



CHAPTER-8

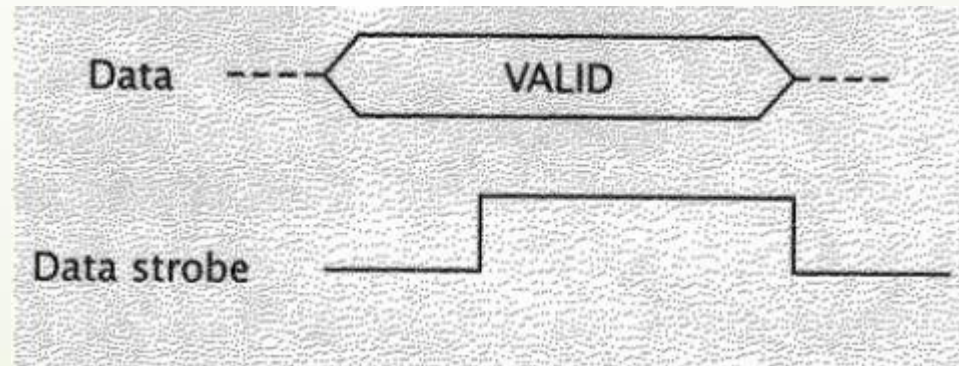


Synchronous and asynchronous data transfer

- Synchronous data transfer occurs when peripherals are located within the same computer as the CPU because their close proximity allows them to share a common clock.
=> data does not have to travel far physically.
- Asynchronous data transfer do not require that the source and destination use the same system clock.
=> Asynchronous transfers use control signals and their associated hardware to coordinate the movement of data.
- Four types of asynchronous data transfer:
 1. source-initiated data transfer
 2. destination-initiated data transfer
 3. source-initiated data transfer with handshaking
 4. destination-initiated data transfer with handshaking

Source-initiated data transfer

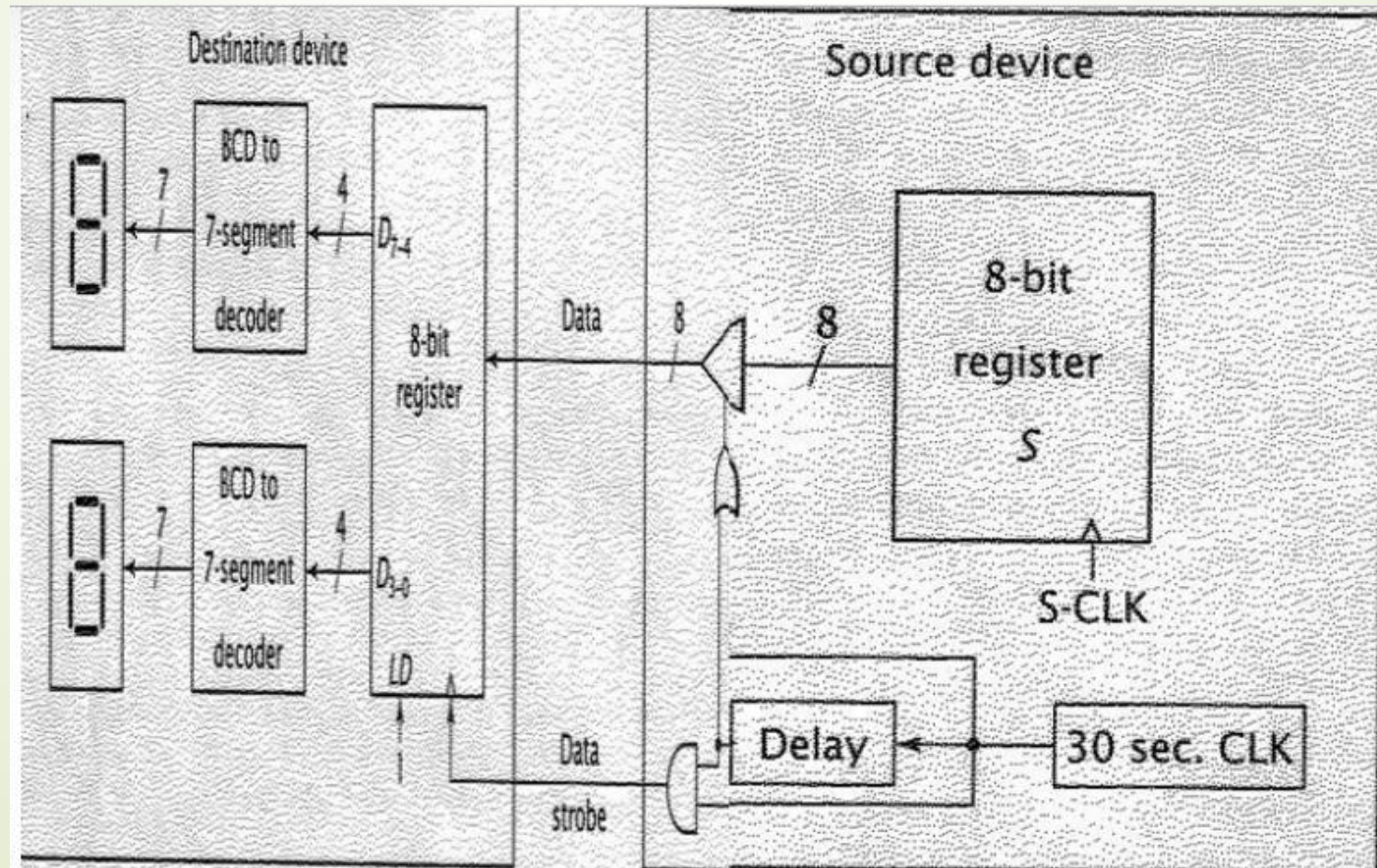
- Source outputs its data, then strobes a control signal for set amount of time; destination device reads in the data during this time.
- The source device next deasserts the strobe and stops outputting data.
- Destination device never sends any information back to the source, so the source cannot know for sure whether the data was received.



Contd...

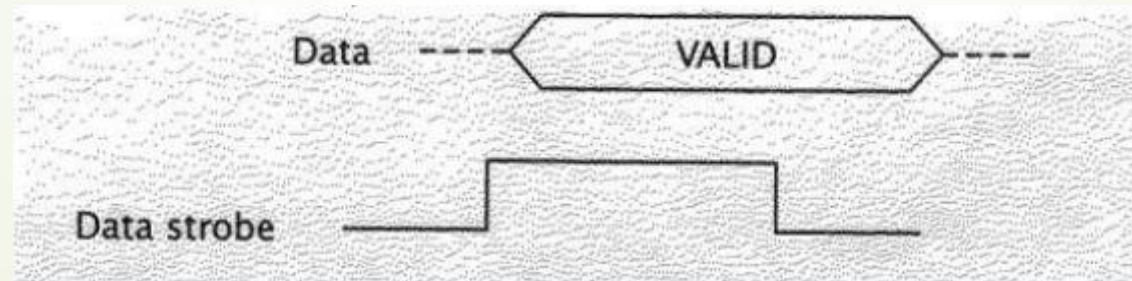
=> A clock with a period of 30s enables the tristate buffer, which causes valid data to be made available to the output module.

