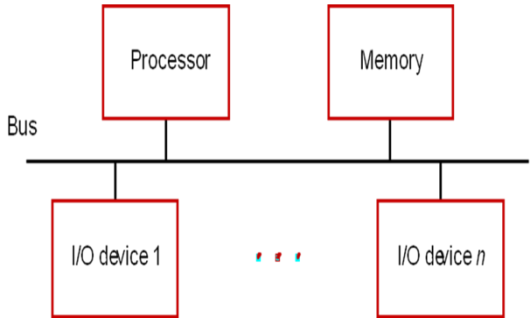
INPUT-OUTPUT ORGANIZATION

INPUT/OUTPUT ORGANIZATION: Accessing I/O Devices, Interrupts: Interrupt Hardware, Enabling and Disabling Interrupts, Handling Multiple Devices, Direct Memory Access, Buses: Synchronous Bus, Asynchronous Bus, Standard I/O Interface: Peripheral Component Interconnect (PCI) Bus, Universal Serial Bus (USB)

Accessing I/O devices

- A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement as shown in the figure below.



- The bus enables all the devices connected to it to exchange information.
- Multiple I/O devices may be connected to the processor and the memory via a bus.
- Bus consists of three sets of lines to carry address, data and control signals.
- Each I/O device is assigned a unique address.
- When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines.
- The processor requests either a read or a write operation and the requested data are transferred over the data lines.
- I/O devices and the memory may share the same address space:
  - Memory-mapped I/O.
  - Any machine instruction that can access memory can be used to transfer data to or from an I/O device.
- I/O devices and the memory may have different address spaces:
  - Special instructions to transfer data to and from I/O devices.
  - I/O devices may have to deal with fewer address lines.

- I/O address lines need not be physically separate from memory address lines.
- In fact, address lines may be shared between I/O devices and memory, with a control signal to indicate whether it is a memory address or an I/O address.

❑ The following figure illustrates the hardware required to connect an I/O device to the bus.

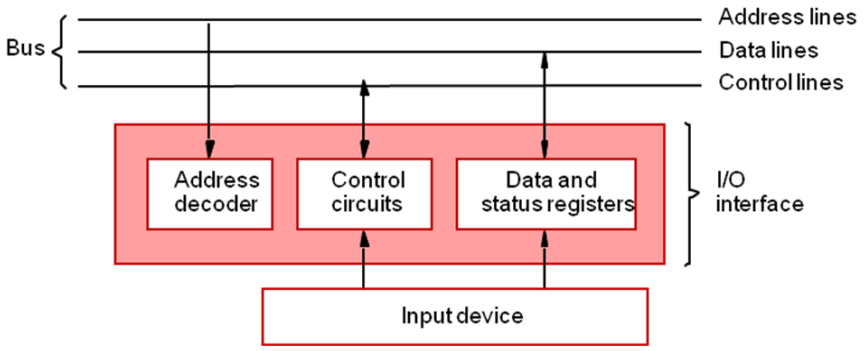


Figure: I/O interface for an input device.

❑ An I/O device is connected to the bus using an I/O interface circuit which has:

- Address decoder, control circuit, data and status registers.

❑ The address decoder decodes the address placed on the address lines thus enabling the device to recognize its address.

❑ The data register holds the data being transferred to or from the processor.

❑ The status register holds information relevant to the operation of the I/O device.

❑ Both data and status registers are connected to the data bus, and assigned unique addresses.

❑ I/O interface circuit coordinates I/O transfers.

❑ Recall that the rate of transfer to and from I/O devices is slower than the speed of the processor. This creates the need for mechanisms to synchronize data transfers between them.

➢ Program-controlled I/O:

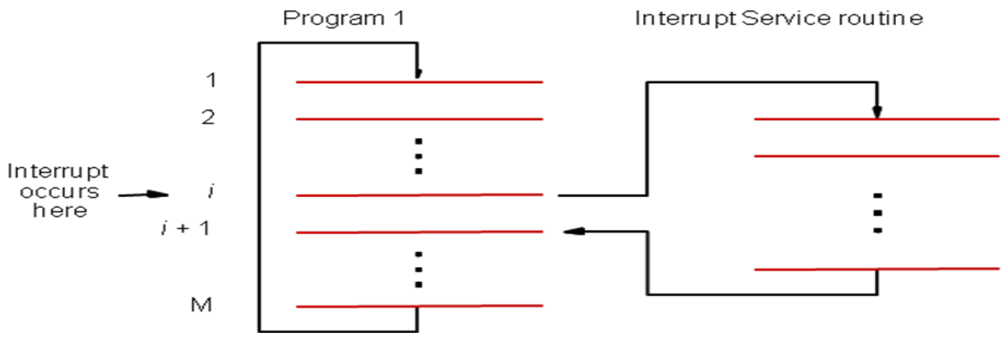
- In this method processor repeatedly monitors a status flag to achieve the required synchronization between the processor and an I/O device.
- Processor polls the I/O device.

➢ 2. There are two other mechanisms used for synchronizing the data transfers between the processor and memory:

- Interrupts.
- Direct Memory Access.

Interrupts

- In program-controlled I/O, the processor continuously monitors the status of the device; it does not perform any useful tasks.
- An alternate approach would be for the I/O device to alert the processor when it becomes ready.
  - Do so by sending a hardware signal called an interrupt to the processor.
  - At least one of the bus control lines, called an ***interrupt-request*** line is dedicated for this purpose.
- Processor can perform other useful tasks in the mean time.



- Processor is executing the instruction located at address ***i*** when an interrupt occurs.
- Routine executed in response to an interrupt request is called the interrupt-service routine.
- When an interrupt occurs, control must be transferred to the interrupt service routine.
- But before transferring control, the current contents of the PC ***(i+1)***, must be saved in a known location.
- This will enable the return-from-interrupt instruction to resume execution at i+1.
- Return address, or the contents of the PC are usually stored on the processor stack.

Treatment of an interrupt-service routine is very similar to that of a subroutine. However there are significant differences:

- A subroutine performs a task that is required by the calling program.
- Interrupt-service routine may not have anything in common with the program it interrupts.
- Interrupt-service routine and the program that it interrupts may belong to different users.
- As a result, before branching to the interrupt-service routine, not only the PC, but other information such as condition code flags, and processor registers used by both the interrupted program and the interrupt service routine must be stored.

- This will enable the interrupted program to resume execution upon return from interrupt service routine.

- Saving and restoring information can be done automatically by the processor or explicitly by program instructions.

- Saving and restoring registers involves memory transfers:
  - Increases the total execution time.
  - 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***.

