad-Core Xeon processor chips.

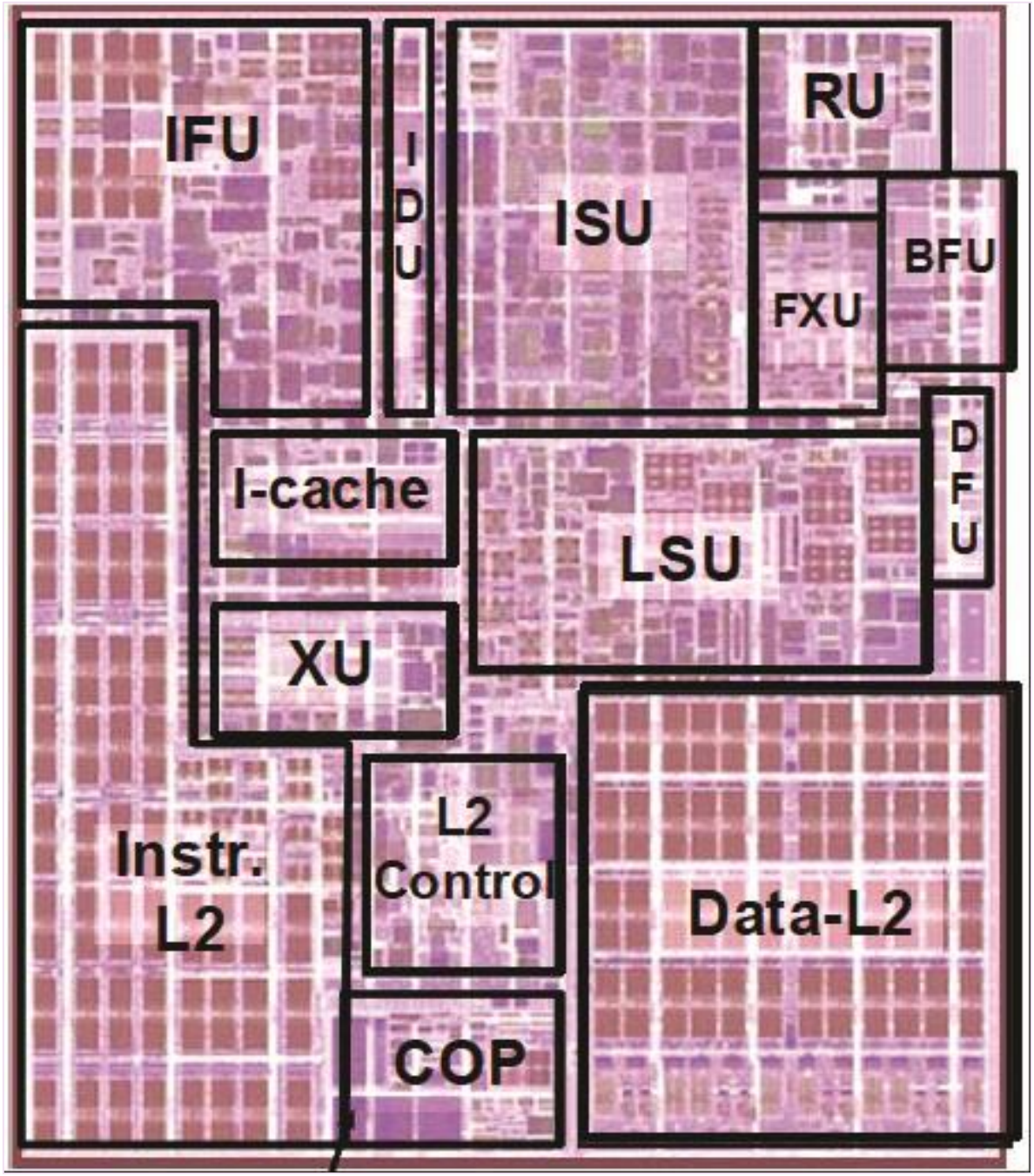
Motherboard with Two Intel Quad-Core Xeon Processors

- Figure next shows a photograph of the motherboard for a computer built around two Intel Quad-Core Xeon processor chips.
- PCI-Express slots for a high-end display adapter and for additional peripherals.
- Ethernet controller and Ethernet ports for network connections.
- USB sockets for peripheral devices.
- Serial ATA (SATA) sockets for connection to disk memory.
- Interfaces for DDR (double data rate) main memory.
- Intel 3420 chipset is an I/O controller for direct memory access operations between peripheral devices and main memory.



Figure

zEnterprise
EC12 Processor
Unit (PU)
Chip Diagram



Figure

zEnterprise
EC12
Core Layout

zEnterprise EC12 Core

- ISU (instruction sequence unit):
- IFU (instruction fetch unit):
- IDU (instruction decode unit):
- LSU (load-store unit):
- XU (translation unit):
- FXU (fixed-point unit):
- BFU (binary floating):
- DFU (decimal floating-point unit):
- RU (recovery unit):
- COP (dedicated co-processor): :
- L2 control:
- L2D:
- L2I:

Top-Level View of Computer Function and Interconnection

- Organization and Architecture
- Structure and Function
- **Computer Components**
- Computer Function
- Interconnection Structures
- Bus Interconnection

Computer Components

- Computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton.
- Referred to as *von Neumann architecture* and is based on three key concepts:
  - Data and instructions are stored in a single read-write memory
  - The contents of this memory are addressable by location, without regard to the type of data contained there
  - Execution occurs sequentially from one instruction to the next
- *Hardwired program*
  - The result of the process of connecting the various components in the desired configuration

Hardware and Software Approaches

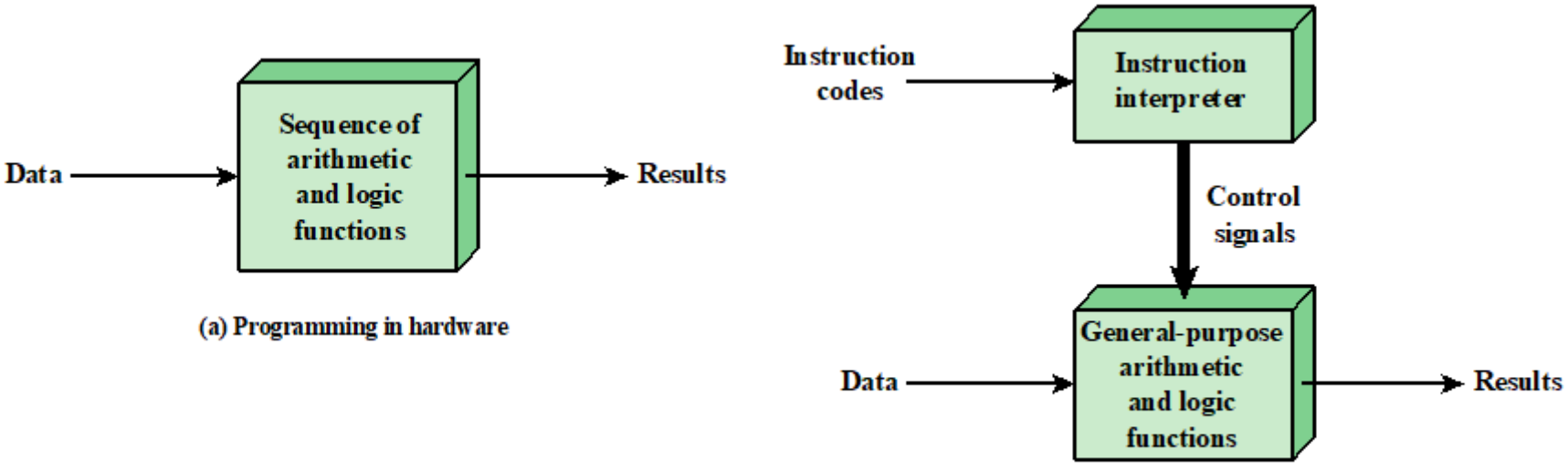


Figure 3.1 Hardware and Software Approaches

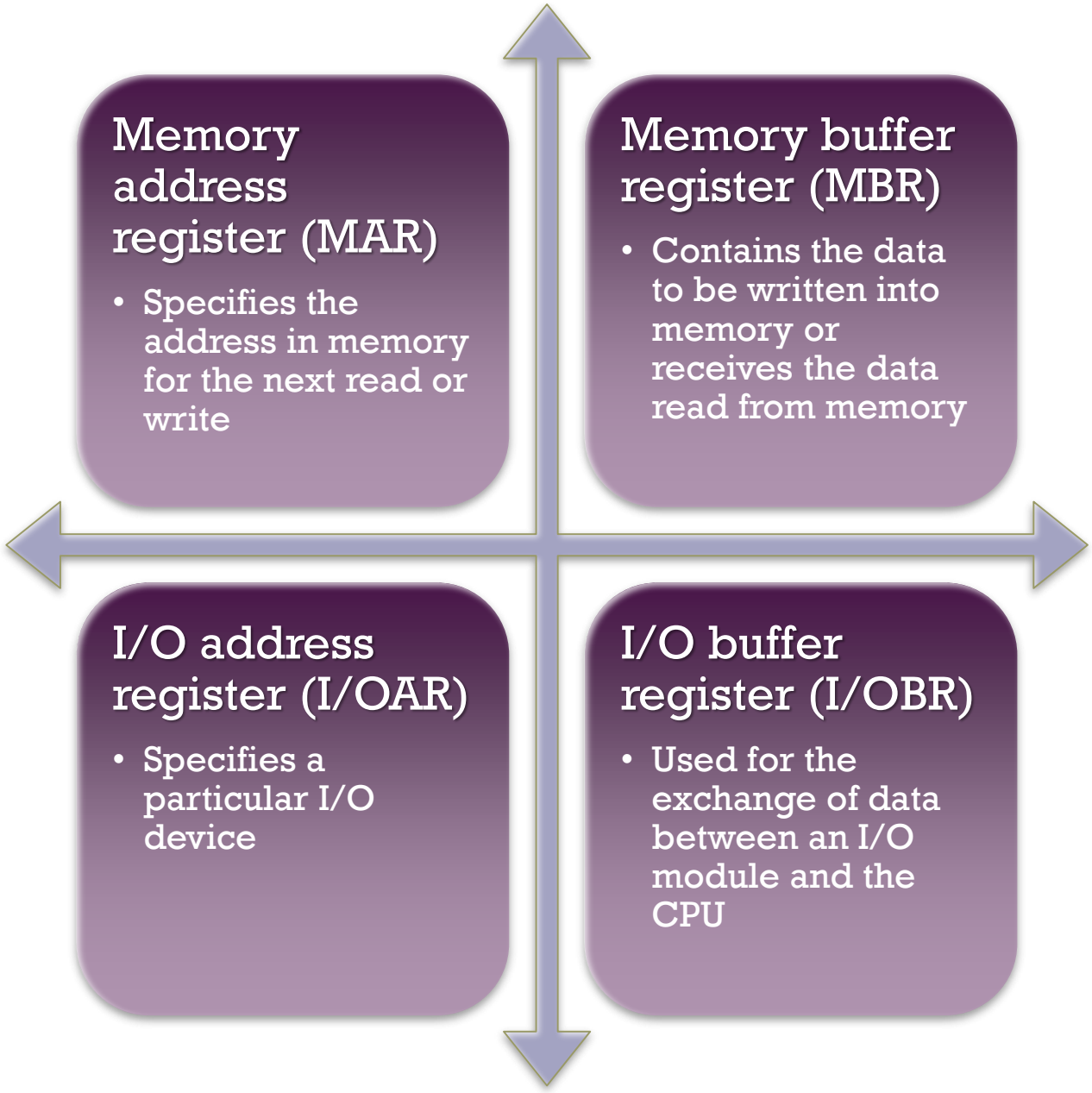
- In customized hardware, system accepts data and produces results (Figure 3.1a).
- Instead of rewiring the hardware for each new program, the programmer merely needs to supply a new set of control signals.
- The entire program is a sequence of steps.
- At each step, some arithmetic or logical operation is performed on some data.
- Provide a unique code for each possible set of control signals, and add to the general-purpose hardware a segment that can accept a code and generate control signals (Figure 3.1b).

Software

- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

Major components:

- CPU
  - Instruction interpreter
  - Module of general-purpose arithmetic and logic functions
- I/O Components
  - Input module
    - Contains basic components for accepting data and instructions and converting them into an internal form of signals usable by the system
  - Output module
    - Means of reporting results



MEMORY

MAR

MBR

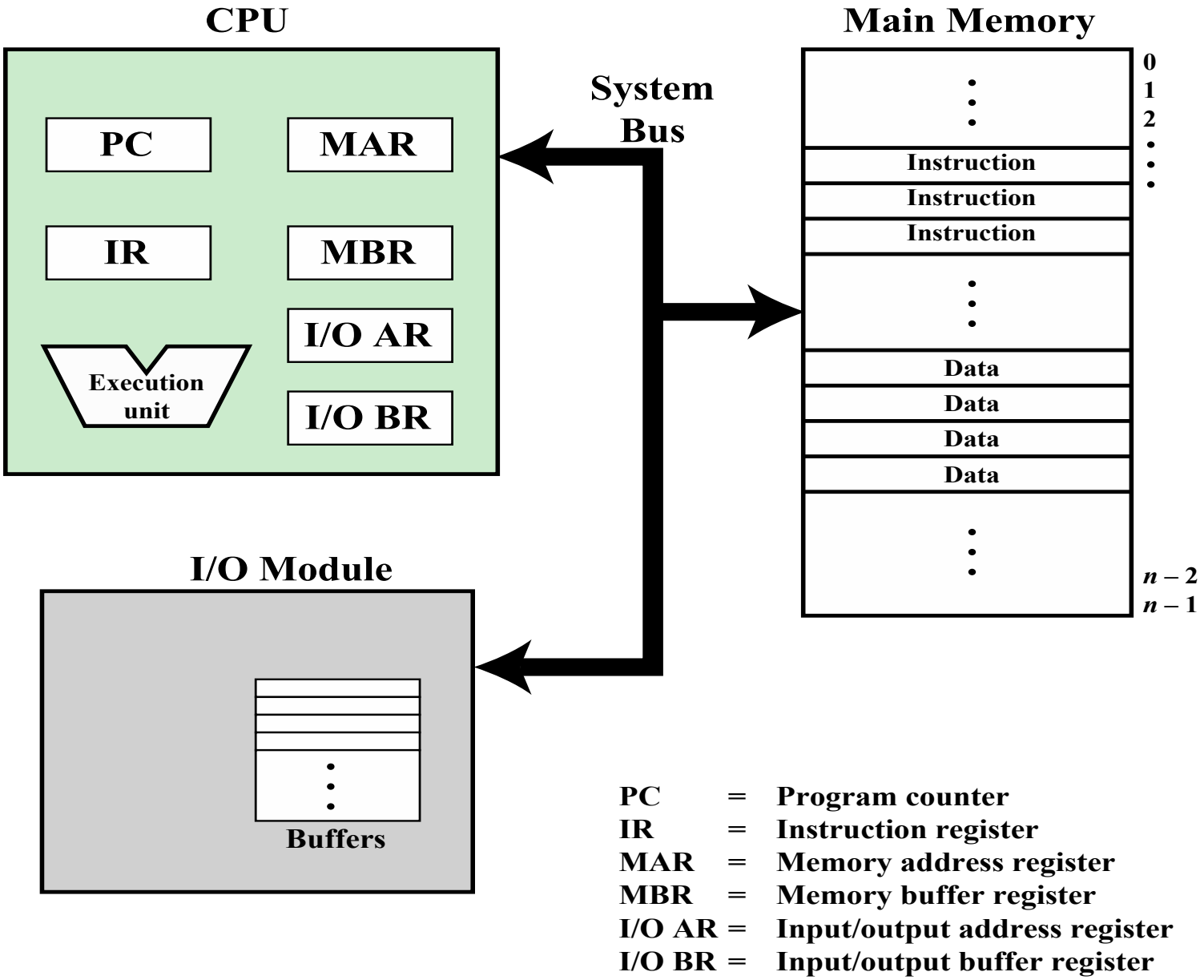


Figure 3.2 Computer Components: Top-Level View

Top-Level View of Computer Function and Interconnection

- Organization and Architecture
- Structure and Function
- Computer Components

- **Computer Function**

- Interconnection Structures
- Bus Interconnection

Computer Function

- The processing required for a single instruction is called an **instruction cycle**.
- Using the simplified two-step description given, the instruction cycle is depicted in Figure.
- The two steps are referred to as the **fetch** and the **execute cycle**.
- Program execution halts only if the machine is turned off, some sort of unrecoverable error occurs, or a program instruction that halts the computer is encountered.

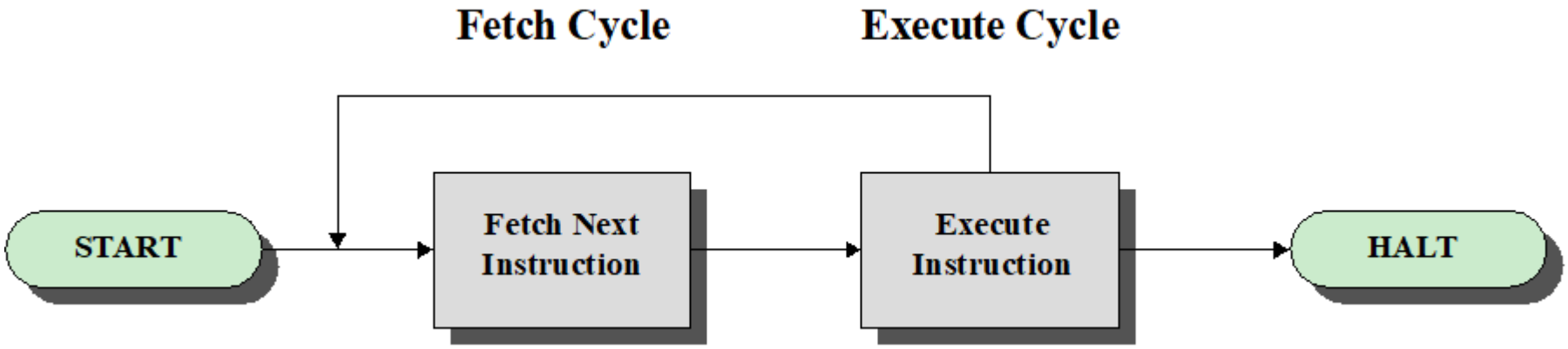


Figure 3.3 Basic instruction cycle

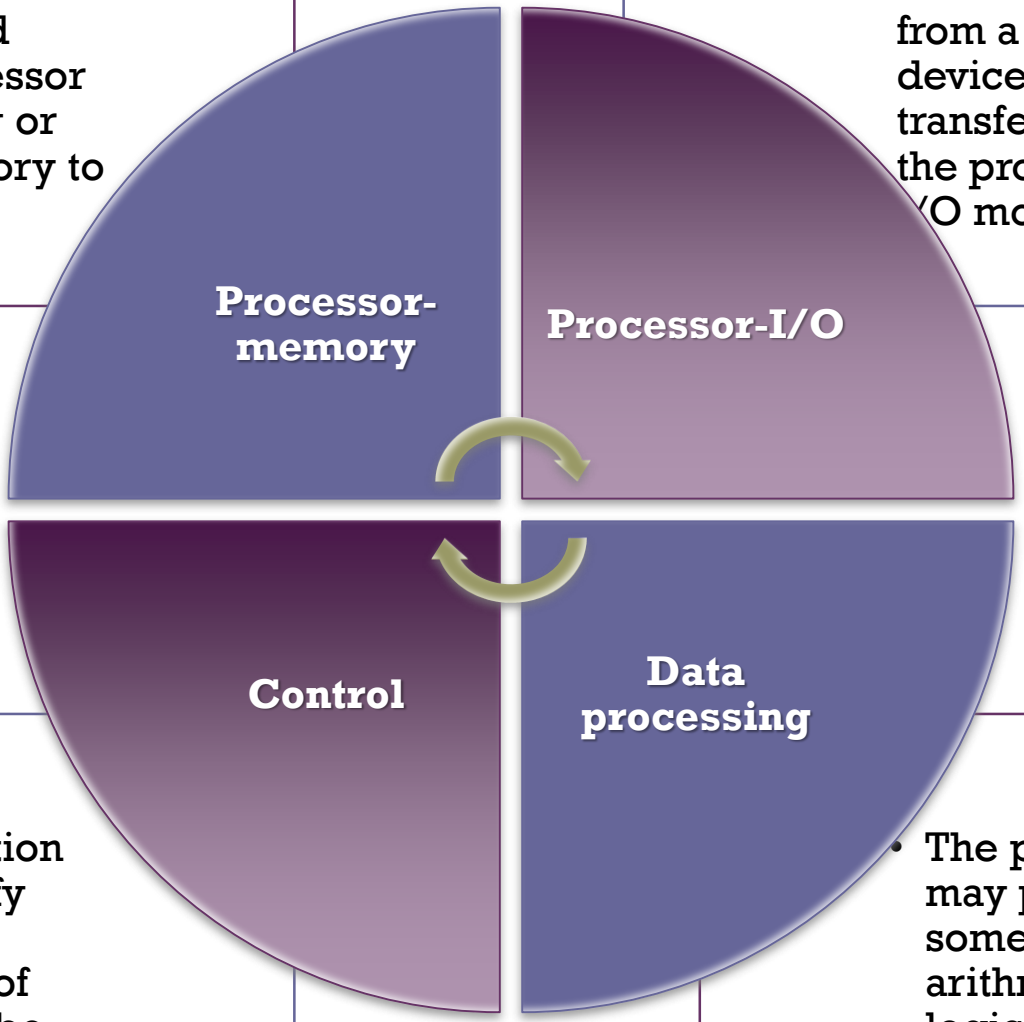
Instruction Fetch and Execute

- At the beginning of each instruction cycle the processor fetches an instruction from memory.
- The *program counter* (PC) holds the address of the instruction to be fetched next.
- The processor increments the PC after each instruction fetch so that it will fetch the next instruction in sequence.
- The fetched instruction is loaded into the *instruction register* (IR).
- The processor interprets the instruction and performs the required action.

