INPUT-OUTPUT ORGANIZATION

- **Peripheral Devices**
- **Input-Output Interface**
- **Asynchronous Data Transfer**
- **Modes of Transfer**
- **Priority Interrupt**
- **Direct Memory Access**
- **Input-Output Processor**
- **Serial Communication**

PERIPHERAL DEVICES

Input Devices

- **Keyboard**
- **Optical input devices**
  - **- Card Reader**
  - **- Paper Tape Reader**
  - **- Bar code reader**
  - **- Digitizer**
  - **- Optical Mark Reader**
- **Magnetic Input Devices**
  - **- Magnetic Stripe Reader**
- **Screen Input Devices**
  - **- Touch Screen**
  - **- Light Pen**
  - **- Mouse**
- **Analog Input Devices**

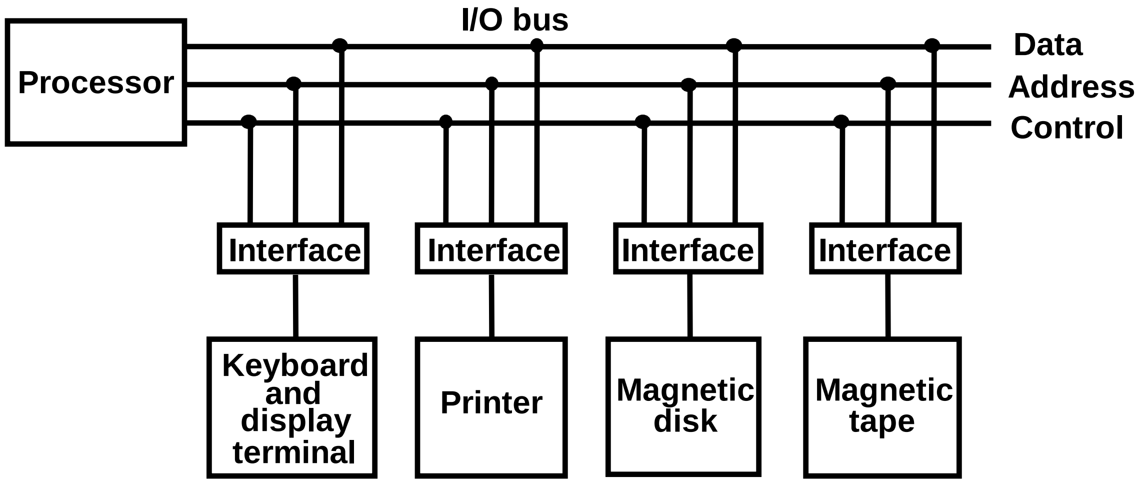
Output Devices

- **Card Puncher, Paper Tape Puncher**
- **CRT**
- **Printer (Impact, Ink Jet, Laser, Dot Matrix)**
- **Plotter**
- **Analog**
- **Voice**

INPUT/OUTPUT INTERFACE

- Provides a method for transferring information between internal storage and external I/O devices.
- Peripherals connected to a computer need special communication links for interfacing them with CPU.
- The purpose of communication link is to resolve the differences between the computer and peripheral devices.
- The major differences are:
  - Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory which are electronic devices.
  - The data transfer rate of peripherals is usually slower than the transfer rate of the CPU and consequently a synchronization mechanism may be needed.
  - Data codes and formats in peripherals differ from the word format in the CPU and memory.
  - The Operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to CPU.
- To resolve these differences computer systems include special hardware components between CPU and peripherals to supervise and synchronize all input and output transfers.
- These components are called interface units.

I/O BUS AND INTERFACE MODULES



✓Each peripheral has an interface module associated with it .
✓Each peripheral has it’s own controller that operates particular electromechanical device.
✓I/O bus from the processor is attached to all peripheral interfaces.
✓To communicate with a particular device processor places a device address on the address lines..
✓Each interface attached to I/O bus contains an address decoder that monitors the address lines.
✓When an interface detects its own address it activates the path between bus lines and device that it controls.

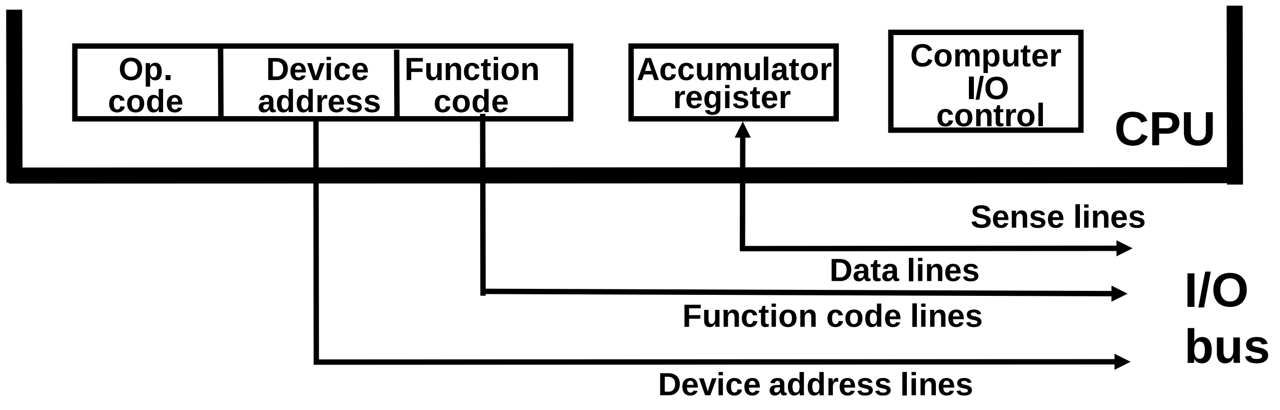
Typical I/O instruction

Op. code	Device address	Function code

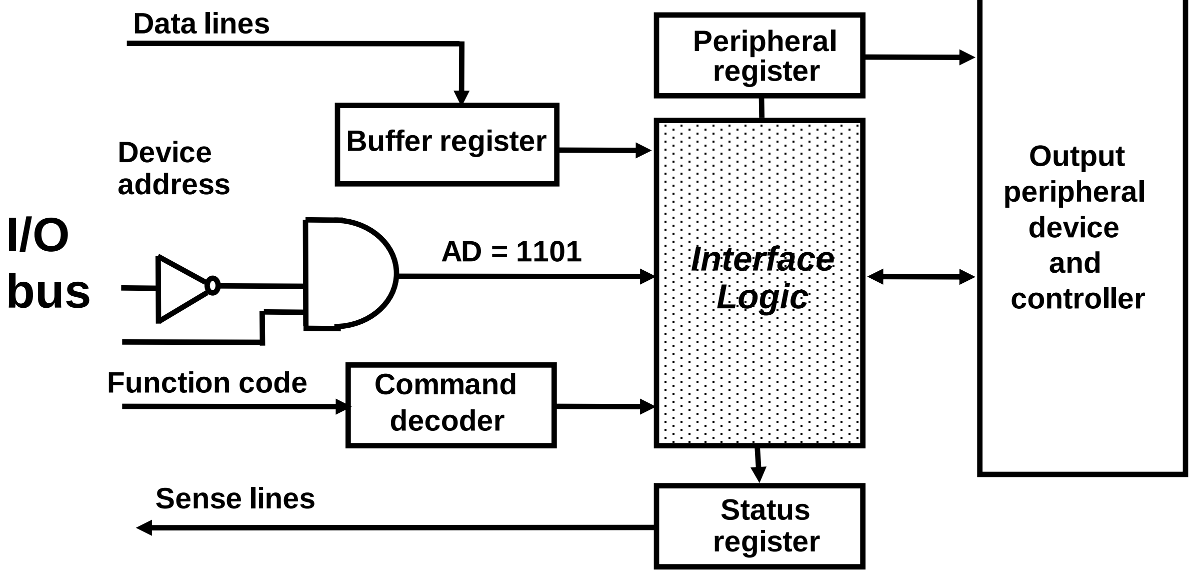
(Command)

CONNECTION OF I/O BUS

Connection of I/O Bus to CPU



Connection of I/O Bus to One Interface



I/O BUS AND MEMORY BUS

- MEMORY BUS is for information transfers between CPU and the MM.
- I/O BUS is for information transfers between CPU & I/O devices through their I/O interface.
- There are three ways that computer buses can be used to communicate with memory and I/o:
  - Use two separate buses, one for memory and the other for I/O.
  - Use one common bus for both memory and I/O but have separate control lines fro each.
  - Use one common bus for memory and I/O with common control lines.
- In the first method computer has independent sets of data, address and control buses, one for accessing memory and other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to CPU.
- An interface connected to a peripheral device may have a number of *data registers* , a *control register*, and a *status register*
- A command is passed to the peripheral by sending to the appropriate interface register.
- Function code and sense lines are not needed (Transfer of data, control, and status information is always via the common I/O Bus)

ISOLATED vs MEMORY MAPPED I/O

Isolated I/O

- **Separate I/O read/write control lines in addition to memory read/write control lines**
- **Separate (isolated) memory and I/O address spaces**
- **Distinct input and output instructions**

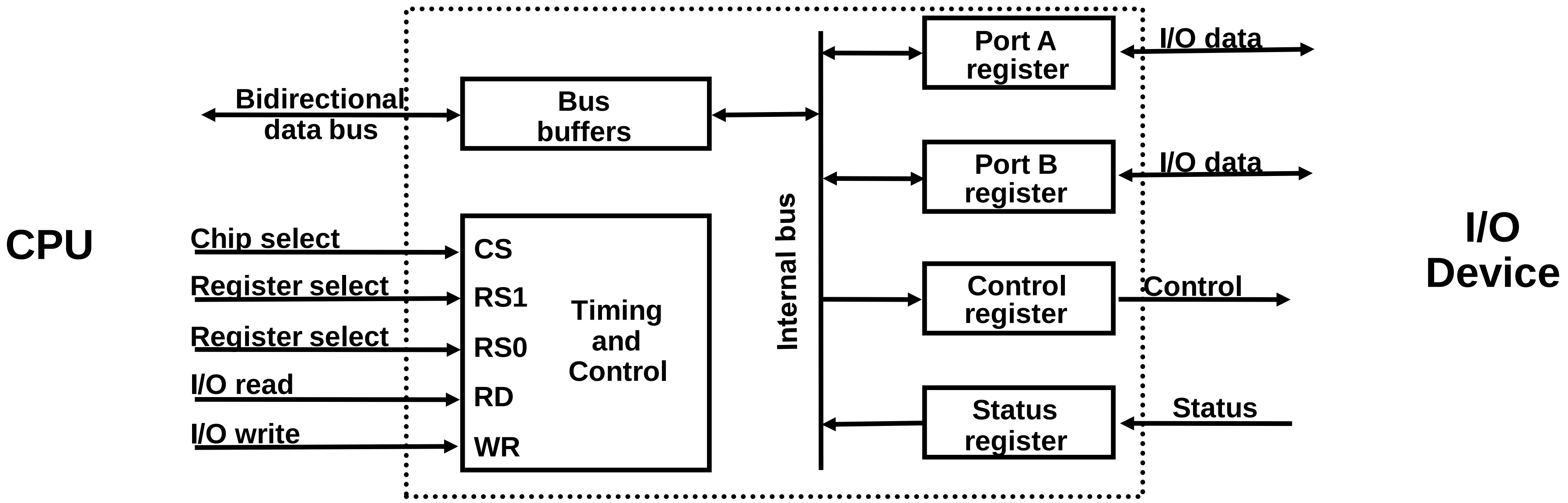
Memory-mapped I/O

- **A single set of read/write control lines (no distinction between memory and I/O transfer)**
- **Memory and I/O addresses share the common address space**