=> After a set delay, to account for the propagation delay of tristate buffer and to allow data to be stable, the data strobe signal is set high.



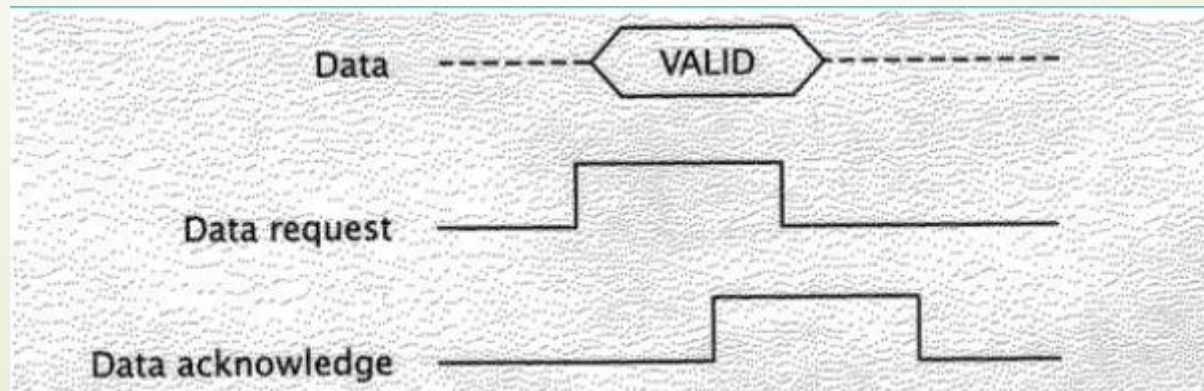
Destination-initiated data transfer

- The destination device transmits 0 data strobe signal to the source, which after a brief delay, makes data available.
- The destination device reads in this data and deasserts the data strobe.
- This in turn causes the source to stop transmitting valid data.



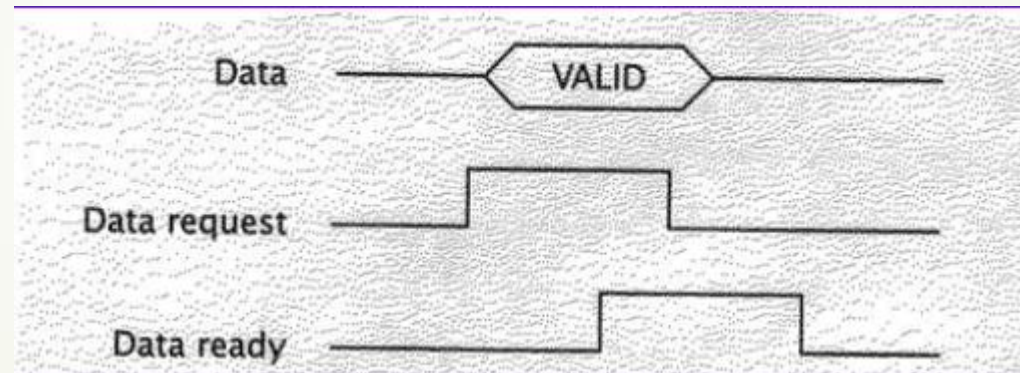
Source-initiated data transfer with handshaking

- The source sets the data request signal high and then makes valid data available to the destination device.
- After the requisite delay to allow the data to stabilize, the destination device reads in the data.
- Once the destination device has read the data, it sends a data acknowledge signal to the source.
- This tells the source that the destination has read in and no longer needs this data. The source sets its data request line low and stops sending data. The destination then resets its data acknowledge signal.



Destination-initiated data transfer with handshaking

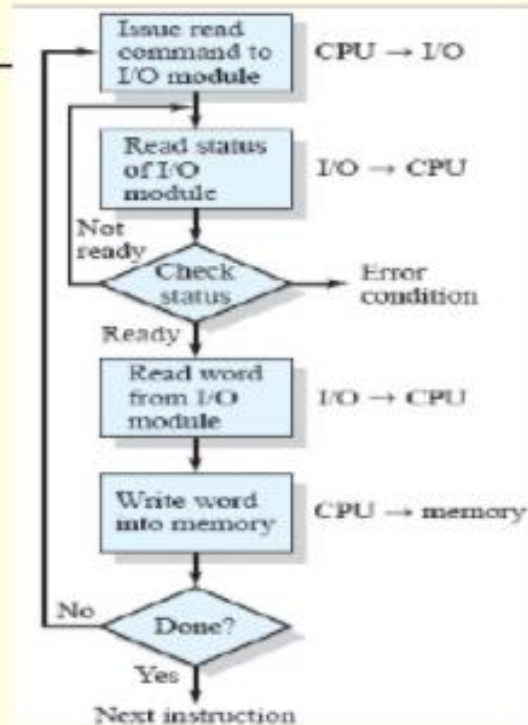
- Similar to source-initiated data transfer with handshaking, except that the data-acknowledge signal is replaced by a data-ready signal.



Programmed I/O

Programmed I/O

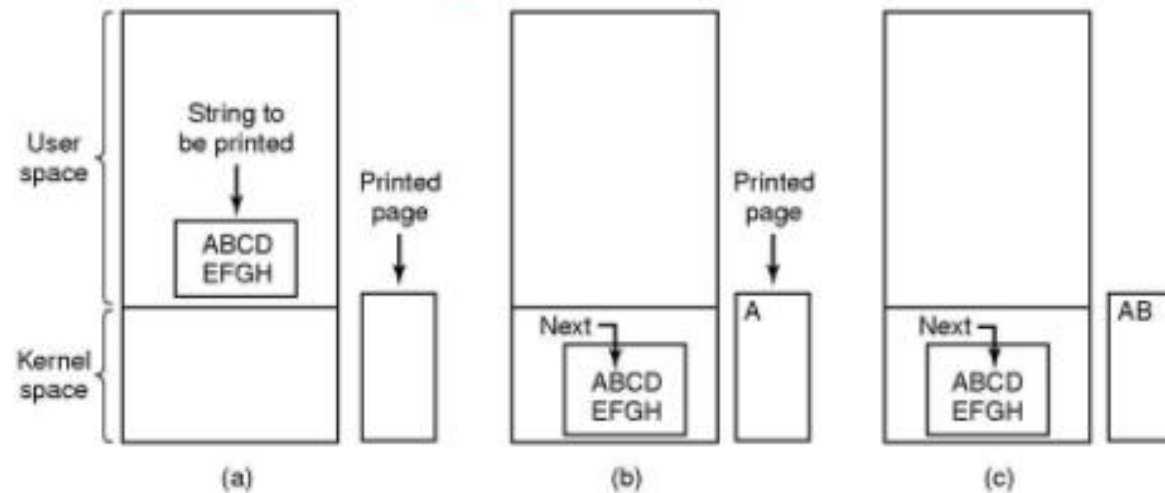
- CPU while executing a program encounters an I/O instruction
- CPU issues I/O command to I/O module
- I/O module performs the requested action & set status registers
- CPU is responsible to check status registers periodically to see if I/O operation is complete. **SO**
- No Interrupt to alert the processor



Isolated I/O uses separate instructions to access I/O ports. Memory mapped I/O treats I/O ports as memory locations. From CPU's view, all memory locations are the same but certain addresses are located for I/O.