Action Categories

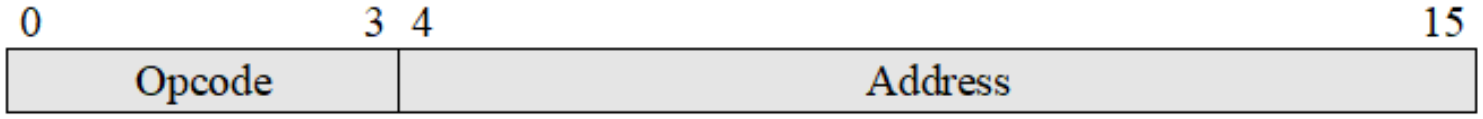
- Data transferred from processor to memory or from memory to processor

- Data transferred to or from a peripheral device by transferring between the processor and an I/O module

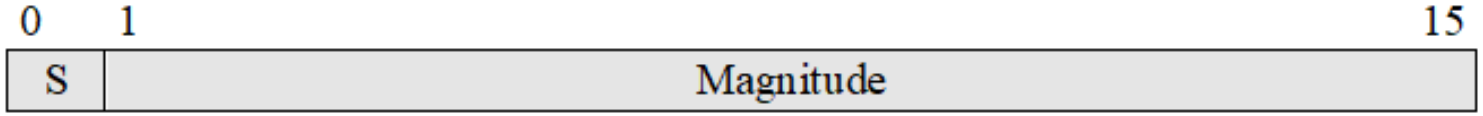


- An instruction may specify that the sequence of execution be altered

- The processor may perform some arithmetic or logic operation on data



(a) Instruction format



(b) Integer format

Program Counter (PC) = Address of instruction
Instruction Register (IR) = Instruction being executed
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

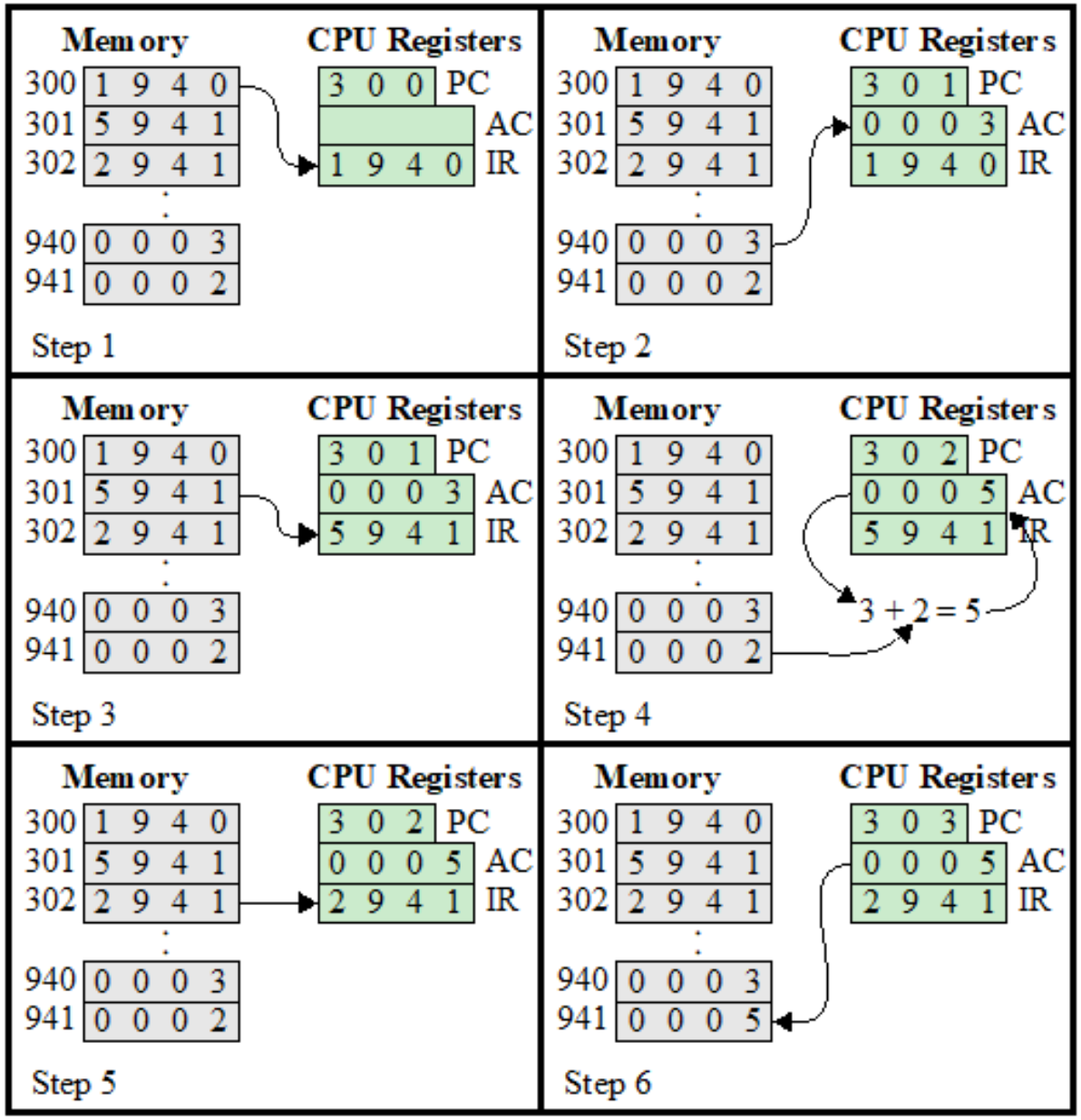
0001 = Load AC from Memory
0010 = Store AC to Memory
0101 = Add to AC from Memory

(d) Partial list of opcodes

Figure 3.4 Characteristics of a Hypothetical Machine

Partial Program Execution: Example

Adds the contents of the memory word at address 940 to the contents of the memory word at address 941 and stores the result in the latter location.



Instruction Cycle

- Figure 3.6 provides a detailed view of the basic instruction cycle.
- The states can be described as follows:
  - **Instruction address calculation (iac):** Determine the address of the next instruction to be executed. This involves adding a fixed number to the address of the previous instruction.
  - **Instruction fetch (if):** Read instruction from its memory location into the processor.
  - **Instruction operation decoding (iod):** Analyze instruction to determine the type of operation to be performed and operand(s) to be used.
  - **Operand address calculation (oac):** If the operation involves reference to an operand in memory or is available via I/O, then determine the address of the operand.
  - **Operand fetch (of):** Fetch the operand from memory or read it in from I/O.
  - **Data operation (do):** Perform the operation indicated in the instruction.
  - **Operand store (os):** Write the result into memory or out to I/O.

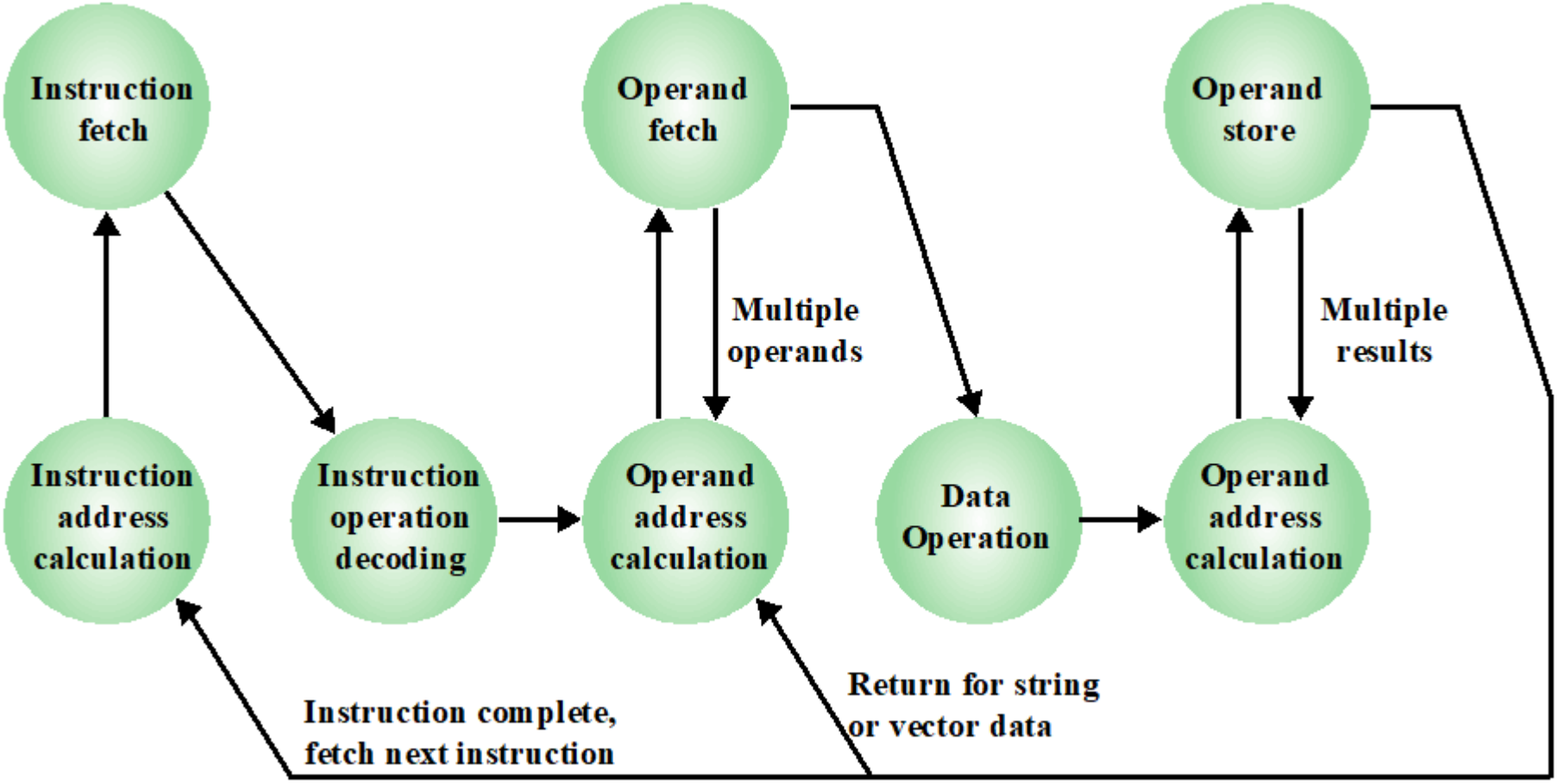


Figure 3.6 Instruction Cycle State Diagram

- States in the upper part of Figure involve an exchange between the processor and either memory or an I/O module.
- States in the lower part of the diagram involve only internal processor operations.

Interrupts

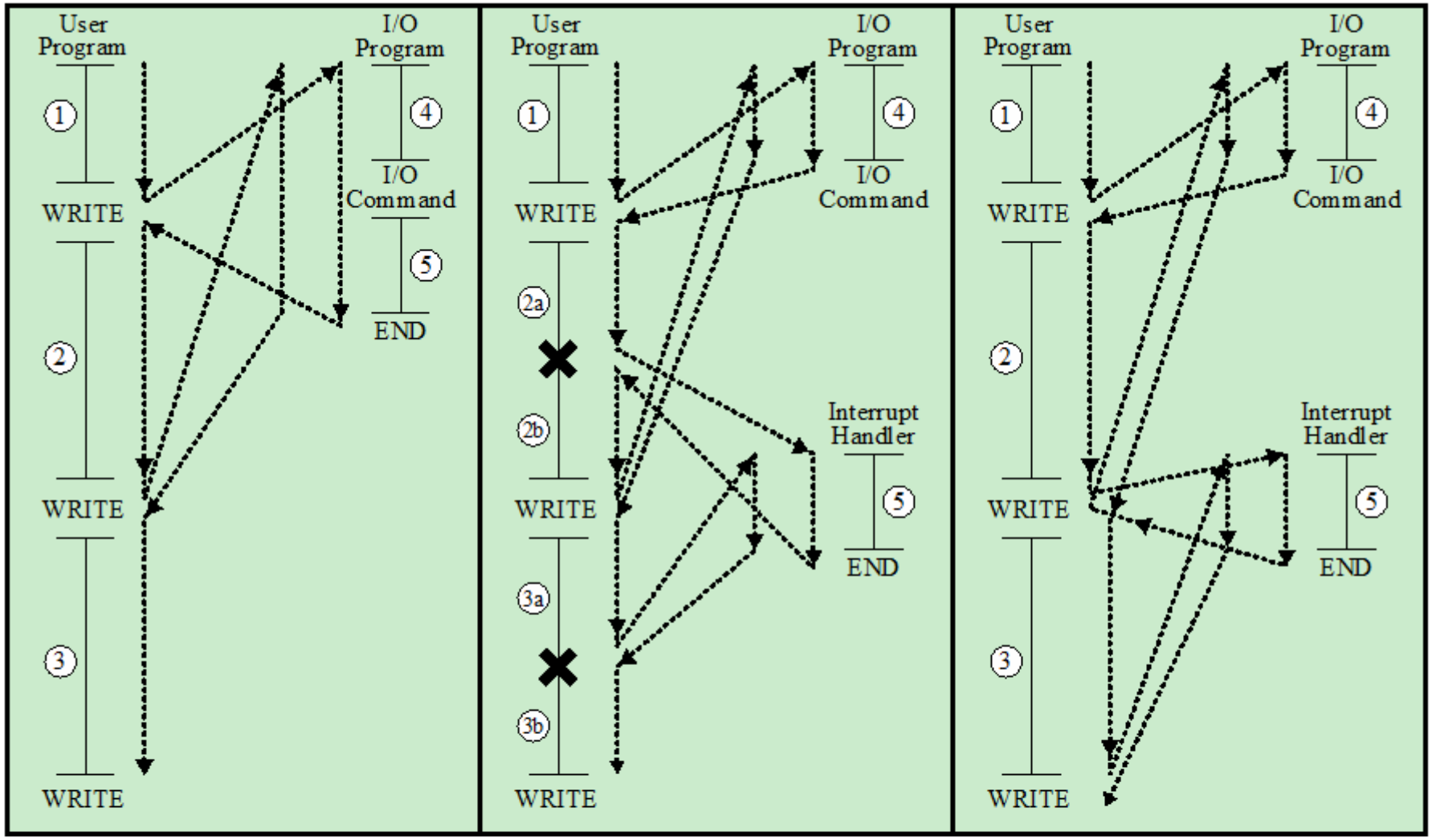
- All computers provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.
- Table 3.1 lists the most common classes of interrupts.
- Interrupts are provided to improve processing efficiency.
- From the point of view of the user program, an interrupt is an interruption of the normal sequence of execution.
- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.

Table 3.1: 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.

Interrupts

- The user program performs a series of WRITE calls interleaved with processing.
- Code segments 1, 2, and 3 refer to sequences of instructions that do not involve I/O in Figure 3.7a.
- The WRITE calls are to an I/O program which is a system utility and that will perform the actual I/O operation.
- The I/O program consists of three sections:
  - A sequence of instructions, labeled 4 in the Figure, to prepare for the actual I/O operation. This may include copying the data to be output into a special buffer and preparing the parameters for a device command.
  - The actual I/O command. Without the use of interrupts, once this command is issued, the program must wait for the I/O device to perform the requested function (or periodically poll the device).
  - A sequence of instructions, labeled 5 in the Figure, to complete the operation. This may include setting a flag indicating the success or failure of the operation.



(a) No interrupts (b) Interrupts; short I/O wait (c) Interrupts; long I/O wait

✖ = interrupt occurs during course of execution of user program

Figure 3.7 Program Flow of Control Without and With Interrupts

Interrupts and Instruction Cycle

- The I/O program that is invoked consists of the preparation code and the actual I/O command.
- After these few instructions have been executed, the control returns to the user program.
- An external device accepts data from computer memory and prints it.
- This I/O operation is conducted concurrently with the execution of instructions in the user program.
- When the external device becomes ready to be serviced—that is, when it is ready to accept more data from the processor—the I/O module for that external device sends an *interrupt request* signal to the processor.
- The processor responds by suspending the operation of the current program, branching off to a program to service that particular I/O device, known as an **interrupt handler,** and resuming the original execution after the device is serviced.

Interrupts and Instruction Cycle

- The WRITE command invokes the I/O program provided by the OS.
- 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.
- An interrupt is an interruption of the normal sequence of execution.
- When the interrupt processing is completed, execution resumes as shown in Figure 3.8.
- The user program does not have to contain any special code to accommodate interrupts; the processor and the operating system are responsible for suspending the user program and then resuming it at the same point.

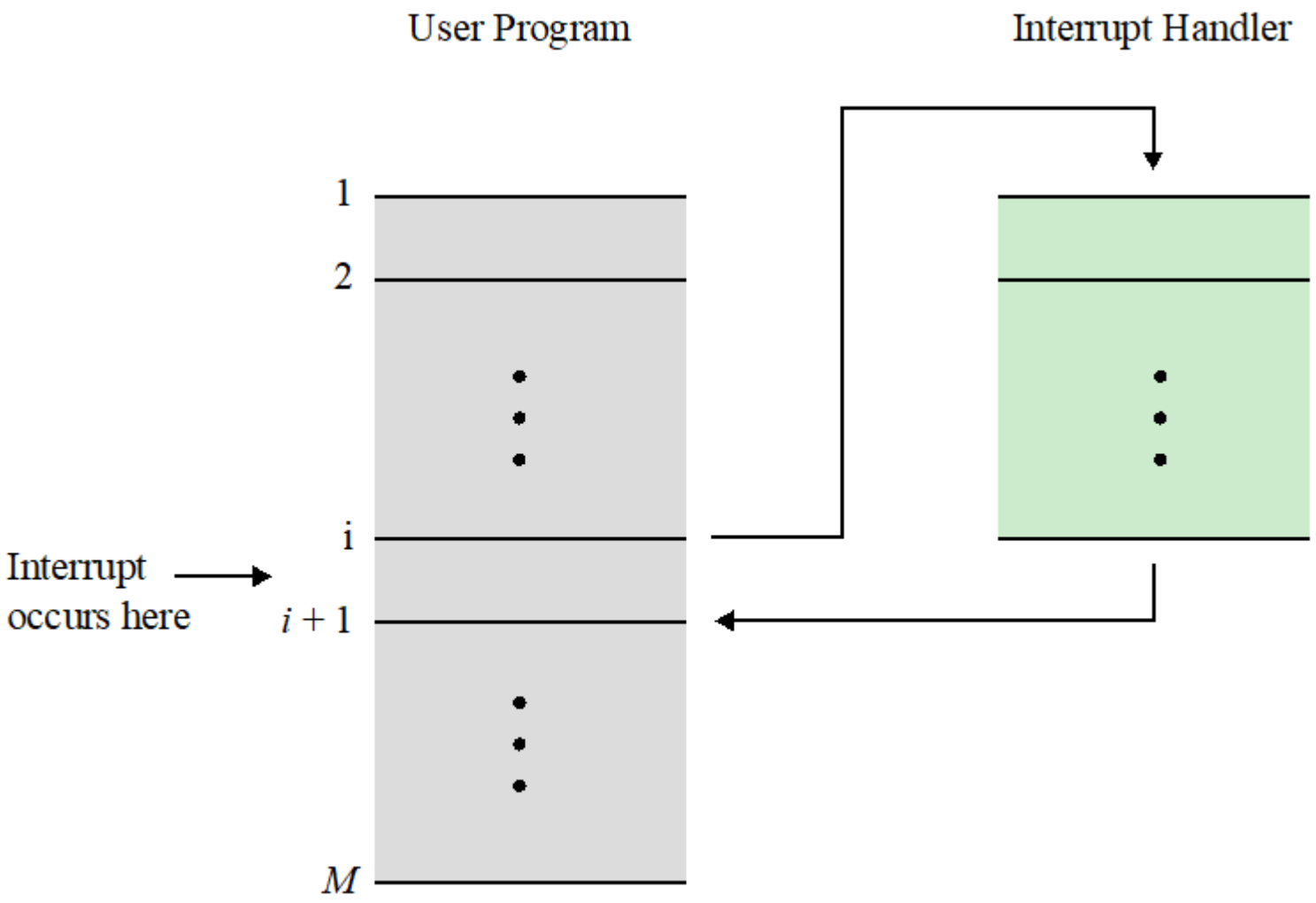
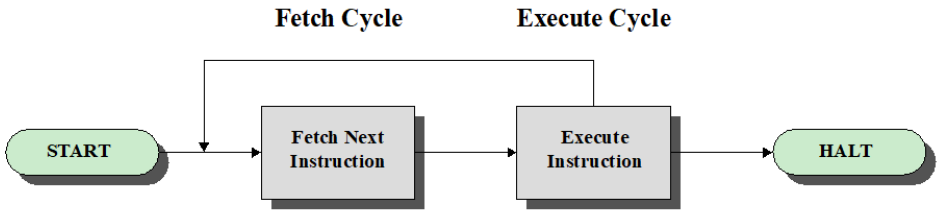


Figure 3.8 Transfer of Control via Interrupts

Interrupts and Instruction Cycle

- To accommodate interrupts, an *interrupt cycle* is added to the instruction cycle, as shown in Figure 3.9.
- In the *interrupt cycle*, the processor checks to see if any interrupts have occurred, indicated by the presence of an interrupt signal.
- If no interrupts are pending, the processor proceeds to the fetch cycle and fetches the next instruction of the current program.
- If an interrupt is pending, the processor does the following:
  - It suspends execution of the current program being executed and saves its context.
  - It sets the program counter to the starting address of an *interrupt handler routine*.
- The processor now proceeds to fetch cycle and fetches the first instruction in the interrupt handler program, which will service the interrupt.
- This program determines the nature of the interrupt and performs whatever actions are needed.
- There is some overhead involved in this process.
- Extra instructions must be executed (in the interrupt handler) to determine the nature of the interrupt and to decide on the appropriate action.



