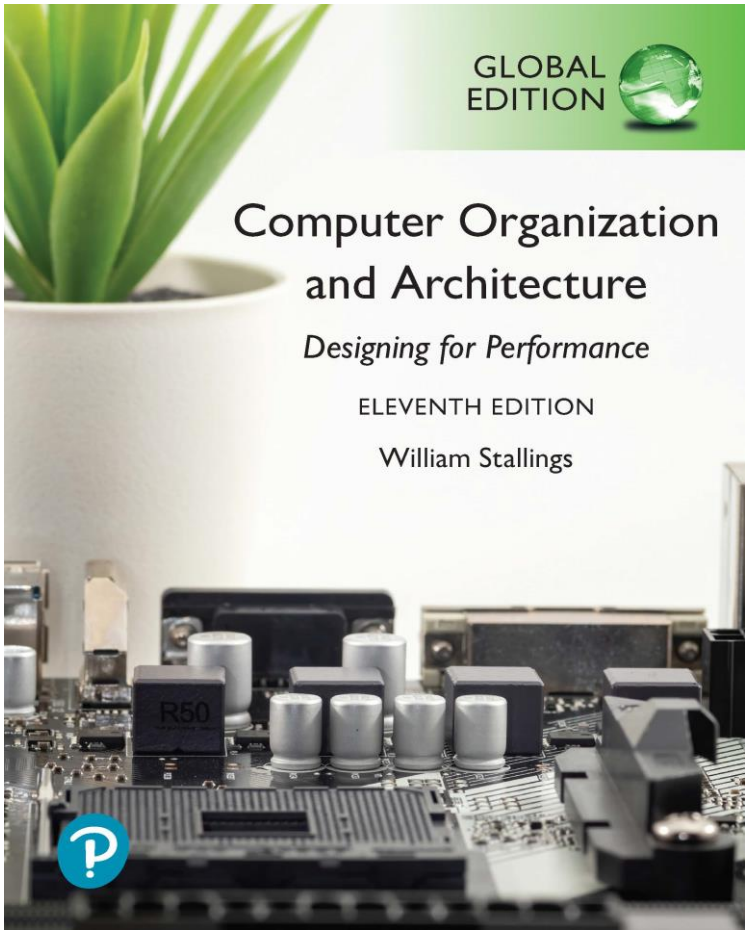
Computer Organization and Architecture

Designing for Performance

11th Edition, Global Edition



Chapter 3

A Top-Level View of Computer Function and Interconnection

Computer Components

- Contemporary computer designs are based on concepts developed by John von Neumann at the Institute for Advanced Studies, Princeton
- Referred to as the *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 in a sequential fashion (unless explicitly modified) from one instruction to the next

– *Hardwired program*

  - Is the result of the process of connecting the various logic components in the desired configuration for a particular computation. The resulting "program" is in the form of hardware and is termed a hardwired program.

Hardware and Software Approaches

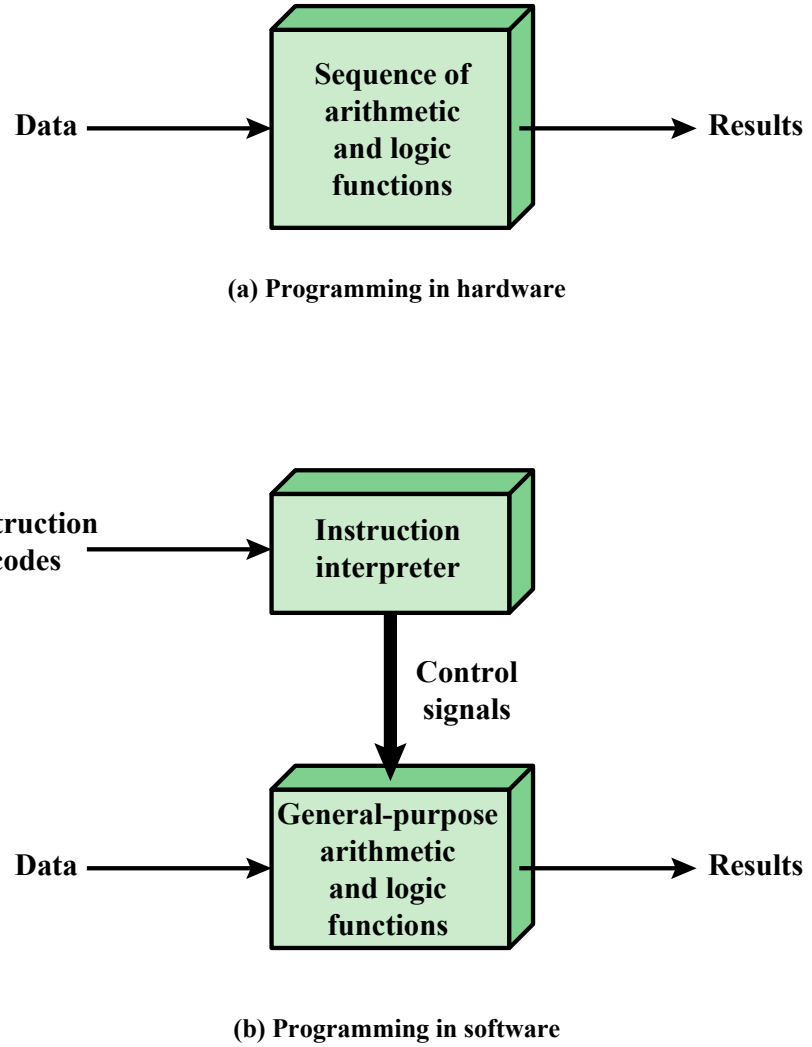


Figure 3.1 Hardware and Software Approaches

Software

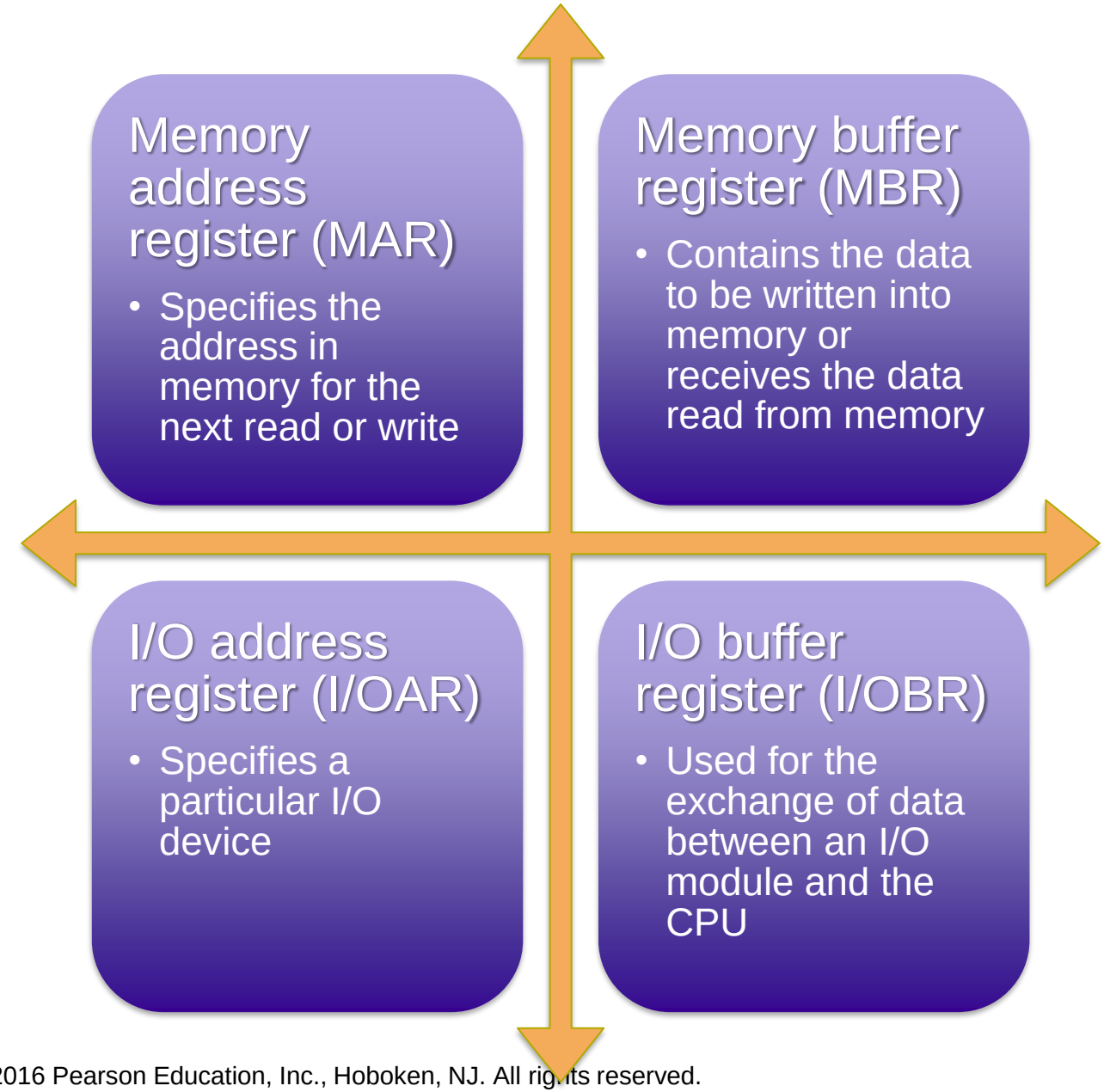
- A sequence of codes or instructions
- Part of the hardware interprets each instruction and generates control signals
- Provide a new sequence of codes for each new program instead of rewiring the hardware

Software

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

I/O Components



MEMORY

MAR

MBR

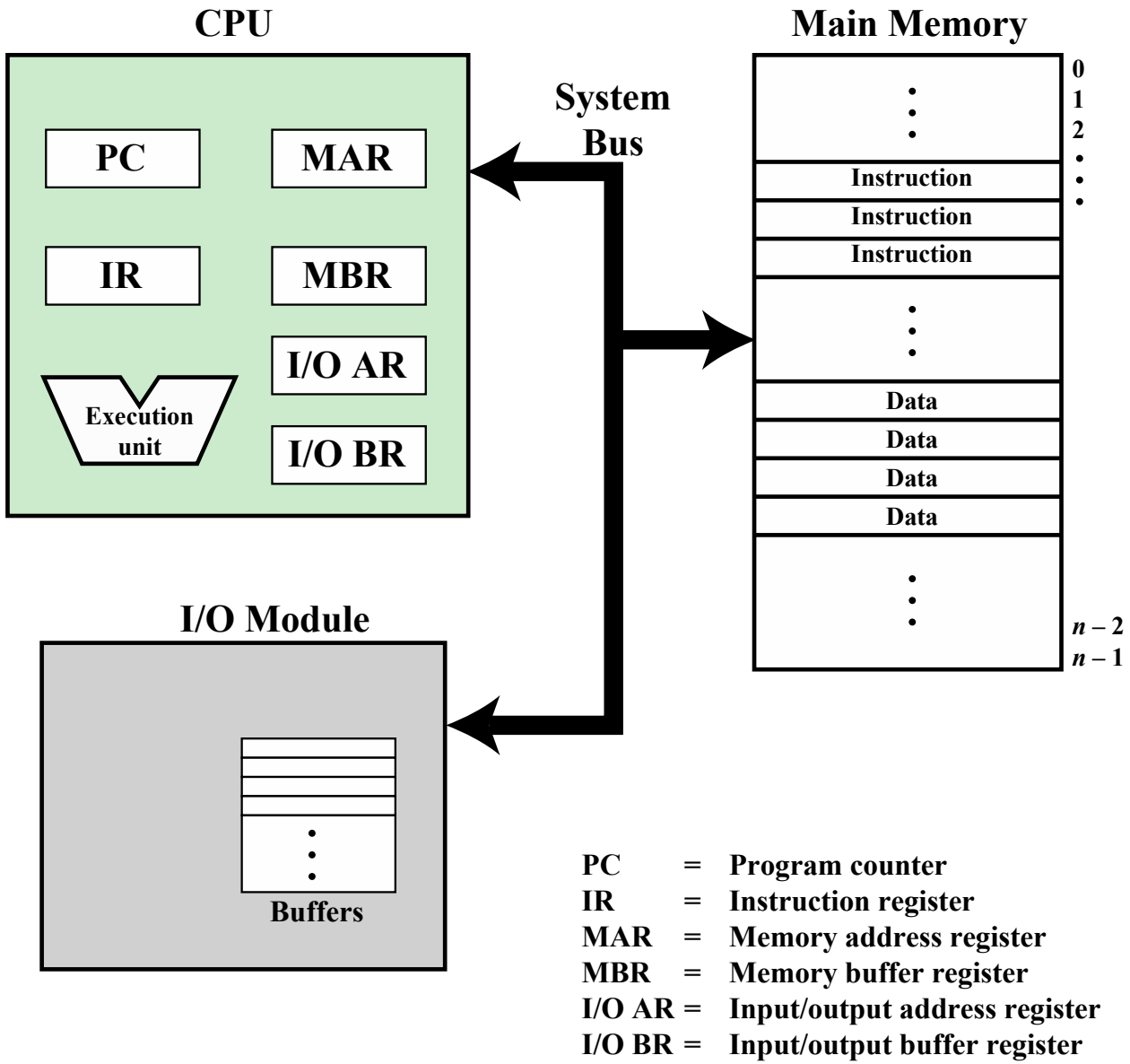


Figure 3.2 Computer Components: Top-Level View

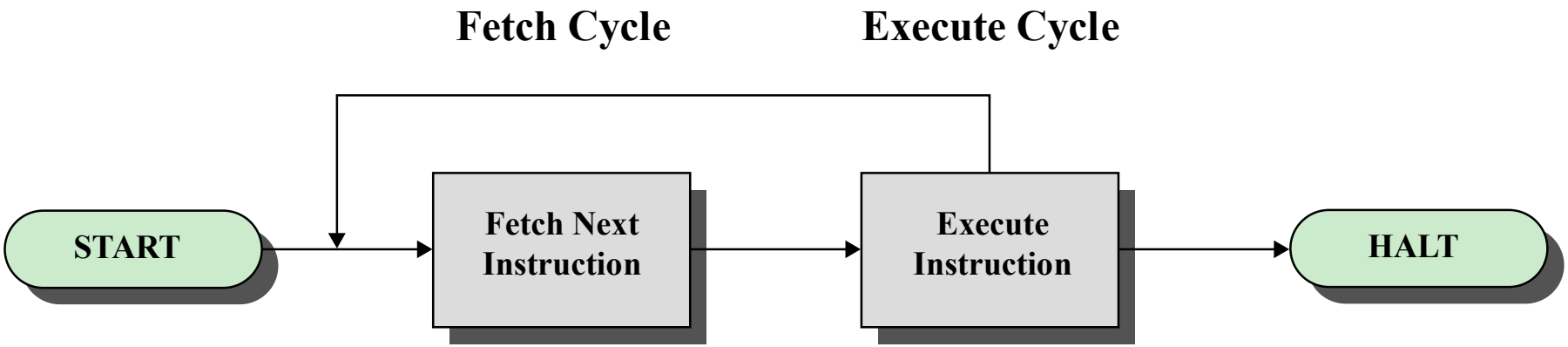


Figure 3.3 Basic Instruction Cycle

Fetch Cycle

- At the beginning of each instruction cycle the processor fetches an instruction from memory