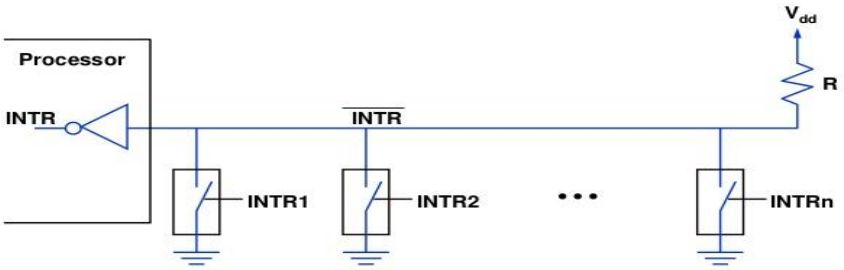
- In order to reduce the interrupt latency, most processors save only the minimal amount of information:
  - This minimal amount of information includes Program Counter and processor status registers.

- Any additional information that must be saved must be saved explicitly by the program instructions at the beginning of the interrupt service routine and restored at the end of the routine.

Interrupt Hardware

- An I/O device requests an interrupt by activating a bus line called ***interrupt-request***.

- Most computers are likely to have several I/O devices that can request an interrupt. A single interrupt –request line may be used to serve ***n*** devices as depicted in the following figure.



- All devices are connected to the line via switches to ground.

- To request an interrupt, a device closes its associated switch. Thus if all interrupt-request signals $INTR_1$ to $INTR_n$ are inactive, that is if all switches are open, the voltage on the interrupt-request line will be equal to V_{dd}. This is the inactive state of the line.

- When a device requests an interrupt by closing its switch the voltage on the line drops to 0, causing the interrupt-request signal, INTR received by the processor to go to 1.

- Since the closing of one or more switches will cause the line voltage to drop to 0,the value of INTR is the logical OR of the requests from individual devices, that is $INTR=INTR_1+\ldots\ldots+INTR_n$

- It is customary to use the complemented form, $\overline{INTR}$ because this signal is active when in the low-voltage state.
- In the electronic implementation of the above circuit special gates known as open-collector or open-drain are used.
- The output of an open-collector or an open-drain gate is equivalent to a switch to ground that is open when the gate's input is in the 0 state and closed when it is in the 1 state.
- Resistor R is called a pull-up resistor because it pulls the line voltage up to the high-voltage state when the switches are open.

Enabling and disabling interrupts

- Interrupt-requests interrupt the execution of a program, and may alter the intended sequence of events:
  - Sometimes such alterations may be undesirable, and must not be allowed.
  - For example, the processor may not want to be interrupted by the same device while executing its interrupt-service routine.
- Processors generally provide the ability to enable and disable such interruptions as desired.
- One simple way is to provide machine instructions such as ***Interrupt-enable*** and ***Interrupt-disable*** for this purpose.
- To avoid interruption by the same device during the execution of an interrupt service routine:
  - First instruction of an interrupt service routine can be ***Interrupt-disable***.
  - Last instruction of an interrupt service routine can be ***Interrupt-enable***.

Handling Multiple Devices

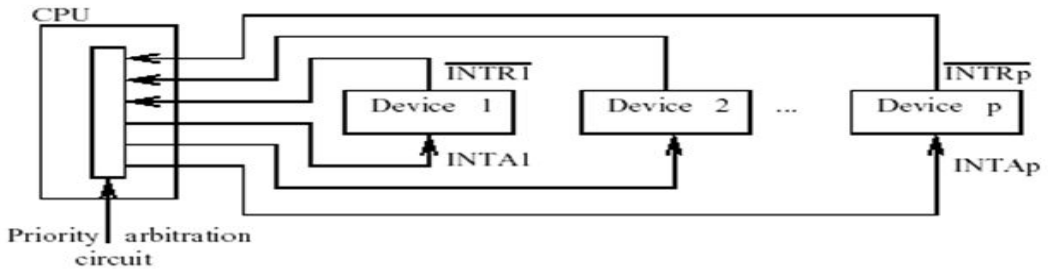
- Multiple I/O devices may be connected to the processor and the memory via a bus. Some or all of these devices may be capable of generating interrupt requests.
  - Each device operates independently, and hence no definite order can be imposed on how the devices generate interrupt requests?
- How does the processor know which device has generated an interrupt?
- How does the processor know which interrupt service routine needs to be executed?
- When the processor is executing an interrupt service routine for one device, can other device interrupt the processor?
- If two interrupt-requests are received simultaneously, then how to break the tie?

- Consider a simple arrangement where all devices send their interrupt-requests over a single control line in the bus.
- When the processor receives an interrupt request over this control line, how does it know which device is requesting an interrupt?

1. This information is available in the status register of the device requesting an interrupt:
    - The status register of each device has an ***IRQ*** bit which it set to 1 when it requests an interrupt.
    - Interrupt service routine can poll the I/O devices connected to the bus. The first device encountered with its ***IRQ*** *bit set (*equal to 1) is the one that should be serviced.
    - Polling mechanism is easy, but time consuming to query the status bits of all the I/O devices connected to the bus.

2. An alternative approach is to use ***vectored interrupts***. The device requesting an interrupt may identify itself directly to the processor.
    - The device can do so by sending a special code (4 to 8 bits) to the processor over the bus.
    - Code supplied by the device may represent a part of the starting address of the interrupt-service routine.
    - The remainder of the starting address is obtained by the processor based on other information such as the range of memory addresses where interrupt service routines are located.

- Which devices can be allowed to interrupt a processor when it is executing an interrupt service routine of another device?
- I/O devices are organized in a priority structure:
    - An interrupt request from a high-priority device is accepted while the processor is executing the interrupt service routine of a low priority device.
- A priority level is assigned to a processor that can be changed under program control.
    - Priority level of a processor is the priority of the program that is currently being executed.
    - When the processor starts executing the interrupt service routine of a device, its priority is raised to that of the device.
    - If the device sending an interrupt request has a higher priority than that of the processor, the processor accepts the interrupt request.
- A multiple-priority scheme can be implemented easily by using separate interrupt-request and interrupt-acknowledge lines for each device, as in the figure given below.



- Each device has a separate ***interrupt-request*** and ***interrupt-acknowledge line***.
- Each interrupt-request line is assigned a different priority level.
- Interrupt requests received over these lines are sent to a priority arbitration circuit in the processor.
- If the interrupt request has a higher priority level than the priority of the processor, then the request is accepted.

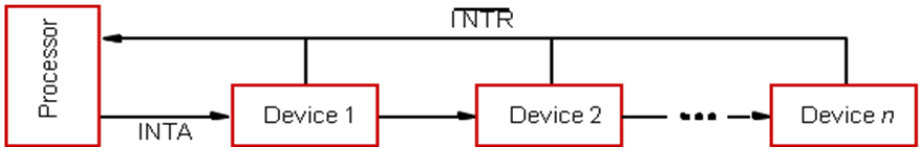
Simultaneous Requests

- Which interrupt request does the processor accept if it receives interrupt requests from two or more devices simultaneously?
- If the I/O devices are organized in a priority structure, the processor accepts the interrupt request from a device with higher priority.
- However, if the devices share an interrupt request line, then how does the processor decide which interrupt request to accept?

