Chapter 8

IO Organization

Assistant Professor

Er. Shiva Ram Dam

Contents:

8.1 Asynchronous Data Transfer

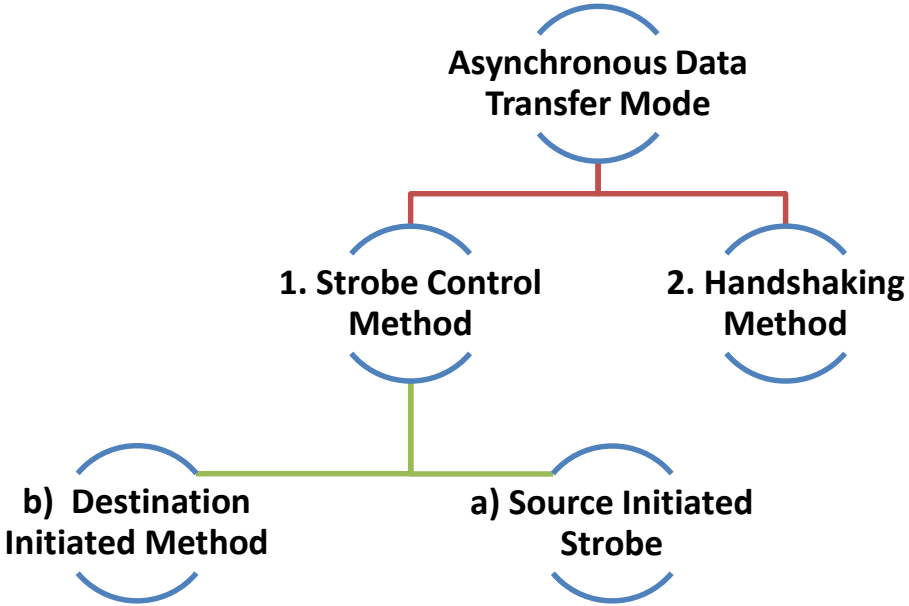
- The internal operations in an individual unit of a digital system are synchronized using clock pulse. It means clock pulse is given to all registers within a unit. And all data transfer among internal registers occurs simultaneously during the occurrence of the clock pulse. Now, suppose any two units of a digital system are designed independently, such as CPU and I/O interface.
- If the registers in the I/O interface share a common clock with CPU registers, then transfer between the two units is said to be **synchronous**. But in most cases, the internal timing in each unit is independent of each other, so each uses its private clock for its internal registers. In this case, the two units are said to be asynchronous to each other, and if data transfer occurs between them, this data transfer is called **Asynchronous Data Transfer**.

- But, the Asynchronous Data Transfer between two independent units requires that control signals be transmitted between the communicating units so that the time can be indicated at which they send data. These two methods can achieve this asynchronous way of data transfer:
  - **Strobe control:** A strobe pulse is supplied by one unit to indicate to the other unit when the transfer has to occur.
  - **Handshaking:** This method is commonly used to accompany each data item being transferred with a control signal that indicates data in the bus. The unit receiving the data item responds with another signal to acknowledge receipt of the data.

8.2 Asynchronous Data Transfer Modes

- The asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate when they send the data.
- Thus, the two methods can achieve the asynchronous way of data transfer.
  - Strobe Control Method
  - Handshaking Method

Asynchronous Data Transfer Modes (Contd.)

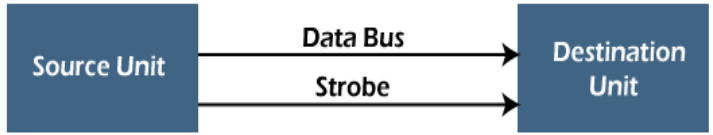


1. Strobe Control Method

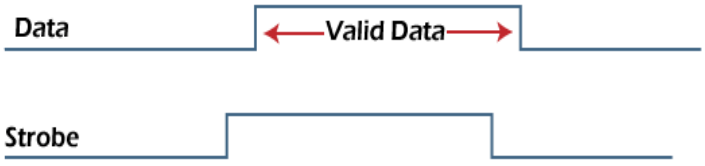
- The Strobe Control method of asynchronous data transfer employs a single control line to time each transfer.
- This control line is also known as a strobe, and it may be achieved either by source or destination, depending on which initiate the transfer.

a) Source Initiated Strobe

- strobe is initiated by source, and as shown in the timing diagram, the source unit first places the data on the data bus.
- After a brief delay to ensure that the data resolve to a stable value, the source activates a strobe pulse.
- The information on the data bus and strobe control signal remains in the active state for a sufficient time to allow the destination unit to receive the data.



(a) Block Diagram

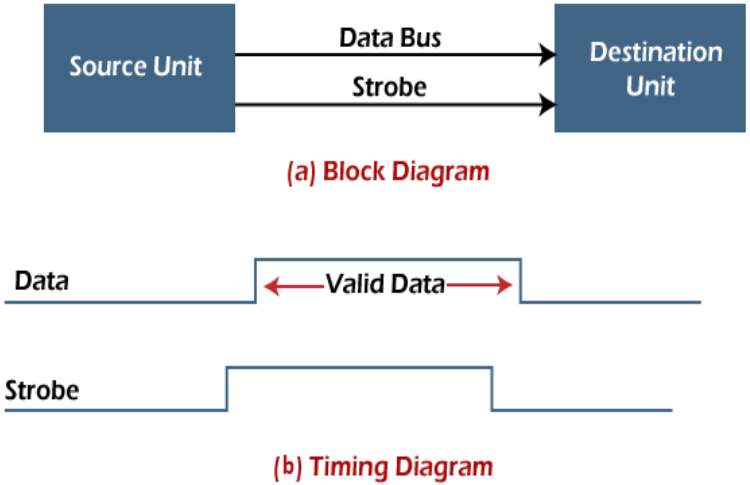


(b) Timing Diagram

a) Source Initiated Strobe

- The destination unit uses a falling edge of strobe control to transfer the contents of a data bus to one of its internal registers.
- The source removes the data from the data bus after it disables its strobe pulse.
- Thus, new valid data will be available only after the strobe is enabled again.

