CHAPTER:8

INPUT OUTPUT ORGANIZATION

BY: ER. PRALHAD CHAPAGAIN

(Contact: pralhad.chapagain@gmail.com)

Syllabus:

- **Asynchronous data transfer**
- **Modes of Asynchronous data transfer**
- **Programmed I/O**
- **Interrupts, Interrupts driven data transfer, Types of interrupts, interrupts processing, interrupt hardware and priority**
- **Direct memory access (DMA), DMA transfer modes, I/O processors**
- **Serial communication, UART**
- **USB standards**

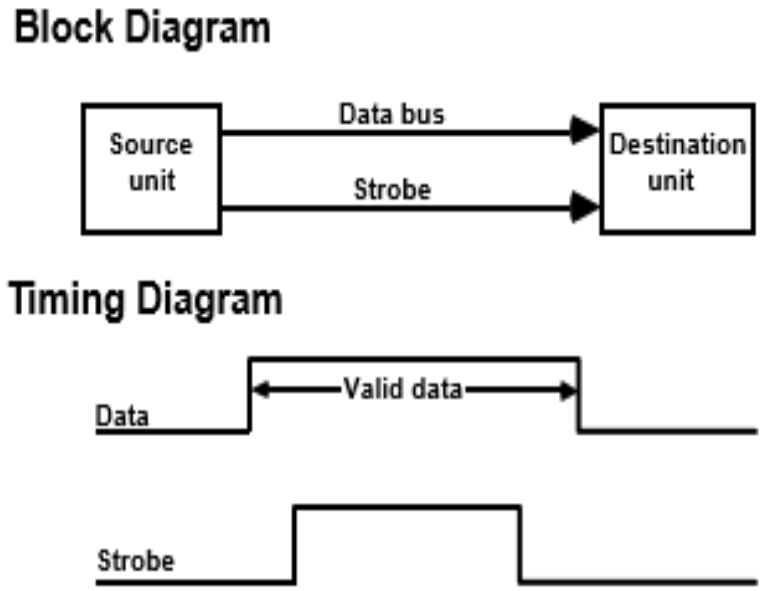
Synchronous and Asynchronous Data Transfer:

- **Synchronous data transfer** usually occurs when peripherals are located within the same computer as the CPU, because their close proximity allows them to share a common clock .
- **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
- Asynchronous Data Transfer is used when speed of I/O devices do not match with microprocessors, and timing characteristics of I/O device is not predictable.
- 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 (1):

- The strobe may be activated by either the source or destination unit.

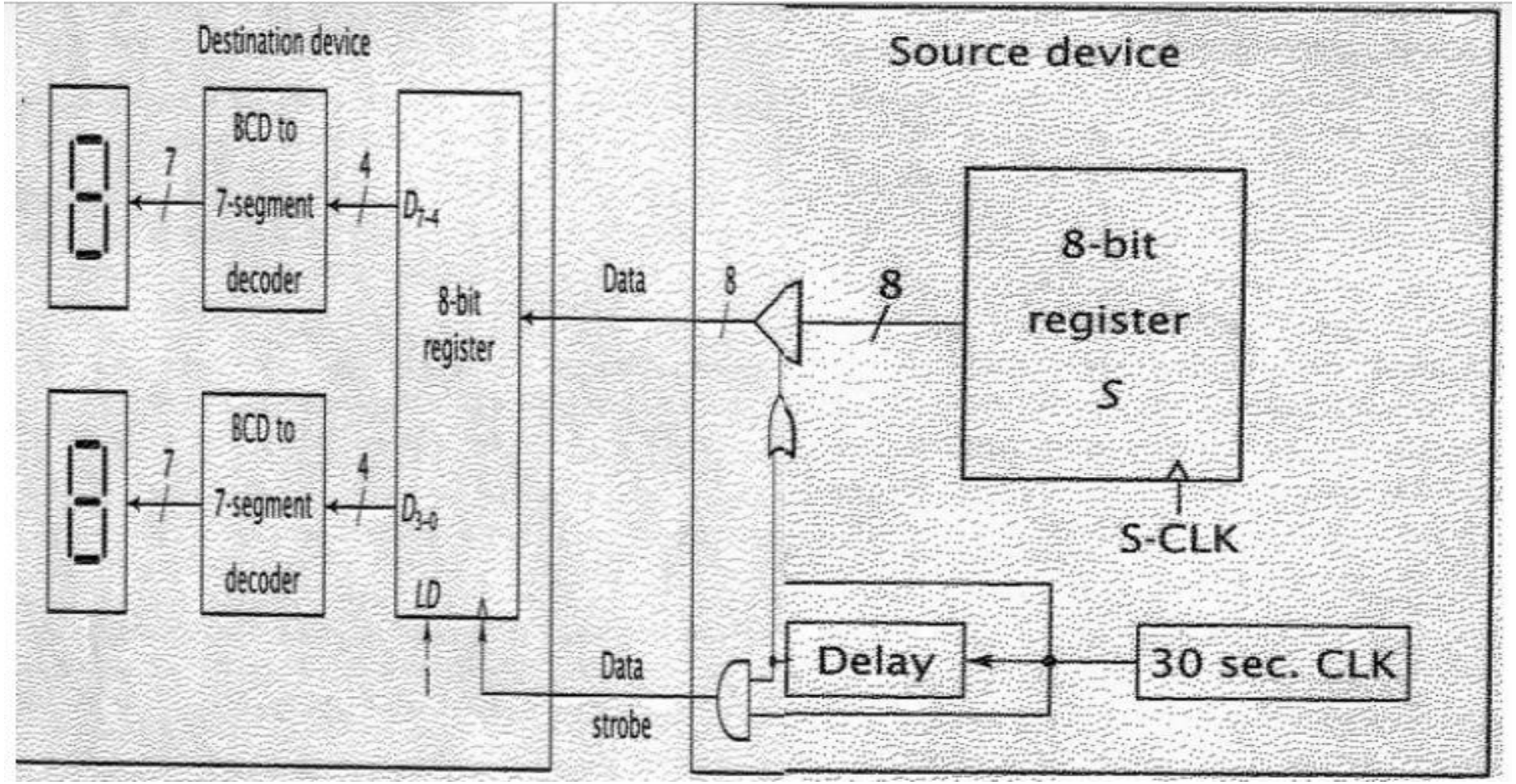
1) Source initiated strobe for data transfer :