1) Polling scheme:

- Polling the status registers of the I/O devices is the simplest mechanism.
- In this case the priority is determined by the order in which the devices are polled.
- The first device with status bit set to 1 is the device whose interrupt request is accepted.

2) Daisy chaining scheme



- Devices are connected in a serial fashion.
- The interrupt-request line **INTR** is common to all devices.
- The interrupt-acknowledge line INTA is connected in a daisy-chain fashion such that the INTA signal propagates serially through the devices.
- When several devices raise an interrupt request and the **INTR** line is activated, the processor responds by setting the INTA line to 1.
- This signal is received by device 1.If device 1 does not need any service; it passes the signal to device 2.
- If device 1 has a pending request for interrupt, it blocks the INTA signal and proceeds to put its identifying code on the data lines.

- In daisy chaining arrangement, the device that is electrically closest to the processor has the highest priority.

Direct Memory Access

❑ To transfer large blocks of data at high speed, a special control unit may be provided to allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor. This approach is called ***direct memory access*** **(DMA)**

❑ DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a ***DMA controller***.

❑ The DMA controller performs functions that would normally be carried out by the processor:

- ◆ For each word transferred, it provides the memory address and all the control signals.
- ◆ To transfer a block of data, it increments the memory addresses and keeps track of the number of transfers.

DMA Controller

❑ DMA controller can transfer a block of data from an external device to the processor, without any intervention from the processor.

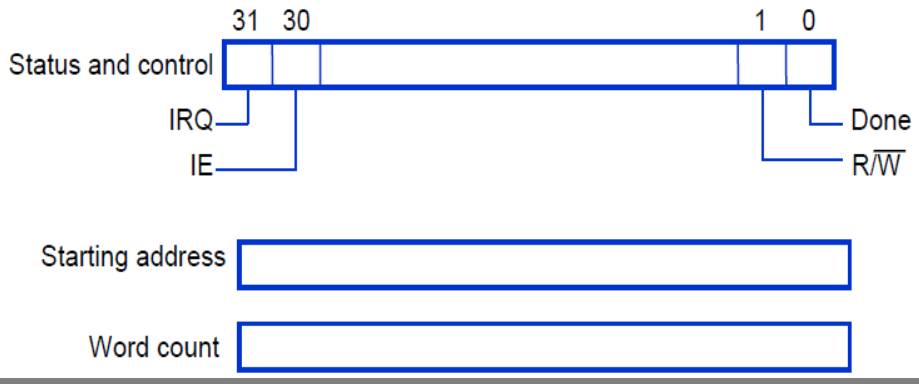
❑ However, the operation of the DMA controller must be under the control of a program executed by the processor. That is, the processor must initiate the DMA transfer.

To initiate the DMA transfer, the processor informs the DMA controller of:

- ◆ Starting address,
- ◆ Number of words in the block.
- ◆ Direction of transfer (I/O device to the memory, or memory to the I/O device).

❑ Once the DMA controller completes the DMA transfer, it informs the processor by raising an interrupt signal.

DMA Controller registers

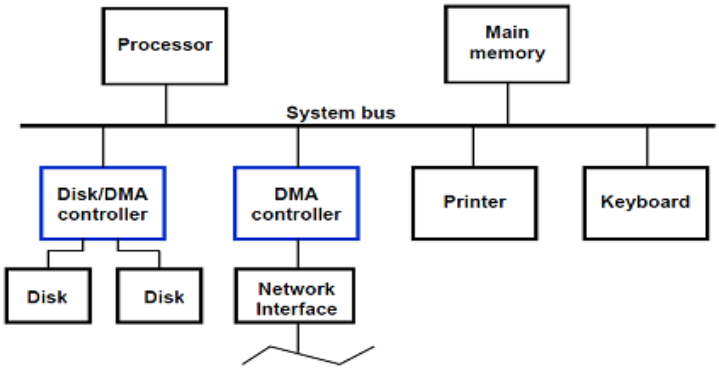


The above figure shows an example of the DMA controller registers that are accessed by the processor to initiate transfer operations.

- 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 data transfer.
  - When the 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.

DMA controller in a computer system

- An example of a computer system is given in the below figure showing how DMA controllers may be used.
  - A DMA controller connects a high speed network to the computer bus.
  - The disk controller, which controls two disks also has DMA capability and provides two DMA channels.
  - It can perform two independent DMA operations as if each disk had its own DMA controller.
  - The registers needed to store the memory address, the word count and so on are duplicated so that one set can be used with each device.



- Memory accesses by the processor and the DMA controllers are interwoven. Requests by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to "steal"

memory cycles from the processor. Hence, this interweaving technique is usually called ***cycle stealing.***

- The DMA controller may transfer a block of data to the main memory without interruption. This is called ***block/burst*** **mode**

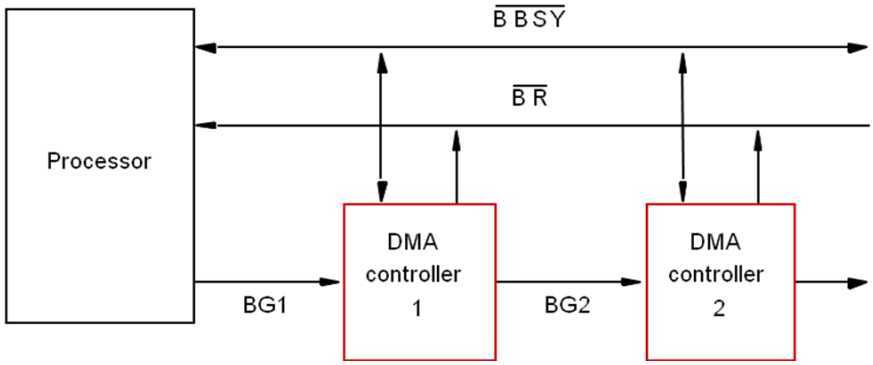
Bus Arbitration

- A conflict may arise if two DMA controllers try to use the bus at the same time to access the main memory.
- To resolve this problem, an arbitration procedure is implemented on the bus.
- The device that is allowed to initiate data transfer on the bus at any given time is called the ***bus master***.
- When the current master relinquishes control of the bus, 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***.

There are two approaches to bus arbitration:

1. Centralized and
2. distributed
   - In centralized arbitration, a single bus arbiter performs the required arbitration.
   - In distributed arbitration, all devices participate in the selection of the next bus master.