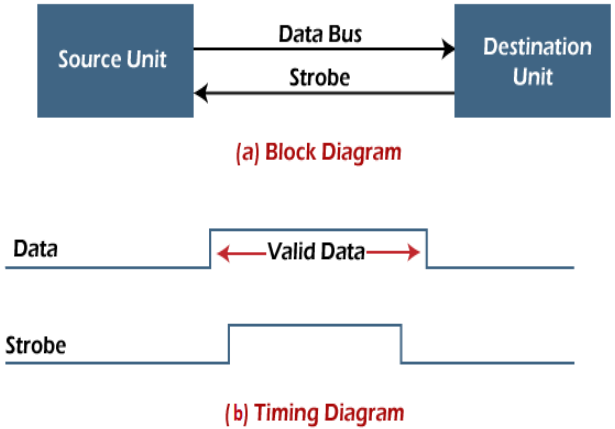
- In this case, the strobe may be a memory-write control signal from the CPU to a memory unit.
- The CPU places the word on the data bus and informs the memory unit, which is the destination.



b) Destination Initiated Strobe

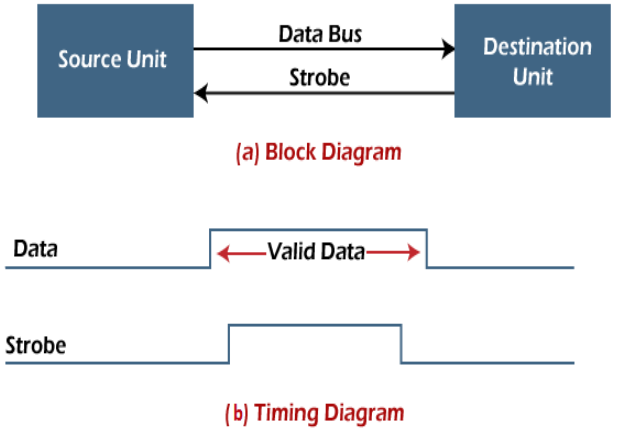
- Strobe initiated by destination, and in the timing diagram, the destination unit first activates the strobe pulse, informing the source to provide the data.

- The source unit responds by placing the requested binary information on the data bus.
- The data must be valid and remain on the bus long enough for the destination unit to accept it.



b) Destination Initiated Strobe (Contd.)

- The falling edge of the strobe pulse can use again to trigger a destination register.
- The destination unit then disables the strobe.
- Finally, and source removes the data from the data bus after a determined time interval.



- In this case, the strobe may be a memory read control from the CPU to a memory unit.
- The CPU initiates the read operation to inform the memory, which is a source unit, to place the selected word into the data bus.

Limitation of Strobe method

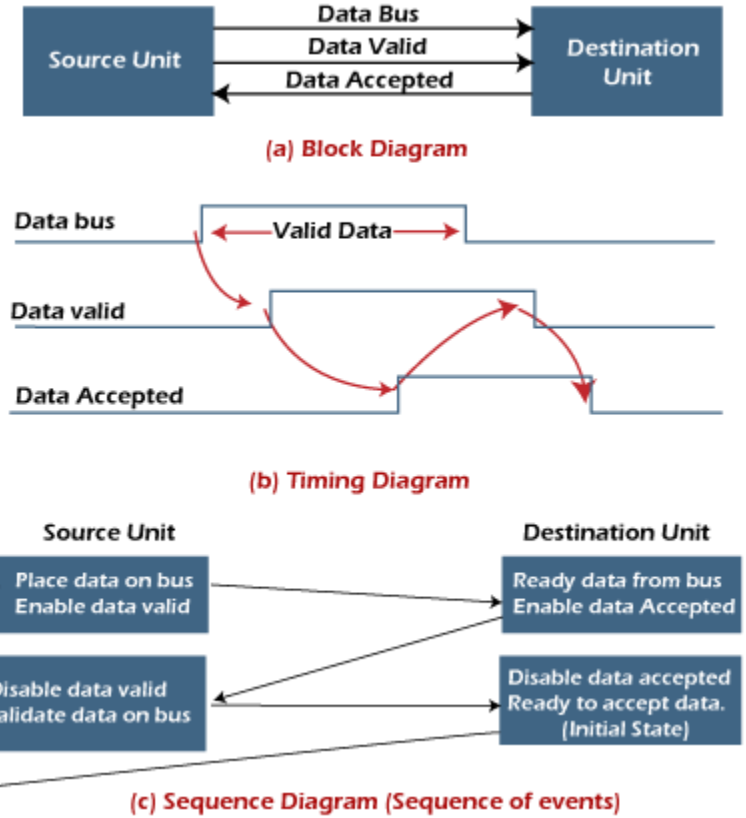
- The strobe method has the disadvantage that the source unit that initiates the transfer has no way of knowing whether the destination has received the data that was placed in the bus.
- Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has placed data on the bus.

2. Handshaking Method

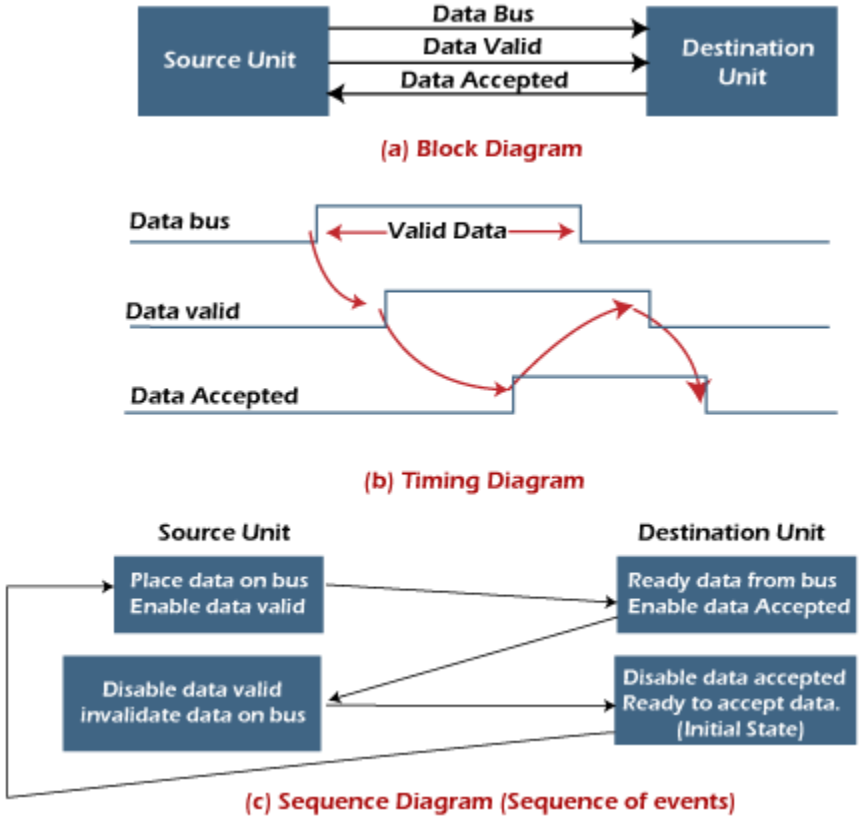
- The limitation of strobe is solved by the handshaking method.
- The handshaking method introduces a second control signal line that replays the unit that initiates the transfer.
- In this method, one control line is in the same direction as the data flow in the bus from the source to the destination. The source unit uses it to inform the destination unit whether there are valid data in the bus.
- The other control line is in the other direction from the destination to the source. This is because the destination unit uses it to inform the source whether it can accept data. And in it also, the sequence of control depends on the unit that initiates the transfer. So it means the sequence of control depends on whether the transfer is initiated by source and destination.