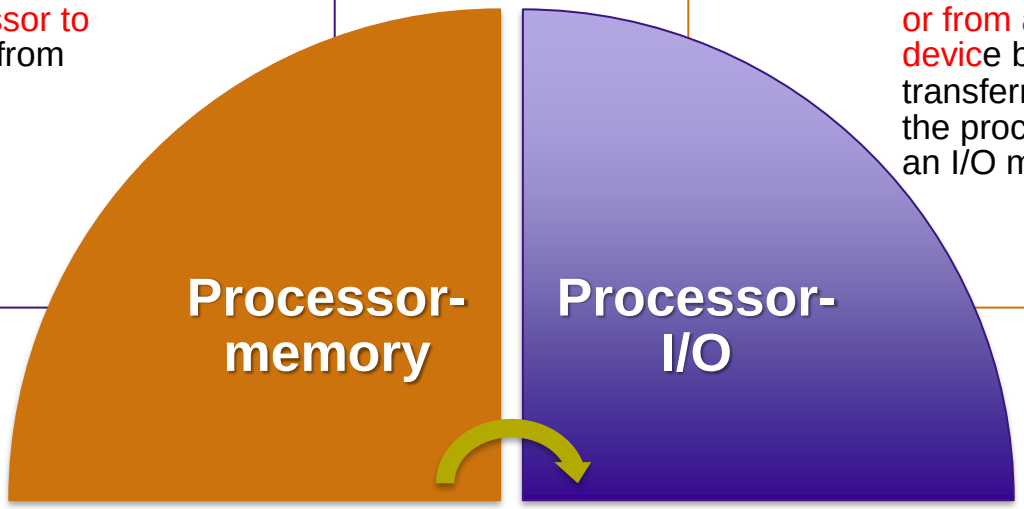
– The program counter (PC) holds the address of the instruction to be fetched next

– 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

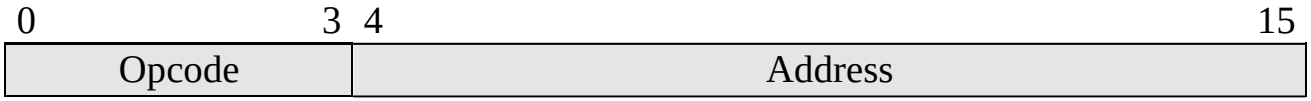


- Data transferred from processor to memory or from memory to processor

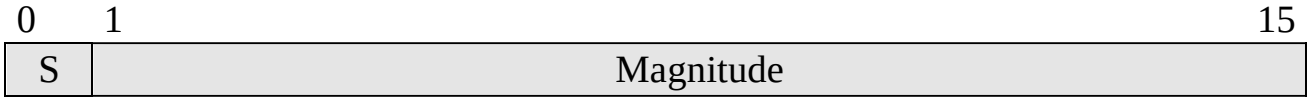
- Data transferred to or from a peripheral device by transferring between the processor and an I/O module

- 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

LOAD AC, 940
Op Code: 1

ADD AC, 941
Op Code: 5

STO AC, 941
Op Code: 2

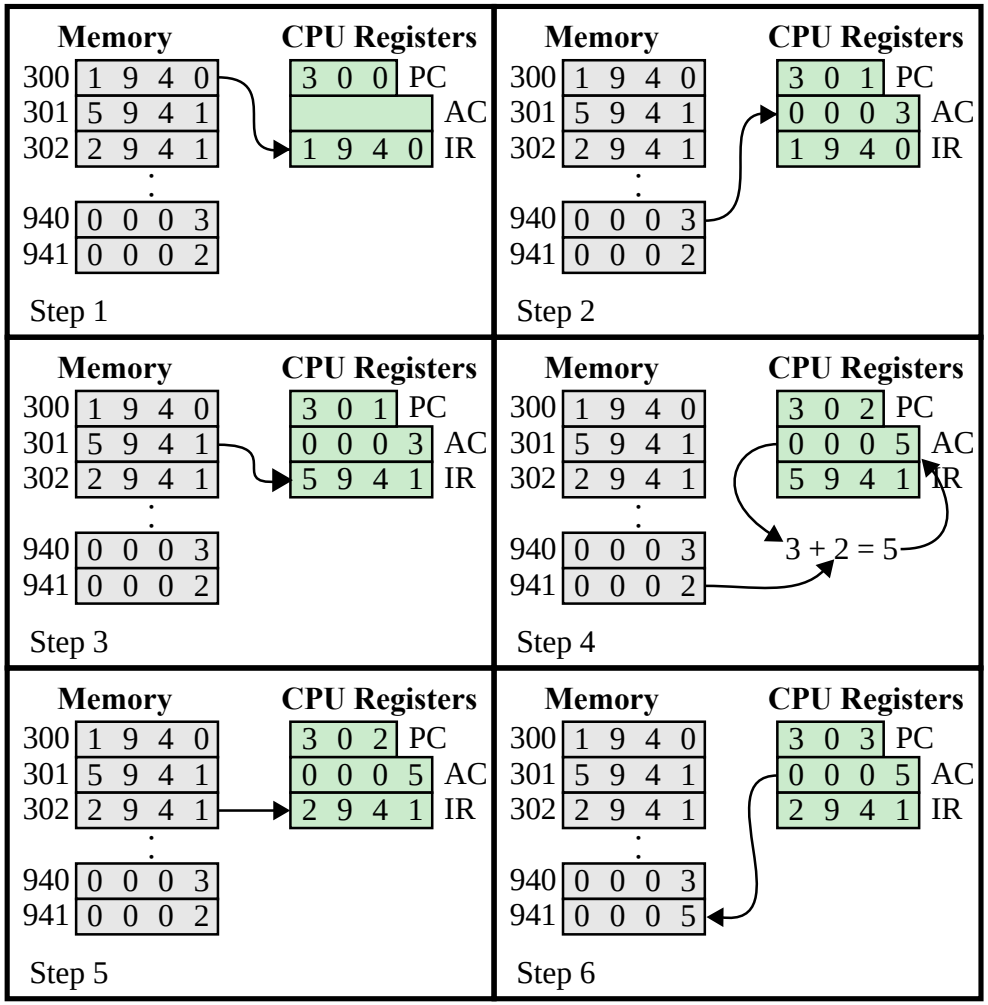


Figure 3.5 Example of Program Execution (contents of memory and registers in hexadecimal)

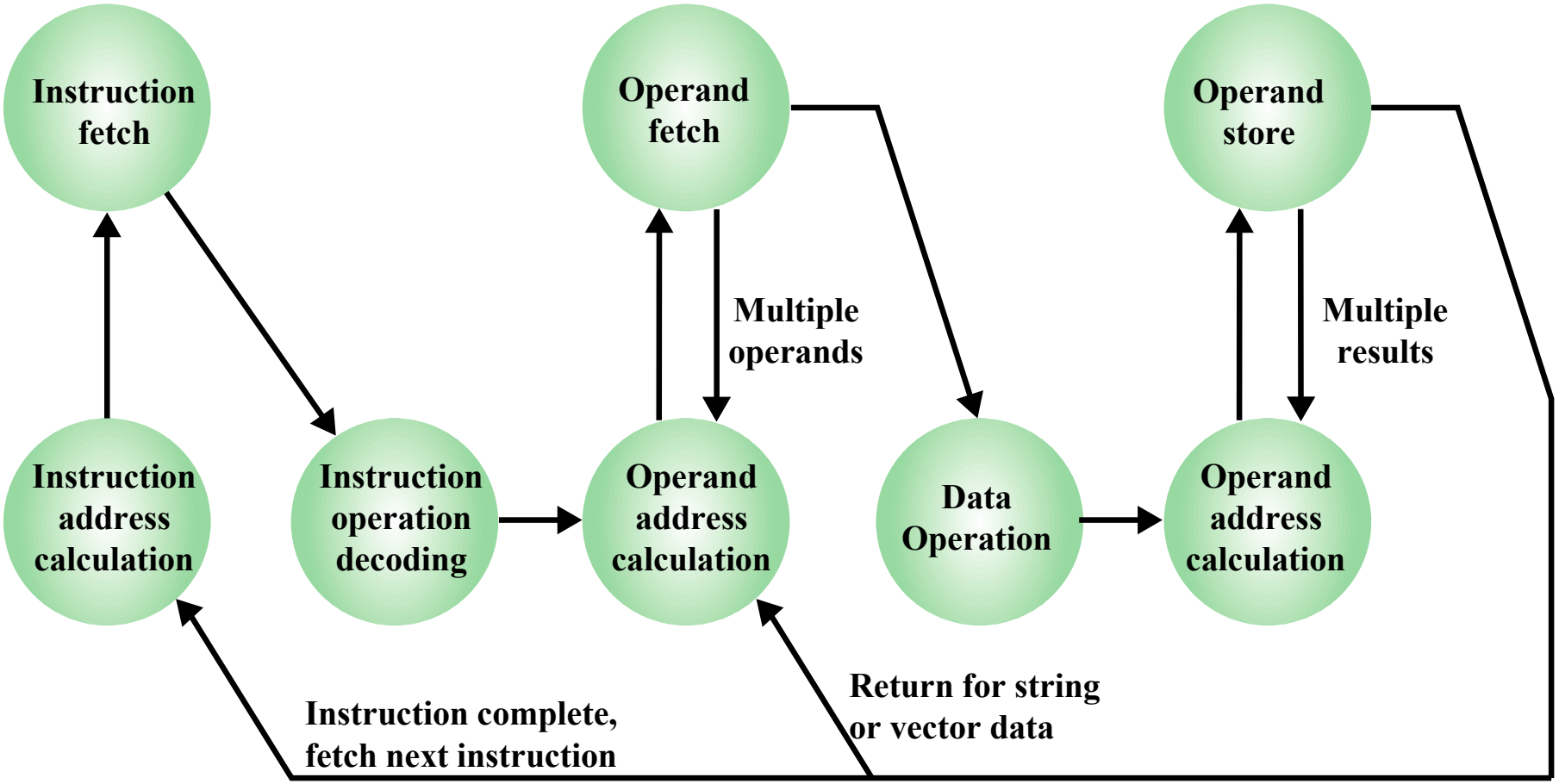


Figure 3.6 Instruction Cycle State Diagram

Interrupts

- Virtually all computers provide a mechanism by which other modules (I/O, memory) may **interrupt** the normal processing of the processor.
- Interrupts are provided primarily as a way to improve processing efficiency.
- Suppose that the processor is transferring data to a printer. After each write operation, the processor must pause and remain idle until the printer catches up. The length of this pause may be on the order of many hundreds or even thousands of instruction cycles that do not involve memory. Clearly, this is a very wasteful use of the processor.

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.