Centralized Arbitration



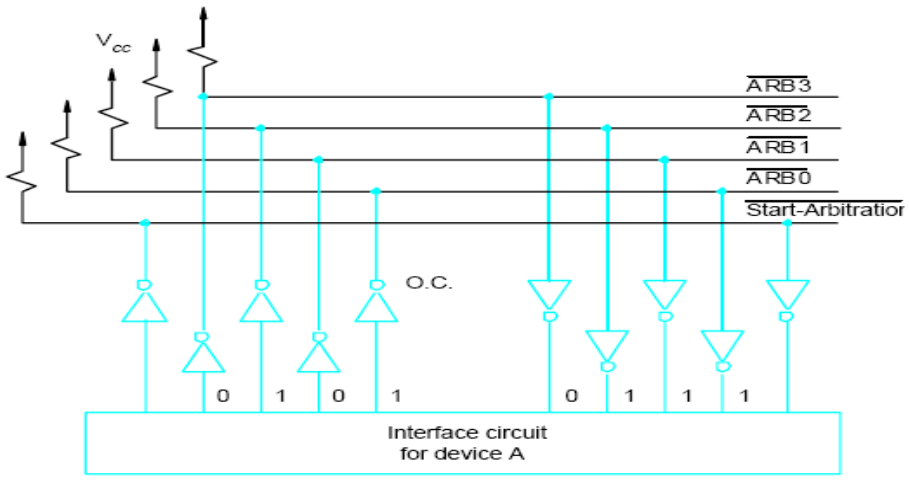
- Bus arbiter may be the processor or a separate unit connected to the bus.
- Normally, the processor is the bus master, unless it grants bus mastership to one of the DMA controllers.
- A DMA controller requests the control of the bus by asserting the Bus Request $\overline{BR}$ line.
- In response, the processor activates the Bus-Grant1 (BG1) line, indicating that the DMA controllers that they may use the bus when it becomes free.
- BG1 signal is connected to all DMA controllers in a daisy chain fashion.
  - If DMA controller 1 is requesting the bus, it blocks the propagation of the grant signal to other devices.

- Otherwise, it passes the grant downstream by asserting BG2.

The current bus master indicates to all the devices that it is using the bus by activating $\overline{BBBSY}$ signal. If BBSY is 0, it indicates that the bus is busy. When BBSY becomes 1, the DMA controller which asserted $\overline{BR}$ can acquire control of the bus.

Distributed Arbitration

- In this method all devices waiting to use the bus share the responsibility in carrying out the arbitration process.
  - Arbitration process does not depend on a central arbiter and hence distributed arbitration has higher reliability.
- Each device on the bus is assigned a 4-bit unique ID (identification) 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 $\overline{\text{Start} - \text{Arbitration}}$ signal.
  - Places its 4-bit ID number on the arbitration lines $\overline{ARB0}$ to $\overline{ARB3}$.



Example:

- Device A has the ID **'5'** and wants to request the bus:
  - It transmits the pattern 0101 on the arbitration lines.
- Device B has the ID **'6'** and wants to request the bus:

- It transmits the pattern 0110 on the arbitration lines.

❑ Pattern that appears on the arbitration lines is the logical OR of the patterns of devices A and B:

- 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 at any bit position, it disables its drivers lines at that bit position and for all lower-order bits.
- It does so by placing a 0 at the input of these drivers.
- Device A compares its ID 5 (0101) with the pattern 0111.
- It detects a difference on the line ARB1.Hence it disables its drivers on the lines ARB1 AND ARB0.This causes the pattern on the arbitration lines to change to0100.
- The current pattern that appears on the arbitration lines is the logical-OR of 0100 and 0110, which is 0110.
- This pattern is same as the ID of device B, and hence B has won the arbitration.

Buses

❑ Processor, main memory, and I/O devices are interconnected by means of a bus.

❑ Bus provides a communication path for the transfer of data.

- Bus also includes lines to support interrupts and arbitration.

❑ A ***bus protocol*** is the set of rules that govern the behavior of various devices connected to the bus, as to when to place information on the bus, when to assert control signals, etc.

❑ Bus lines may be grouped into three types:

- Data
- Address
- Control

❑ Control signals specify:

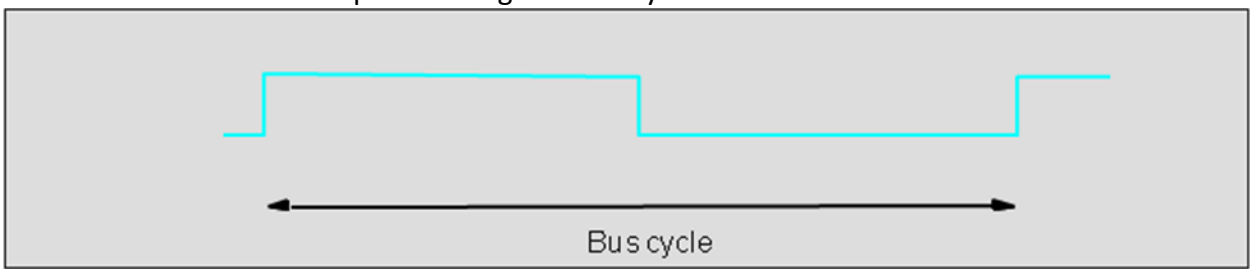
- Whether it is a read or a write operation.
- Required size of the data, when several operand sizes (byte, word, long word) are possible.

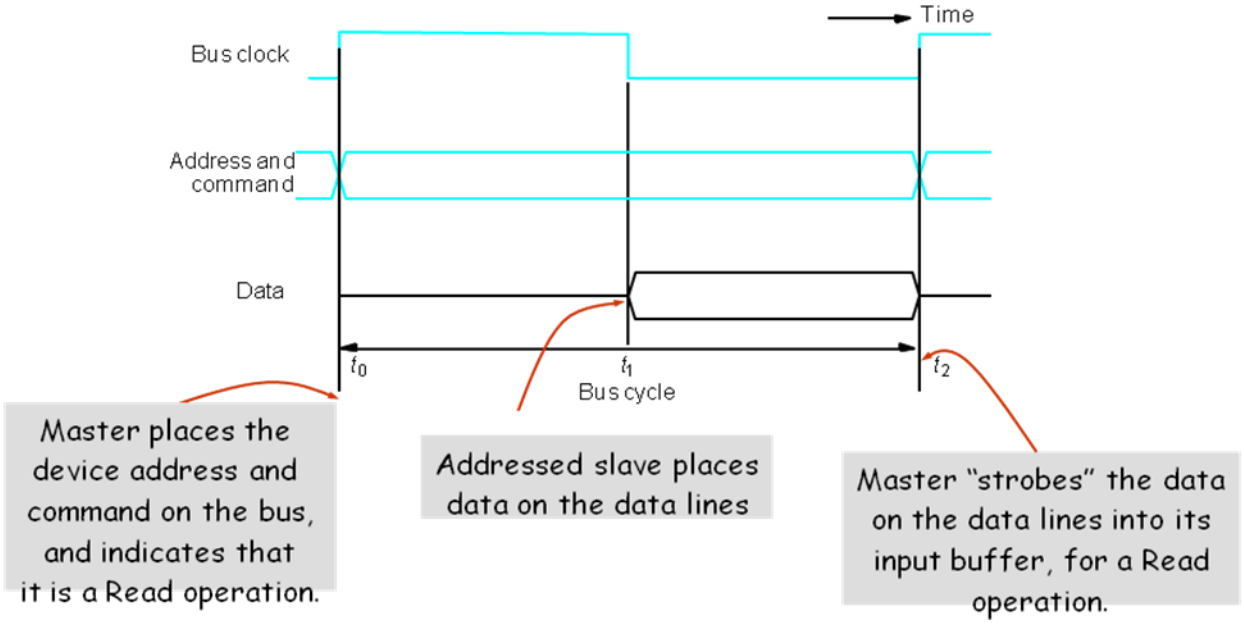
- Timing information to indicate when the processor and I/O devices may place data or receive data from the bus.

