Unit - 4
Input/Output Organization

1. **Accessing I/O devices**

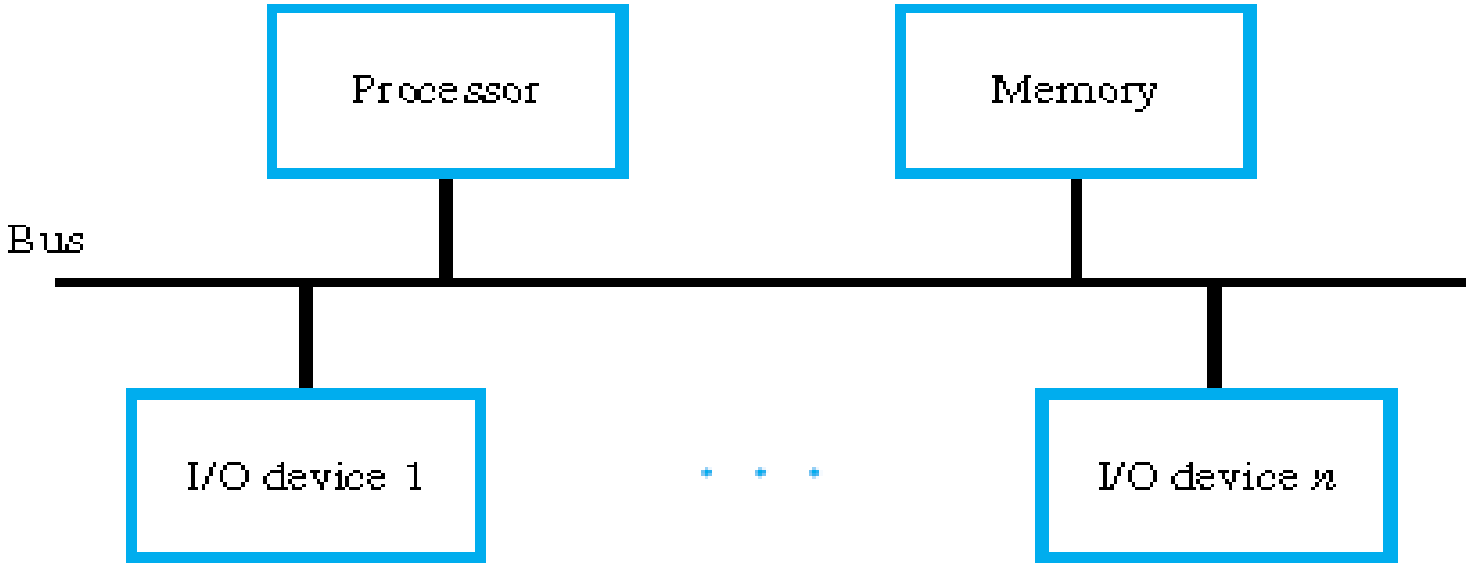


Figure 7.1 A single-bus structure.

- Memory-mapped I/O
- From the total address space of the memory, part of it is reserved for I/O devices
- For processor I/O devices seem like these addressed memory locations
- Shared address space
- Hence, with memory-mapped I/O, any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- Simplified addressing method
- For example,
  - Load R2, DATAIN
  - Store R2, DATAOUT

Note: I/O mapped I/O where separate ports and their addresses have to be used for accessing I/O devices

Most computer systems use memory-mapped I/O. Some processors have special In and Out instructions to perform I/O transfers. For example, processors in the Intel family described in Chapter 3 have special I/O instructions and a separate 16-bit address space for I/O devices. When building a computer system based on these processors, the designer has the option of connecting I/O devices to use the special I/O address space or simply incorporating them as part of the memory address space. The latter approach is by far the most common as it leads to simpler software. One advantage of a separate I/O address space is that I/O devices deal with fewer address lines. Note that a separate I/O address space does not necessarily mean that the I/O address lines are physically separate from the memory address lines. A special signal on the bus indicates that the requested read or write transfer is an I/O operation. When this signal is asserted, the memory unit ignores the requested transfer. The I/O devices examine the low-order bits of the address bus to determine whether they should respond.

1.1 I/O Device Interface

One register may serve as a buffer for data transfers, another may hold information about the current status of the device, and yet another may store the information that controls the operational behavior of the device.

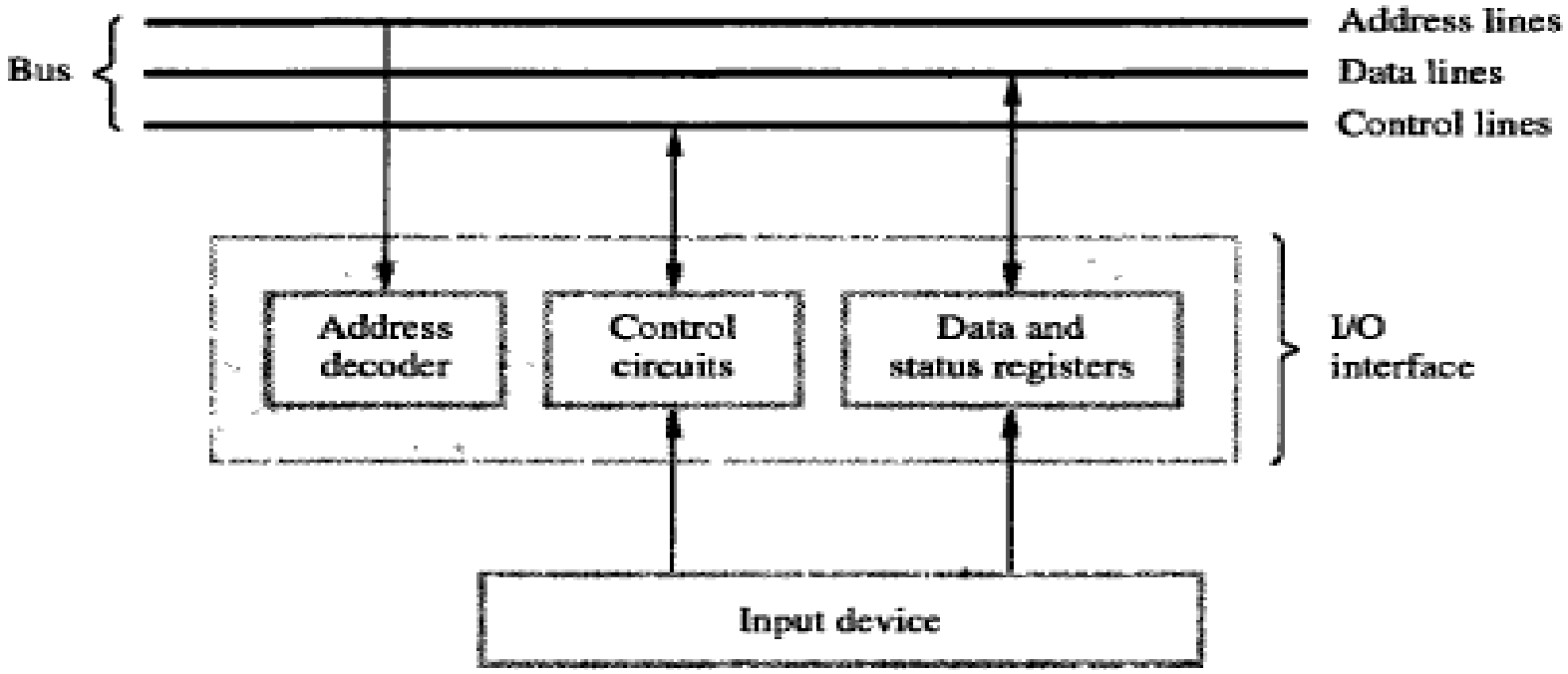


Figure 4.2 I/O interface for an input device.

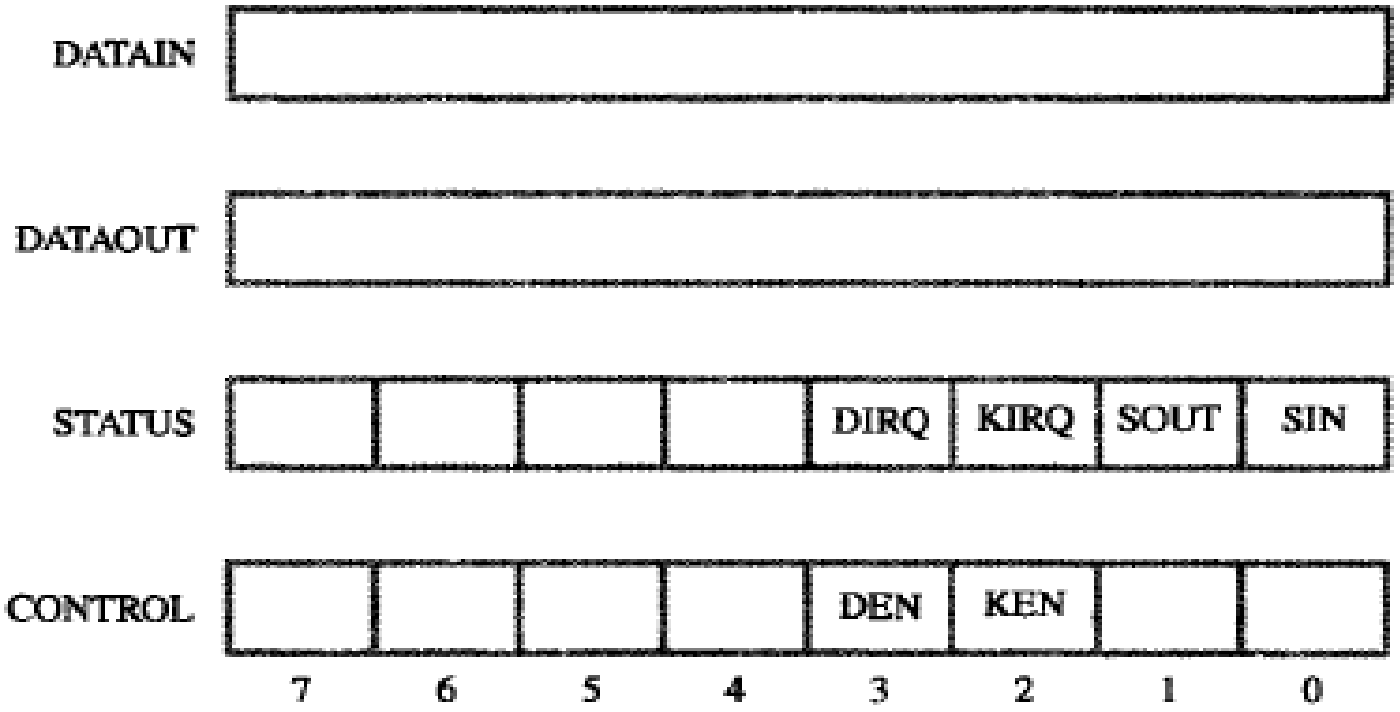


Figure 4.3 Registers in keyboard and display interfaces.

```
        Move        #LINE,R0          Initialize memory pointer.
WAITK   TestBit     #0,STATUS         Test SIN.
        Branch=0    WAITK             Wait for character to be entered.
        Move        DATAIN,R1         Read character.
WAITD   TestBit     #1,STATUS         Test SOUT.
        Branch=0    WAITD             Wait for display to become ready.
        Move        R1,DATAOUT        Send character to display.
        Move        R1,(R0)+          Store charater and advance pointer.
        Compare     #$0D,R1           Check if Carriage Return.
        Branch≠0    WAITK             If not, get another character.
        Move        #$0A,DATAOUT      Otherwise, send Line Feed.
        Call        PROCESS           Call a subroutine to process the
                                        the input line.
```

Figure 4.4 A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display.

2. Interrupts

- When a program enters a wait loop in which it repeatedly tests the device status, the processor is not performing any useful computation.
- There are many situations where other tasks can be performed while waiting for an I/O device to become ready.
- To allow this to happen, we can arrange for the I/O device to alert the processor when it becomes ready.
- It can do so by sending a hardware signal called an interrupt to the processor.
- Since the processor is no longer required to continuously poll the status of I/O devices, it can use the waiting period to perform other useful tasks.
- Hence, by using interrupts, waiting periods can be eliminated.

Example demonstrating the need for the interrupts

- **C**onsider a task that requires continuous extensive computations to be performed and the results to be displayed on a display device every 10 seconds.
- The ten-second intervals can be determined by a simple timer circuit, which generates an appropriate signal.
- The processor treats the timer circuit as an input device that produces a signal that can be interrogated.
- If this is done by means of polling, the processor will waste considerable time checking the state of the signal.
- A better solution is to have the timer circuit raise an interrupt request once every ten seconds.
- In response, the processor displays the latest results.