  -> reduces memory address range available
- **No specific input or output instruction**

  -> The same memory reference instructions can be used for I/O transfers
- **Considerable flexibility in handling I/O operations**

I/O INTERFACE



CS	RS1	RS0	Register selected
0	x	x	None - data bus in high-impedence
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Programmable Interface

- **Information in each port can be assigned a meaning depending on the mode of operation of the I/O device**
  → Port A = Data; Port B = Command; Port C = Status

- **CPU initializes(loads) each port by transferring a byte to the Control Register**
  → Allows CPU can define the mode of operation of each port
  → *Programmable Port*: By changing the bits in the control register, it is possible to change the interface characteristics

ASYNCHRONOUS DATA TRANSFER

Synchronous and Asynchronous Operations

Synchronous - All devices derive the timing information from common clock line

Asynchronous - No common clock

Asynchronous Data Transfer

Asynchronous data transfer between two independent units requires that *control signals* be transmitted between the communicating units *to indicate the time at which data is being transmitted*

Two Asynchronous Data Transfer Methods

Strobe pulse

- **A strobe pulse is supplied by one unit to indicate the other unit when the transfer has to occur**

Handshaking

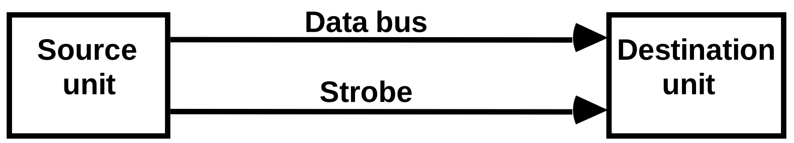
- **A control signal is accompanied with each data being transmitted to indicate the presence of data**
- **The receiving unit responds with another control signal to acknowledge receipt of the data**

STROBE CONTROL

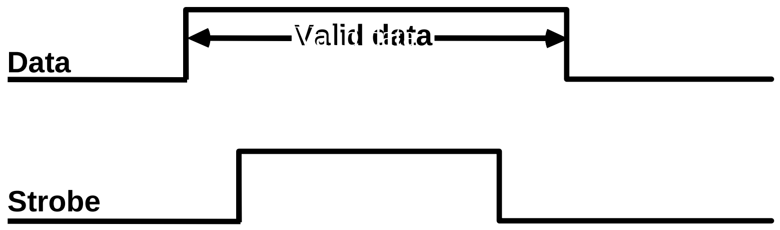
* Employs a single control line to time each transfer
* The strobe may be activated by either the source or the destination unit

Source-Initiated Strobe for Data Transfer

Block Diagram

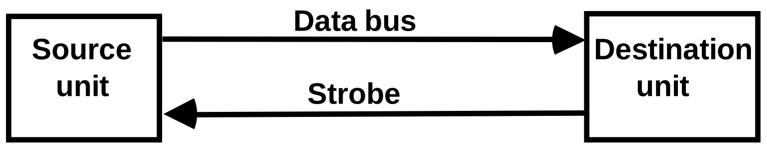


Timing Diagram

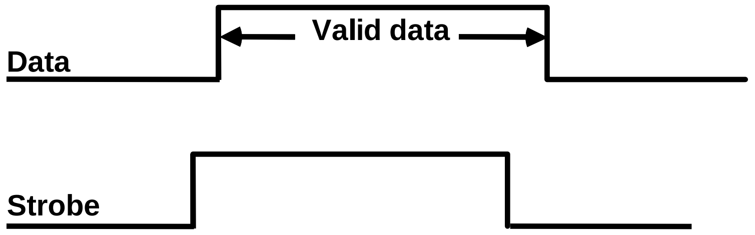


Destination-Initiated Strobe for Data Transfer

Block Diagram



Timing Diagram



HANDSHAKING

Strobe Methods

Source-Initiated

The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received data

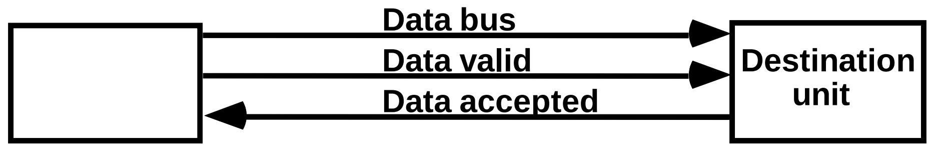
Destination-Initiated

The destination unit that initiates the transfer no way of knowing whether the source has actually placed the data on the bus

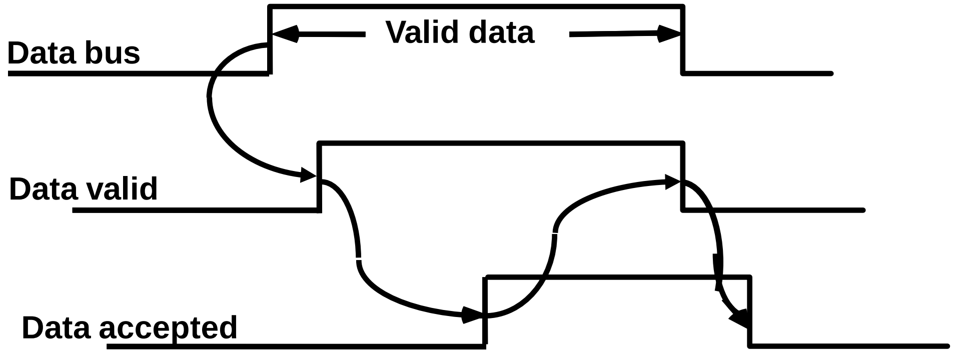
To solve this problem, the *HANDSHAKE* method introduces a second control signal to provide a *Reply* to the unit that initiates the transfer

SOURCE-INITIATED TRANSFER USING HANDSHAKE

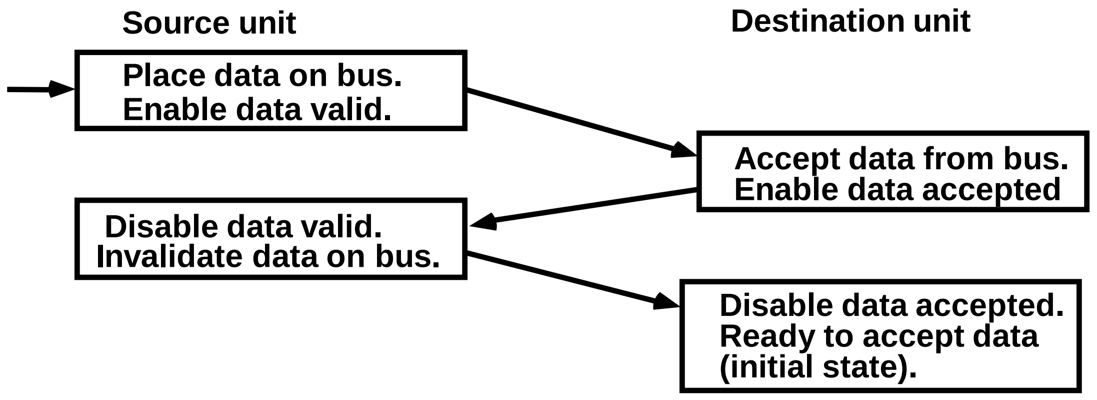
Block Diagram



Timing Diagram



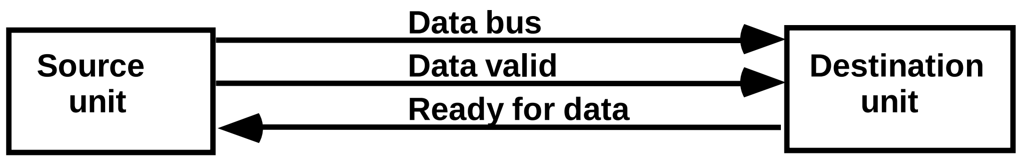
Sequence of Events



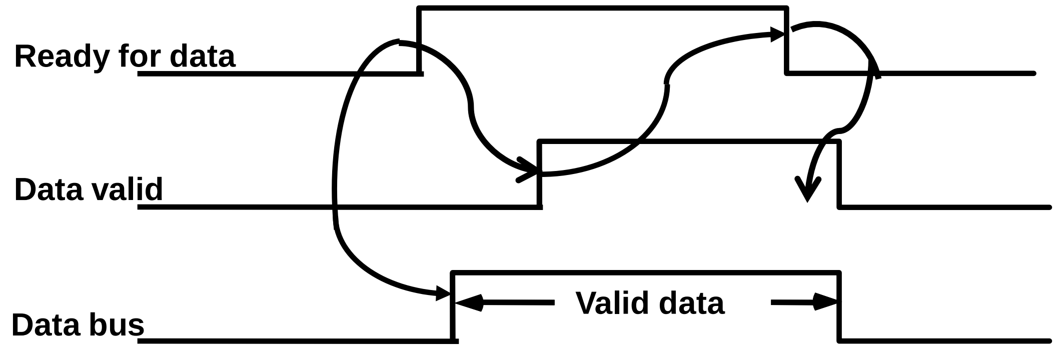
•Allows arbitrary delays from one state to the next
•Permits each unit to respond at its own data transfer rate
•The rate of transfer is determined by the slower unit