Contd...

Programmed I/O




- Steps in printing a string
 - String in the user buffer
 - A System call to transfer the string to the kernel.
 - String printed



Interrupts



- I/O devices are slower than memory and the amount of time they require will vary.
- The uncertainty of when the device will be ready complicates the task of accessing I/O devices.
- Interrupt is a mechanism for alleviating the delay caused by this uncertainty and for maximizing system performance.
- Interrupt is a signal which causes the normal operation of the system to halt or pause.

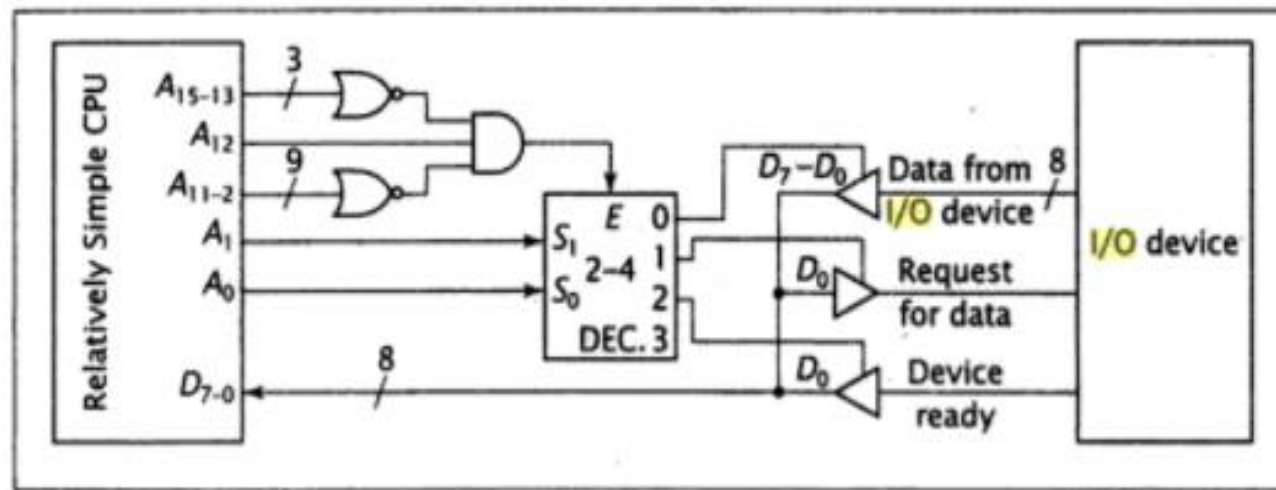


Transferring data between CPU and I/O devices

- Method to alleviate the problem of I/O devices with variable delays is called polling.
- In polling, the CPU sends a request to transfer data to an I/O device.
- I/O devices processes the request and sets a device-ready signal when it is ready to transfer.
- The CPU reads signal; if it is high, it performs the data transfer. If not, it loops back, continually reading and testing the value of the device ready signal.
- A slow device causes the CPU to remain in the polling loop for quite a long time
- Interrupts were developed to make use of this wasted time.
- When interrupts were used with I/O devices, CPU may output a request to I/O device & instead of entering into waiting state, CPU continues executing instructions(useful work).
- When device is ready to transfer data, it sends interrupt request signal to the CPU.
- The CPU then acknowledges the interrupt and completes the data transfer.
- Unlike polling or wait states, interrupt do not waste time waiting for the I/O device to become ready.

Contd..

Hardware to implement an I/O port that uses polling





Types of interrupt:

1. External Interrupt:

- Used by CPU to interact with I/O device.
- This improves system performance by allowing the CPU to execute instructions, instead of just waiting for the I/O device, while still performing the required data transfers.

2. Internal Interrupt:

- Occurs entirely within CPU, I/O devices play no role in these interrupts.
- For example, a timer built into the CPU may generate an interrupt at a pre-determined interval.

3. Software Interrupt:

- Generated by the program itself.
- It itself is like a subroutine. Eg: HLT
- When we give HLT, it calls a predefined subroutine for it, which stops the program we are using.