- Schemes for timing of data transfers over a bus can be classified into:
  - Synchronous,
  - Asynchronous.

Synchronous Bus

- All devices derive timing information from a common clock line.
- The clock line has equally spaced pulses which define equal time intervals.
  - In a simple synchronous bus, each of these pulses constitutes a bus cycle.
- One data transfer can take place during one bus cycle.

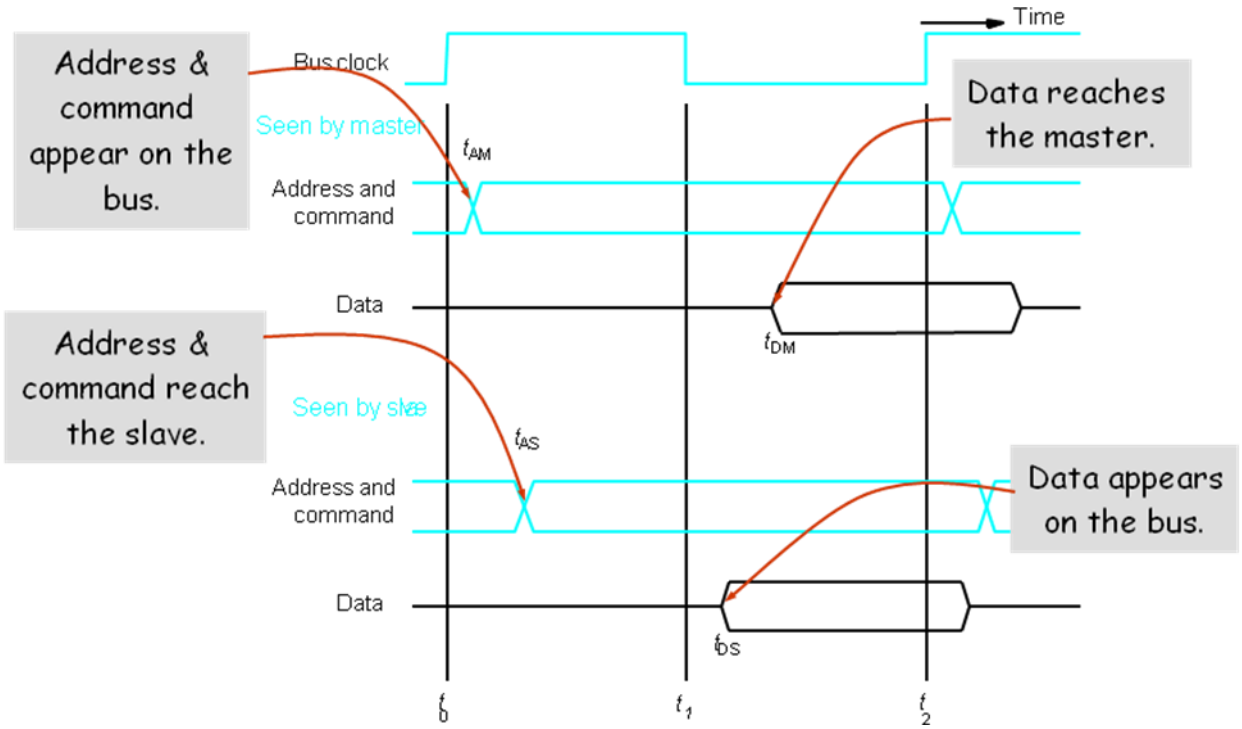




- In case of a Write operation, the master places the data on the bus along with the address and commands.
- The slave strobes the data into its input buffer at time t_2.
  - Once the master places the device address and command on the bus, it takes time for this information to propagate to the devices. This time depends on the physical and electrical

characteristics of the bus. Also, all the devices have to be given enough time to decode the address and control signals, so that the addressed slave can place data on the bus.

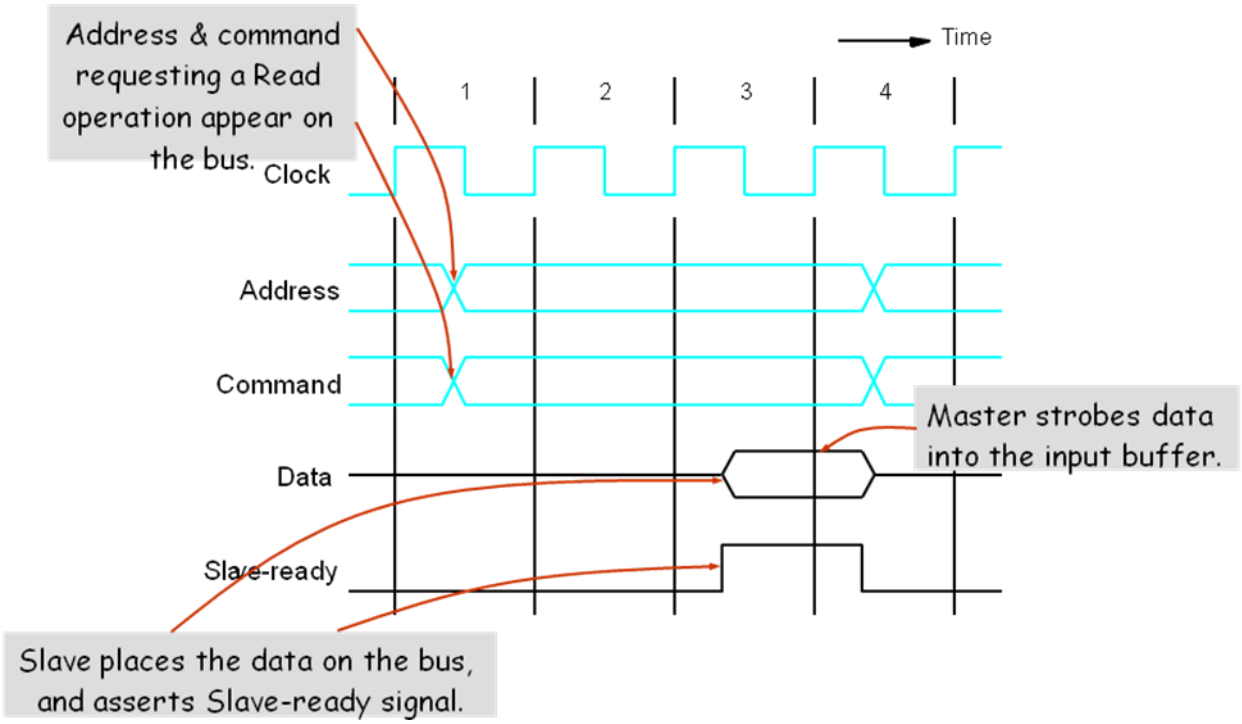
- Width of the pulse $t_1 - t_0$ depends on:
  - Maximum propagation delay between two devices connected to the bus.
  - Time taken by all the devices to decode the address and control signals, so that the addressed slave can respond at time t_1.
- At the end of the clock cycle, at time t_2, the master strobes the data on the data lines into its input buffer if it's a Read operation.
  - "Strobe" means to capture the values of the data and store them into a buffer.
- When data are to be loaded into a storage buffer register, the data should be available for a period longer than the setup time of the device.
- Width of the pulse $t_2 - t_1$ should be longer than:
  - Maximum propagation time of the bus plus
  - Set up time of the input buffer register of the master.



