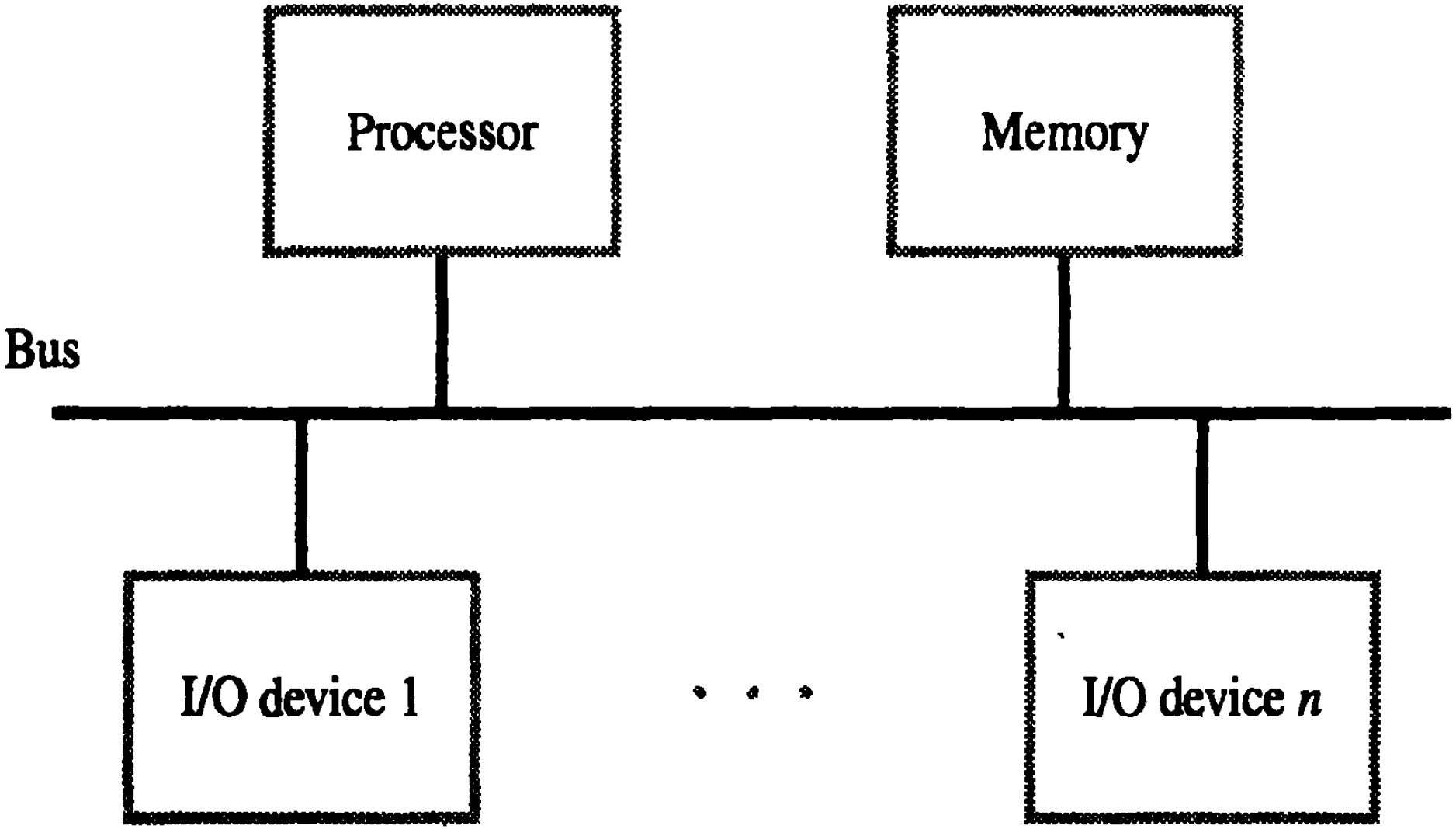
INPUT/OUTPUT ORGANIZATION

- **Accessing I/O Devices**
- **I/O interface**
- **Input/output mechanism**
  - Memory-mapped I/O
  - Programmed I/O
  - Interrupts
  - Direct Memory Access
- **Buses**
  - Synchronous Bus
  - Asynchronous Bus

I/O in CO and O/S

- **Programmed I/O**
- **Interrupts**
- **DMA (Direct memory Access)**

Device	Behavior	Partner	Data rate (Mbit/sec)
Keyboard	input	human	0.0001
Mouse	input	human	0.0038
Voice input	input	human	0.2640
Sound input	input	machine	3.0000
Scanner	input	human	3.2000
Voice output	output	human	0.2640
Sound output	output	human	8.0000
Laser printer	output	human	3.2000
Graphics display	output	human	800.0000–8000.0000
Modem	input or output	machine	0.0160–0.0640
Network/LAN	input or output	machine	100.0000–1000.0000
Network/wireless LAN	input or output	machine	11.0000–54.0000
Optical disk	storage	machine	80.0000
Magnetic tape	storage	machine	32.0000
Magnetic disk	storage	machine	240.0000–2560.0000



A bus is a shared communication link, which uses one set of wires to connect multiple subsystems.

The two major advantages of the bus organization are versatility and low cost.

Accessing I/O Devices

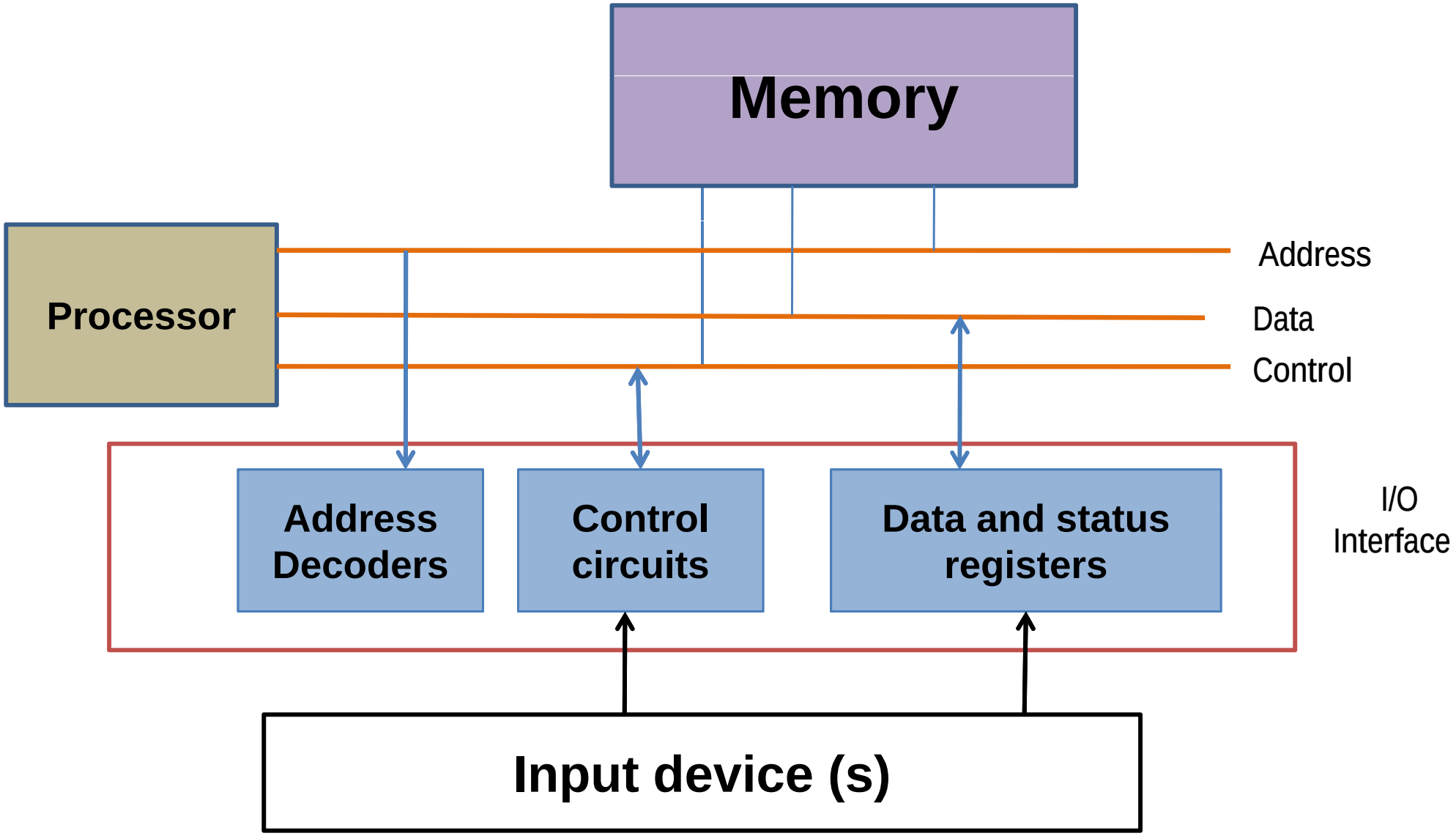
- Most modern computers use single bus arrangement for connecting I/O devices to CPU & Memory
- The bus enables all the devices connected to it to exchange information
- Bus consists of 3 set of lines : *Address, Data, Control*
- Processor places a particular address (unique for an I/O Dev.) on address lines
- Device which recognizes this address responds to the commands issued on the Control lines
- Processor requests for either Read / Write
- The data will be placed on Data lines

Hardware to connect I/O devices to bus

- **Interface Circuit**
  - **Address Decoder**
  - **Control Circuits**
  - **Data registers**
  - **Status registers**

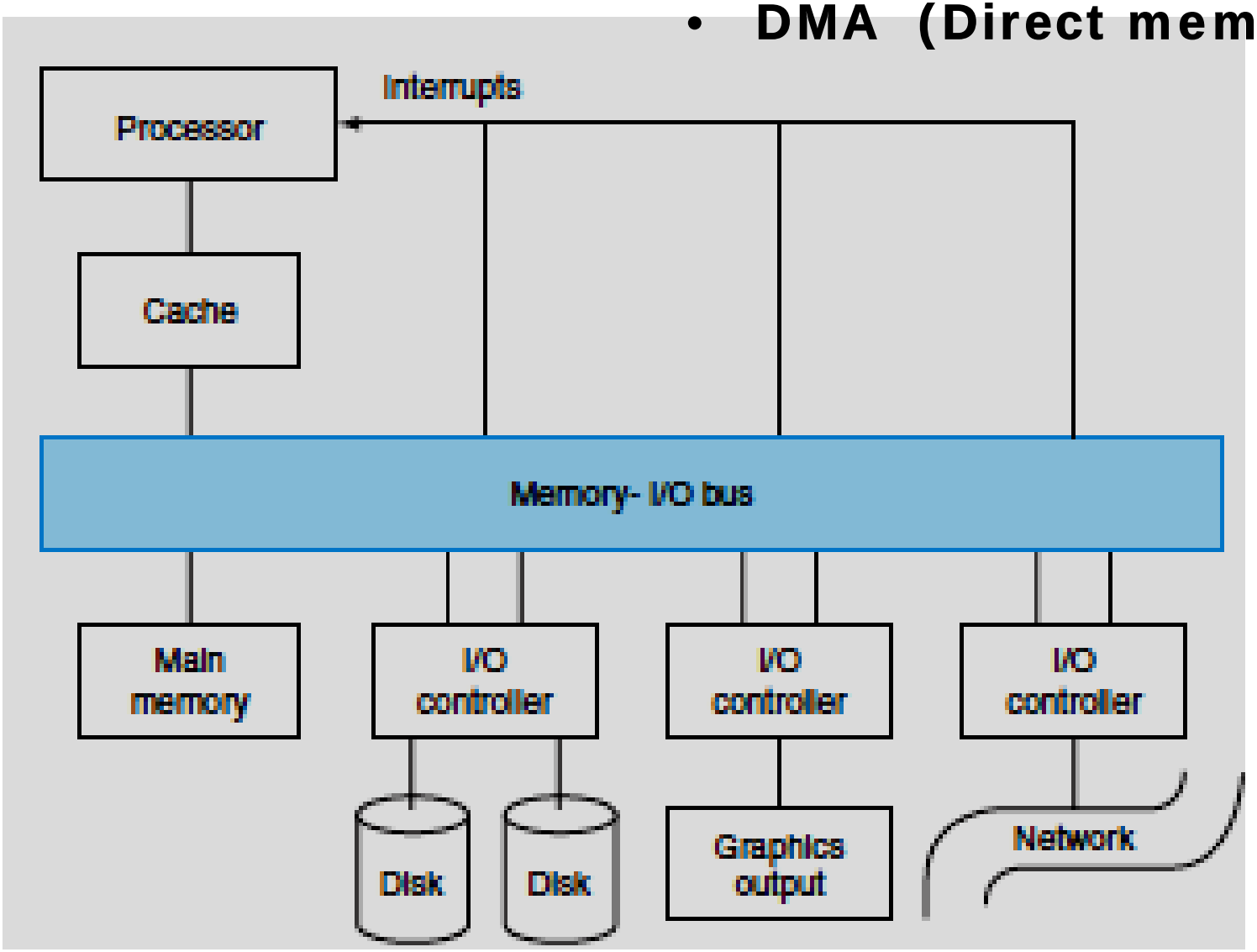
- **The Registers in I/O Interface – buffer and control**
- **Flags in Status Registers, like SIN, SOUT**
- **Data Registers, like Data-IN, Data-OUT**

I / O interface for an input device



Input Output mechanism

- Memory mapped I/O
- Programmed I/O
- Interrupts
- DMA (Direct memory Access)



A bus generally contains a set of control lines and a set of data lines.

