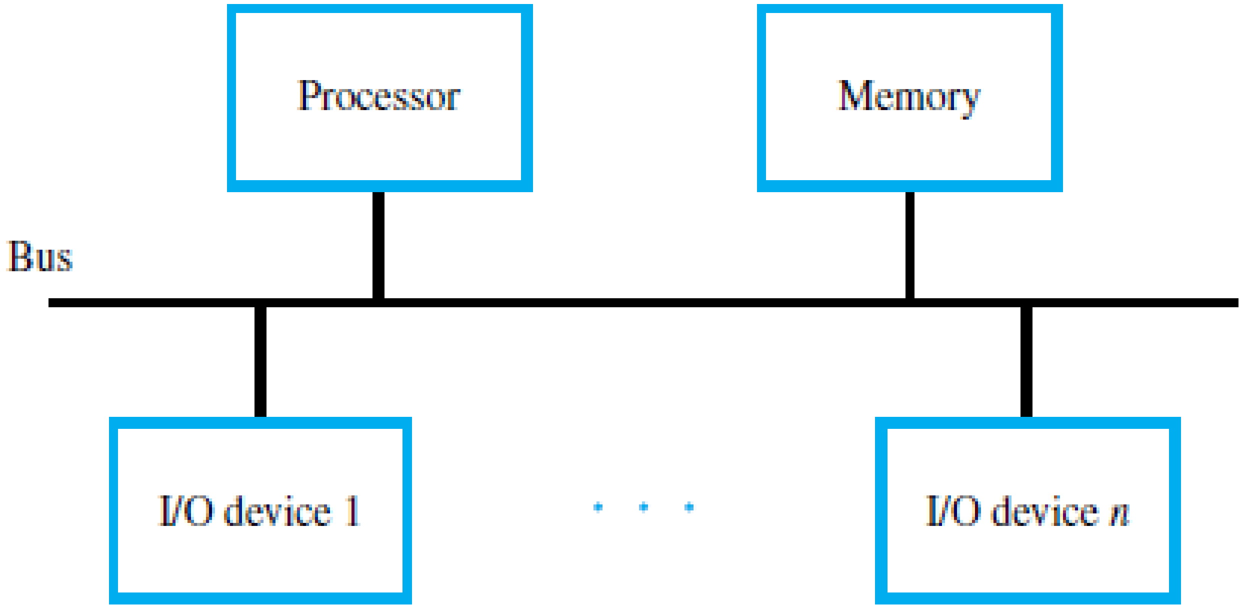
UNIT-4

Input/Output Organization

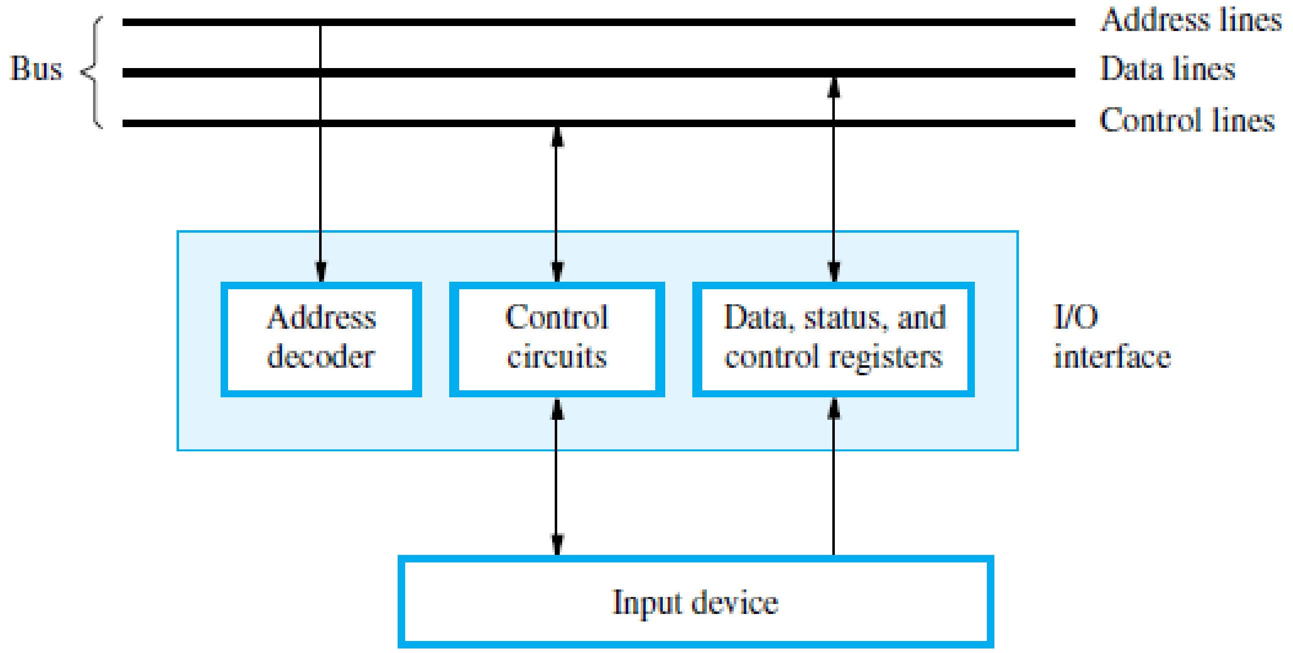
Bus Structure

A single-bus structure



- Is a simple structure that implements the interconnection Network .
- Only one source/destination pair of units can use this bus to transfer data at any one time.

I/O interface for an input device.



- The bus consists of three sets of lines used to carry address, data, and control signals.

- I/O device interfaces are connected to these lines, for an input device.

- Each I/O device is assigned a unique set of addresses for the registers in its interface.

- When the processor places a particular address on the address lines, it is examined by the address decoders of all devices on the bus.
- The device that recognizes this address responds to the commands issued on the control lines.
- The processor uses the control lines to request either a Read or a Write operation, and the requested data are transferred over the data lines.
- Any machine instruction that can access memory can be used to transfer data to or from an I/O device.

For example,

- if the input device is a keyboard and if DATAIN is its data register, the instruction
  Load R2, *DATAIN* - reads the data from DATAIN and stores them into processor register R2.

- Similarly, The instruction
  Store R2, DATAOUT - sends the contents of register R2 to location DATAOUT which may be the data register of a display device interface.

- The status and control registers contain information relevant to the operation of the I/O device.

- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device’s interface circuit.

Bus Operation

- A bus requires a set of rules, often called a bus protocol, that govern how the bus is used by various devices.
- The bus protocol determines when a device may place information on the bus, when it may load the data on the bus into one of its registers, and so on.
- These rules are implemented by control signals that indicate what and when actions are to be taken.
- One control line, usually labeled $R/\overline{W}$, specifies whether a Read or a Write operation is to be performed.
- When several data sizes are possible, such as byte, half word, or word, the required size is indicated by other control lines.
- The bus control lines also carry timing information.

- They specify the times at which the processor and the I/O devices may place data on or receive data from the data lines.

- A variety of schemes have been devised for the timing of data transfers over a bus. These can be broadly classified as either synchronous or asynchronous schemes.

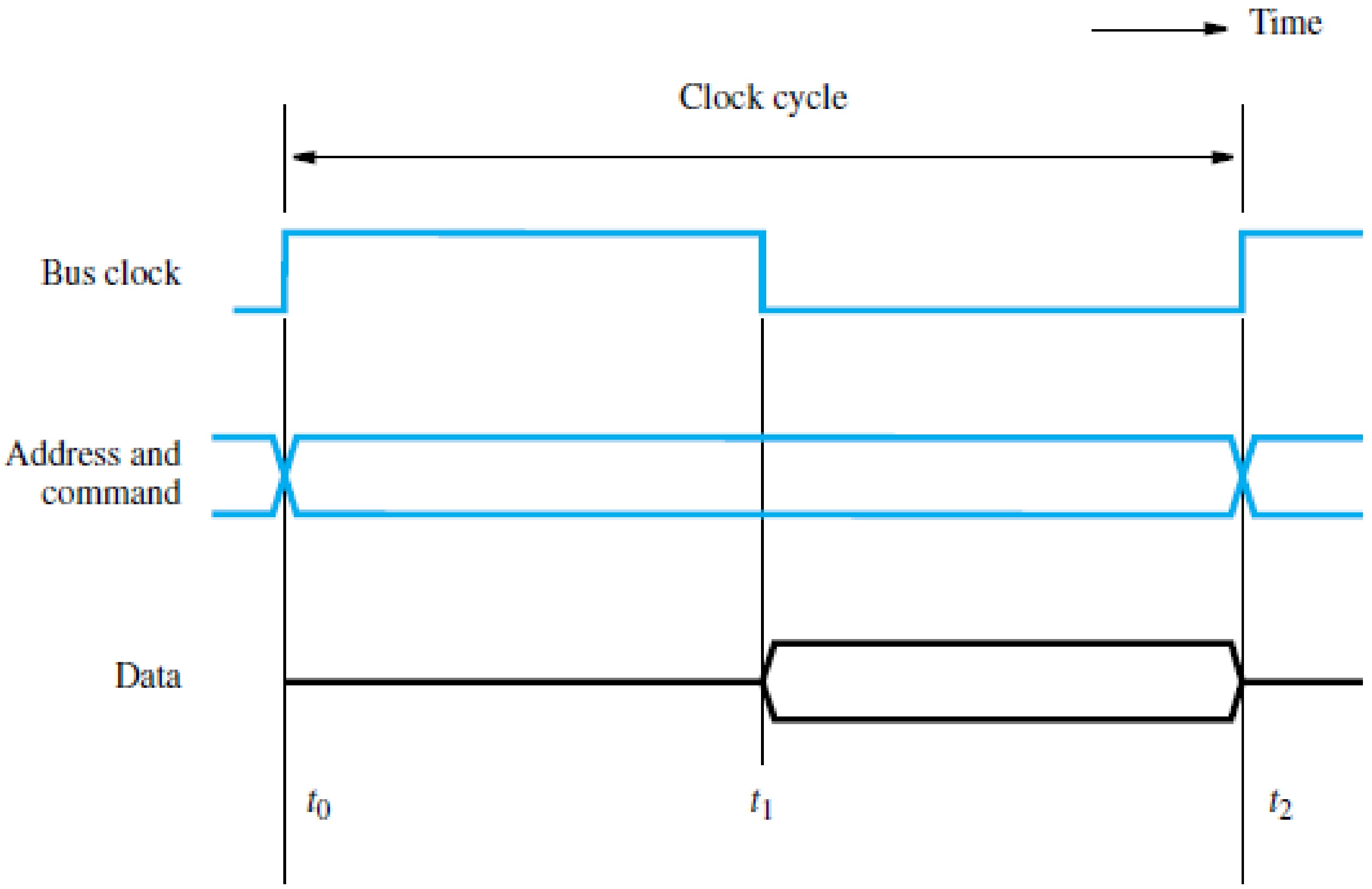
- In any data transfer operation, one device plays the role of a master. This is the device that initiates data transfers by issuing Read or Write commands on the bus.