- Data transfer has to be completed within one clock cycle.
  - Clock period ***t2 - t0*** must be such that the longest propagation delay on the bus and the slowest device interface must be accommodated.
  - Forces all the devices to operate at the speed of the slowest device.
- Processor just assumes that the data are available at t2 in case of a Read operation, or are read by the device in case of a Write operation.
  - What if the device is actually failed, and never really responded?
- Most buses have control signals to represent a response from the slave.
- Control signals serve two purposes:

- Inform the master that the slave has recognized the address, and is ready to participate in a data transfer operation.
- Enable to adjust the duration of the data transfer operation based on the speed of the participating slaves.

- High-frequency bus clock is used.

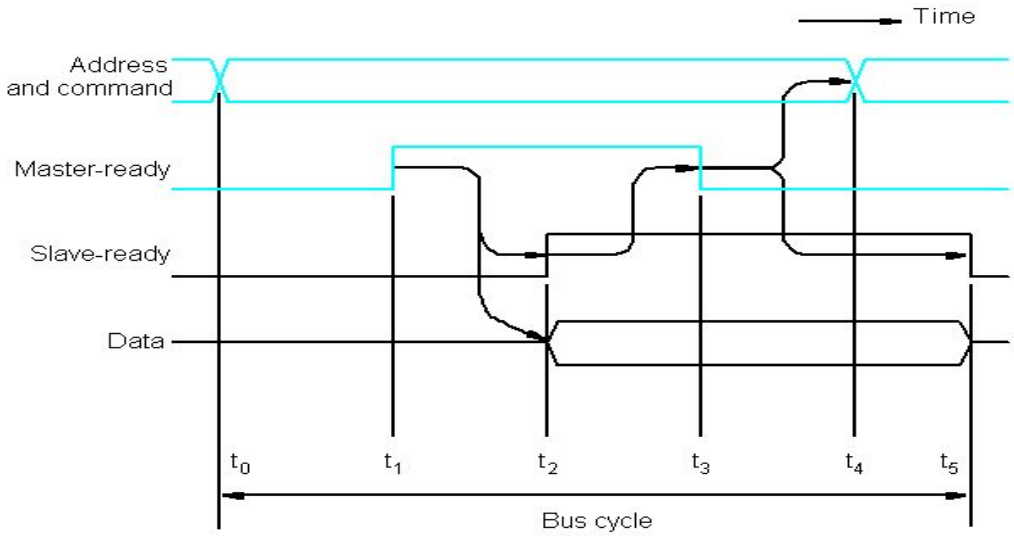


- Clock signal used on the bus is not necessarily the same as the processor clock.
  - Processor clock is much faster than the bus clock.
- Modern processor clocks are typically above 500 MHz.
- Clock frequencies used on the memory and I/O buses may be in the range of 50 to 150 MHz.

Asynchronous Bus

- A data transfer on the bus is controlled by a handshake between the master and the slave.
- Common clock in the synchronous bus case is replaced by two timing control lines:
  - Master-ready,
  - Slave-ready.
- Master-ready signal is asserted by the master to indicate that it is ready to participate in a data transfer.
- Slave-ready signal is asserted by the slave in response to the master-ready from the master, and it indicates to the master that the slave is ready to participate in the data transfer.
- Data transfer using the handshake protocol:
  - Master places the address and command information on the bus.
  - Asserts the Master-ready signal to indicate to the slaves that the address and command information has been placed on the bus.
  - All devices on the bus decode the address.

- Addressed slave performs the required operation, and informs the processor it has done so by asserting the Slave-ready signal.
- Master removes all the signals from the bus, once Slave-ready is asserted.
- If the operation is a Read operation, Master also strobes the data into its input buffer.



- **t_0 -** Master places the address and command information on the bus. Command indicates that it is a read operation that is the data are transferred from the device to the memory.
- **t_1 -** Master asserts the Master-ready signal. Master-ready signal is asserted at t_1 instead of t_0 to allow for bus skew. Bus skew occurs when two signals transmitted simultaneously reach the destination at different times. This may occur because different bus lines may have different speeds. t_1-t_0 should be greater than maximum skew. t_1-t_0 should also include the time taken to decode the address information.
- **t_2 -** Addressed slave places the data on the bus and asserts the Slave-ready signal. The period t_2-t_1, depends on the propagation delay between the master and the slave, and the delay in the slave's interface circuit.
- **t_3 -** Slave-ready signal arrives at the master. Slave-ready signal was placed on the bus at the same time that data were placed on the bus. The master now strobes the data. It also deactivates the Master-ready signal to indicate that it has received the data.
- **t_4 -** Master removes the address and command information. Once Master-ready signal is set to 0, it should reach all the devices before the address and command information is removed from the bus.
- **t_5 -** Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

Standard I/O Interface

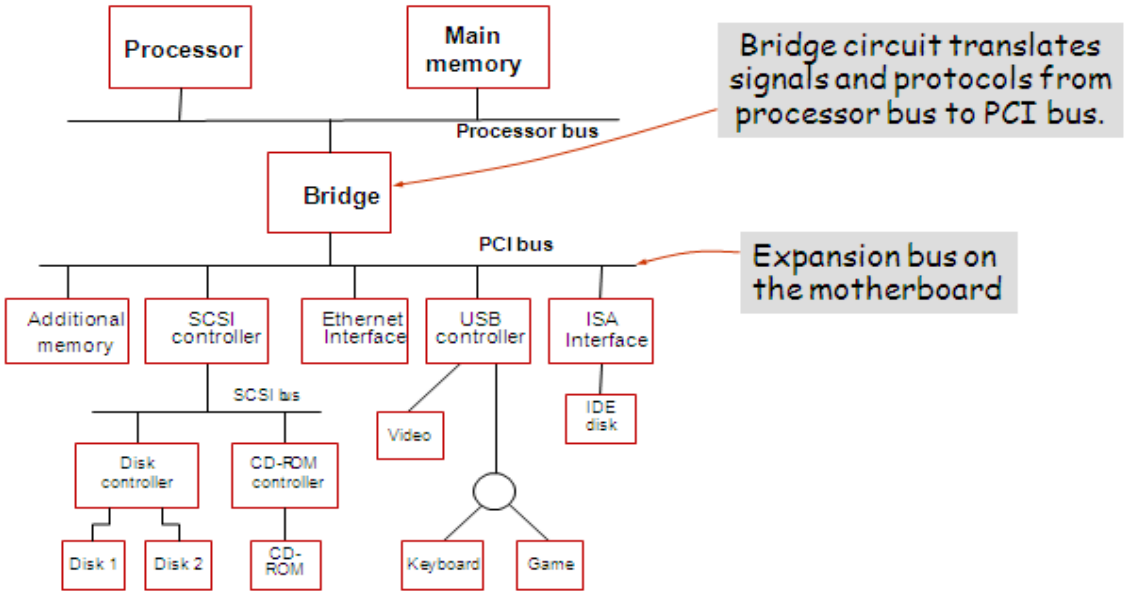
- I/O device is connected to a computer using an interface circuit.
- Do we have to design a different interface for every combination of an I/O device and a computer?
- A practical approach is to develop standard interfaces and protocols.