- The task can be implemented with a program that consists of two routines, COMPUTE and DISPLAY. The processor continuously executes the COMPUTE routine.
- When it receives an interrupt request from the timer, it suspends the execution of the COMPUTE routine and executes the DISPLAY routine which sends the latest results to the display device.
- Upon completion of the DISPLAY routine, the processor resumes the execution of the COMPUTE routine.
- Since the time needed to send the results to the display device is very small compared to the ten-second interval, the processor in effect spends almost all of its time executing the COMPUTE routine.

ISR- Interrupt Service Routine

- The routine executed in response to an interrupt request is called the *interrupt-service routine*, which is the DISPLAY routine in our example.
- Assume that an interrupt request arrives during execution of instruction. The processor first completes execution of instruction *i*. Then, it loads the program counter with the address of the first instruction of the interrupt-service routine.
- After execution of the interrupt-service routine, the processor returns to instruction *i* + 1.
- A Return-from-interrupt instruction at the end of the interrupt-service routine reloads the PC from that temporary storage location, causing execution to resume at instruction *i* + 1.
- The *return address* must be saved either in a designated general-purpose register or on the processor stack.
- Processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal.
- This can be accomplished by means of a special control signal, called *interrupt acknowledge*
- interrupt-service routine that accesses the status or data register in the device interface implicitly informs the device that its interrupt request has been recognized.

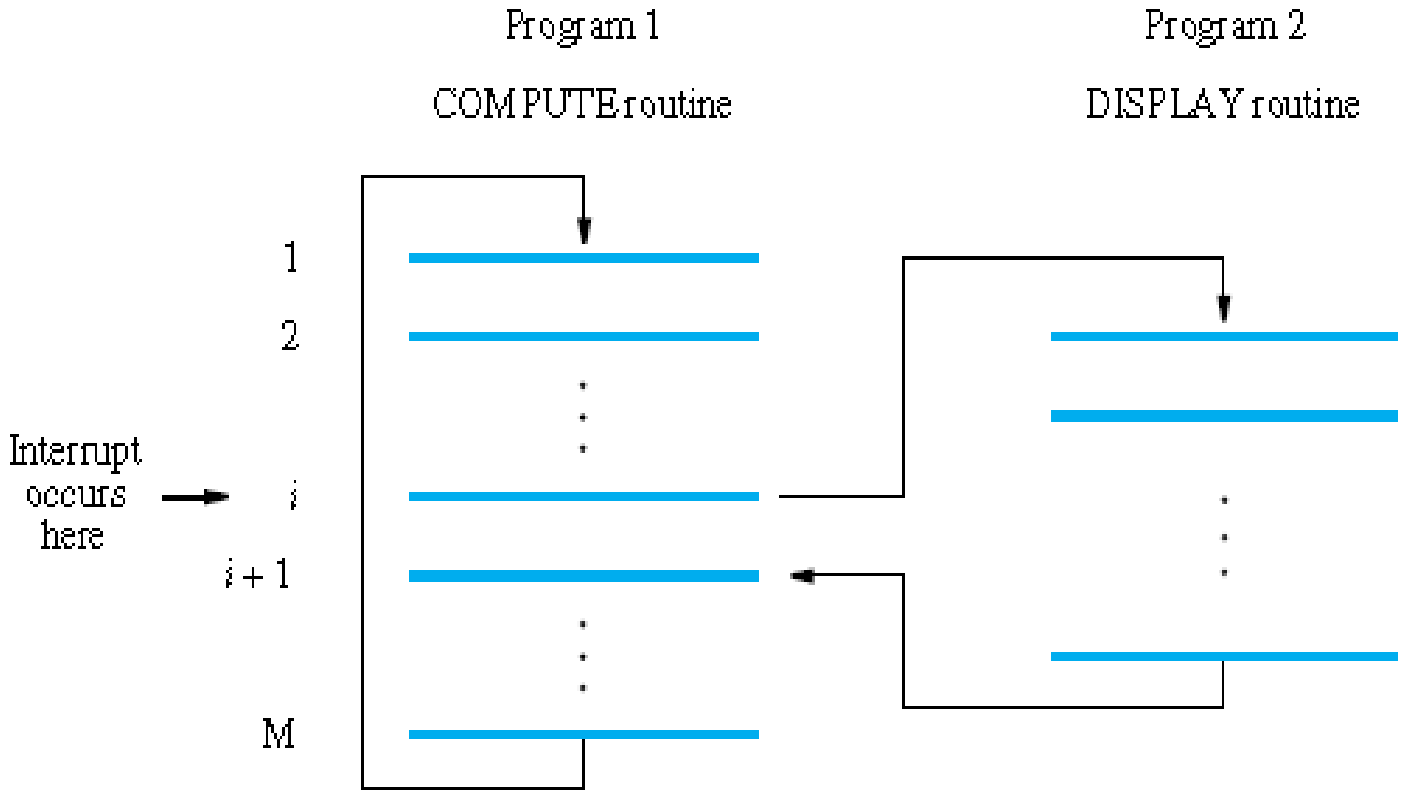


Figure 3.6 Transfer of control through the use of interrupts.

- Before starting execution of the interrupt service routine, status information and contents of processor registers of interrupted program must be saved. This saved information must be restored before execution of the interrupted program is resumed.
- Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. This delay is called interrupt latency.
- Some computers provide two types of interrupts. One saves all register contents, and the other does not. Another interesting approach is to provide duplicate sets of processor registers. In this case, a different set of registers can be used by the interrupt-service routine, thus eliminating the need to save and restore registers. The duplicate registers are called the shadow registers.

Interrupt Hardware

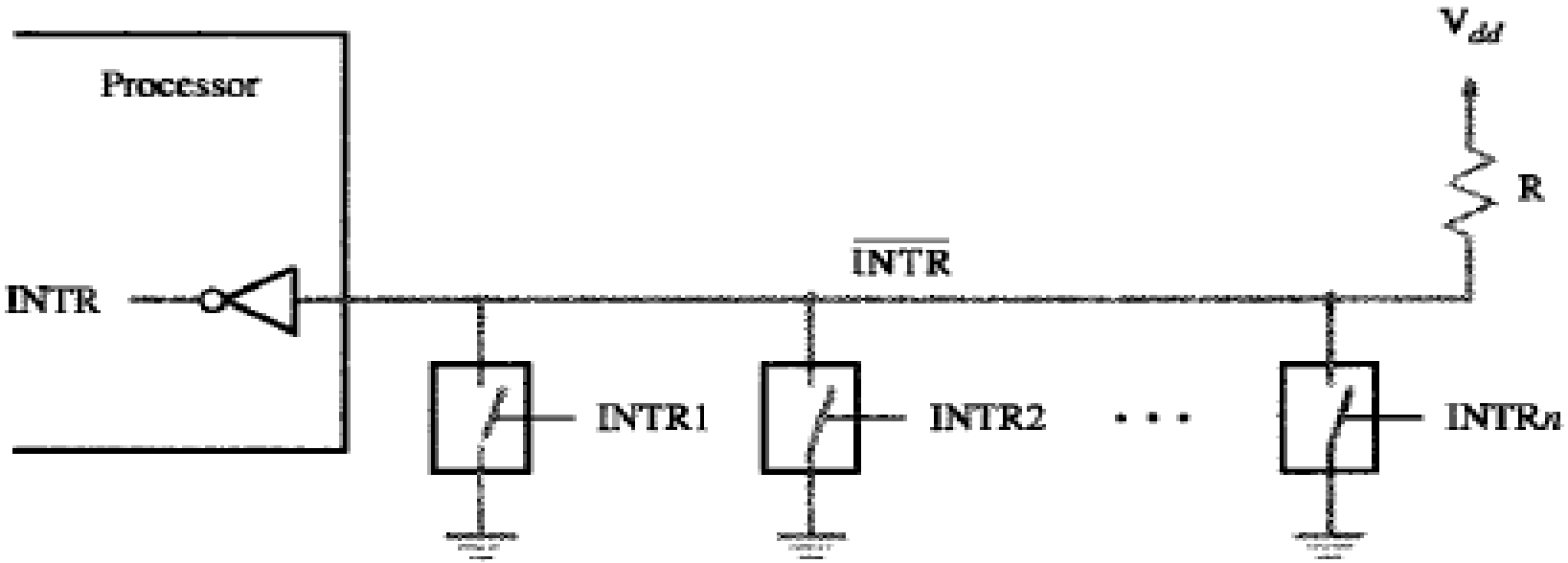


Figure 4.6 An equivalent circuit for an open-drain bus used to implement a common interrupt-request line.

of INTR is the logical OR of the requests from individual devices, that is,

$$\text{INTR} = \text{INTR}_1 + \cdots + \text{INTR}_n$$

Enabling and Disabling Interrupts

- It is convenient to be able to enable and disable interrupts at both the processor and I/O device ends.
- The processor can either accept or ignore interrupt requests.
- An I/O device can either be allowed to raise interrupt requests or prevented from doing so.
- A commonly used mechanism to achieve this is to use some control bits in registers that can be accessed by program instructions.

Enabling and Disabling Interrupts

- The processor has a *status register* (PS), which contains information about its current state of operation.
- IE flag is assigned for enabling/disabling interrupts.
- Then, the programmer can set or clear IE to cause the desired action.
- When IE = 1, interrupt requests from I/O devices are accepted and serviced by the processor.
- When IE = 0, the processor simply ignores all interrupt requests from I/O devices.

- A good choice is to have the processor automatically disable interrupts before starting the execution of the interrupt-service routine.
- The processor saves the contents of the program counter and the processor status register.
- After saving the contents of the PS register, with the IE bit equal to 1, the processor clears the IE bit in the PS register, thus disabling further interrupts.
- Then, it begins execution of the interrupt-service routine.
- When a Return-from-interrupt instruction is executed, the saved contents of the PS register are restored, setting the IE bit back to 1.
- Hence, interrupts are again enabled.

Enabling and Disabling Interrupts

the sequence of events involved in handling an interrupt request from a single device. Assuming that interrupts are enabled, the following is a typical scenario:

1. The device raises an interrupt request.
2. The processor interrupts the program currently being executed.
3. Interrupts are disabled by changing the control bits in the PS (except in the case of edge-triggered interrupts).
4. The device is informed that its request has been recognized, and in response, it deactivates the interrupt-request signal.
5. The action requested by the interrupt is performed by the interrupt-service routine.
6. Interrupts are enabled and execution of the interrupted program is resumed.

2.3 Handling Multiple Devices

1. How can the processor determine which device is requesting an interrupt?
2. Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
3. Should a device be allowed to interrupt the processor while another interrupt is being serviced?
4. How should two or more simultaneous interrupt requests be handled?

There are many ways to identify the interrupting device:

1. **Polling**
2. **Vectored Interrupts**
3. **Interrupt Nesting**

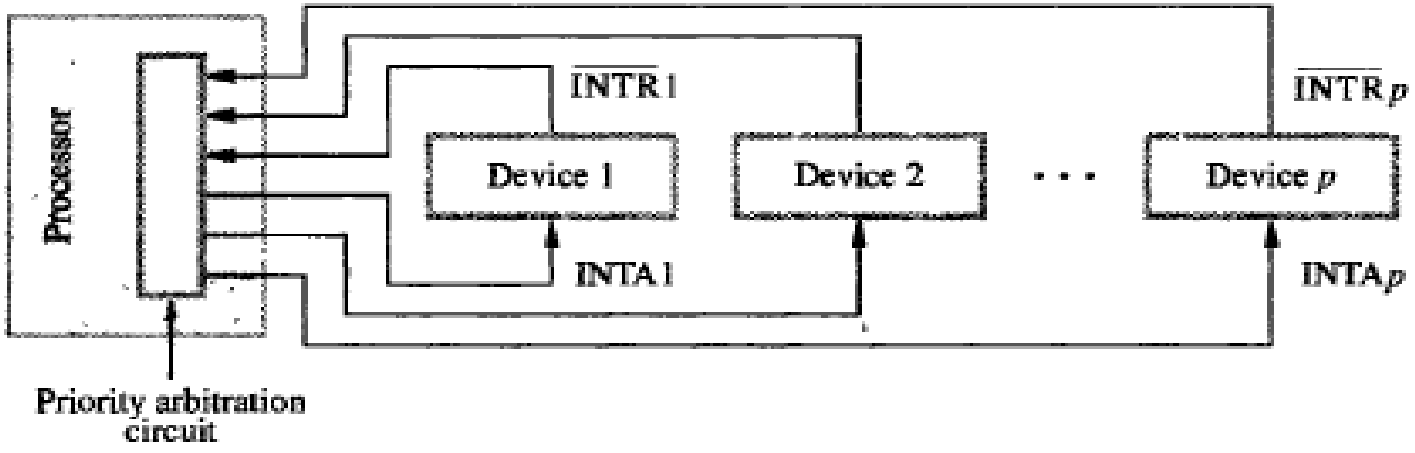


Figure 4.7 Implementation of interrupt priority using individual interrupt-request and acknowledge lines.