The **control lines** are used to signal requests and acknowledgments, and to indicate what type of information is on the data lines. The control lines are used to indicate what the bus contains and to implement the bus protocol.

The **data lines** of the bus carry information between the source and the destination. This information may consist of data, complex commands, or addresses.

Buses are traditionally classified as processor-memory buses or I/O buses or special purposed buses (Graphics, etc.). **Processor memory buses** are short, generally high speed, and matched to the memory system so as to maximize memory-processor bandwidth.

I/O buses, by contrast, can be lengthy, can have many types of devices connected to them, and often have a wide range in the data bandwidth of the devices connected to them. I/O buses do not typically interface directly to the memory but use either a processor-memory or a backplane bus to connect to memory.

The major disadvantage of a bus is that it creates a communication bottleneck, possibly limiting the maximum I/O throughput.

When I/O must pass through a single bus, the bandwidth of that bus limits the maximum I/O throughput.

Reason why bus design is so difficult :

- the maximum bus speed is largely limited by physical factors: the length of the bus and the number of devices. These physical limits prevent us from running the bus arbitrarily fast.

- In addition, the need to support a range of devices with widely varying latencies and data transfer rates also makes bus design challenging.

- it becomes difficult to run many parallel wires at high speed due to clock skew and reflection.

The two basic schemes for communication on the bus are synchronous and asynchronous.

If a bus is synchronous (e.g. Processor-memory), it includes a clock in the control lines and a fixed protocol for communicating that is relative to the clock.

This type of protocol can be implemented easily in a small finite state machine. Because the protocol is predetermined and involves little logic, the bus can run very fast and the interface logic will be small.

Synchronous buses have two major disadvantages:

- First, every device on the bus must run at the same clock rate.
- Second, because of clock skew problems, synchronous buses cannot be long if they are fast.

An asynchronous bus is not clocked. It can accommodate a wide variety of devices, and the bus can be lengthened without worrying about clock skew or synchronization problems.

To coordinate the transmission of data between sender and receiver, an asynchronous bus uses a handshaking protocol.

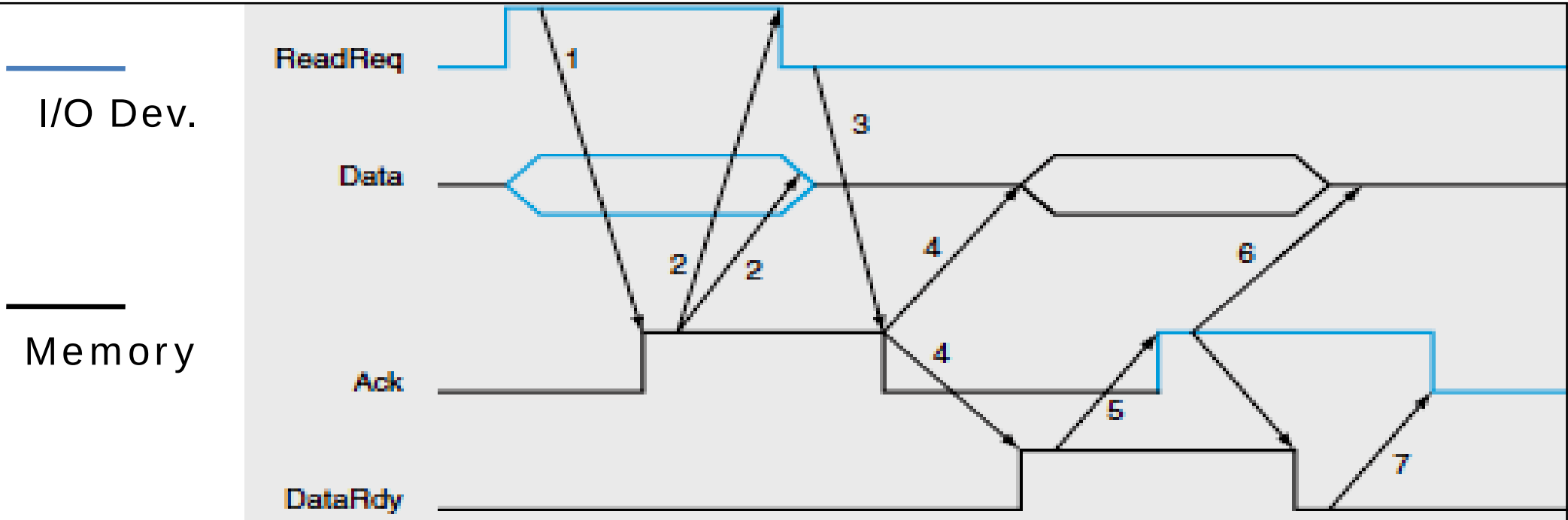
Characteristic	Firewire (1394)	USB 2.0
Bus type	I/O	I/O
Basic data bus width (signals)	4	2
Clocking	asynchronous	asynchronous
Theoretical peak bandwidth	50 MB/sec (Firewire 400) or 100 MB/sec (Firewire 800)	0.2 MB/sec (low speed), 1.5 MB/sec (full speed), or 60 MB/sec (high speed)
Hot plugable	yes	yes
Maximum number of devices	63	127
Maximum bus length (copper wire)	4.5 meters	5 meters
Standard name	IEEE 1394, 1394b	USE Implementors Forum

Three special control lines required for hand-shaking:

ReadReq: Used to indicate a read request for memory. The address is put on the data lines at the same time.

DataRdy: Used to indicate that the data word is now ready on the data lines; asserted by: Output / Memory and Input / I_O Device.

Ack: Used to acknowledge the ReadReq or the DataRdy signal of the other party.



Steps after the device signals a request by raising ReadReq and putting the address on the Data lines:

1. When memory sees the ReadReq line, it reads the address from the data bus and raises Ack to indicate it has been seen.

2. As the Ack line is high - I/O releases the ReadReq and data lines.

3. Memory sees that ReadReq is low and drops the Ack line to acknowledge the ReadReq signal (*Mem. Reading in progress now*).

4. This step starts when the memory has the data ready. It places the data from the read request on the data lines and raises DataRdy.

5. The I/O device sees DataRdy, reads the data from the bus, and signals that it has the data by raising Ack.

6. On the Ack signal, M/M drops DataRdy, and releases the data lines.

7. Finally, the I/O device, seeing DataRdy go low, drops the Ack line, which indicates that the transmission is completed.

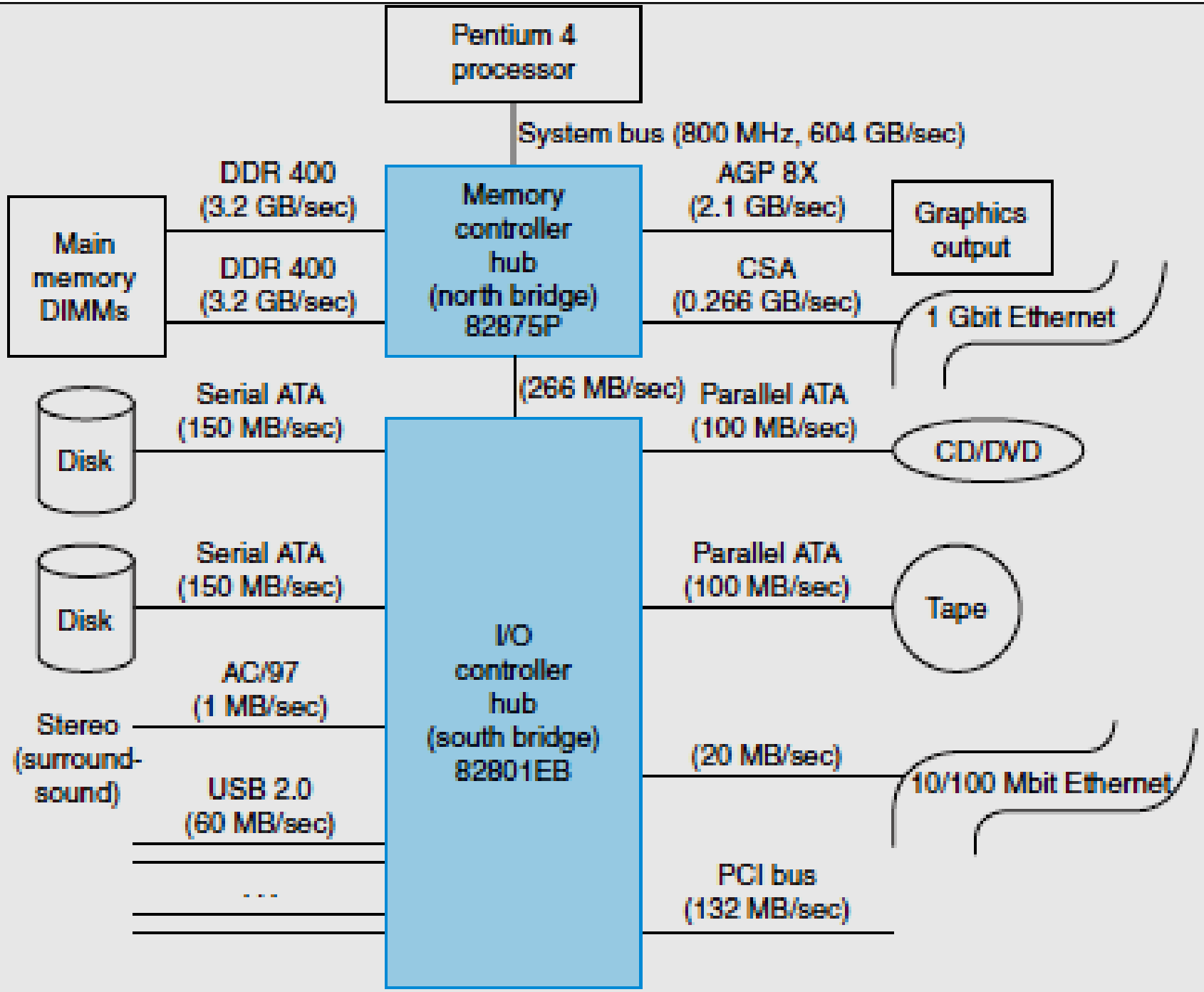


FIGURE 8.11 Organization of the I/O system on a Pentium 4 PC using the Intel 875 chip set. Note that the maximum transfer rate between the north bridge (memory hub) and south bridge (I/O hub) is 266 MB/sec, which is why Intel put the AGP bus and Gigabit Ethernet on the north bridge.

Memory mapped I/O

- I/O devices and the memory share the same address space, the arrangement is called *Memory-mapped I/O.*

- In Memory-mapped I/O portions of address space are assigned to I/O devices and reads and writes to those addresses are interpreted as commands to the I/O device.

"DATAIN" is the address of the input buffer associated with the keyboard.

- Move DATAIN, R0

reads the data from DATAIN and stores them into processor register R0;

- Move R0, DATAOUT

sends the contents of register R0 to location DATAOUT

Option of special I/O address space or incorporate as a part of memory address space (address bus is same always).