INTERRUPT PROCESSING

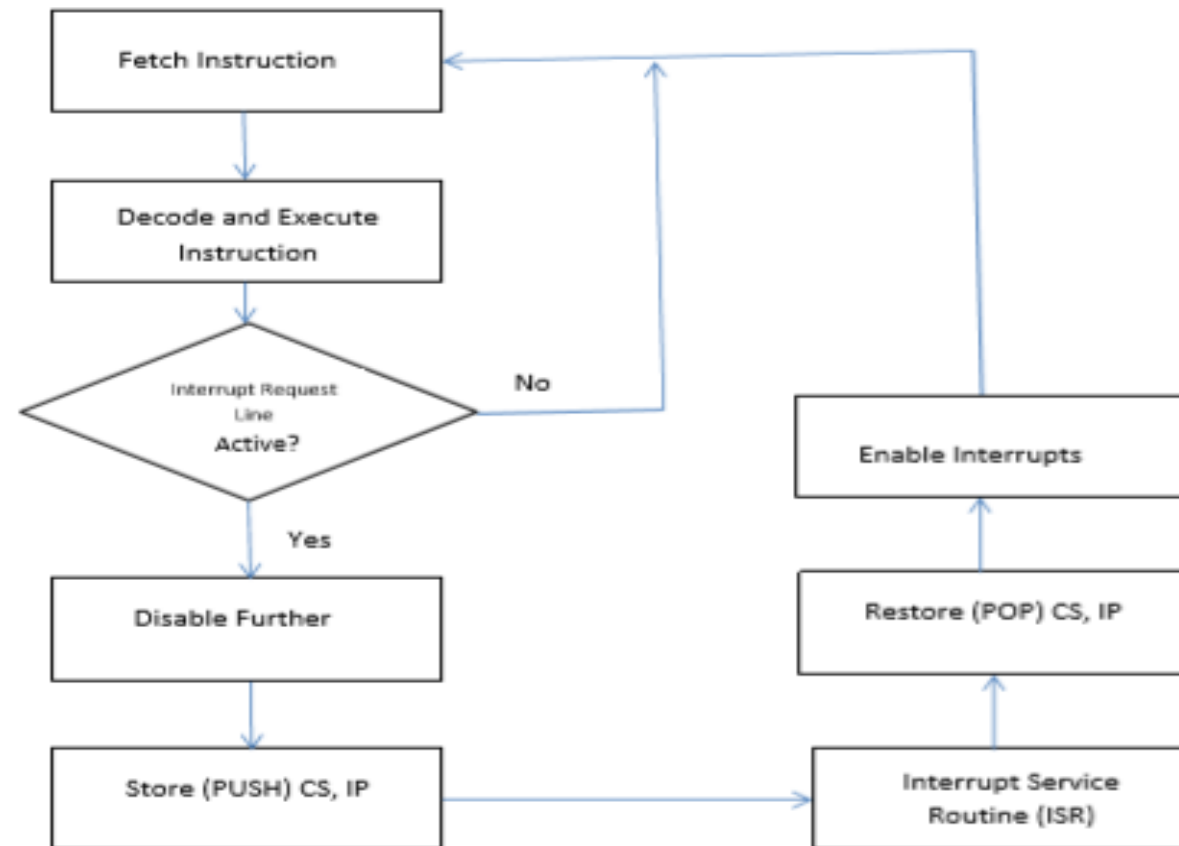
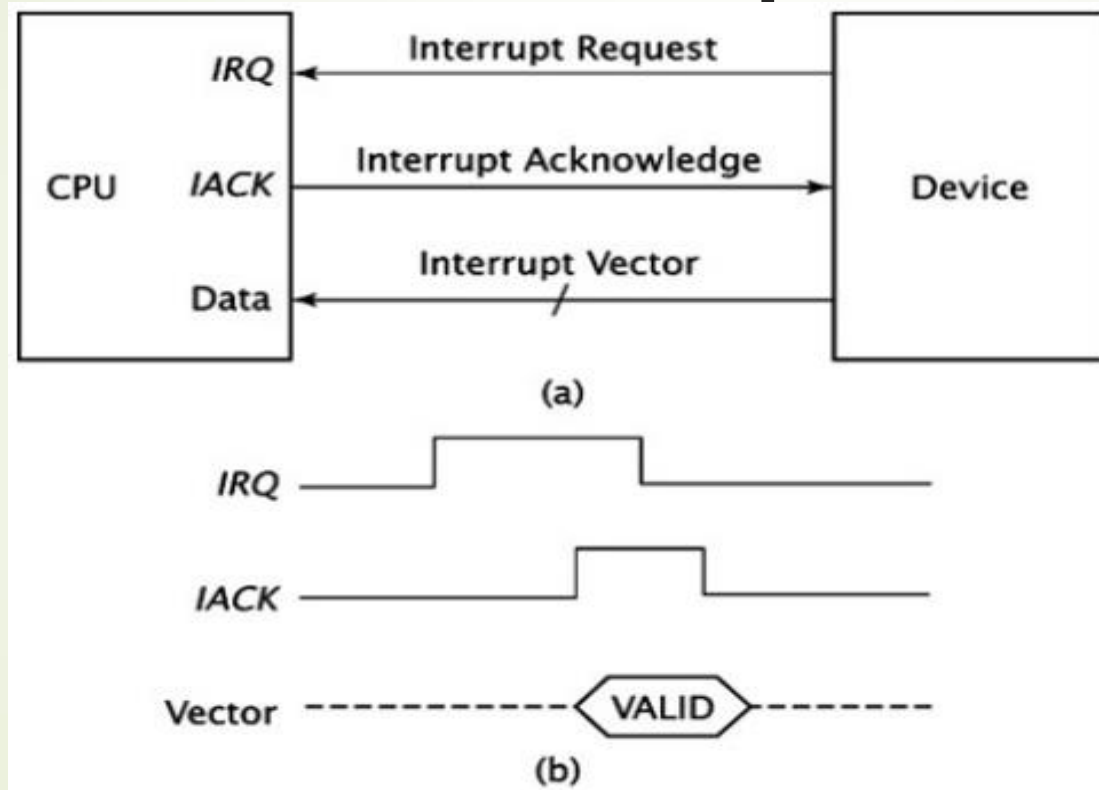


Figure: Interrupt Processing

- The processor checks for interrupts. If interrupt has occurred, processor will complete the instruction currently being executed.
- The processor will disable the further interrupts.
- The processor stores the current state of program by PUSH operation i.e. the value of flag register and CS: IP will be stored into stack by PUSH operation.
- The processor will load the address of the ISR and execute the ISR. At the end of ISR, instruction IRET is used which makes the processor return from the ISR to the original program.
- After the execution of ISR, the processor restores the previous state of program i.e. it will restore the value of flag register and CS: IP from stack by POP operation.
- The processor enables the interrupts and then starts program execution from where it has been interrupted.