- A personal computer has:

- A motherboard which houses the processor chip, main memory and some I/O interfaces.
- A few connectors into which additional interfaces can be plugged.

- Processor bus is defined by the signals on the processor chip.
  - Devices which require high-speed connection to the processor are connected directly to this bus.
- Because of electrical reasons only a few devices can be connected directly to the processor bus.
- Motherboard usually provides another bus that can support more devices.
  - Processor bus and the other bus (called as expansion bus) are interconnected by a circuit called "bridge".
  - Devices connected to the expansion bus experience a small delay in data transfers.
- Design of a processor bus is closely tied to the architecture of the processor.
  - No uniform standard can be defined.
- Expansion bus however can have uniform standard defined.
- There are three widely used bus standards:
  - PCI (Peripheral Component Interconnect)
  - SCSI (Small Computer System Interface)
  - USB (Universal Serial Bus)

The way these standards are used in a typical computer system is illustrated in the following figure.



- The PCI standard defines an expansion bus on the mother board.
- SCSI and USB are used for connecting additional devices, both inside and outside the computer box.
- A given computer may use more than one bus standard.

Peripheral Component Interconnect (PCI) Bus

- 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.

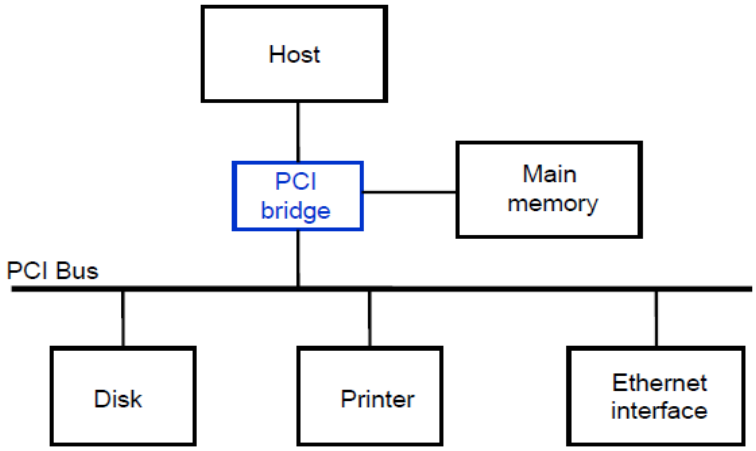


Fig- Use of a PCI bus in a computer system

Data transfer signals on the PCI bus.

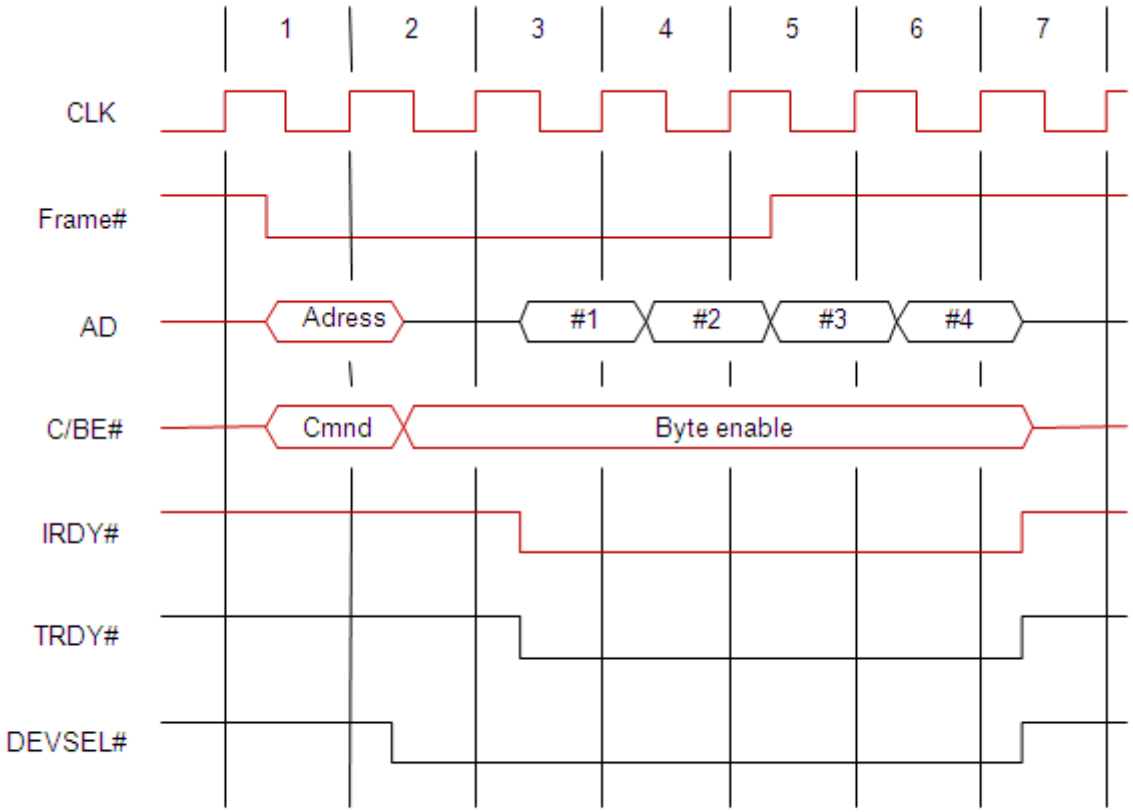
Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRDY#, TRDY#	Initiator-ready and Target-ready signals.
DEVSEL#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

Note: Signals whose name ends with the symbol # are asserted when in the low-voltage state.

- 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***.