- Normally, the processor acts as the master, but other devices may also become masters .

- The device addressed by the master is referred to as a slave.

Synchronous Bus

- On a synchronous bus, all devices derive timing information from a control line called the bus clock.



Timing of an input transfer on a synchronous bus.

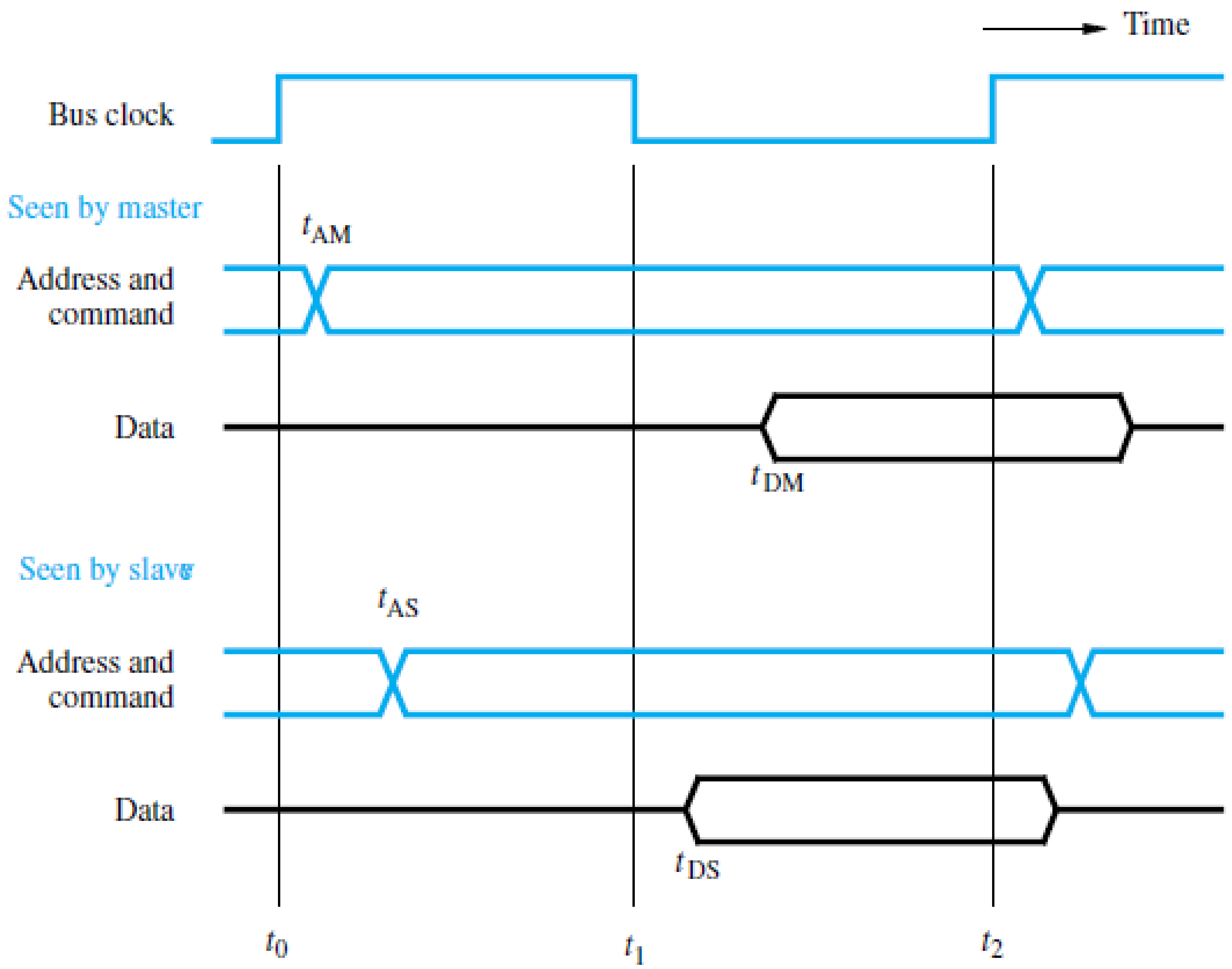
Following is signal events during an input (Read) operation.

- At time t0, the master places the device address on the address lines and sends a command on the control lines indicating a Read operation. The command may also specify the length of the operand to be read.
- At the end of the clock cycle, at time t_2, the master loads the data on the data lines into one of its registers.

A similar procedure is followed for a output(Write)operation.

- The master places the output data on the data lines when it transmits the address and command information.
- At time t_2, the addressed device loads the data into its data register.

The exact times at which signals change state are somewhat different from those shown, because of propagation delays on bus wires and in the circuits of the devices.

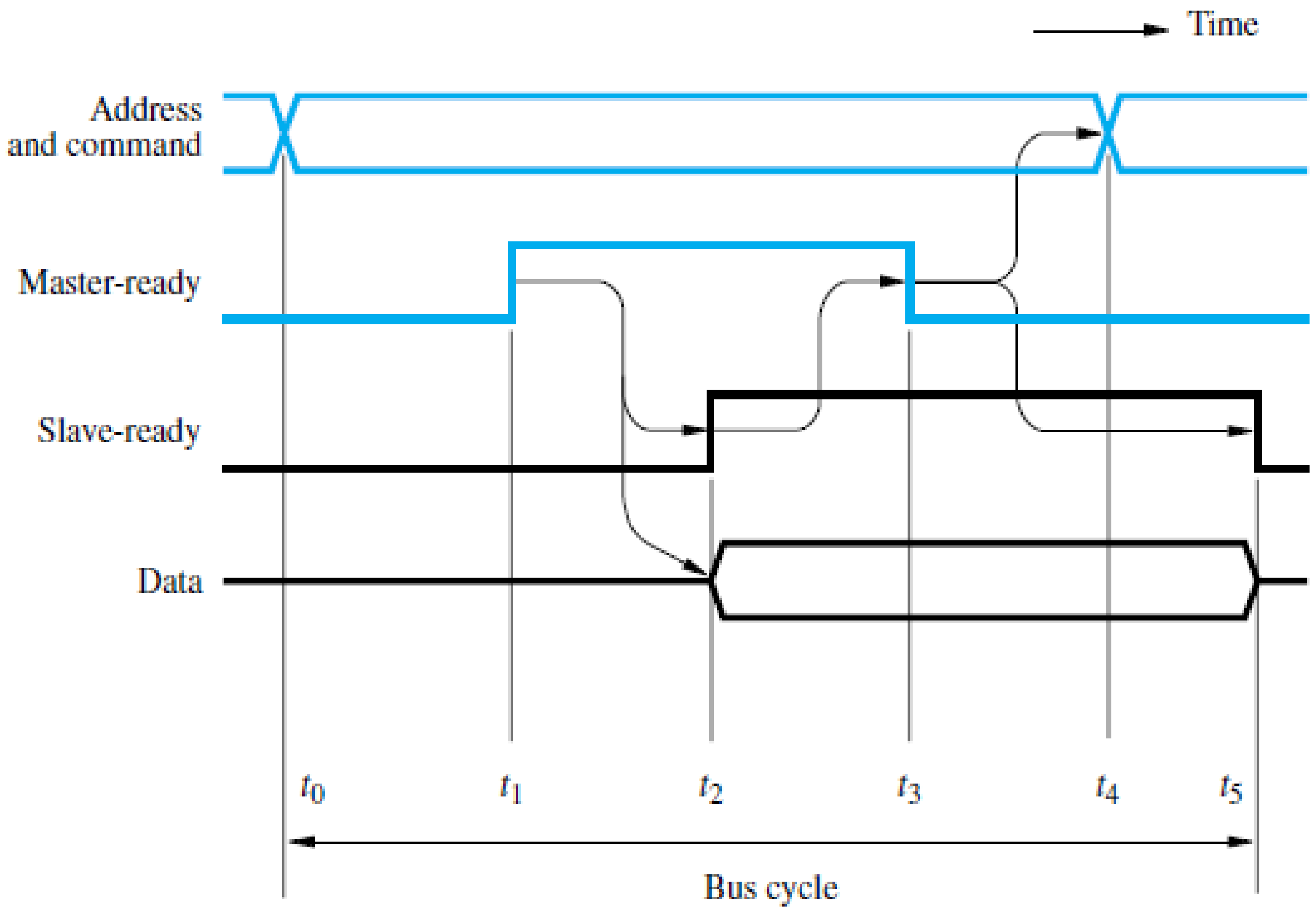


A detailed timing diagram for the input transfer

- The master sends the address and command signals on the rising edge of the clock at the beginning of the clock cycle (at t_0).
- However, these signals do not actually appear on the bus until t_{AM}, largely due to the delay in the electronic circuit output from the master to the bus lines. A short while later, at t_{AS}, the signals reach the slave.
- The slave decodes the address, and at t1 sends the requested data.
- Here again, the data signals do not appear on the bus until t_{DS}. They travel toward the master and arrive at t_{DM}.
- At t_2, the master loads the data into its register. Hence the period $t_2 - t_{DM}$ must be greater than the setup time of that register.
- The data must continue to be valid after t_2 for a period equal to the hold time requirement of the register

Asynchronous Bus

- Approach that does not use a clock signal at all.
- Data transfers on a bus is based on the use of a handshake protocol between the master and the slave.
- A handshake is an exchange of command and response signals between the master and the slave
- A control line called Master-ready is asserted by the master to indicate that it is ready to start a data transfer.
- The Slave responds by asserting Slave-ready.



Handshake control of data transfer during an input operation

Timing of an input data transfer using the handshake protocol depicts the following sequence of events:

t_0—The master places the address and command information on the bus, and all devices on the bus decode this information.

t_1—The master sets the Master-ready line to 1 to inform the devices that the address and command information is ready.

t_3—The Slave-ready signal arrives at the master, indicating that the input data are available on the bus.

t_4—The master removes the address and command information from the bus.

