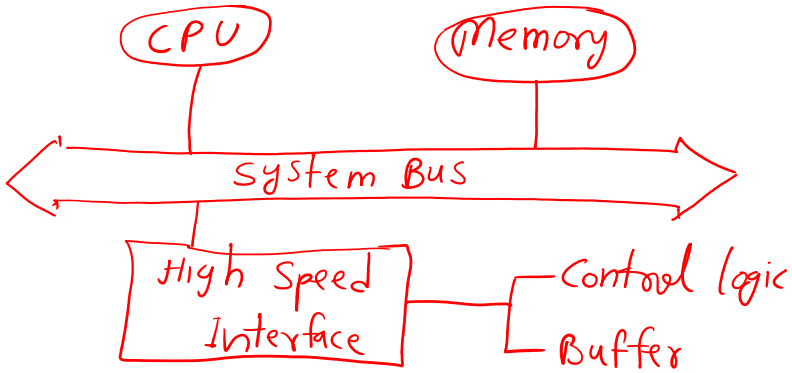
Unit V:
Input/Output Organization

Input/Output Organization

I/O

- Provides an efficient mode of communication between the central system and the outside environment.
- Input or output devices attached to the computer are also called *peripherals*.
- Examples of peripherals are keyboards, display units, and printers.
- Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.



Input/Output Organization

Input-Output Interface

- Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.
- The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral.
- The major differences are:
  1. 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.
  2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.

Input/Output Organization

Input-Output Interface

3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. 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 the CPU.

- These differences are resolved by interface units.
- Interface units are special hardware components that lie between the CPU and peripherals to supervise and synchronize all input and output transfers.

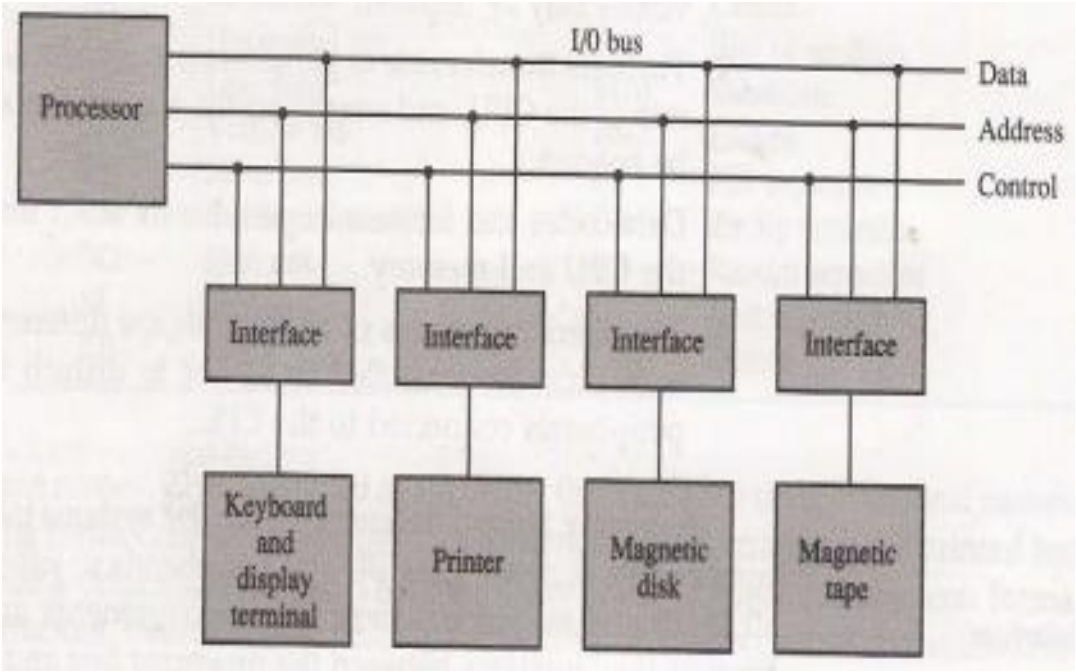


Figure: Connection of I/O bus to I/O devices

Input/Output Organization

Input-Output Interface

- Four types of commands an interface may receive:

i. **Control command** - issued to activate the peripheral and to inform it what to do.
   - For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction.