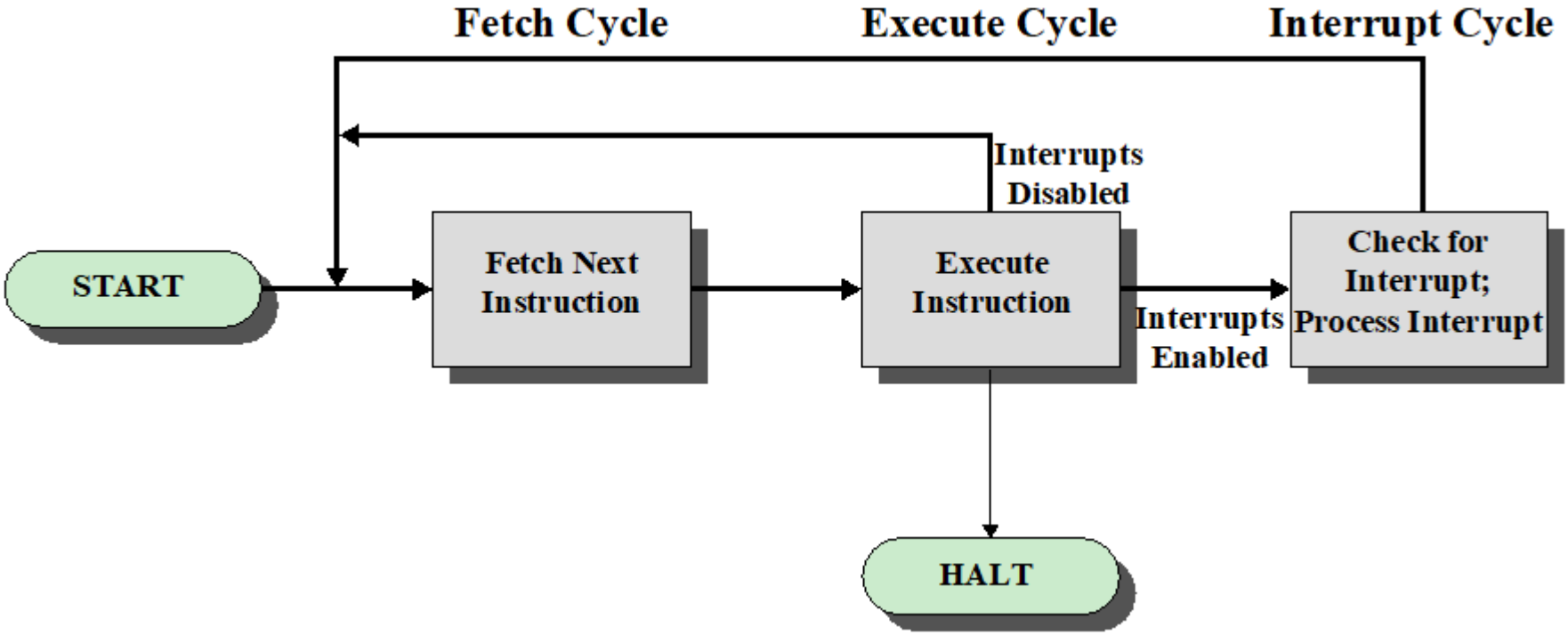
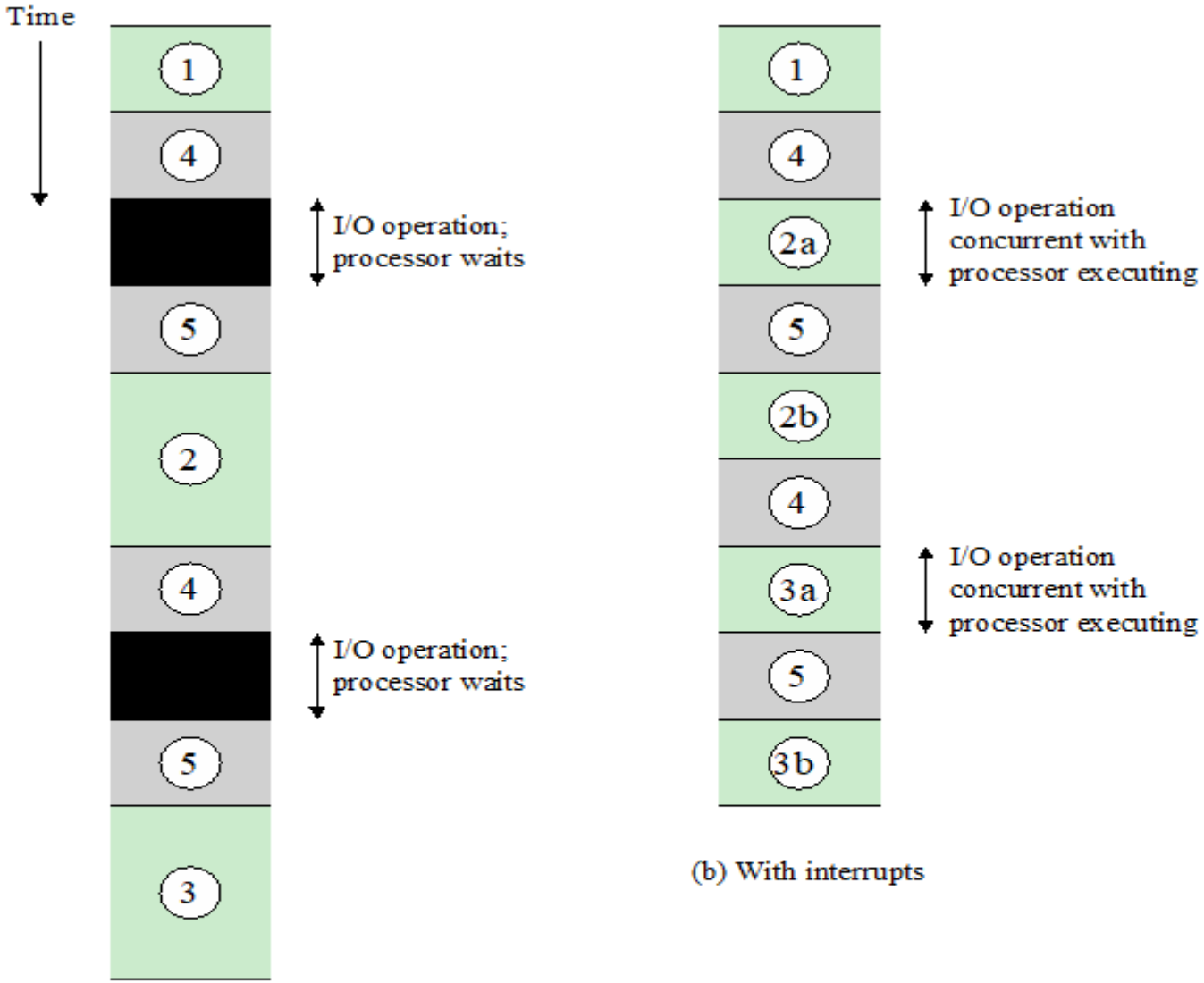
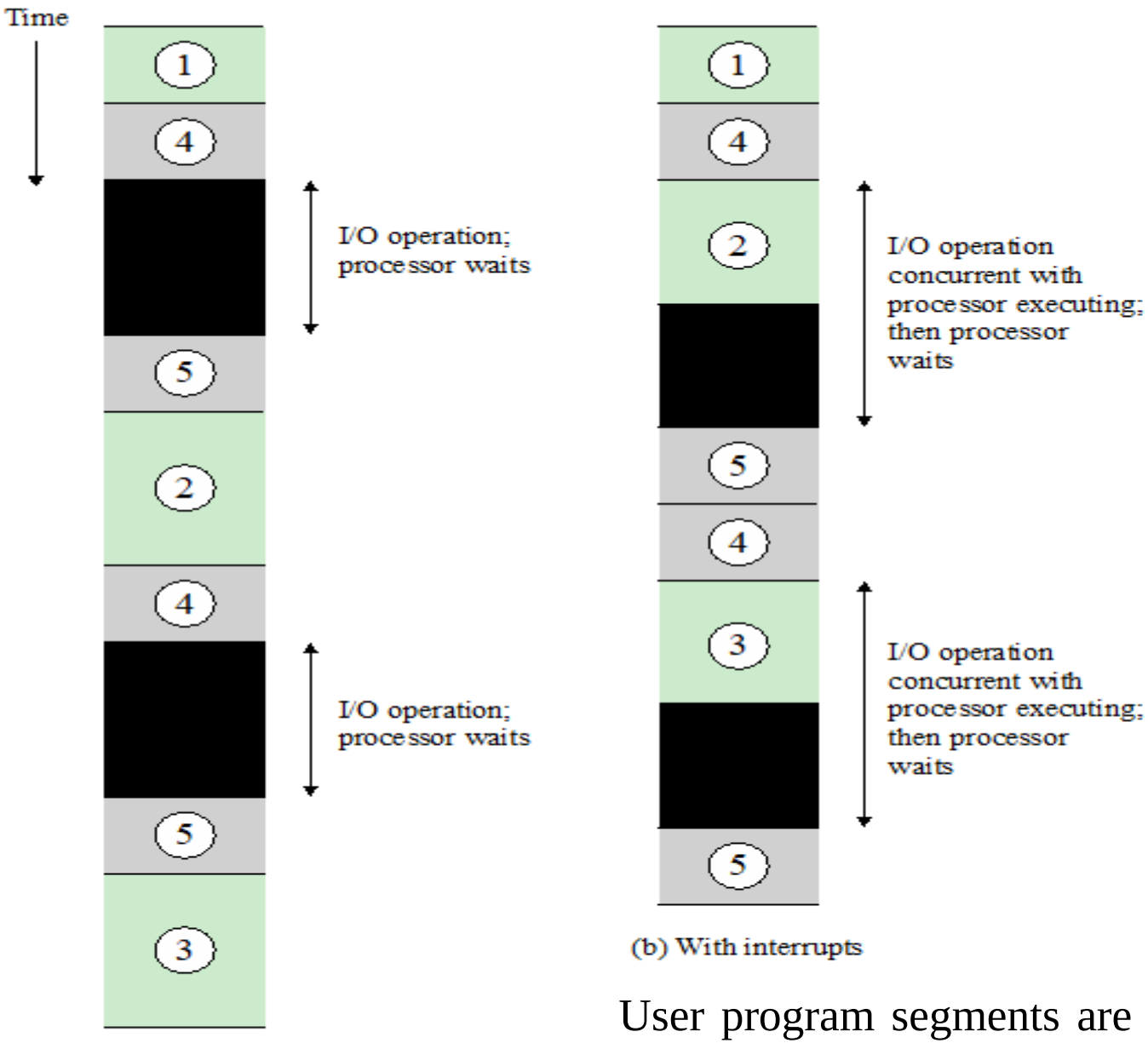


Figure 3.9 Instruction Cycle with Interrupts



User program segments are shaded green, I/O program segments are shaded gray

Figure 3.10 Program Timing: Short I/O Wait



User program segments are shaded green,
I/O program segments are shaded gray

Figure 3.11 Program Timing: Long I/O Wait

Figure 3.12 illustrates a revised instruction cycle state diagram that includes interrupt cycle processing.

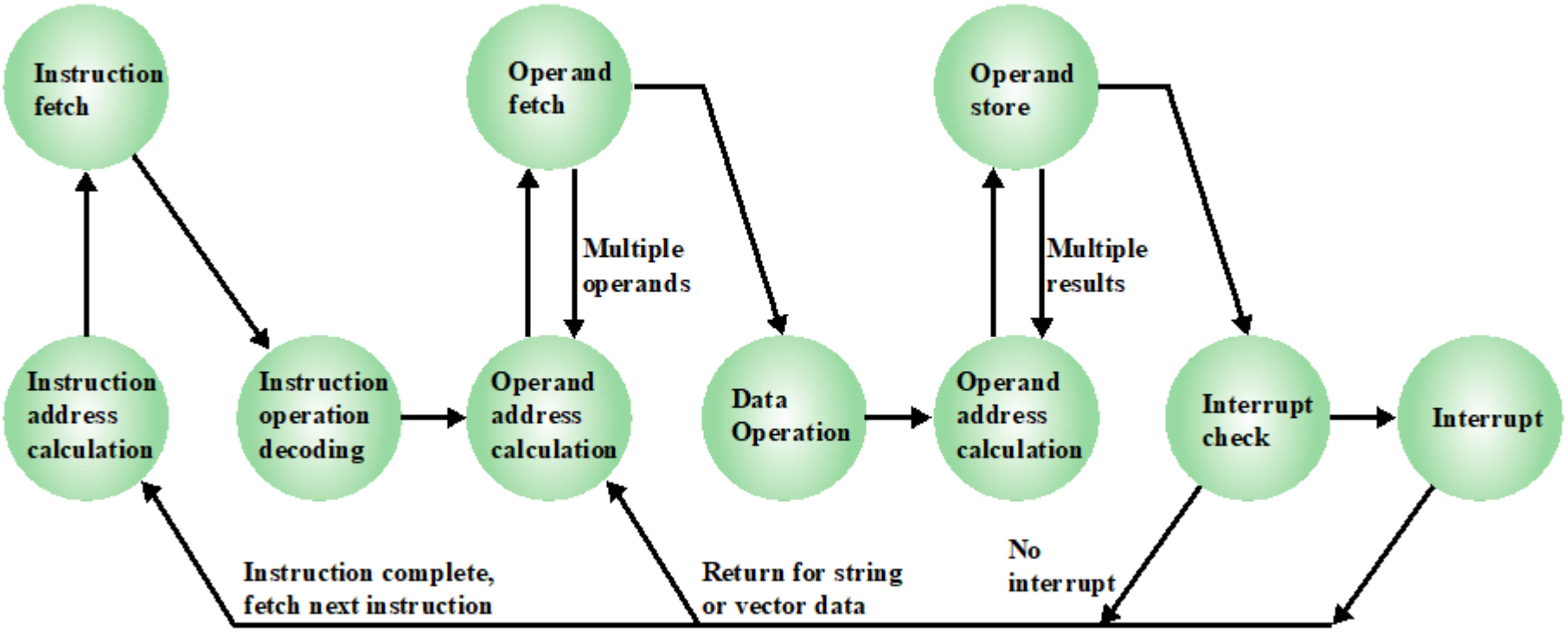


Figure 3.12 Instruction Cycle State Diagram, With Interrupts

Multiple Interrupts

- Two approaches can be taken to deal with multiple interrupts.
- The first is to disable interrupts while an interrupt is being processed.
- A **disabled interrupt** means that the processor can and will ignore that interrupt request signal.
- If an interrupt occurs during this time, it remains pending and will be checked by the processor after the processor has enabled interrupts.
- When a user program is executing and an interrupt occurs, interrupts are disabled immediately.
- After the interrupt handler routine completes, interrupts are enabled before resuming the user program, and the processor checks to see if additional interrupts have occurred.

- The approach in Figure 3.13a is simple, as interrupts are handled in strict sequential order.
- Drawback, it does not take into account relative priority or time-critical needs.
- The approach in Figure 3.13b defines priorities for interrupts and allow an interrupt of higher priority to cause a lower-priority interrupt handler to be itself interrupted.

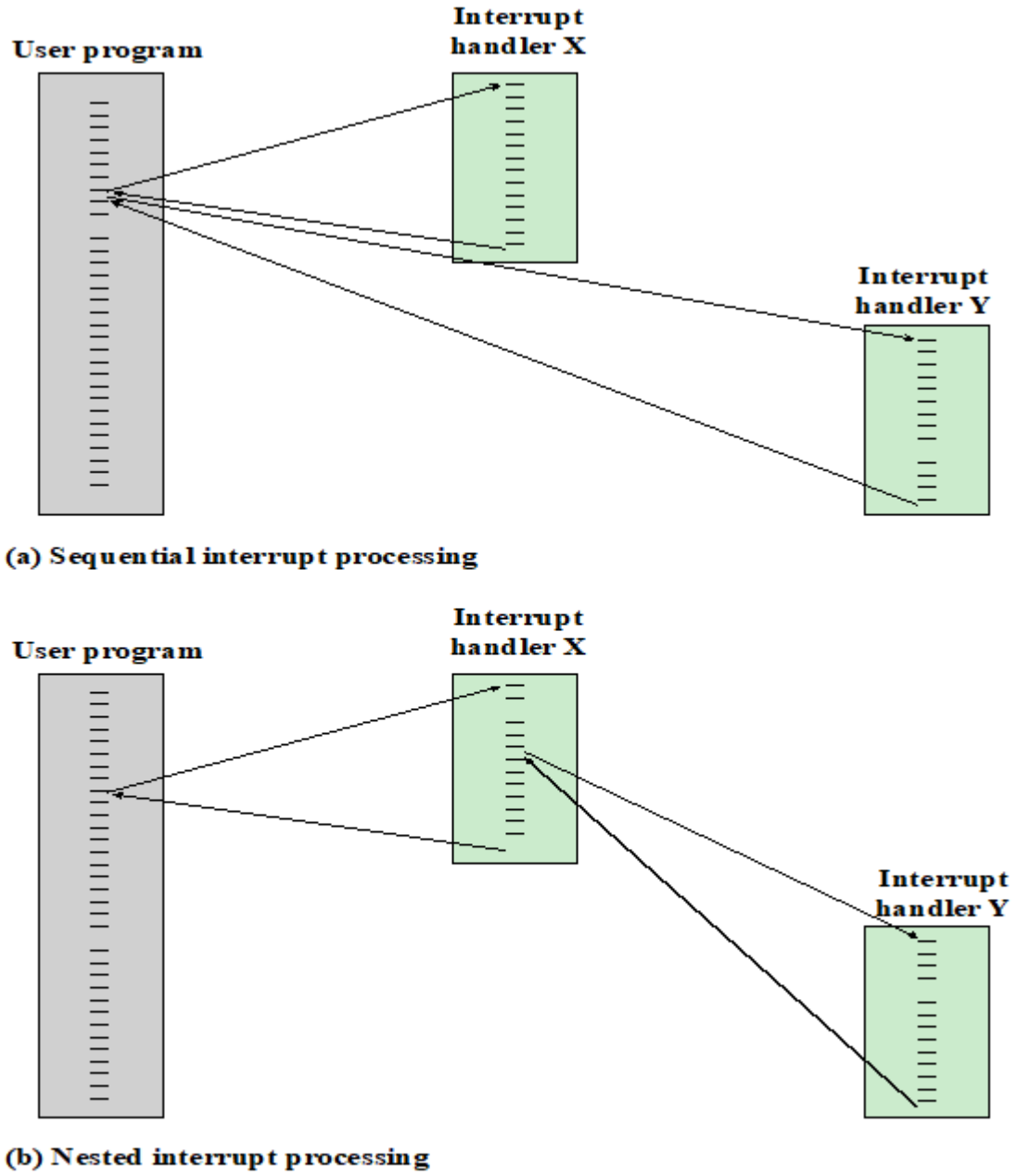


Figure 3.13 Transfer of Control with Multiple Interrupts

Multiple interrupts: Example

- 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.
- A user program begins at *t = 0.*
- *At t = 10,* a printer interrupt occurs; user information is placed on the system stack and execution continues at the printer **interrupt service routine (ISR).**
- When a routine is still executing, at *t = 15, a* communications interrupt occurs.
- The printer ISR is interrupted, its state is pushed onto the stack, and execution continues at the communications ISR.
- While this routine is executing, a disk interrupt occurs (*t = 20), because this* interrupt is of lower priority, it is held, and the communications ISR runs to completion.
- When the communications ISR is complete (*t = 25),* the *previous processor* state is restored, which is the execution of the printer ISR.
- Before an instruction in that routine can be executed, the processor honors the higher-priority disk interrupt and control transfers to the disk ISR.
- Only when that routine is complete (*t = 35)* is the *printer ISR resumed. When that routine completes (t = 40), the* control finally returns to the user program.