DESTINATION-INITIATED TRANSFER USING HANDSHAKE

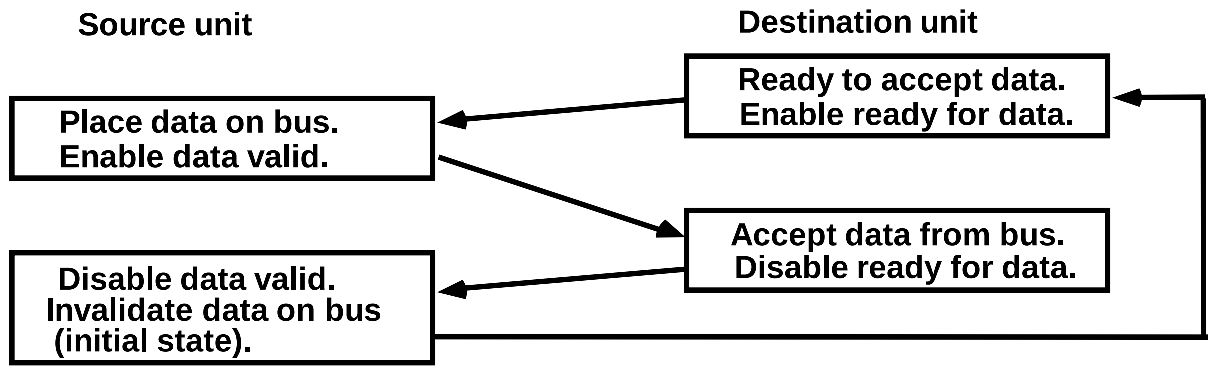
Block Diagram



Timing Diagram



Sequence of Events


Source unit
Destination unit
Ready to accept data. Enable ready for data.
Place data on bus. Enable data valid.
Accept data from bus. Disable ready for data.
Disable data valid. Invalidate data on bus (initial state).

*** Handshaking provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units**

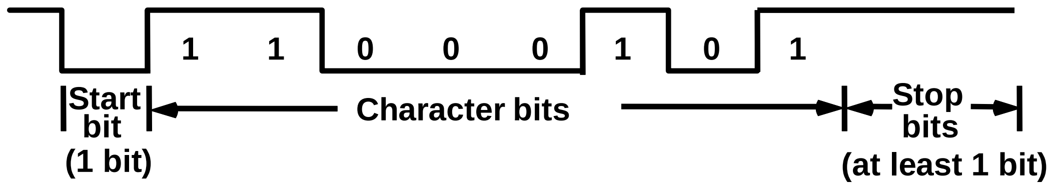
*** If one unit is faulty, data transfer will not be completed**
-> Can be detected by means of a *timeout* mechanism

ASYNCHRONOUS SERIAL TRANSFER

Four Different Types of Transfer

Asynchronous Serial Transfer

- Employs special bits which are inserted at both ends of the character code
- Each character consists of three parts; Start bit; Data bits; Stop bits.



• A transmitted character can be detected by the receiver from knowledge of the transmitted rules:

•When data are not being sent, the line is kept in the 1-state (idle state)

•The initiation of a character transmission is detected by a Start Bit , which is always a 0

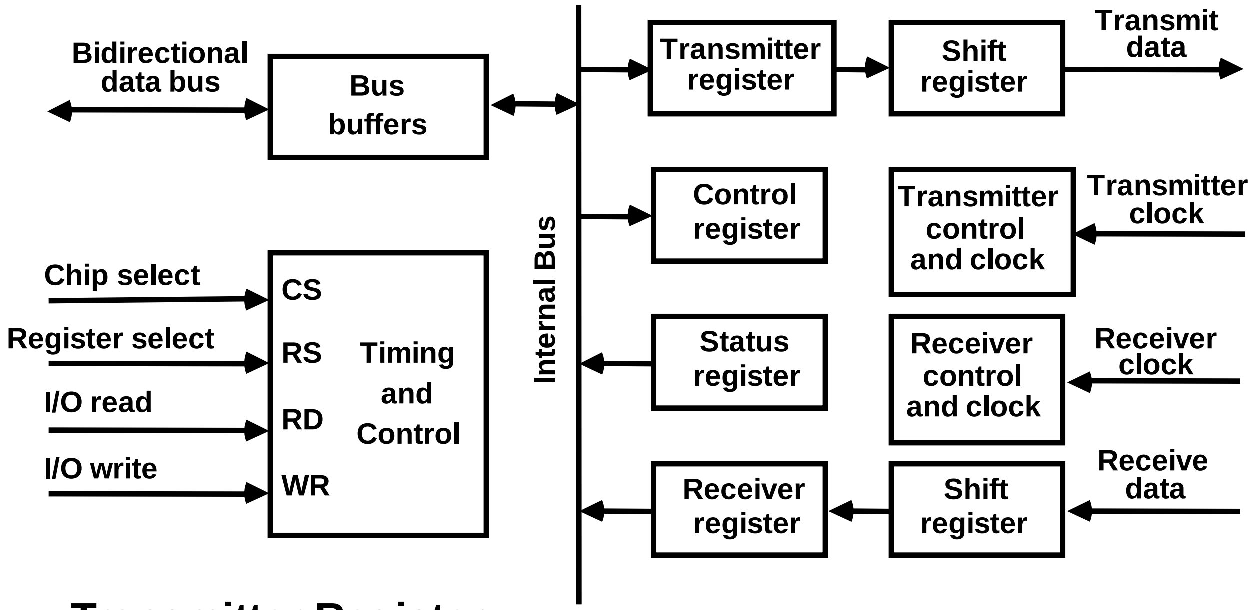
• The character bits always follow the Start Bit

• After the last character , a Stop Bit is detected when the line returns to the 1-state for at least 1 bit time

•The receiver knows in advance the transfer rate of the bits and the number of information bits to expect.

UNIVERSAL ASYNCHRONOUS RECEIVER-TRANSMITTER - UART -

A typical asynchronous communication interface available as an IC



CS	RS	Oper.	Register selected
0	x	x	None
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Transmitter Register

- **Accepts a data byte(from CPU) through the data bus**
- **Transferred to a shift register for serial transmission**

Receiver

- **Receives serial information into another shift register**
- **Complete data byte is sent to the receiver register**

Status Register Bits

- **Used for I/O flags and for recording errors**

Control Register Bits

- **Define baud rate, no. of bits in each character, whether to generate and check parity, and no. of stop bits**

FIRST-IN-FIRST-OUT(FIFO) BUFFER

- Important feature is it can Input the data and output the data at two different rates.
- Output data are always in the same order in which the data entered the buffer.
- Useful in some applications when data is transferred asynchronously
- The following diagrams shows a 4 x 4 FIFO Buffer. It consists of 4-bit registers R_i and a control registers with flip-flops F_i, associated with each R_i)

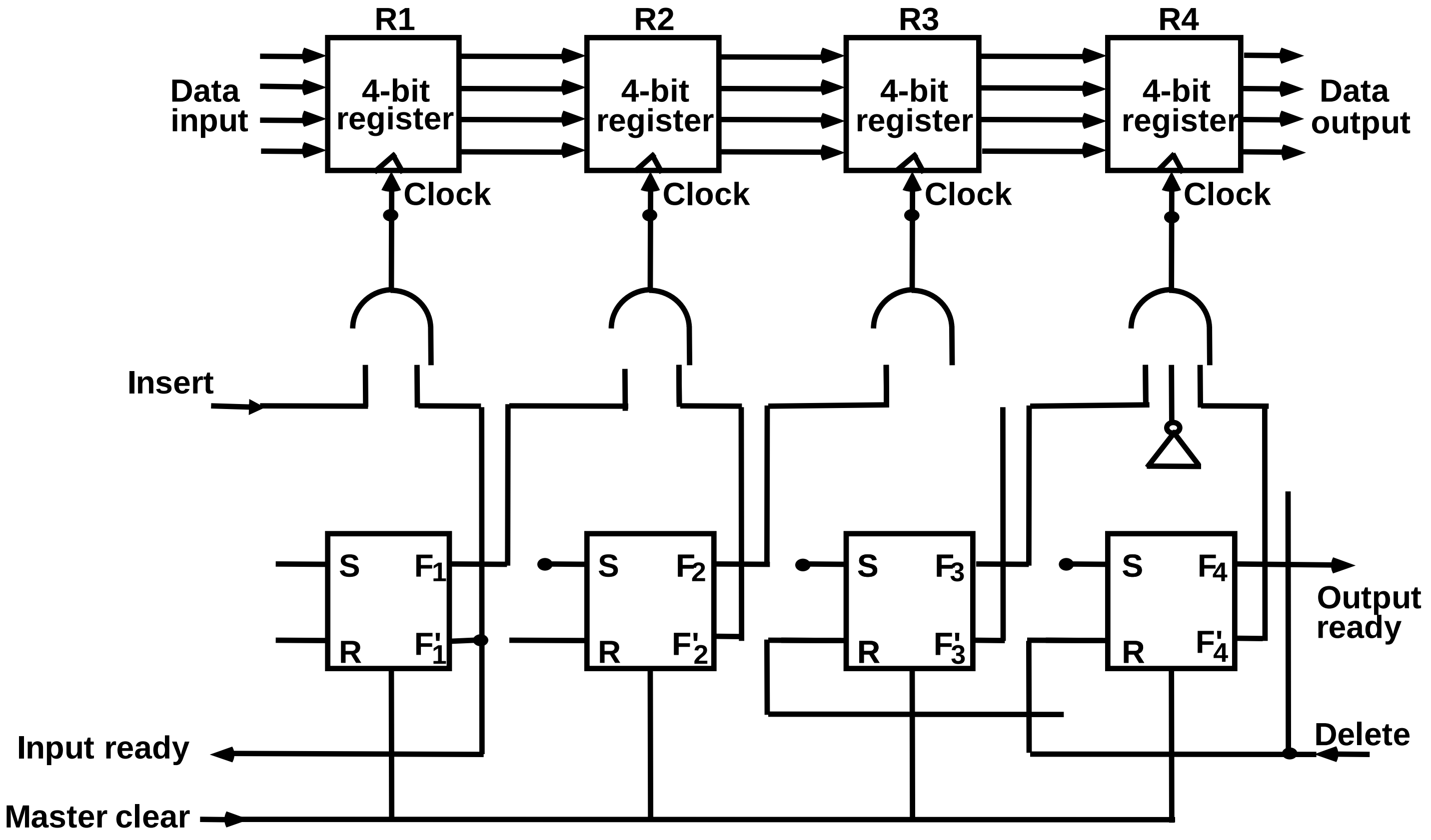


Fig: Circuit diagram of 4x4 FIFO Buffer

MODES OF TRANSFER

- There are 3 different Data Transfer Modes between the central computer (CPU or memory) and peripherals.
- They are :
  1. Program-Controlled I/O
  2. Interrupt-initiated I/O
  3. Direct Memory Access (DMA)