ii. **Status command** - used to test various status conditions in the interface and the peripheral.
   - For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

Input/Output Organization

Input-Output Interface

- Four types of commands an interface may receive:

iii. **Data output command** - causes the interface to transfer data from the bus into one of its registers.

- Interface transfers the information from the data lines in the bus to its buffer register. It then communicates with the tape controller and sends the data to be stored on tape.

iv. **Data input command** – it is opposite of the data output.

- Interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

Input/Output Organization

I/O versus Memory Bus

- There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.

Input/Output Organization

I/O versus Memory Bus

- **IOP (Input-Output Processor):**
- The computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O.
- The memory communicates with both the CPU and the IOP through a memory bus.
- The IOP communicates with the input and output devices through a separate I/O bus with its own address, data and control lines.
- The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory.

Input/Output Organization

I/O versus Memory Bus

- **Isolated IO:**
- The CPU has distinct input and output instructions for I/O and memory.
- When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.
- When the CPU fetches an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface.

Input/Output Organization

I/O versus Memory Bus

- **Memory Mapped IO:**
- The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space.
- The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as **memory-mapped I/O**.

Input/Output Organization

Modes of Transfer

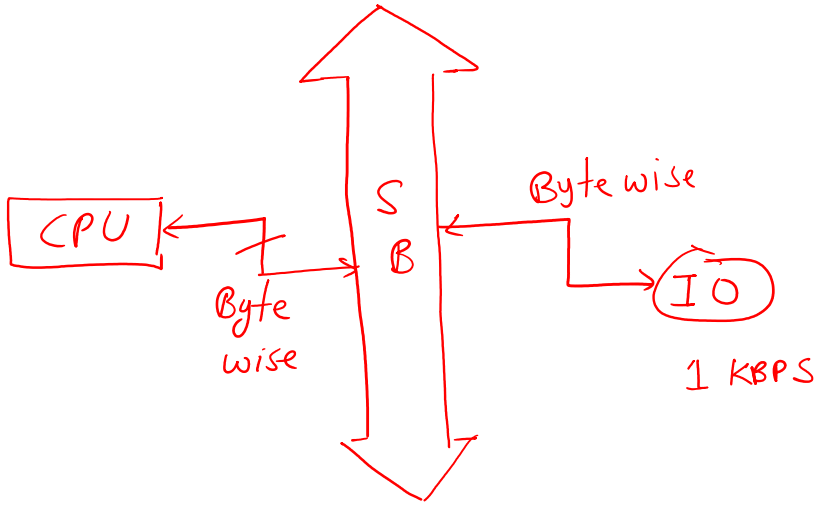
- Data transfer to and from peripherals may be handled in one of three possible modes:
  i. Programmed I/O
  ii. Interrupt-initiated I/O
  iii. Direct memory access (DMA)

Input/Output Organization

Modes of Transfer

i. Programmed I/O:

- In this mode, high speed interface is not present i.e., the CPU is directly connected with the IO devices. Therefore, processor utilization is inefficient.
- Processor undergoes waiting until completion of the I/O operation. This waiting depends on the speed of the I/O device.