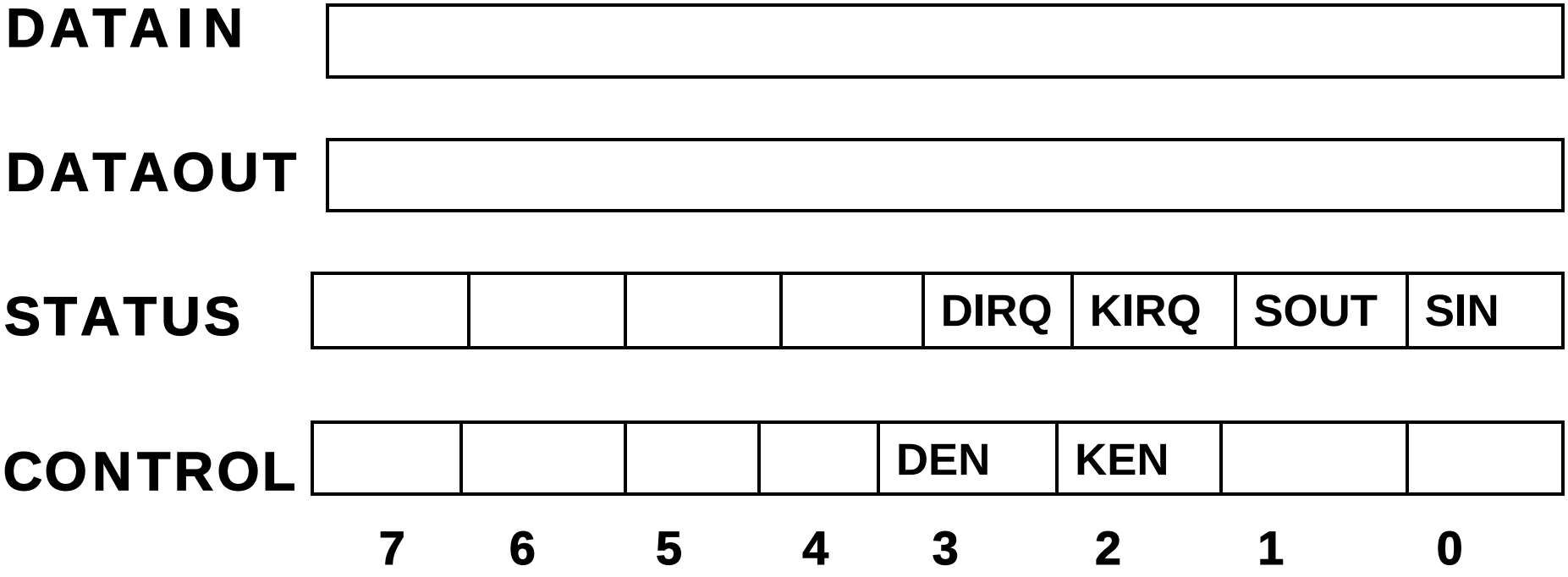
When the processor places the address and data on the memory bus, the memory system ignores the operation because the address indicates a portion of the memory space used for I/O.

The device controller, however, sees the operation, records the data, and transmits it to the device as a command.

User programs are prevented from issuing I/O operations directly because the OS does not provide access to the address space assigned to the I/O devices and thus the addresses are protected by the address translation.

Memory mapped I/O can also be used to transmit data by writing or reading to select addresses. The device uses the address to determine the type of command, and the data may be provided by a write or obtained by a read.

A program request usually requires several separate I/O operations. Furthermore, the processor may have to interrogate the status of the device between individual commands to determine whether the command completed successfully.



I/O operation involving keyboard and display devices

Registers: DATAIN, DATAOUT, STATUS, CONTROL

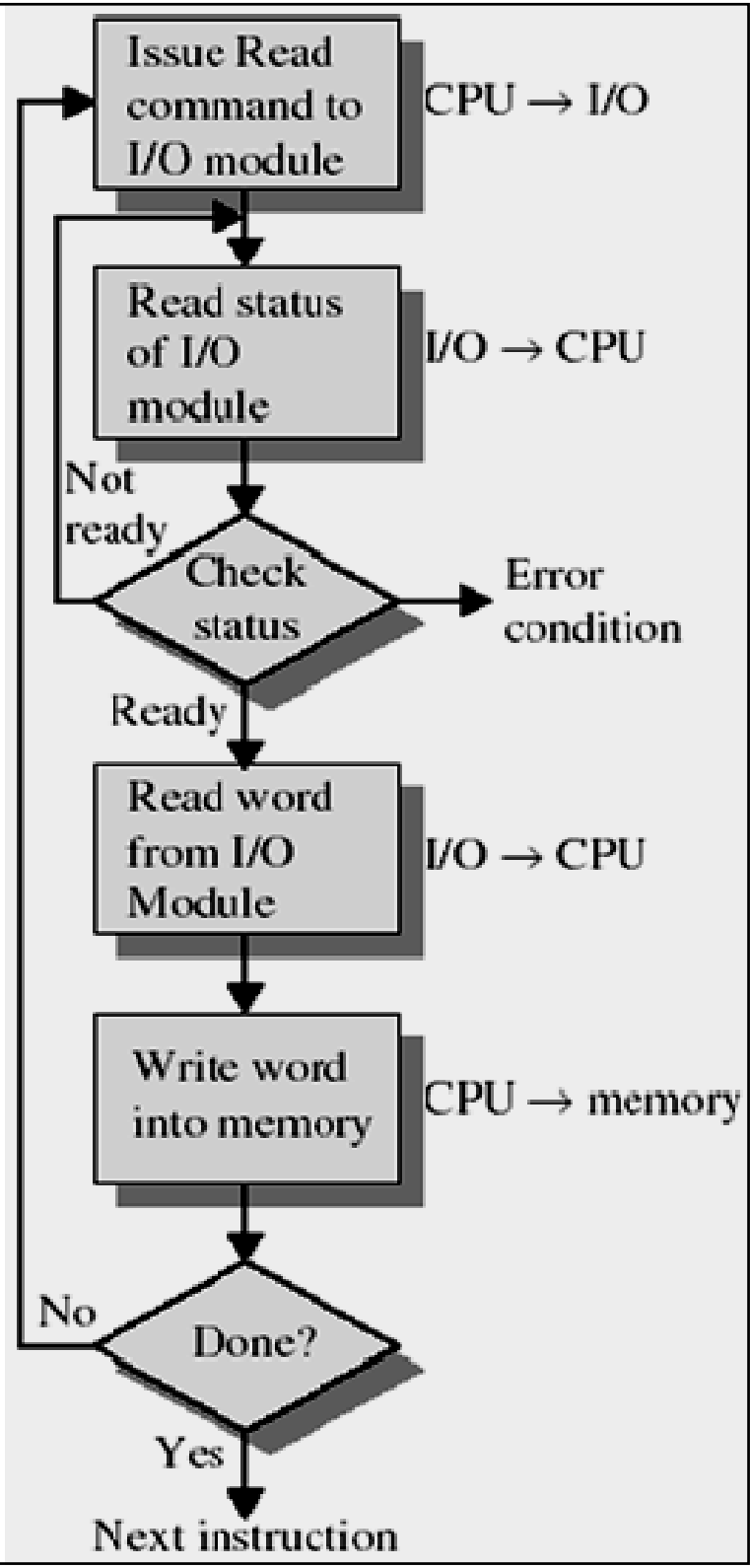
Flags: SIN, SOUT - Provides status information for keyboard and display unit

KIRQ, DIRQ – Keyboard, Display Interrupt request bits

DEN, KEN –Keyboard, Display Enable bits

Programmed I/O

- CPU has direct control over I/O
  - Sensing status
  - Read/write commands
  - Transferring data
- CPU waits for I/O module to complete operation
- Wastes CPU time



In this case, use dedicated I/O instructions in the processor. These I/O instructions can specify both the device number and the command word (or the location of the command word in memory).

The processor communicates the device address via a set of wires normally included as part of the I/O bus. The actual command can be transmitted over the data lines in the bus. (example - Intel IA-32).

By making the I/O instructions illegal to execute when not in kernel or supervisor mode, user programs can be prevented from accessing the devices directly.

The process of periodically checking status bits to see if it is time for the next I/O operation, is called polling. Polling is the simplest way for an I/O device to communicate with the processor.

The I/O device simply puts the information in a Status register, and the processor must come and get the information. The processor is totally in control and does all the work.

A ISA program to read one line from the keyboard, store it in memory buffer, and echo it back to the display

```
        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.
```

The disadvantage of polling is that it can waste a lot of processor time because processors are so much faster than I/O devices.

The processor may read the Status register many times, only to find that the device has not yet completed a comparatively slow I/O operation, or that the mouse has not budged since the last time it was polled.

When the device completes an operation, we must still read the status to determine whether it (I/O) was successful.

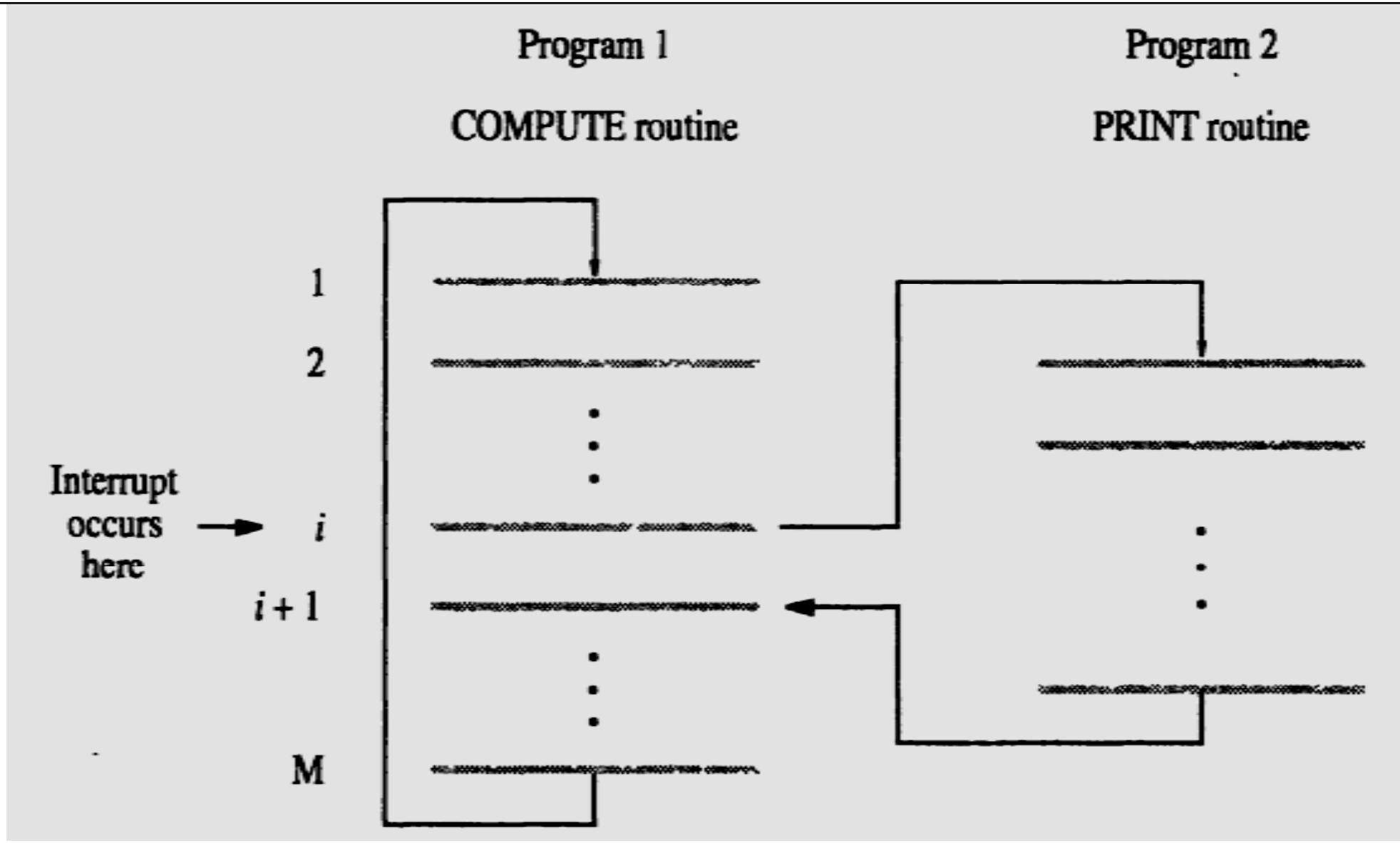
Overhead in a polling interface lead to the invention of interrupts to notify the processor when an I/O device requires attention from the processor.

Interrupt-driven I/O, employs I/O interrupts to indicate to the processor that an I/O device needs attention.

When a device wants to notify the processor that it has completed some operation or needs attention, it causes the processor to be interrupted.