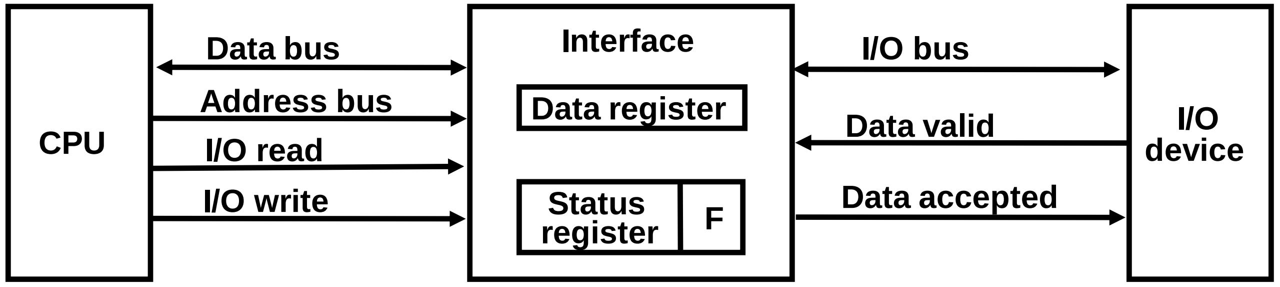


Fig: Data transfer from I/O device to CPU

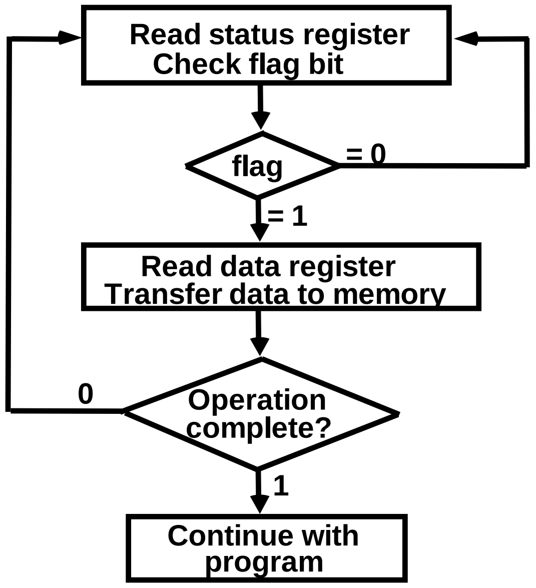


Fig: Flowchart for CPU program to input data

- Programmed I/O operations are result of I/O instructions written in the computer program.
- Each data item transfer is initiated by an instruction in the program.
- Usually the transfer is to and from a CPU register and peripheral.
- Transferring data under program control requires constant monitoring of the peripheral by the CPU.
- Once a data transfer is initiated CPU is required to monitor the interface to see when a transfer can again be made.
- In programmed I/O method CPU stays in a program loop until I/O unit indicates that it is ready for data transfer. This is time consuming process.
- It can be avoided by using an Interrupt request signal when the data are available from the device.
- In the meantime CPU can proceed to execute another program.
- In the meanwhile interface keeps monitoring the device. When it determines that device is ready for data transfer it generates the interrupt request to the computer.

- Up on detecting the external interrupt signal CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer and then returns to the task it was originally performing.
- Transfer of data under programmed I/O is between CPU and peripheral.
- In DMA the interface transfers data into and out of the memory unit through the memory bus.
- CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.

MODES OF TRANSFER - INTERRUPT INITIATED I/O & DMA

Interrupt Initiated I/O

- Polling takes valuable CPU time
- Open communication only when some data has to be passed -> *Interrupt*.
- I/O interface, instead of the CPU, monitors the I/O device
- When the interface determines that the I/O device is ready for data transfer, it generates an *Interrupt Request* to the CPU
- Upon detecting an interrupt, CPU stops momentarily the task it is doing, branches to the service routine to process the data transfer, and then returns to the task it was performing

DMA (Direct Memory Access)

- Large blocks of data transferred at a high speed to or from high speed devices, magnetic drums, disks, tapes, etc.
- DMA controller
Interface that provides I/O transfer of data directly to and from the memory and the I/O device
- CPU initializes the DMA controller by sending a memory address and the number of words to be transferred
- Actual transfer of data is done directly between the device and memory through DMA controller
-> Freeing CPU for other tasks

PRIORITY INTERRUPT

Priority

- **Determines which interrupt is to be served first when two or more requests are made simultaneously**
- **Also determines which interrupts are permitted to interrupt the computer while another is being serviced**
- **Higher priority interrupts can make requests while servicing a lower priority interrupt**

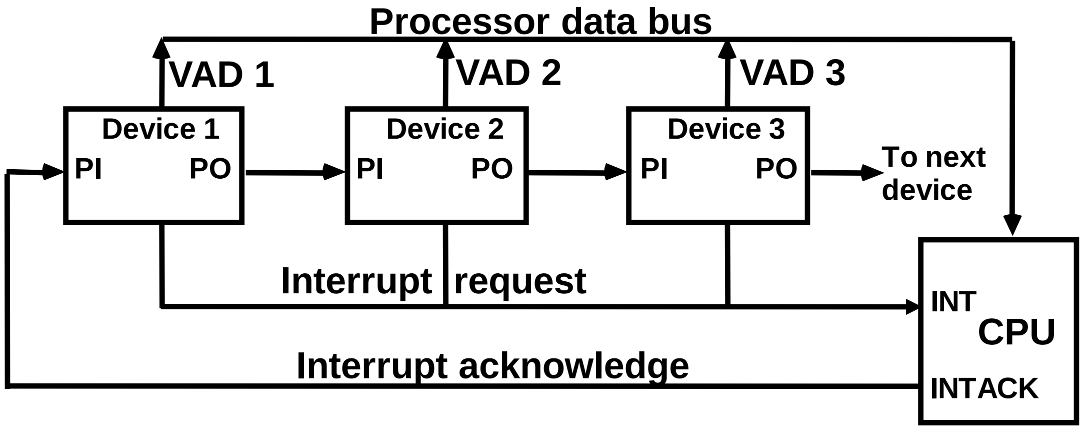
Priority Interrupt by Software(Polling)

- **Priority is established by the order of polling the devices(interrupt sources)**
- **Flexible since it is established by software**
- **Low cost since it needs a very little hardware**
- **Very slow**

Priority Interrupt by Hardware

- **Require a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request**
- **Fast since identification of the highest priority interrupt request is identified by the hardware**
- **Fast since each interrupt source has its own interrupt vector to access directly to its own service routine**

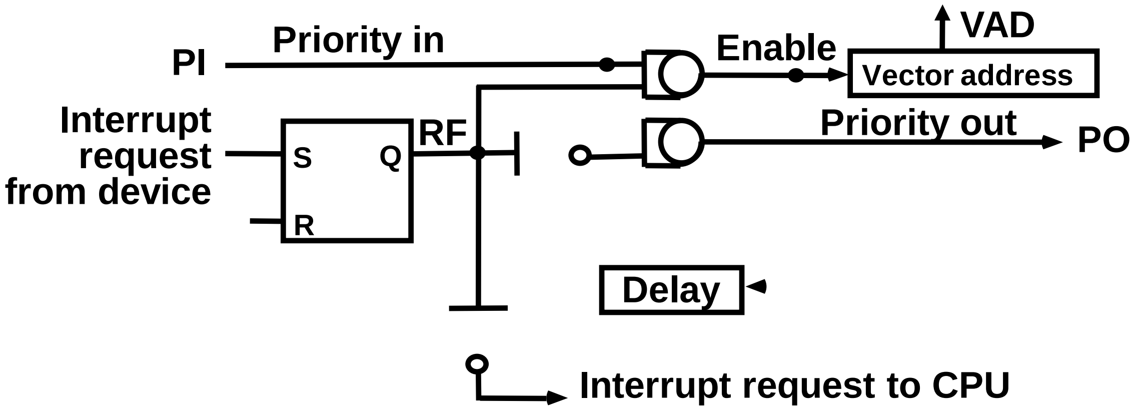
HARDWARE PRIORITY INTERRUPT - DAISY-CHAIN



VAD → Vector Address
PI → Priority Input
PO → Priority Output

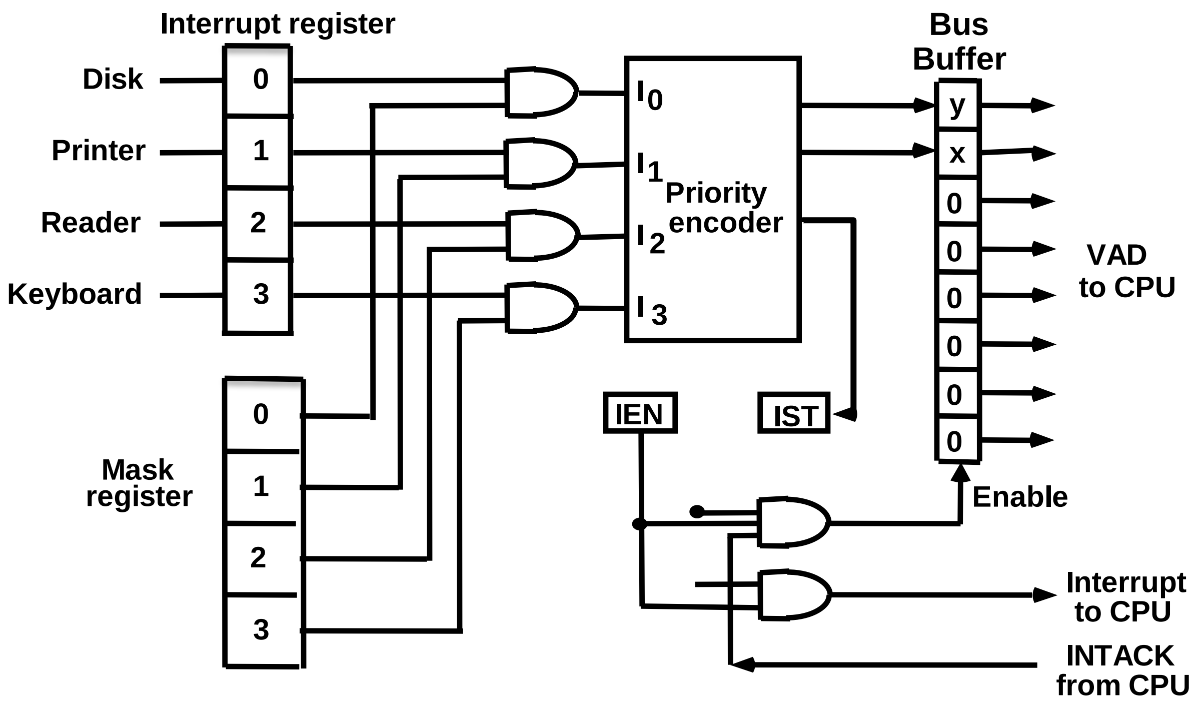
•A device i.e. Requesting an interrupt and has a 1 in PI input will intercept the acknowledge signal by placing a 0 n its PO output.
•If the device does not have pending interrupts, it transmits the acknowledge signal to next device by placing 1 in its PO output. Thus a device with PI=1 and PO=0 is the one with highest priority i.e. Requesting an interrupt and this device places its VAD on the data bus.