a) Source Initiated Handshaking

- This mechanism uses two handshaking lines:
  - "**data valid**", which is generated by the source unit, and
  - "**data accepted**", generated by the destination unit.

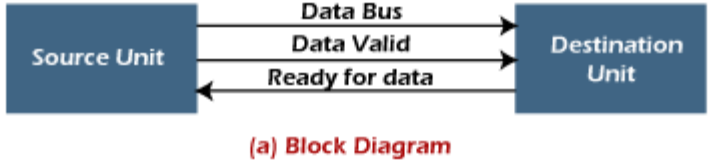


- The source initiates a transfer by placing data on the bus and enabling its data valid signal.
- The destination unit then activates the data accepted signal after it accepts the data from the bus.
- The source unit then disables its valid data signal, which invalidates the data on the bus.
- After this, the destination unit disables its data accepted signal, and the system goes into its initial state.
- The source unit does not send the next data item until after the destination unit shows readiness to accept new data by disabling the data accepted signal.

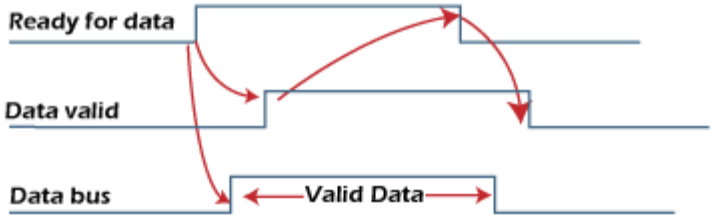


b) Destination Initiated Handshaking

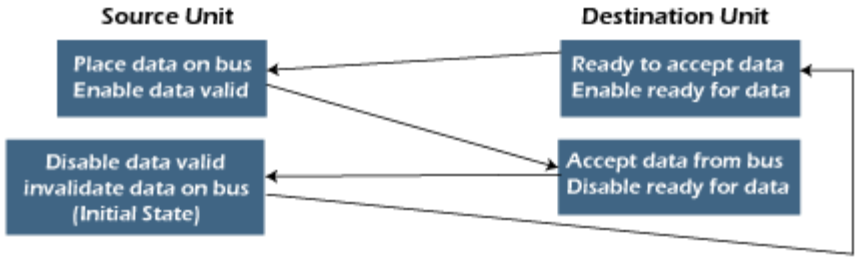
- The two handshaking lines are:
  - "**data valid**", generated by the source unit, and
  - "**ready for data**" generated by the destination unit.
- The destination transfer is initiated, so the source unit does not place data on the data bus until it receives a ***ready data signal*** from the destination unit.
- After that, the handshaking process is the same as that of the source initiated.



(a) Block Diagram



(b) Timing Diagram



(c) Sequence Diagram (Sequence of events)

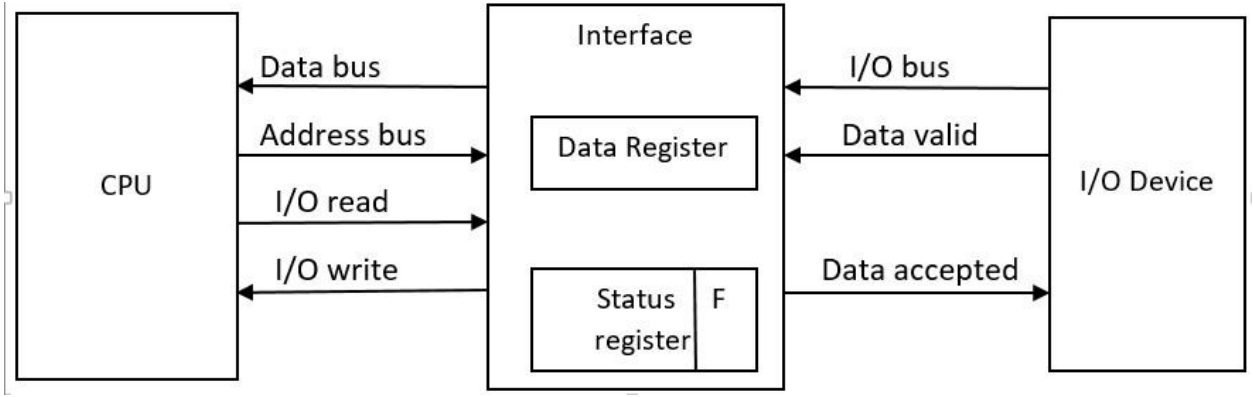
8.3 IO Techniques (Modes of IO transfer)

- I/O operation deals with the exchanges of data between the memory and the external devices either in the direction to the memory (READ) or in the direction from the memory (WRITE).
- But the problem arises on how the processor will manage the flow of data to and from the external devices in term of transfer speed, processor idle time, complexity and etc.
- So in general, there are three techniques for I/O operation, which are:
  - Programmed I/O
  - Interrupt Driven I/O
  - DMA

I) Programmed I/O

- In the programmed I/O method, the I/O device doesn't have direct memory access.
- The data transfer from an I/O device to memory requires the execution of a program or several instructions by CPU So that this method says Programmed I/O.
- In this method, the CPU stays in a program loop until the I/O unit indicates that is ready for data transfer.
- It is a time-consuming process for the CPU.
- The programmed I/O method is particularly useful in small low-speed computers.
- The CPU sends the '**Read**' command to the I/O device and wait in the program loop until it receives a response from the I/O device.