1. Polling

- The information needed to determine whether a device is requesting an interrupt is available in its status register.
- When the device raises an interrupt request, it sets to 1 a bit in its status register, which we will call the IRQ bit.
- The simplest way to identify the interrupting device is to have the interrupt-service routine poll all I/O devices in the system.
- The first device encountered with its IRQ bit set to 1 is the device that should be serviced.
- An appropriate subroutine is then called to provide the requested service.
- The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating the IRQ bits of devices that may not be requesting any service

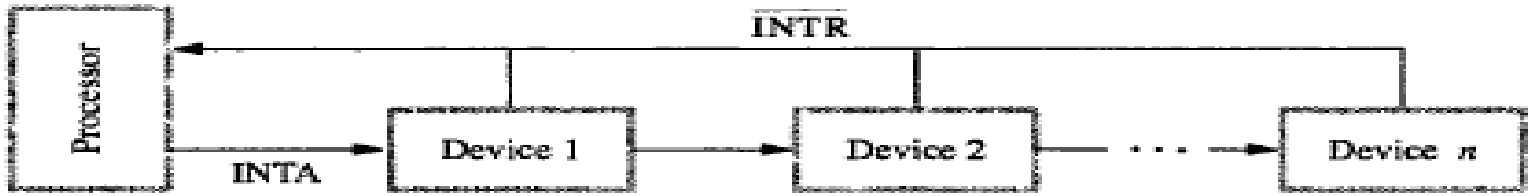
2. Vectored Interrupts

- To reduce the time involved in the polling process, a device requesting an interrupt may identify itself directly to the processor to start executing the corresponding interrupt-service routine.
- A device requesting an interrupt can identify itself if it has its own interrupt-request signal, or if it can send a special code to the processor through the interconnection network.
- The processor's circuits determine the memory address of the required interrupt-service routine.
- These addresses are usually referred to as *interrupt vectors*, and they are said to constitute the *interrupt-vector table*. For example, 128 bytes may be allocated to hold a table of 32 interrupt vectors.
- The interrupt-service routines may be located anywhere in the memory.
- When an interrupt request arrives, the information provided by the requesting device is used as a pointer into the interrupt-vector table, and the address in the corresponding interrupt vector is automatically loaded into the program counter.
- When vectored interrupts are used, we must ensure that only one device is selected to send its interrupt vector code. This is done in hardware, by using arbitration circuits

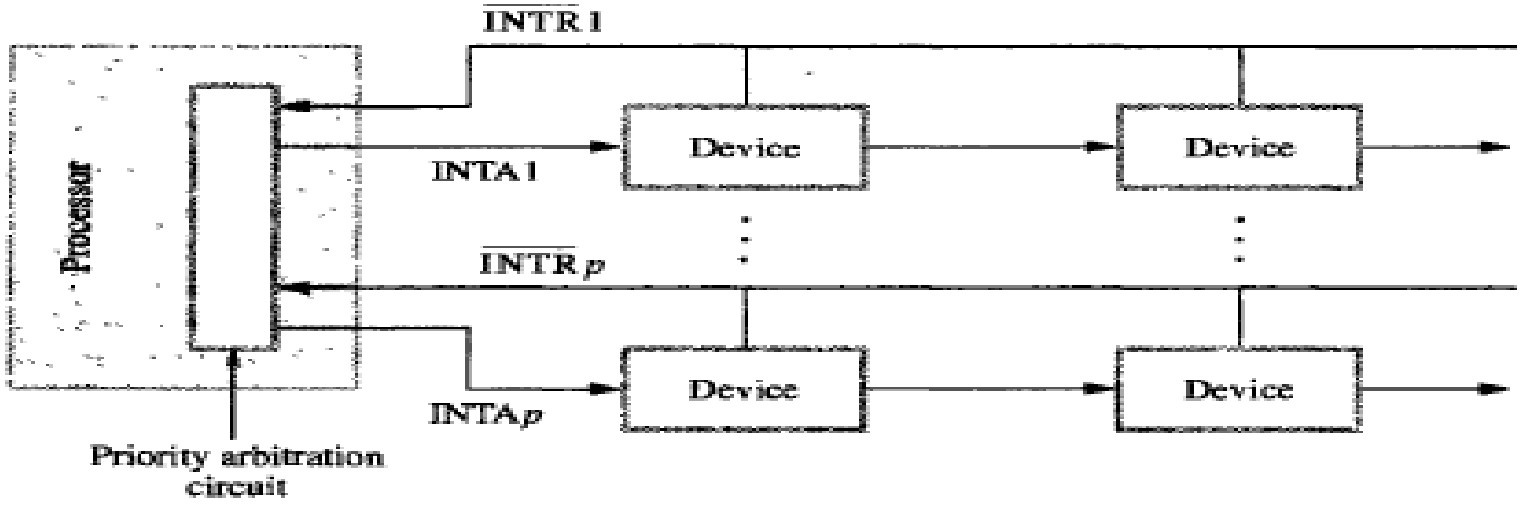
3. Interrupts Nesting

- I/O devices should be organized in a priority structure.
- An interrupt request from a high-priority device should be accepted while the processor is servicing a request from a lower-priority device.
- To implement this scheme, we can assign a priority level to the processor that can be changed under program control.
- The priority level of the processor is the priority of the program that is currently being executed.
- The processor accepts interrupts only from devices that have priorities higher than its own.

4. **Simultaneous Requests**



(a) Daisy chain



(b) Arrangement of priority groups

Figure 4.8 Interrupt priority schemes.

Simultaneous Requests

- Simultaneous arrival of two or more I/O devices
- Generally device with higher priority is accepted first
- If devices are connected on one Interrupt request line:
  - Polling the status registers of the I/O devices is used
  - In this case, priority is determined by the order in which devices are polled
  - If vectored interupts are used, daisy chain can be formed so that INTR line is connected to all devices and INTA line is connected in a daisy chain fashion.
  - When INTR is received, processor sends INTA as 1 and this is received by device 1 first. If it has raised INTR, then it blocks INTA signal and proceeds to put identifying code on the data lines.
  - So devices priority is decided from the order in the daisy chain.
- Devices can be arranged in groups with different priority levels.
- Within a group, devices can be connected in a daisy chain

2.4 Direct Memory Access

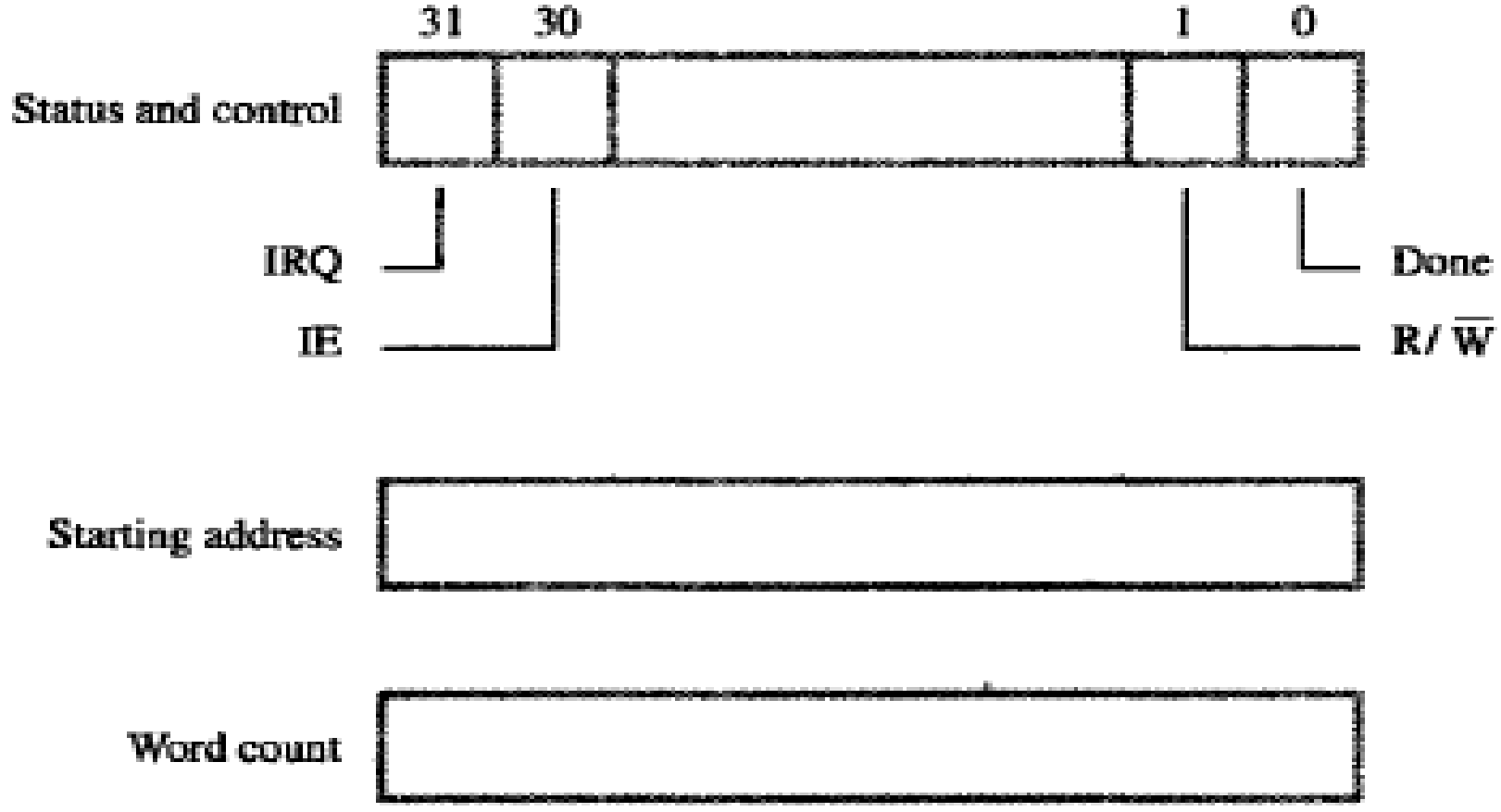


Figure 4.18 Registers in a DMA interface.

DMA

- Blocks of data are often transferred between the main memory and I/O devices such as disks.
- single-word or single-byte data transfers between the processor and I/O devices.
- Data are transferred from an I/O device to the memory by first reading them from the I/O device using an instruction such as
- **Load R2, DATAIN**, which loads the data into a processor register.
- Then, the data read are stored into a memory location. The reverse process takes place for transferring data from the memory to an I/O device.
- An instruction to transfer input or output data is executed only after the processor determines that the I/O device is ready, either by polling its status register or by waiting for an interrupt request.
- In either case, considerable overhead is incurred

DMA

- This section discusses a technique for controlling such transfers without frequent, program-controlled intervention by the processor.
- An alternative approach is used to transfer blocks of data directly between the main memory and I/O devices, such as disks.
- A special control unit is provided to manage the transfer, without continuous intervention by the processor.
- This approach is called *direct memory access*, or DMA. The unit that controls DMA transfers is referred to as a *DMA controller*.
- It may be part of the I/O device interface, or it may be a separate unit shared by a number of I/O devices.
- The DMA controller performs the functions that would normally be carried out by the processor when accessing the main memory.
- For each word transferred, it provides the memory address and generates all the control signals needed.
- It increments the memory address for successive words and keeps track of the number of transfers.