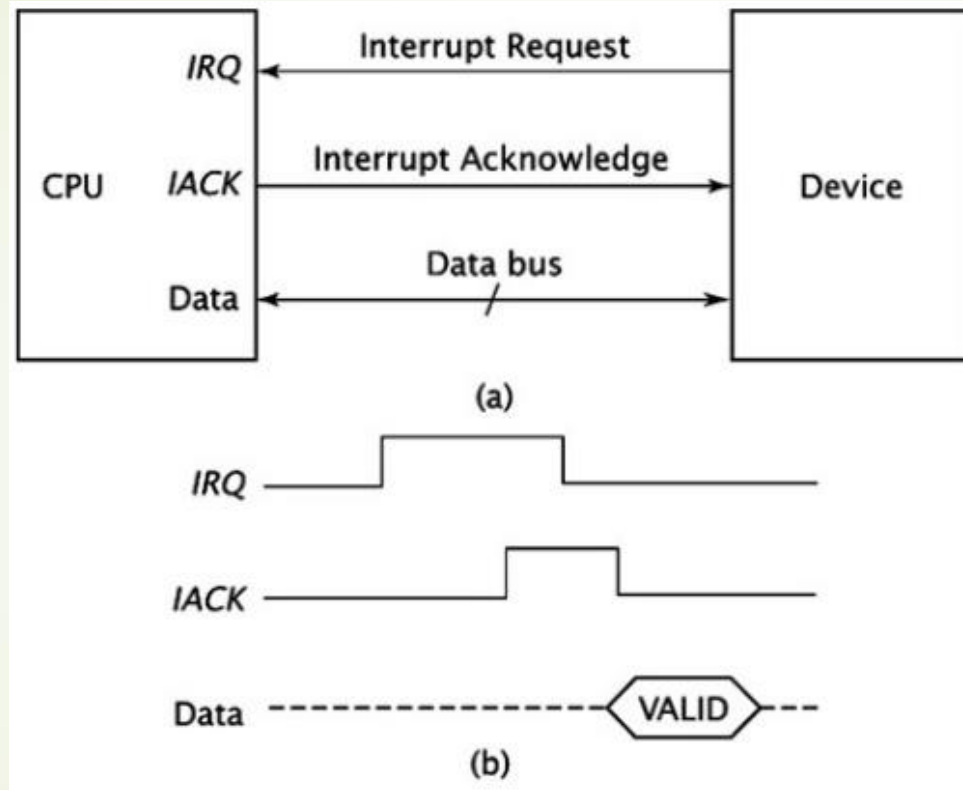
Interrupt hardware and priority

Vectored Interrupt



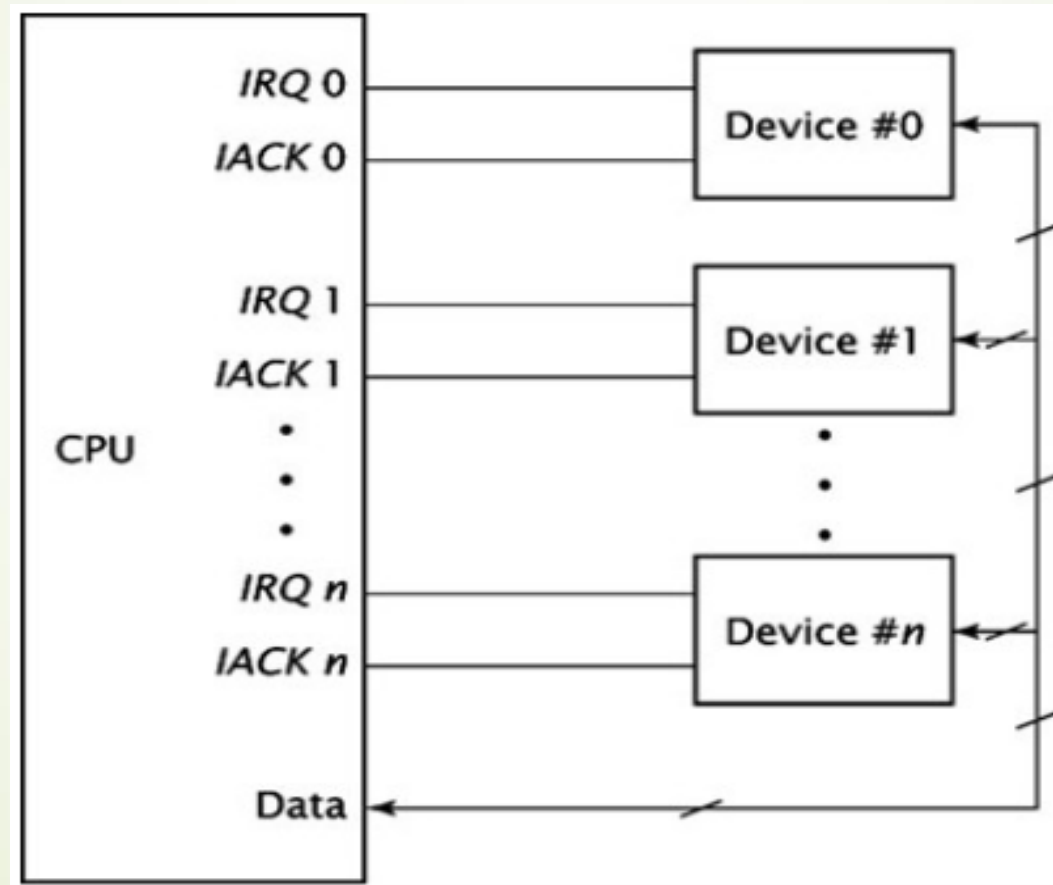
- CPU knows the address of ISR in advance.
- All it needs is that interrupting device sends its unique vector via a data bus and through its I/O interface to the CPU.
- CPU takes this vector, checks an interrupt table in memory and then carries correct ISR for the device.

Non-vectored interrupt



- Interrupting device never sends an interrupt vector.
- CPU receives the interrupt, and it jumps the program counter to a fixed address in hardware.
- CPU crucially does not know which device caused the interrupt without polling each I/O interface in a loop and checking status register of each I/O interface.

Prioritizing multiple interrupts: **Extension of non-vectored Interrupt**





Contd...

- IRQ_n has the highest priority and IRQ_0 has the lowest.
- If more than one device requests an interrupt, the CPU acknowledges and serves the interrupt with the highest priority first.
- This method works well when there are only a few IRQ/ACK pairs.
- As the number of interrupt increases, the number of pins needed by the CPU to accommodate these signals become prohibitive.

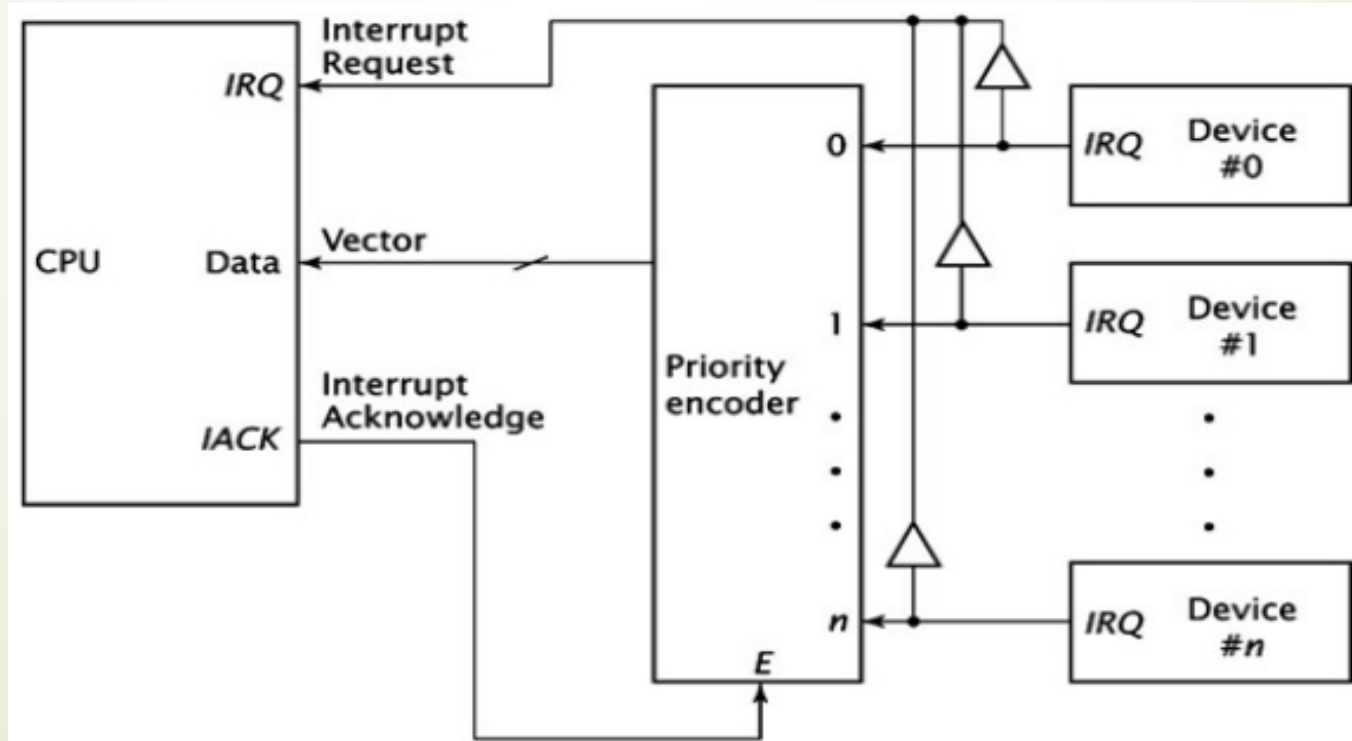


Contd...