Interrupts

- When I/O Device is ready, it sends the INTERRUPT signal to processor via a dedicated controller line
- Using interrupt we are ideally eliminating WAIT period
- In response to the interrupt, the processor executes the Interrupt Service Routine (ISR)
- All the registers, flags, program counter values are saved by the processor before running ISR
- The time required to save status & restore contribute to execution overhead → "Interrupt Latency"


Program 1
COMPUTE routine
Program 2
PRINT routine
1
2
Interrupt occurs here
i
i + 1
M

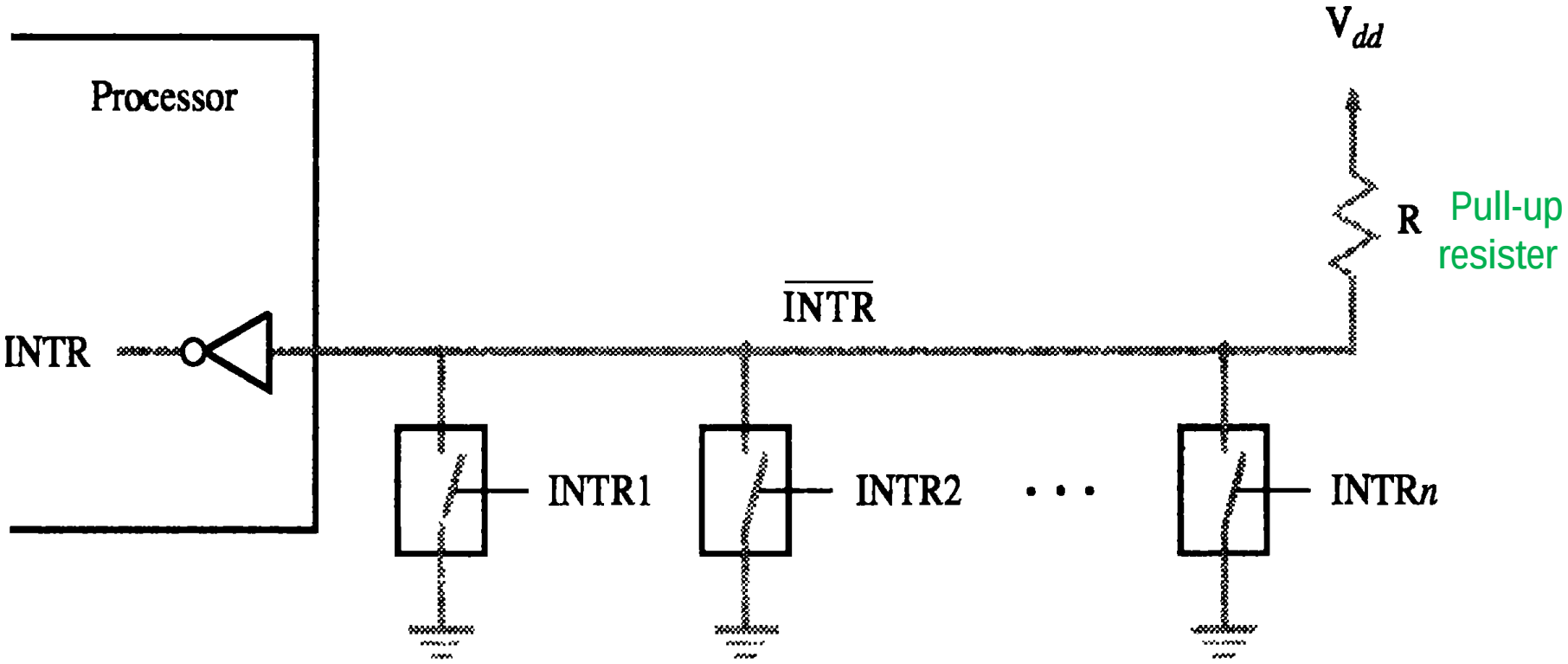
interrupt-acknowledge signal - I/O device interface accomplishes this by execution of an instruction in the interrupt-service routine (ISR) that accesses a status or data register in the device interface; implicitly informs the device that its interrupt request has been recognized. IRQ signal is then removed by device.

ISR is a sub-routine – may belong to a different user than the one being executed and then halted.

The condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine are saved and restored.

The concept of interrupts is used in operating systems and in many control applications, where processing of certain routines must be accurately timed relative to external events (e.g. real-time processing).

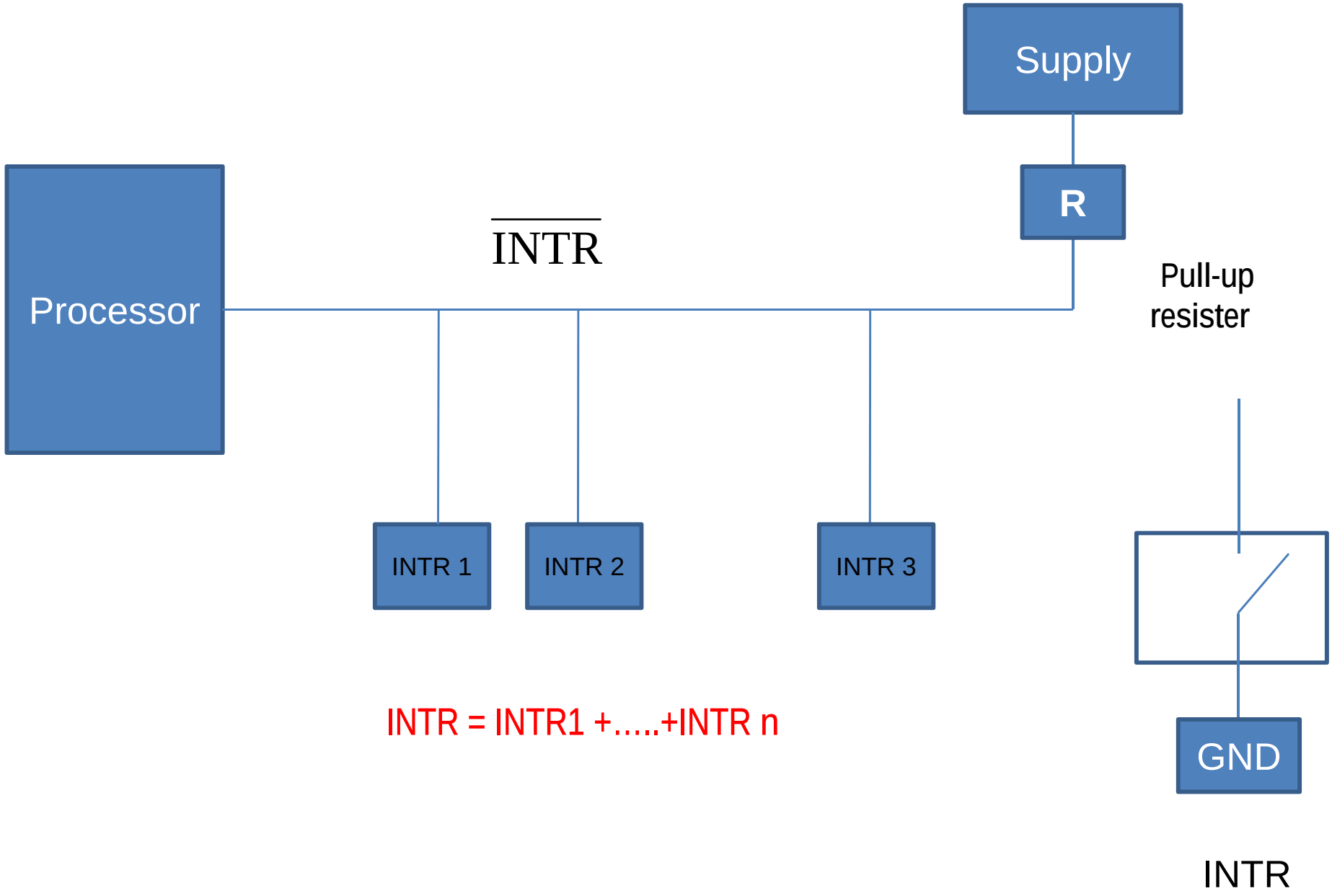
Interrupt Hardware



INTR = INTR1 +.....+INTR n

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

Interrupt Hardware



Enabling and Disabling Interrupts

- Device activates interrupt signal line and waits with this signal activated until processors attends
- The interrupt signal line is active during execution of ISR and till the device caused interrupt is serviced
- Necessary to ensure that the active signal does not lead to successive interruptions (level-triggered input) causing the system to fall in infinite loop.
- What if the same device interrupts again, within an ISR ?
- Three methods of Controlling Interrupts (single device)
  - Ignoring interrupt
  - Disabling interrupts
  - Special Interrupt request line

- **Ignoring Interrupts**
  - Processor hardware ignores the interrupt request line until the execution of the first instruction of the ISR completed
  - Using an interrupt disable instruction after the first instruction of the ISR – no further interrupts
  - A return from interrupt instruction is completed before further interruptions can occur

- **Disabling Interrupts**
  - Processor automatically disables interrupts before starting the execution of the ISR
  - The processor saves the contents of PC and PS (status register) before performing interrupt disabling.
  - The interrupt-enable is set to 0 – no further interrupts allowed
  - When return from interrupt instruction is executed the contents of the PS are restored from the stack, and the interrupt enable is set to 1

- **Special Interrupt line**
  - Special interrupt request line for which the interrupt handling circuit responds only to the leading edge of the signal
  - Edge –triggered
  - Processor receives only one request regardless of how long the line is activated
  - No separate interrupt disabling instructions

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.

Handling Multiple Devices

- **Multiple devices can initiate interrupts**
- **They uses the common interrupt request line**
- **Techniques are**
  - **Polling**
  - **Vectored Interrupts**
  - **Interrupt Nesting**
  - **Daisy Chaining**

Polling Scheme

- **The IRQ (interrupt request) bit in the status register is set when a device is requesting an interrupt.**
- **The Interrupt service routine polls the I/O devices connected to the bus.**
- **The first device encountered with the IRQ bit set is serviced and the subroutine is invoked.**
- **Easy to implement, but too much time spent on checking the IRQ bits of all devices, though some devices may not be requesting service.**

Vectored Interrupts

- Device requesting an interrupt identifies itself directly to the processor
- The device sends a special code to the processor over the bus.
- The code contains the
  - identification of the device,
  - starting address for the ISR,
  - address of the branch to the ISR
- PC finds the ISR address from the code.
- To add flexibility for multiple devices - corresponding ISR is executed by the processor using a branch address to the appropriate routine - device specified *Interrupt Vector*.

An interrupt vector is the memory address of an interrupt handler, or an index into an array called an interrupt vector table or dispatch table - a table of interrupt vectors (pointers to routines that handle interrupts).

Interrupt vector tables contain the memory addresses of interrupt handlers. When an interrupt is generated, the processor saves its execution state via a context switch, and begins execution of the interrupt handler at the interrupt vector.

The Interrupt Descriptor Table (IDT) is specific to the I386 architecture. It tells where the Interrupt Service Routines (ISR) are located.

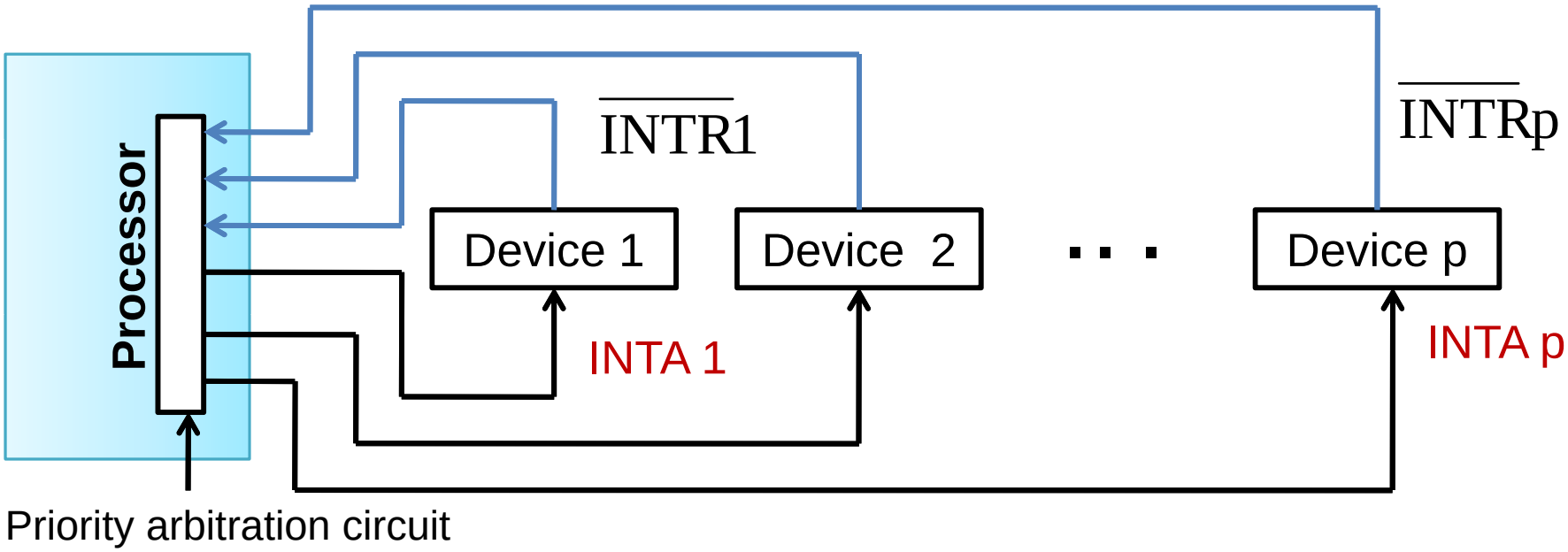
Each interrupt number is reserved for a specific purpose. For example, 16 of the vectors are reserved for the 16 IRQ lines.