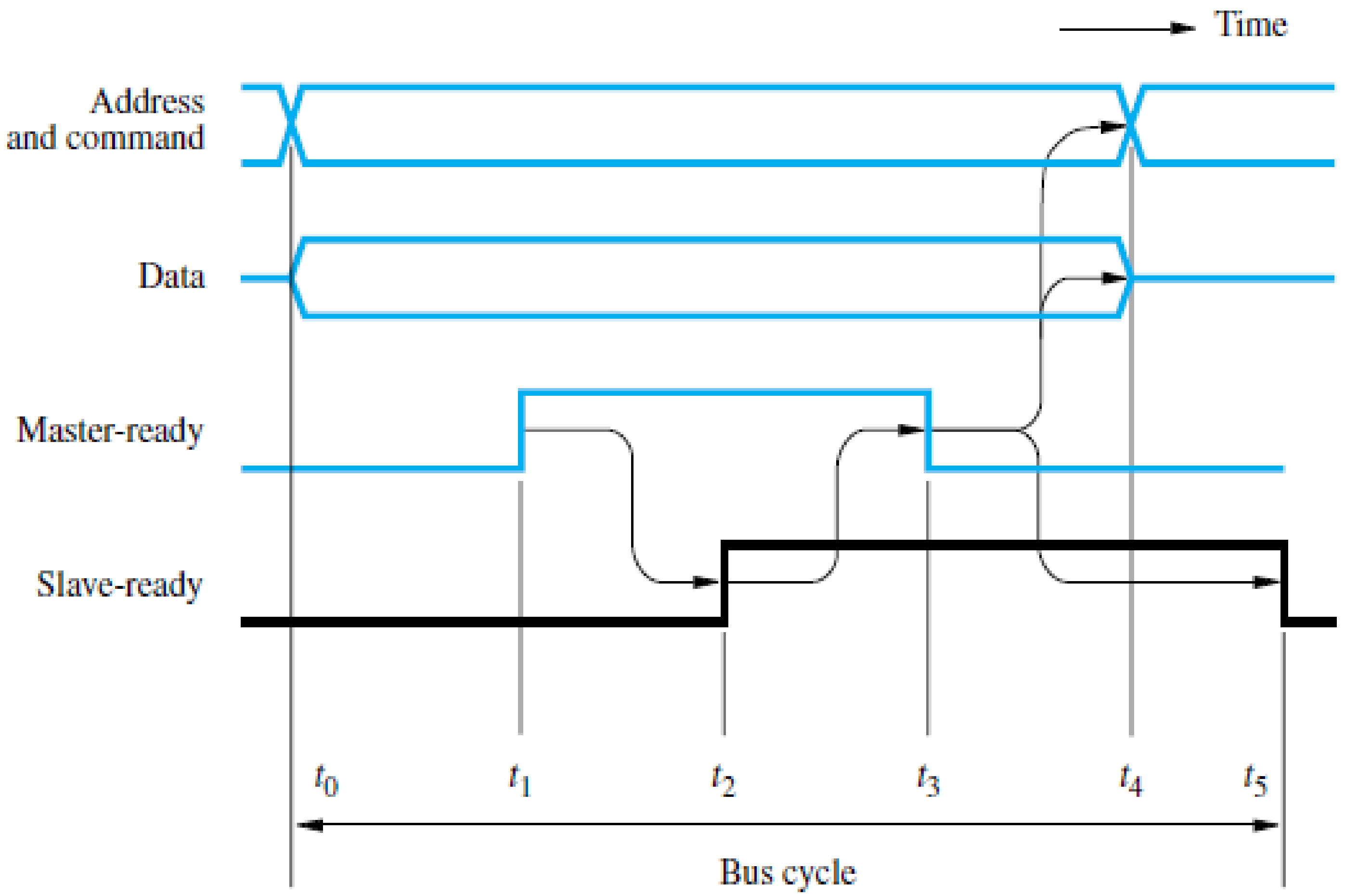
t_5—When the device interface receives the 1-to-0 transition of the Master-ready signal, it removes the data and the Slave-ready signal from the bus. This completes the input transfer.

The timing for an output operation is essentially the same as for an input operation.

• In this case, the master places the output data on the data lines at the same time that it transmits the address and command information.

• The selected slave loads the data into its data register when it receives the Master-ready signal and indicates that it has done so by setting the Slave-ready signal to 1.

•The remainder of the cycle is similar to the input operation.



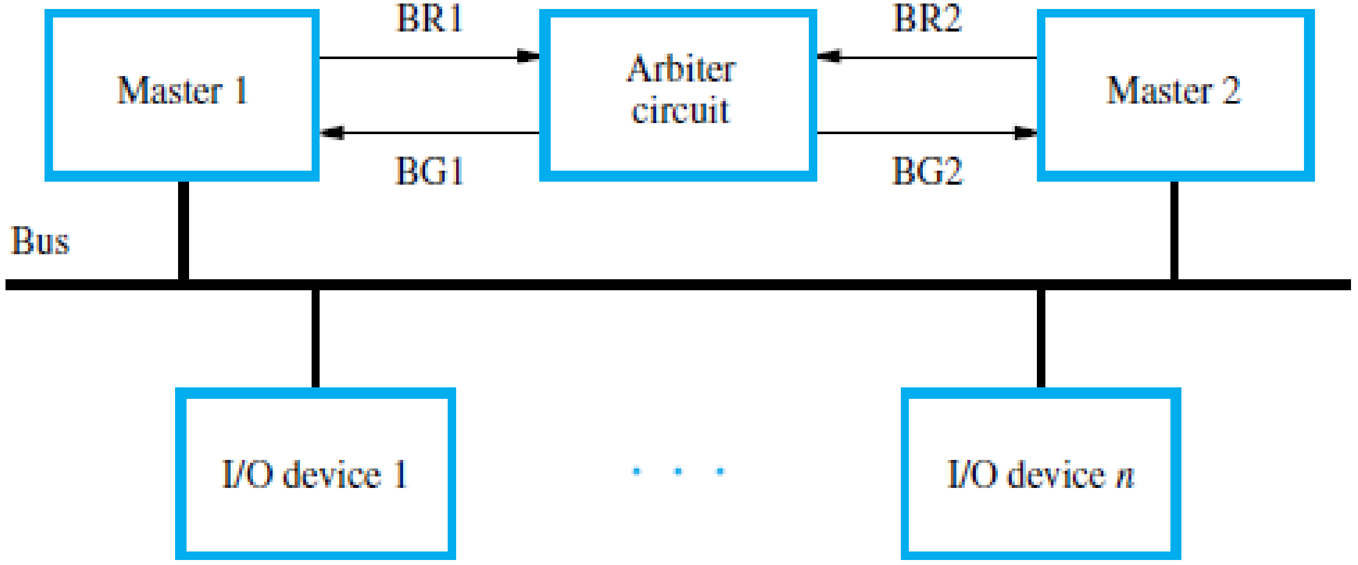
Handshake control of data transfer during an output operation.

Arbitration

- There are occasions when two or more entities contend for the use of a single resource in a computer system.
  For example, two devices may need to access a given slave at the same time.
- In such cases, it is necessary to decide which device will access the slave first.
- The decision is usually made in an arbitration process performed by an arbiter circuit.
- The arbitration process starts by each device sending a request to use the shared resource.
- The arbiter associates priorities with individual requests. If it receives two requests at the same time, it grants the use of the slave to the device having the higher priority first.

To illustrate the arbitration process, we consider the case where a single bus is the shared resource.

- The device that initiates data transfer requests on the bus is the bus master.
- Since the bus is a single shared facility, it is essential to provide orderly access to it by the bus masters.
- A device that wishes to use the bus sends a request to the arbiter.
- When multiple requests arrive at the same time, the arbiter selects one request and grants the bus to the corresponding device.
- For some devices, a delay in gaining access to the bus may lead to an error. Such devices must be given high priority.
- If there is no particular urgency among requests, the arbiter may grant the bus using a simple round-robin scheme.



Bus arbitration.

- There are two Bus-request lines, BR1and BR2, and two Bus-grant lines, BG1and BG2, connecting the arbiter to the masters.

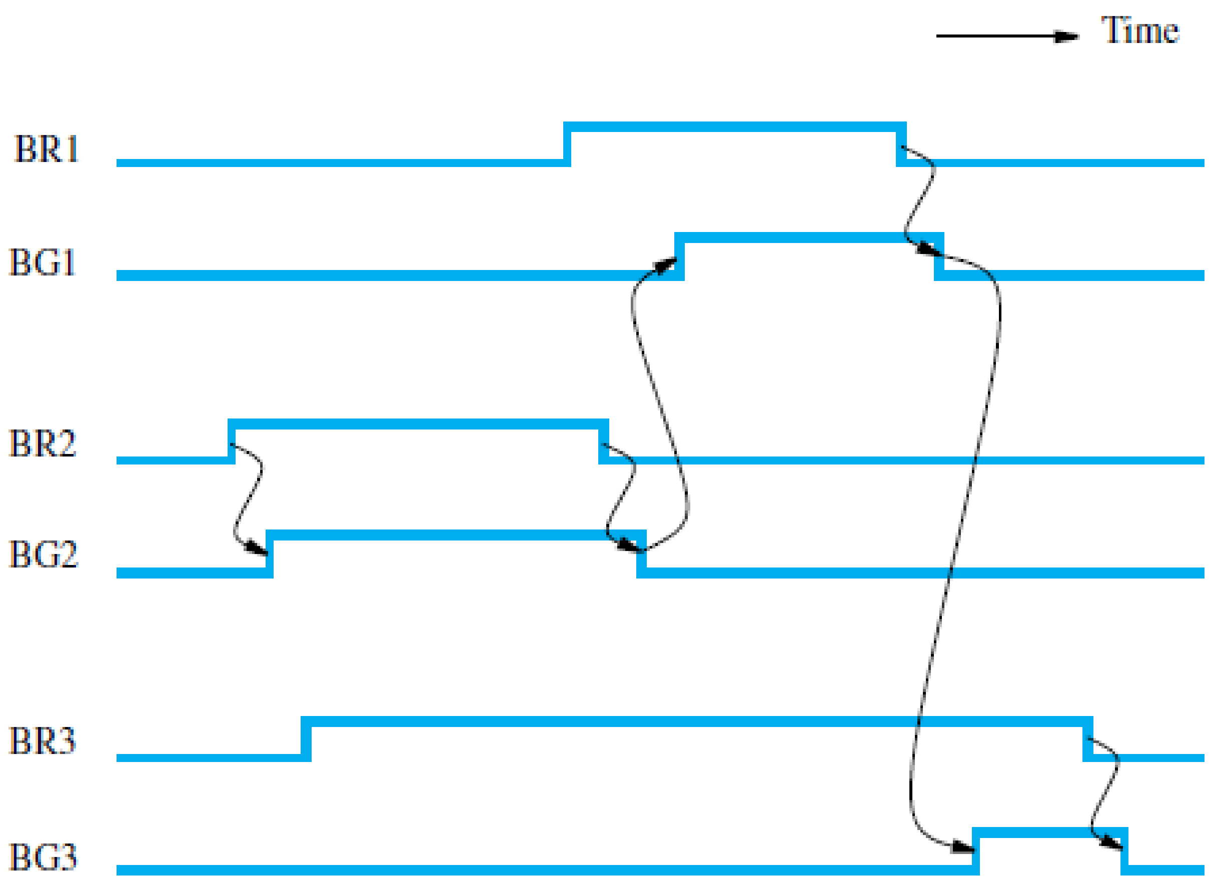
- A master requests use of the bus by activating its Bus-request line.

- If a single Bus-request is activated, the arbiter activates the corresponding Bus-grant.

- This indicates to the selected master that it may now use the bus for transferring data.

- When the transfer is completed, that master deactivates its Bus-request, and the arbiter deactivates its Bus-grant.

Assume that master 1 has the highest priority, followed by the others in increasing numerical order. Figure below illustrates a possible sequence of events for the case of three masters.



Granting use of the bus based on priorities.