Table 3.1

Classes of Interrupts

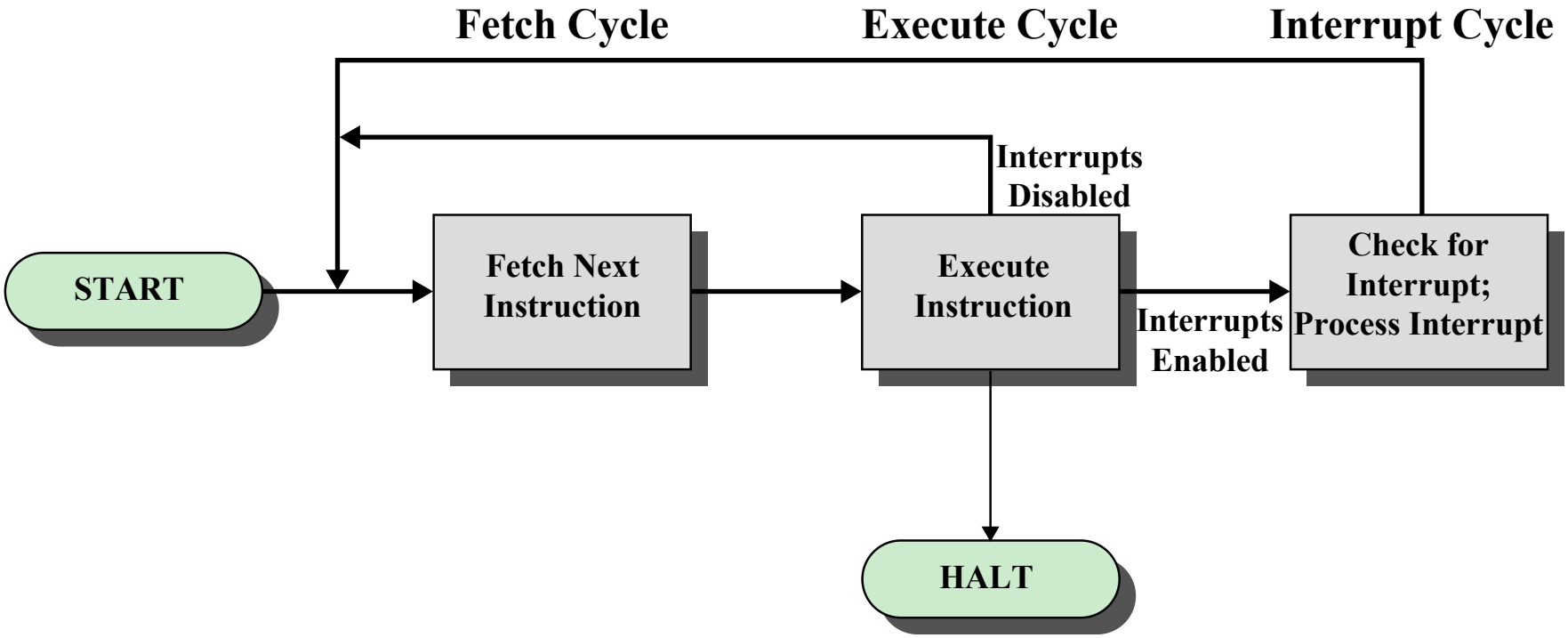


Figure 3.9 Instruction Cycle with Interrupts

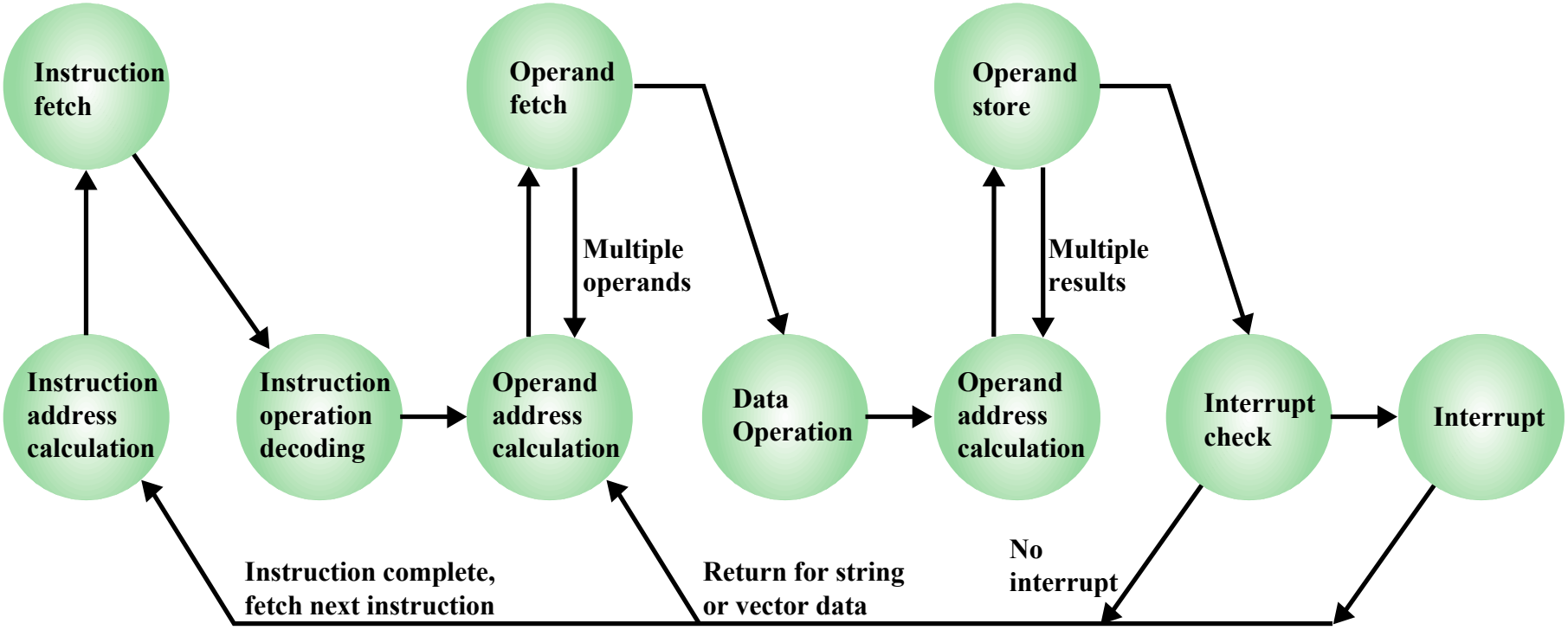


Figure 3.12 Instruction Cycle State Diagram, With Interrupts

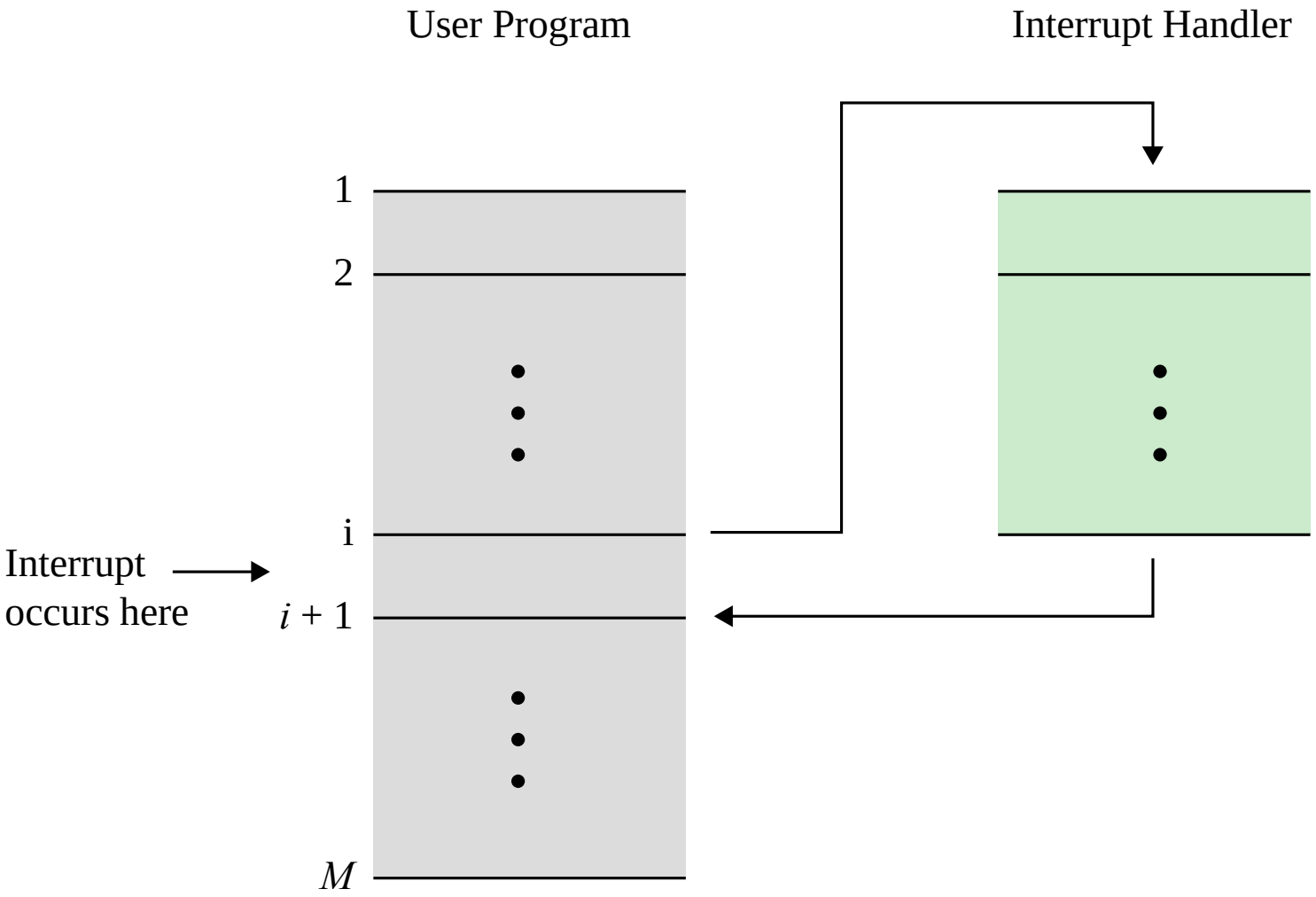
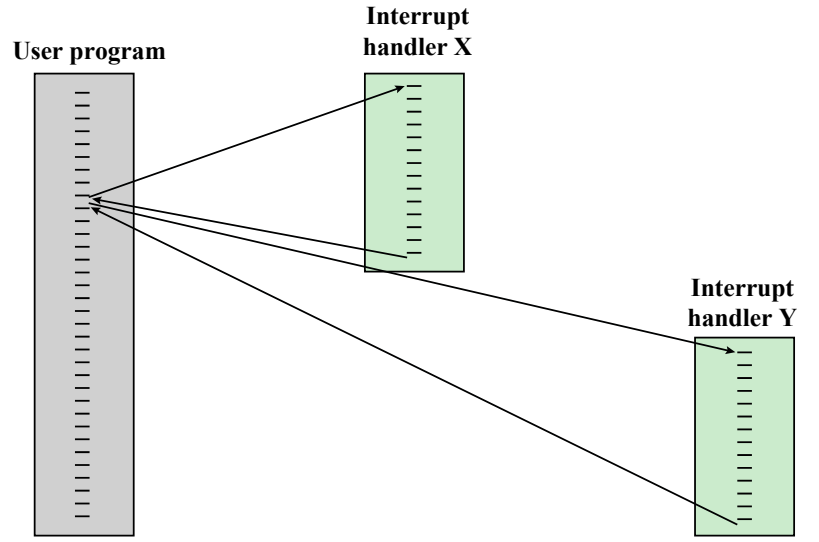
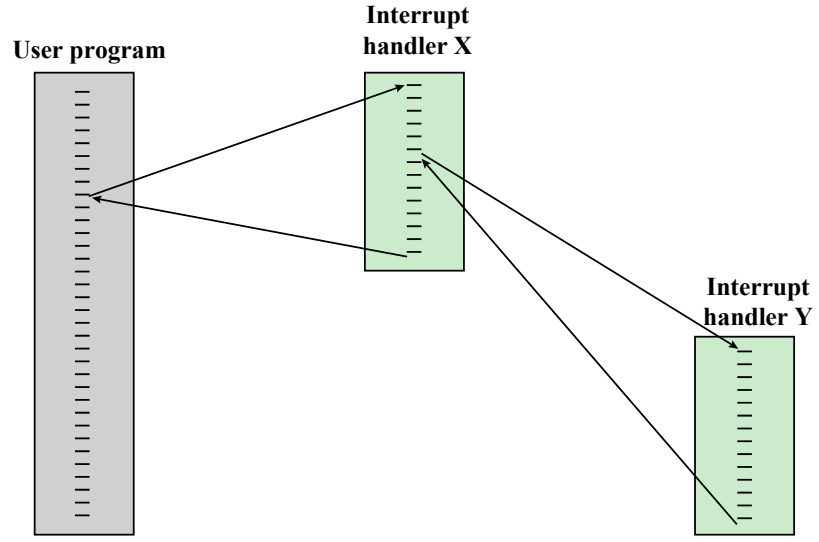