- The interrupt request signals from the devices are ORed together.
- When the CPU receives an active IRQ input, it doesn't know which device generated the interrupt request.
- It sends out an acknowledgement signal and leaves it to the devices to work among themselves.
- Device #n receives the IACK signal directly from the CPU.
- If IACK is asserted, usually 1, this device has requested an interrupt; it sets its $IACK_{out}$ signal to 0 and places its vector on the data bus.
- If this device did not request the interrupt, it sets its $IACK_{out}$ signal to 1, thus passing it to device n-1 and so on until the IACK is asserted
- Straightforward and easy to implemented.
- The configuration is sequential, and thus can introduce hardware delays, if the chain is too long.

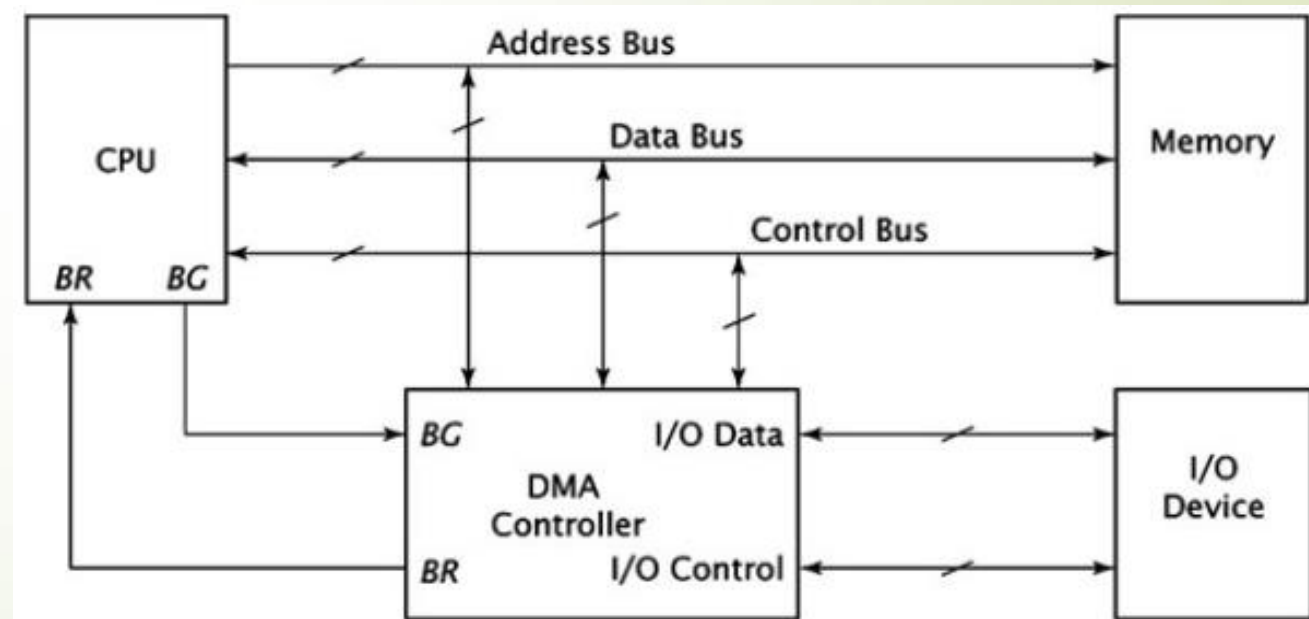
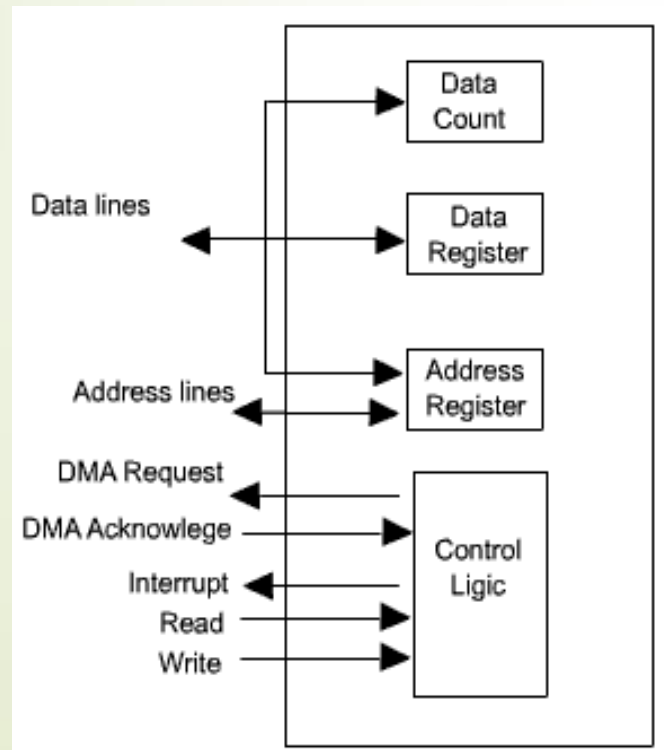
Implementing priority interrupts in parallel

- Implemented using priority encoder.
- Output of this encoder is the value of the highest priority device requesting an interrupt.
- Priority of the device doesn't determine the time needed to acknowledge the interrupt.



Direct Memory Access(DMA)

- Involves an additional module on the system bus capable of mimicking the processor and taking control of system from processor to transfer data to and fro from memory over the bus.
- DMA module must use bus only when processor does not need it, or must force the processor to suspend operation temporarily.





Incorporating DMA into a computer system

- A DMA controller implements direct memory access in the computer.
- Connects directly to I/O devices to one end and to the system bus at other end.
- Also interacts with CPU.
- To transfer data from an I/O device to memory, DMA controller first sends a bus request to the CPU by setting BR to 1.
- When CPU is ready to grant the request, CPU sets its bus grant signal, BG to 1.
- DMA controller, now, has control of the system bus.
- To load data from I/O device into memory, it asserts the appropriate I/O control signals and load data from I/O device into its internal DMA register.
- DMA controller also asserts appropriate signals on the system's control bus to cause memory to read the data.
- Once this is done, DMA controller no longer needs to use the System buses.
- It relinquishes its request by setting BR to 0.
- CPU, then sets BG to 0; resumes its normal operation.



DMA Transfer modes:

- **Burst mode/Block Transfer mode:**

- => Entire block of data is transferred in one contiguous sequence.

- => Once the DMA controller is granted access to the system buses by the CPU, it transfers all bytes of data in the data block before relinquishing control of system bus back to CPU.

- => Useful for loading programs or data files into memory but it does render the CPU inactive for relatively long periods of time.