On PCs, the interrupt vector table (IVT or IDT) consists of 256 4-byte pointers - the first 32 (0-31 or 00-1F) of which are reserved for processor exceptions; the rest for hardware interrupts, software interrupts. This resides in the first 1 K of addressable memory.

Interrupt Nesting

- **Pre-Emption of low priority Interrupt by another high priority interrupt is known as Interrupt nesting.**
- **Disabling Interrupts during the execution of the ISR may not favor devices which need immediate attention.**
- **Need a priority of IRQ devices and accepting IRQ from a high priority device.**
- **The priority level of the processor can be changed dynamically.**
- **The privileged instruction write in the PS (processor status word), that encodes the processors priority.**

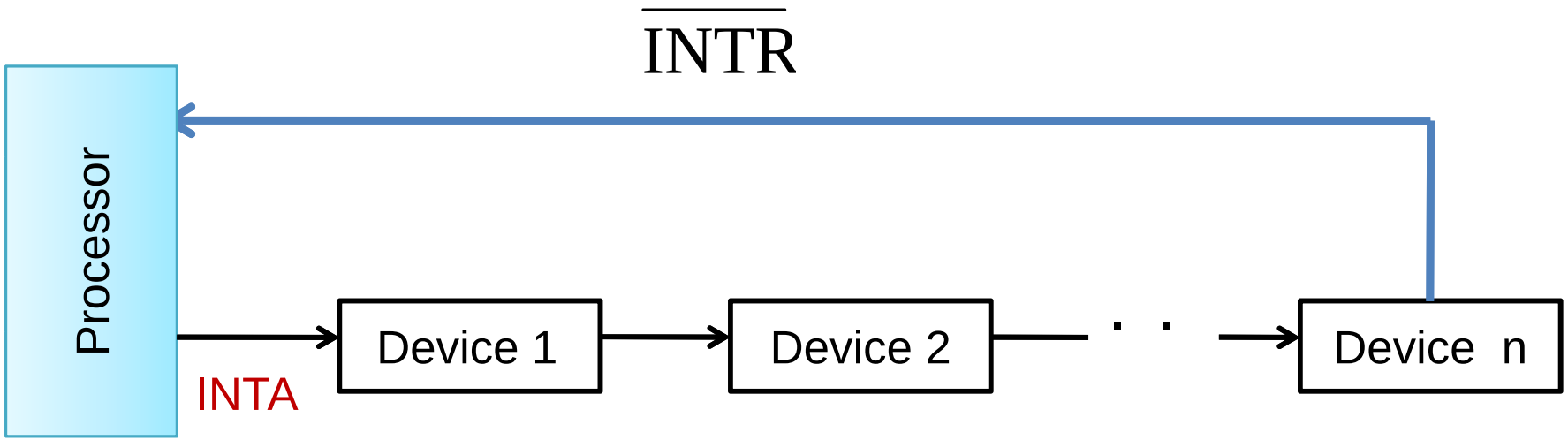
Interrupt Nesting (contd.)



- **Organizing I/O devices in a prioritized structure.**
- **Each of the interrupt-request lines is assigned a different priority level.**
- **The processor is interrupted only by a high priority device.**

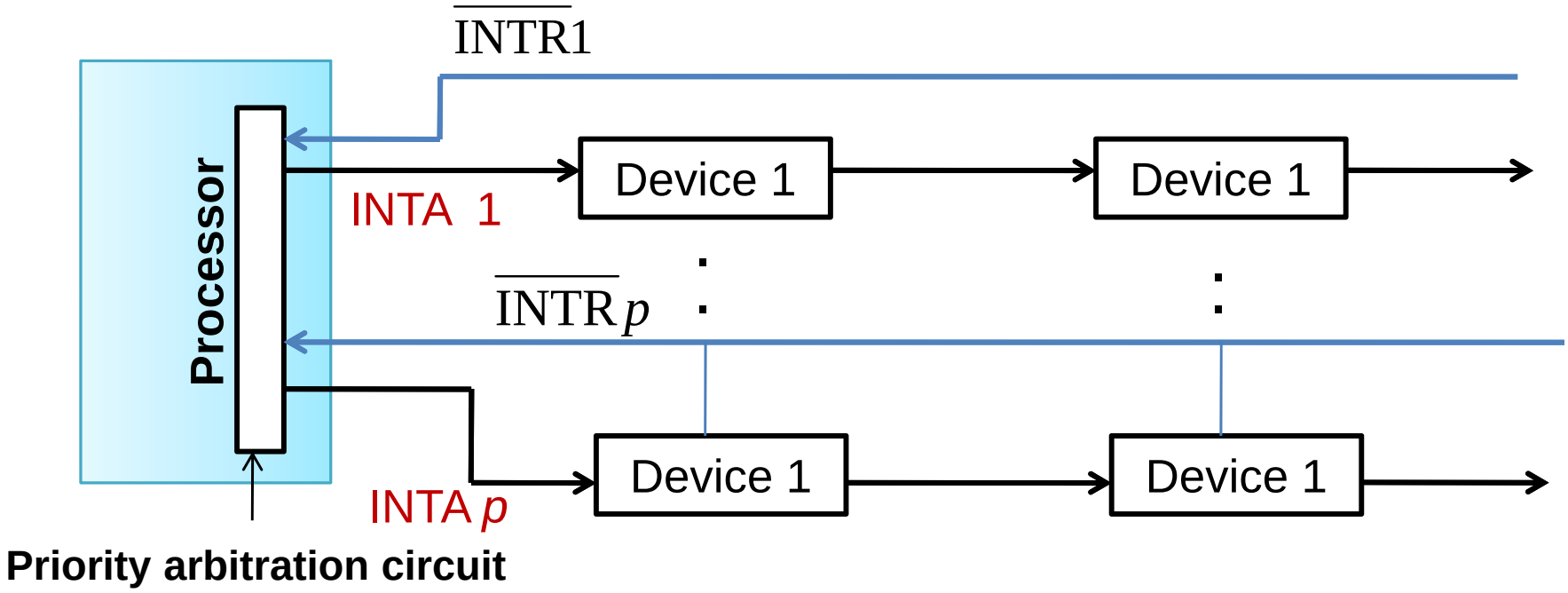
Daisy Chaining

- **The interrupt request line INTR is common to all the devices**
- **The interrupt acknowledgement line INTA is connected to devices in a DAISY CHAIN way**
- **INTA propagates serially through the devices**
- **Device that is electrically closest to the processor gets high priority**
- **Low priority device may have a danger of STARVATION**



Daisy Chaining with Priority Group

- **Combining Daisy chaining and Interrupt nesting to form priority group**
- **Each group has different priority levels and within each group devices are connected in daisy chain way**



Arrangement of priority groups

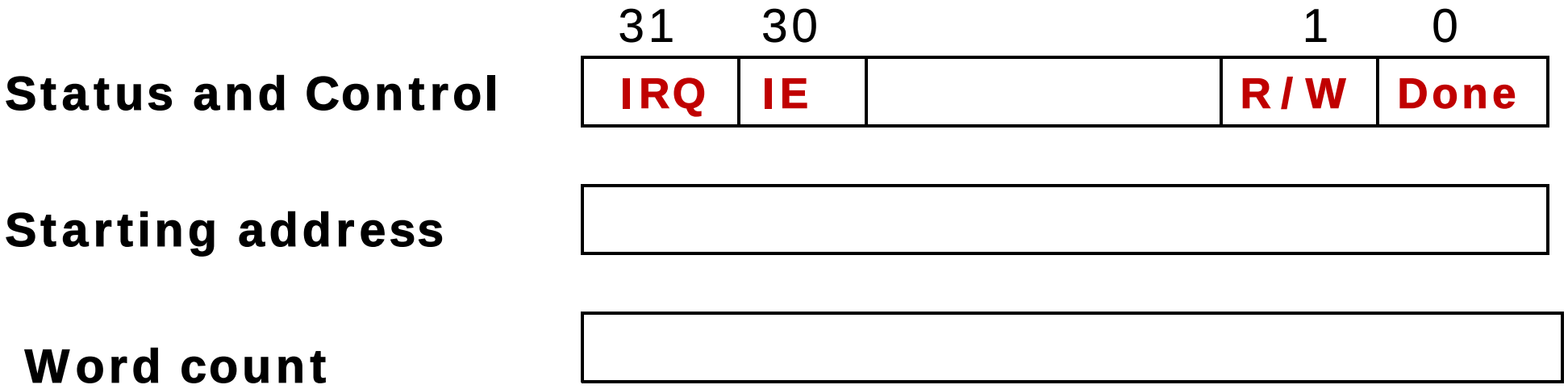
Direct Memory Access (DMA)

- For I/O transfer, Processor determines the status of I/O devices, by
  - Polling
  - Waiting for Interrupt signal

- Considerable overhead is incurred in above I/O transfer processing
- To transfer large blocks of data at high Speed, between EXTERNAL devices & Main Memory, DMA approach is often used
- DMA controller allows data transfer directly between I/O device and Memory, with minimal intervention of processor.

Direct Memory Access (DMA)

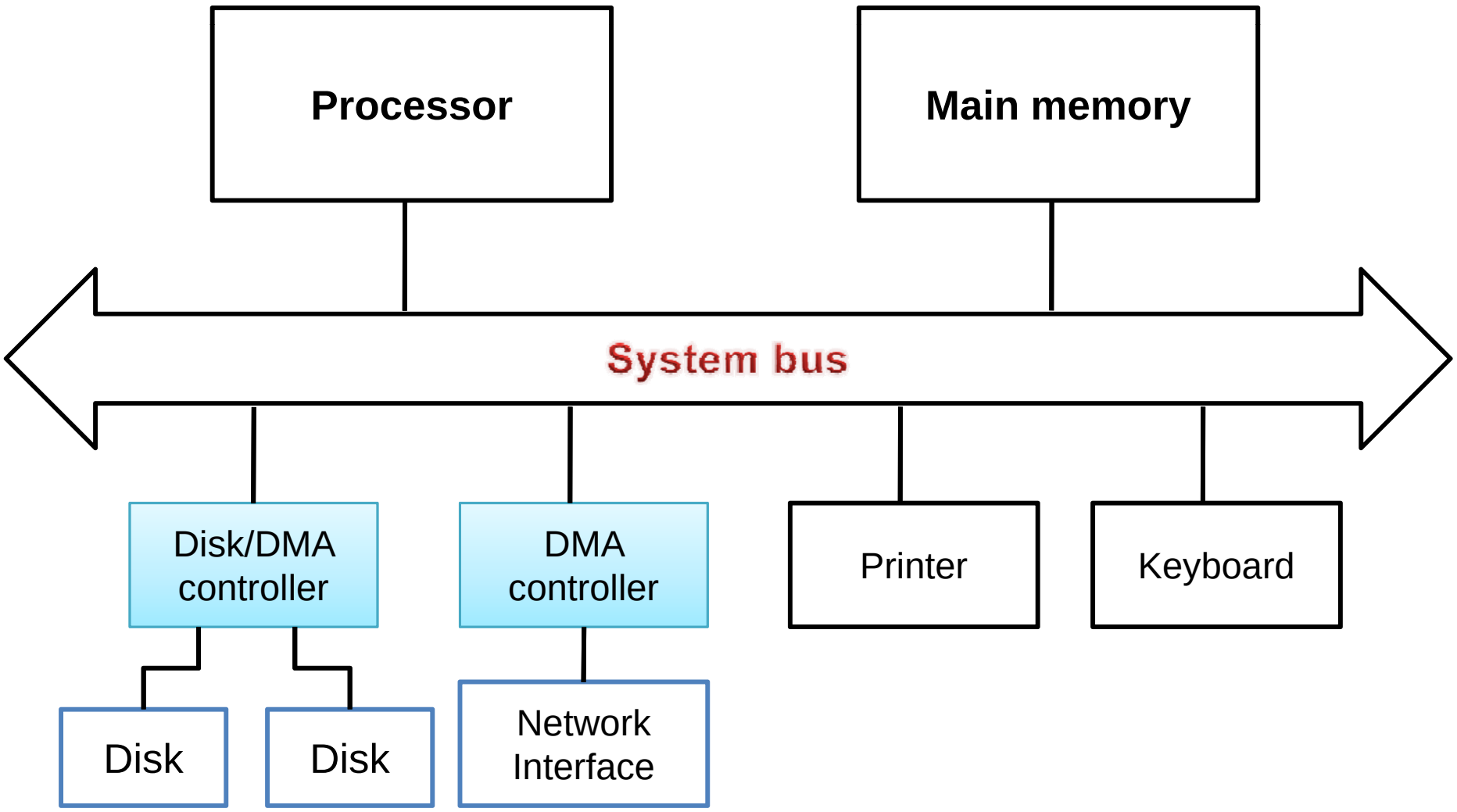
- DMA controller acts as a Processor, but it is controlled by CPU
- To initiate transfer of a block of words, the processor sends the following data to controller
  - The starting address of the memory block
  - The word count
  - Control to specify the mode of transfer such as read or write
  - A control to start the DMA transfer
- DMA controller performs the requested I/O operation and sends a interrupt to the processor upon completion