- The data bus carries the binary information from source to destination unit
- The strobe is single line that informs the destination unit when a valid data word is available
- The source unit place the data on the data bus
- The information on the data bus and strobe signal remain in the active state to allow the destination unit to receive the data

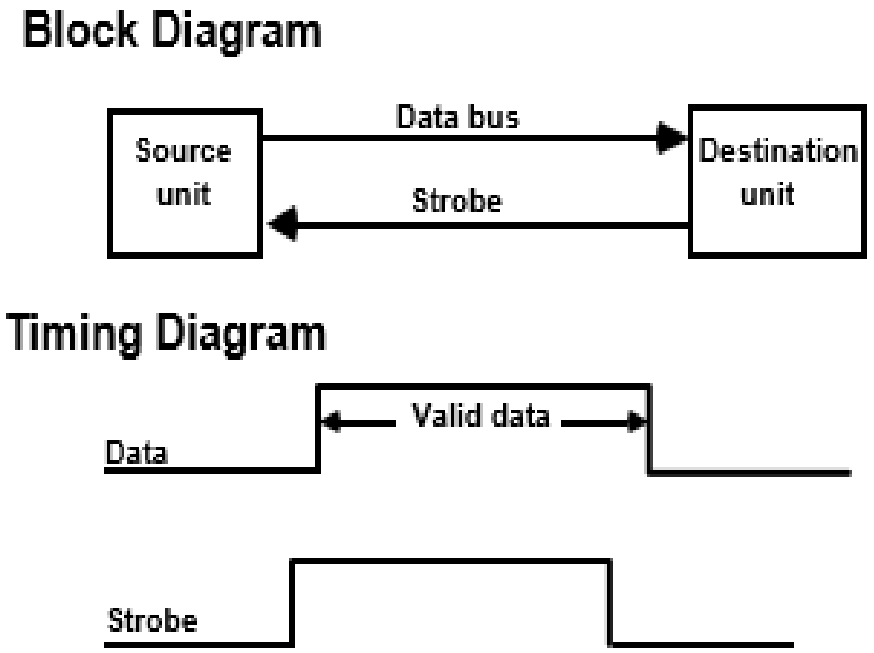
Strobe control (2):

Source initiated strobe for data transfer : Implementation



Strobe control (3):