Figure 3.8 Transfer of Control via Interrupts



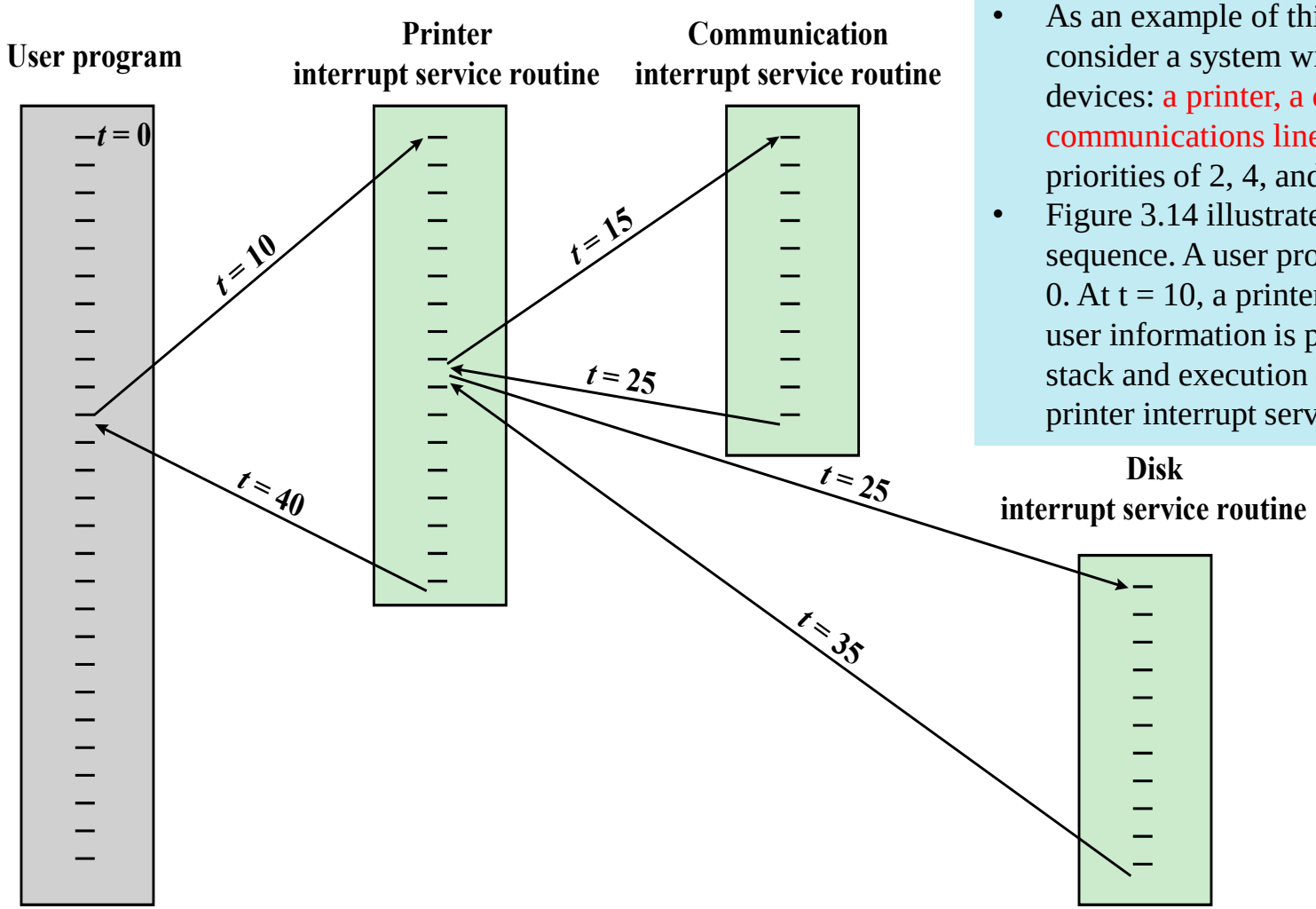
(a) Sequential interrupt processing



(b) Nested interrupt processing

Figure 3.13 Transfer of Control with Multiple Interrupts

- Two approaches can be taken to dealing with multiple interrupts.
- The first is to disable interrupts while an interrupt is being processed. A disabled interrupt simply means that the processor can and will ignore that interrupt request signal. If an interrupt occurs during this time, it generally remains pending and will be checked by the processor after the processor has enabled interrupts.
- 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 be itself interrupted (Figure 3.13b).



- As an 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 3.14 illustrates a possible sequence. 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).

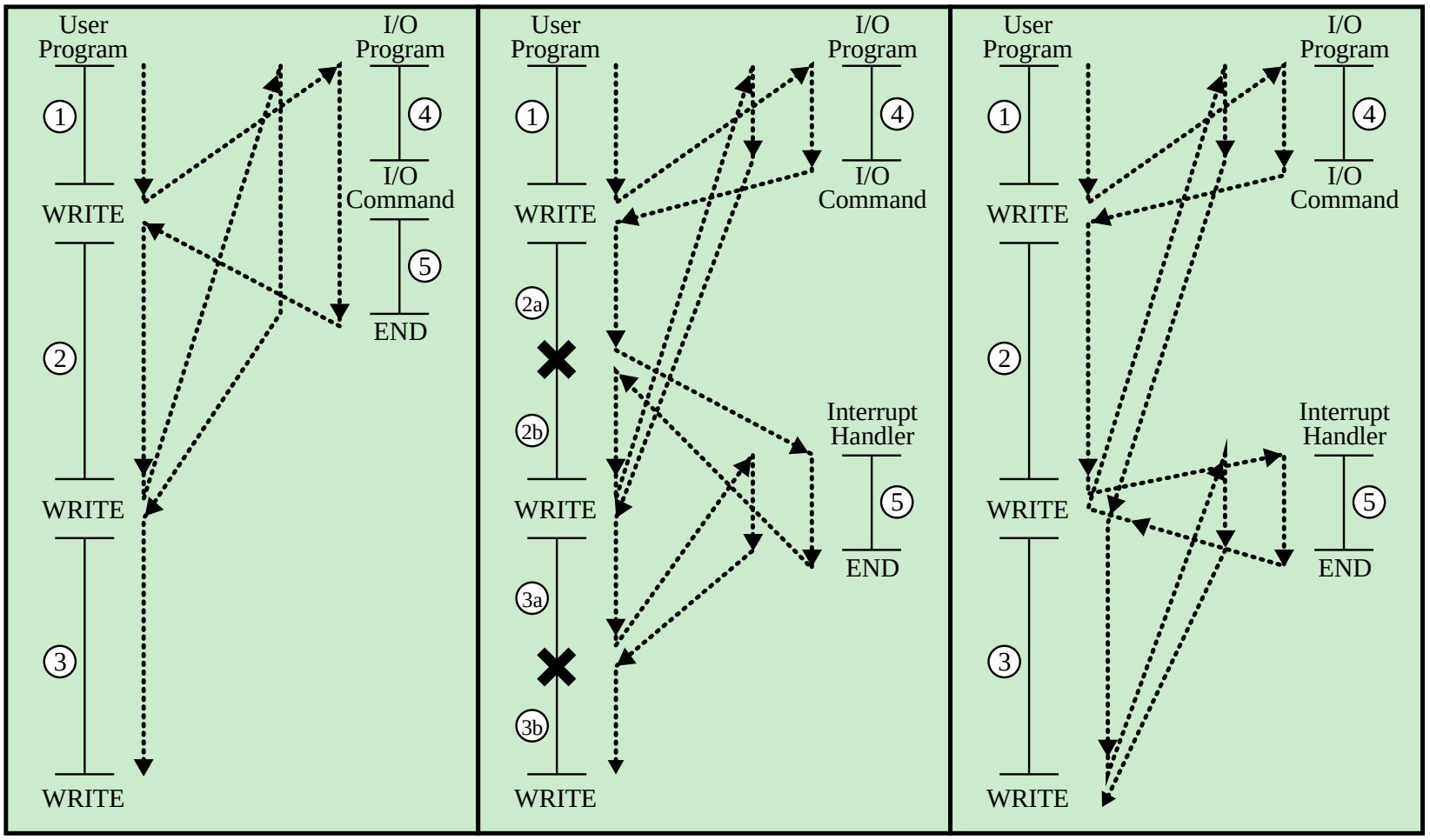
Figure 3.14 Example Time Sequence of Multiple Interrupts

I/O operation

- Figure 3.7a illustrates this state of affairs. 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. The WRITE calls are to an I/O program that 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). The program might wait by simply repeatedly performing a test operation to determine if the I/O operation is done.
  - • 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.
- Because the I/O operation may take a relatively long time to complete, the I/O program is hung up waiting for the operation to complete; hence, the user program is stopped at the point of the WRITE call for some considerable period of time.

I/O operation

- With interrupts, the processor can be engaged in executing other instructions while an I/O operation is in progress.
- Consider the flow of control in Figure 3.7b. As before, the user program reaches a point at which it makes a system call in the form of a WRITE call.
- The I/O program that is invoked in this case consists only of the preparation code and the actual I/O command. After these few instructions have been executed, control returns to the user program.
- Meanwhile, the external device is busy accepting data from computer memory and printing 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 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. The points at which such interrupts occur are indicated by an asterisk in Figure 3.7b.
- Figure 3.7c indicates this state of affairs with a slow I/O device like printer. In this case, the user program reaches the second WRITE call before the I/O operation spawned by the first call is complete. The result is that the user program is hung up at that point



(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

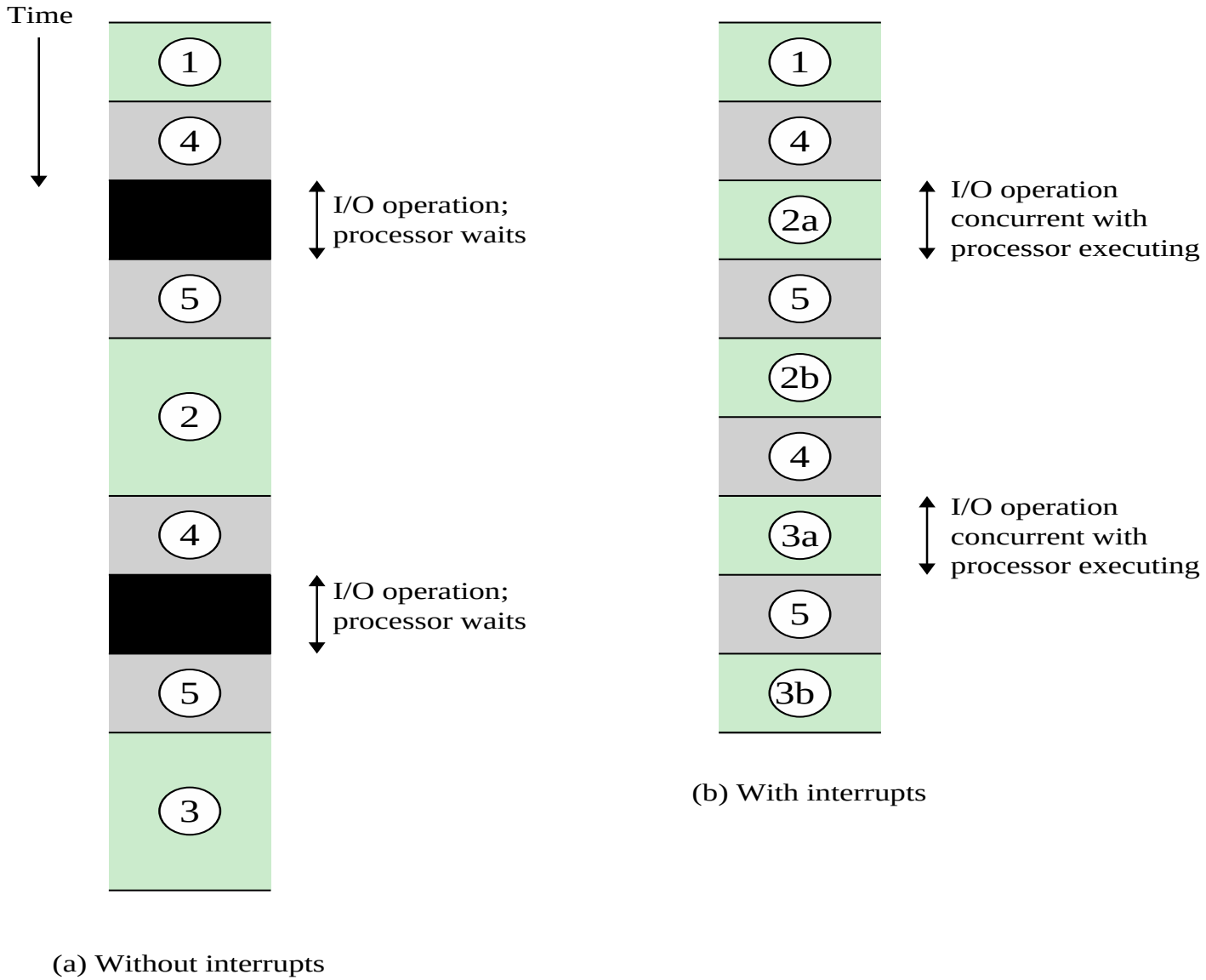


Figure 3.10 Program Timing: Short I/O Wait

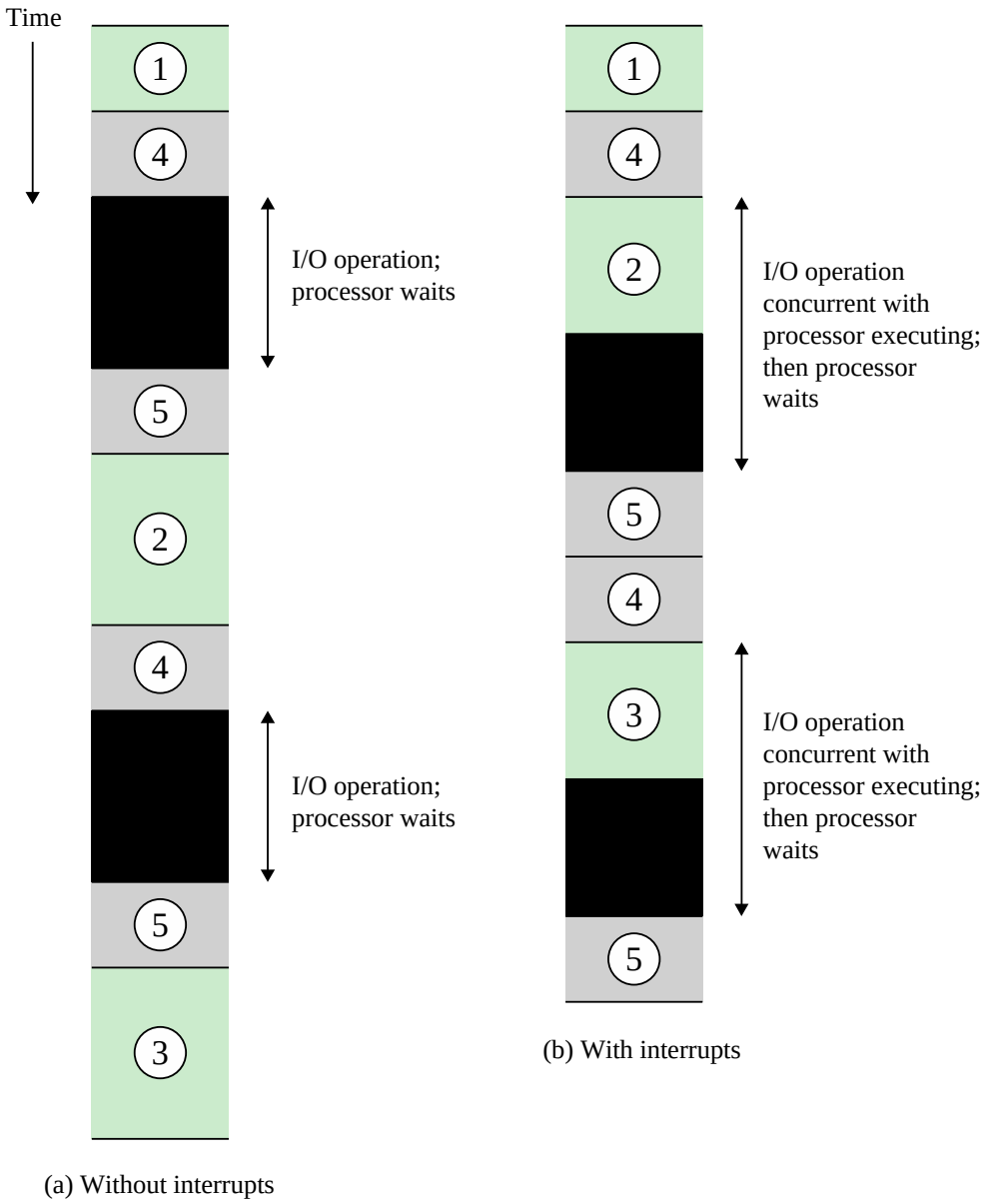


Figure 3.11 Program Timing: Long I/O Wait

I/O Function

- I/O modules can exchange data directly with the processor
- Processor can read data from or write data to an I/O module
  - 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
  - This operation is known as direct memory access (DMA)