For More Understanding: https://www.youtube.com/watch?v=4SezIktwRhk



- The device transfers **bytes of data** one at a time as they are available.
- When a byte of data is available , the device places it in the **I/O bus and enables its data valid line.**
- The **interface accepts** the byte into its **data register** and enables **the data accepted line.**
- The interface sets a bit in the **status register** that we will refer to as an **F or " flag"** bit.
- The device can now disable the **data valid line**, but it will not transfer another byte until the **data accepted line** is disabled by the interface.

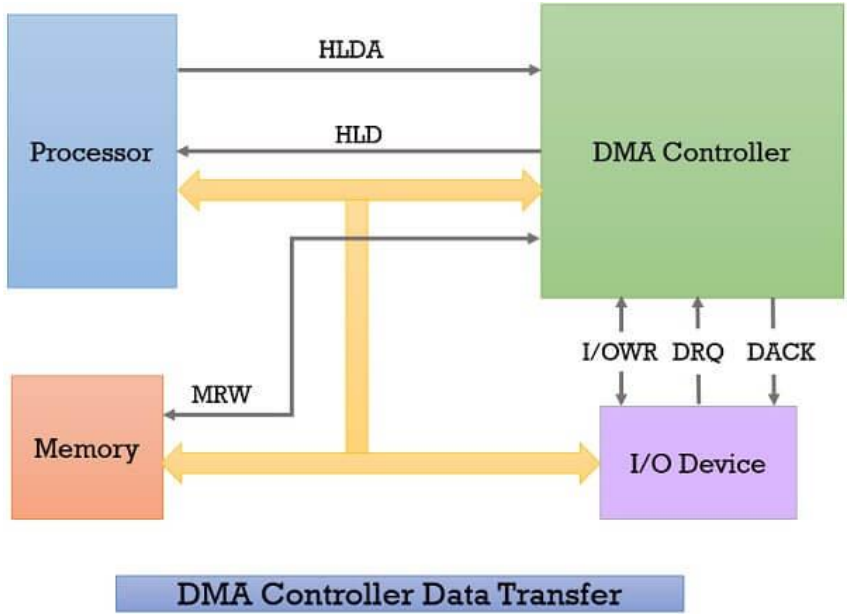
ii) Interrupt-initiated I/O

- The problem in programmed I/O is that the CPU has to wait for the ready signal from the I/O device.
- The alternative method for this problem is **Interrupt-initiated I/O** or **Interrupt driven I/O**.
- In this method, the CPU issues a read command to the I/O device about the status, and then go on to do some useful work.
- When the I/O device is ready, the I/O device sends an interrupt signal to the processor.
- When the CPU receives the interrupt signal from the I/O device, it checks the status.
- If the status is ready, then the CPU read the word from the I/O device and write the word into the main memory.
- If the operation was done successfully, then the processor goes on to the next instruction.

iii) Direct Memory Access

- **Direct Memory Access** (DMA) transfers the block of data between the *memory* and *peripheral devices* of the system, **without the participation** of the **processor**.
- The unit that controls the activity of accessing memory directly is called a **DMA controller**.
- The processor **relinquishes the system bus** for a few clock cycles. So, the DMA controller can accomplish the task of data transfer via the system bus.

- Whenever an I/O device wants to transfer the data to or from memory, it sends the DMA request (**DRQ**) to the DMA controller. DMA controller accepts this DRQ and asks the CPU to hold for a few clock cycles by sending it the Hold request (**HLD**).
- CPU receives the Hold request (HLD) from DMA controller and relinquishes the bus and sends the Hold acknowledgement (**HLDA**) to DMA controller.
- After receiving the Hold acknowledgement (HLDA), DMA controller acknowledges I/O device **(DACK)** that the data transfer can be performed and DMA controller takes the charge of the system bus and transfers the data to or from memory.
- When the data transfer is accomplished, the DMA raise an **interrupt** to let know the processor that the task of data transfer is finished and the processor can take control over the bus again and start processing where it has left.



DMA Controller Data Transfer

Interrupts and its types:

- Interrupt is an external event or signal generated by I/O device to process to request for service.
- Types of Interrupts: -
  - External Interrupts
  - Internal Interrupts
  - Software Interrupts

i) External interrupt

- External interruptions are generated by peripheral devices, such as keyboards, printers, communication cards, etc.
- The External Interrupt occurs when any Input and Output Device request for any Operation and the CPU will Execute that instructions first.
- These interruptions are not sent directly to the CPU but they are sent to an integrated circuit whose function is to exclusively handle this type of interruptions.

ii) Internal Interrupt

- As clear to its name, internal interrupts are those which are occurred due to Some Problem in the Execution.
- It seems to be used mostly to refer to what are also known as exceptions: interrupts that occur in response to a processing error, such as referencing an invalid address in memory, division by zero, or similar error condition.

iii) Software Interrupt

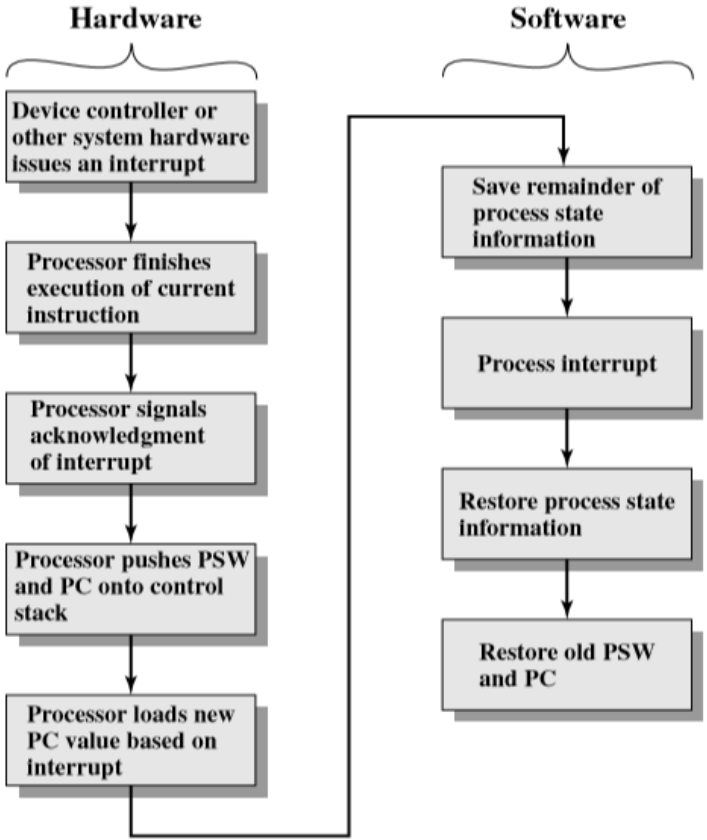
- A software interrupt is a type of interrupt generated by executing an instruction is called software interrupt.
- A software interrupt is invoked by software, unlike a hardware interrupt, and is considered one of the ways to communicate with the kernel or to invoke system calls, especially during error or exception handling.
- Software interrupts are generally used to make system calls.