- Master 2 sends a request to use the bus first.
- Since there are no other requests, the arbiter grants the bus to this master by asserting BG2.
- When master 2 completes its data transfer operation, it releases the bus by deactivating BR2.
- By that time, both masters 1 and 3 have activated their request lines.
- Since device 1 has a higher priority, the arbiter activates BG1 after it deactivates BG2, thus granting the bus to master 1.
- Later, when master 1 releases the bus by deactivating BR1, the arbiter deactivates BG1 and activates BG3 to grant the bus to master 3.
- Note that the bus is granted to master 1 before master 3 even though master 3 activated its request line before master 1.

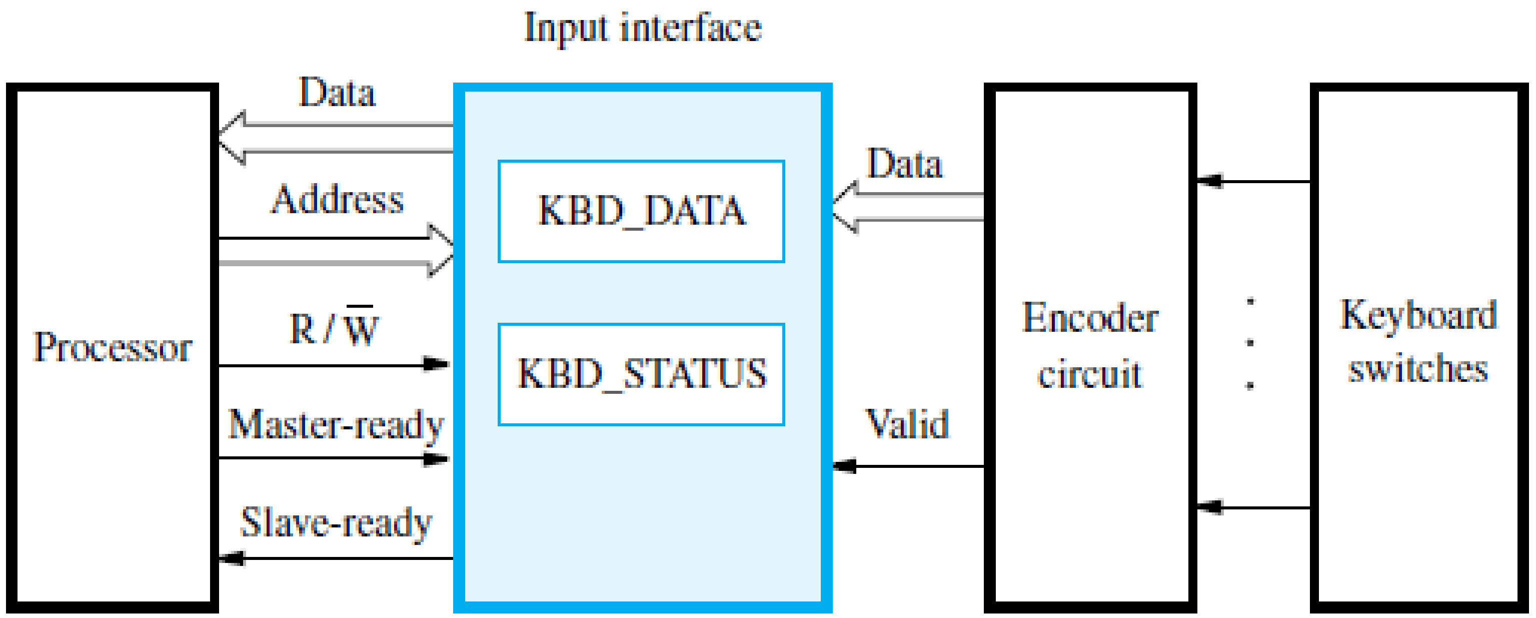
Interface Circuits

- The I/O interface of a device consists of the circuitry needed to connect that device to the bus.
- On one side of the interface are the bus lines for address, data, and control. On the other side are the connections needed to transfer data between the interface and the I/O device. This side is called a port, and it can be either a parallel or a serial port.
- A parallel port transfers multiple bits of data simultaneously to or from the device.
- A serial port sends and receives data one bit at a time.
- Communication with the processor is the same for both formats
- conversion from a parallel to a serial format and vice versa takes place inside the interface circuit.

Parallel Interface

Following examples explain the key aspects of interface design. we describe an interface circuit for an 8-bit input port that can be used for connecting a simple input device, such as a keyboard. Then, we describe an interface circuit for an 8-bit output port, which can be used with an output device such as a display. We assume that these interface circuits are connected to a 32-bit processor that uses the asynchronous bus protocol.

Input Interface



Keyboard to processor connection

- There are only two registers: a data register, KBD_DATA, and a status register, KBD_STATUS. The latter contains the keyboard status flag, KIN.

- The output of the encoder consists of one byte of data representing the encoded character and one control signal called Valid.

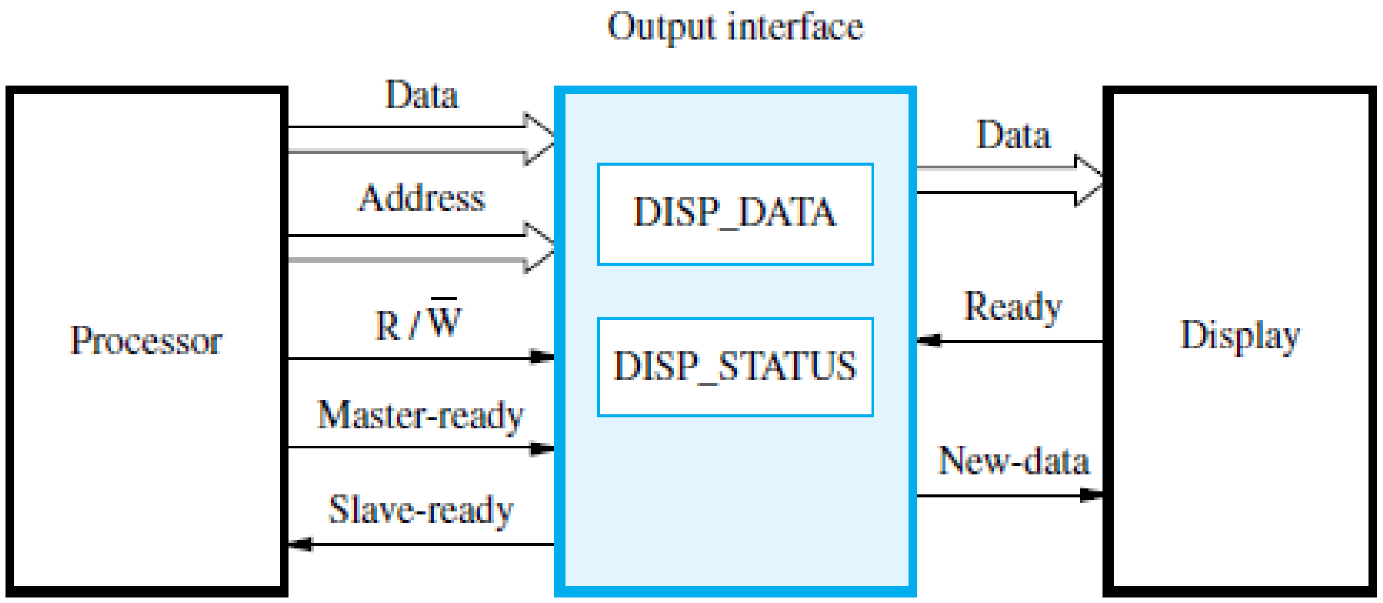
- When a key is pressed, the Valid signal changes from 0 to 1, causing the ASCII code of the corresponding character to be loaded into the KBD_DATA register and the status flag KIN to be set to 1.

- The status flag is cleared to 0 when the processor reads the contents of the KBD_DATA register.

- The interface circuit is shown connected to an asynchronous bus on which transfers are controlled by the handshake signals Master-ready and Slave-ready.

- The bus has one other control line, R/W, which indicates a Read operation when equal to 1.

Output Interface



Display to processor connection

Let us now consider the output interface , which can be used to connect an output device such as a display. We have assumed that the display uses two handshake signals, New-data and Ready.

- When the display is ready to accept a character, it asserts its Ready signal, which causes the DOUT flag in the DISP_STATUS register to be set to 1.

- When the I/O routine checks DOUT and finds it equal to 1, it sends a character to DISP_DATA.

- This clears the DOUT flag to 0 and sets the New-data signal to 1.

- In response, the display returns Ready to 0 and accepts and displays the character in DISP_DATA.

- When it is ready to receive another character, it asserts Ready again, and the cycle repeats.

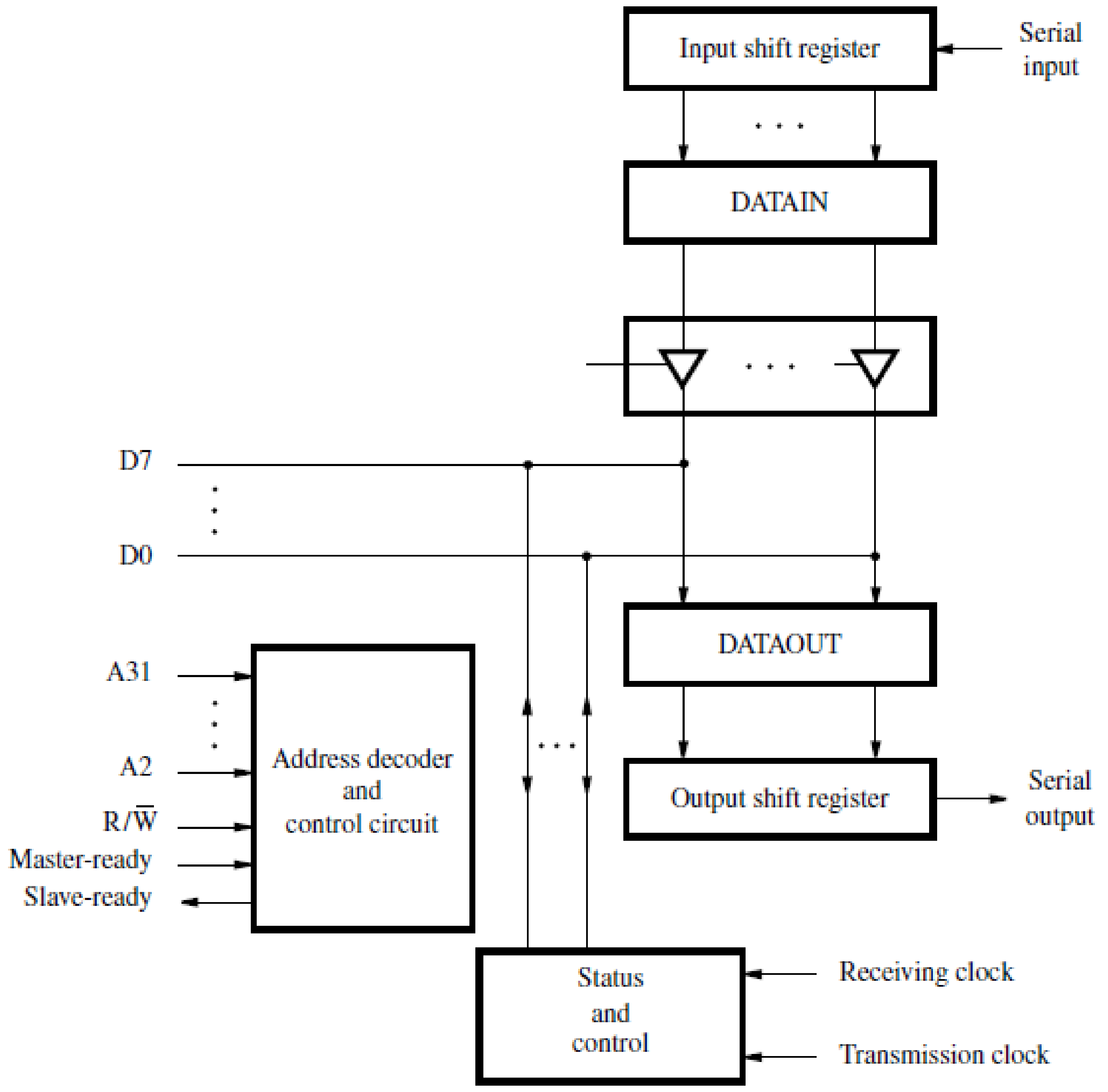
Serial Interface

- A serial interface is used to connect the processor to I/O devices that transmit data one bit at a time.