- **Cycle Stealing Mode:**

- => CPU is not disabled for longer period of time like that in burst mode.

- => DMA controller obtains access to the system bus as in burst mode. However, it transfers one byte of data and then deasserts BR, returning control of the system buses back to CPU.

- => The process goes on until the entire block of data is transferred.

- => Data block is not transferred as quickly as in burst mode, but the CPU is not idled for as long as in burst mode.



Contd...

- ▶ **Transparent mode:**

- =>DMA controller only transfers data when the CPU is performing operations that do not make use of system bus.

- =>Primary advantage is that CPU never stops executing its programs.

- =>The hardware needed to determine when the CPU is not using the system buses can be quite complex and relatively expensive.

- =>This mode is generally not used.

I/O processors:

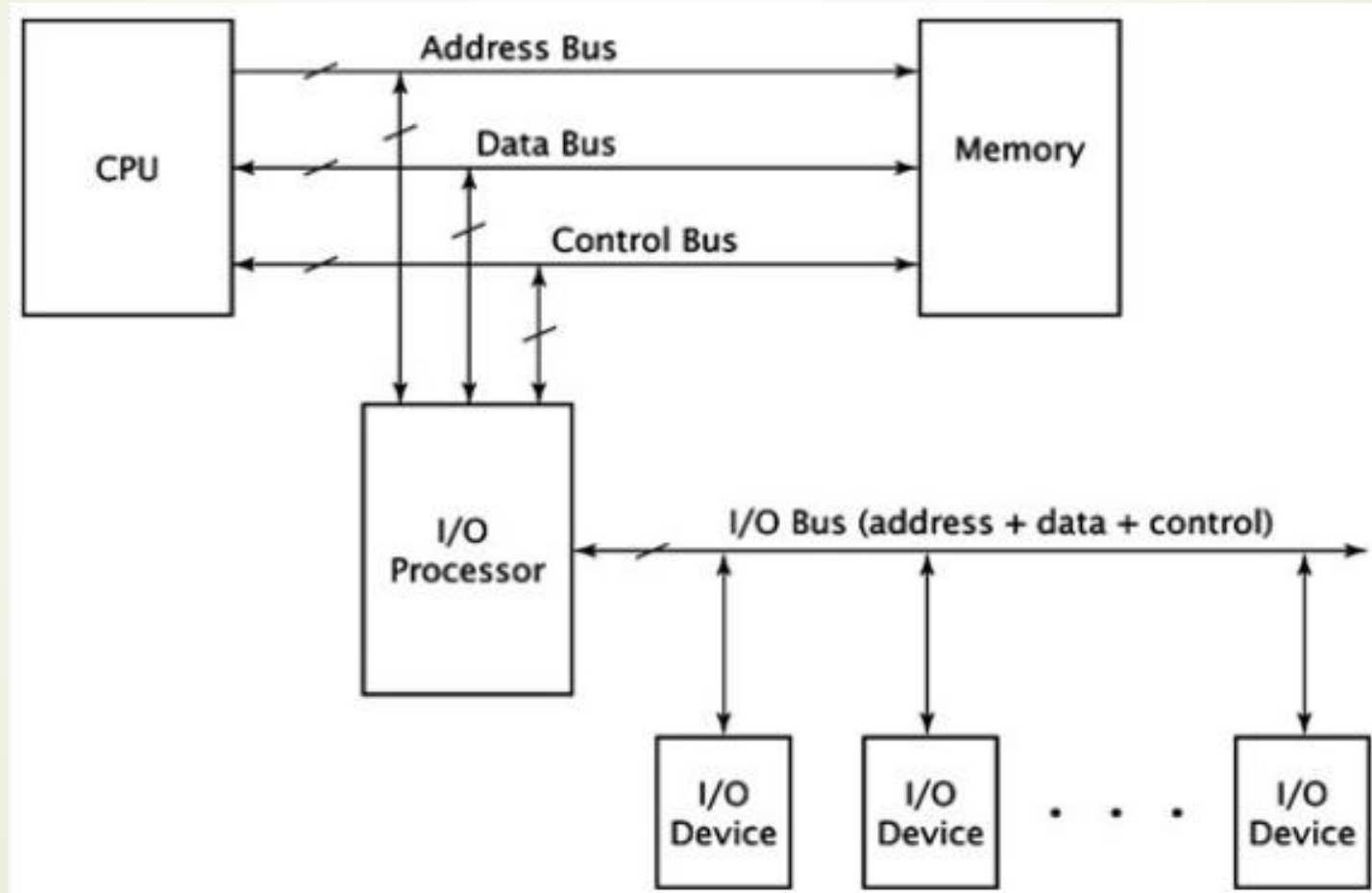


Figure: system configuration incorporating an I/O processor



Contd...

- Are also called I/O controllers, channel controllers or peripheral processing units(PPUs).
- I/O processors usually incorporate several DMA controllers within their circuitry.
- DMA controller can improve system performance by speeding up data transfers between memory and I/O devices.
- In some cases, data must be manipulated once it is read from the I/O device; the DMA controller can only transfer data.
- Unlike DMA controller, the I/O processor connects to more than one I/O device.
- The I/O devices are grouped together on an I/O bus.
- I/O processor can coordinate transfers from several different I/O devices. The only exception is that the CPU coordinates the transfer of data between itself and I/O processor.
- Instead of loading values into registers, as with DMA, the CPU issues a series of I/O instructions to the I/O processor. These instructions are often called commands.



Contd...

- The first are the **block transfer commands**. These move block of data and include the necessary parameters.
=>These instructions can be used to swap pages in and out of physical memory, and to load programs from disk to memory.
- The second type of command performs **arithmetic, logic and branch operations**.
=> used by the CPU for data manipulation.
- The third type of command is **control commands**.
=>These are usually hardware dependent and critical to the proper functioning of the computer system.



Serial communication:

- The I/O devices, DMA controllers and I/O processors use parallel communication. They transmit more than one bit of data at a time.
- Some device cannot handle more than one bit of data at any given time by design; they utilize serial communication.
- Almost always, the CPU does not communicate serially with such devices. Instead, the CPU interacts with the device using parallel communications; an interface or the device itself converts the data between serial and parallel form.
- **Asynchronous serial communication**
 - => used to interact with devices outside of the computer.
 - => connected devices do not share common clock.
 - => they transmit individual bytes of data, rather than large block.
- **Synchronous serial communication**
 - => is more efficient.
 - => transmits block of data in frames, which consist of leading transmission information, the data, and trailing transmission information.

Basics of serial communication

- When two devices communicate using asynchronous signal transmission, they do not share a common clock. They must have some means of synchronization.
- For doing this, they must agree beforehand on several transmission parameters. One of these parameters is speed, the number of bits per second(bps).
=> If the data is transmitted at 28,800 bps but the receiver is reading at 14,400 bps, half the bits are being lost, including control bits, leading for the corruption of data.
- The two devices must agree on the number of data bits per data transmission, whether or not a priority bit is transmitted along with the data and the number of stop bits at the end of transmission.
- Each byte of data is transmitted as a separate entity. The receiving device must recognize when a transmission is occurring, when to read a bit of data; when the transmission is ending; and when the transmission line is idle.
- When the transmission line is idle, its value is logic 1. to signal the start of transmission, the transmitting device outputs a single start bit of 0 onto the line(for one bit time).
[if a system transmits data at 28,800 bps, it has a bit time of $1/28,800 = 34.2\mu\text{s}$.]