Maskable and Non-maskable Interrupt

- An interrupt that can be disabled by writing some instruction is known as Maskable Interrupt.
- A non-maskable interrupt (NMI) is a hardware interrupt that standard interrupt-masking techniques in the system cannot ignore.

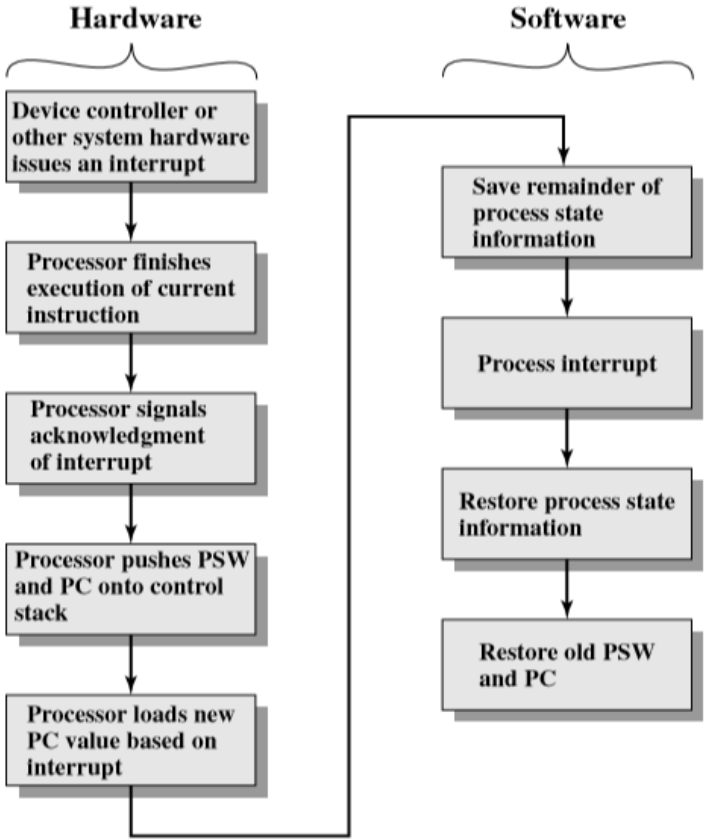
Interrupt Processing

1. Input output device issues the interrupts.
2. The Processor finishes the execution of an instruction before responding to the interrupt.
3. The Processor checks for an interrupt. If there is one, it then sends an **acknowledgement signal** to the input output device that issued the interrupt. The signal allows the device to remove its interrupt signal.
4. To switch to run **interrupt Handler,** information about the current program is stored, so that it execution maybe resumed later including Program status word(psw) and program counter(PC).



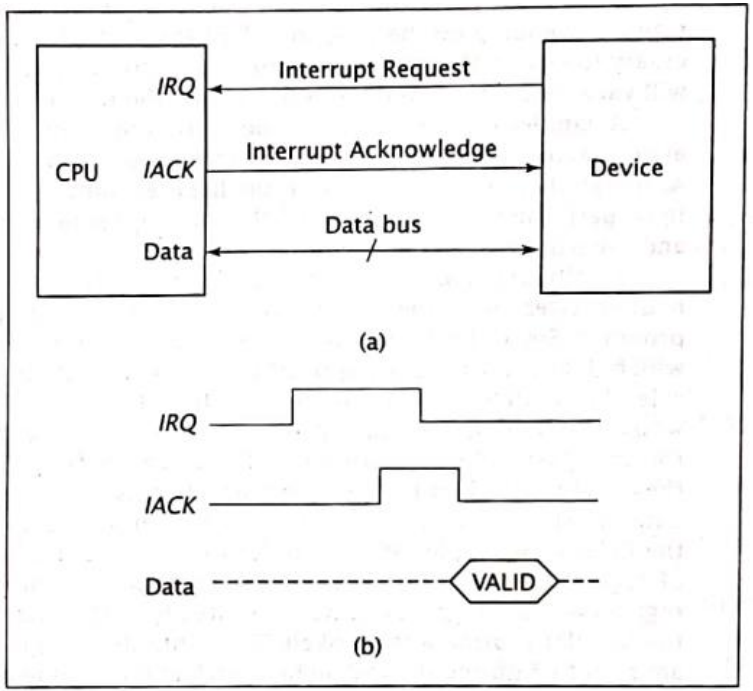
Interrupt Processing

5. The processor loads the **program counter** with the entry location of the interrupt Handler. Typical case is there are set of routines. Each for one type of interrupt or each for one device.
6. The **Interrupt Handler** may continue to save other information that is considered as part of process state.
7. The handler performs the interrupt processing.
8. When handler finishes, the saved Register values are stored into the registers that originally hold them when interrupt handler returns.
9. Finally PSW and PC values of the interrupted program are restored, thus the program may continue to execute.