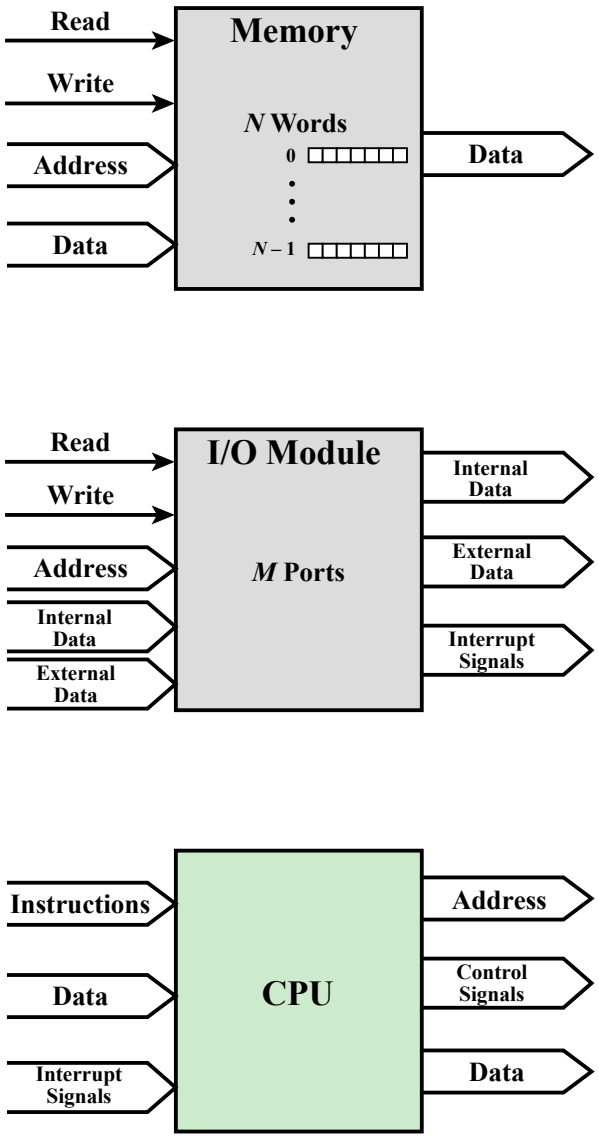
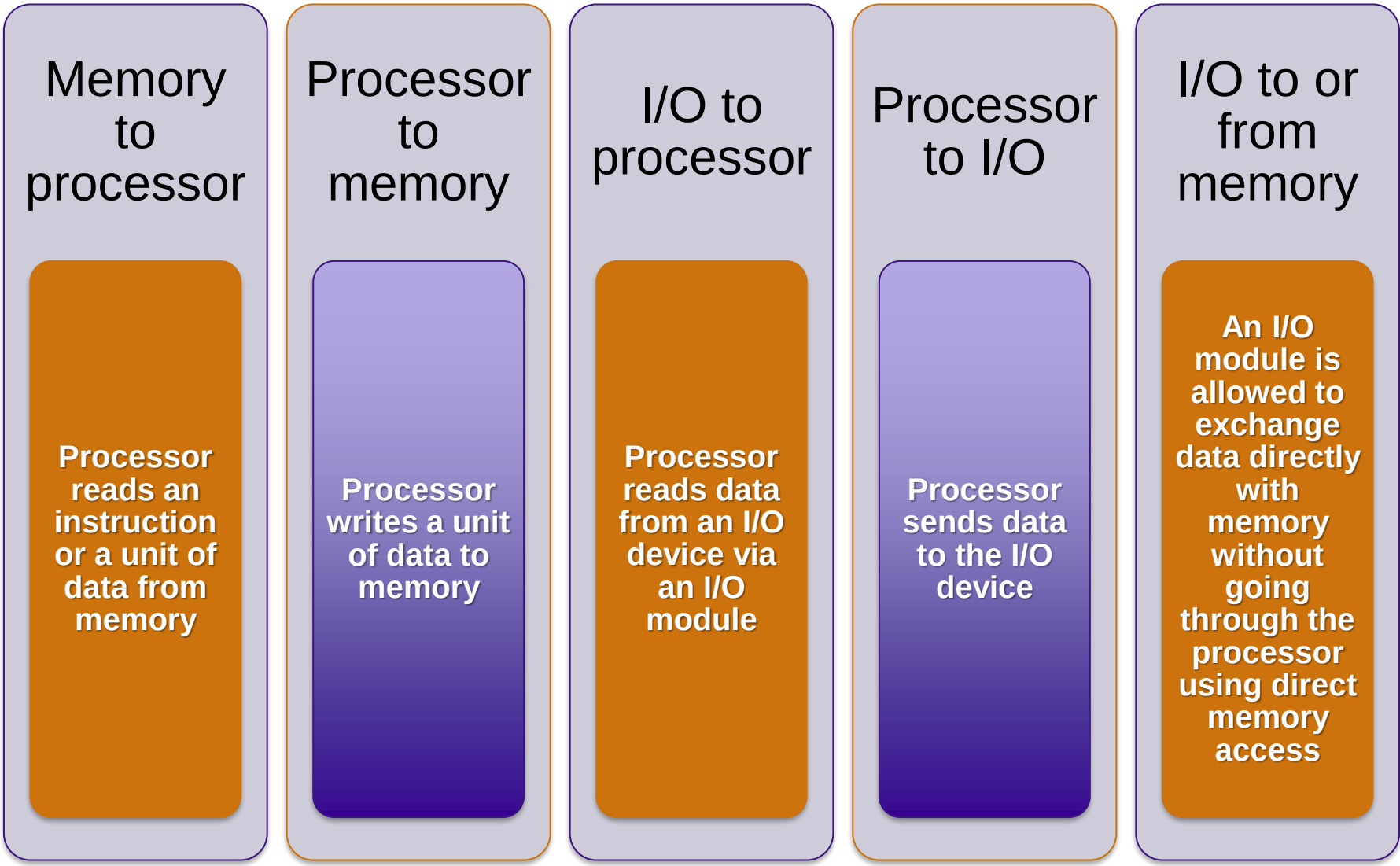


Figure 3.15 Computer Modules

- A computer consists of a set of components or modules of three basic types (processor, memory, I/O) that communicate with each other.
- In effect, a computer is a network of basic modules. Thus, there must be paths for connecting the modules

The interconnection structure must support the following types of transfers:



Interconnection Bus

A bus is 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

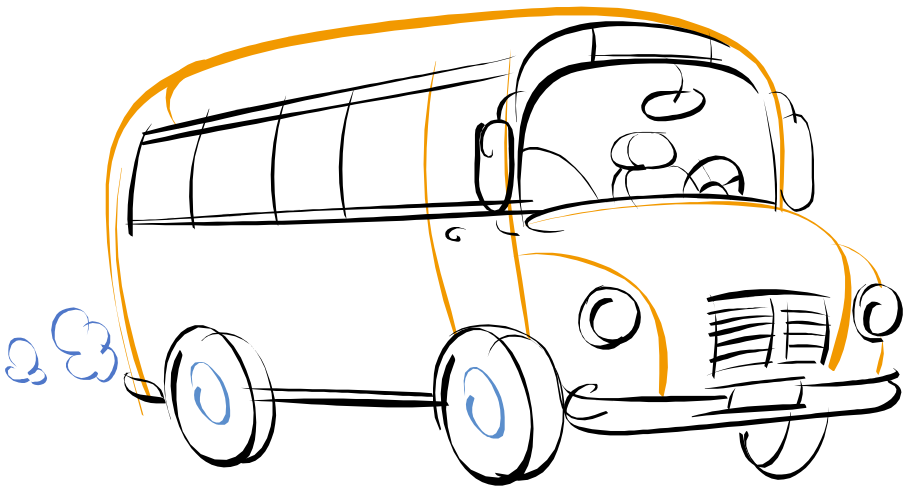
System bus

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

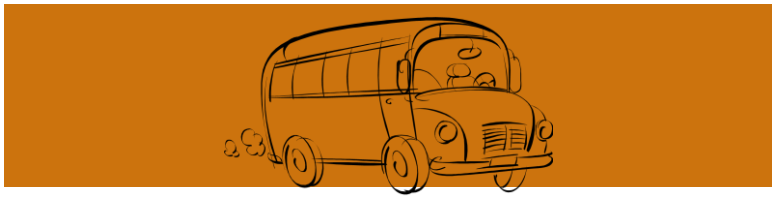
The most common computer interconnection structures are based on the use of one or more system buses

Data Bus

- Data lines that provide a path for moving data among system modules
- 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

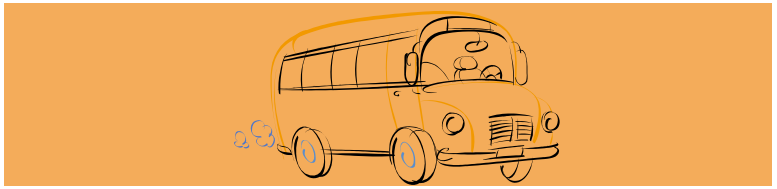


Address Bus



- 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

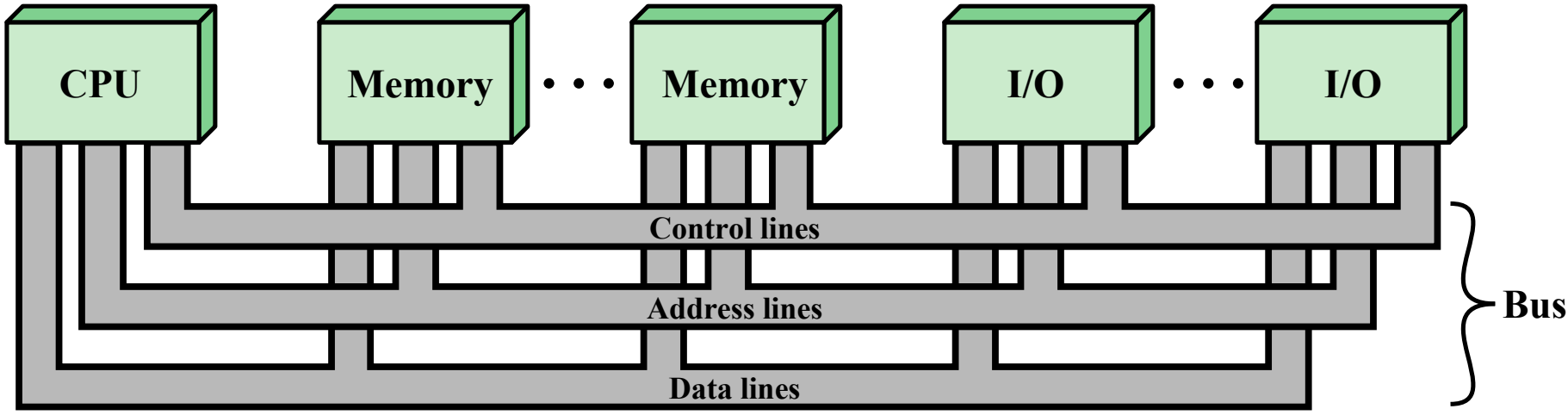
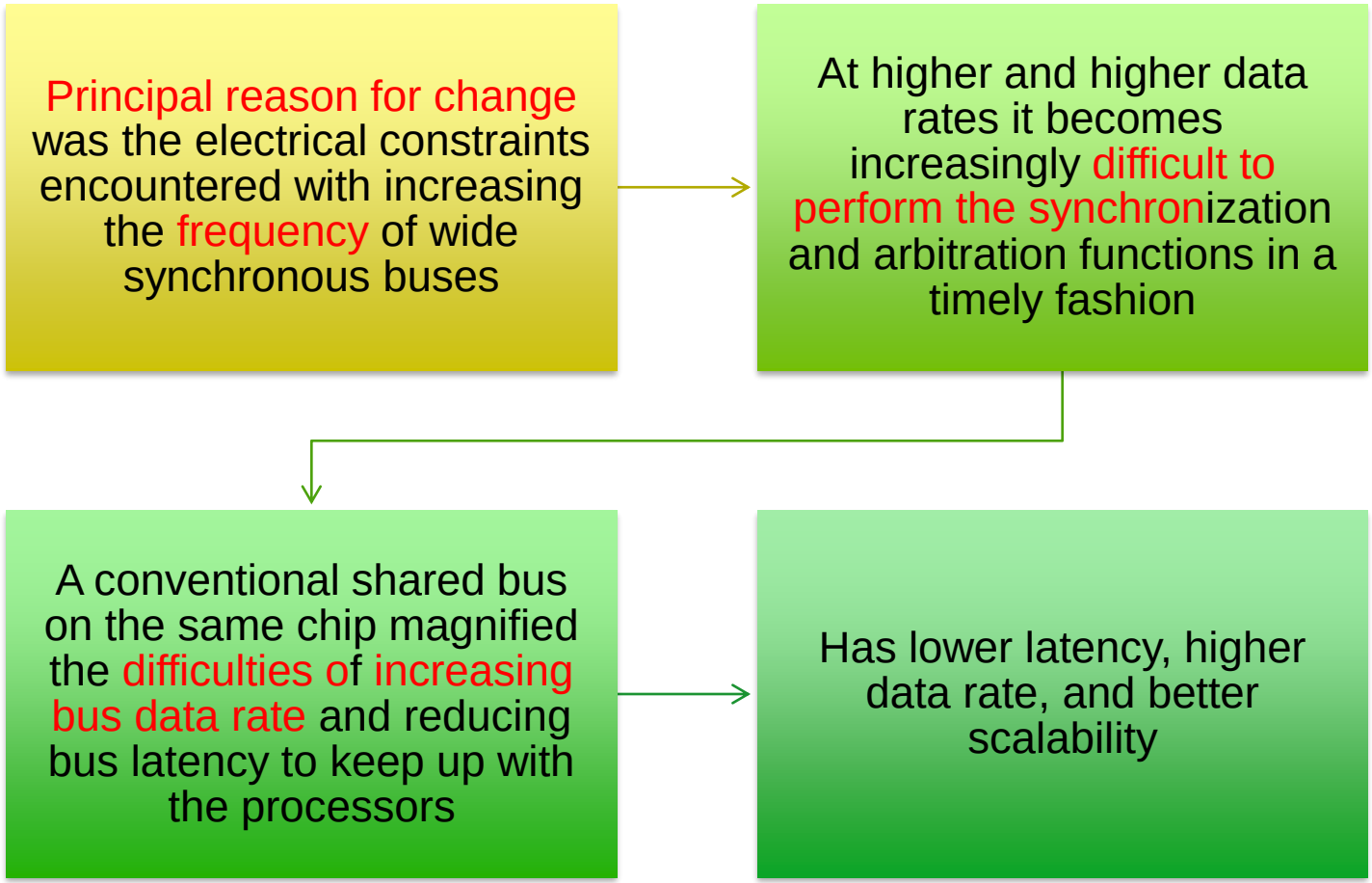