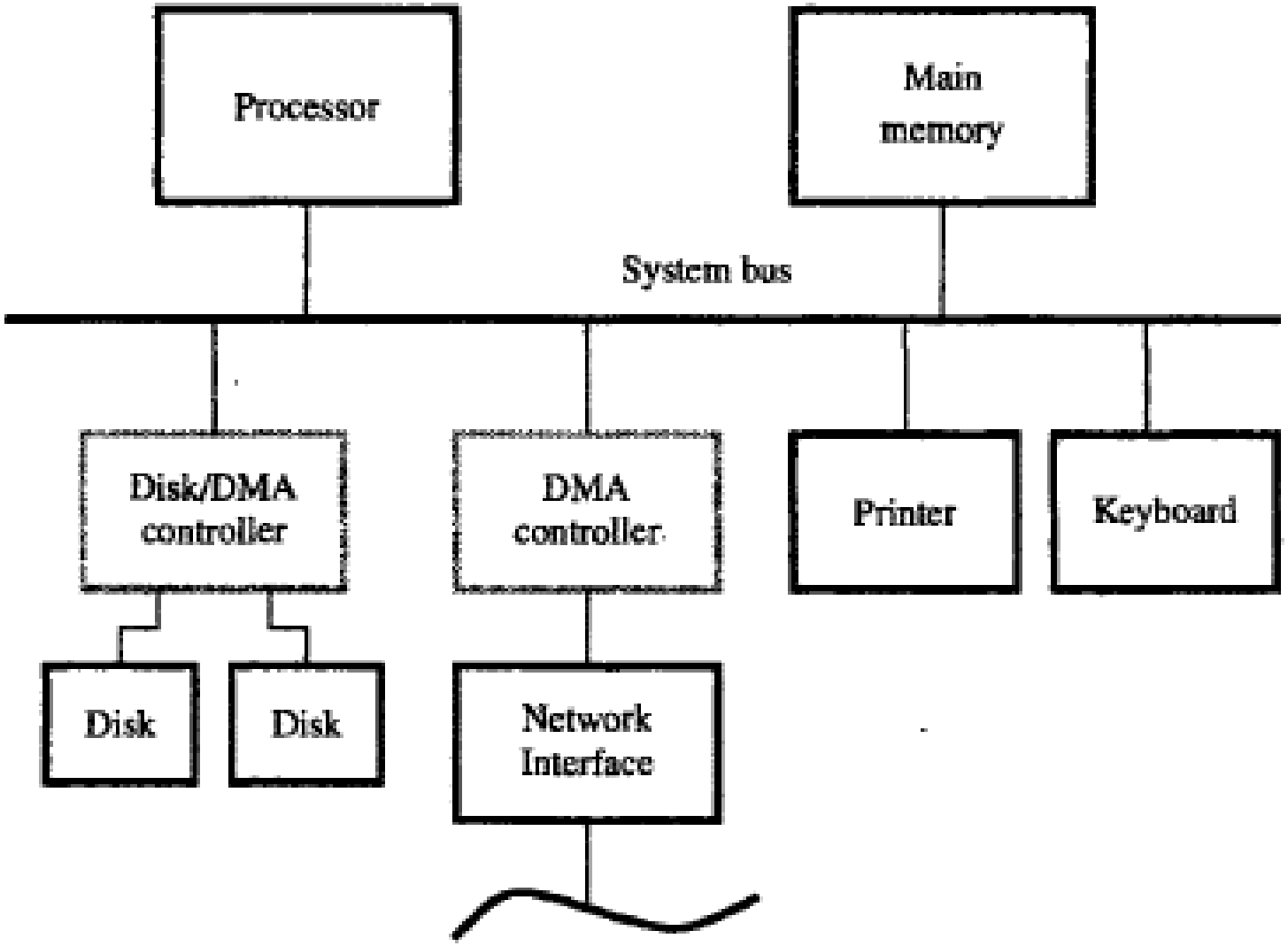
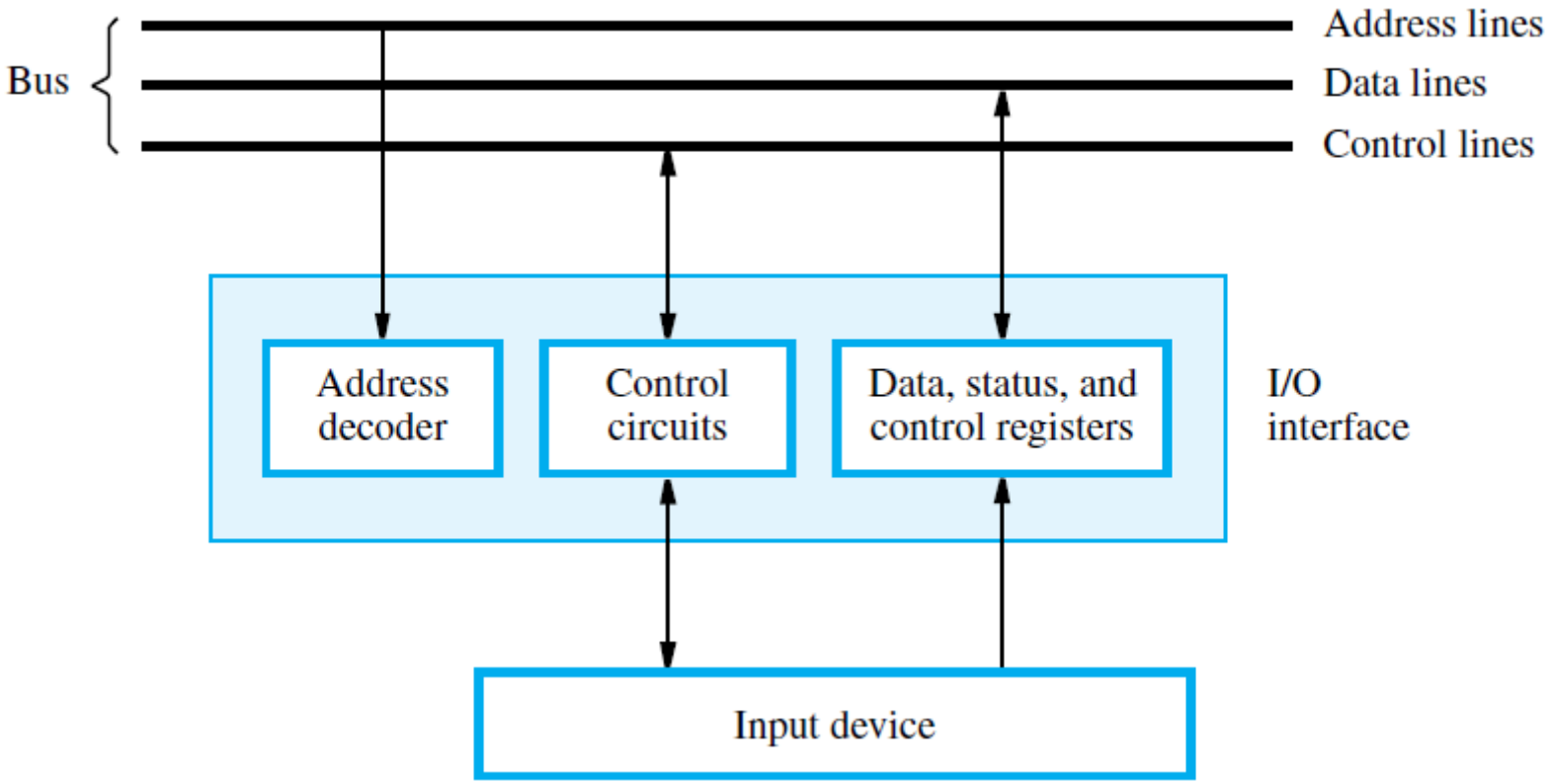


Figure 4.19 Use of DMA controllers in a computer system.

- To initiate the transfer of a block of words, the processor sends to the DMA controller the starting address, the number of words in the block, and the direction of the transfer.
- The DMA controller then proceeds to perform the requested operation.
- When the entire block has been transferred, it informs the processor by raising an interrupt.
- Figure shows an example of the DMA controller registers that are accessed by the processor to initiate data transfer operations.
- Two registers are used for storing the starting address and the word count.
- The third register contains status and control flags. The R/W bit determines the direction of the transfer.
- When this bit is set to 1 by a program instruction, the controller performs a Read operation, that is, it transfers data from the memory to the I/O device. Otherwise, it performs a Write operation

- When the controller has completed transferring a block of data and is ready to receive another command, it sets the Done flag to 1. Bit 30 is the Interrupt-enable flag, IE.
- When this flag is set to 1, it causes the controller to raise an interrupt after it has completed transferring a block of data.
- Finally, the controller sets the IRQ bit to 1 when it has requested an interrupt

Buses and I/O devices Interfaces



- 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, as shown in 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.

4. Bus Operation

4.1 Synchronous Bus

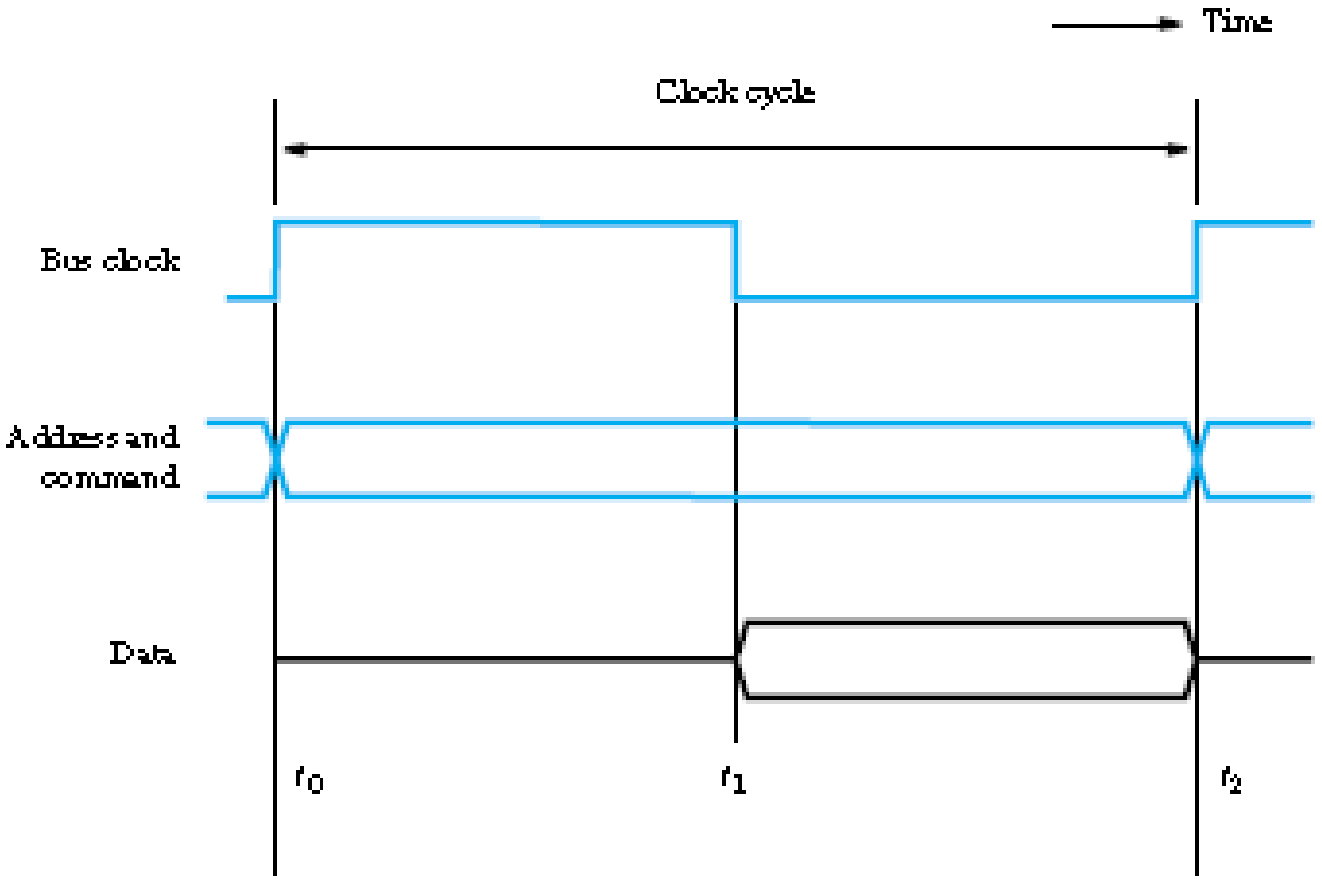


Figure 7.3 Timing of an input transfer on a synchronous bus.