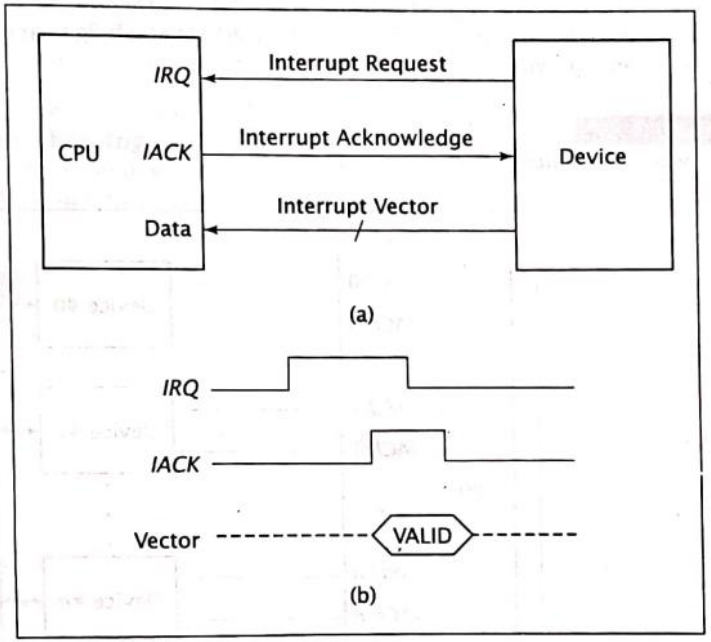
Interrupt Hardware and Priority

- An external device sends and interrupt to the CPU by asserting its Interrupt Request (IRQ) signal.
- When the CPU is ready to process the interrupt request, it asserts its interrupt acknowledge signal (IACK), thus informing the I/O device that it is ready to proceed.
- Because its request has been acknowledged, the device sets IRQ low, which causes the CPU to set IACK low.
- As the handler routine proceeds, it transfers data between the CPU and the interrupting device.



(a) Hardware and (b) timing for a nonvectored interrupt for a single device

- In a vectored interrupt, data is transferred later via a synchronous transfer.
- A **vectored interrupt** is more complex.
- After acknowledging the interrupt, the CPU must input an interrupt vector from the device and call an Interrupt Service Routine (ISR) handler; the address of this routine is a function of this vector.
- Figure aside show a hardware block diagram and timing diagram for this system.

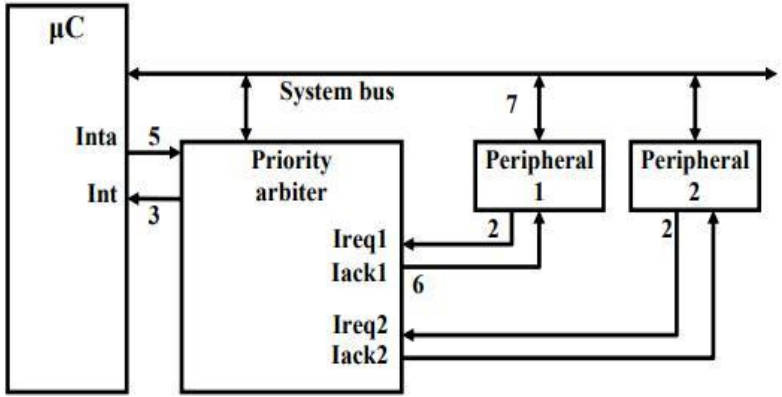


(a) Hardware and (b) timing for a vectored interrupt for a single device

Priority Interrupt

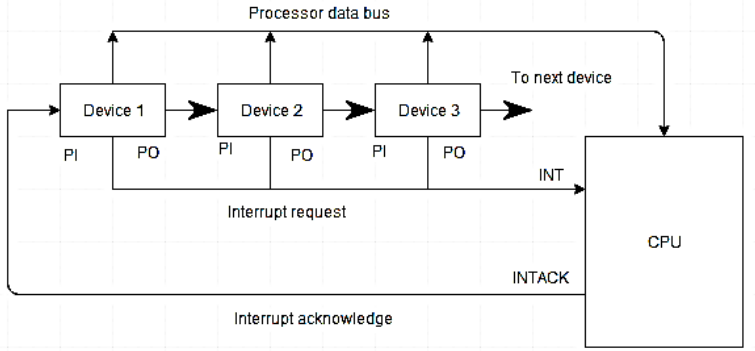
- A priority interrupt is a system which decides the priority at which various devices, which generates the interrupt signal at the same time, will be serviced by the CPU.
- The system has authority to decide which conditions are allowed to interrupt the CPU, while some other interrupt is being serviced.
- Generally, devices with high speed transfer such as *magnetic disks* are given high priority and slow devices such as *keyboards* are given low priority.
- When two or more devices interrupt the computer simultaneously, the computer services the device with the higher priority first.

1. The Processor is executing its program.
2. Peripheral1 needs servicing so asserts **Ireq1**. Peripheral2 also needs servicing so asserts **Ireq2**.
3. Priority arbiter sees at least one I**req** input asserted, so asserts **Int**.
4. Processor stops executing its program and stores its state.
5. Processor asserts **Inta**.
6. Priority arbiter asserts **Iack1** to acknowledge Peripheral1.
7. Peripheral1 puts its interrupt address vector on the system bus
8. Processor jumps to the address of ISR read from data bus, ISR executes and returns (and completes handshake with arbiter).



Daisy Chaining Priority

- In daisy chaining system all the devices are connected in a serial form. The device with the highest priority is placed at the first position followed by lower priority devices
- The interrupt line request is common to all devices.
- When there is no interrupt, the interrupt line (i.e. Interrupt request) stays in high level state.
- If any device has interrupt signal in low level state then interrupt line goes to low level state and enables the interrupt input in the CPU.
- The CPU respond to the interrupt by enabling the interrupt acknowledge line.
- This signal is received by the device 1 at its PI input.
- The acknowledge signal passes to next device through PO output only if device 1 is not requesting an interrupt. (it sets PI=1 ad PO=1).
- However, if the device had requested the interrupt, (PI =1 & PO = 0)