Figure 3.14 illustrates an time sequence of multiple interrupts example

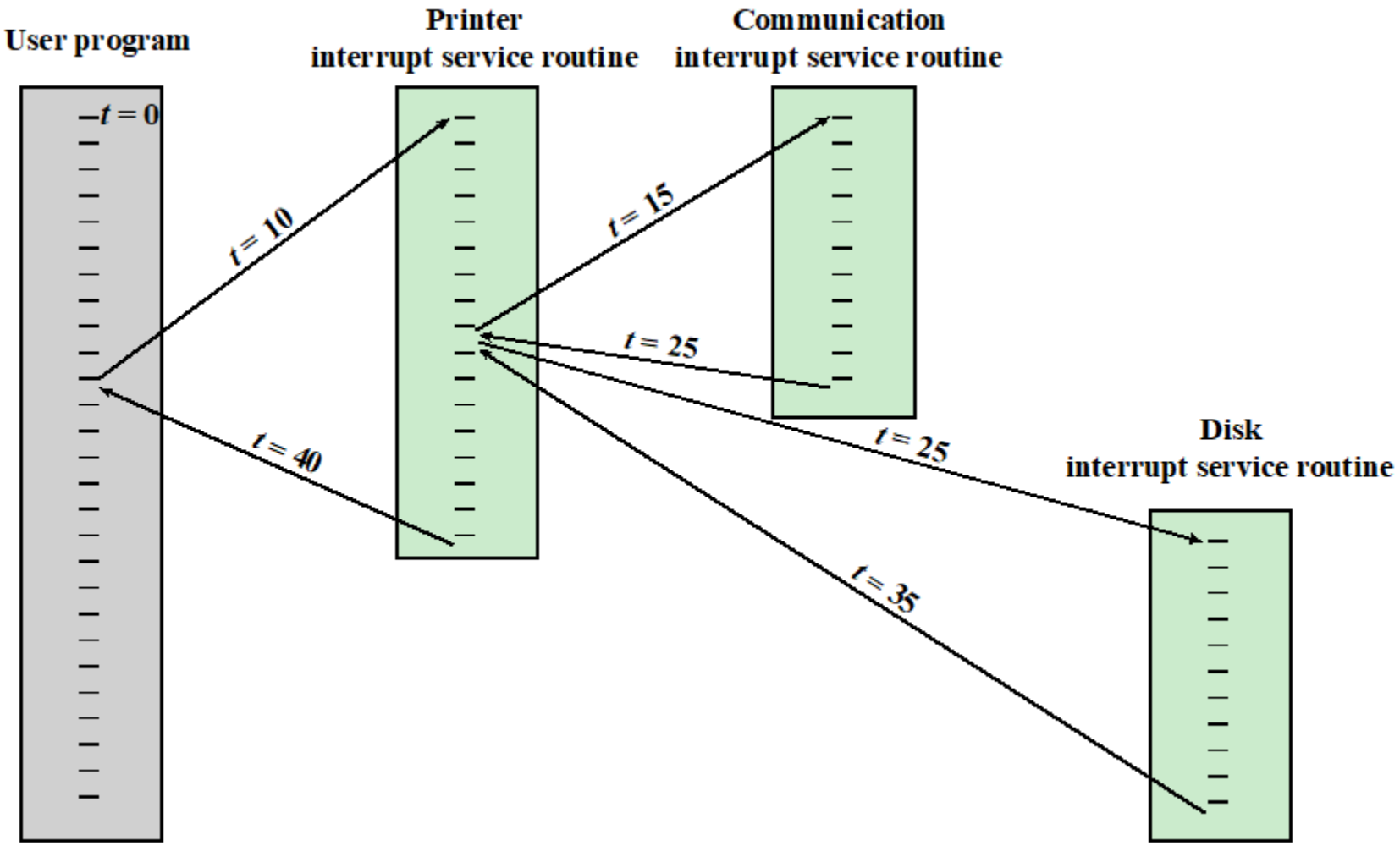


Figure 3.14 Example Time Sequence of Multiple Interrupts

I/O Function

- The I/O module can exchange data directly with the processor.
- The processor can read data from or write data to an I/O module:
  - The processor identifies a specific device that is controlled by a particular I/O module.
  - I/O instructions rather than memory referencing instructions.
- In some cases it is desirable to allow I/O exchanges to occur directly with memory:
  - The processor grants to an I/O module the authority to read from or write to memory so that the I/O memory transfer can occur without tying up the processor.
  - The I/O module issues read or write commands to memory relieving the processor of responsibility for the exchange, operation is known as *direct memory access* (DMA).

Top-Level View of Computer Function and Interconnection

- Organization and Architecture
- Structure and Function
- Computer Components
- Computer Function
- **Interconnection Structures**
- **Bus Interconnection**

Interconnection Structures

- The collection of paths connecting the various modules is called the *interconnect structure.*
- The design of this structure will depend on the exchanges that must be made among modules.
- Figure 3.15 suggests the types of exchanges that are needed by indicating the major forms of input and output for each module type:
  - **Memory**
  - **I/O module**
  - **Processor**

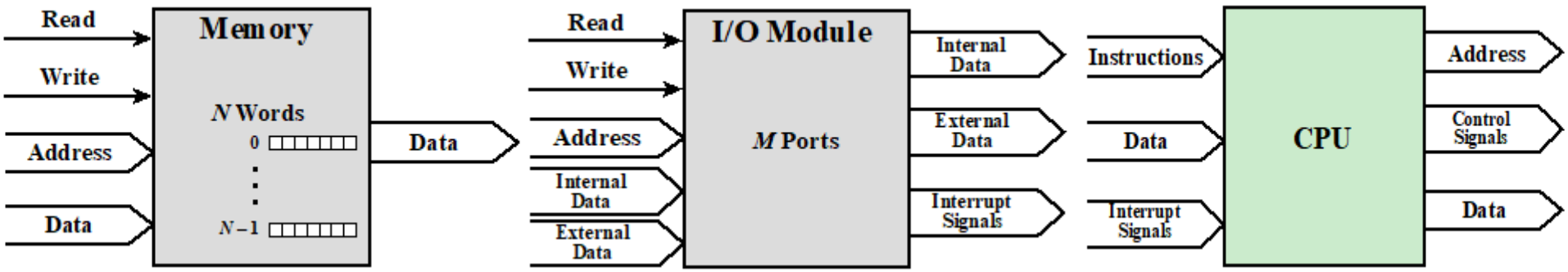


Figure 3.15 Computer Modules

Interconnection Structures: Types of Transfers

- The interconnection structures must support the following types of transfers:
  - **Memory to processor:** The processor reads an instruction or a unit of data from memory.
  - **Processor to memory:** The processor writes a unit of data to memory.
  - **I/O to processor:** The processor reads data from an I/O device via an I/O module.
  - **Processor to I/O:** The processor sends data to the I/O device.
  - **I/O to or from memory:** An I/O module is allowed to exchange data directly with memory, without going through the processor, using DMA.

- The most common interconnection structures are:
  1) the bus and various multiple-bus structures
  2) a point-to-point interconnection structure with packetized data transfer

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
Processor reads an instruction or a unit of data from memory	Processor writes a unit of data to memory	Processor reads data from an I/O device via an I/O module	Processor sends data to the I/O device	An I/O module is allowed to exchange data directly with memory without going through the processor using direct memory access

Bus Interconnection

- A bus is a communication pathway connecting two or more devices.
- A bus is a shared transmission medium.
- Multiple devices connect to the bus, and a signal transmitted by any one device is available for reception by all other devices attached to the bus.
- If two devices transmit during the same time period, their signals will overlap and become garbled.
- Only one device at a time can successfully transmit.
- A bus consists of multiple communication pathways or lines.
- A bus that connects major computer components (processor, memory, I/O) is called a *system bus*.
- The most common computer interconnection structures are based on the use of one or more system buses.

Bus Interconnection

- A system bus consists of about fifty to hundreds of separate lines.
- Each line is assigned a particular function.
- Bus lines can be classified into three functional groups (Figure 3.16): *data, address, and control lines*.
- In addition, there may be power distribution lines that supply power to the attached modules.

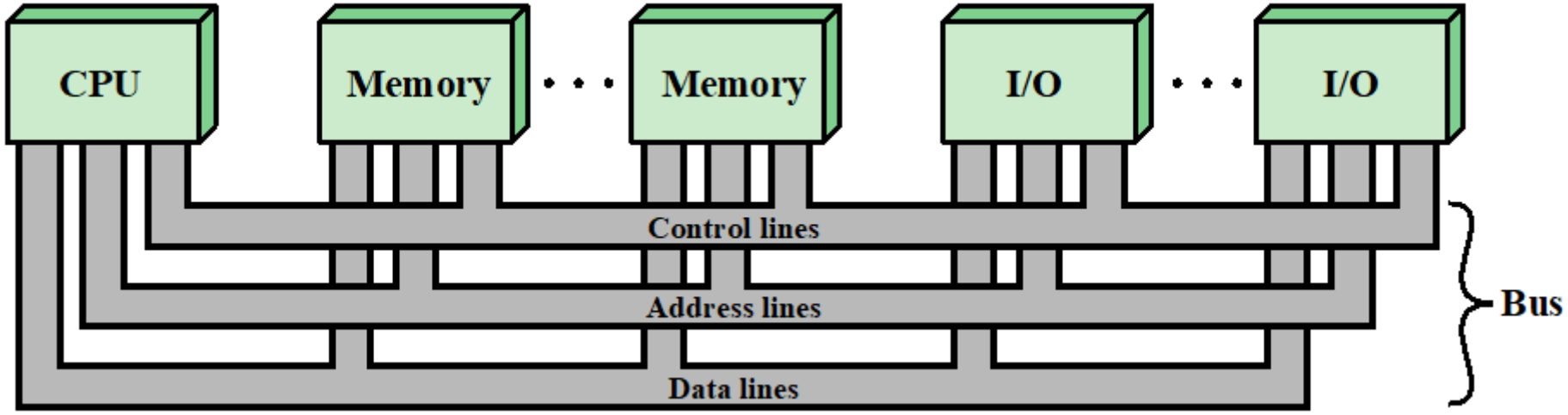


Figure 3.16 Bus Interconnection Scheme

Bus Interconnection: Data Bus

- The **data lines** provide a path for moving data among system modules, these lines, collectively, are called the **data bus**.
- **Data bus** may consist of 32, 64, 128, or more separate lines.
- The number of lines is referred to as the *width* of the data bus.
- The number of lines determines how many bits can be transferred at a time.
- The width of the data bus is a key factor in determining overall system performance.

Bus Interconnection: Address Lines

- The ***address lines*** are used to designate the source or destination of the data on the data bus.
- The width of the ***address bus*** determines the maximum possible memory capacity of the system.
- The address lines are generally also used to address ***I/O ports***.
- The higher-order bits are used to select a particular module on the bus, and the lower-order bits select a memory location or I/O port within the module.
- For example, on an 8-bit address bus, address 01111111 and below might reference locations in a memory module (module 0) with 128 words of memory, and address 10000000 and above refer to devices attached to an I/O module (module 1).

A communication pathway connecting two or more devices

- Key characteristic is that it is a shared transmission medium

Signals transmitted by any one device are available for reception by all other devices attached to the bus

- If two devices transmit during the same time period their signals will overlap and become garbled

Typically consists of multiple communication lines

- Each line is capable of transmitting signals representing binary 1 and binary 0

Computer systems contain a number of different buses that provide pathways between components at various levels of the computer system hierarchy

System bus

- A bus that connects major computer components (processor, memory, I/O)

The most common computer interconnection structures are based on the use of one or more system buses

Bus Interconnection: Control Lines

- The ***control lines*** are used to control the access to and the use of the data and address lines.
- Control signals transmit both command and timing information among system modules.
- Timing signals indicate the validity of data and address information.
- Command signals specify operations to be performed.
- Typical control lines include:
  - **Memory write:** causes data on the bus to be written into the addressed location
  - **Memory read:** causes data from the addressed location to be placed on the bus
  - **I/O write:** causes data on the bus to be output to the addressed I/O port
  - **I/O read:** causes data from the addressed I/O port to be placed on the bus
  - **Transfer ACK**: indicates that data have been accepted from or placed on the bus
  - **Bus request:** indicates that a module needs to gain control of the bus
  - **Bus grant:** indicates that a requesting module has been granted control of the bus
  - **Interrupt request:** indicates that an interrupt is pending
  - **Interrupt ACK:** acknowledges that the pending interrupt has been recognized
  - **Clock:** is used to synchronize operations
  - **Reset:** initializes all modules

Address Bus

- Used to designate the source or destination of the data on the data bus
  - If the processor wishes to read a word of data from memory it puts the address of the desired word on the address lines
- Width determines the maximum possible memory capacity of the system
- Also used to address I/O ports
  - The higher order bits are used to select a particular module on the bus and the lower order bits select a memory location or I/O port within the module

Control Bus

- Used to control the access and the use of the data and address lines
- Because the data and address lines are shared by all components there must be a means of controlling their use
- Control signals transmit both command and timing information among system modules
- Timing signals indicate the validity of data and address information
- Command signals specify operations to be performed

Bus Interconnection

The operation of the bus is as follows:

- If one module wishes to send data to another, it must:
  - obtain the use of the bus
  - transfer data via the bus
- If one module wishes to request data from another module, it must:
  - obtain the use of the bus
  - transfer a request to the other module over the appropriate control and address lines.
  - it must then wait for that second module to send the data.

Point-to-Point Interconnect

Principal reason for change was the electrical constraints encountered with increasing the frequency of wide synchronous buses

At higher and higher data rates it becomes increasingly difficult to perform the synchronization and arbitration functions in a timely fashion

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

Has lower latency, higher data rate, and better scalability

Cache Memory

- **Computer Memory System Overview**
  - Cache Memory Principles
  - Elements of Cache Design
  - Pentium 4 Cache Organization

Characteristics of Memory Systems

- The complex subject of computer memory is made manageable if we classify memory systems according to their key characteristics.

Table 4.1: Key Characteristics of Computer Memory Systems

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

Characteristics of Memory Systems

- Location
  - Refers to whether memory is internal or external to the computer
  - Internal memory is often equated with main memory
  - The processor requires its local memory, in the form of registers
  - The cache is another form of internal memory
  - External memory consists of peripheral storage devices that are accessible to the processor via I/O controllers
- Capacity
  - Memory is typically expressed in terms of words or bytes
- 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

Characteristics of Memory Systems

- The three related concepts for internal memory are:
- **Word:** The size of a word is 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.
  - Many systems allow addressing at the byte level.
  - The relationship between the length in bits *A of an address and the number N of addressable* units is $2^A = N$.
- **Unit of transfer:**
  - For main memory, this is the number of bits read out of or written into memory at a time.
  - The unit of transfer need not equal a word or an addressable unit.
  - For external memory, data are often transferred in much larger units than a word, and these are referred to as blocks.

Method of Accessing Units of Data

Sequential access

- Memory is organized into units of data called records
- Access must be made in a specific linear sequence
- Access time is variable

Direct access

- Involves a shared read-write mechanism
- Individual blocks or records have a unique address based on physical location
- Access time is variable

Random access

- Each addressable location in memory has a unique, physically wired-in addressing mechanism
- The time to access a given location is independent of the sequence of prior accesses and is constant
- Any location can be selected at random and directly addressed and accessed
- Main memory and some cache systems are random access

Associative

- A word is retrieved based on a portion of its contents rather than its address
- Each location has its own addressing mechanism and retrieval time is constant independent of location or prior access patterns
- Cache memories may employ associative access

Capacity and Performance:

The two most important characteristics of memory are **capacity and performance**.

Three performance parameters are used:

Access time (latency)

- For random-access memory it is the time it takes to perform a read or write operation
- For non-random-access memory it is the time it takes to position the read-write mechanism at the desired location

Memory cycle time

- Access time plus any additional time required before second access can commence
- Additional time may be required for transients to die out on signal lines or to regenerate data if they are read destructively
- Concerned with the system bus, not the processor

Transfer rate

- The rate at which data can be transferred into or out of a memory unit
- For random-access memory it is equal to 1/(cycle time)

Capacity and Performance:

- Transfer rate:

  For non-random-access memory, the following relationship holds:

$$T_n = T_A + \frac{n}{R}$$

where

T_n = Average time to read or write n bits

T_A = Average access time

n = Number of bits

R = Transfer rate, in bits per second (bps)

Memory

- The most common forms of memory are:
  - Semiconductor memory
  - Magnetic surface memory
  - Optical
  - Magneto-optical
- Several **physical characteristics** of data storage are important:
  - Volatile memory
    - Information decays naturally or is lost when electrical power is switched off
  - Nonvolatile memory
    - Information remains without deterioration until deliberately changed
    - No electrical power is needed to retain information
  - Magnetic-surface memories
    - Are nonvolatile
  - Semiconductor memory
    - May be either volatile or nonvolatile
  - Nonerasable memory
    - Cannot be altered, except by destroying the storage unit
    - Semiconductor memory of this type is ROM
- For random-access memory, the organization is a key design issue
  - Organization refers to the physical arrangement of bits to form words

Memory Hierarchy

- Design constraints on a computer's memory can be summed up by three questions:
  - How much? how fast? how expensive?
- There is a trade-off between capacity, access time, and cost:
  - Faster access time, greater cost per bit
  - Greater capacity, smaller cost per bit
  - Greater capacity, slower access time
- The way out of the memory dilemma is not to rely on a single memory component or technology, but to employ a ***memory hierarchy***.

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

The key to the success of this organization is (d): decreasing the frequency of access.