One stage of the daisy chain priority arrangement



PI	RF	PO	Enable
0	0	0	0
0	1	0	0
1	0	1	0
1	1	1	1

PARALLEL PRIORITY INTERRUPT



IEN → Interrupt Enable
IST → Interrupt Status

- ➢Parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device.
- ➢Priority is established according to the position of the bits in the register.
- ➢In addition to the interrupt register the circuit may include mask register whose purpose is to control the status of each interrupt request.
- ➢Mask register can be programmed to disable lower priority interrupts while a higher priority device is being serviced.
- ➢It can also provide a facility that allows high priority device to interrupt CPU while a lower priority device is being serviced.

INTERRUPT PRIORITY ENCODER

- Priority encoder is a circuit that implements the priority function.
- The logic of this priority encoder is such that if two or more inputs arrive at the same time, then the input having the highest priority will take precedence.
- **d** in the truth table represents a don't care conditions. So regardless of other inputs when the input is 1, the output generates a value xy = 00 (for I_0)

Priority Encoder Truth table

Inputs				Outputs			Boolean functions
I_0	I_1	I_2	I_3	x	y	IST	
1	d	d	d	0	0	1	
0	1	d	d	0	1	1	
0	0	1	d	1	0	1	$x = I_0' I_1'$
0	0	0	1	1	1	1	$y = I_0' I_1 + I_0' I_2'$
0	0	0	0	d	d	0	$(IST) = I_0 + I_1 + I_2 + I_3$

IST → Interrupt Status

INTERRUPT CYCLE

At the end of each Instruction cycle

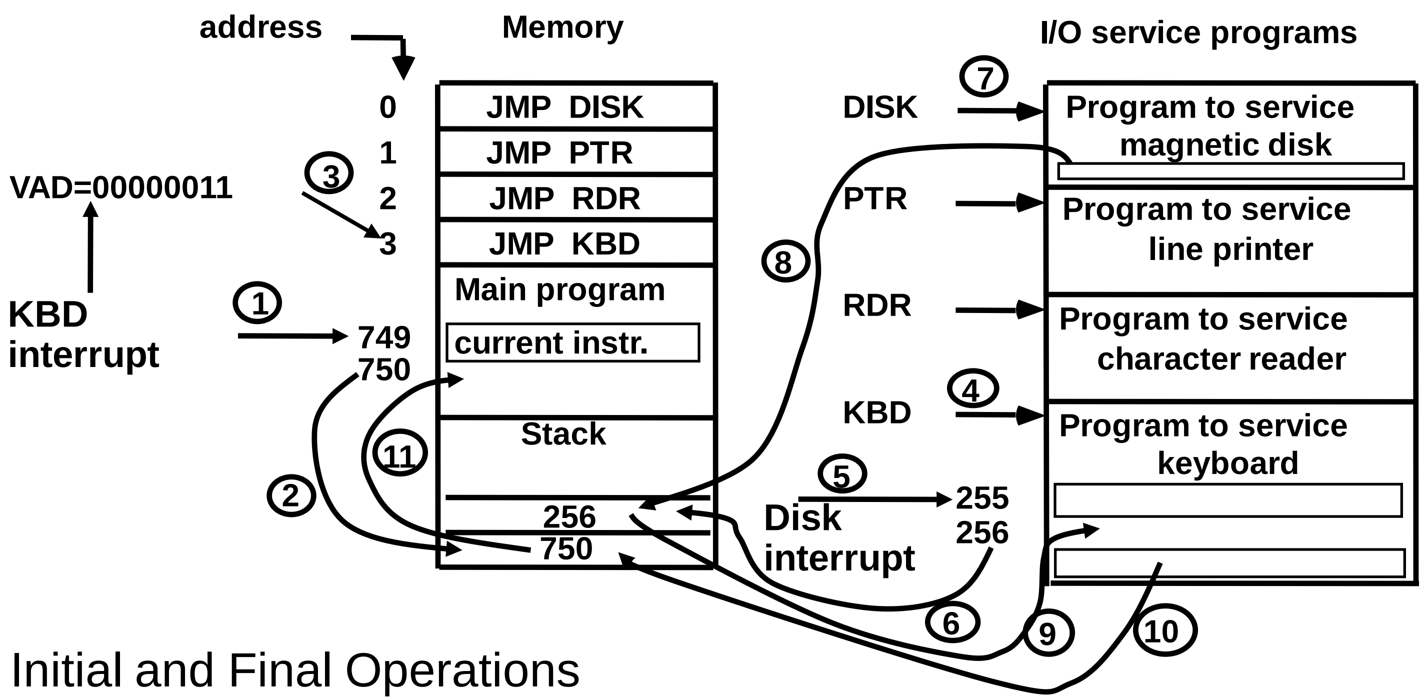
- CPU checks IEN and IST

If either is equal to 0 control continues with the next instruction.

if both are equal to 1 CPU goes to an interrupt cycle. During interrupt cycle CPU performs the following sequence of micro operations.

$SP \leftarrow SP - 1$	**Decrement stack pointer**
$M[SP] \leftarrow PC$	**Push Program Counter into stack**
$INTACK \leftarrow 1$	**Enable interrupt acknowledge**
$PC \leftarrow VAD$	**Transfer vector address to PC**
$IEN \leftarrow 0$	**Disable further interrupts**
Go To Fetch	**to execute the first instruction in the interrupt service routine**

INTERRUPT SERVICE ROUTINE



Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system

Initial Sequence

[1] Clear lower level Mask register bits.
[2] Clear interrupt status bit IST.
[3] Save contents of processor registers
[4] Set interrupt enable bit IEN.
[5] Go to Interrupt Service Routine

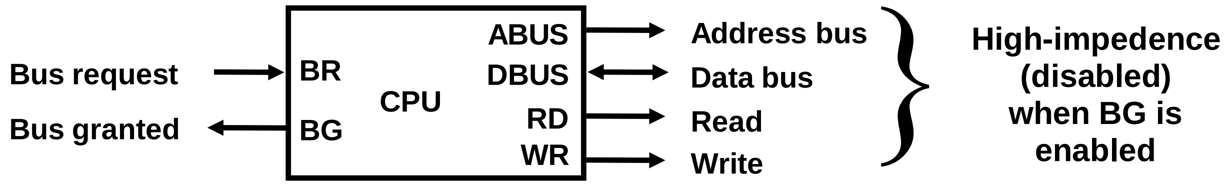
Final Sequence

[1] Clear interrupt enable bit IEN.
[2] Restore contents of processor registers
[3] Clear the bit in the Interrupt register belonging to source that has been serviced
[4] Set lower level Mask reg. Bits
[5] Restore return address, IEN <- 1

DIRECT MEMORY ACCESS

•Block of data transfer from high speed devices, Drum, Disk, Tape
•DMA controller - Interface which allows I/O transfer directly between memory and Device, freeing CPU for other tasks.
•CPU initializes DMA Controller by sending starting address of the memory block, word count (no. of words in memory block), control to specify mode of transfer such as read or write and a control to start DMA transfer.

Fig: CPU bus signals for DMA transfer



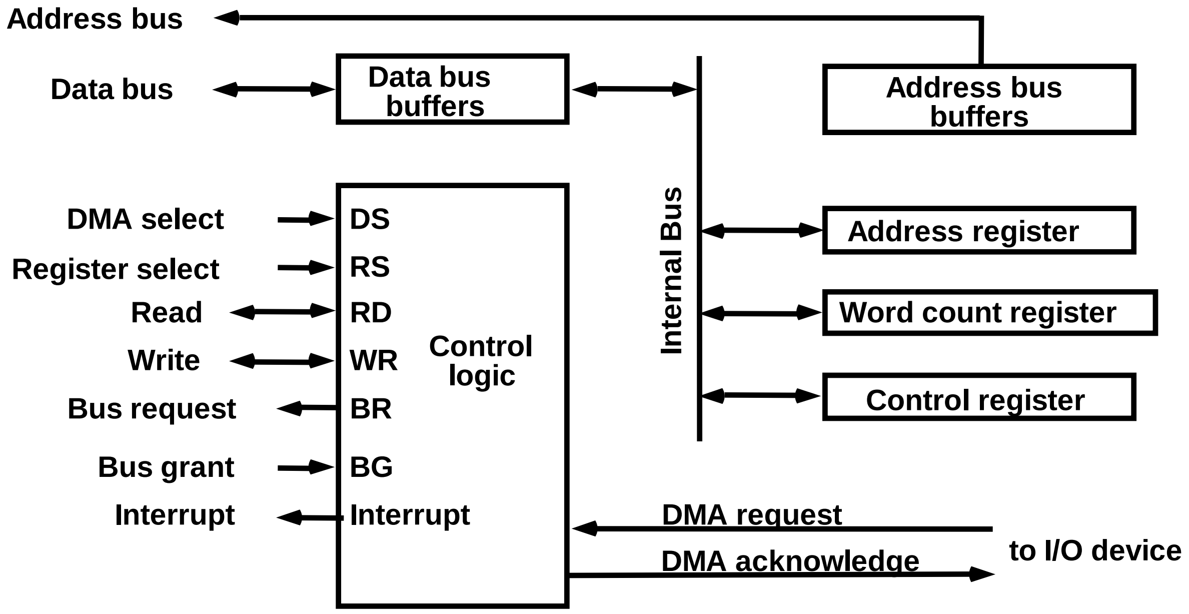


Fig: Block diagram of DMA controller

DMA I/O OPERATION

Starting an I/O

- **CPU executes instruction to**
  - **Load Memory Address Register**
  - **Load Word Counter**
  - **Load Function(Read or Write) to be performed**
  - **Issue a GO command**