- On a *synchronous* bus, all devices derive timing information from a control line called the *bus clock*, shown at the top of Figure .
- The signal on this line has two *phases*: a high level followed by a low level.
- The two phases constitute a *clock cycle*.
- The first half of the cycle between the low-to-high and high-to-low transitions is often referred to as a clock pulse.
- The address and data lines in Figure are shown as if they are carrying both high and low signal levels at the same time.
- The crossing points indicate the times at which these patterns change

- Let us consider the sequence of 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.
- Information travels over the bus at a speed determined by its physical and electrical characteristics.
- The clock pulse width, $t1 - t0$, must be longer than the maximum propagation delay over the bus.
- Also, it must be long enough to allow all devices to decode the address and control signals, so that the addressed device (the slave) can respond at time $t1$ by placing the requested input data on the data lines.
- At the end of the clock cycle, at time $t2$, the master loads the data on the data lines into one of its registers.
- To be loaded correctly into a register, data must be available for a period greater than the setup time of the register (see Appendix A).
- Hence, the period $t2 - t1$ must be greater than the maximum propagation time on the bus plus the setup time of the master's register.
- A similar procedure is followed for a Write operation

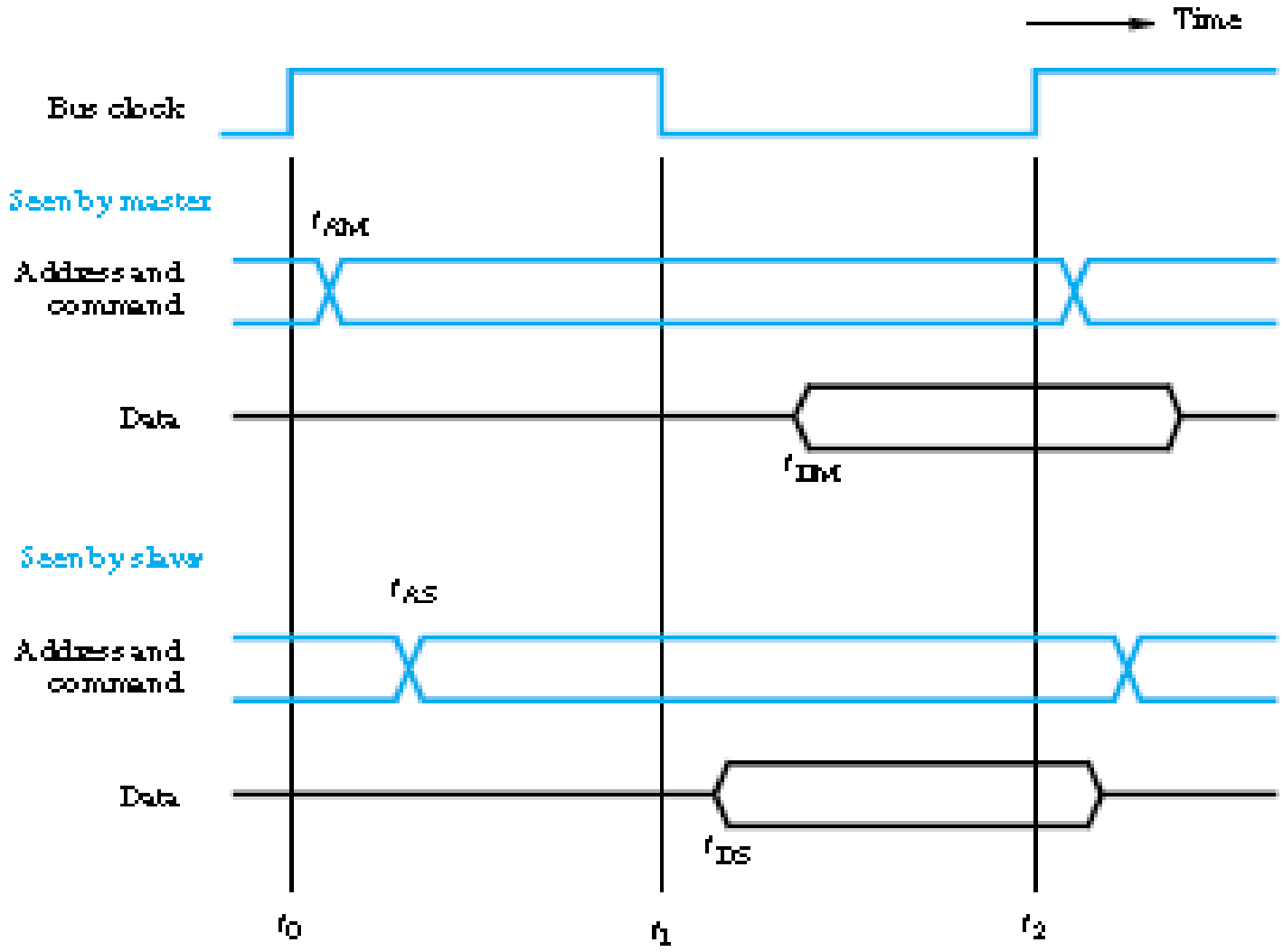


Figure 7.4 A detailed timing diagram for the input transfer of Figure 7.3.

Multiple-Cycle Data Transfer

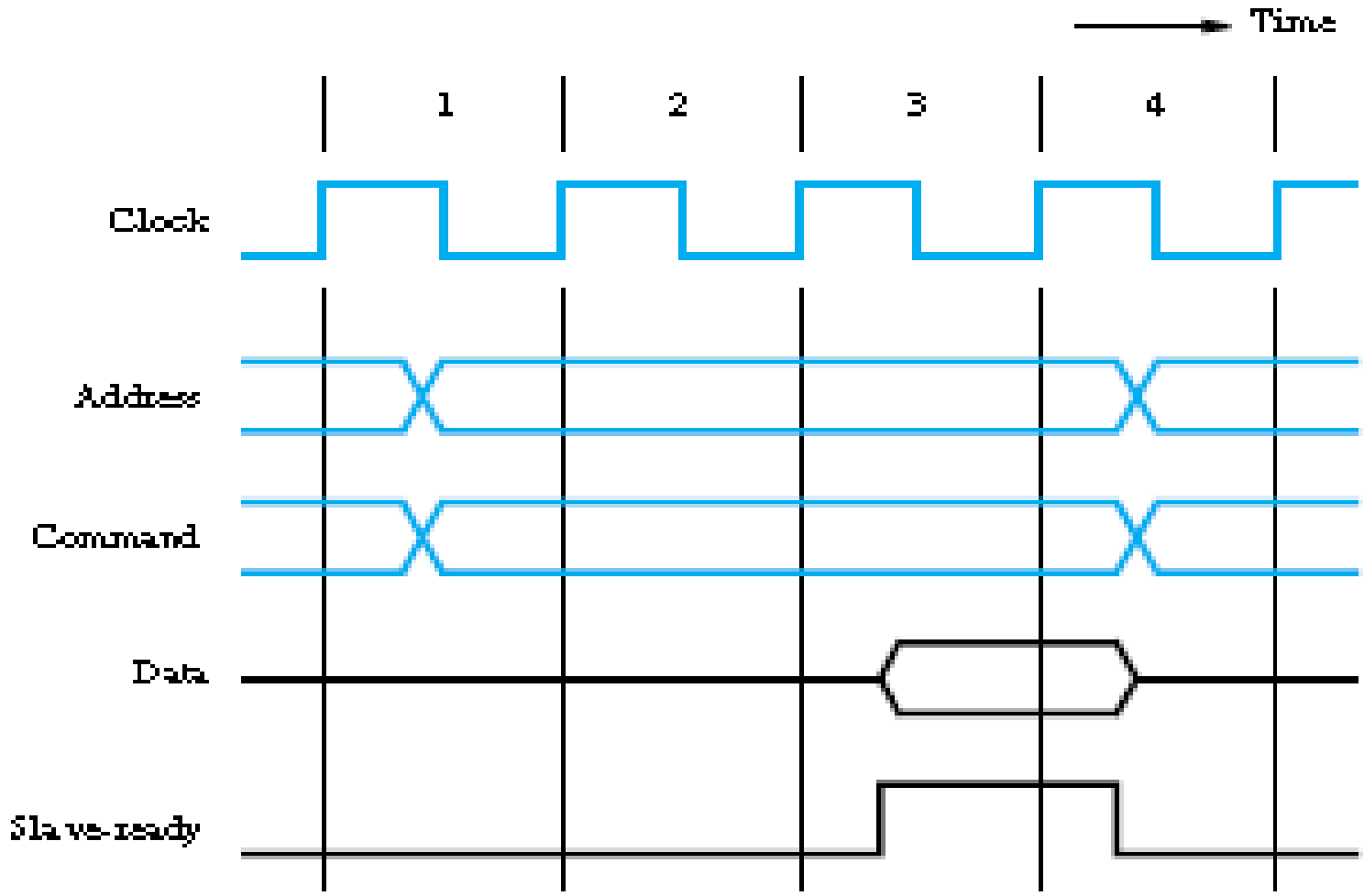


Figure 7.5 An input transfer using multiple clock cycles.

Asynchronous Bus

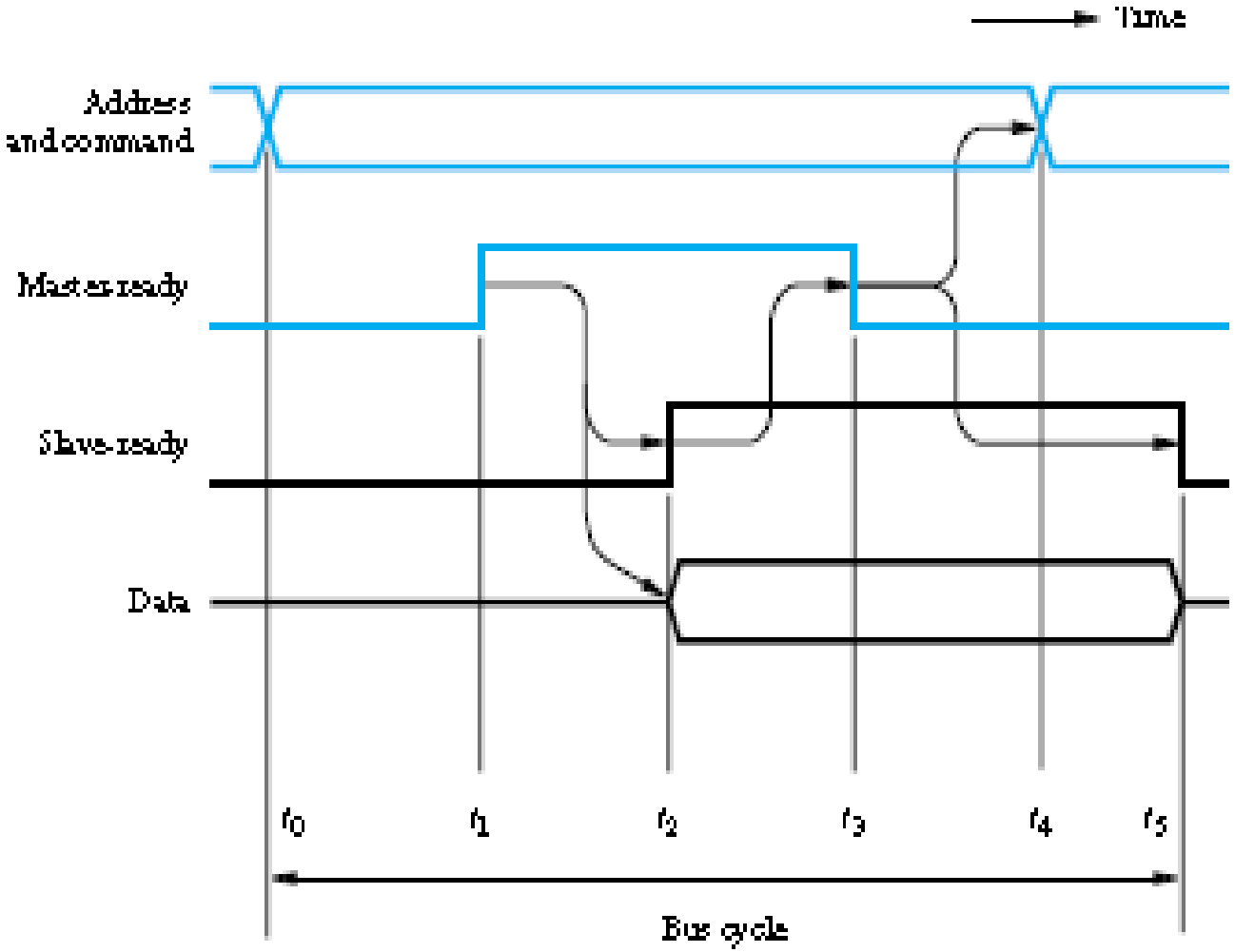


Figure 7.6 Handshake control of data transfer during an input operation.

Asynchronous Bus

- An alternative scheme for controlling 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
- An example of the timing of an input data transfer using the handshake protocol is given in , which figure depicts the following sequence of events:
- $t0$—The master places the address and command information on the bus, and all devices on the bus decode this information.
- $t1$—The master sets the Master-ready line to 1 to inform the devices that the address and command information is ready. The delay $t1 - t0$ is intended to allow for any *skew* that may occur on the bus. This happens because different lines of the bus may have different propagation speeds.

- *t*2—The selected slave, having decoded the address and command information, performs the required input operation by placing its data on the data lines. At the same time, it sets the Slave-ready signal to 1. If extra delays are introduced by the interface circuitry before it places the data on the bus, the slave must delay the Slave-ready signal accordingly..
- *t*3—The Slave-ready signal arrives at the master, indicating that the input data are available on the bus. The master must allow for bus skew. It must also allow for the setup time needed by its register. After a delay equivalent to the maximum bus skew and the minimum setup time, the master loads the data into its register. Then, it drops the Master-ready signal, indicating that it has received the data.
- *t*4—The master removes the address and command information from the bus. The delay between *t*3 and *t*4 is again intended to allow for bus skew. Erroneous addressing may take place if the address, as seen by some device on the bus, starts to change while the Master-ready signal is still equal to 1.
- *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.