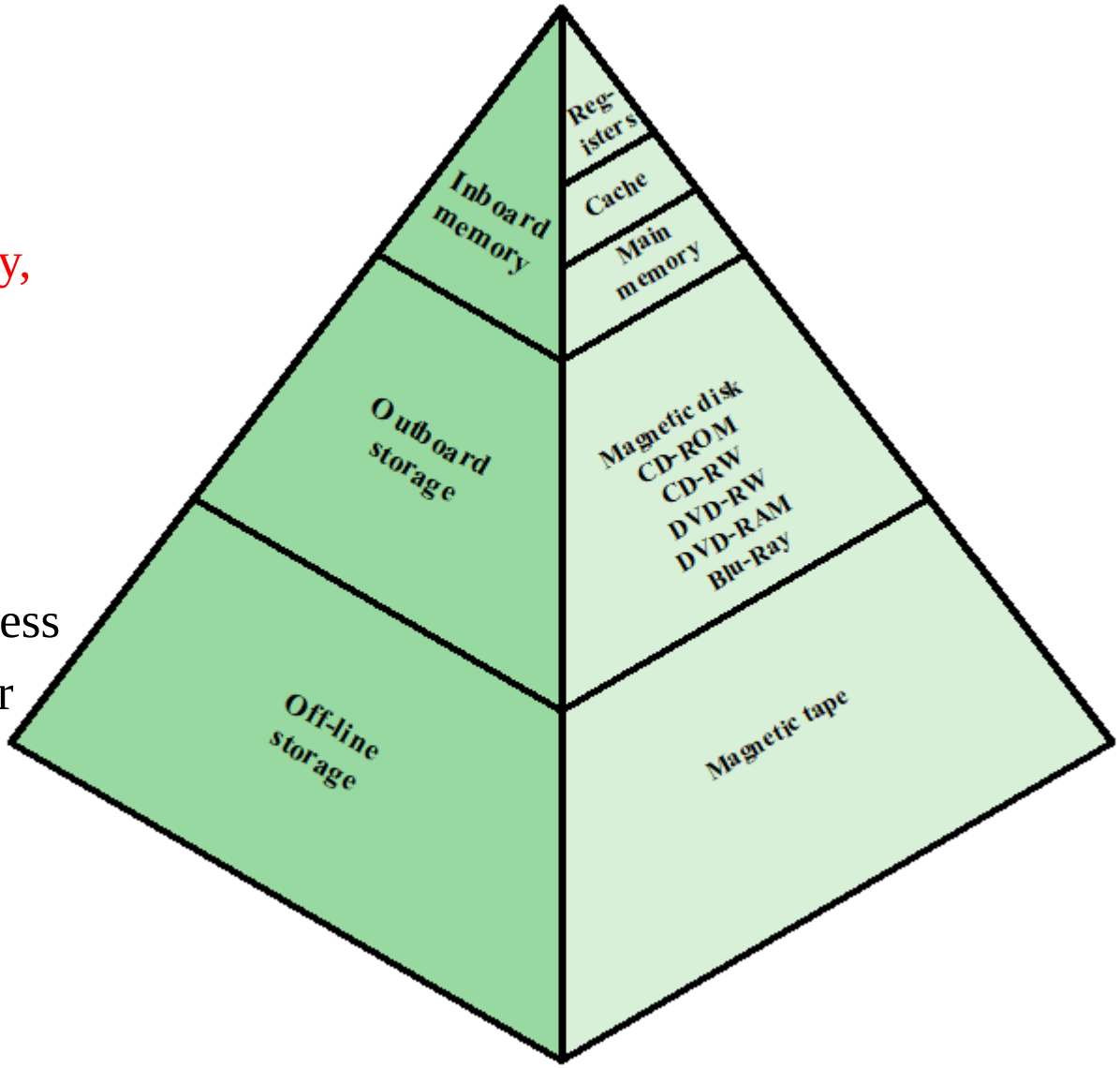


Figure 4.1 The Memory Hierarchy

Memory Hierarchy

- Smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories.
- The use of two levels of memory to reduce average access time works in principle, but only if conditions (a) through (d) apply.
- The basis for validity of condition (d) is a principle known as **locality of reference.**
- During the course of the execution of a program, memory references by the processor, for both instructions and data, tend to cluster.
- Programs typically contain several iterative loops and subroutines.
- Once a loop or subroutine is entered, there are repeated references to a small set of instructions.
- Operations on tables and arrays involve access to a clustered set of data words.
- Over a long period, the clusters in use change, but over a short period, the processor is working with fixed clusters of memory references.

Memory Hierarchy: Example

EXAMPLE 4.1 Suppose that the processor has access to two levels of memory. Level 1 contains 1000 words and has an access time of 0.01 μs; level 2 contains 100,000 words and has an access time of 0.1 μs. 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. For simplicity, we ignore the time required for the processor to determine whether the word is in level 1 or level 2. Figure 4.2 shows the general shape of the curve that covers this situation. The figure shows the average access time to a two-level memory as a function of the hit ratio H, where H is defined as the fraction of all memory accesses that are found in the faster memory (e.g., the cache), T_1 is the access time to level 1, and T_2 is the access time to level 2.[1] As can be seen, for high percentages of level 1 access, the average total access time is much closer to that of level 1 than that of level 2.

In our example, suppose 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 μs than to 0.1 μs, as desired.

Memory Hierarchy: Example

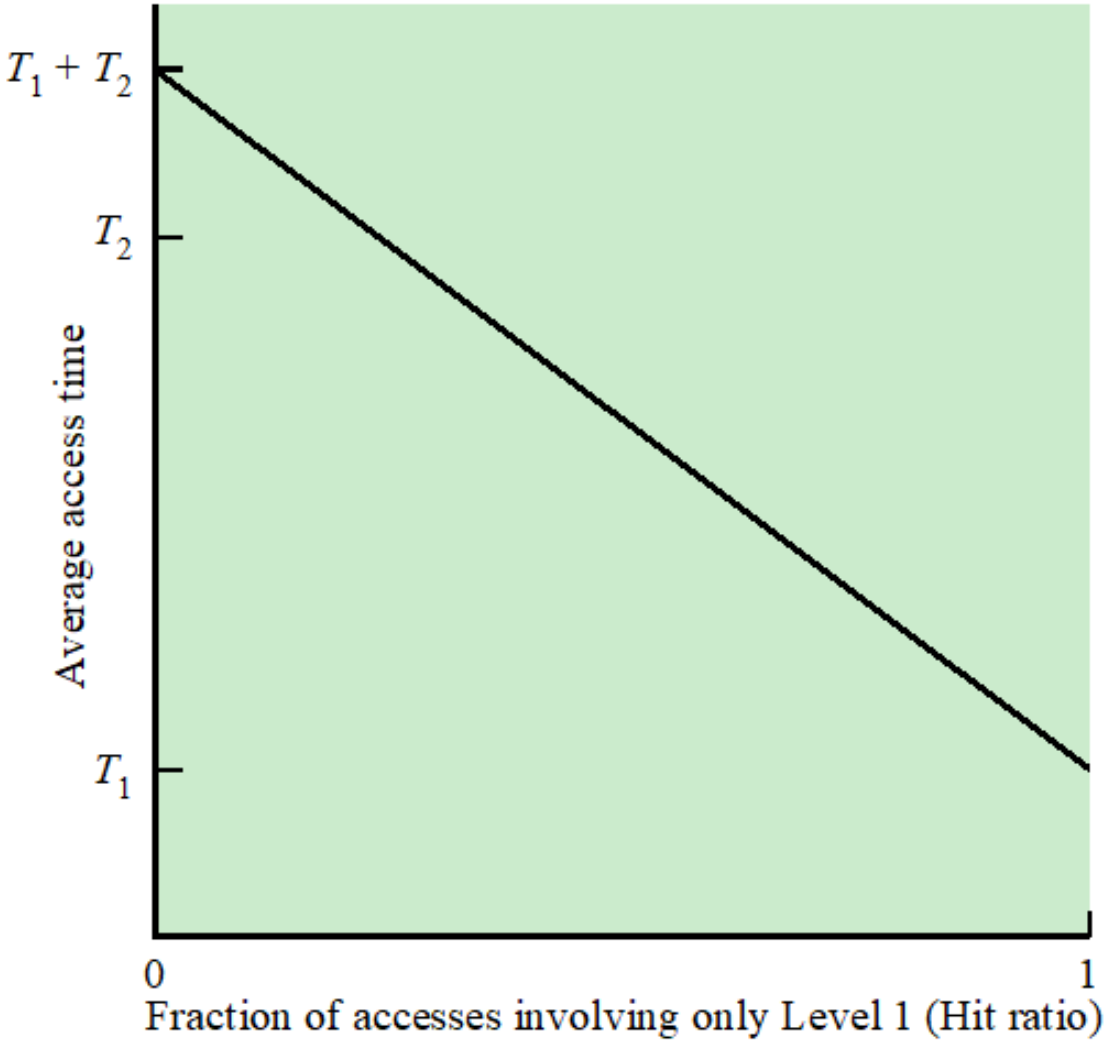


Figure 4.2 Performance of a Simple Two-Level Memory

Memory Hierarchy

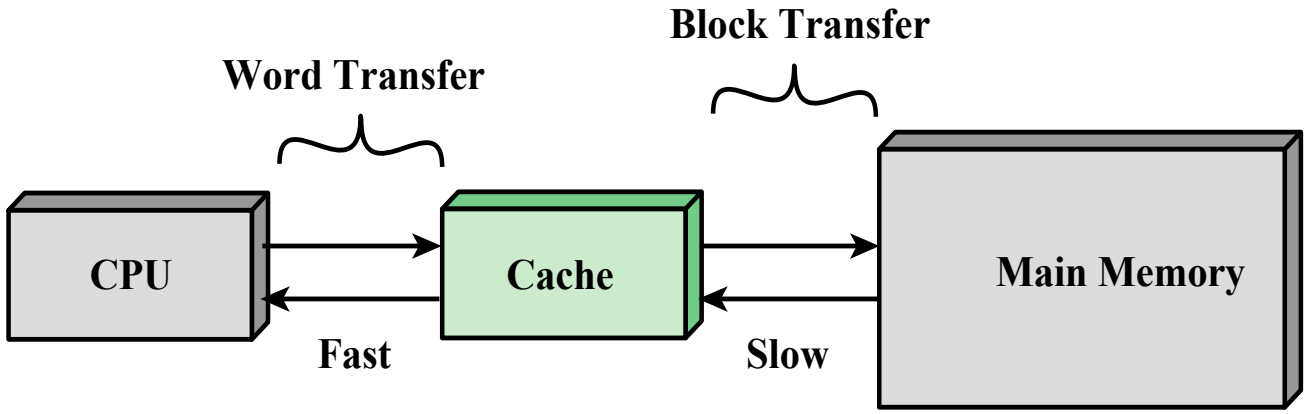
- The fastest, smallest, and most expensive type of memory consists of the registers internal to the processor.
- The semiconductor memory comes in a variety of types which differ in speed and cost.
- Data are stored permanently on external mass storage devices.
- External, nonvolatile memory is also referred to as **secondary** memory or **auxiliary** memory.
- Disk cache:
  - A portion of main memory can be used as a buffer to hold data temporarily that is to be read out to disk
  - A few large transfers of data can be used instead of many small transfers of data
  - Data can be retrieved rapidly from the software cache rather than slowly from the disk

Cache Memory

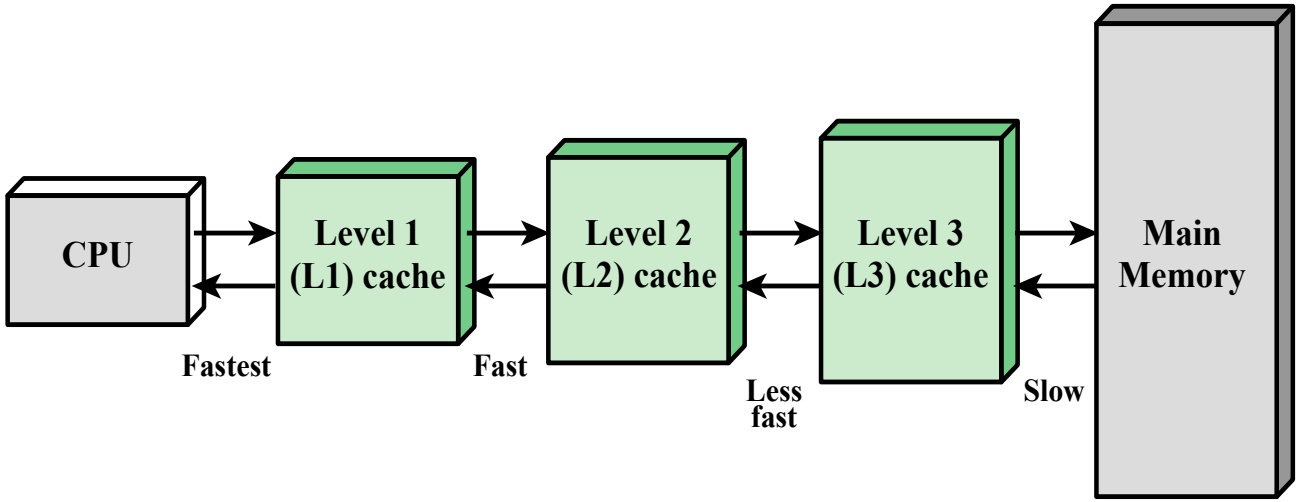
- Computer Memory System Overview
- **Cache Memory Principles**
- Elements of Cache Design
- Pentium 4 Cache Organization

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.
- There is a relatively large and slow main memory together with a smaller, faster cache memory.
- 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.
- Because of the locality of reference, when a block of data is fetched into the cache to satisfy a single memory reference, there will likely be future references to that same memory location or other words in the block.
- Figure 4.3b 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.



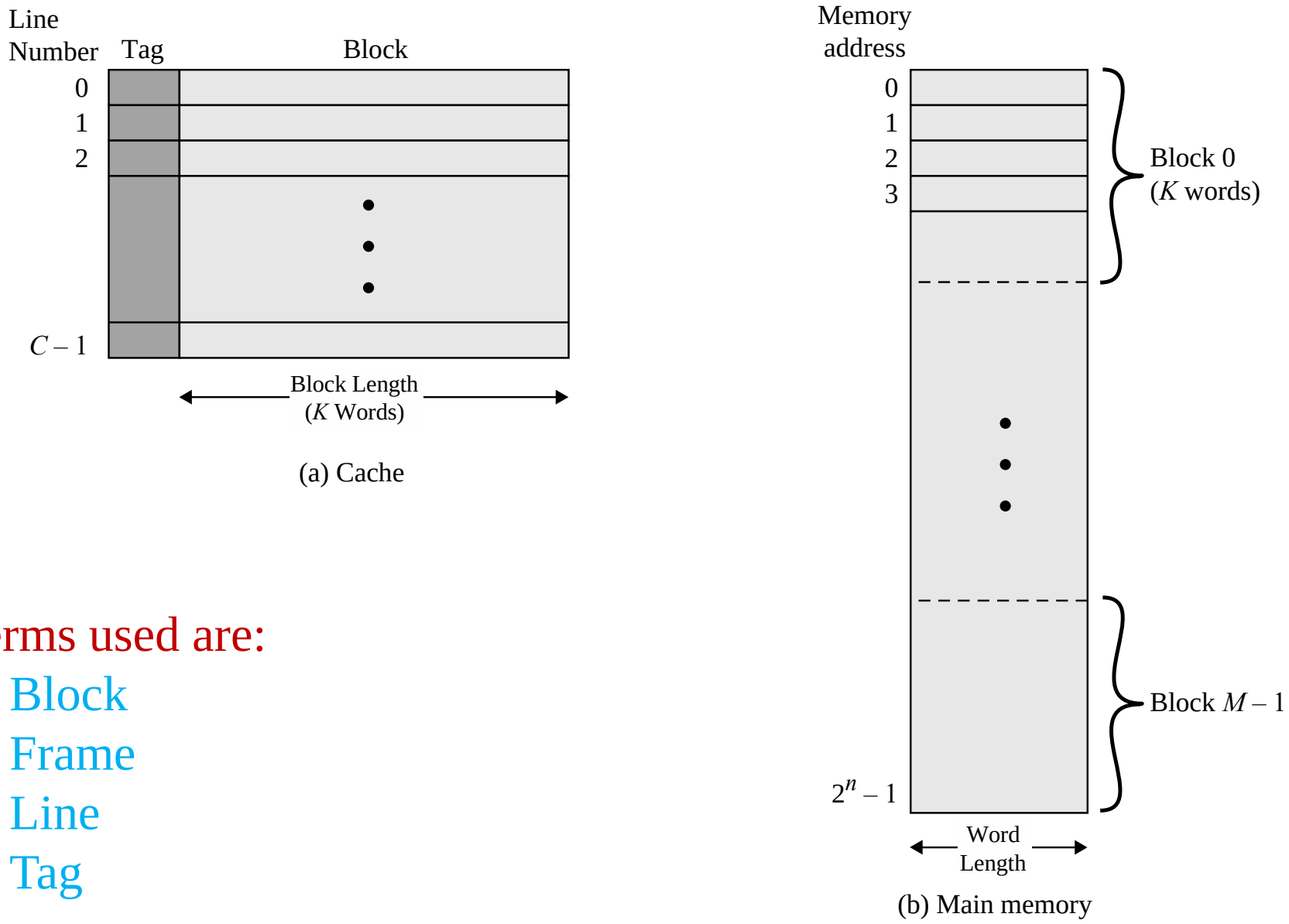
(a) Single cache



(b) Three-level cache organization

Figure 4.3 Cache and Main Memory

Figure 4.4 depicts the structure of a cache/main-memory system.



Terms used are:

- Block
- Frame
- Line
- Tag

Figure 4.4 Cache/Main-Memory Structure

Cache Read Operation

- Figure 4.5 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, it is delivered to the processor.
- If a cache miss occurs?
- Otherwise, the block containing that word is loaded into the cache, and the word is delivered to the processor.
- Figure 4.5 shows these last two operations occurring in parallel and reflects the organization shown in Figure 4.6, which is typical of contemporary cache organizations.

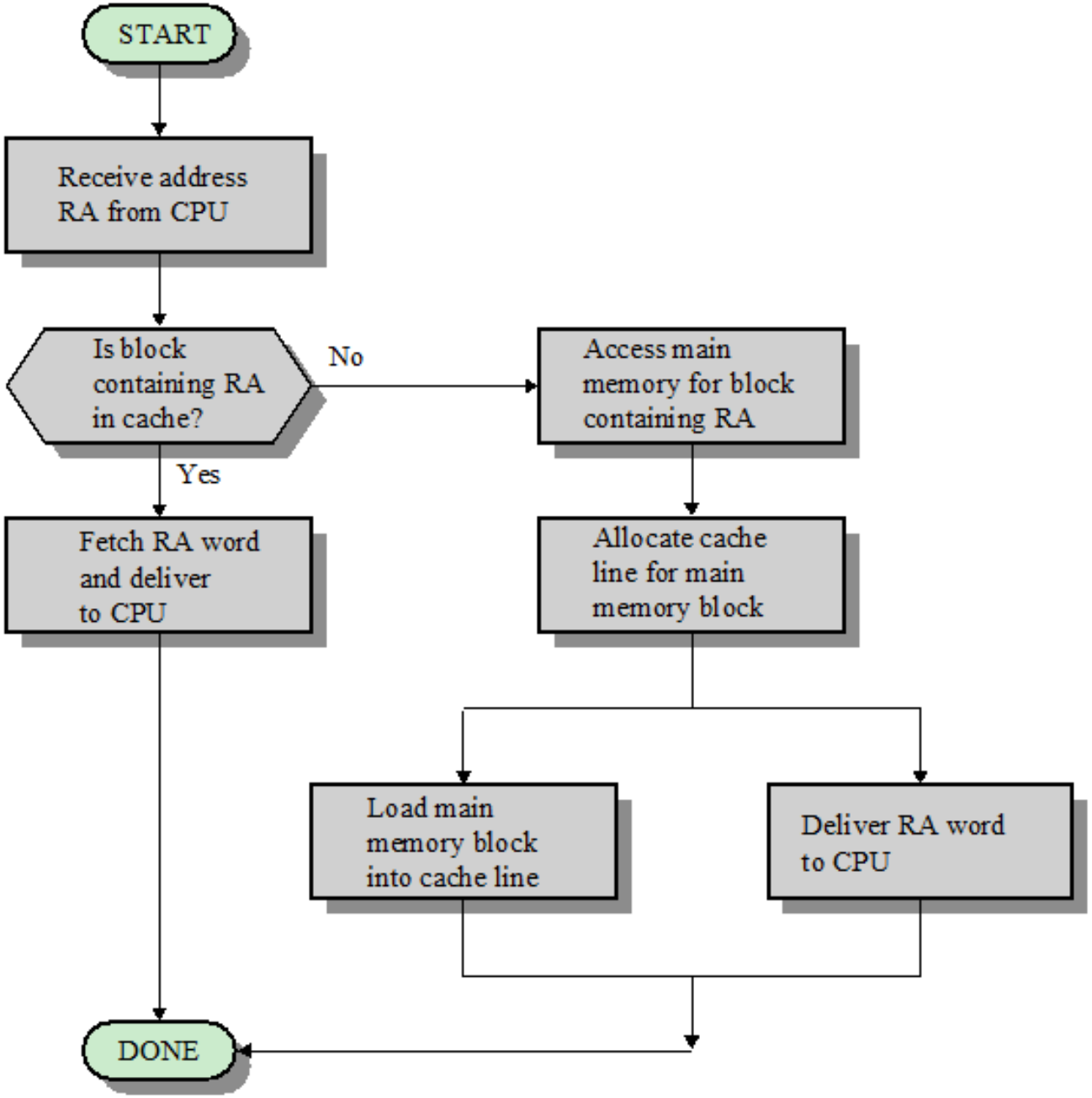


Figure 4.5 Cache Read Operation

Cache Read Operation

- 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.
- The cache is physically interposed between the processor and the main memory for all data, address, and control lines.
- For a cache miss, the desired word is first read into the cache and then transferred from the cache to the processor.