Read operation on the PCI bus



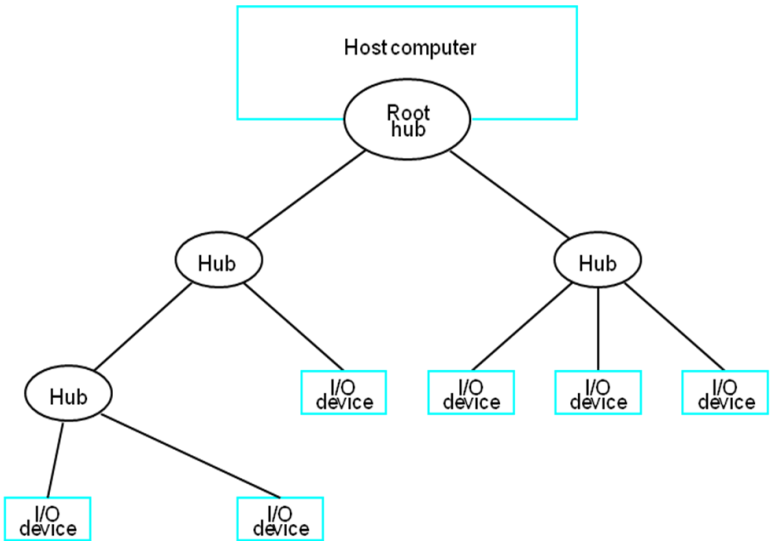
A read operation on the PCI bus

- In clock cycle 1, the processor asserts FRAME# to indicate the beginning of a transaction. At the same time it sends the address on the AD 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.
- Clock cycle 2 is used to turn the AD bus lines around. The processor removes the address and disconnects from the AD lines. The selected target enables its drivers on the AD lines and fetches the requested data to be placed on the bus during clock cycle3.
- During clock cycle 3, the initiator asserts the initiator ready signal, IRDY# to indicate that it is ready to receive data. If the target has data ready to send at this time, it asserts target ready, TRDY# and sends a word of data. The initiator loads the data into its input buffer at the end of the clock cycle.
- The initiator uses the FRAME# signal to indicate the duration of the burst. It negates this signal during the second last word of the transfer. Since it wishes to read four words, the initiator negates FRAME# during clock cycle 5, the cycle in which it receives the third word. After sending the fourth word in clock cycle 6, the target disconnects its drivers and negates DEVSEL# at the beginning of clock cycle7.

Universal Serial Bus (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)
- Plug-and-play
- Simple, low cost bus

USB Tree structure



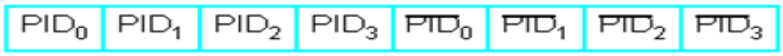
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.
- Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root 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.

USB Protocols

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information.

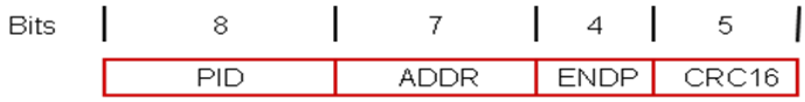
- There are many types of packets that perform a variety of control functions.
- The information transferred on the USB can be divided into two broad categories: *control and data.*
  - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been 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.



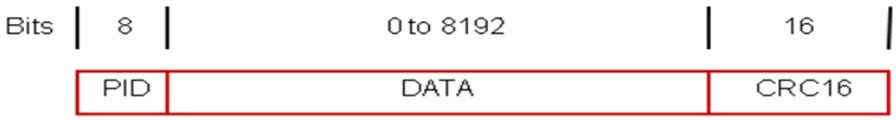
(a) Packet identifier field

- There are four bits of information in this field, but they are transmitted twice.
- The first time they are sent with their true values, and the second time with each bit complemented. This enables the receiving device to verify that the PID byte has been received correctly.
- The four PID bits identify one of 16 different packet types.



(b) Token packet, IN or OUT

- Control packets used for controlling data transfer operations are called ***token packets***.
- A token packet starts with a PID field, using one of two PID values to distinguish between an IN and OUT packet, which control input and output transfers respectively.
- The PID field is followed by the 7-bit address of a device and the 4-bit endpoint number within that device.
- The packet ends with 5 bits for error checking using a method called Cyclic Redundancy check(CRC).
- The CRC bits are computed based on the contents of the address and endpoints fields.

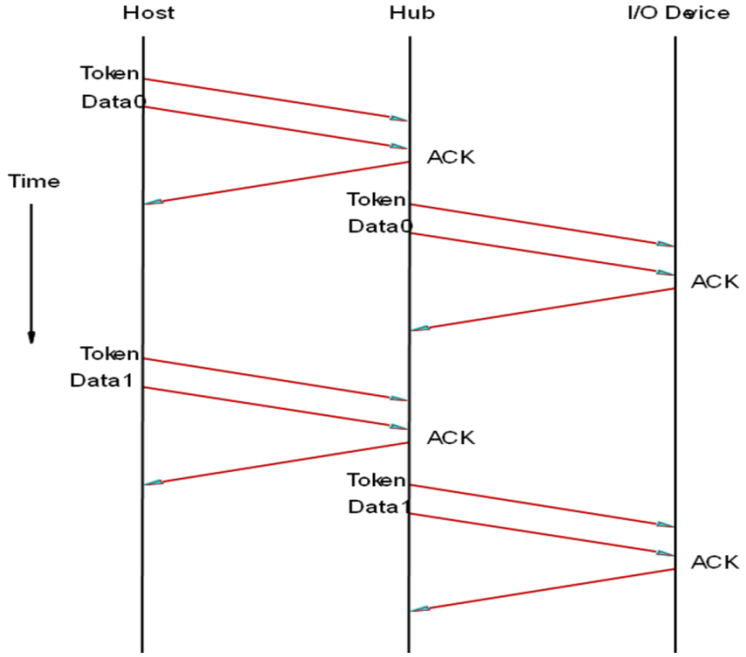


(c) Data packet

- Data packets carry input and output data.
- The PID field is followed by 8192 bits of data, then 16 error-checking bits.
- Three different PID patterns are used to identify data packets.

An Output Data Transfer

Consider an output device connected to a USB hub, which in turn is connected to a host computer. An example of an output operation is shown in the figure given below.



- The host computer sends a token packet of type OUT to the hub, followed by a data packet containing the output data.
- The PID field of the data packet identifies it as data packet number 0.
- The hub verifies that the transmission has been error free by checking the error control bits, and then sends an acknowledgement packet (ACK) back to the host.
- The hub forwards the token and data packets downstream.
- All the I/O devices receive this sequence of packets, but only the device that recognizes its address in the token packet accepts the data in the packet that follows.
- After verifying that transmission has been error free, it sends an ACK packet to the hub.
- If a token, data, or acknowledgement packet is lost as a result of a transmission error, the sender resends the entire sequence. By checking the data packet number in the PID field, the receiver can detect and discard duplicate packets.

Electrical Characteristics

- The cables used for USB connections consist of four wires.
- Two are used to carry power, +5V and Ground.
- The other two wires are used to carry data.
- Different signaling schemes are used for different speeds of transmission.
  - At low speed, 1s and 0s are transmitted by sending a high voltage state (5V) on one or the other of the two signal wires. For high-speed links, differential transmission is used.