Figure 3.16 Bus Interconnection Scheme

- The shared bus architecture was the standard approach to interconnection between the processor and other components (memory, I/O, and so on) for decades.
- But contemporary systems increasingly rely on point-to-point interconnection rather than shared buses.

Point-to-Point Interconnect

- The shared bus architecture was the standard approach to interconnection between the processor and other components (memory, I/O, and so on) for decades.
- But Contemporary systems increasingly rely on point-to-point interconnection rather than shared buses.



Quick Path Interconnect

QPI

- Introduced in 2008
- has multiple direct connections

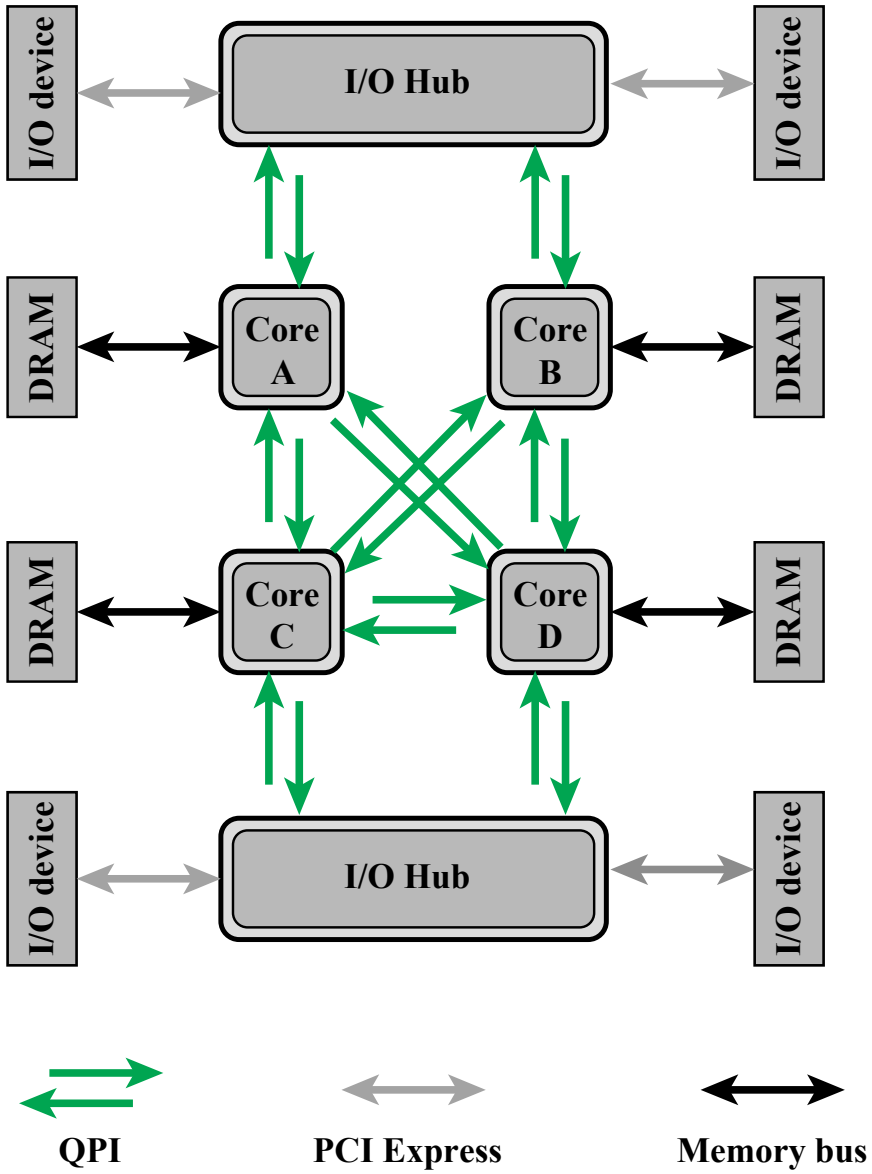
  - Direct pairwise connections to other components eliminating the need for arbitration found in shared transmission systems

- Layered protocol architecture

  - These processor level interconnects use a layered protocol architecture rather than the simple use of control signals found in shared bus arrangements

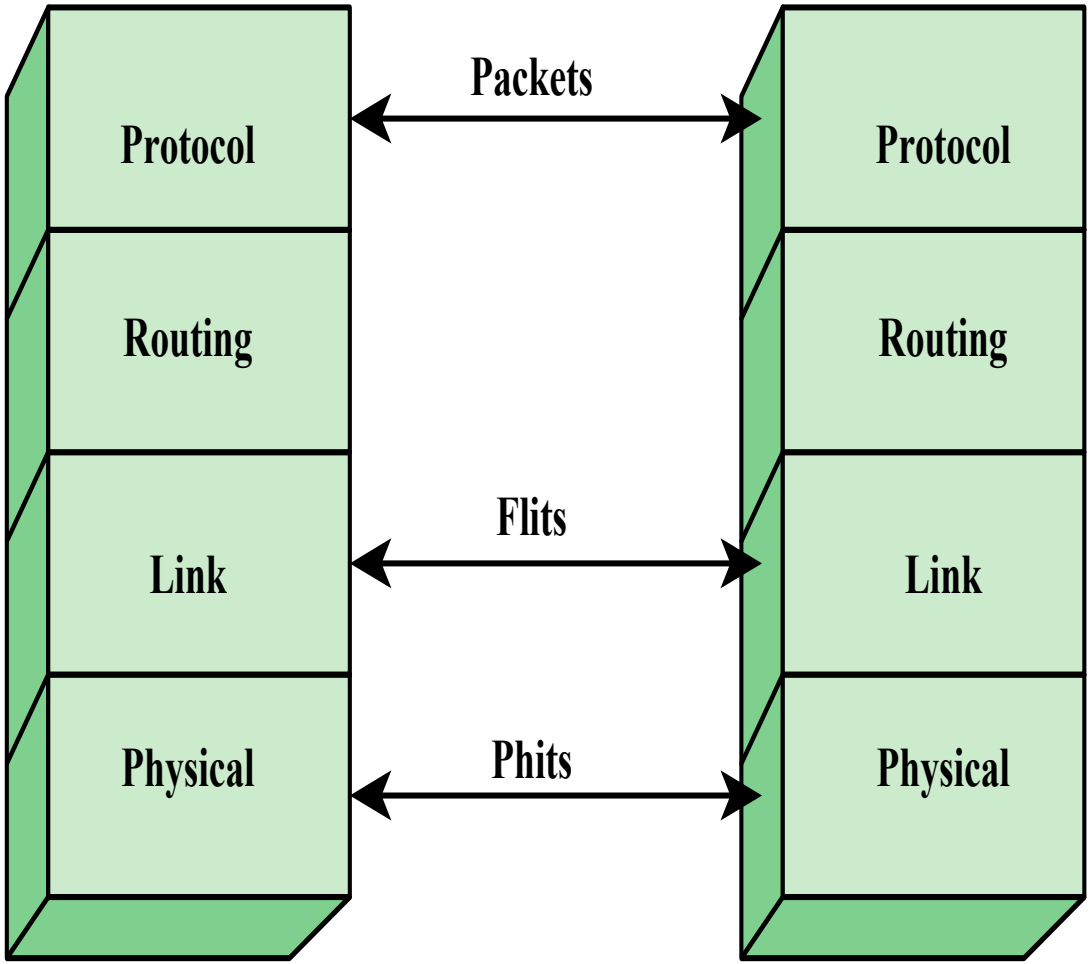
- Packetized data transfer

  - Data are sent as a sequence of packets each of which includes control headers and error control codes



- Figure 3.17 illustrates a typical use of QPI on a multi-core computer. The QPI links (indicated by the green arrow pairs in the figure) form a switching fabric that enables data to move throughout the network. Direct QPI connections can be established between each pair of core processors
- In addition, QPI is used to connect to an I/O module, called an I/O hub (IOH). The IOH acts as a switch directing traffic to and from I/O devices. Typically in newer systems, the link from the IOH to the I/O device controller uses an interconnect technology called PCI Express (PCIe), described later in this chapter. The IOH translates between the QPI protocols and formats and the PCIe protocols and formats

Figure 3.17 Multicore Configuration Using QPI



QPI is defined as a four-layer protocol architecture

Figure 3.18 QPI Layers

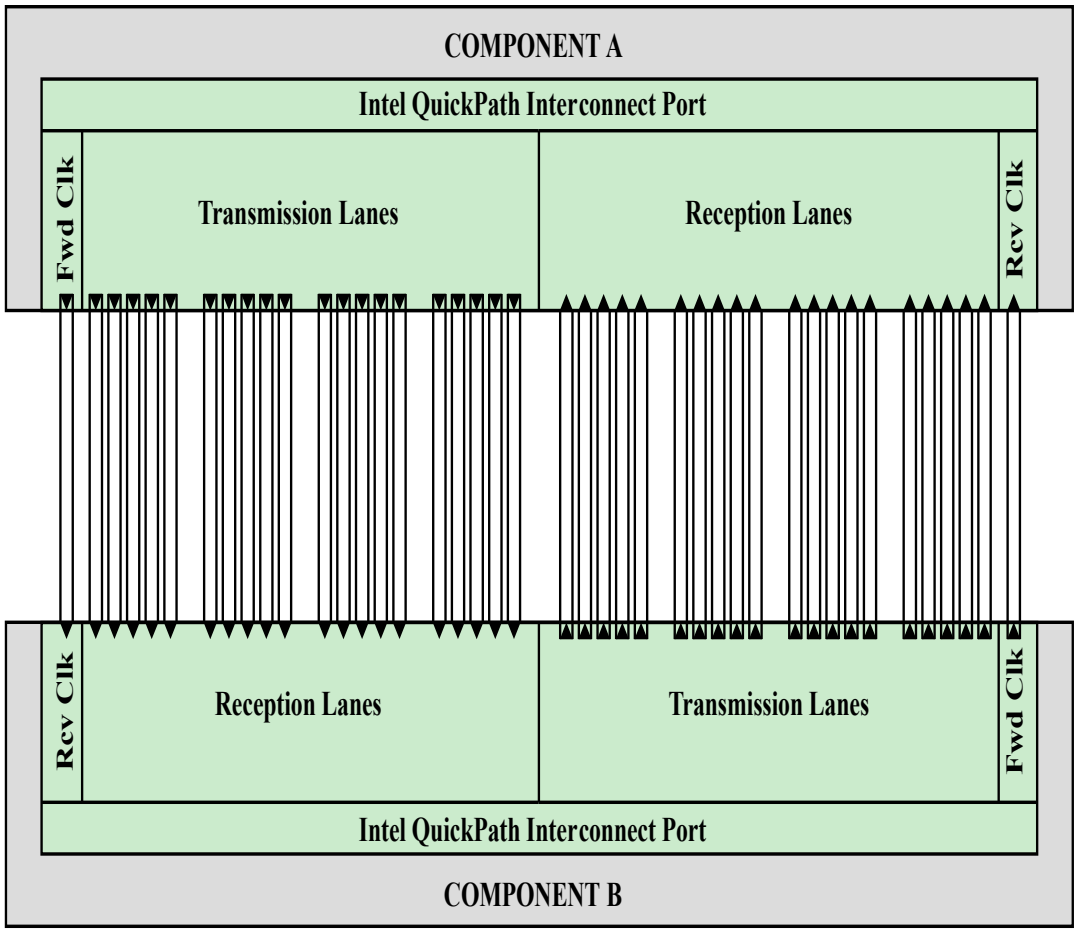


Figure 3.19 Physical Interface of the Intel QPI Interconnect

- Figure 3.19 shows the physical architecture of a QPI port. The QPI port consists of 84 individual links grouped as follows. Each data path consists of a pair of wires that transmits data one bit at a time; the pair is referred to as a lane. There are 20 data lanes in each direction (transmit and receive), plus a clock lane in each direction.
- Thus, QPI is capable of transmitting 20 bits in parallel in each direction. The 20-bit unit is referred to as a phit. Typical signaling speeds of the link in current products calls for operation at 6.4 GT/s (transfers per second).
- At 20 bits per transfer, that adds up to 16 GB/s, and since QPI links involve dedicated bidirectional pairs, the total capacity is 32 GB/s.