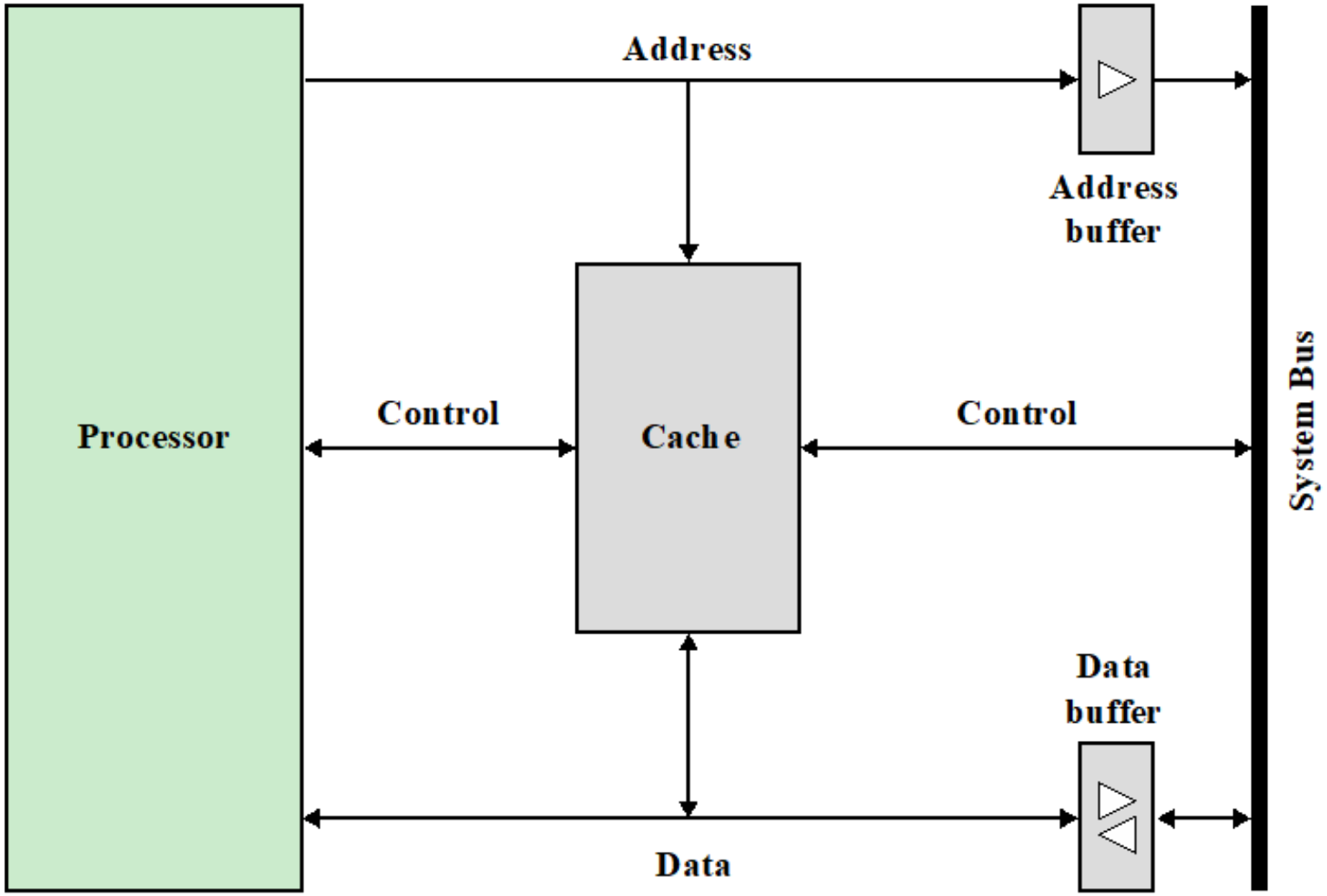


Figure 4.6 Typical Cache Organization

Cache Memory

- Computer Memory System Overview
- Cache Memory Principles
- **Elements of Cache Design**
- Pentium 4 Cache Organization

Elements of Cache Design

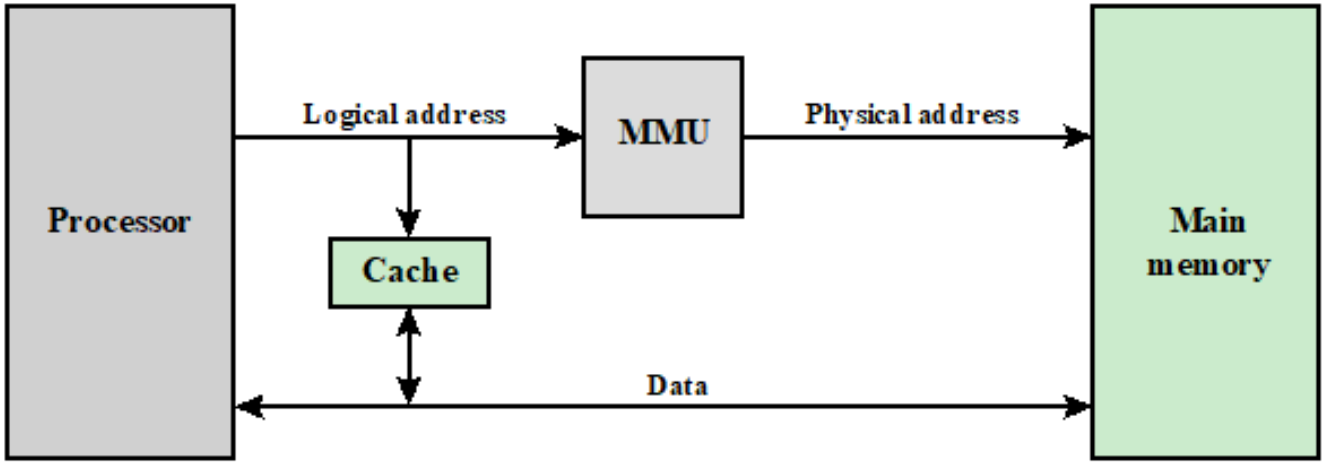
- The caches are used in high-performance computing (HPC).
- HPC deals with supercomputers and their software, especially for scientific applications that involve large amounts of data, vector and matrix computation, and the use of parallel algorithms.
- The cache design for HPC is quite different than for other hardware platforms and applications.
- Researchers have since shown that a cache hierarchy can be useful in improving performance if the application software is tuned to exploit the cache.
- Although there are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures.
- Table 4.2 lists key elements.

Table 4.2. Elements of Cache Design

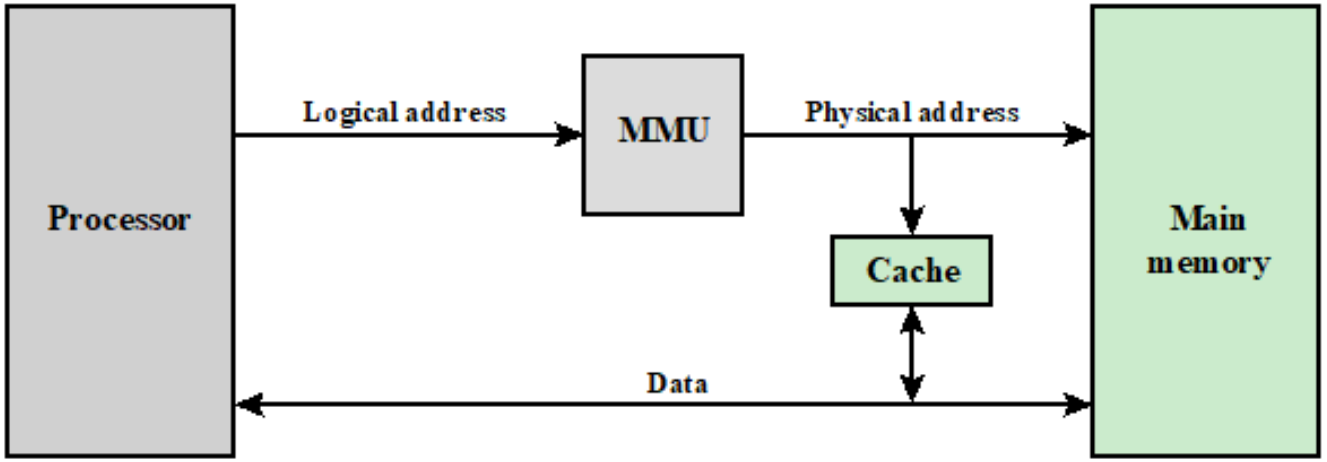
Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of caches**
Direct	Single or two level
Associative	Unified or split
Set Associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently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

Cache Addresses: Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory available.
- When virtual memory is used, the address fields of machine instructions contain virtual 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 4.7).
- A logical cache, also known as a virtual cache, stores data using virtual addresses.
- A 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 4.7 Logical and Physical Caches

Cache Addresses: Virtual Memory

- A physical cache stores data using main memory physical addresses.
- The advantage of the logical cache is that cache access speed is faster than for a physical cache because the cache can respond before the MMU performs an address translation.
- The disadvantage is that most virtual memory systems supply each application with the same virtual memory address space.
- Each application sees a virtual memory that starts at address 0.
- Thus, the same virtual address in two different applications refers to two different physical addresses.
- The cache memory must be flushed with each application context switch or extra bits must be added to each line of the cache to identify which virtual address space this address refers to.

Cache Size

- The size of the cache is small enough so that the overall average cost per bit is close to that of the 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 factors for minimizing cache size.
- The larger the cache, the larger the number of gates involved in addressing the cache.
- The large caches tend to be slower than small ones, even when built with the same integrated circuit technology and put in the same place on-chip and circuit board.
- The available chip and board area also limits cache size.
- As the performance of the cache is very sensitive to the nature of the workload, it is impossible to arrive at a single “optimum” cache size.
- Table 4.3 lists the cache sizes of some current and past 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 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 MTAb	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 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	1MB	—
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
Intel Core i7 EE 990	Workstaton/ server	2011	6 ´ 32 kB/32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ Server	2011	24 ´ 64 kB/ 128 kB	24 ´ 1.5 MB	24 MB L3 192 MB L4

Table 4.3

Cache Sizes of Some Processors

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

Mapping Function

- There are fewer cache lines than main memory blocks, so an algorithm is needed for mapping main memory blocks into cache lines.
- Three techniques can be used:

Direct

- The simplest technique
- Maps each block of main memory into only one possible cache line

Associative

- Permits each main memory block to be loaded into any line of the cache
- The cache control logic interprets a memory address simply as a Tag and a Word field
- To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's Tag for a match

Set Associative

- A compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages

1. Direct Mapping

- The mapping is expressed as

$$i = j \text{ modulo } m$$

where i = *cache line number*

j = *main memory block number*

m = *number of lines in the cache*

- Figure 4.8a 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 the cache, block B_{m+1} *maps into line* L_1, *and so on.*
- The mapping function is implemented using the main memory address.
- Figure 4.9 illustrates the general mechanism.

Direct Mapping

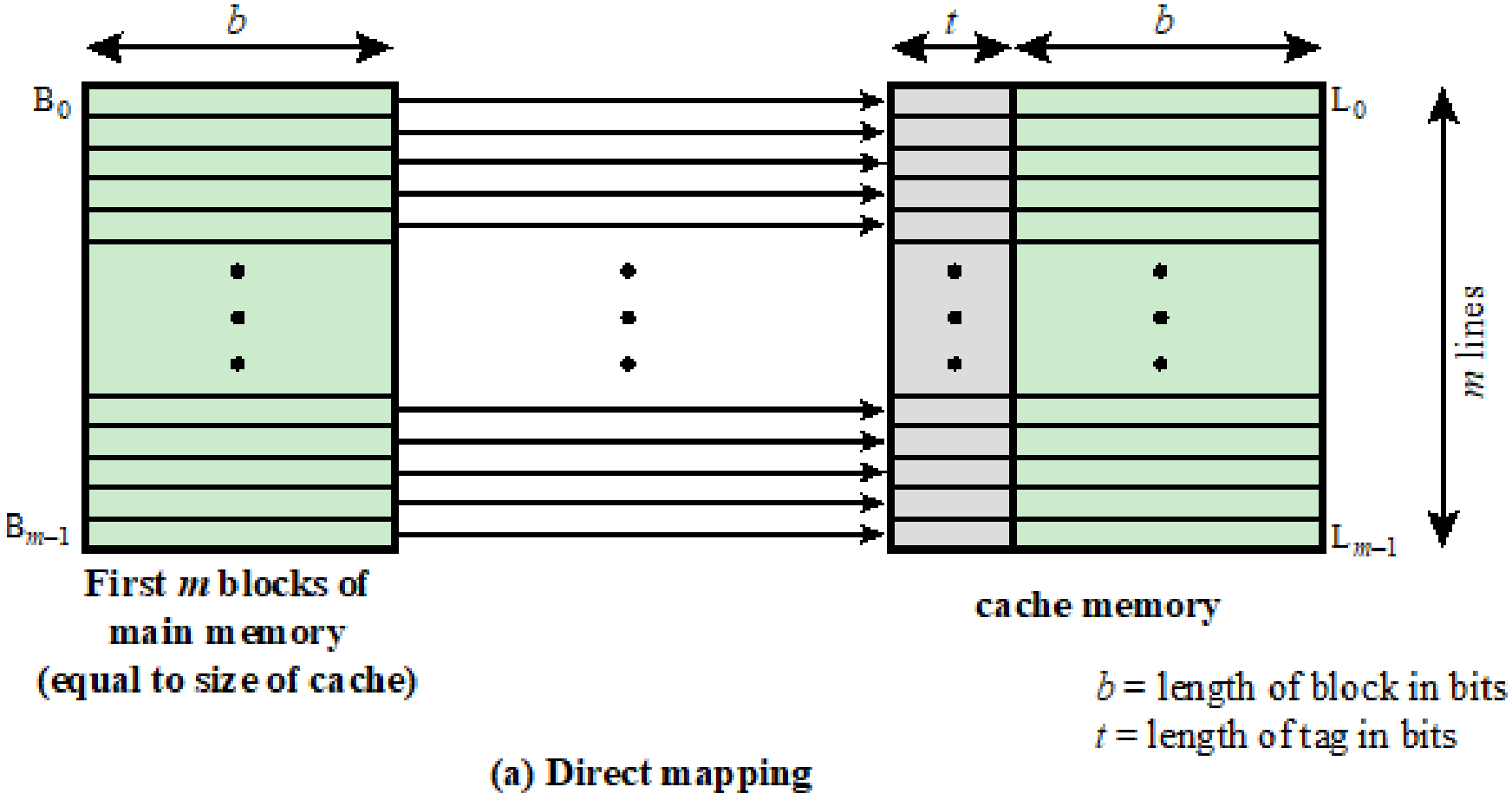


Figure 4.8 Mapping From Main Memory to Cache: Direct and Associative

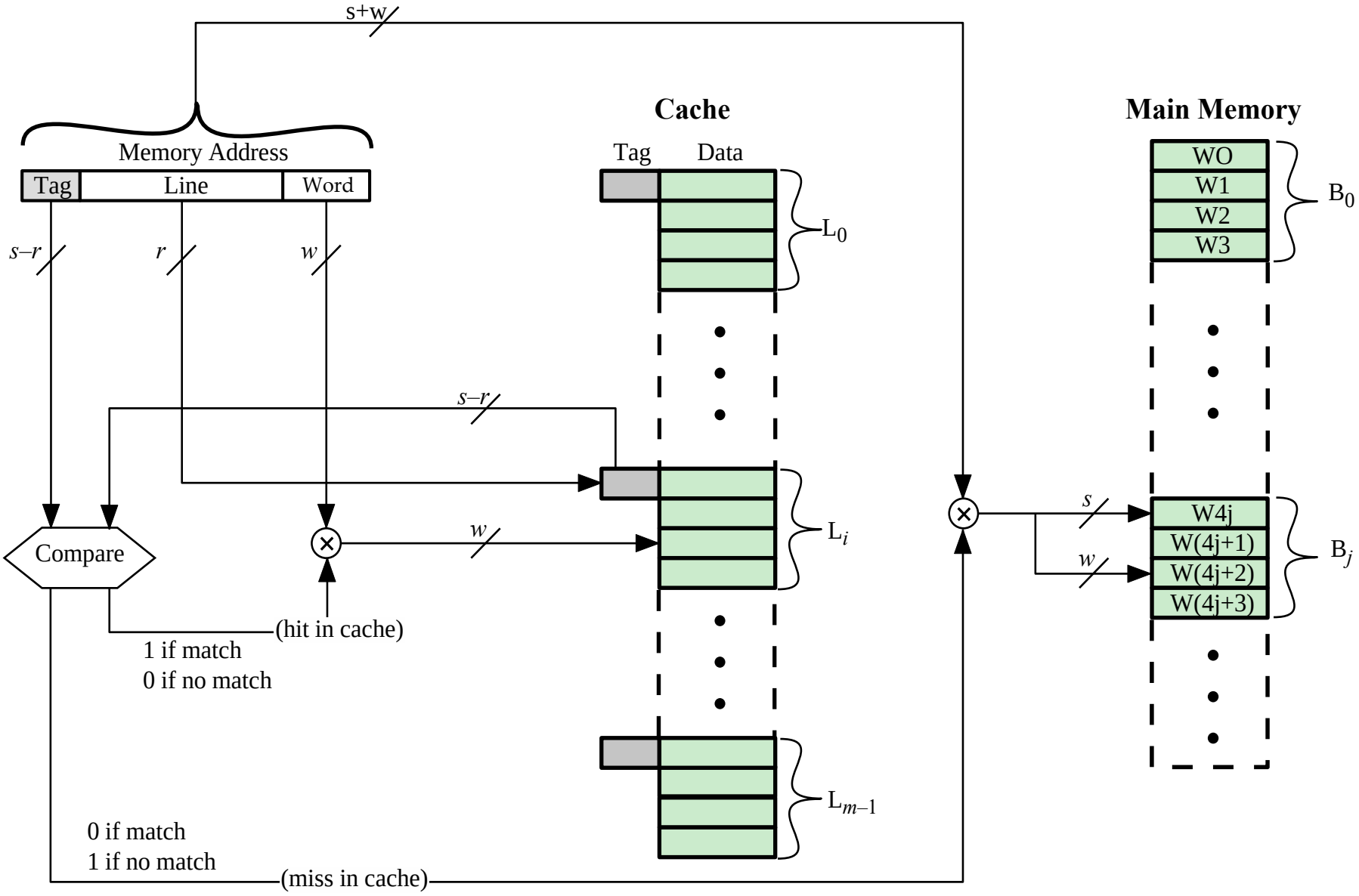


Figure 4.9 Direct-Mapping Cache Organization

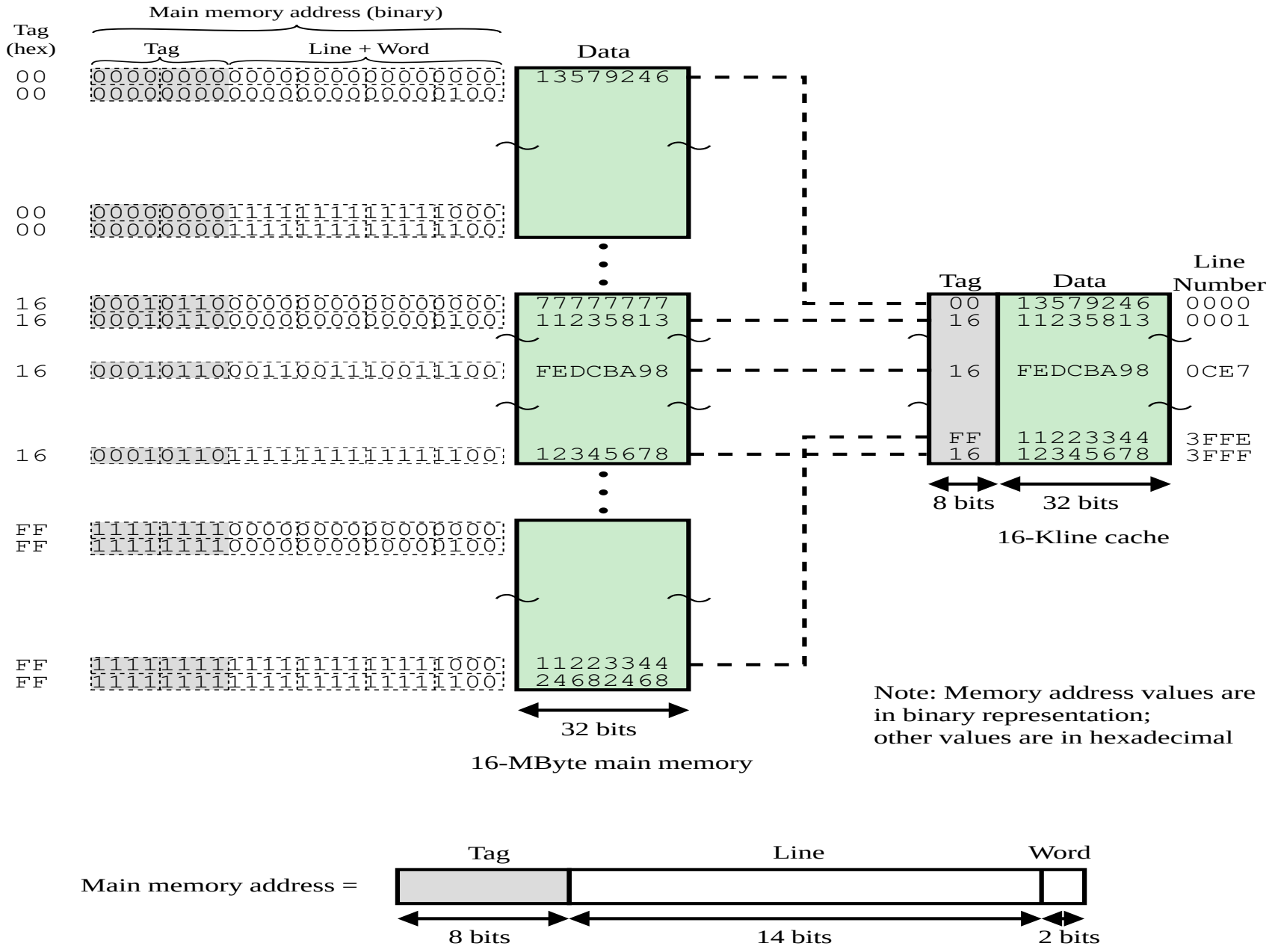


Figure 4.10 Direct Mapping Example

Direct Mapping Summary

- 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 = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of cache = m = 2^{r+w} words or bytes
- Size of tag = (s – r) bits