Contd...

- It then waits 1 bit time and reads a data bit off of the line, repeating this process for however many data bits are in the transmission.
[the two devices agreed on the number of data bits before beginning the transmission.]
- The LSB of data is transmitted first, then the remaining bits are transmitted in order; the MSB is transmitted last.
- If there's a parity bit, the receiver waits the requisite 1-bit time and reads that bit in as well.
- Whether or not a parity bit is used, the receiver then reads in the stop bit or bits, which must be logic 1.
- Usually 1, 1½ or 2 stop bits are used.

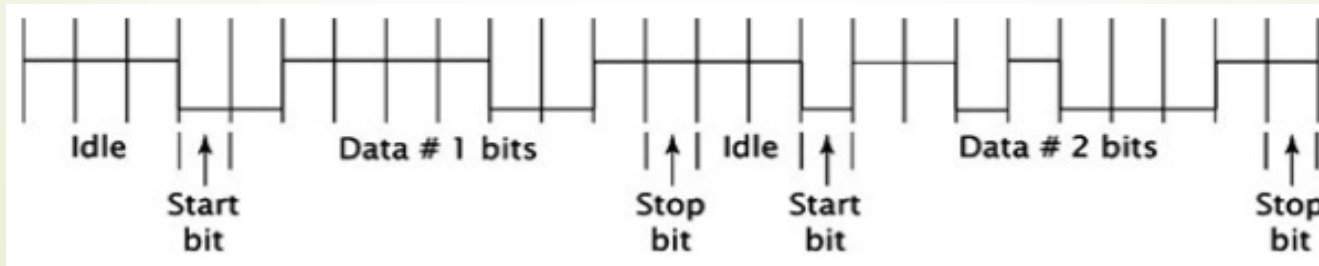
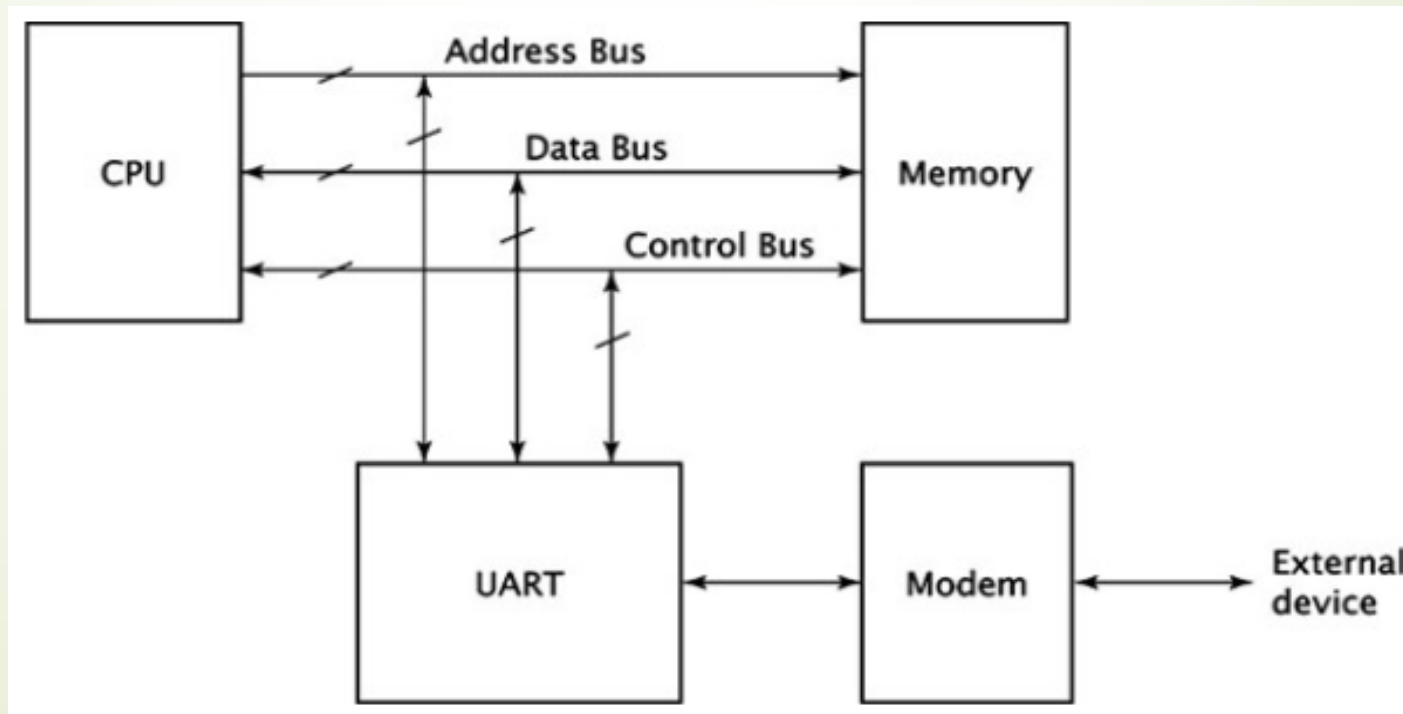


Fig: sample transmission of two byte of data

Universal Asynchronous Receiver/Transmitter(UART):

- For asynchronous serial communication, these specialized chips are called universal asynchronous receiver/transmitter or UARTs.





Contd...

- ▶ CPU sees the UART as just another parallel I/O device and interfaces with it accordingly.
- ▶ UART transmits and receives data serially.
- ▶ Can interact with any device that can access serial data.
- ▶ UART exchanges data with a modem.
- ▶ To transmit data, UART outputs sequential data to the modem, which modulates the data, combining with a carrier frequency and transmitting it.
- ▶ To receive data, the modem accepts a signal at a different carrier frequency and demodulates it, extracts the data, and transmit it serially to the UART.
- ▶ A typical UART contains registers to hold transmitted and received data, a control register and status register.
- ▶ It contains shift registers, which convert data from parallel to serial(for data transmission) and from serial to parallel(for data reception).



USB standards:

- Typically, only one device is connected to a single RS-232-C port, so no address is needed.
- In contrast, the universal serial bus(USB), can connect to several devices.
- The universal serial bus transmits data in packets.
- The USB standard specifies four types of packets, which are used to communicate between a computer and its USB peripherals.
 - => **Token packets** specify address and end points(sender or receiver) for a transfer, or a frame maker.
 - => **Data packets** contain data transferred to or from a device.
 - => **Handshake packets** transfer information used to coordinate data transfers, such as the ACK packets.
 - => **Special packets** with several different functions.