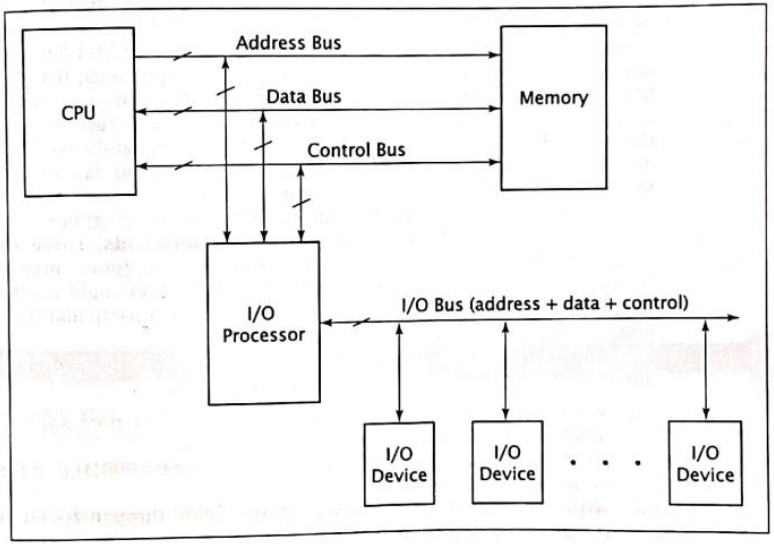


DMA Transfer Modes

1. **Burst transfer mode (Block Transfer Mode)**
   - In this mode, an 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 transfer all bytes of data in the data block before relinquishing control of the system buses back to the CPU.
   - This transfer mode is needed for faster devices such as magnetic desks, where data transmission cannot be stopped or slowed down until an entire block is transformed.
2. **Cycle stealing: -**
   - In this mode, the DMA controller obtains access to the system buses as in the Burst mode.
   - However, it transfers one byte of data and then assert BR signal, returning control of the system buses to the CPU.
   - It continually issues requests via BR, transferring one byte of data per request, until it has transferred its entire block of data.
3. **Transparent mode: -**
   - In the transparent mode the DMA controller transfers data only when the CPU is performing operations that do not use the system bus.
   - This way the DMA transfer never interferes with the CPU executing its programs.
   - Disadvantage of transparent mode is: complexity of determining when the CPU is not using the system bus.
   - This can be inefficient in terms of overall system performance

I/O parocessors

- IO processors, sometimes also called I/O controllers, Channel controllers or peripheral processing units (PPUs)
- The perform the functions of DMA controllers and much more. In fact, they usually incorporate several DMA controllers within their circuitry.
- As shown in figure, the IO processor is situated between the IO devices and the rest of the system.
- The IO processor, unlike DMA controller, connects to more than one IO device and coordinate transfers from several different IO devices.
- It handles all of the interactions between the IO devices and the CPU.



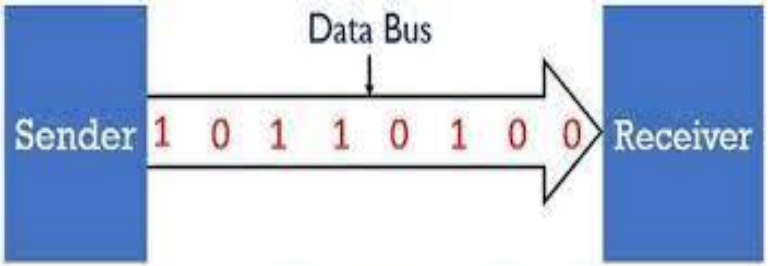
System configuration incorporating an I/O processor

8.6 Serial Communication, UART

- Some devices cannot handle more than one bit of data at a time by design; they utilize serial communication.
- This may be done to reduce cost or to facilitate communication with devices outside of the computer system.
- In fact, CPU interacts with devices using parallel communication. So, and interface or the device itself converts the data between serial parallel forms.
- Asynchronous serial communication is used to interact with devices outside of the computer, such as when connecting to another computer via modem. The connected devices do not share a common clock.
- Synchronous Serial transmission is more efficient. It transmits block of data in frames, which consist of leading transmission information.

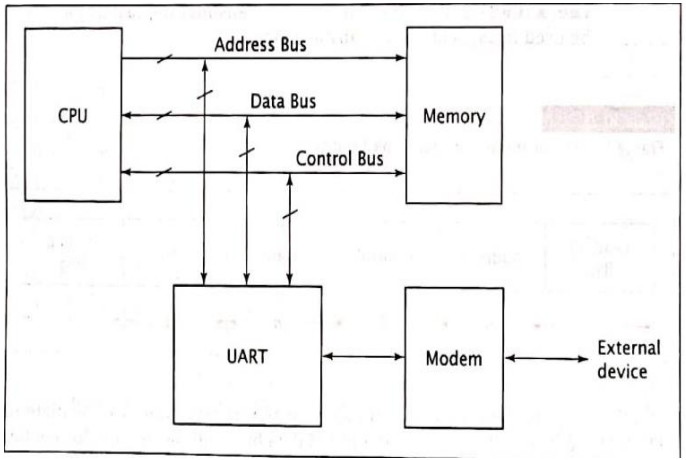
Serial Communication

- Serial communication involves a physical layer that carries one bit of data at a time.
- The data bus is composed of a single data wire, with control and possibly power wires, running from one device to another.
- When data is to be sent, the sender first transmits a start bit. This start bit merely signals the receiver to wakeup and start receiving data.
- The start bit is followed by N data bits, and a stop bit.
- Advantage: low cost and suitable for long distance communication
- Disadvantage: low speed and low data rate.

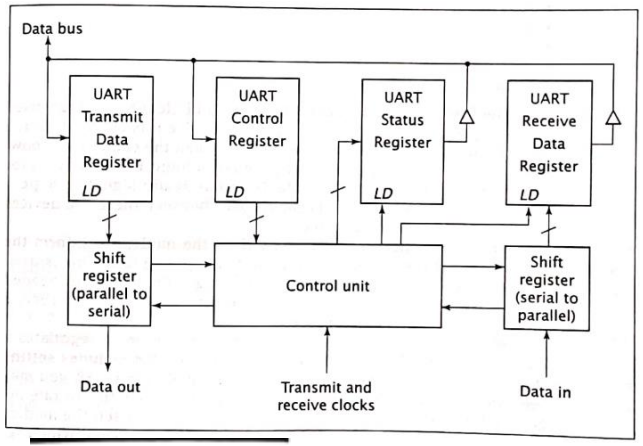


Universal Asyncheonous Receiver/Transmitters (UART)

- A device for asynchronous communication
- UART is parallel I/O device as seen by CPU.
- But it inputs and outputs data serially. It can interact with any device that can access serial data.
- In given figure, UART exchange data with a modem
- For transmitting data, UART O/P sequential data to modem in which modules the data, combining with carrier frequency and transmitting it.
- For receiving data, modem accepts a signal at different carrier frequency and demodulates it, extracts data and transmit it serially to UART.