Direct Mapping Summary

- The direct mapping technique is simple and inexpensive to implement.
- The disadvantage is that there is a fixed cache location for any given block.
- If a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low (a phenomenon known as *thrashing).*

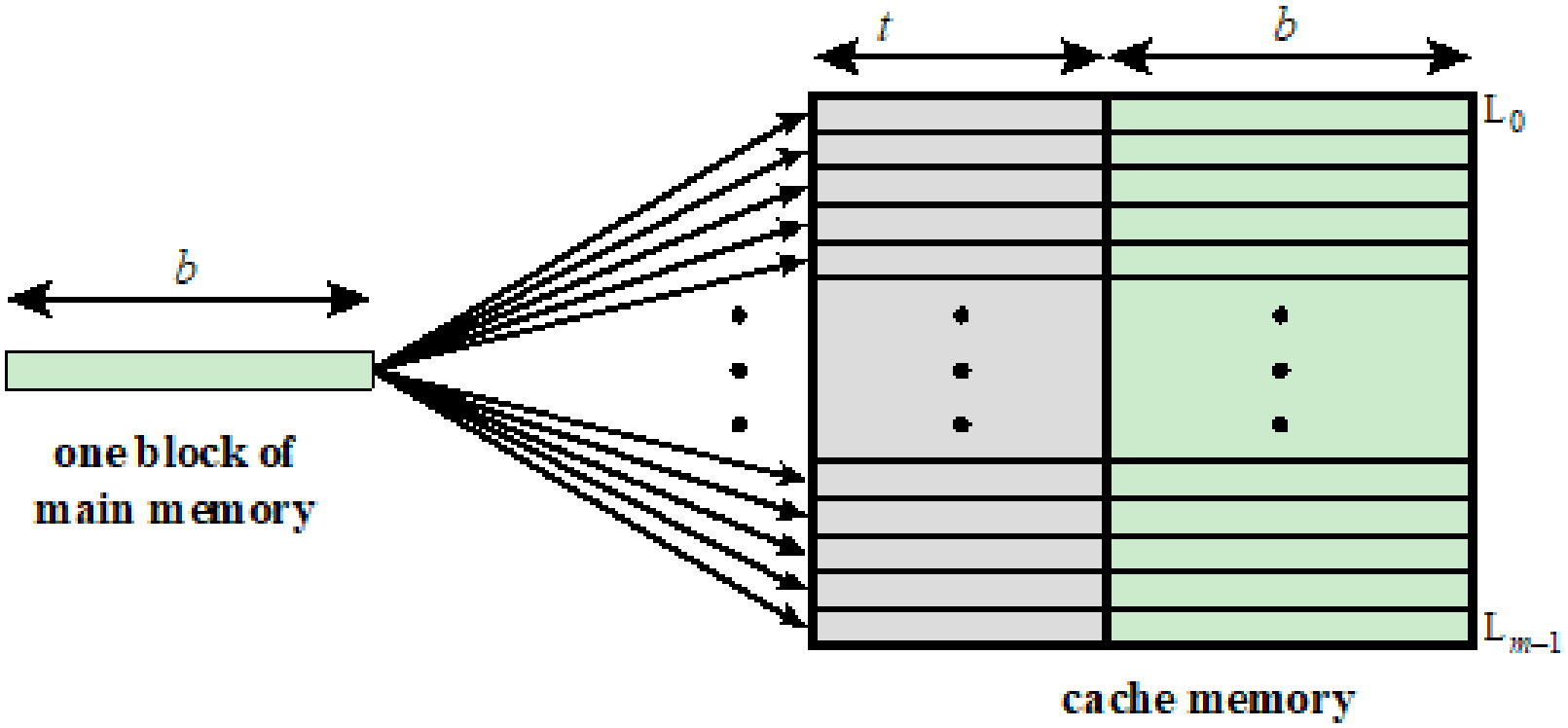
Victim Cache

- One approach to lower the miss penalty is to remember what was discarded in case it is needed again.
- Since the discarded data has already been fetched, it can be used again at a small cost.
- Such recycling is possible using a *victim cache*.
- Victim cache was proposed as an approach to reduce the conflict misses of direct-mapped caches without affecting their fast access time.
- A victim cache is a fully associative cache, whose size is typically 4 to 16 cache lines, residing between a direct mapped L1 cache and the next level of memory.

2. Associative Mapping

- Overcomes the disadvantage of direct mapping by permitting each main memory block to be loaded into any line of cache (Fig. 4.8b).
- The cache control logic interprets a memory address as a Tag and a Word field.
- The Tag field uniquely identifies a block of main memory.
- The cache control logic must simultaneously examine every line's tag for a match to determine whether a block is in the cache.
- Figure 4.11 illustrates the logic.
- With associative mapping, there is flexibility as to which block to replace when a new block is read into the cache.
- The disadvantage of associative mapping is the complex circuitry required to examine the tags of all cache lines in parallel.

Associative Mapping



(b) Associative mapping

Figure 4.8 Mapping From Main Memory to Cache: Direct and Associative

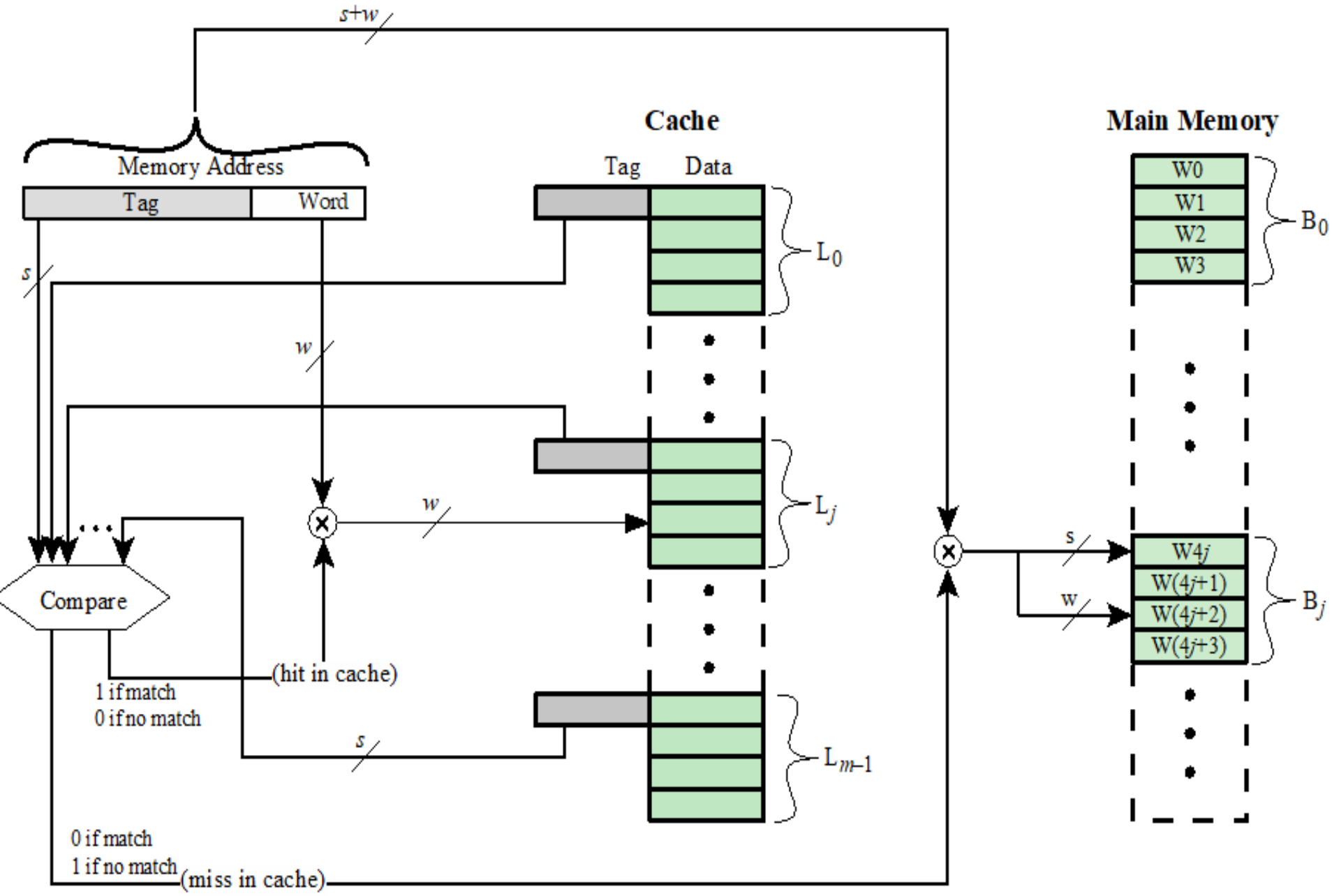


Figure 4.11 Fully Associative Cache Organization

Associative Mapping: Example

- Figure 4.12 shows an 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.
- The leftmost significant 22 bits of the address that form the tag.
- The 24-bit hexadecimal address **16339C** has the 22-bit tag 058CE7 as shown.

Memory address	0001	0110	0011	0011	1001	1100	(binary)
	1	6	3	3	9	C	(hex)
Tag (leftmost 22 bits)	00	0101	1000	1100	1110	0111	(binary)
	0	5	8	C	E	7	(hex)

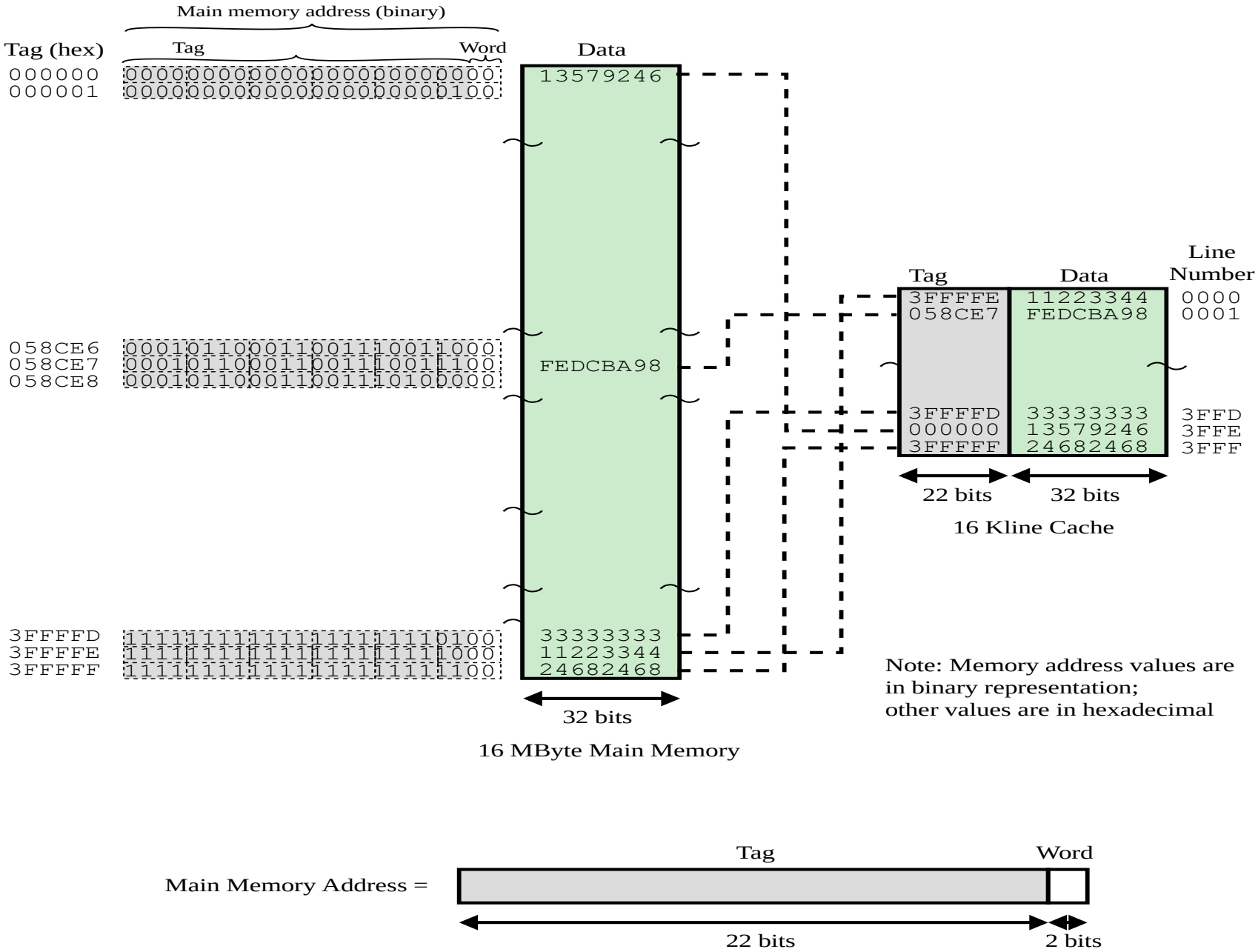


Figure 4.12 Associative Mapping Example

Associative Mapping Summary

- 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 = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

3. Set Associative Mapping

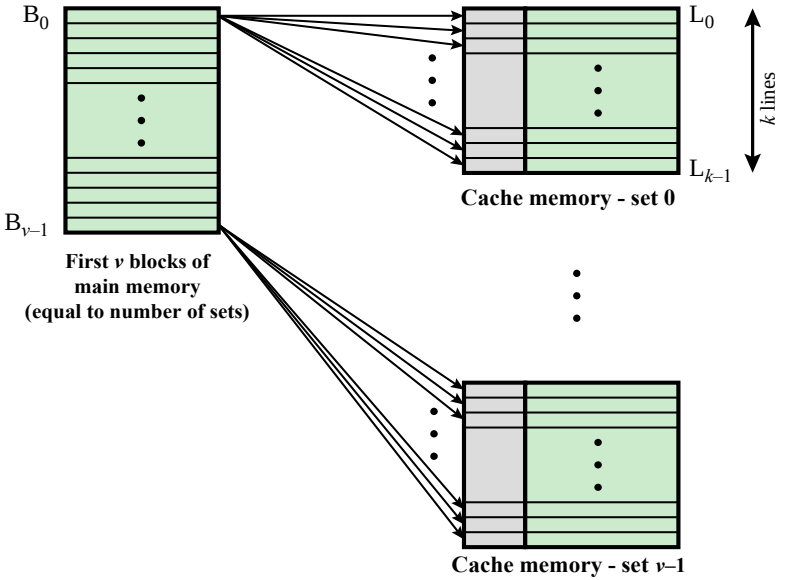
- Exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.
- A cache consists of several sets.
- Each set contains several lines.
- The relationships are

$$m = v \times k$$

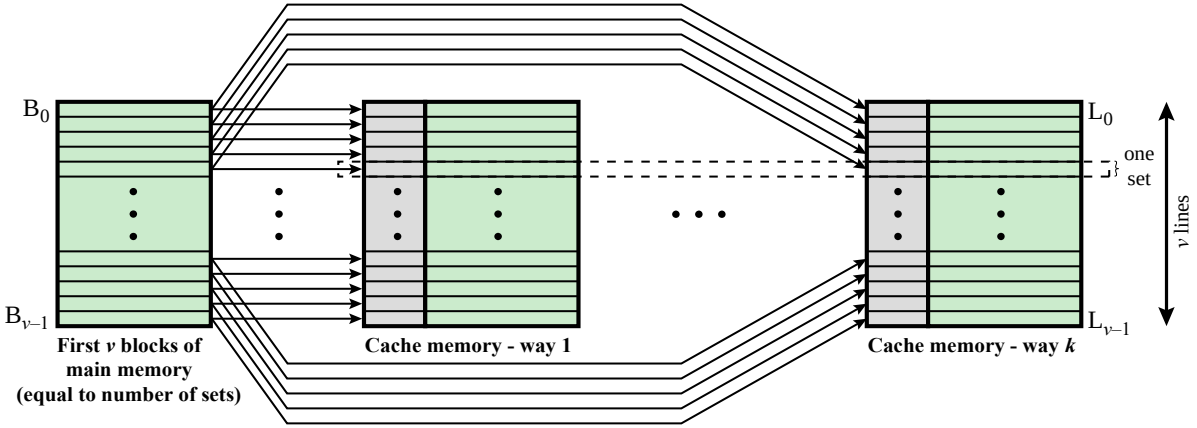
$$i = j \text{ modulo } v$$

Where: *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 set*

- This is referred to as *k-way set-associative mapping.*
- Block $\mathbf{B_j}$ can be mapped into any of the lines of set **j.**



(a) *v* associative-mapped caches



(b) *k* direct-mapped caches

Figure 4.13 Mapping From Main Memory to Cache: *k*-way Set Associative

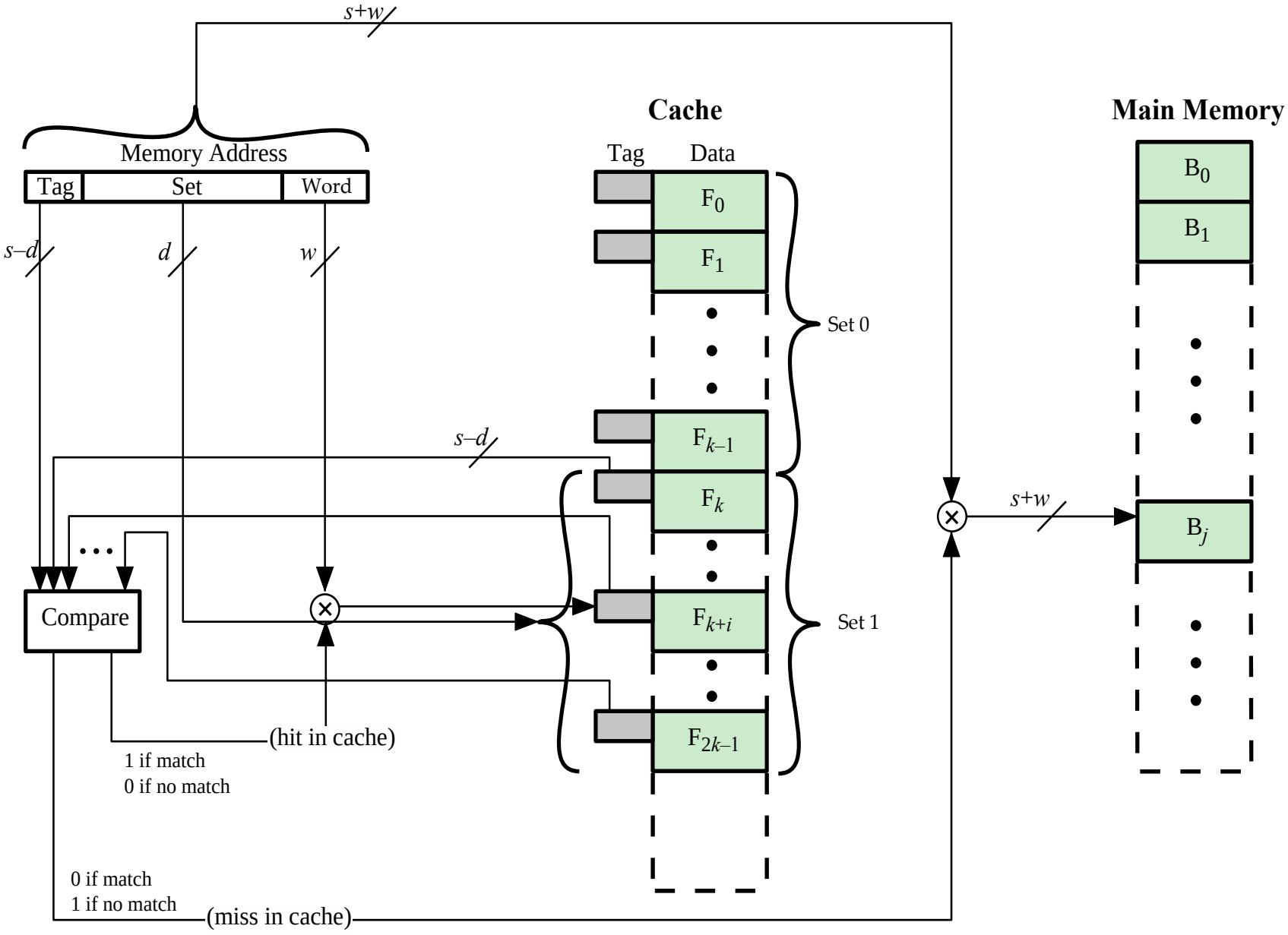


Figure 4.14 *k*-Way Set Associative Cache Organization

Set Associative Mapping Summary

- 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 = $2^{s+w}/2^{w}=2^{s}$
- Number of lines in set = k
- Number of sets = v = 2^{d}
- Number of lines in cache = m=kv = k x 2^{d}
- Size of cache = *k* x 2^{d+w} words or bytes
- Size of tag = (s – d) bits

Set Associative Mapping: Example

- Figure 4.15 shows an example using set-associative mapping with two lines in each set, referred to as two-way set-associative.
- A 13-bit set number identifies a unique set of two lines within cache.
- It also gives the number of the block in the main memory, modulo 2^{13}.
- This determines the mapping of blocks into lines, blocks 000000, 008000, …, FF8000 of main memory map into cache set 0.
- Any of those blocks can be loaded into either of the two lines in set.
- No two blocks that map into the same cache set should 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.