- Data are transferred in a bit-serial fashion on the device side and in a bit-parallel fashion on the processor side.

- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.

A block diagram of a typical serial interface is shown in next slide.



A serial interface.

- The input shift register accepts bit-serial input from the I/O device.
- When all 8 bits of data have been received, the contents of this shift register are loaded in parallel into the DATAIN register.
- Similarly, output data in the DATAOUT register are transferred to the output shift register, from which the bits are shifted out and sent to the I/O device.
- Two status flags, SIN and SOUT, are maintained by the Status and control block.
- The SIN flag is set to 1 when new data are loaded into DATAIN from the shift register, and cleared to 0 when these data are read by the processor.
- The SOUT flag indicates whether the DATAOUT register is available.
- It is cleared to 0 when the processor writes new data into DATAOUT and set to 1 when data are transferred from DATAOUT to the output shift register.

Interconnection Standards

- A typical desktop or notebook computer has several ports that can be used to connect I/O devices, such as a mouse, a memory key, or a disk drive.

- Standard interfaces have been developed to enable I/O devices to use interfaces that are independent of any particular processor.

  For example, a memory key that has a USB connector can be used with any computer that has a USB port.

- Most standards are developed by a collaborative effort among a number of companies. In many cases, the IEEE develops these standards further and publishes them as IEEE Standards.

Universal Serial Bus (USB)

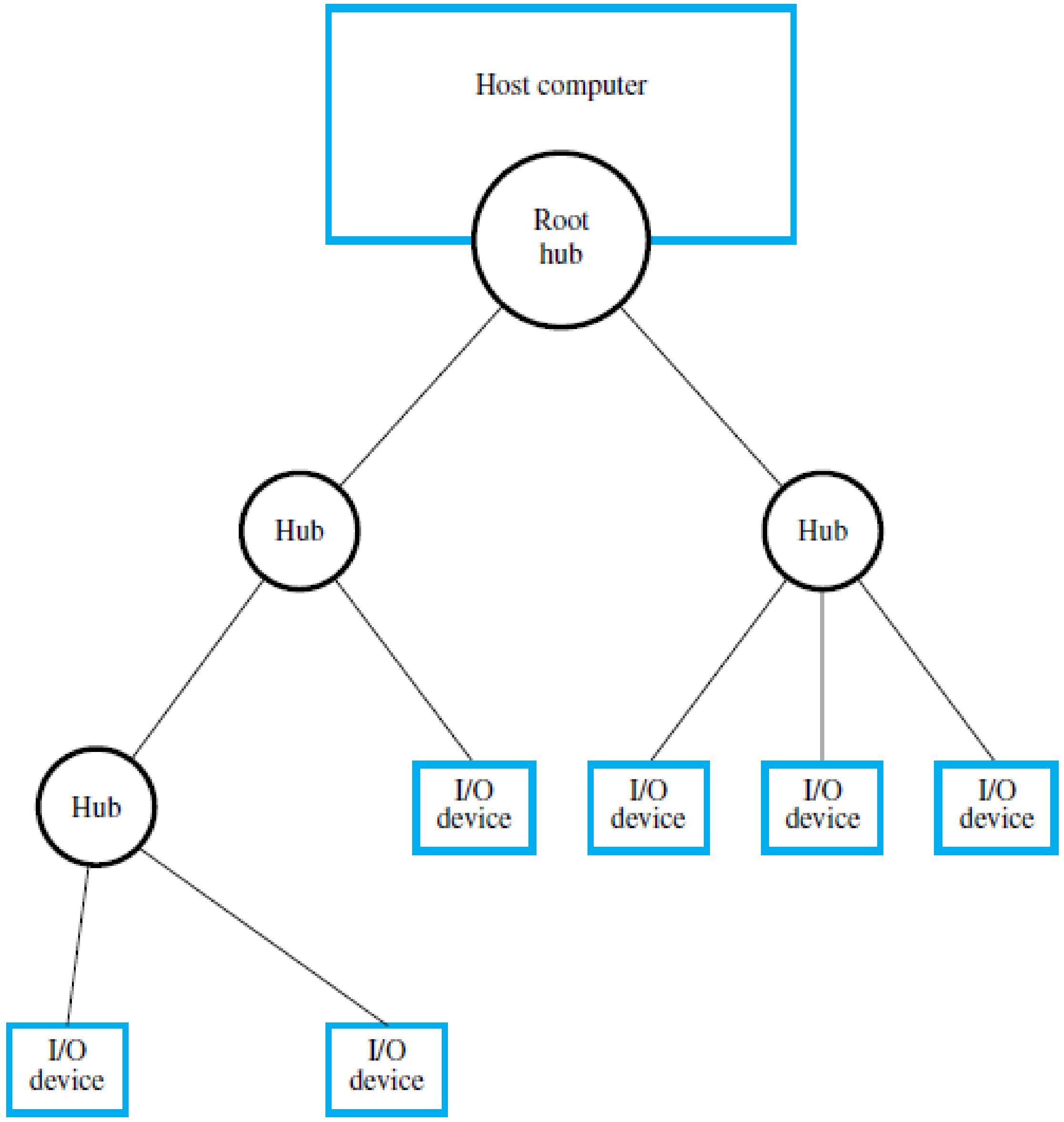
- The Universal Serial Bus (USB) is the most widely used interconnection standard.
- A large variety of devices are available with a USB connector, including mice, memory keys, disk drives, printers, cameras, and many more.
- The commercial success of the USB is due to its simplicity and low cost.
- The original USB specification supports two speeds of operation, called low-speed (1.5 Megabits/s) and full-speed (12 Megabits/s). Later, USB 2, called High-Speed USB, was introduced.
  It enables data transfers at speeds up to 480 Megabits/s. As I/O devices continued to evolve with even higher speed requirements, USB 3 (called Superspeed) was developed. It supports data transfer rates up to 5 Gigabits/s.

The USB has been designed to meet several key objectives:

- Provide a simple, low-cost, and easy to use interconnection system
- Accommodate a wide range of I/O devices and bit rates, including Internet connections, and audio and video applications
- Enhance user convenience through a “plug-and-play” mode of operation

USB Architecture

- The USB uses point-to-point connections and a serial transmission format.
- When multiple devices are connected, they are arranged in a tree structure



Universal Serial Bus tree structure

- Each node of the tree has a device called a *hub, which acts as an intermediate transfer point* between the host computer and the I/O devices.
- At the root of the tree, a *root hub connects* the entire tree to the host computer.
- The leaves of the tree are the I/O devices: a mouse, a keyboard, a printer, an Internet connection, a camera, or a speaker.
- The tree structure makes it possible to connect many devices using simple point-to-point serial links.
- If I/O devices are allowed to send messages at any time, two messages may reach the hub at the same time and interfere with each other.
- For this reason, the USB operates strictly on the basis of polling.

- A device may send a message only in response to a poll message from the host processor.

- Hence, no two devices can send messages at the same time.

- This restriction allows hubs to be simple, low-cost devices.

FireWire

- FireWire is another popular interconnection standard. It was originally developed by Apple and has been adopted as IEEE Standard 1394 .

- Like the USB, it uses differential point to- point serial links.

The following are some of the salient differences between FireWire and USB.

- Devices are organized in a daisy chain manner on a FireWire bus, instead of the tree structure of USB. One device is connected to the computer, a second device is connected to the first one, a third device is connected to the second one, and so on.

- FireWire is well suited for connecting audio and video equipment. It can be operated in an isochronous mode that is highly optimized for carrying high-speed isochronous traffic.

- I/O devices connected to the USB communicate with the host computer. If data are to be transferred from one device to another, for example from a camera to a display or printer, they are first read by the host then sent to the display or printer.

- FireWire, on the other hand, supports a mode of operation called *peer-to-peer. This means that data may be* transferred directly from one I/O device to another, without the host's involvement.

- The basic FireWire connector has six pins. There are two pairs of data wires, one for transmission in each direction, and two for power and ground.

- The FireWire bus can deliver considerably more power than the USB. Hence, it can support devices with moderate power requirements.

- Several versions of the standard have been defined, which can operate at speeds ranging from 400 Megabits/s to 3.6 Gigabits/s.

PCI Bus

- The PCI (Peripheral Component Interconnect) bus was developed as a low-cost, processor-independent bus.

- It is housed on the motherboard of a computer and used to connect I/O interfaces for a wide variety of devices.

- A device connected to the PCI bus appears to the processor as if it is connected directly to the processor bus. Its interface registers are assigned addresses in the address space of the processor.

Bus Structure

- The PCI bus is connected to the processor bus via a controller called a bridge.

- The bridge has a special port for connecting the computer’s main memory.