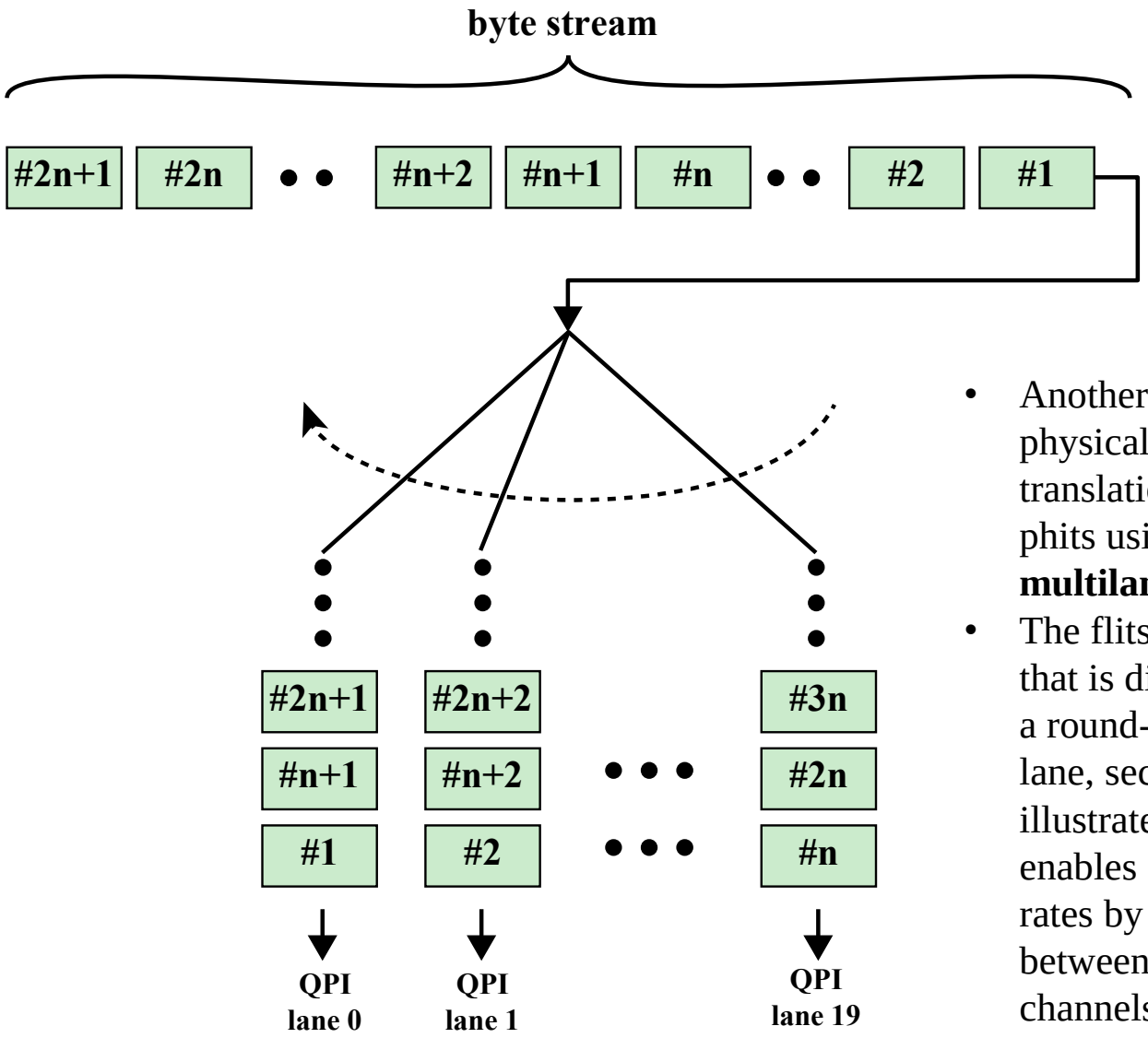


Figure 3.20 QPI Multilane Distribution

- Another function performed by the physical layer is that it manages the translation between 80-bit flits and 20-bit phits using a technique known as **multilane distribution.**
- The flits can be considered as a bit stream that is distributed across the data lanes in a round-robin fashion (first bit to first lane, second bit to second lane, etc.), as illustrated in Figure 3.20. This approach enables QPI to achieve very high data rates by implementing the physical link between two ports as multiple parallel channels.

QPI Link Layer

- Performs two key functions: *flow control* and *error control*
  - Operate on the level of the flit (flow control unit)
  - Each flit consists of a 72-bit message payload and an 8-bit error control code called a *cyclic redundancy check* (CRC)

- Flow control function
  - Needed to ensure that a sending QPI entity does not overwhelm a receiving QPI entity by sending data faster than the receiver can process the data and clear buffers for more incoming data

Error control function

  - Detects and recovers from bit errors, and so isolates higher layers from experiencing bit errors

QPI Routing and Protocol Layers

Routing Layer

- Used to determine the course that a packet will traverse across the available system interconnects
- Defined by firmware and describe the possible paths that a packet can follow

Protocol Layer

- Packet is defined as the unit of transfer
- One key function performed at this level is a cache coherency protocol which deals with making sure that main memory values held in multiple caches are consistent
- A typical data packet payload is a block of data being sent to or from a cache

Peripheral Component Interconnect (PCI)

- A popular high bandwidth, processor independent bus that can function as a mezzanine or peripheral bus
- Delivers better system performance for high speed I/O subsystems
- PCI Special Interest Group (SIG)
  - Created to develop further and maintain the compatibility of the PCI specifications

– PCI Express (PCIe)
  - Point-to-point interconnect scheme intended to replace bus-based schemes such as PCI
  - Key requirement is high capacity to support the needs of higher data rate I/O devices, such as Gigabit Ethernet
  - Another requirement deals with the need to support time dependent data streams

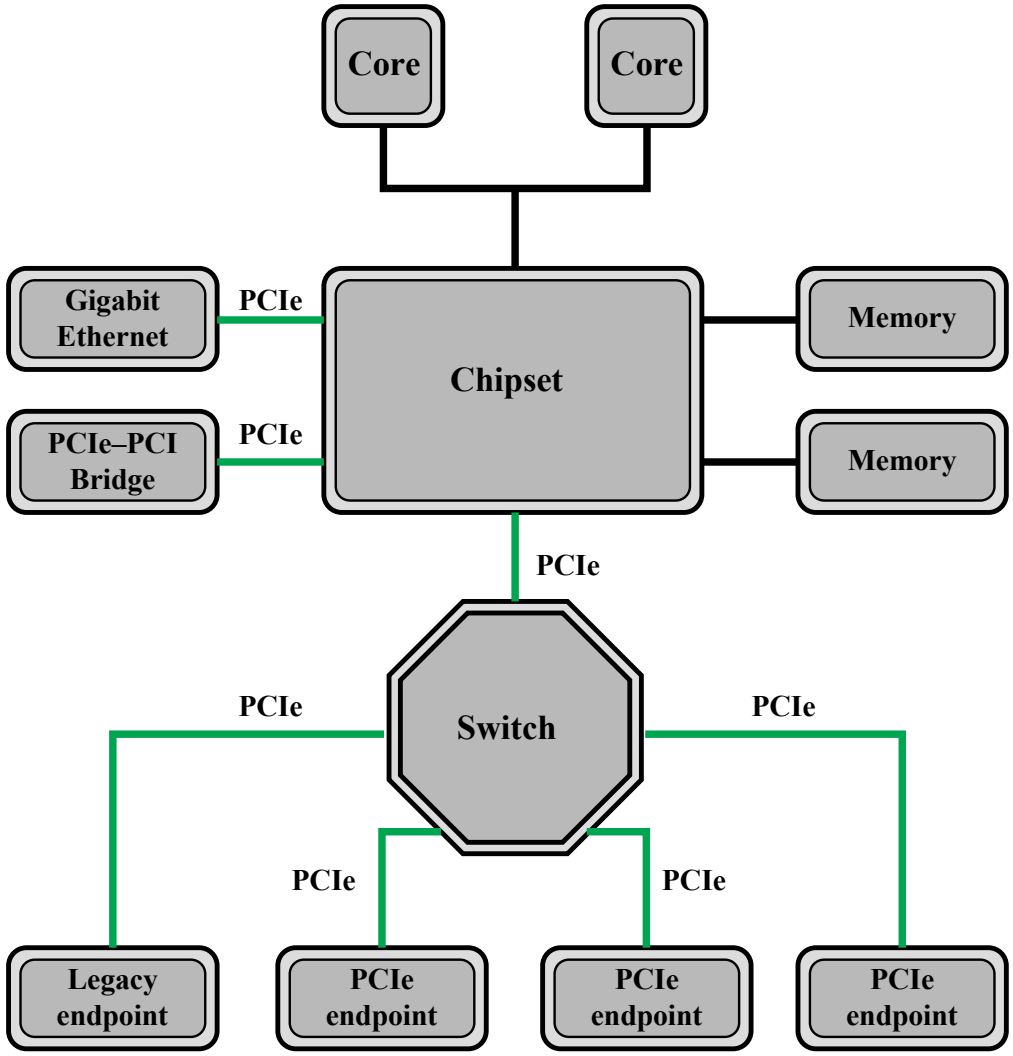


Figure 3.21 Typical Configuration Using PCIe

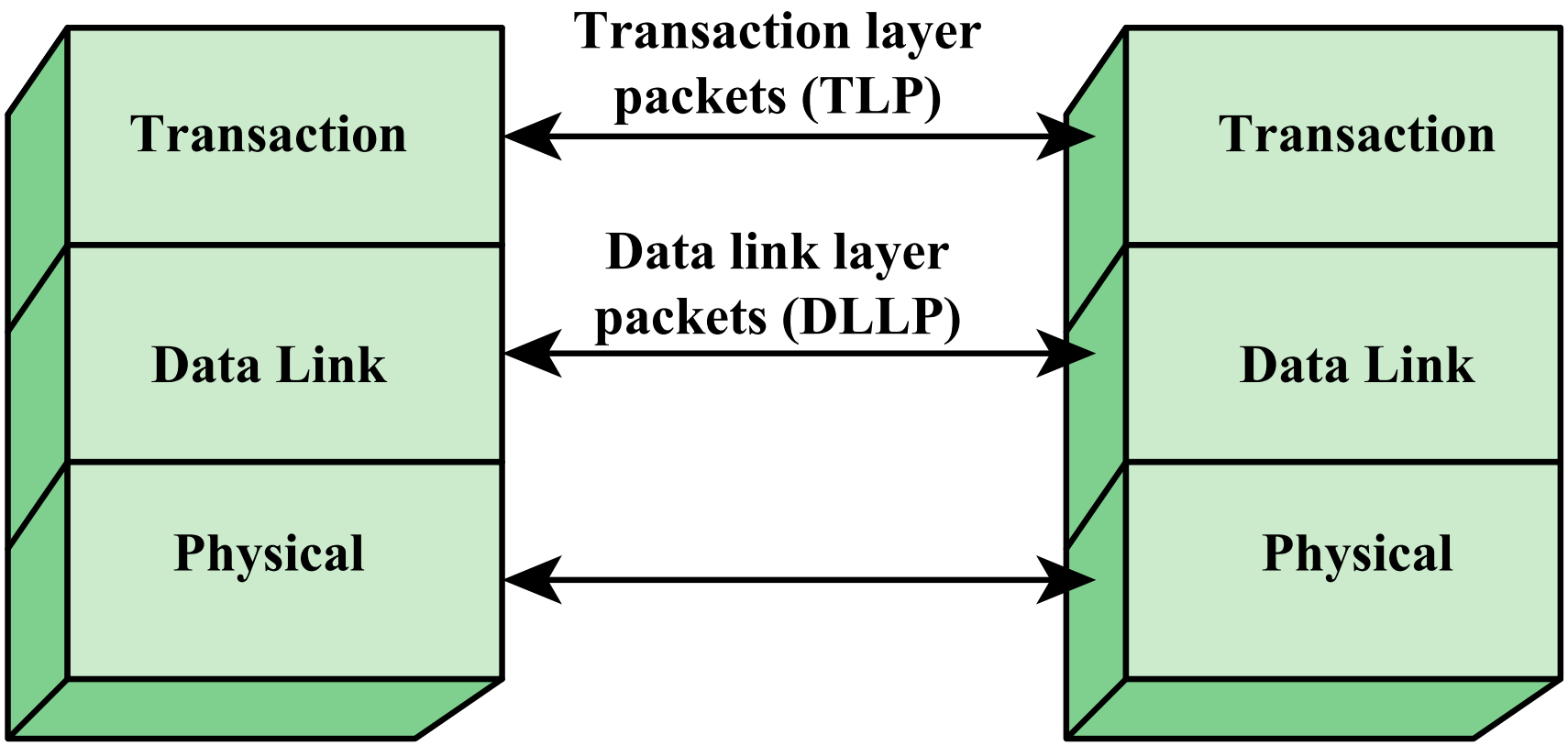


Figure 3.22 PCIe Protocol Layers

Figure 3.23

- As with QPI, PCIe uses a multilane distribution technique.
- Figure 3.23 shows an example for a PCIe port consisting of four lanes. Data are distributed to the four lanes 1 byte at a time using a simple round-robin scheme