1 KB — 1 sec

1 Byte = $\frac{1}{1K}$ sec

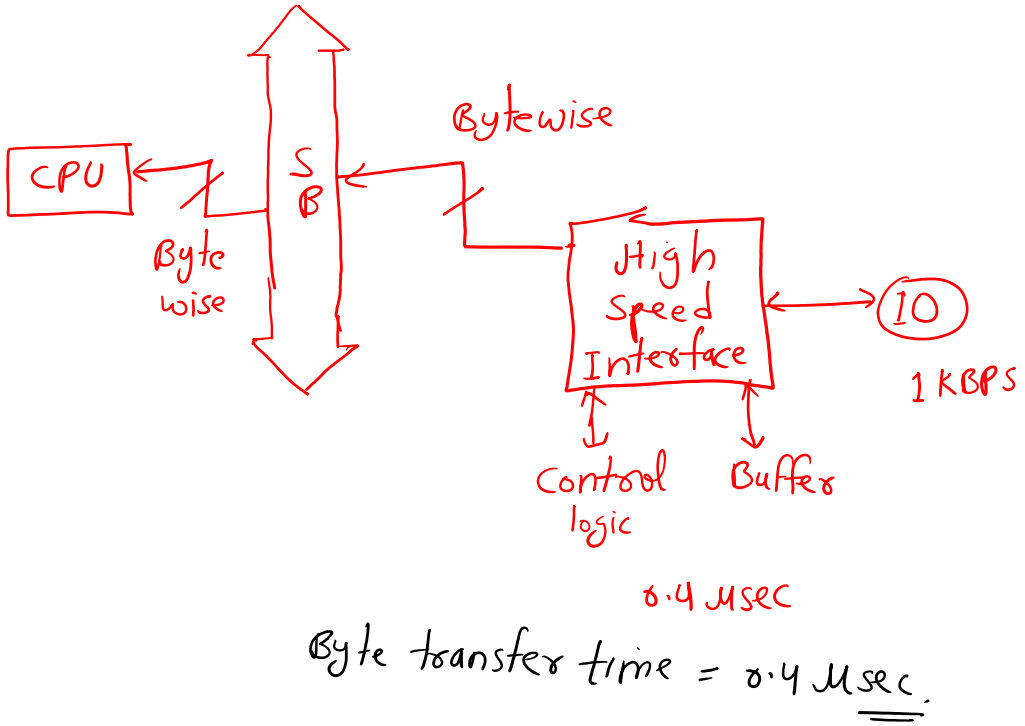
= 1 millisec

Input/Output Organization

Modes of Transfer

ii. Interrupt Driven I/O:

- In this mode, high speed interface is used to connect the basic I/O devices.
- Processor utilization is efficient because processor is only communicating with the high speed interface.
- The CPU waiting time depends on the latency of the interface.

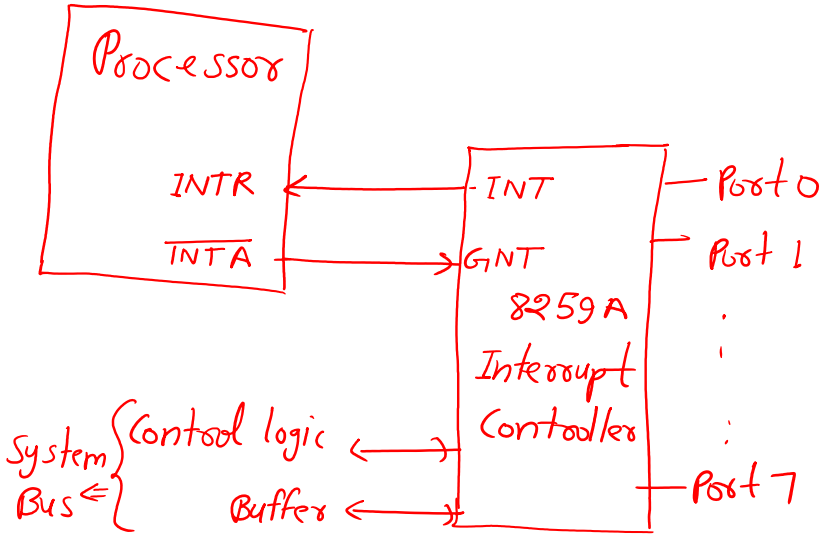


Input/Output Organization

Modes of Transfer

ii. Interrupt Driven I/O:

- In this mode, high speed interface is used to connect the basic I/O devices.
- Processor utilization is efficient because processor is only communicating with the high speed interface.
- The CPU waiting time depends on the latency of the interface.