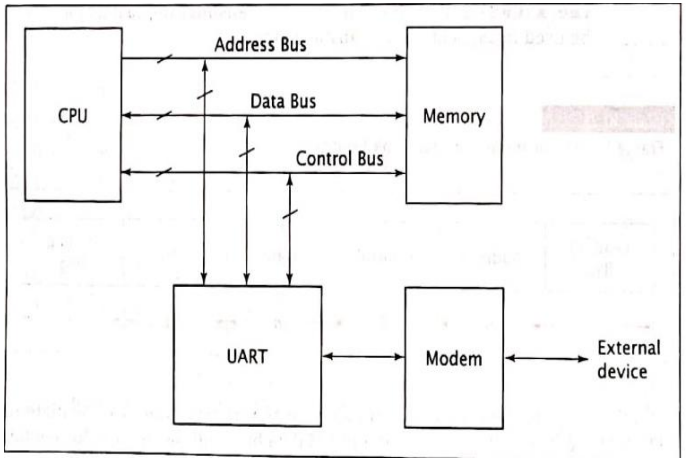
A computer system incorporating a UART



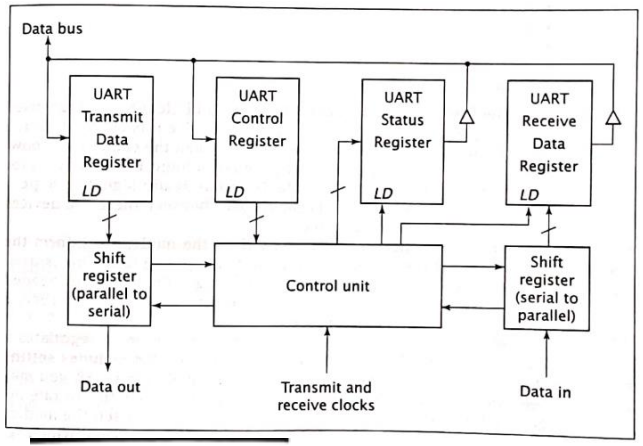
Internal configuration of a UART

Universal Asyncheonous Receiver/Transmitters (UART)

- Th UART has several internal registers, just as in the DMA controller.
- Like the DMA controller, the CPU treats these registers as individual IO ports.
- A typical UART contains registers to hold transmitted and received data, a control register, and a status register, all of which are accessible to the CPU.
- It also contains shift registers, which convert data from parallel to serial (For data transmission) and vice-versa (For data reception)
- The UART's control unit also coordinates the insertion and removal of start, parity and stop bits.



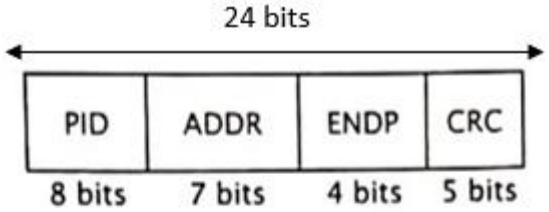
A computer system incorporating a UART



Internal configuration of a UART

8.7 USB standards

- USB is a specification to establish communication between devices and computer. It replace serial and parallel ports.
- USB is an industry standard developed to provide two speed of operations called low speed and full speed. They provide simple, low cost and easy to use interconnection system.
- Low speed- 1.5mbps
- High speed- 12mbps
- USB transmits data in packets. This packet can specify an address, allowing several devices to be connected to USB- port.
- USB port is faster than RS-232C port.
- The USB standard specifies 4 types of packets which are used to communicate between a computer and its USB peripherals.
  - Token packets: - It is used to initiates data transfer. It specifies address and end point for a transfer.
  - Data packets: - It contains data transferred to or from a device
  - Handshake packet: - It transfer information used to coordinate data transfers
  - Special packets: - It includes different other functions.

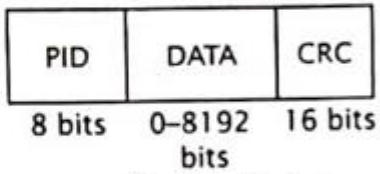


a) Token Packet

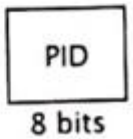
PID = Packet identifier

ENDP= End of packet

CRC= Cyclic Redundancy Check



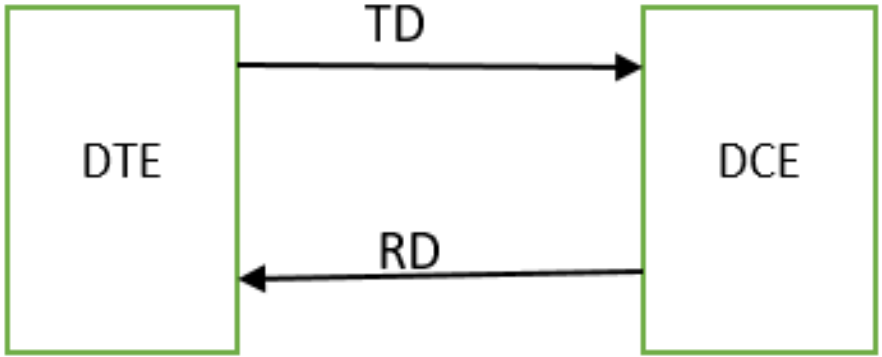
b) Data Packet



c) Handshake Packet

RS 232 C-standard

- RS 232-C is an interface convention developed to standardize the interface between data terminal equipment (DTE) and data common equipment (DCE), serial binary data exchange.



RS 422A Standard:

- The main problem with RS 232-C is that it can only transfer data reliably about so at a maximum rate of 20,000 Bd. This limitation is due to open signal lines with a common ground that are used for RS 232-C
- RS 422A specifies that each signal will send differentially over two adjacent wires in a ribbon or a twisted pair cable. Data rates for this standard are 10 mbps for a distance of soft or 100,000 Bd for a distance of 4000ft.
- It is because the differential lines are terminated by resistors so that they act as simple tx-lines of simply open wire.
- For RS 422A, a logic high or 1 is indicated by B signal line being more positive than A signal line. And logic low or 0 by vice versa. The voltage difference between two lines must be greater than 0.4v but not greater than 12v.

Exam Questions;

1. What is handshaking? How are asynchronous data transfers done in CPU? Explain. [PU 2017 Fall]
2. What are IO processors? Describe how they work using suitable diagram. [PU 2017 Fall]
3. What is DMA? Describe how they work using suitable diagram. [PU 2017 Spring]
4. Differentiate between: [PU 2018 Fall]
   a) Interrupt driven IO and Programmed IO
   b) Vectored and Non Vectored Interrupt Hardware.
5. Why do we need input output module? Compare programmed IO, interrupt driven data transfer and Direct Memory Access (DMA). [PU 2020 Spring]
6. What is DMA? Describe how it works. [PU 2018 Spring]
7. Write Short notes on:
   a) Interrupt Vector [PU 2018 Spring]
   b) DMA [PU 2018 Fall]
   c) Programmed IO [PU 2017 Fall]

End of chapter 8