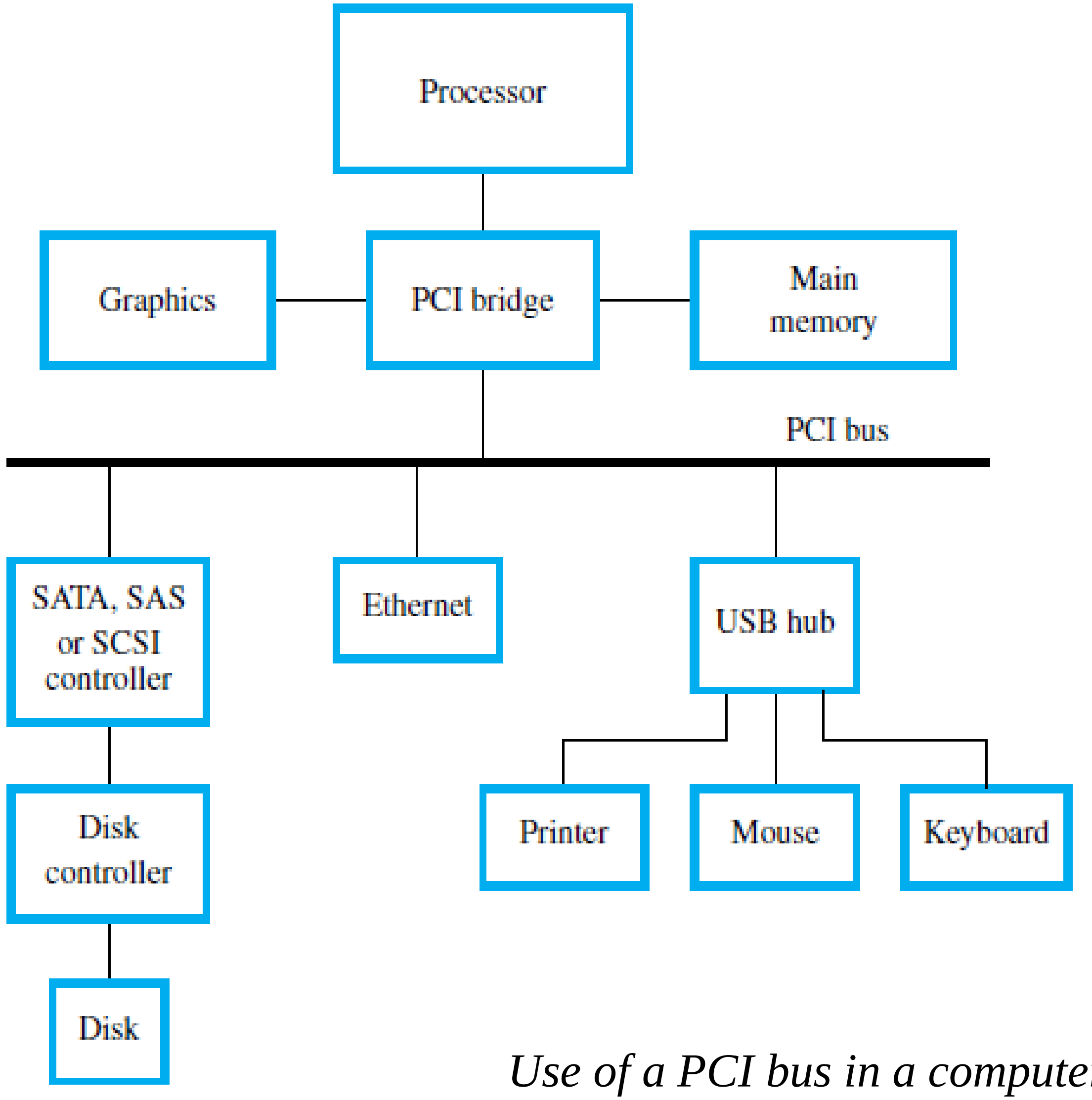
- It may also have another special high speed port for connecting graphics devices.

- The bridge translates and relays commands and responses from one bus to the other and transfers data between them.

For example,

- when the processor sends a Read request to an I/O device, the bridge forwards the command and address to the PCI bus. When the bridge receives the device’s response, it forwards the data to the processor using the processor bus

- I/O devices are connected to the PCI bus, possibly through ports that use standards such as Ethernet, USB, SATA, SCSI, or SAS.



Use of a PCI bus in a computer system.

Data Transfer

To understand the operation of the bus and its various features, we will examine a typical bus transaction.

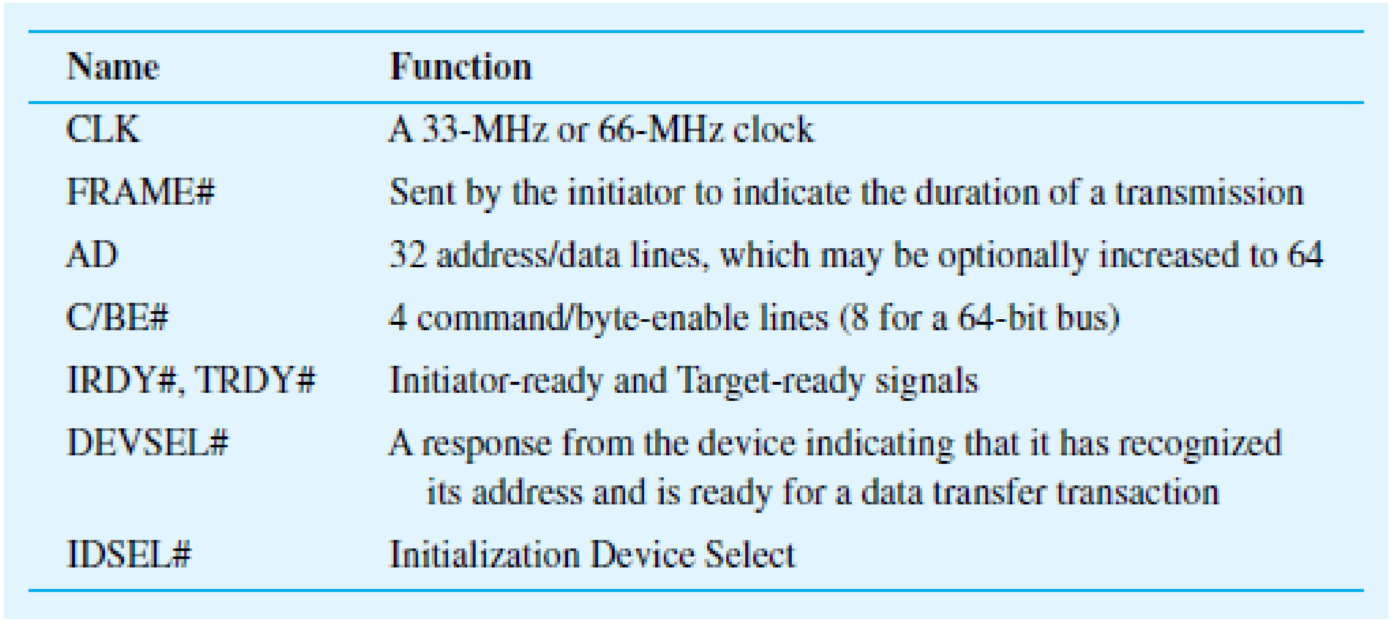
• The bus master, which is the device that initiates data transfers by issuing Read and Write commands, is called the *initiator in PCI terminology.*

• *The addressed* device that responds to these commands is called a *target.*

• *The main bus signals used for* transferring data are listed in a Table Shown in next slide

•There are 32 or 64 lines that carry address and data using a synchronous signaling scheme.

Data transfer signals on the PCI bus.



Name	Function
CLK	A 33-MHz or 66-MHz clock
FRAME#	Sent by the initiator to indicate the duration of a transmission
AD	32 address/data lines, which may be optionally increased to 64
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus)
IRDY#, TRDY#	Initiator-ready and Target-ready signals
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction
IDSEL#	Initialization Device Select

- The target-ready, TRDY#, signal is equivalent to the Slave-ready signal.

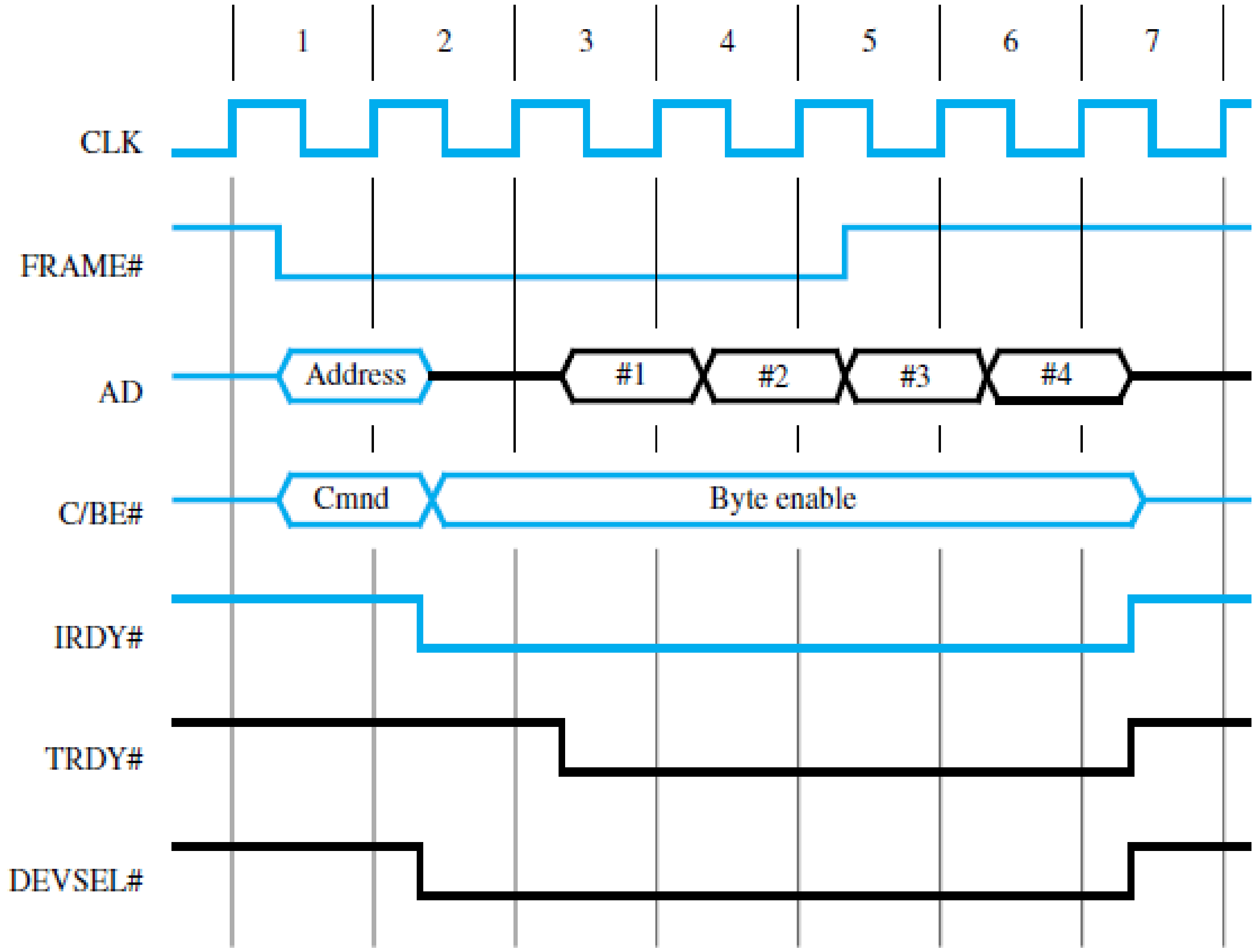
- In addition, PCI uses an initiator-ready signal, IRDY#, to support burst transfers.

• A complete transfer operation on the PCI bus, involving an address and a burst of data, is called a *transaction.*

Consider a bus transaction in which an initiator reads four consecutive 32-bit words from the memory. The sequence of events on the bus is illustrated in next slide.