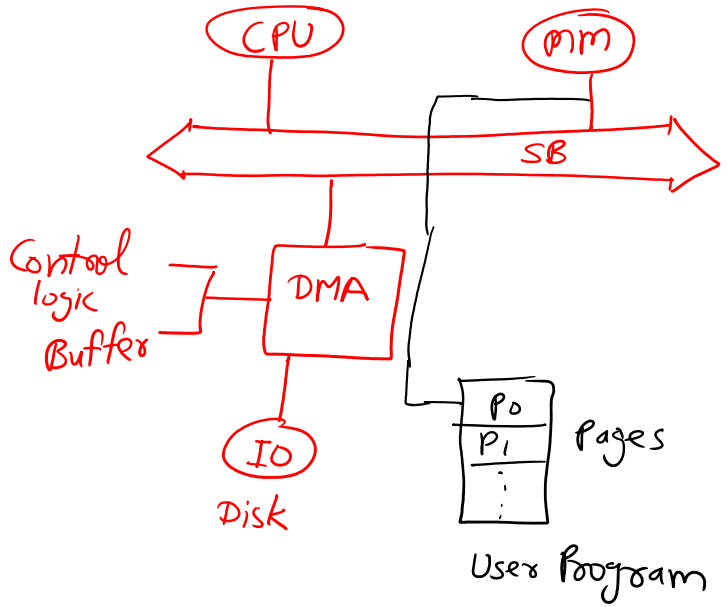


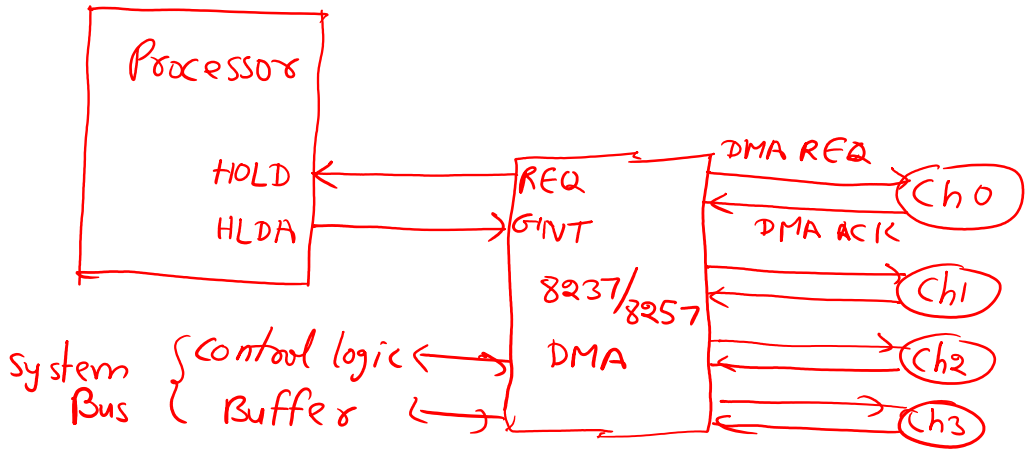
Input/Output Organization

Modes of Transfer

iii. DMA (Direct Memory Access):

- In this mode, the bulk amount of data is transferred from the I/O device to main memory without involvement of the CPU.





Input/Output Organization

Modes of Transfer

iii. DMA (Direct Memory Access):

- The CPU initializes the DMA along with the IO address, memory address, control signals and count value. Later the CPU gets busy with other tasks.
- DMA control logic interprets the request and enables the corresponding port for the respective operation.
- Based on the speed of the disk, it prepares the data. Later, it transfers the data to buffer.
- When the buffer contain the data, the DMA enables the HOLD signal to gain the control of the bus and waits for the HLDA signal.
- After receiving HLDA signal, the DMA transfers the data from IO to Main Memory (MM) until count becomes zero.