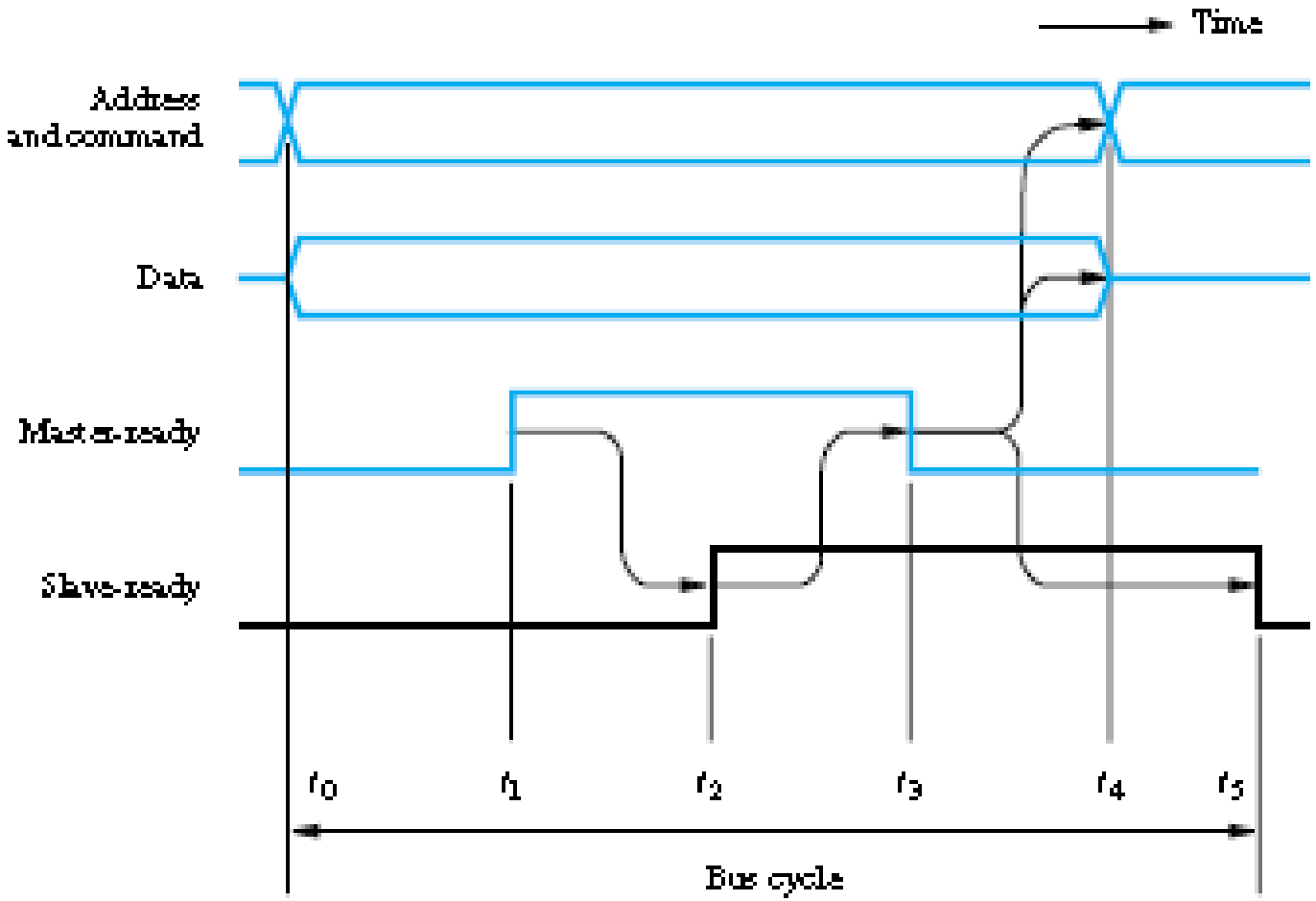


Figure 7.7 Handshake control of data transfer during an output operation.

- **Synchronous buses**
- Use a common clock to synchronize all devices on the bus, which allows for faster data transfer. However, all devices must use the same clock rate, and the bus must be short to avoid clock skew. Synchronous buses are easier to implement than asynchronous buses because they don't use handshaking.
- **Asynchronous buses**
- Don't use a clock, so each device has its own timing mechanism. Asynchronous buses use a handshaking protocol and additional control lines to communicate. Asynchronous buses are simpler and cheaper to implement than synchronous buses, but they offer lower data transfer rates

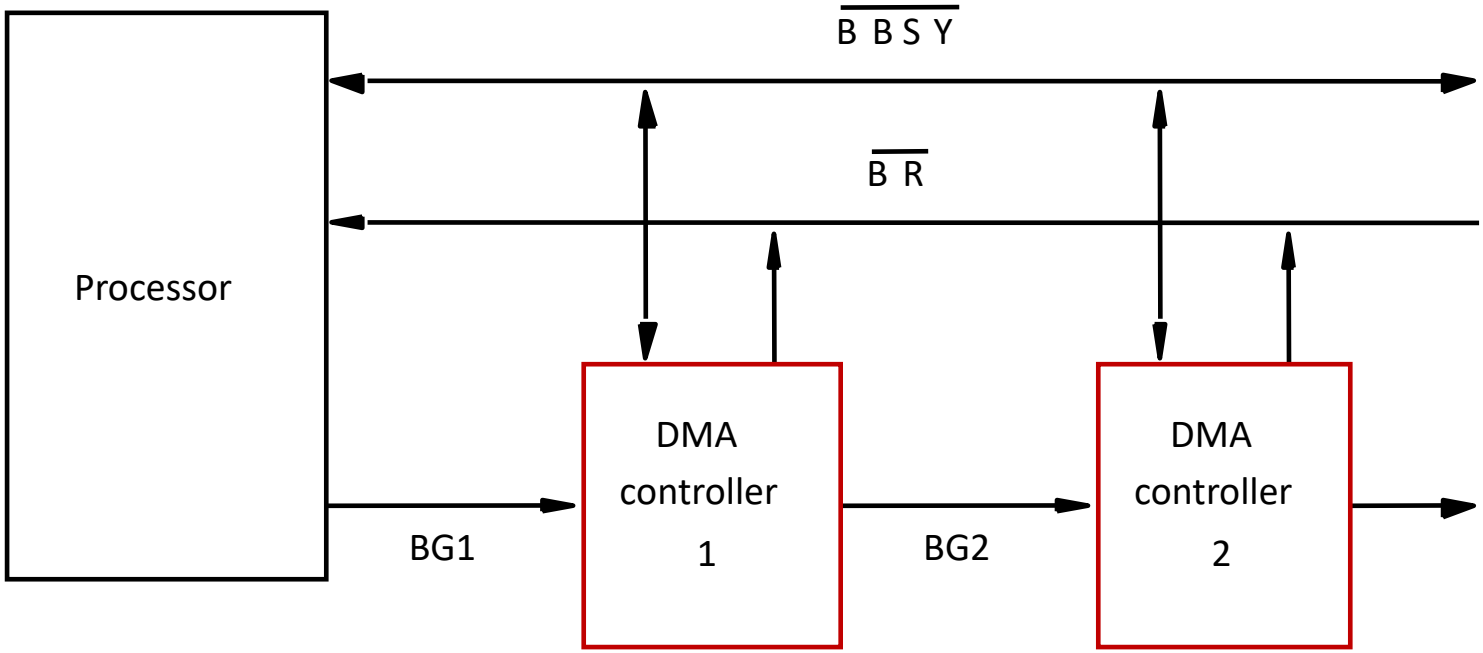
Arbitration

- 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.

Bus arbitration

- Processor and DMA controllers both need to initiate data transfers on the bus and access main memory.
- The device that is allowed to initiate transfers on the bus at any given time is called the bus master.
- When the current bus master relinquishes its status as the bus master, another device can acquire this status.
  - The process by which the next device to become the bus master is selected and bus mastership is transferred to it is called bus arbitration.
- Centralized arbitration:
  - A single bus arbiter performs the arbitration.
- Distributed arbitration:
  - All devices participate in the selection of the next bus master.

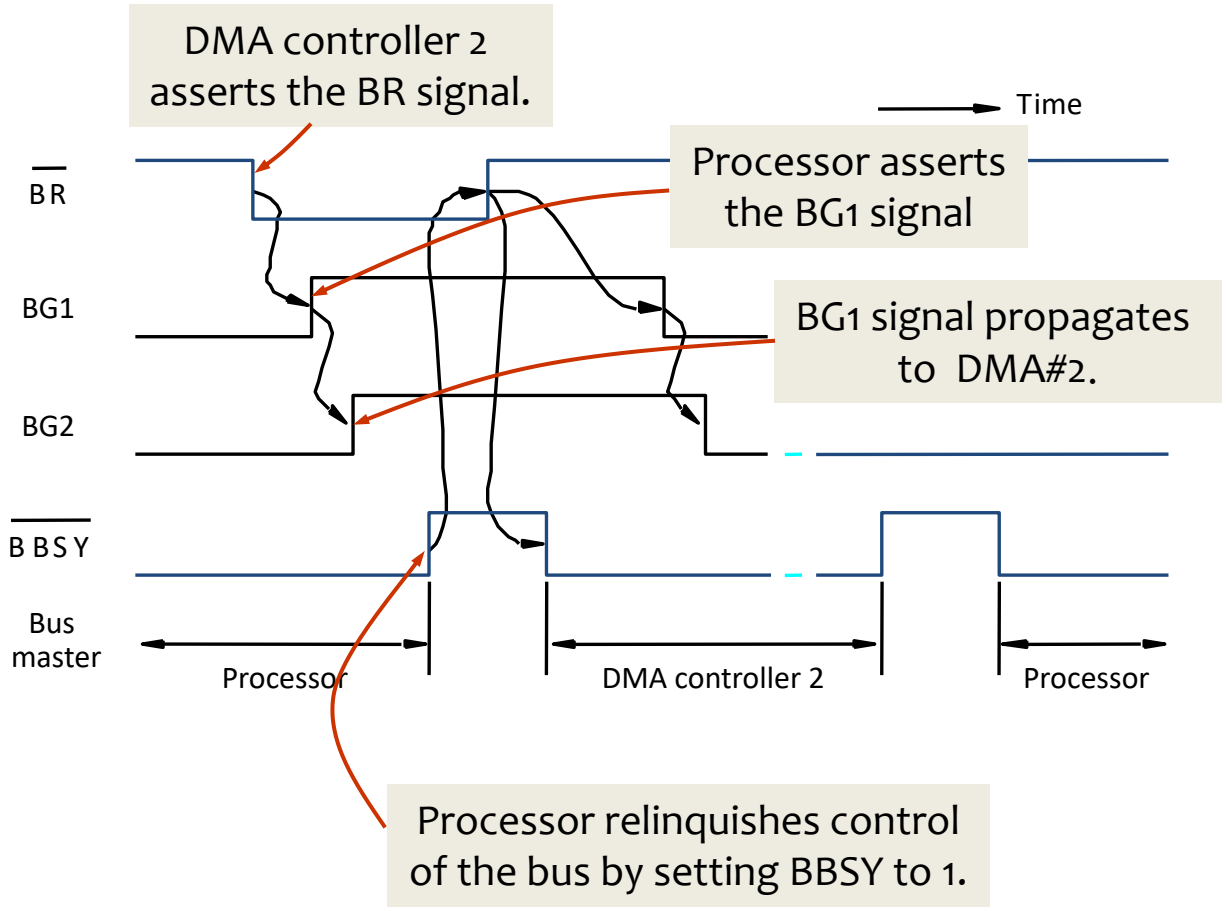
Centralized Bus Arbitration



Centralized Bus Arbitration(cont.,)

- *Bus arbiter may be the processor or a separate unit connected to the bus.*
- *Normally, the processor is the bus master, unless it grants bus membership to one of the DMA controllers.*
- *DMA controller requests the control of the bus by asserting the Bus Request (BR) line.*
- *In response, the processor activates the Bus-Grant1 (BG1) line, indicating that the controller may use the bus when it is free.*
- *BG1 signal is connected to all DMA controllers in a daisy chain fashion.*
- *BBSY signal is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted BR can acquire control of the bus.*