Upon receiving a GO Command DMA performs I/O operation as follows independently from CPU

Input

```
[1] Input Device <- R (Read control signal)
[2] Buffer(DMA Controller) <- Input Byte; and
     assembles the byte into a word until word is full
[4] M <- memory address, W(Write control signal)
[5] Address Reg <- Address Reg +1;  WC(Word Counter) <- WC - 1
[6] If WC = 0, then Interrupt to acknowledge done, else go to [1]
```

Output

```
[1] M <- M Address, R
     M Address R <- M Address R + 1, WC <- WC - 1
[2] Disassemble the word
[3] Buffer <- One byte; Output Device <- W, for all disassembled bytes
[4] If WC = 0, then Interrupt to acknowledge done, else go to [1]
```

CYCLE STEALING

While DMA I/O takes place, CPU is also executing instructions

DMA Controller and CPU both access Memory -> Memory Access Conflict

Memory Bus Controller

- **Coordinating the activities of all devices requesting memory access**
- **Priority System**

Memory accesses by CPU and DMA Controller are interwoven, with the top priority given to DMA Controller
-> Cycle Stealing

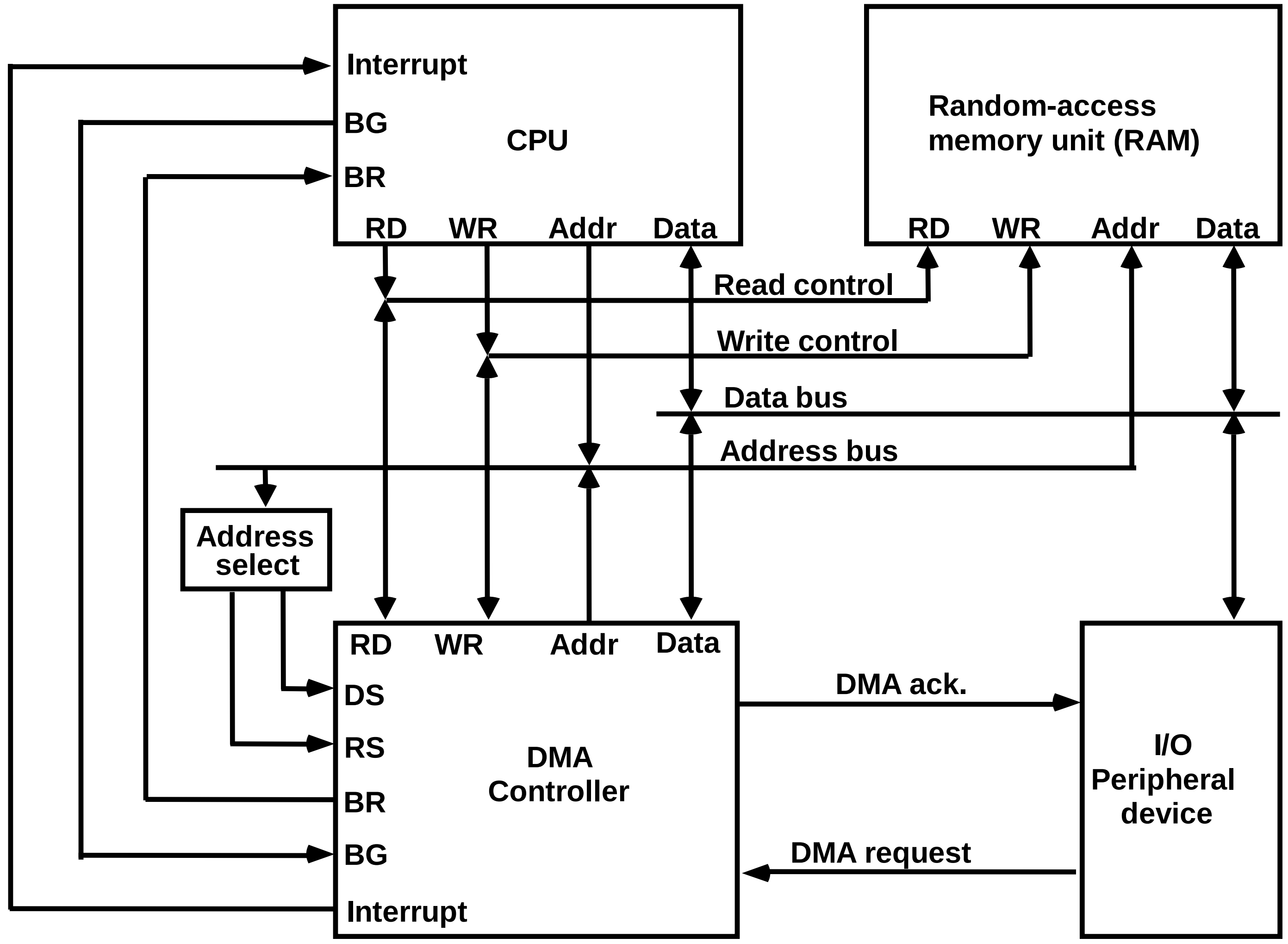
Cycle Steal

- **CPU is usually much faster than I/O(DMA), thus CPU uses the most of the memory cycles**
- **DMA Controller steals the memory cycles from CPU**
- **For those stolen cycles, CPU remains idle**
- **For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time**

DMA TRANSFER

- CPU communicates with the DMA through the address and data buses as with any interface unit.
- DMA has it’s own address which activates the DS (DMA Select)and RS (Register Select)lines.
- CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start transfer between peripheral device and memory.
- When peripheral device sends DMA request , DMA controller activates the BR(bus request) line informing CPU to relinquish the buses.
- CPU responds with its BG(bus grant) line informing the DMA that its buses are disabled.
- DMA then puts the current value of its address register in to the address bus, initiates the RD or WR signal and sends a DMA acknowledge to the peripheral device.

DMA TRANSFER



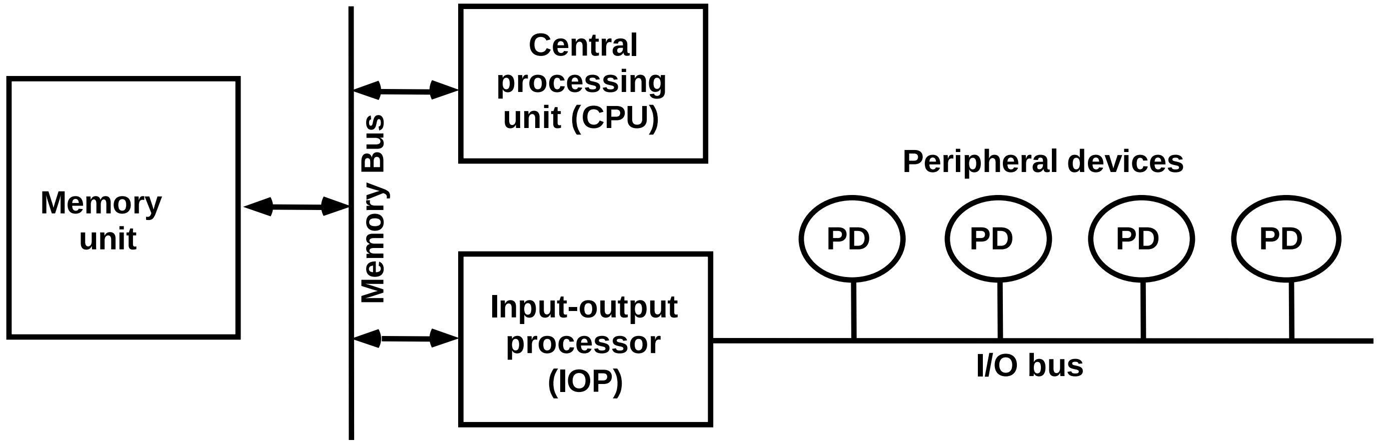
<u>Note:</u> RD and WR lines in DMA controller are bidirectional.

DMA TRANSFER

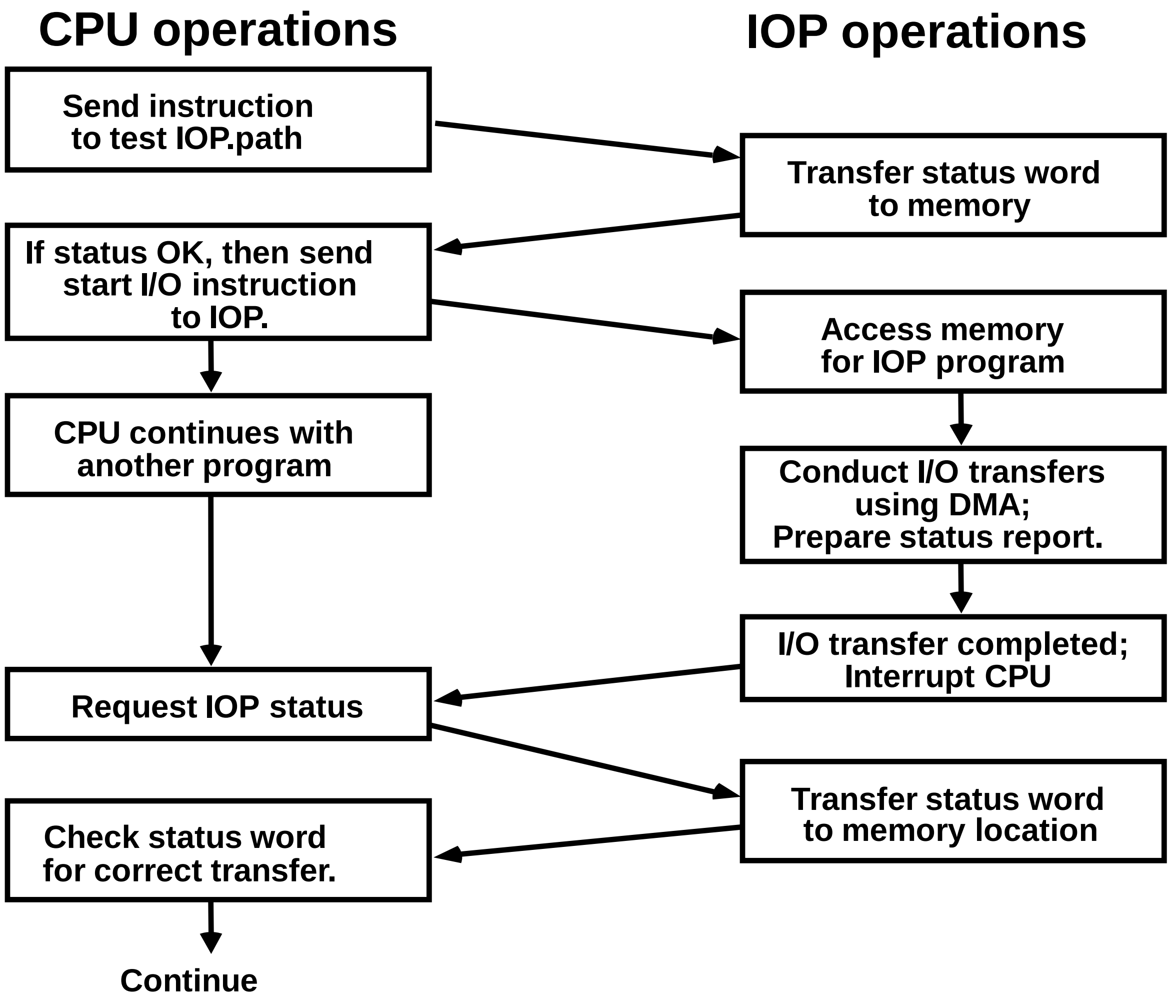
- The direction of transfer depends on the status of the BG line.
- When BG = 0 then
  - RD and WR are input lines allowing the CPU to communicate with the internal DMA registers.
- When BG = 1 then
  - RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation for the data.
- When the peripheral device receives DMA acknowledge, it puts a word in the data bus (for write) or receives a word from data bus (for read).
- Thus DMA controls read or write operations and supplies the address for the memory.