Set Associative Mapping: Example

- Figure 4.15 shows an example using set-associative mapping with two lines in each set, referred to as two-way set-associative.
- In the extreme case of:
  - v = m, k = 1, it reduces to direct mapping
  - v = 1, k = m, it reduces to associative mapping
- The use of two lines per set (v = m/2, k = 2) is the most common set-associative organization.
- It significantly improves the hit ratio over direct mapping.
- Four-way set associative (v = m/4, k = 4) makes a modest additional improvement for a relatively small additional cost.

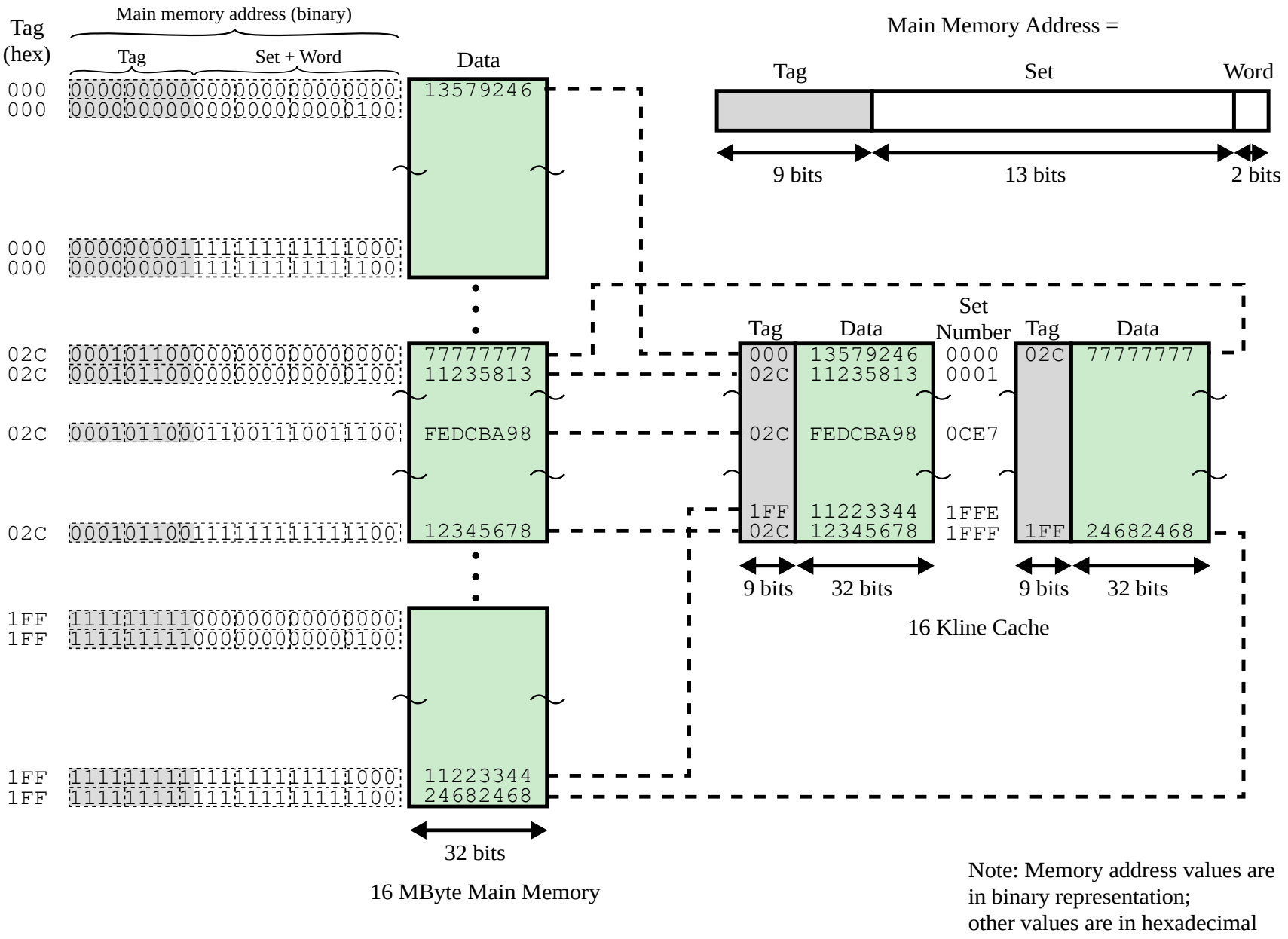


Figure 4.15 Two-Way Set Associative Mapping Example

Replacement Algorithms

- Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced.
- For direct mapping, there is only one possible line for any particular block and no choice is possible.
- For the associative and set-associative techniques, a ***replacement algorithm*** is needed.
- To achieve high speed, an algorithm must be implemented in hardware.

Most Common Replacement Algorithms

- Least recently used (LRU)
  - Most effective
  - Replace that block in the set that has been in the cache longest with no reference to it
  - Because of its simplicity of implementation, LRU is the popular replacement algorithm
- First-in-first-out (FIFO)
  - Replace that block in the set that has been in the cache longest
  - Easily implemented as a round-robin or circular buffer technique
- Least frequently used (LFU)
  - Replace that block in the set that has experienced the fewest references
  - This could be implemented by associating a counter with each line

Write Policy

When a block that is resident in the cache is to be replaced there are two cases to consider:

If the old block in the cache has not been altered then it may be overwritten with a new block without first writing out the old block

If at least one write operation has been performed on a word in that line of the cache then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

There are two problems to contend with:

More than one device may have access to main memory

A more complex problem occurs when multiple processors are attached to the same bus and each processor has its own local cache - if a word is altered in one cache it could conceivably invalidate a word in other caches

Write Through and Write Back

- Write Through
  - Simplest technique
  - All write operations are made to the main memory as well as to the cache
  - The disadvantage of this technique is that it generates substantial memory traffic and may create a bottleneck
- Write Back
  - Minimizes memory writes
  - Updates are made only in the cache
  - Portions of main memory are invalid and hence accesses by I/O modules can be allowed only through the cache
  - This makes for complex circuitry and a potential bottleneck

Write Policy

- In a bus organization, if more than one device (processor) has a cache and the main memory is shared, a new problem is introduced.
- If data in one cache are altered, this invalidates not only the corresponding word in main memory but also that same word in other caches (if any other cache happens to have that same word).
- Even if a write-through policy is used, the other caches may contain invalid data.
- A system that prevents this problem is said to maintain cache coherency.
- Possible approaches to cache coherency include the following:
  - Bus-watching with write-through
  - Hardware transparency
  - Non-cacheable memory

Write Policy

- **Bus watching with write through:**
  - Each cache controller monitors the address lines to detect write operations to memory by other bus masters.
  - If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry.
  - Strategy depends on the use of a write-through policy by all cache controllers.
- **Hardware transparency:**
  - Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches.
  - If one processor modifies a word in its cache, this update is written to the main memory.
  - Any matching words in other caches are similarly updated.
- **Non-cacheable memory:**
  - Only a portion of the main memory is shared by more than one processor, and this is designated as *non-cacheable*.
  - All accesses to shared memory are cache misses because the shared memory is never copied into the cache.
  - Non-cacheable memory is identified using chip-select logic or high-address bits

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 are retrieved.
- As the block size increases, the hit ratio will at first **increase** because of the principle of locality, which states that data in the vicinity of a referenced word are likely to be referenced soon.
- As the block size increases, more useful data are brought into the cache.
- The hit ratio will **decrease**, as the block becomes bigger and the probability of using the newly fetched information becomes less than the probability of reusing the information that has to be replaced.
- Two specific effects come into play:
  - Larger blocks reduce the number of blocks that fit into a cache. Because each block fetch overwrites older cache contents, a small number of blocks results in data being overwritten shortly after they are fetched.
  - As a block becomes larger, each additional word is farther from the requested word and therefore less likely to be needed shortly.
- For HPC systems, 64- and 128-byte cache line sizes are used.

Multilevel Caches

- As logic density has increased, it is possible to have a cache on the same chip as the processor.
- The on-chip cache reduces the processor's external bus activity and speeds up execution time and increases overall system performance:
  - When the requested instruction or data is found in the on-chip cache, the bus access is eliminated
  - On-chip cache accesses will complete faster than even zero-wait state bus cycles
  - During this period the bus is free to support other transfers
- Two-level cache:
  - Internal cache designated as level 1 (L1)
  - External cache designated as level 2 (L2)
- Potential savings from using an L2 cache depends on the hit rates in both the L1 and L2 caches.
- The use of multilevel caches complicates all design issues related to caches, including size, replacement algorithm, and write policy.

Figure 4.17 shows the results of one simulation study of two-level cache performance as a function of cache size

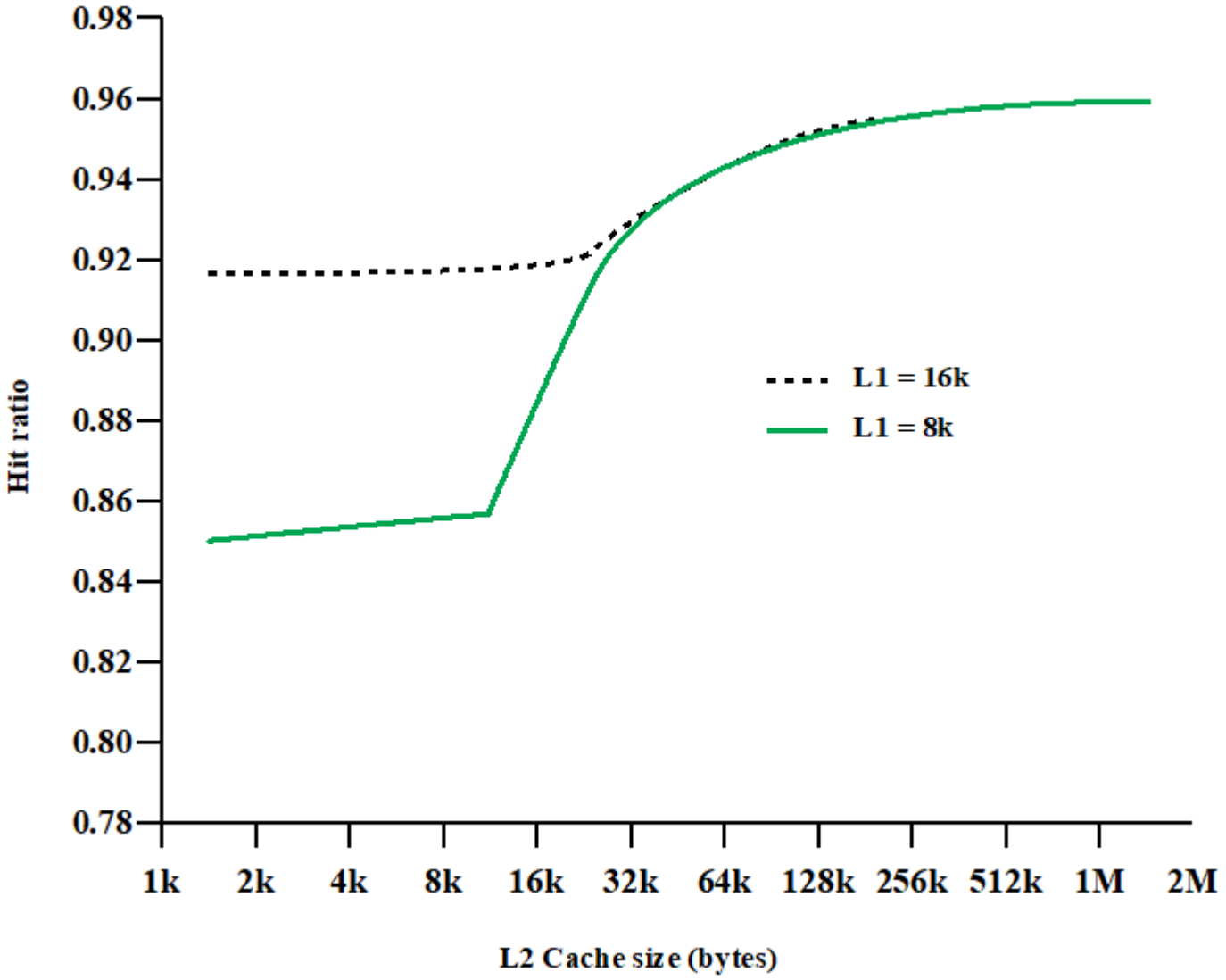


Figure 4.17 Total Hit Ratio (L1 and L2) for 8 Kbyte and 16 Kbyte L1

Unified Versus Split Caches

- Has become common to split cache:
  - One dedicated to instructions
  - One dedicated to data
  - Both exist at the same level, typically as two L1 caches
- Advantages of unified cache:
  - Higher hit rate
    - Balances load of instruction and data fetches automatically
    - Only one cache needs to be designed and implemented
- The trend is toward split caches at the L1 and unified caches for higher levels.
- Advantages of split cache:
  - Eliminates cache contention for the cache between the instruction fetch/decode unit and execution unit
  - Important in pipelining

Cache Memory

- Computer Memory System Overview
- Cache Memory Principles
- Elements of Cache Design
- **Pentium 4 Cache Organization**

Problem	Solution	Processor on which Feature First Appears
External memory slower than the system bus.	Add external cache using faster memory technology.	386
Increased processor speed results in external bus becoming a bottleneck for cache access.	Move external cache on-chip, operating at the same speed as the processor.	486
Internal cache is rather small, due to limited space on chip	Add external L2 cache using faster technology than main memory	486
Contention occurs when both the Instruction Prefetcher and the Execution Unit simultaneously require access to the cache. In that case, the Prefetcher is stalled while the Execution Unit's data access takes place.	Create separate data and instruction caches.	Pentium
Increased processor speed results in external bus becoming a bottleneck for L2 cache access.	Create separate back-side bus that runs at higher speed than the main (front-side) external bus. The BSB is dedicated to the L2 cache.	Pentium Pro
	Move L2 cache on to the processor chip.	Pentium II
Some applications deal with massive databases and must have rapid access to large amounts of data. The on-chip caches are too small.	Add external L3 cache.	Pentium III
	Move L3 cache on-chip.	Pentium 4

Table 4.4

Intel Cache Evolution

Pentium 4 Organization

- 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
  - Out-of-order execution logic
  - Execution units
  - Memory subsystem
- The Pentium 4 instruction cache sits between the instruction decode logic and the execution core.
- The data cache employs a write-back policy:
  - Data are written to main memory only when they are removed from the cache and there has been an update.
  - Dynamically configured to support write-through caching.

Simplified view of the Pentium 4 Organization

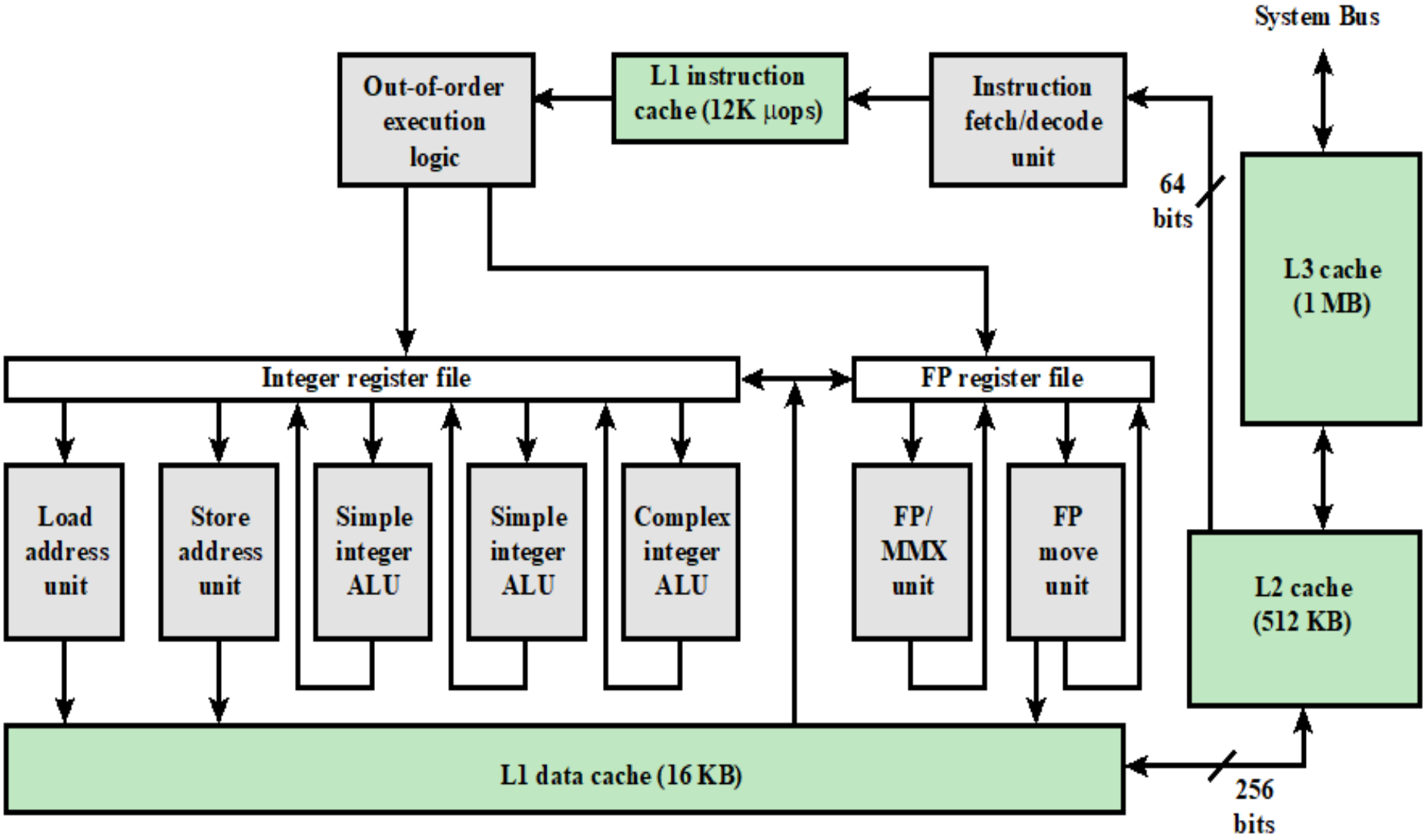


Figure 4.18 Pentium 4 Block Diagram

- The L1 data cache is controlled by two bits in one of the control registers, labeled CD (cache disable) and NW (not write-through) bits.
- The Pentium 4 instructions used to control the data cache are:
  - INVD invalidates (flushes) the internal cache memory and signals the external cache (if any) to invalidate.
  - WBINVD writes back and invalidates the internal cache and then writes back and invalidates the external cache.
- Both the L2 and L3 caches are eight-way set-associative with a line size of 128 bytes.

Table 4.5 Pentium 4 Cache Operating Modes

Control Bits		Operating Mode		
CD	**NW**	**Cache Fills**	**Write Throughs**	**Invalidates**
0	0	Enabled	Enabled	Enabled
1	0	Disabled	Enabled	Enabled
1	1	Disabled	Disabled	Disabled

Note: CD = 0; NW = 1 is an invalid combination.

Summary

- Computer memory system overview
  - Characteristics of Memory Systems
  - Memory Hierarchy
- Cache memory principles
- Pentium 4 cache organization
- Elements of cache design
  - Cache addresses
  - Cache size
  - Mapping function:
    - Direct, Associative, Set-associative
    - Replacement algorithms
    - Write policy
    - Line size
    - Number of caches

Numericals

Thank You