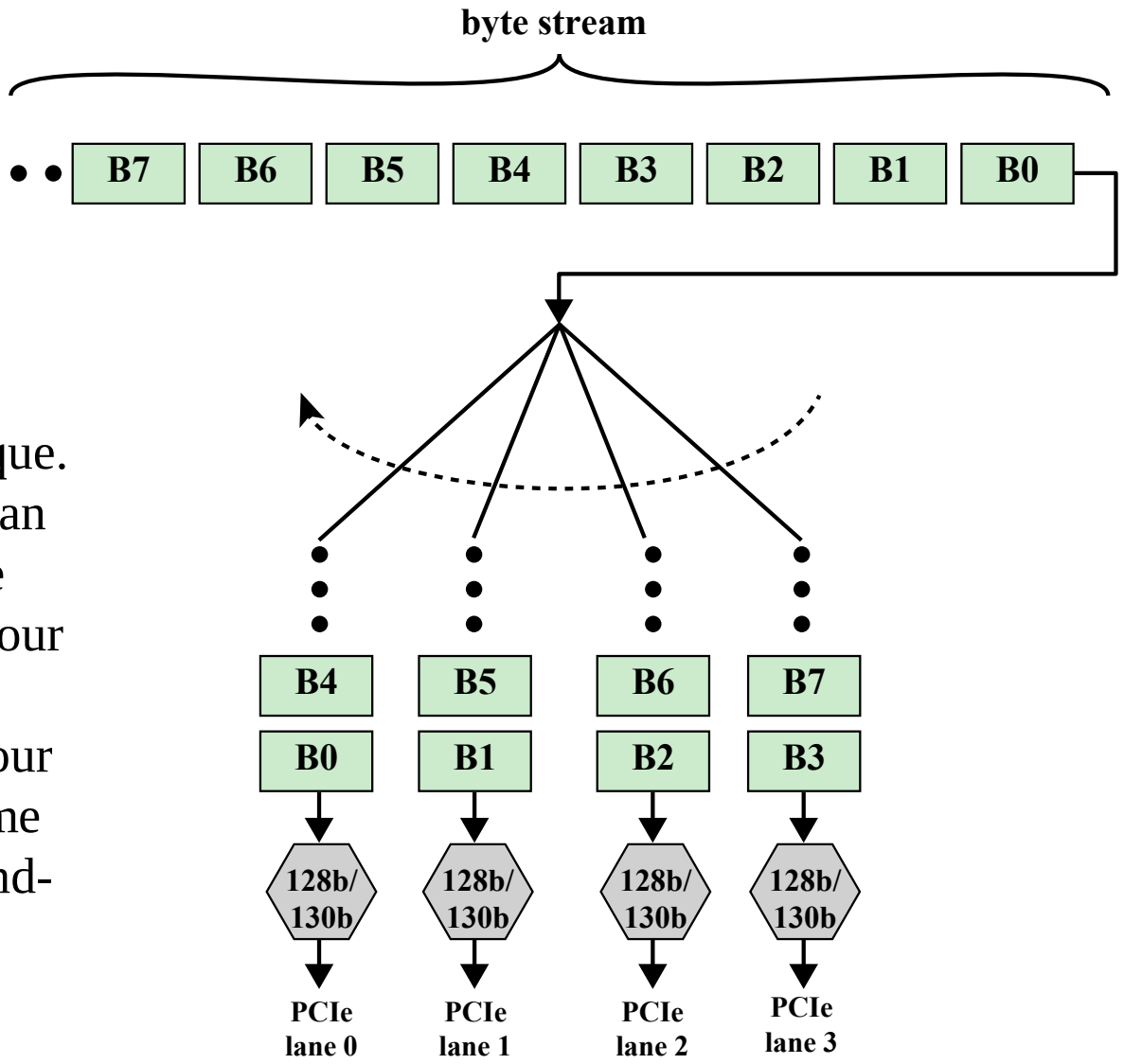


Figure 3.23 PCIe Multilane Distribution

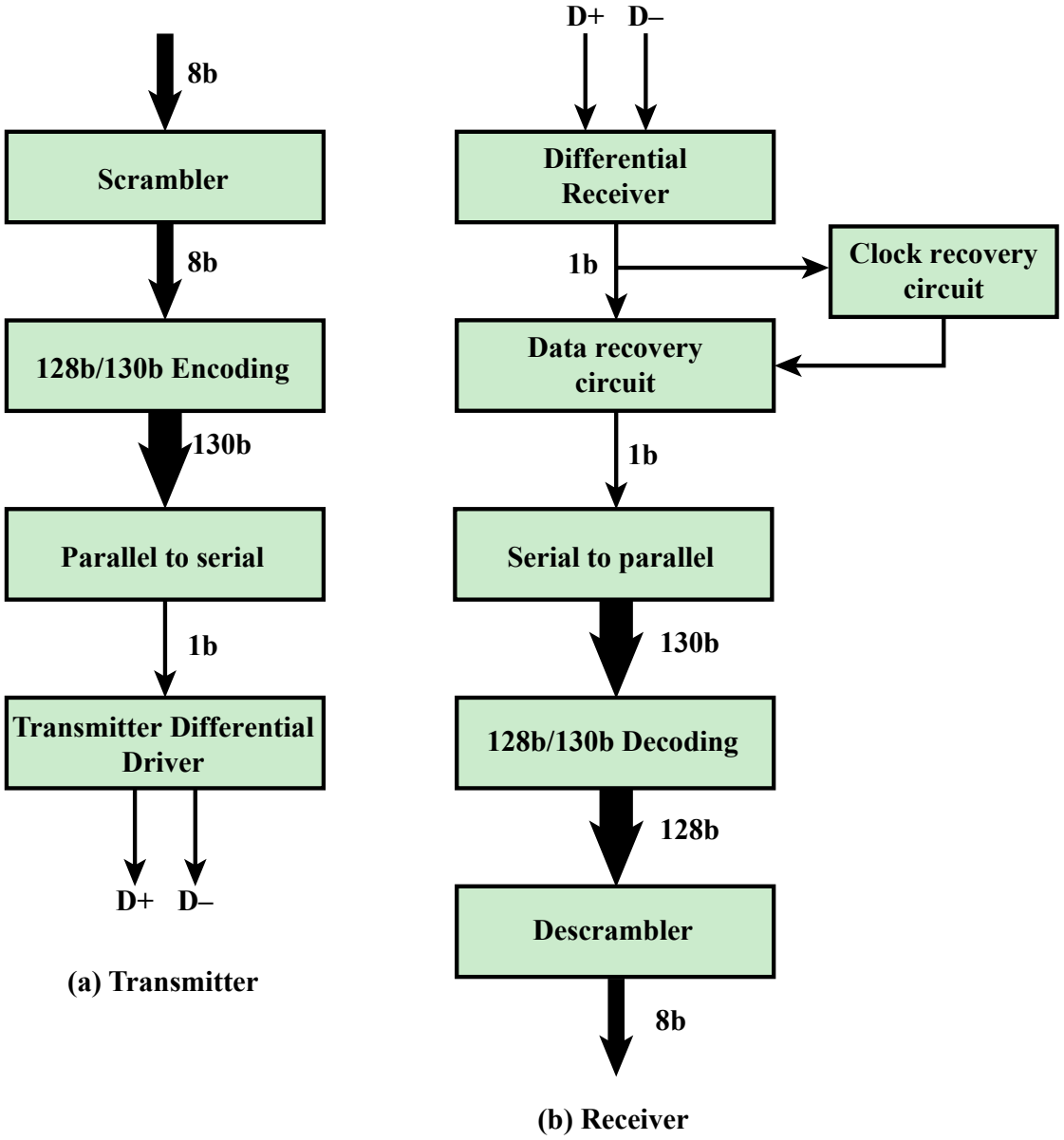


Figure 3.24 PCIe Transmit and Receive Block Diagrams

PCIe Transaction Layer (TL)

- Receives read and write requests from the software above the TL and creates request packets for transmission to a destination via the link layer
- Most transactions use a *split transaction* technique
  - A request packet is sent out by a source PCIe device which then waits for a response called a *completion* packet
- TL messages and some write transactions are posted transactions (meaning that no response is expected)
- TL packet format supports 32-bit memory addressing and extended 64-bit memory addressing

The TL supports four address spaces:

- Memory
  - The memory space includes system main memory and PCIe I/O devices
  - Certain ranges of memory addresses map into I/O devices
- Configuration
  - This address space enables the TL to read/write configuration registers associated with I/O devices
- I/O
  - This address space is used for legacy PCI devices, with reserved address ranges used to address legacy I/O devices
- Message
  - This address space is for control signals related to interrupts, error handling, and power management

Table 3.2
PCIe TLP Transaction Types

Address Space	TLP Type	Purpose
Memory	Memory Read Request	Transfer data to or from a location in the system memory map.
	Memory Read Lock Request	
	Memory Write Request	
I/O	I/O Read Request	Transfer data to or from a location in the system memory map for legacy devices.
	I/O Write Request	
Configuration	Config Type 0 Read Request	Transfer data to or from a location in the configuration space of a PCIe device.
	Config Type 0 Write Request	
	Config Type 1 Read Request	
	Config Type 1 Write Request	
Message	Message Request	Provides in-band messaging and event reporting.
	Message Request with Data	
Memory, I/O, Configuration	Completion	Returned for certain requests.
	Completion with Data	
	Completion Locked	
	Completion Locked with Data	

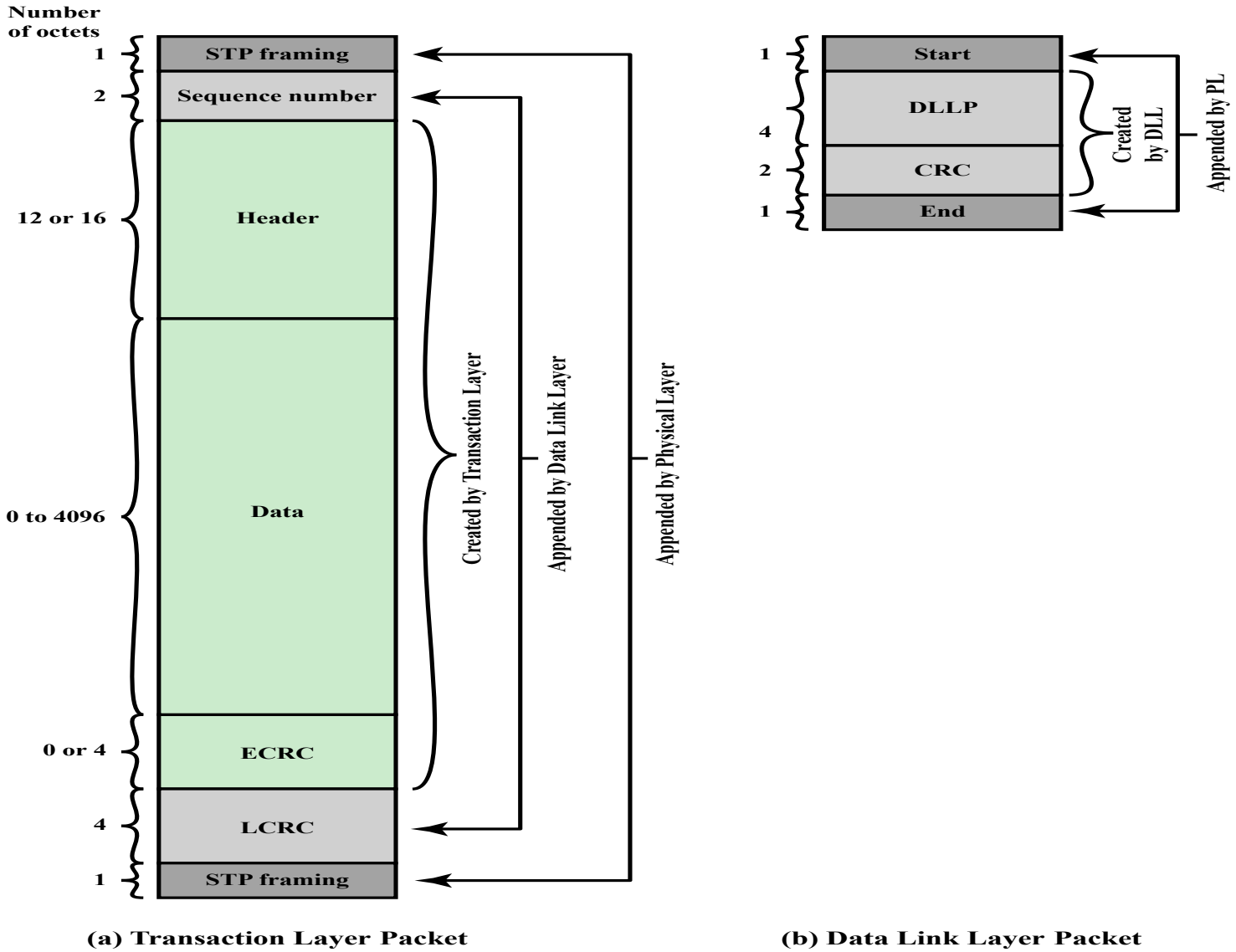


Figure 3.25 PCIe Protocol Data Unit Format

Summary

Chapter 3

A Top-Level View of Computer Function and Interconnection

Computer components

Computer function
Instruction fetch and execute
Interrupts
I/O function

Interconnection structures

Bus interconnection

Point-to-point interconnect
QPI physical layer
QPI link layer
QPI routing layer
QPI protocol layer

PCI express
PCI physical and logical architecture
PCIe physical layer
PCIe transaction layer
PCIe data link layer