Input/Output Organization

Modes of Transfer

iii. DMA (Direct Memory Access):

- After the operation, the DMA re-establish the bus connection to CPU.
- In the DMA operation, CPU is present in two states: Busy state and Blocked state (HOLD state).
- Until preparing the data, the CPU is busy and until transferring the data, the CPU is blocked.

% CPU is busy = $\left(\frac{x}{x+y}\right) * 100$

% CPU is blocked = $\left(\frac{y}{x+y}\right) * 100$

depends on disk speed

x : data preparation time

y : " transfer time

depends on mm latency.

Input/Output Organization

Modes of Transfer

iii. DMA (Direct Memory Access):

- DMA operates in 3 different modes: Burst, Cycle Stealing, and Block Mode.
- In the Burst mode of DMA, after receiving the HLDA signal the bulk amount of data is transferred from IO device to MM.
- In the Cycle Stealing mode of DMA, before receiving the HLDA signal it forcefully suspend the CPU operation and gain the control of the bus and transfers the very important data from IO to MM.
- In the Block mode of DMA, after receiving the HLDA signal, the data is transferred from IO to MM in the block wise manner.

Priority Interrupt [1]

- Data transfer between CPU and I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal.

- The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then program branches to the service routine that processes the required transfer.

- Number of I/O devices are attached to the computer, with each device being able to originate an interrupt request.

- **Interrupt system has to find the source of interrupt and has to decide which device to serve first in case several devices raised an interrupt simultaneously.**

Priority Interrupt [2]

- Priority interrupt is a system that establishes a priority over various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously.

- The system also determines which conditions are permitted to interrupt the computer while another interrupt is being serviced.

- Establishing the priority of simultaneous interrupts can be done by software or hardware

  - **Polling**: software means used to identify the highest priority source.
  - **Daisy Chaining**: Hardware approach

Polling

- In this method there is a one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt.

- Highest priority source is tested first, if its interrupt signal is on, control branches to a service routine for this source. Otherwise the next lower priority source is tested and so on.

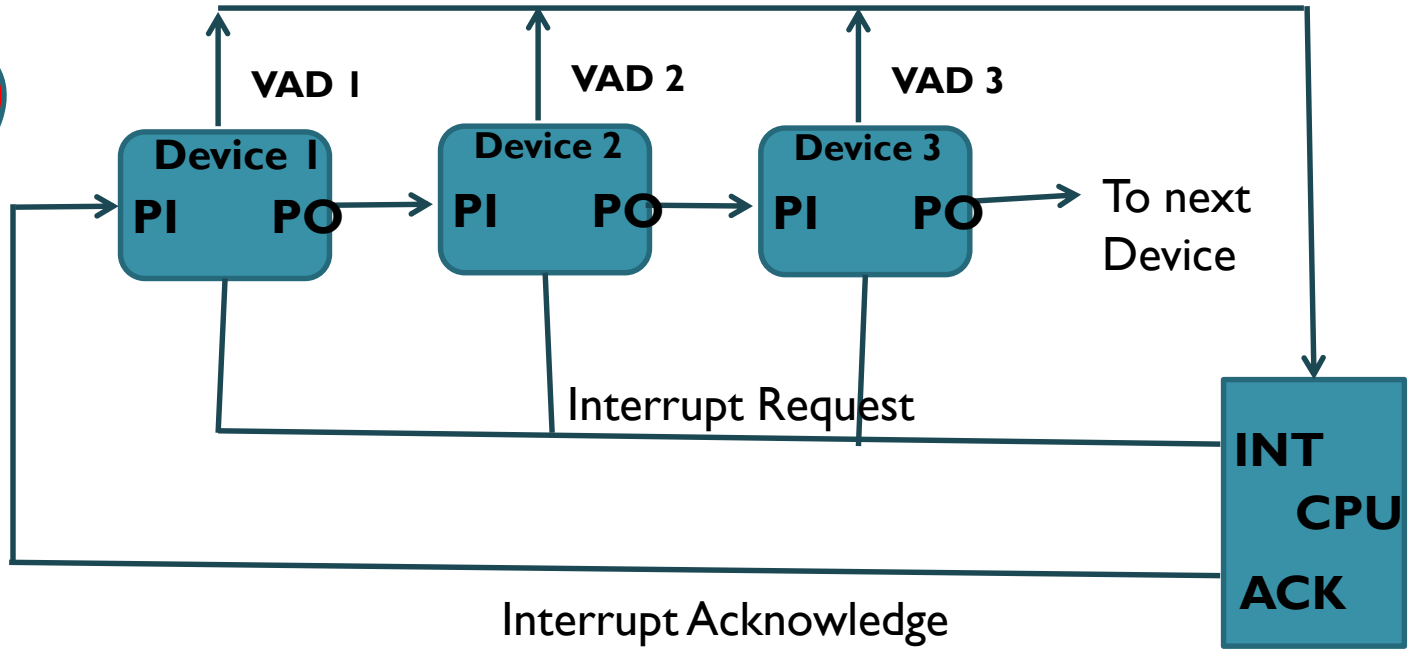
- The disadvantage of software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device.