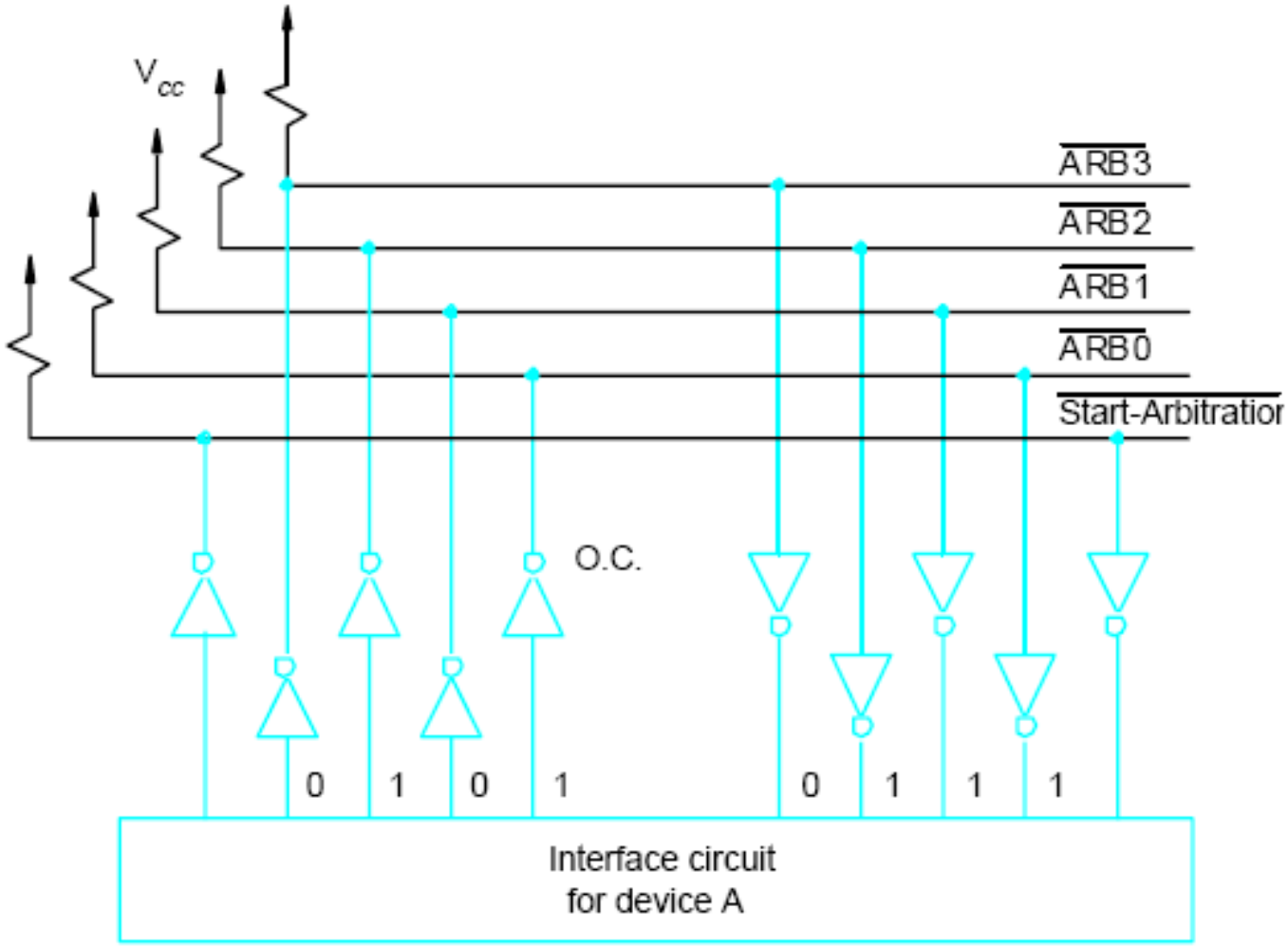
Centralized arbitration (contd..)



Distributed arbitration

- All devices waiting to use the bus share the responsibility of carrying out the arbitration process.
  - Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.
- Each device is assigned a 4-bit ID number.
- All the devices are connected using 5 lines, 4 arbitration lines to transmit the ID, and one line for the Start-Arbitration signal.
- To request the bus a device:
  - Asserts the Start-Arbitration signal.
  - Places its 4-bit ID number on the arbitration lines.
- The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.

Distributed arbitration



Distributed arbitration(Contd.,)

- *Arbitration process:*
  - *Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.*
  - *If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.*
  - *The pattern that appears on the arbitration lines is the logical-OR of all the 4-bit device IDs placed on the arbitration lines.*

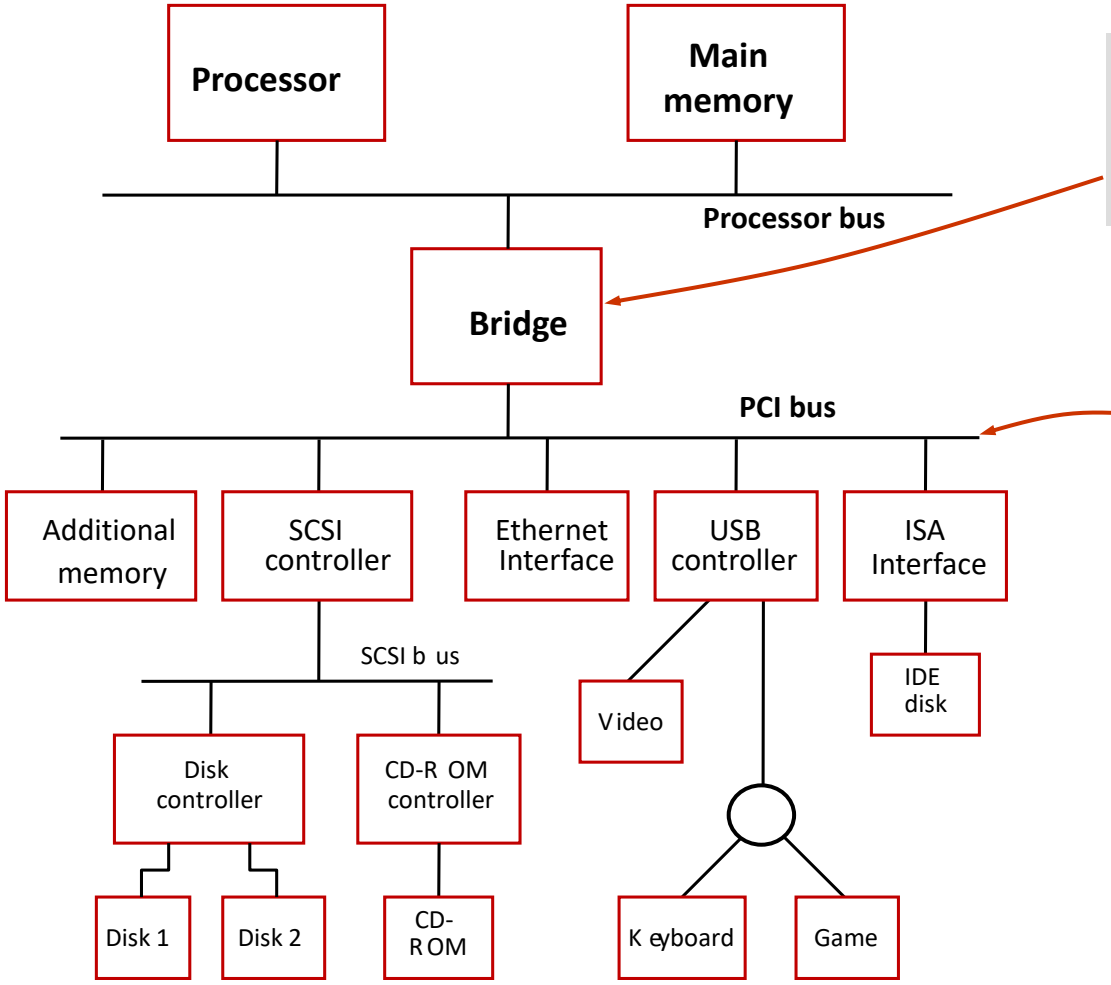
Distributed arbitration (contd..)

- *Device A has the ID 5 and wants to request the bus:*
  - *Transmits the pattern 0101 on the arbitration lines.*
- *Device B has the ID 6 and wants to request the bus:*
  - *Transmits the pattern 0110 on the arbitration lines.*
- *Pattern that appears on the arbitration lines is the logical OR of the patterns:*
  - *Pattern 0111 appears on the arbitration lines.*

Arbitration process:

- *Each device compares the pattern that appears on the arbitration lines to its own ID, starting with MSB.*
- *If it detects a difference, it transmits 0s on the arbitration lines for that and all lower bit positions.*
- *Device A compares its ID 5 with a pattern 0101 to pattern 0111.*
- *It detects a difference at bit position 0, as a result, it transmits a pattern 0100 on the arbitration lines.*
- *The pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.*
- *This pattern is the same as the device ID of B, and hence B has won the arbitration.*

Standard I/O interfaces (contd..)



Bridge circuit translates signals and protocols from processor bus to PCI bus.

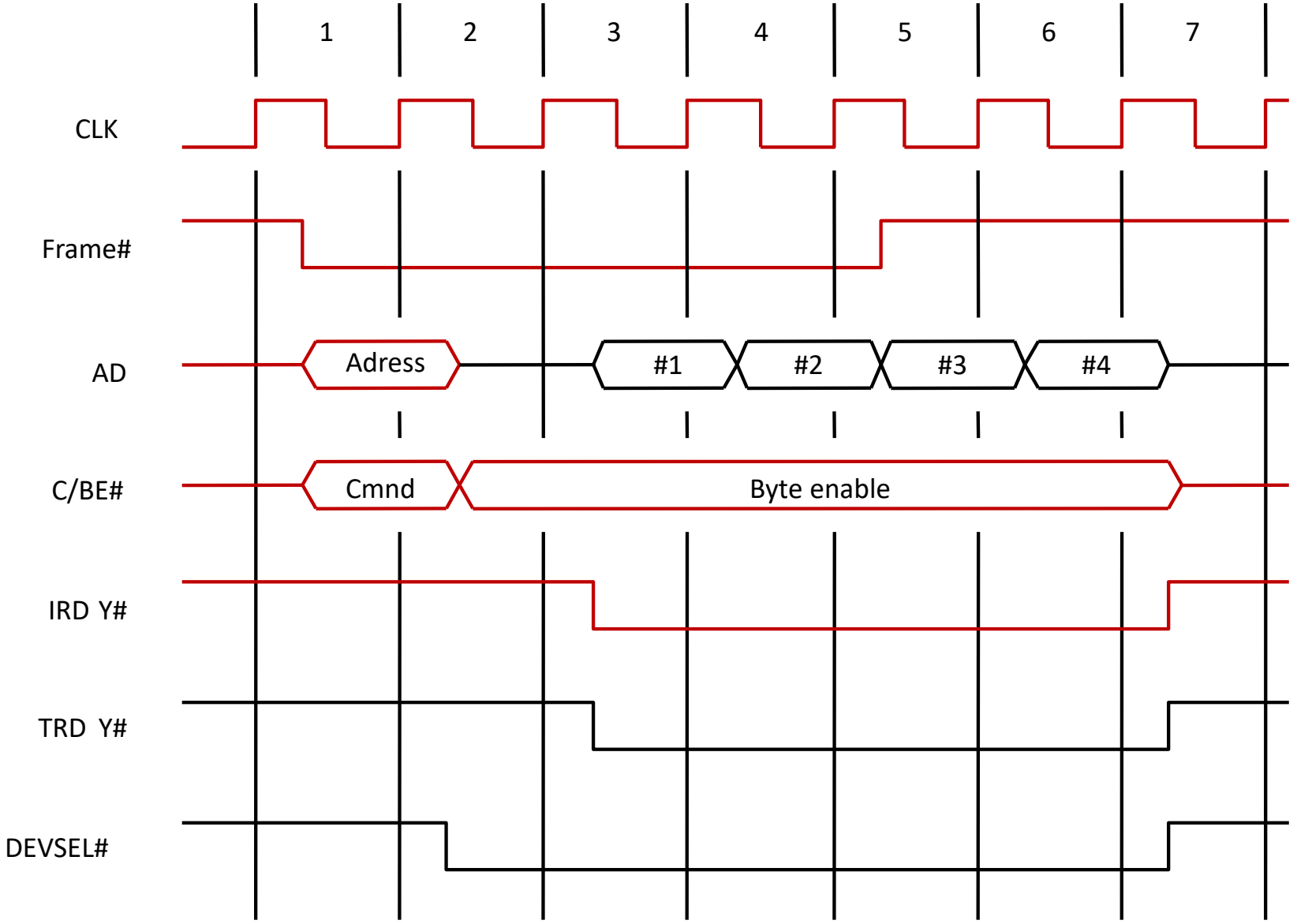
Expansion bus on the motherboard

PCI Bus

- *Peripheral Component Interconnect*
- Introduced in 1992
- Low-cost bus
- Processor independent
- Plug-and-play capability
- In today's computers, most memory transfers involve a burst of data rather than just one word. The PCI is designed primarily to support this mode of operation.
- The bus supports three independent address spaces: memory, I/O, and configuration.
- we assumed that the master maintains the address information on the bus until data transfer is completed. But, the address is needed only long enough for the slave to be selected. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for sending data in subsequent clock cycles. The result is a significant cost reduction.
- A master is called an initiator in PCI terminology. The addressed device that responds to read and write commands is called a target.