In DMA interface

- **First register stores the starting address**
- **Second register stores Word count**
- **Third register contains status and control flags**

Bits and Flags	1	0
R/W	READ	WRITE
Done	Data transfer finishes	
IRQ	Interrupt request	
IE	Raise interrupt (enable) after Data Transfer	



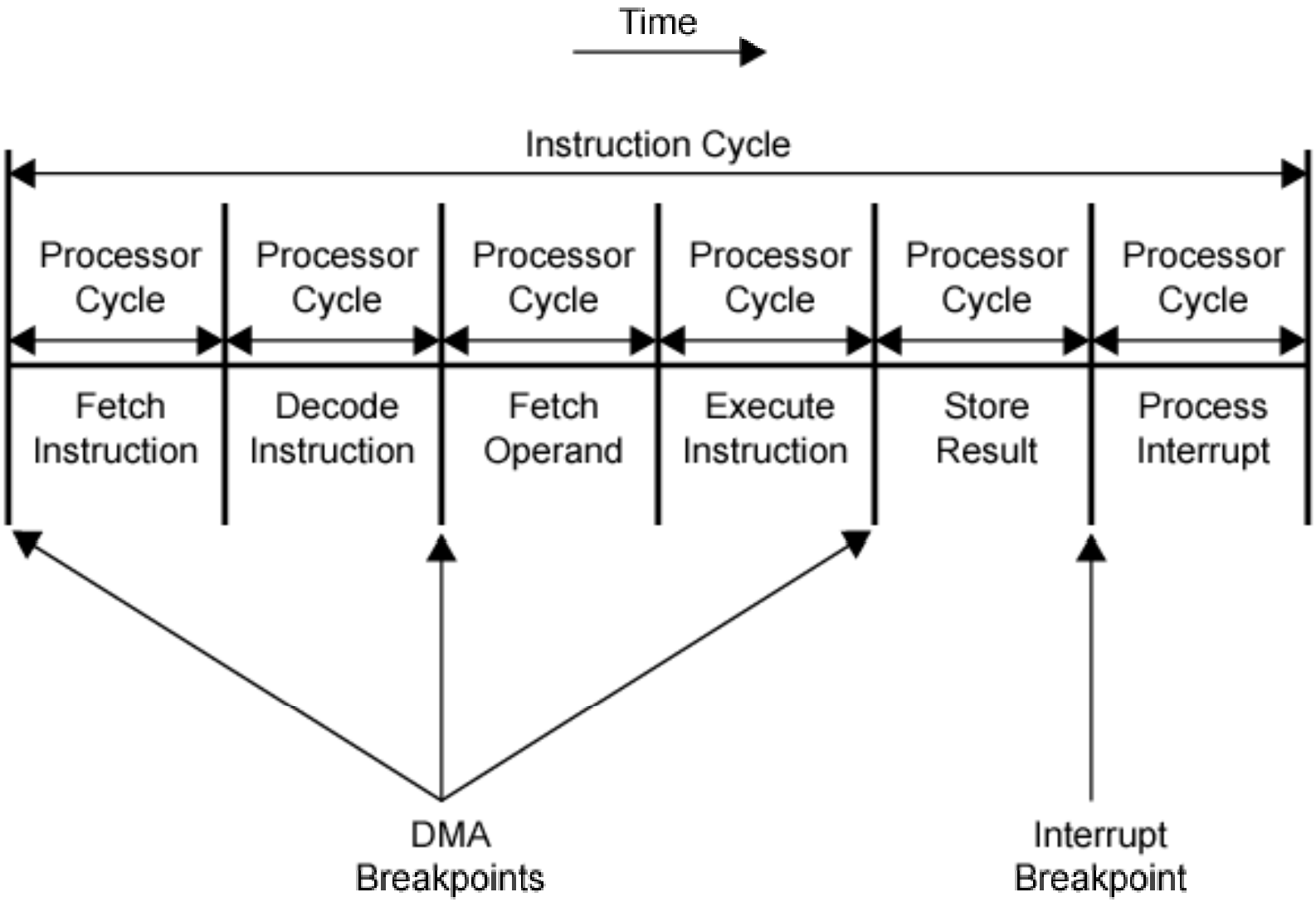
Use of DMA Controller in a computer system

- **Memory accesses by the processor and DMA Controller are interwoven**
- **DMA devices have higher priority then processor over BUS control**
- **Cycle Stealing:- DMA Controller "steals" memory cycles from processor, though processor originates most memory access.**
- **Block or Burst mode:- The DMA controller may given exclusive access to the main memory to transfer a block of data without interruption**
- **Conflicts in DMA:**

  - Processor and DMA,

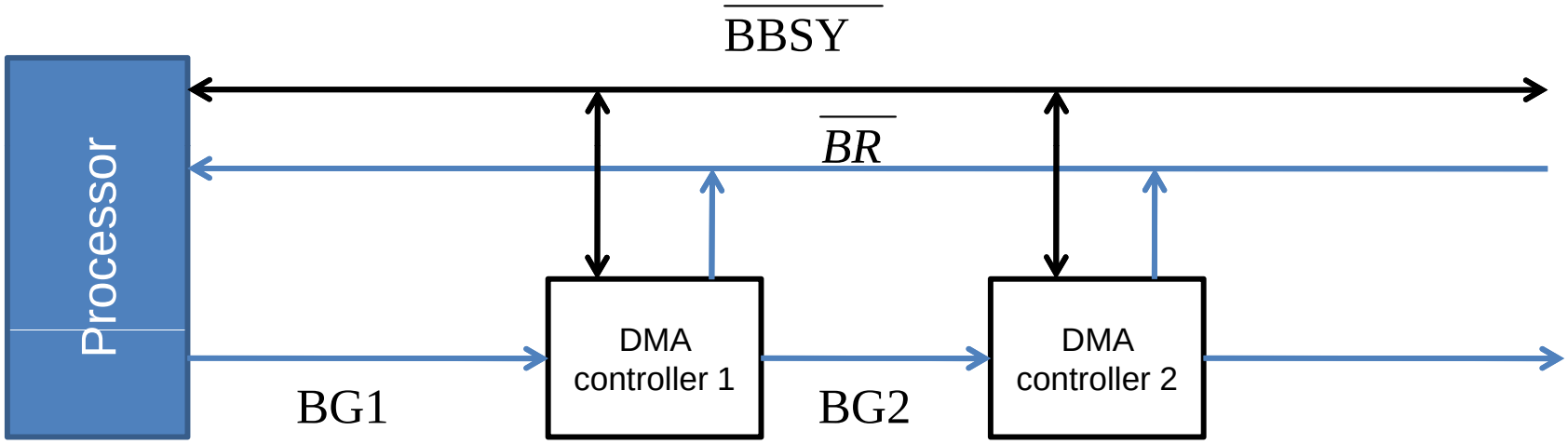
  - Two DMA controllers, try to use the Bus at the same time to access the main memory

DMA and Interrupt Breakpoints During an Instruction Cycle



Bus Arbitration

- **Bus master**: device that initiates data transfers on the bus.
- The next device can take control of the bus after the current master relinquishes control
- **Bus Arbitration**: process by which the next device to become master is selected
- **Centralized and Distributed Arbitration**

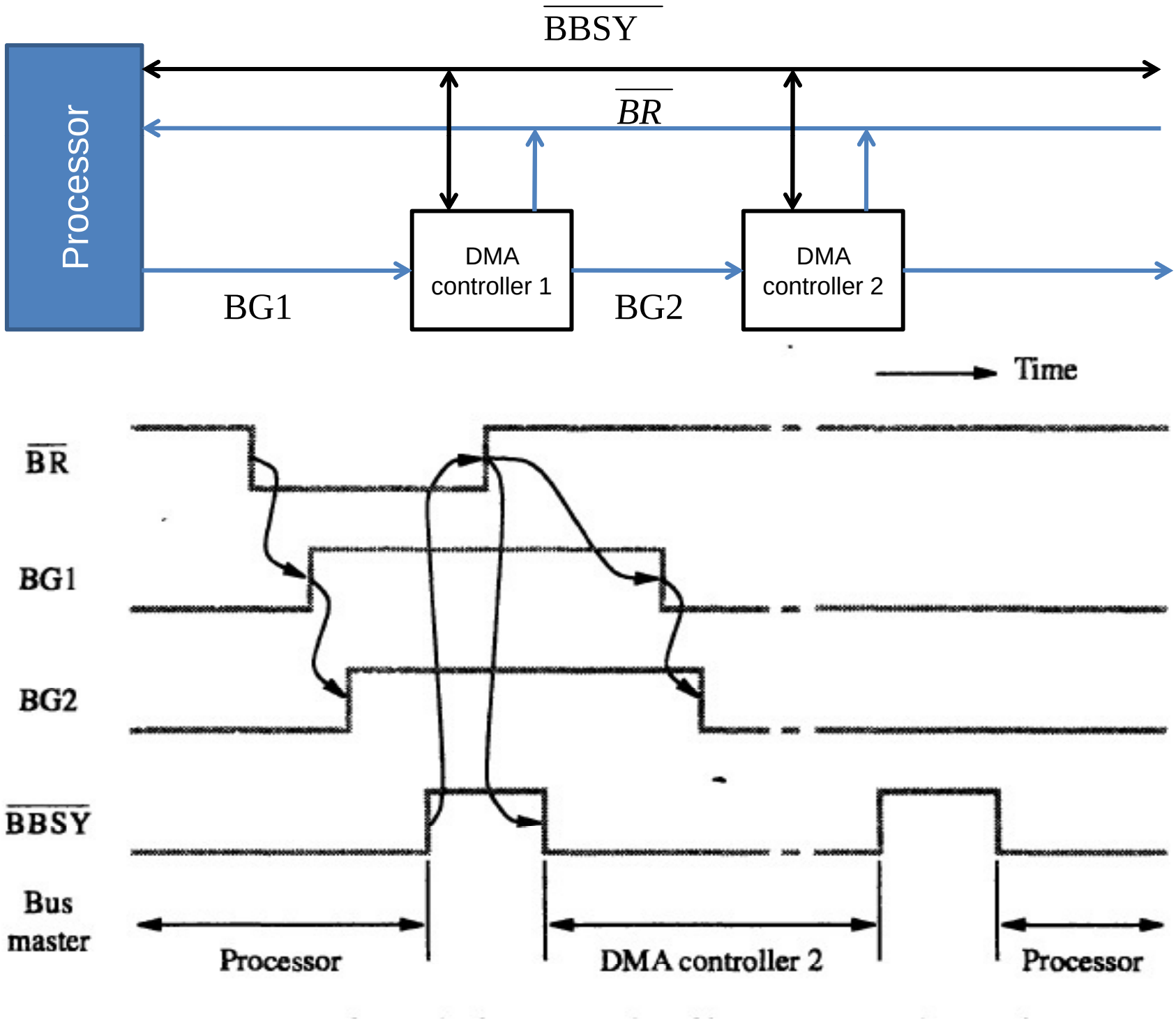


A simple arrangement for bus arbitration using a daisy chain

- **BR (bus request) line - open drain line - the signal on this line is a logical OR of the bus request from all the DMA devices**