Daisy-Chain priority Interrupt [1]

- Consists of serial connection of all devices that request the interrupt.
- Device with highest priority is placed in the first position, followed by lower priority devices up to the device with the lowest priority, which is placed last in the chain.
- The interrupt request line is common to all devices and forms a wired logic connection.
- If any device has its interrupt signal in the low-level state, the interrupt line goes to low level state and enables the interrupt input in the CPU.
- When no interrupts are pending the interrupt line stays in the high level state and no interrupts are recognized by the CPU.

Daisy-Chain priority Interrupt [2]

Farther the device from first position, the lower is its priority



Device with PI=1 and PO=0 is the one with the highest priority that is requesting an interrupt. And this device places its VAD on the data bus

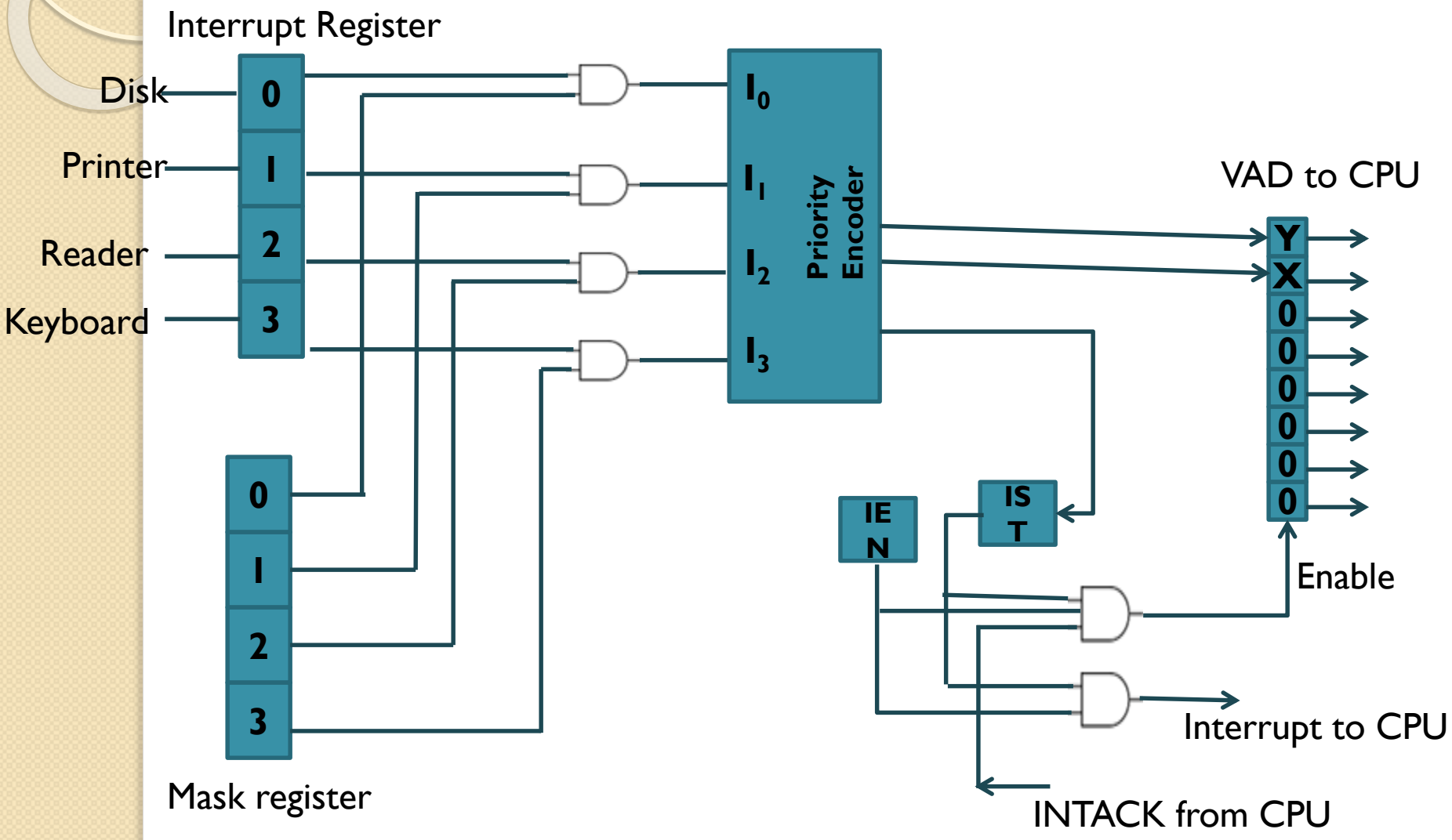
Daisy-Chain priority Interrupt [3]

- CPU responds to an interrupt request by enabling interrupt acknowledge line. This signal is received by device 1 at its PI (priority IN) input. The acknowledge signal passes on to the next device through PO (priority out) output only if device 1 is not requesting an interrupt.
- If device 1 has pending interrupt, it blocks the acknowledge signal from the next device by placing 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during interrupt cycle
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output.
- If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.

Parallel Priority Interrupt [1]

- 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.
- Includes a 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 a high priority device to interrupt the CPU while a lower priority device is being serviced.

Parallel Priority Interrupt [2]



Parallel Priority Interrupt [3]

- BY means of program instructions, it is possible to set or reset any bit in the mask register.
- Each interrupt bit and its corresponding Mask bit are applied to an AND gate to produce the four inputs to the priority encoder .
- An interrupt is recognized only if its corresponding mask bit is set to 1 by the program.
- Priority encoder generates two bits of the vector address, which is transferred to the CPU.

Parallel Priority Interrupt [4]

- Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs.
- The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system.
- The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU.
- The interrupt acknowledge INTACK signal from the CPU enables the bus buffers in the output register and the vector address VAD is placed into the bus.

Priority Encoder

Inputs				Outputs		
I_0	I_1	I_2	I_3	X	Y	IST
1	×	×	×	0	0	1
0	1	×	×	0	1	1
0	0	1	×	1	0	1
0	0	0	1	1	1	1
0	0	0	0	×	×	0

$X = I_0' I_1'$

$Y = I_0' I_1 + I_0' I_2'$

$IST = I_0 + I_1 + I_2 + I_3$

Input with low index is having higher priority