- All signal transitions are triggered by the rising edge of the clock..

- A signal whose name ends with the symbol # is asserted when in the low-voltage state.



A Read *operation* on the PCI bus.

- The bus master, acting as the initiator, asserts FRAME# in clock cycle 1 to indicate the beginning of a transaction. At the same time, it sends the address on theAD lines and a command on the C/BE# lines. In this case, the command will indicate that a Read operation is requested and that the memory address space is being used.

- In clock cycle 2, the initiator removes the address, disconnects its drivers from the AD lines, and asserts IRDY# to indicate that it is ready to receive data.

  The selected target asserts DEVSEL# to indicate that it has recognized its address and is ready to respond. At the same time, it enables its drivers on the AD lines, so that it can send data to the initiator in subsequent cycles. Clock cycle 2 is used to accommodate the delays involved in turning the AD lines around, as the initiator turns its drivers off and the target turns its drivers on.

- The target asserts TRDY# in clock cycle 3 and begins to send data. It maintains DEVSEL# in the asserted state until the end of the transaction.

- The C/BE# lines, which are used to send a bus command in clock cycle 1, are used for a different purpose during the rest of the transaction.

- Each of these four lines is associated with one byte on the AD lines. The initiator asserts one or more of the C/BE# lines to indicate which byte lines are to be used for transferring data.

- The initiator uses the FRAME# signal to indicate the duration of the burst. It deactivates this signal during the second-last word of the transfer. The initiator maintains FRAME# in the asserted state until clock cycle 5, the cycle in which it receives the third word.

- In response, the target sends one more word in clock cycle 6, then stops.

- After sending the fourth word, the target deactivates TRDY# and DEVSEL# and disconnects its drivers on the AD lines.

SCSI Bus

- The acronym SCSI stands for Small Computer System Interface .
- It refers to a standard bus defined by the American National Standards Institute (ANSI).
- The SCSI bus may be used to connect a variety of devices to a computer.
- It is particularly well-suited for use with disk drives.
- It is often found in installations such as institutional databases or email systems where many disks drives are used.

To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive.

• Communication with a disk drive differs substantially from communication with the main memory.

• Data are stored on a disk in blocks called *sectors*, *where each* sector may contain several hundred bytes.

• When a data file is written on a disk, it is not always stored in contiguous sectors. Some sectors may already contain previously stored information; others may be defective and must be skipped.

• Hence, a Read or Write request may result in accessing several disk sectors that are not necessarily contiguous.

• Because of the constraints of the mechanical motion of the disk, there is a long delay, on the order of several milliseconds, before reaching the first sector to or from which data are to be transferred.

- Then, a burst of data are transferred at high speed.
- Another delay may ensue to reach the next sector, followed by a burst of data.
- A single Read or Write request may involve several such bursts.
- The SCSI protocol is designed to facilitate this mode of operation.

complete Read operation :

Assume that the processor wishes to read a block of data from a disk drive and that these data are stored in two disk sectors that are not contiguous. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

1. The SCSI controller contends for control of the SCSI bus.

2. When it wins the arbitration process, the SCSI controller sends a command to the disk controller, specifying the required Read operation.

3. The disk controller cannot start to transfer data immediately. It must first move the read head of the disk to the required sector. Hence, it sends a message to the SCSI controller indicating that it will temporarily suspend the connection between them. The SCSI bus is now free to be used by other devices.

4. The disk controller sends a command to the disk drive to move the read head to the first sector involved in the requested Read operation. It reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data, it requests control of the bus. After it wins arbitration, it re-establishes the connection with the SCSI controller, sends the contents of the data buffer, then suspends the connection again.

5. The process is repeated to read and transfer the contents of the second disk sector.

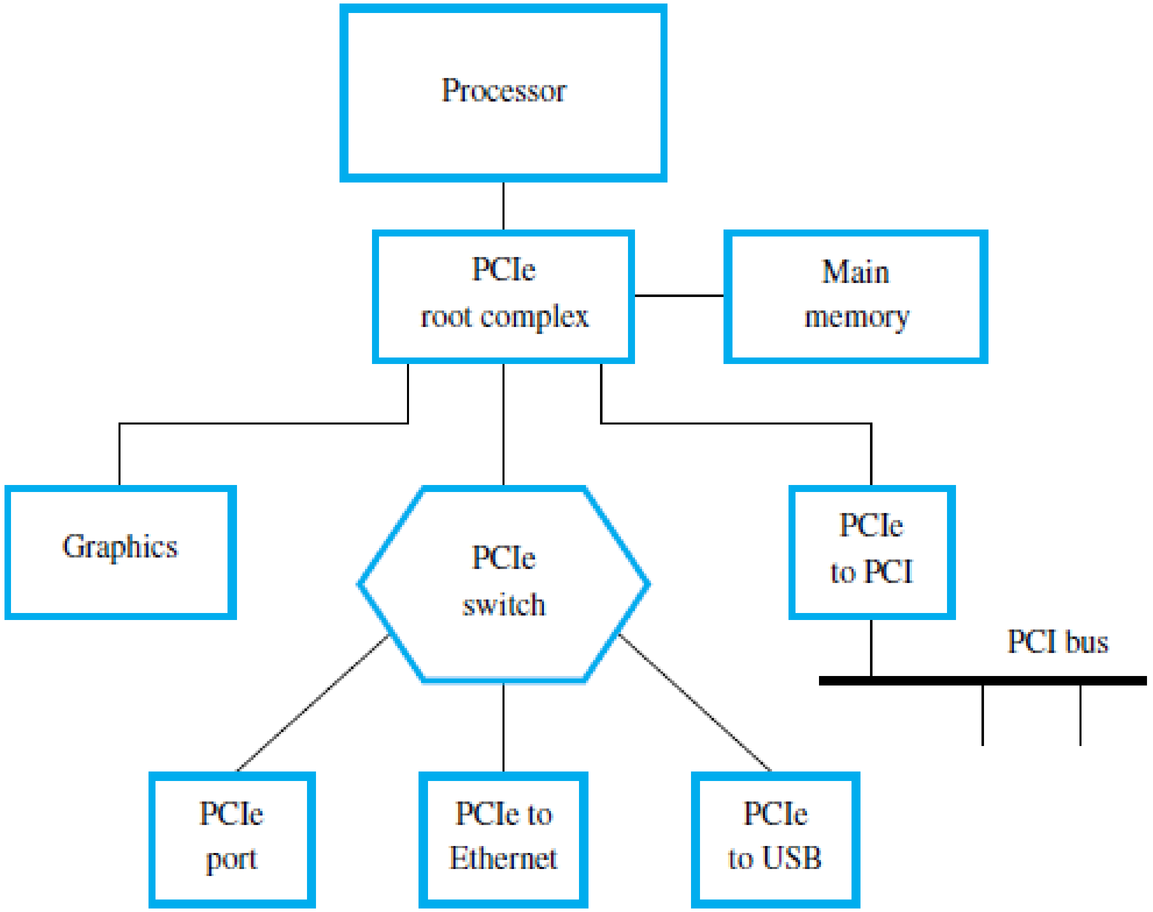
6. The SCSI controller transfers the requested data to the main memory and sends an interrupt to the processor indicating that the data are now available.

PCI Express

- Internet connections, sophisticated graphics devices, streaming video and high-definition television are examples of applications that involve data transfers at very high speed.

- The PCI Express interconnection standard (often called PCIe) has been developed to meet these needs and to anticipate further increases in data transfer rates, which are inevitable as new applications are introduced.

- PCI Express uses serial, point-to-point links interconnected via switches to form a tree structure, as shown in next slide Figure.



PCI Express connections

- PCI Express uses serial, point-to-point links interconnected via switches to form a tree structure.
- The root node of the tree, called the Root complex, is connected to the processor bus. The Root complex has a special port to connect the main memory.
- All other connections emanating from the Root complex are serial links to I/O devices.
- Some of these links may connect to a switch that leads to more serial branches.
- The switch may also connect to bridging interfaces that support other standards, such as PCI or USB.

Hardware Multithreading

- Operating system (OS) software enables multitasking of different programs in the same processor by performing context switches among programs.

- A program, together with any information that describes its current state of execution, is regarded by the OS as an entity called a *process*.

- *Information about the memory and* other resources allocated by the OS is maintained with each process.

- Processes may be associated with applications such as Web-browsing, word-processing, and music-playing programs that a user has opened in a computer.

- Each process has a corresponding thread, which is an independent path of execution within a program.
- The term *thread is used to refer to a thread of control* whose state consists of the contents of the program counter and other processor registers.
- To deal with multiple threads efficiently, a processor is implemented with several identical sets of registers, including multiple program counters.
- Each set of registers can be dedicated to a different thread. Thus, no time is wasted during a context switch to save and restore register contents.
- The processor is said to be using a technique called hardware multithreading.

Vector (SIMD)Processing

- Many computationally demanding applications involve programs that use loops to perform operations on vectors of data, where a vector is an array of elements such as integers or floating-point numbers.

- When a processor executes the instructions in such a loop, the operations are performed one at a time on individual vector elements.

- As a result, many instructions need to be executed to process all vector elements.

- A processor can be enhanced with multiple ALUs. In such a processor, it is possible to operate on multiple data elements in parallel using a single instruction.

- Such instructions are called *single-instruction multiple-data (SIMD) instructions. They are also called vector instructions.*

- *These instructions can only be used when the operations performed in parallel* are independent. This is known as *data parallelism.*

- The data for vector instructions are held in vector registers, each of which can hold several data elements.

- The number of elements, L, in each vector register is called the vector length.

- It determines the number of operations that can be performed in parallel on multiple ALUs.

- If vector instructions are provided for different sizes of data elements using the same vector registers, L may vary.

Some typical examples of vector instructions are given below to illustrate how vector registers are used. We assume that the OP-code mnemonic includes a suffix S which specifies the size of each data element. This determines the number of elements, *L, in a* vector. For instructions that access the memory, the contents of a conventional register are used in the calculation of the effective address.

VectorAdd.S Vi, Vj, Vk
; computes L sums using the elements in vector registers Vj and Vk,
  and places the resulting sums in vector register Vi.

VectorLoad.S Vi, X(Rj)
; causes L consecutive elements beginning at memory location X + [Rj]
  to be loaded into vector register Vi.

VectorStore.S Vi, X(Rj)
; causes the contents of vector register Vi to be stored as L consecutive
  elements in thememory.