- **BG (bus grant) line - processor activates this line indicating (acknowledging) to all the DMA devices (connected in daisy chain fashion) that the BUS may be used when its free.**

- **BBSY (bus busy) line - open collector line - the current bus master indicates devices that it is currently using the bus by signaling this line**



Sequence of signals during data transfer of bus mastership

- **Centralized Arbitration**

  - **Separate unit (bus arbitration circuitry) connected to the bus**

  - **Processor is normally the bus master, unless it grants bus mastership to DMA**

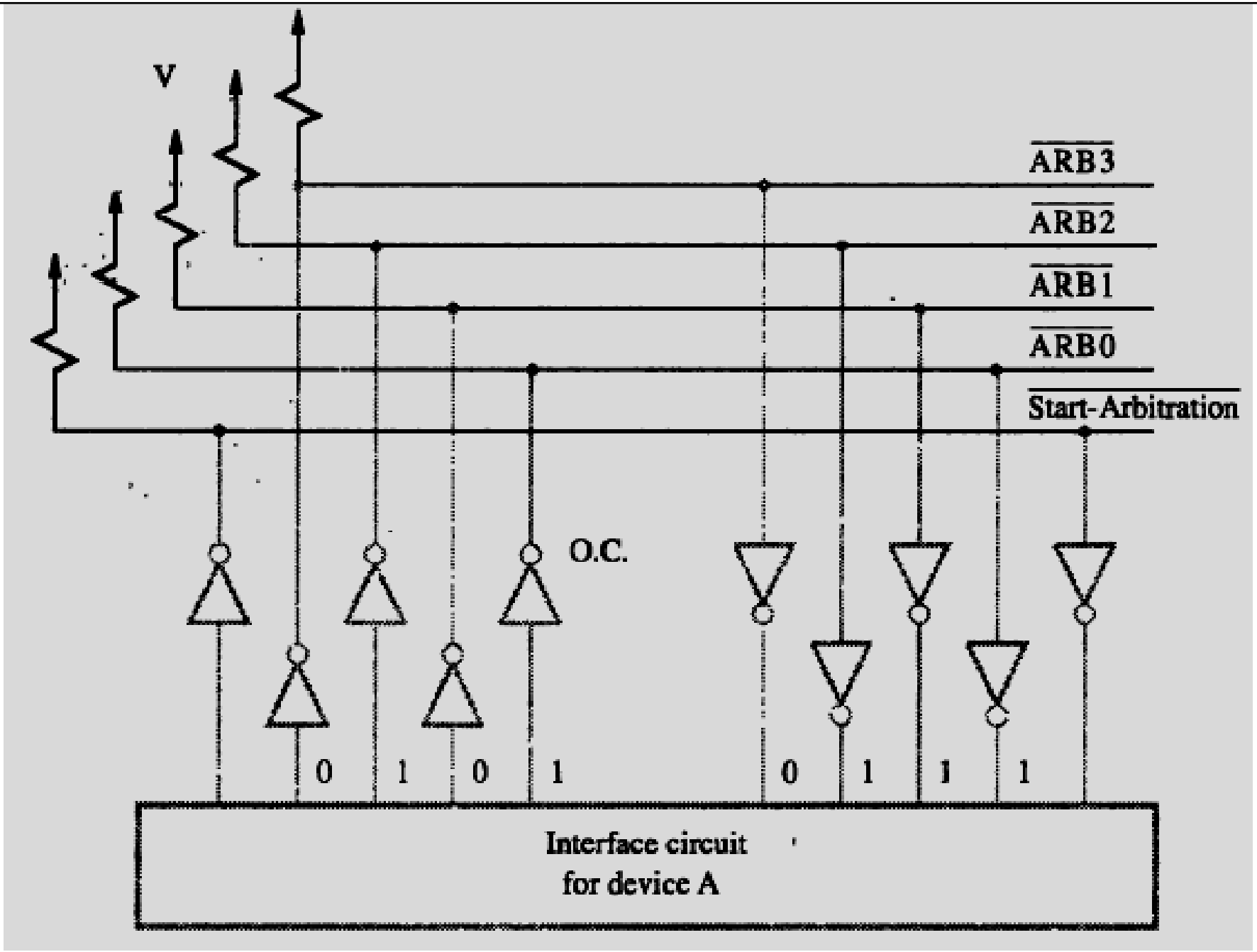
For the timing/control, in previous slide:

DMA controller 2 requests and acquires bus mastership and later releases the bus.

During its tenure as the bus master, it may perform one or more data transfer operations, depending on whether it is operating in the cycle stealing or block mode.

After it releases the bus, the processor resumes bus mastership.

- **Distributed Arbitration**
  - **All devices waiting to use the bus has to carry out the arbitration process - no central arbiter**
  - **Each device on the bus is assigned with a 4-bit identification number**
  - **One or more devices request the bus by asserting the start-arbitration signal and place their identification number on the four open collector lines**
  - **ARB0 through ARB3 are the four open collector lines**
  - **One among the four is selected using the code on the lines and one with the highest ID number**



A distributed arbitration scheme

Assume that two devices, A and B, having ID numbers 5 and 6, respectively, are requesting the use of the bus.

Device A transmits the pattern 0101, and device B transmits the pattern 0110.

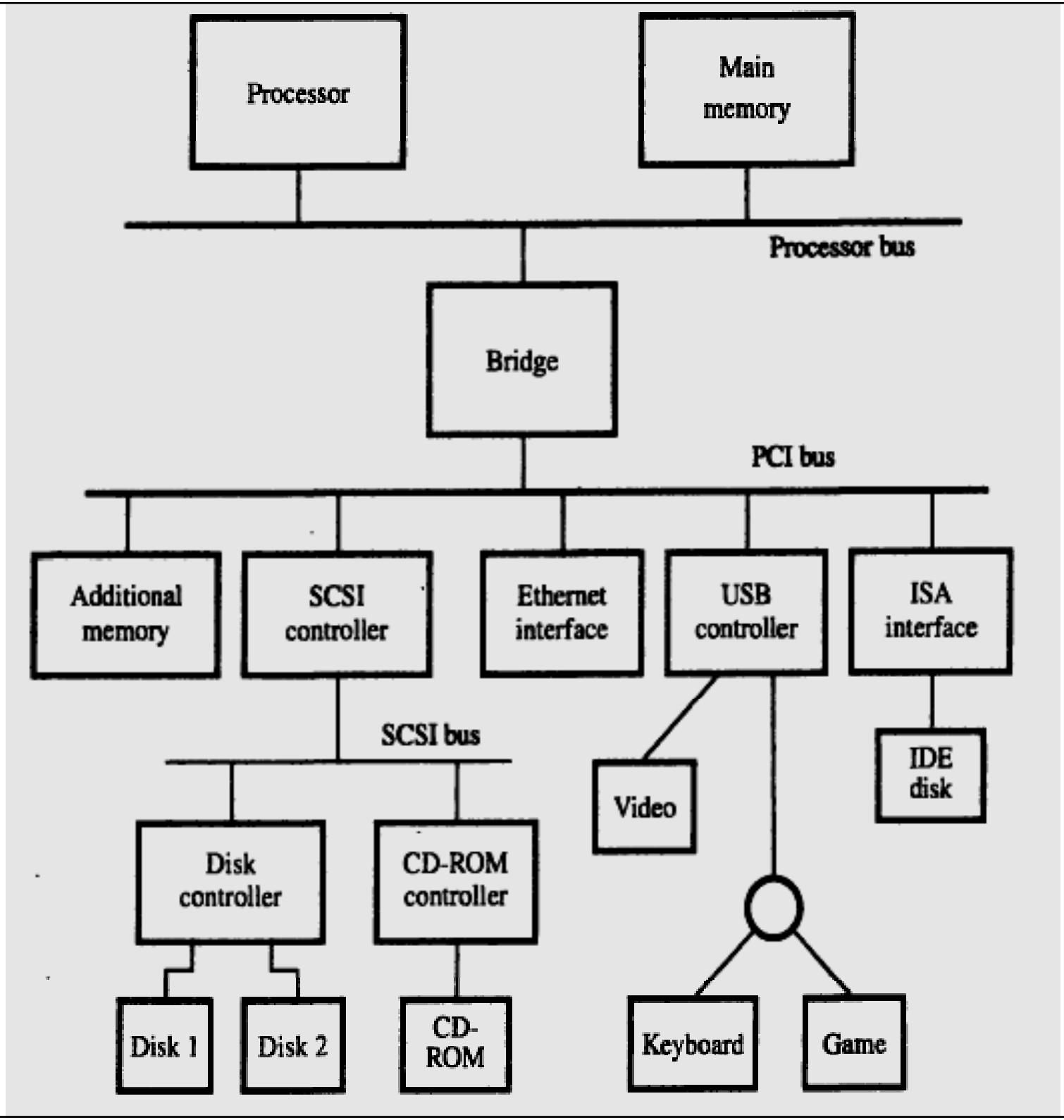
The code seen by both devices is 0111.

Each device compares the pattern on the arbitration lines to its own ID, starting from the most significant bit.

If it detects a difference at any bit position, it disables its drivers at that bit position and for all lower-order bits. It does so by placing a 0 at the input of these drivers.

In the case of our example, device A detects a difference on line ARB I. Hence, it disables its drivers on lines ARB 1 and ARBO.

This causes the pattern on the arbitration lines to change to


Processor
Main memory
Processor bus
Bridge
PCI bus
Additional memory
SCSI controller
Ethernet interface
USB controller
ISA interface
SCSI bus
Video
IDE disk
Disk controller
CD-ROM controller
Disk 1
Disk 2
CD-ROM
Keyboard
Game


IBM

Universal Serial Bus (USB)

The USB supports two speeds of operation, called low-speed (1.5 megabits/s) and full-speed (12 megabits/s).

The most recent revision of the bus specification (USB 2.0) introduced a third speed of operation, called high-speed (480 megabits/s).

The USB has been designed to meet several key objectives:

- Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer

- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections

- Enhance user convenience through a ''plug-and-play'' mode of operation

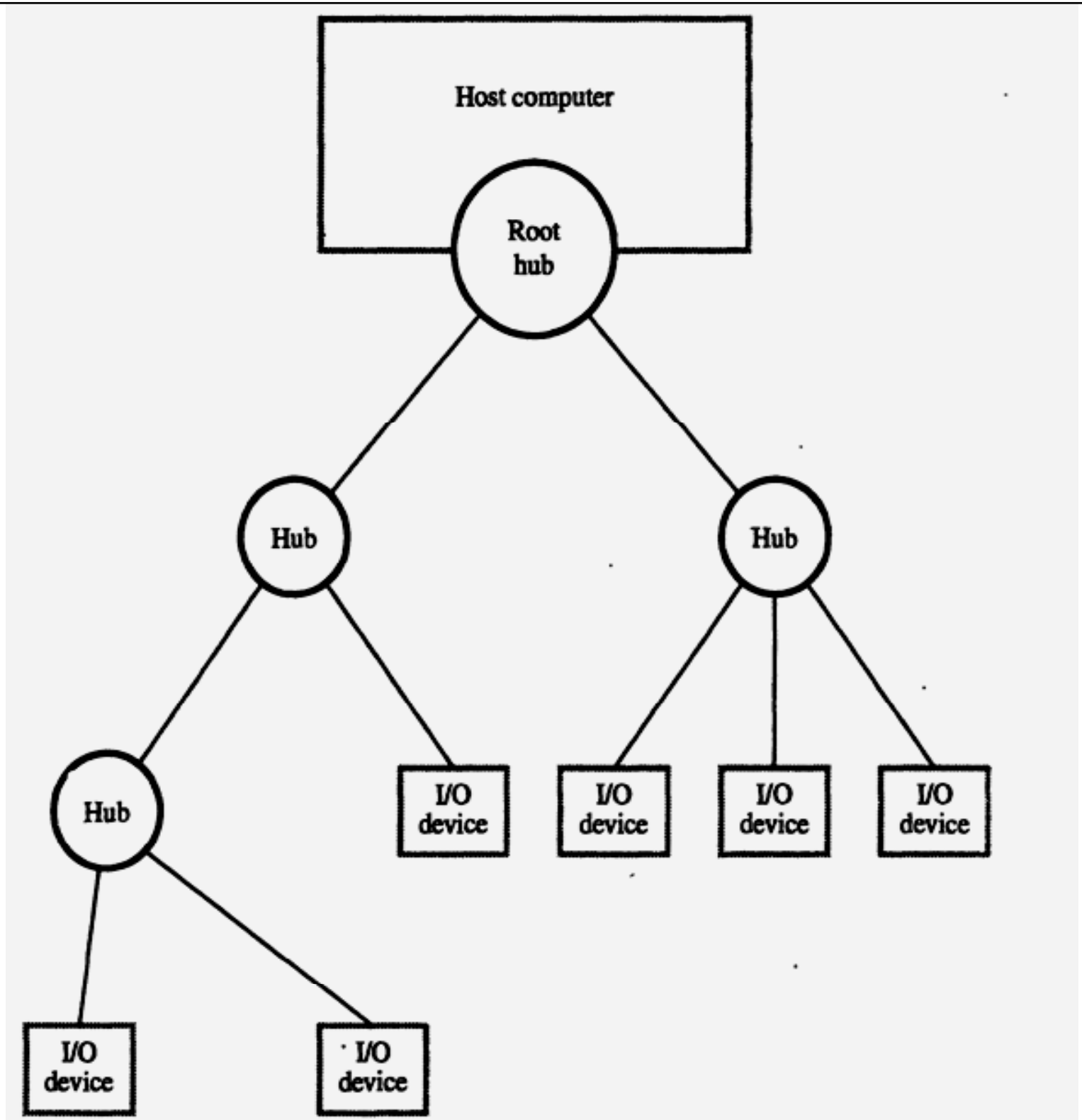
USB Bandwidths:

- A *low-speed* rate of 1.5 Mbit/s (~183 kB/s) is defined by USB 1.0. It is intended primarily to save cost in low-bandwidth human interface devices (HID) such as keyboards, mice, and joysticks.

- The *full-speed* rate of 12 Mbit/s (~1.43 MB/s) is the basic USB data rate defined by USB 1.1. All USB hubs support full-bandwidth.

- A *high-speed* (USB 2.0) rate of 480 Mbit/s (~57 MB/s) was introduced in 2001. All hi-speed devices are capable of falling back to full-bandwidth operation if necessary; they are backward compatible. Connectors are identical.

SuperSpeed (USB 3.0) rate produces upto 4800 Mbit/s (~572 MB/s or 5 Gbps)


Host computer
Root hub
Hub
Hub
Hub
I/O device
I/O device
I/O device
I/O device
I/O device
I/O device

Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/0 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/0 devices being served. The tree structure enables many devices to be connected while using only simple point-to-point serial links.

Each hub has a number of ports where devices may be connected, including other hubs. 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.

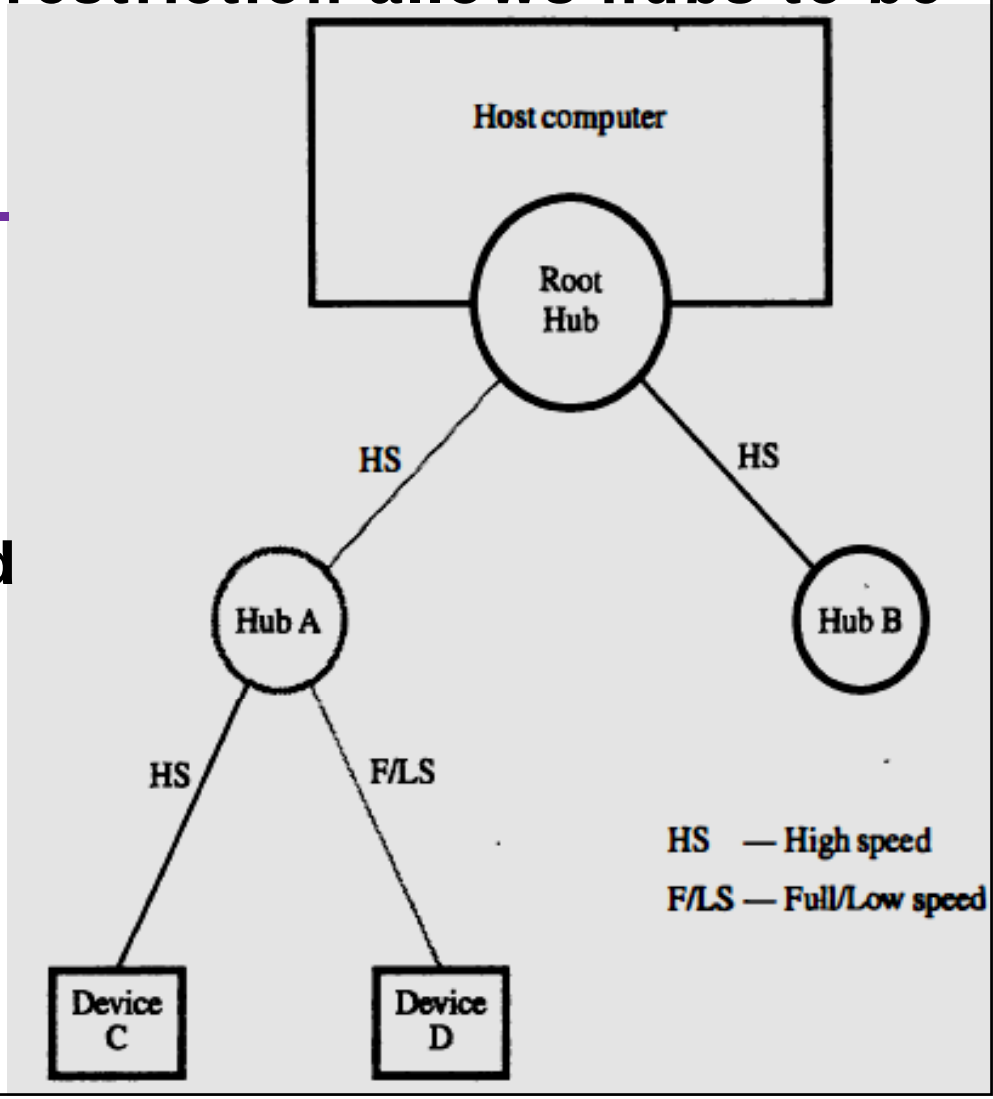
A message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

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.

Hence, upstream messages do not encounter conflicts or interfere with each other, as no two devices can send messages at the same time. This restriction allows hubs to be simple, low-cost devices.

USB protocol requires that a message transmitted on a high-speed link is always transmitted at high speed, even when the ultimate receiver is a low-speed device.

Hence, a message intended for device D is sent at high speed from the root hub to hub A, then forwarded at low speed to device D. The latter transfer will take a long time, during which high-speed traffic to other nodes is allowed to continue.

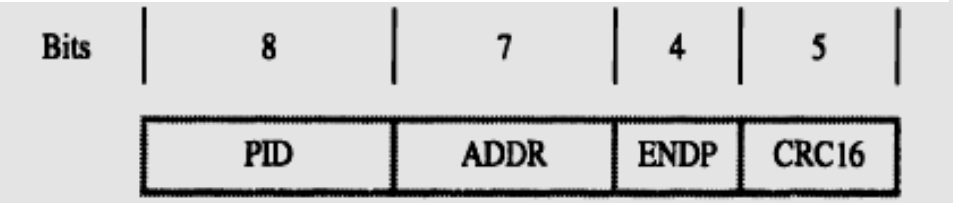


Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.

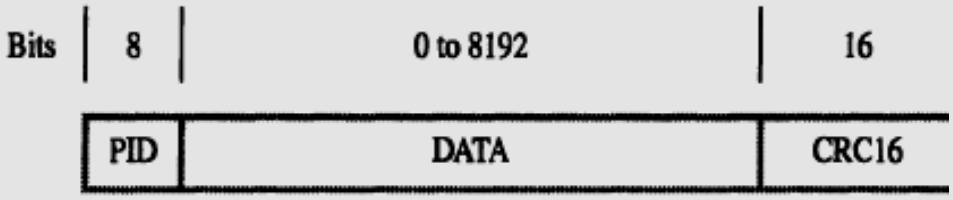
A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily. When a device is first connected to a hub, or when it is powered on, it has the address 0.

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. <*This is key for Plug&PLAY*>



(b) Token packet, IN or OUT



(c) Data packet

Figure 4.45 USB packet formats.

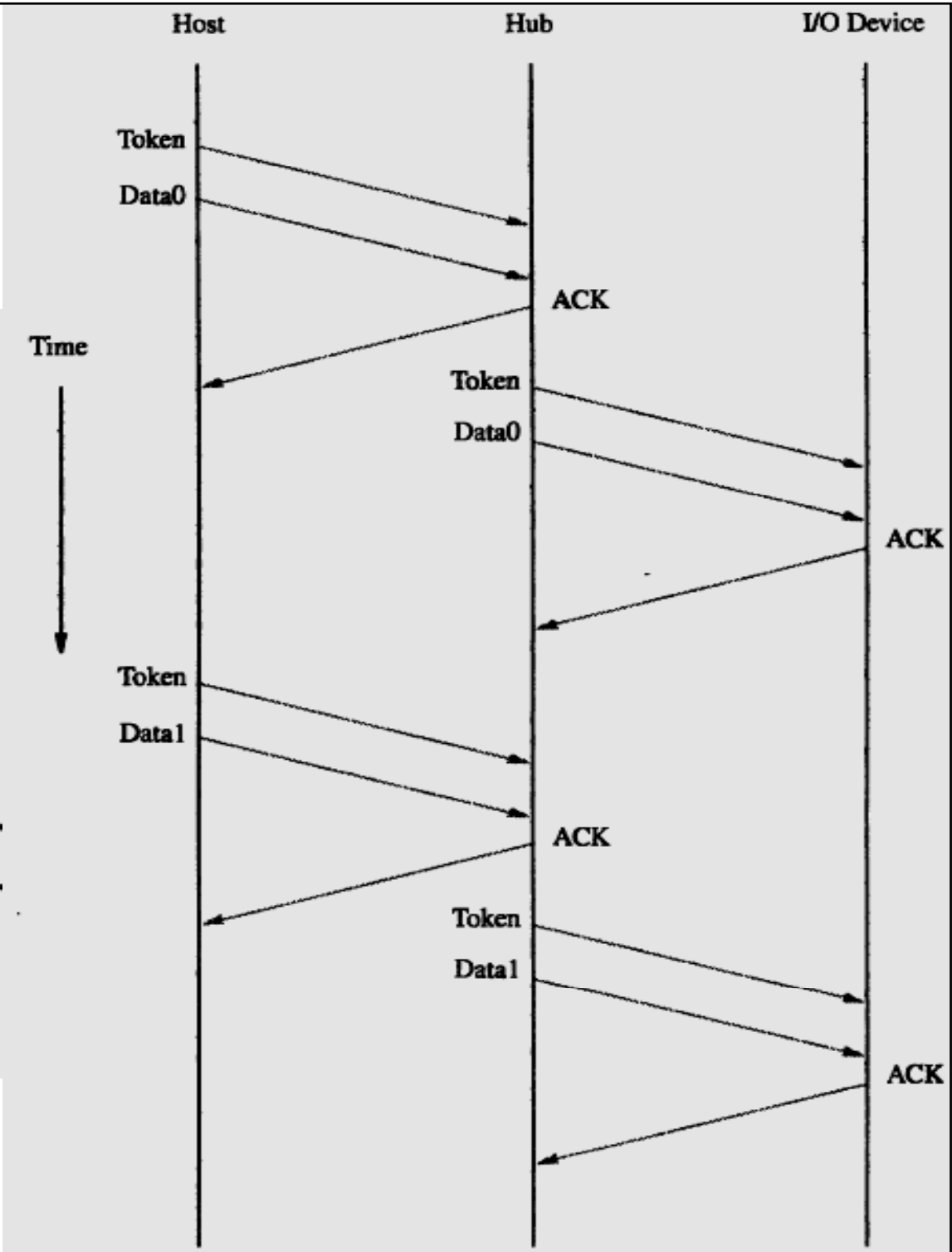


Figure 4.46 An output transfer.

Read about

USB protocols

Isochronous traffic on USB

and

USB FRAME