Data transfer signals on the PCI bus.

Name	F unction
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
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).
IRD Y#, TRD Y#	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.



A read operation on the PCI bus

Device Configuration

- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
- PCI incorporates in each I/O device interface a small configuration ROM memory that stores information about that device.
- The configuration ROMs of all devices are accessible in the configuration address space. The PCI initialization software reads these ROMs and determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn bout various device options and characteristics.
- Devices are assigned addresses during the initialization process.
- This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one.
- Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#
- Electrical characteristics:
  - PCI bus has been defined for operation with either a 5 or 3.3 V power supply

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) under the designation X3.131 .
- In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to 5 megabytes/s.
- The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years.
- SCSI-2 and SCSI-3 have been defined, and each has several options.
- Because of various options SCSI connector may have 50, 68 or 80 pins.

SCSI Bus (Contd.,)

- Devices connected to the SCSI bus are not part of the address space of the processor
- The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa.
- A packet may contain a block of data, commands from the processor to the device, or status information about the device.
- A controller connected to a SCSI bus is one of two types – an initiator or a target.
- An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. The disk controller operates as a target. It carries out the commands it receives from the initiator.
- The initiator establishes a logical connection with the intended target.
- Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.
- While a particular connection is suspended, other device can use the bus to transfer information.
- This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.

SCSI Bus (Contd.,)

- Data transfers on the SCSI bus are always controlled by the target controller.
- To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.
- Then the controller starts a data transfer operation to receive a command from the initiator.

SCSI Bus (Contd.,)

- Assume that processor needs to read block of data from a disk drive and that data are stored in 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, acting as an initiator, contends for control of the bus.
  2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
  3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
  4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
  5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.

SCSI Bus (Contd.,)

6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed

Operation of SCSI bus from H/W point of view

Category	Name	Function
Data	– DB(0) to – DB(7)	Data lines: Carry one byte of information during the information transfer phase and iden tify device during arbitration, selection and reselection phases
	– DB(P)	Parity bit for the data bus
Phase	– BSY	Busy: Asserted when the bus is not free
	– SEL	Selection: Asserted during selection and reselection
Information type	– C/D	Control/Data: Asserted during transfer of con trol information (command, status or message)
	– MSG	Message: indicates that the information being transferred is a message

Table 4. The SCSI bus signals.

Table 4. The SCSI bus signals.(*cont*.)

Category	Name	Function
Handshake	− REQ	Request: Asserted by a target to request a data transfer cycle
	− ACK	Acknowledge: Asserted by the initiator when it has completed a data transfer operation
Direction of transfer	− I/O	Input/Output: Asserted to indicate an input operation (relative to the initiator)
Other	− ATN	Attention: Asserted by an initiator when it wishes to send a message to a target
	− RST	Reset: Causes all device controls to disconnect from the bus and assume their start-up state

Main Phases involved

- Arbitration
  - A controller requests the bus by asserting BSY and by asserting it's associated data line
  - When BSY becomes active, all controllers that are requesting bus examine data lines
- Selection
  - Controller that won arbitration selects target by asserting SEL and data line of target. After that initiator releases BSY line.
  - Target responds by asserting BSY line
  - Target controller will have control on the bus from then
- Information Transfer
  - Handshaking signals are used between initiator and target
  - At the end target releases BSY line
- Reselection

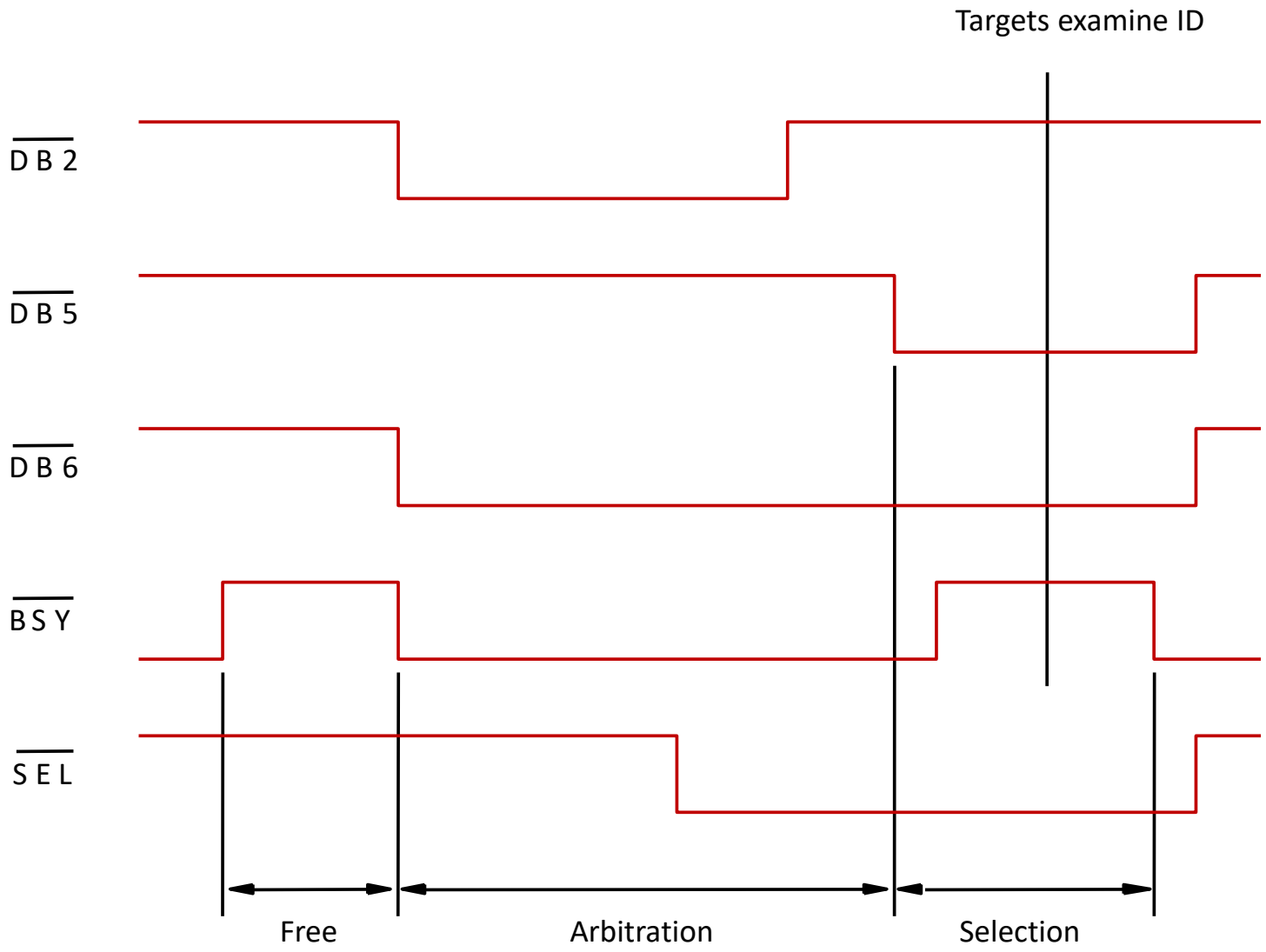
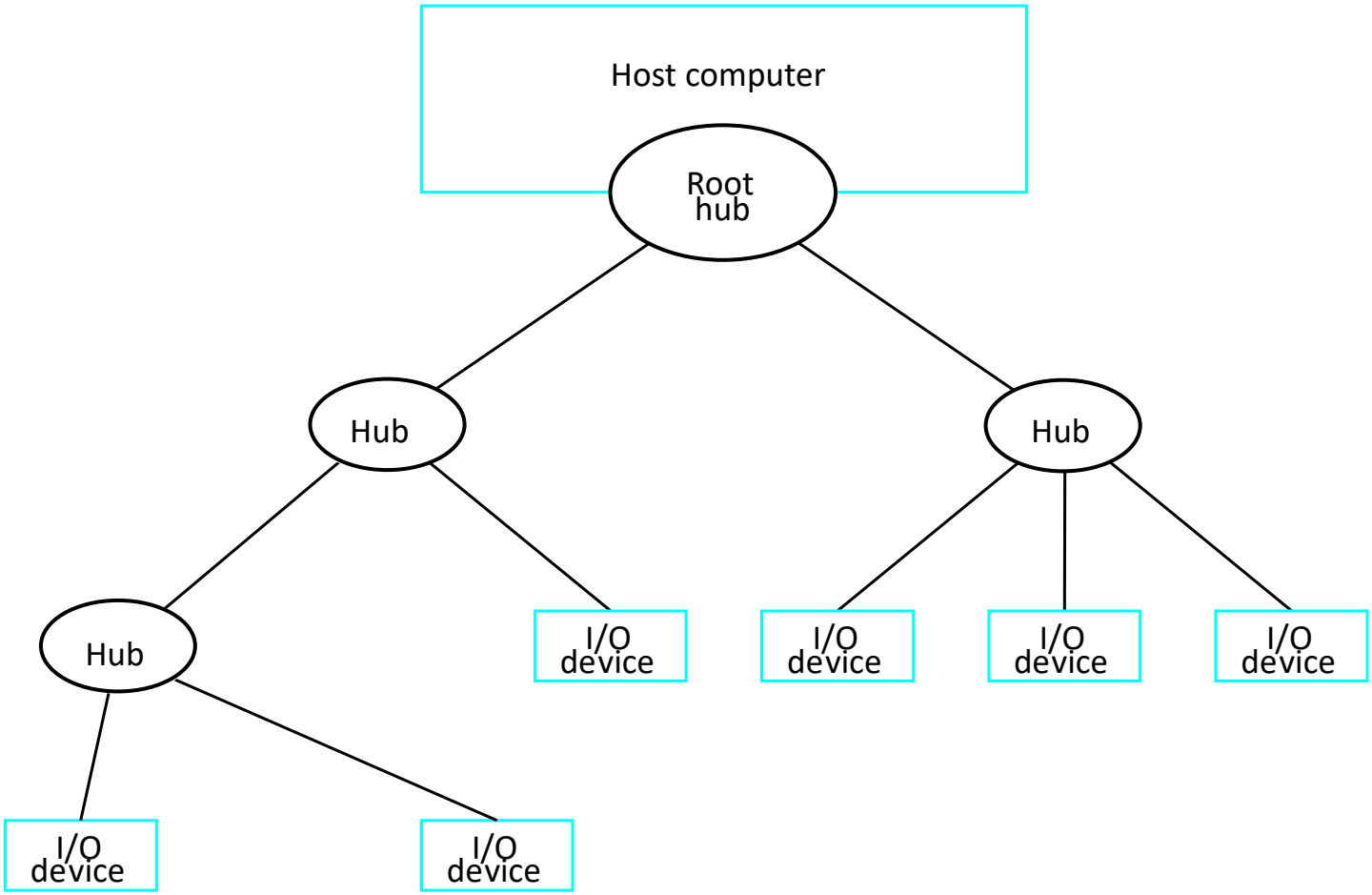


Figure 42. Arbitration and selection on the SCSI bus.
Device 6 wins arbitration and selects device 2.

USB

- Universal Serial Bus (USB) is an industry standard developed through a collaborative effort of several computer and communication companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.
- Speed
  - Low-speed(1.5 Mb/s)
  - Full-speed(12 Mb/s)
  - High-speed(480 Mb/s)
- Port Limitation
- Device Characteristics
- Plug-and-play

Universal Serial Bus tree structure



Universal Serial Bus tree structure

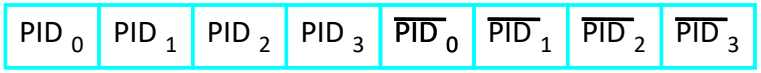
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message. However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

Addressing

- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.
- Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.
- A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily. When a device is first connected to a hub, or when it is powered on, it has the address 0. The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information. Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.
- When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

USB Protocols

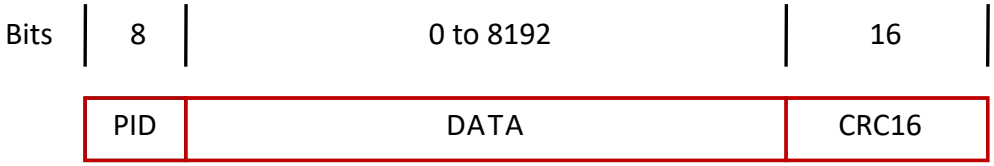
- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information. There are many types of packets that perform a variety of control functions.
- The information transferred on the USB can be divided into two broad categories: control and data.
  - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
  - Data packets carry information that is delivered to a device.
- A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.
- They are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented
- The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte.



(a) Packet identifier field

(b) Token packet, IN or OUT

Control packets used for controlling data transfer operations are called token packets.



(c) Data packet

Figure 45. USB packet format.