Shared-Memory Multiprocessors

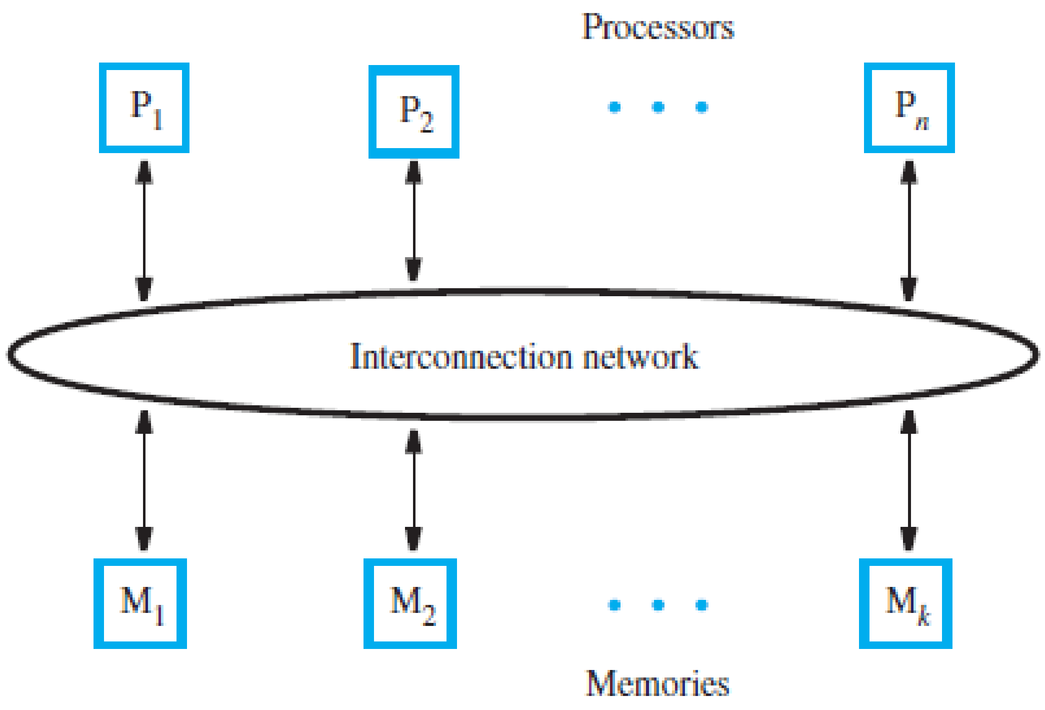
- In a shared-memory multiprocessor, all processors have access to the same memory.
- Tasks running in different processors can access shared variables in the memory using the same addresses.
- The size of the shared memory is likely to be large.
- Implementing a large memory in a single module would create a bottleneck when   many processors make requests to access the memory simultaneously.
- This problem is alleviated by distributing the memory In a shared-memory multiprocessor, across multiple modules so that simultaneous requests from different processors are more likely to access different memory modules, depending on the addresses of those requests.

- An interconnection network enables any processor to access any module that is a part of the shared memory.

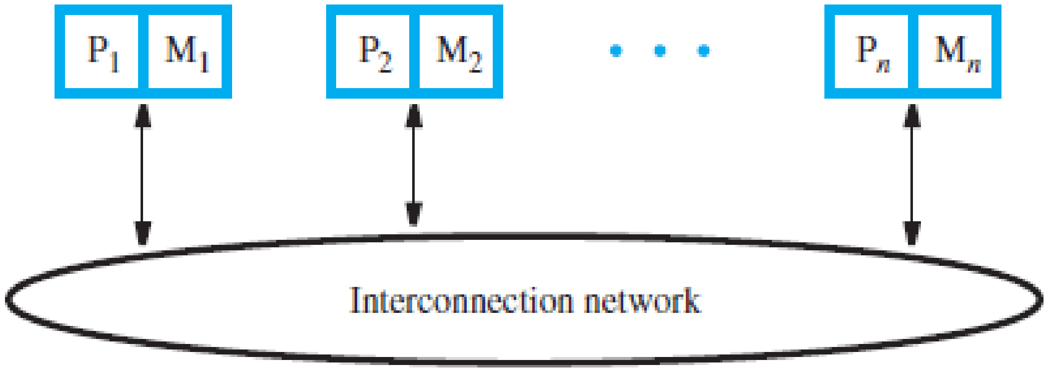
- When memory modules are kept physically separate from the processors, all requests to access memory must pass through the network, which introduces latency.

- A system which has the same network latency for all accesses from the processors to the memory modules is called a Uniform Memory Access (UMA) multiprocessor.

- Although the latency is uniform, it may be large for a network that connects many processors and memory modules.



A UMA multiprocessor.



A NUMA multiprocessor

- For better performance, it is desirable to place a memory module close to each processor.

- The result is a collection of *nodes, each consisting of a processor and a memory module.*

- *The* nodes are then connected to the network.

- The network latency is avoided when a processor makes a request to access its local memory.

- However, a request to access a remote memory module must pass through the network.

- Because of the difference in latencies for accessing local and remote portions of the shared memory, systems of this type are called Non-Uniform Memory Access (NUMA) multiprocessors.

Interconnection Networks

- The interconnection network must allow information transfer between any pair of nodes in the system.

- The network may also be used to broadcast information from one node to many other nodes.

- The traffic in the network consists of requests (such as read and write) and data transfers.

- A few of the interconnection networks that are commonly used in multiprocessors are:

Bus

- A *bus is a set of lines (wires) that provide a single shared path for information transfer.*

- Buses are most commonly used in UMA multiprocessors to connect a number of processors to several shared-memory modules.

- Arbitration is necessary to ensure that only one of many possible requesters is granted use of the bus at any time.

- The bus is suitable for a relatively small number of processors because of the contention for access to the bus and the increased propagation delays caused by electrical loading when many processors are connected.

Ring

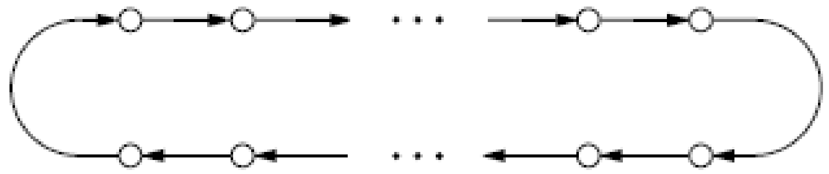
- A *ring network is formed with point-to-point connections between nodes.*

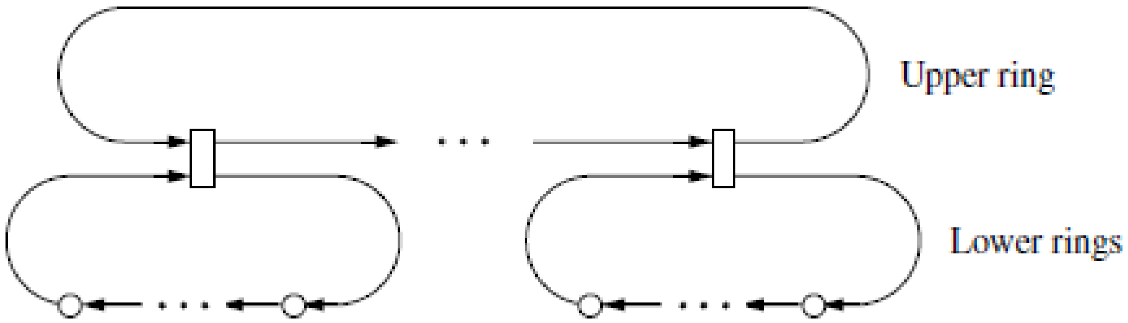
- *A long single ring results in high* average latency for communication between any two nodes.

- This high latency can be mitigated in two different ways.

  A second ring can be added to connect the nodes in the opposite direction. The resulting *bidirectional ring halves the average latency and doubles the bandwidth. However, handling* of communications is more complex.

  Another approach is to use a *hierarchy of rings. The upper-level ring connects the lower-level rings. The average latency* for communication between any two nodes on lower-level rings is reduced with this arrangement.


(a) Single ring



(b) Hierarchy of rings

Ring-based interconnection networks

Crossbar

- A *crossbar is a network that provides a direct link between any pair of units connected to* the network.

- It is typically used in UMA multiprocessors to connect processors to memory modules.

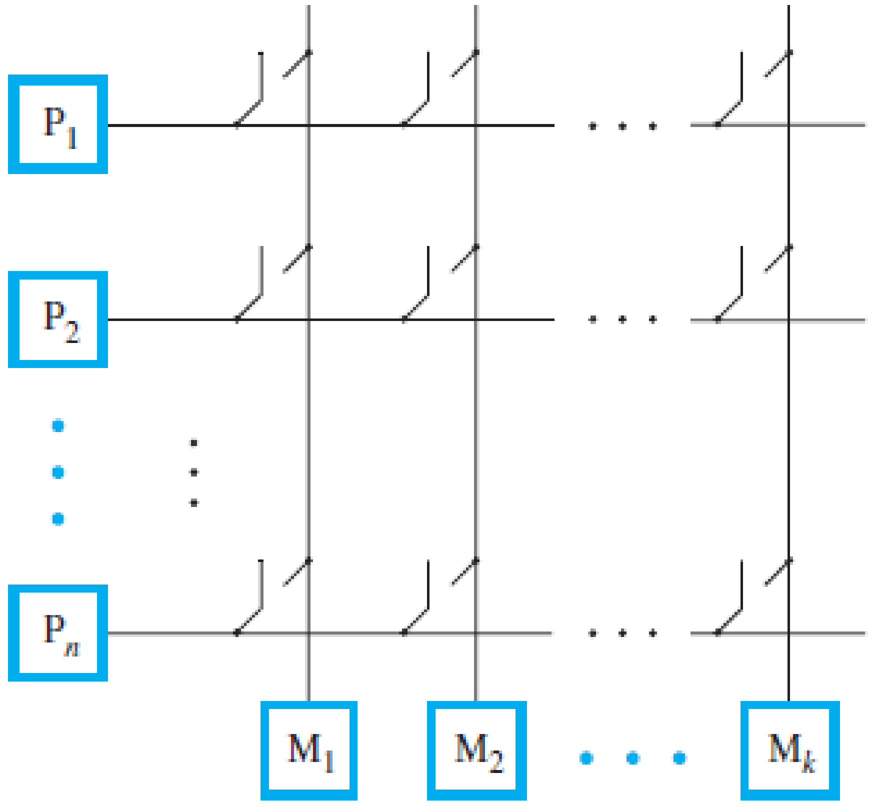
- It enables many simultaneous transfers if the same destination is not the target of multiple requests.

- For *n processors and k* memories, $n \times k$ *switches are needed*.

- Transfers between nodes on the same lower-level ring need not traverse the upper-level ring.

- Transfers between nodes on different lower-level rings include a traversal on part of the upper-level ring.

- The drawback of the hierarchical scheme is that the upper-level ring may become a bottleneck when many nodes on different lower-level rings communicate with each other frequently.



Crossbar interconnection network.

Mesh

- Each internal node of the mesh has four connections, one to each of its horizontal and vertical neighbors.
- Nodes on the boundaries and corners of the mesh have fewer neighbors and hence fewer connections.
- To reduce latency for communication between nodes that would otherwise be far apart in the mesh, wraparound connections may be introduced between nodes at opposite boundaries of the mesh.
- A network with such connections is called a *torus*. *All nodes in a torus have four connections.*
- *Average latency* is reduced, but the implementation complexity for routing requests and responses through a torus is somewhat higher than in the case of a simple mesh.