INPUT/OUTPUT PROCESSOR (IOP)

- Instead of having each interface communicate with CPU, a computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices.
- An IOP can be classified as a processor with direct memory access capability that communicates with I/O devices.
- IOP is similar to CPU except that it is designed to handle the details of I/O processing.
- Unlike DMA controller that must be set up entirely by the CPU, IOP can fetch and execute its own instructions.
- IOP instructions are specifically designed to facilitate I/O transfers.
- In addition IOP can perform other processing tasks like arithmetic, logic, branching and code translation.
- Block diagram of computer with two processors is shown in fig below:



CPU-IOP COMMUNICATION


CPU operations
IOP operations
Send instruction to test IOP.path
Transfer status word to memory
If status OK, then send start I/O instruction to IOP.
Access memory for IOP program
CPU continues with another program
Conduct I/O transfers using DMA; Prepare status report.
I/O transfer completed; Interrupt CPU
Request IOP status
Transfer status word to memory location
Check status word for correct transfer.
Continue

SERIAL COMMUNICATION

- A data communication processor is an I/O processor that distributes and collects data from many remote terminals connected through telephone and other communication lines.
- It is a specialized I/O processor designed to communicate directly with data communication networks.
- A communication network may consist of any of a wide variety of devices like printers , interactive display devices , and so on.
- With the use of data communication processor the computer can service fragments of each network demand in an interspersed (combined) manner.
- Most significant difference between I/O processor and data communication processor is in the way the processor communicates with I/O devices.
- An I/O processor communicates with peripherals through a common I/O bus i.e. comprised of many data and control lines.
- A data communication processor communicates with each terminal through a single pair of wires. Both data and control information are transferred in a serial fashion with the result that transfer rate is much slower.

SERIAL COMMUNICATION

- The task of data communication processor is to transmit and collect digital information to and from each terminal.
- Another task is to determine whether the information is data or control and respond to all requests according to predetermined established procedures.
- The processor also communicate with CPU and memory in the same manner as any I/O processor.
- The way the remote terminals are connected to a data communication processor is through telephone lines or other public or private communication facilities.
- Since telephone lines were originally designed for voice communication and computers communicate in terms of digital signals, some form of converters must be used.
- Converters are called as data sets, acoustic couplers or modems.
- For example a modem converts digital signals into audio tones to be transmitted over telephone lines and also converts audio tones from the lines to digital signals for machine use.

SERIAL COMMUNICATION

- A Communication line may be connected to synchronous or asynchronous interface depending on the transmission method of the remote terminal.
- An asynchronous interface receives serial data with start and stop bits in each character.
- High speed devices use synchronous transmission to realize the efficient use of communication links as it does not use start-stop bits to frame characters.
- In synchronous transmission where an entire block of characters is transmitted, each character has a parity bit for the receiver to check.
- After the entire block is sent the transmitter sends one more character that constitutes a parity over the length of the message.
- Another method used for checking errors in transmission is the cyclic redundancy check (CRC).
- Data can be transmitted between two points in three different modes:
  - Simplex → carries information in one line only
  - Half-duplex → transmitted information in both directions but one at a time.
  - Full-duplex→ transmitted information in both directions simultaneously.
- Communication lines, modems etc.. used in the transmission of information between two or more stations is called Data Link.
- The orderly transfer of information in a data link is accomplished by means of a protocol.

Peripheral Component Interconnect (PCI)

- PCI bus was first introduced in 1992.
- PCI bus supports the functions found on a processor bus but in a standardized format that is independent of any particular processor.
- Devices connected to PCI bus appear to processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.
- PCI follows a sequence of bus standards that were used primarily in IBM PC’s. Early PC’s used the 8-bit XT bus whose signals closely mimicked those of Intel's 80x86 processors.
- Later 16-bit bus used on PC, AT computers became known as the ISA bus. It’s extended 32-bit version is known as the EISA bus.
- PCI was developed as low cost bus that is truly processor independent.
- It’s design expected a rapidly growing demand for bus bandwidth to support high-speed disks and graphic and video devices.
- An important feature of PCI is a plug-and-play capability for connecting I/O devices. To connect a new device , user simply connects the device interface board to the bus

Peripheral Component Interconnect (PCI)

- PCI bus supports three independent address spaces like memory, I/O and configuration.
- The I/O address space is intended for use with processors, like Pentium which has separate I/O address space.
- Configuration space is intended to give the PCI it's plug-and-play capability.
- A 4-bit command that accompanies the address identifies which of these three spaces is being used in a given data transfer operation.
- The following diagram **fig: PCI(1)** shows main memory of computer connected directly to the processor bus.
- An alternative arrangement that is used often with PCI bus is shown in **fig: PCI(2)**.
- PCI bridge provides a separate physical connection for the main memory.
- It is assumed that the master maintains the address information on the bus until data transfer is completed.
- But this is not needed becoz address is needed only long enough for the slave to be selected. slave can store the address in the internal buffer.

Peripheral Component Interconnect (PCI)

- 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 significant cost reduction because the number of wires on a bus is an important cost factor.
- This is the approach used in PCI bus.
- At any given time one device is the bus master. It has the right to initiate data transfers by issuing read and write commands.
- A master is called as **initiator** in PCI terminology which is either a processor or a DMA controller.
- The addressed device that responds to the command read and write is called as **target**.

Device configuration

- When an I/O device is connected to a computer several actions are needed to configure both the device and software that communicates with it.
- Once the device is connected, the software
  - needs to know the address of the device,
  - also need to know relevant device characteristics like speed of transmission link, whether parity bits are used and so on.

Peripheral Component Interconnect (PCI)

- PCI simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device.
- The configuration ROM’s of all devices are accessible in the configuration address space.
- PCI initialization software reads this ROM’s whenever the system is powered up or reset.
- In each case it finds whether the device is a printer, keyboard, an Ethernet interface or a disk controller.
- Devices are assigned addresses during the initialization process.

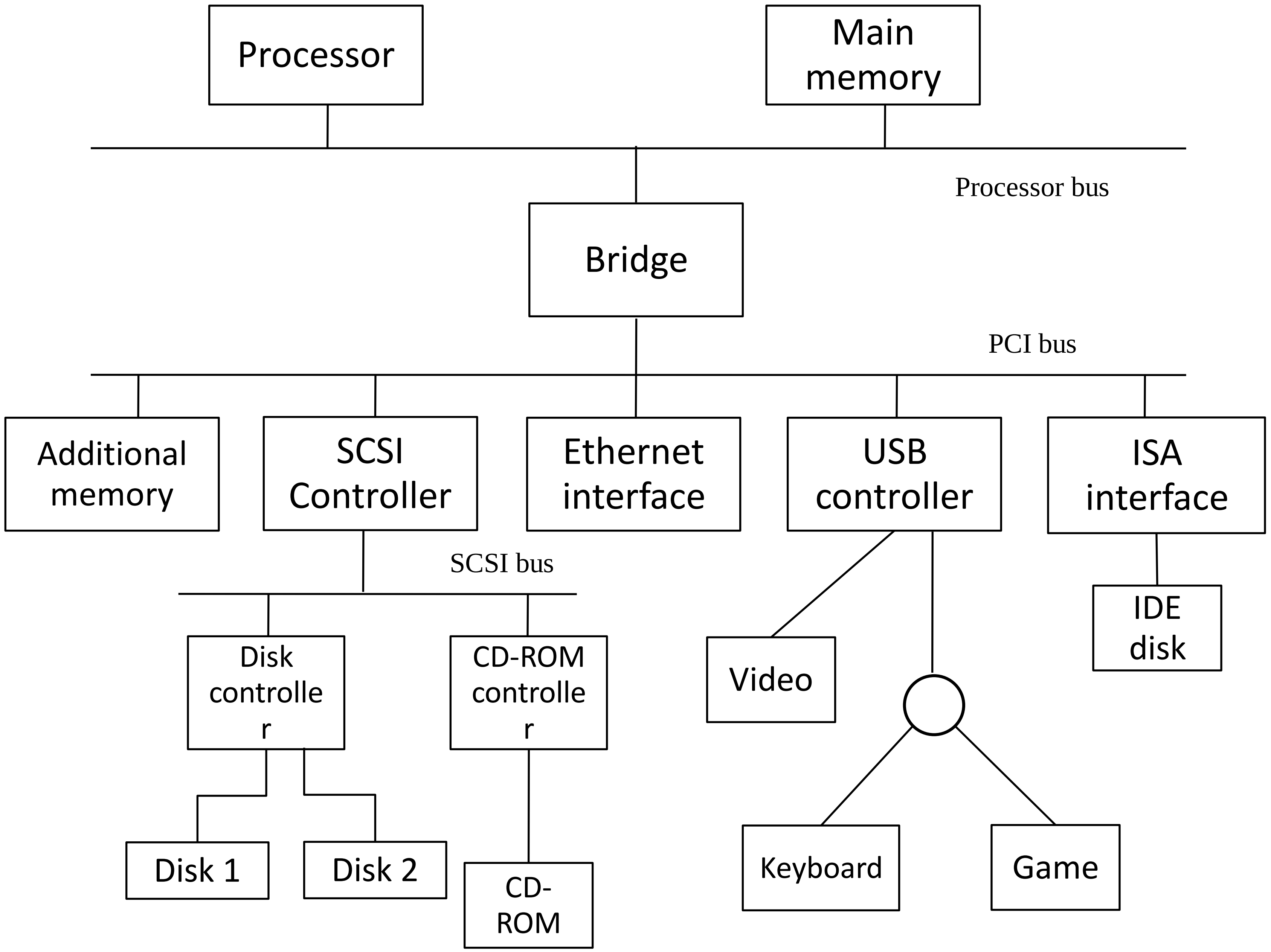


Fig: PCI(1) an example of a computer system using different interface standards

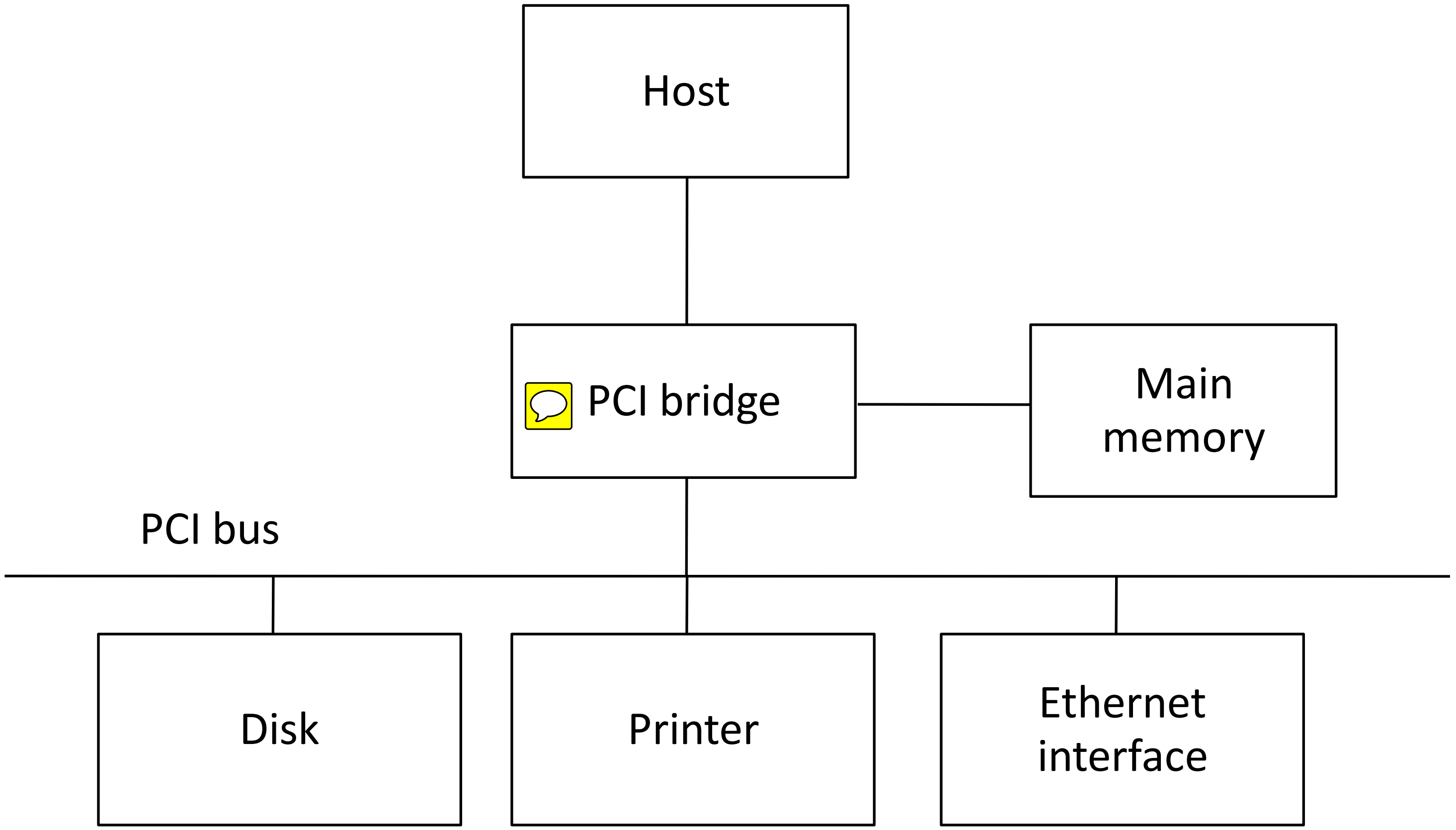


Fig: PCI(2) use of PCI bus in a computer system

Introduction to standard serial communication

RS-232

- Serial communication is basically the transmission or reception of data one bit at a time.
- Today's computers generally address data in bytes or some multiple thereof. A byte contains 8 bits. A bit is basically either a logical 1 or zero.
- DTE stands for Data Terminal Equipment, and DCE stands for Data Communications Equipment.
- These terms are used to indicate the pin-out for the connectors on a device and the direction of the signals on the pins.
- computer is a DTE device, while most other devices such as modem and other serial devices are usually DCE devices.
- RS-232 has been around as a standard for decades as an electrical interface between DTE and DCE.
- It appears under different forms like RS-232C, RS-232D, V.24, V.28 or V.10.
- RS-232 is used for asynchronous data transfer as well as synchronous links such as SDLC, HDLC, Frame Relay and X.25

RS-232

- A typical activity that might use a synchronous protocol would be a transmission of files from one point to another.
- As each transmission is received, a response is returned indicating success or the need to resend.
- The term asynchronous is usually used to describe communications in which data can be transmitted intermittently rather than in a steady stream.
- For example, a telephone conversation is asynchronous because both parties can talk whenever they like.
- If the communication were synchronous, each party would be required to wait a specified interval before speaking.
- *RS-232 (Recommended standard-232) is a standard interface approved by the Electronic Industries Association (EIA) for connecting serial devices.*
- *In other words, RS-232 is a long established standard that describes the physical interface and protocol for relatively low-speed serial data communication between computers and related devices.*

RS-232

- *RS-232 is the interface that your computer uses to talk to and exchange data with your modem and other serial devices.*
- *The serial ports on most computers use a subset of the RS-232C standard.*
- *RS-232 uses a single ended mode of operation.*
- *RS-232 supports a maximum cable length of 50feet and a maximum data rate of 20kbps.*
- *RS-232 supports a minimum driver output range of ±5V to ±15V and a maximum driver output range of ±25V.*

Limitations of RS-232

- *RS-232 has some serious shortcomings as an electrical interface.*
  - *The interface presupposes a common ground between the DTE and DCE.*
  - *A signal on a single line is impossible to screen effectively for noise. By screening the entire cable one can reduce the influence of outside noise, but internally generated noise remains a problem.*

USB

- Universal Serial Bus originally developed in 1995 by a consortium including Compaq, HP, Intel, Lucent, Microsoft, and Philips.
- USB is a communications architecture that gives a personal computer the ability to interconnect a variety of devices using a simple four wire cable.
- The USB is actually a two-wire serial communication link that runs at either 1.5 or 12 megabits per second.
- USB protocols can configure devices at startup or when they are plugged in at run time.
- These devices are broken into various device classes. Each device class defines the common behavior and protocols for devices that serve similar functions.
- Some examples of USB device classes are shown in the following table:

Device Class	**Example Device**
Display	Monitor
Communication	Modem
Audio	Speakers
Mass Storage	Hard Drive
Human Interface	Data glove

USB

- USB 1.1 supports both Low-speed devices and Full-speed devices whose data transfer rates are 1.5Mbps and 12Mbps respectively.
- USB 2.0 supports High-speed devices with data transfer rate up to 480Mbps.

Motivation for USB

- To avoid device-specific interfaces that eliminates large number of interfaces like PS/2, serial, parallel, monitor, microphone, keyboard, and so on.
- To avoid installation and configuration problems and to allow hot attachment of devices.

Advantages of USB

- Power distribution is simple because simple devices can be bus-powered. E.g. keyboard, floppy disk drives, wireless LAN's etc..
- Possible to control peripherals because USB allows data to flow in both directions.
- Also it maintains power conservations because it enters into suspended state if there is no activity for 3ms.
- Supports CRC for error detection and recovery.

IEEE 1394

- Apple originally developed this standard for high-speed peripherals.
- FireWire is Apple's name for the IEEE 1394 High Speed Serial Bus.
- IEEE 1394 is a serial bus architecture for high-speed data transfer.
- FireWire is a serial bus, meaning that information is transferred one bit at a time.
- Parallel buses utilize a number of different physical connections, and as such are usually much less efficient, more costly, and typically heavier .
- FireWire fully supports both isochronous (A sequence of events or if the events occur regularly, or at equal time intervals) and asynchronous applications.
- Apple intended FireWire to be a serial replacement for the parallel SCSI bus while providing connectivity for digital audio and video equipment.
- As of 2007, IEEE 1394 is a composite of four documents: the original IEEE Std. 1394-1995, the IEEE Std. 1394a-2000 amendment, the IEEE Std. 1394b-2002 amendment, and the IEEE Std. 1394c-2006 amendment.

IEEE 1394

- FireWire can connect up to 63 peripherals in a tree or daisy-chain topology.
- It allows peer-to-peer device communications such as communication between a scanner and a printer to take place without using system memory or the CPU.
- FireWire also supports multiple hosts per bus. It is designed to support plug and play and hot swapping.
- The copper cable it uses in its most common implementation can be up to 4.5 meters (15 ft) long and is more flexible than most parallel SCSI cables.
- FireWire devices implement the ISO/IEC 13213 "configuration ROM" model for device configuration and identification, to provide plug-and-play capability.
- FireWire devices are organized at the bus in a tree topology. Each device has a unique self-id. One of the nodes is elected root node and always has the highest id.