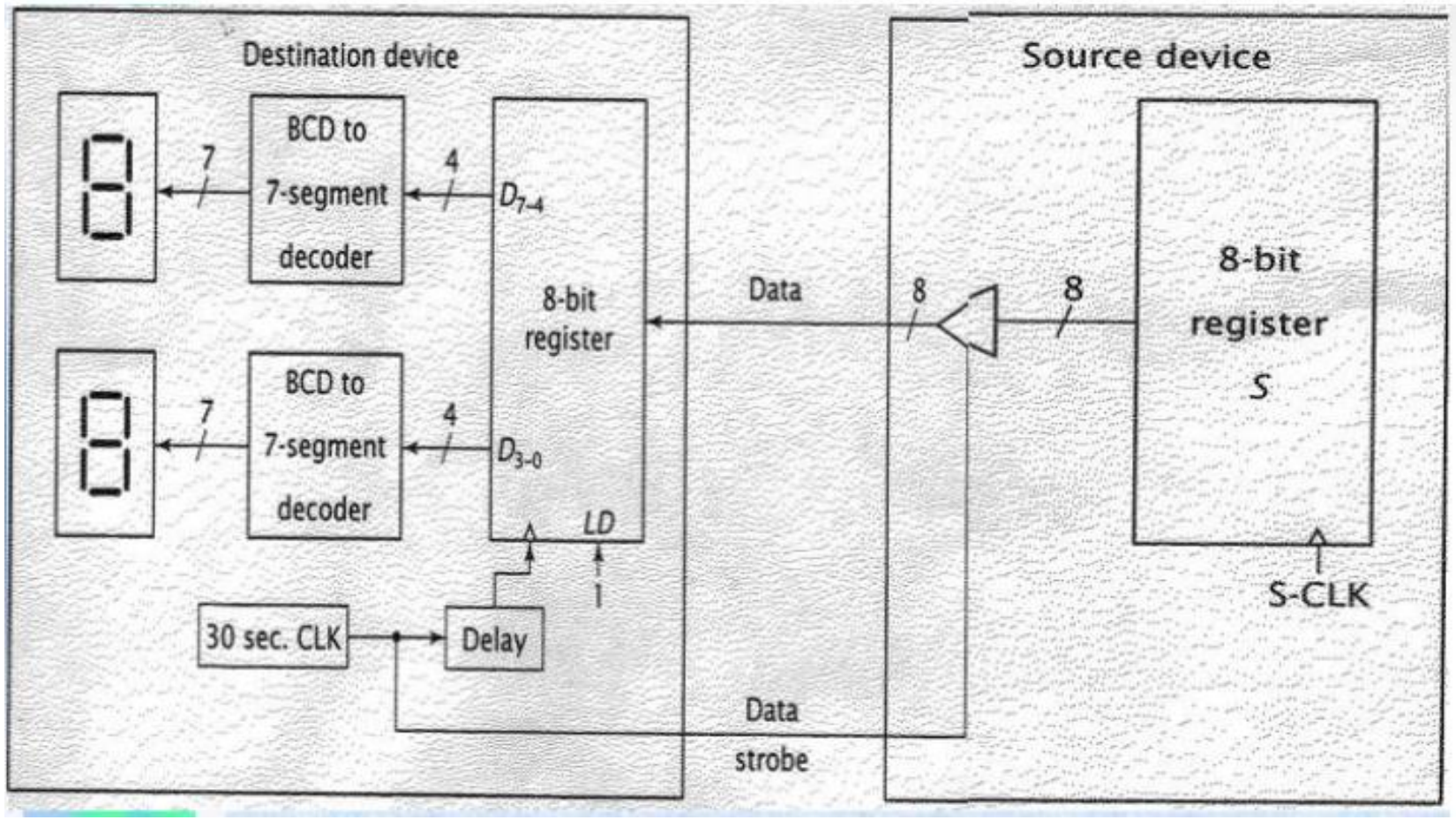
2) Destination initiated strobe for data transfer :



- the destination unit activates the strobe pulse to informing the source to provide the data.
- The source will respond by placing the requested binary information on the data bus
- The data must be valid and remain in the bus long enough for the destination unit to accept it
- When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.

Strobe control (4):

Destination initiated strobe for data transfer : Implementation



Strobe control (5):

Disadvantages:

- The **source** unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was places in the bus.
- Similarly, A **destination** unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus.
- To solve this problem, the **HANDSHAKE** method introduces a second control to provide a reply to the unit that initiates the transfer.

Handshake Control (1):

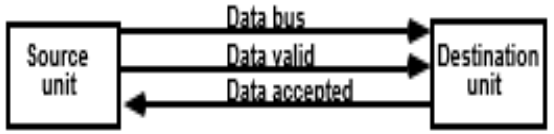
Principle of handshake:

- One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there a valid data in the bus.
- The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data.
- The sequence of control during the transfer depends on the unit that initiate the transfer.

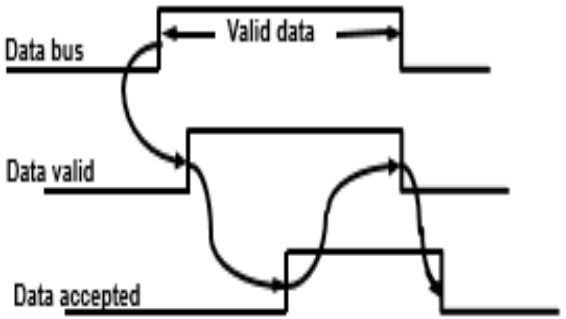
Handshake Control (2):

1) Source initiated transfer using handshaking:

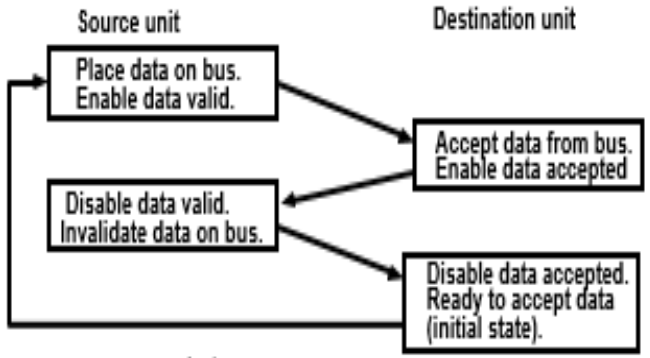
Block Diagram



Timing Diagram



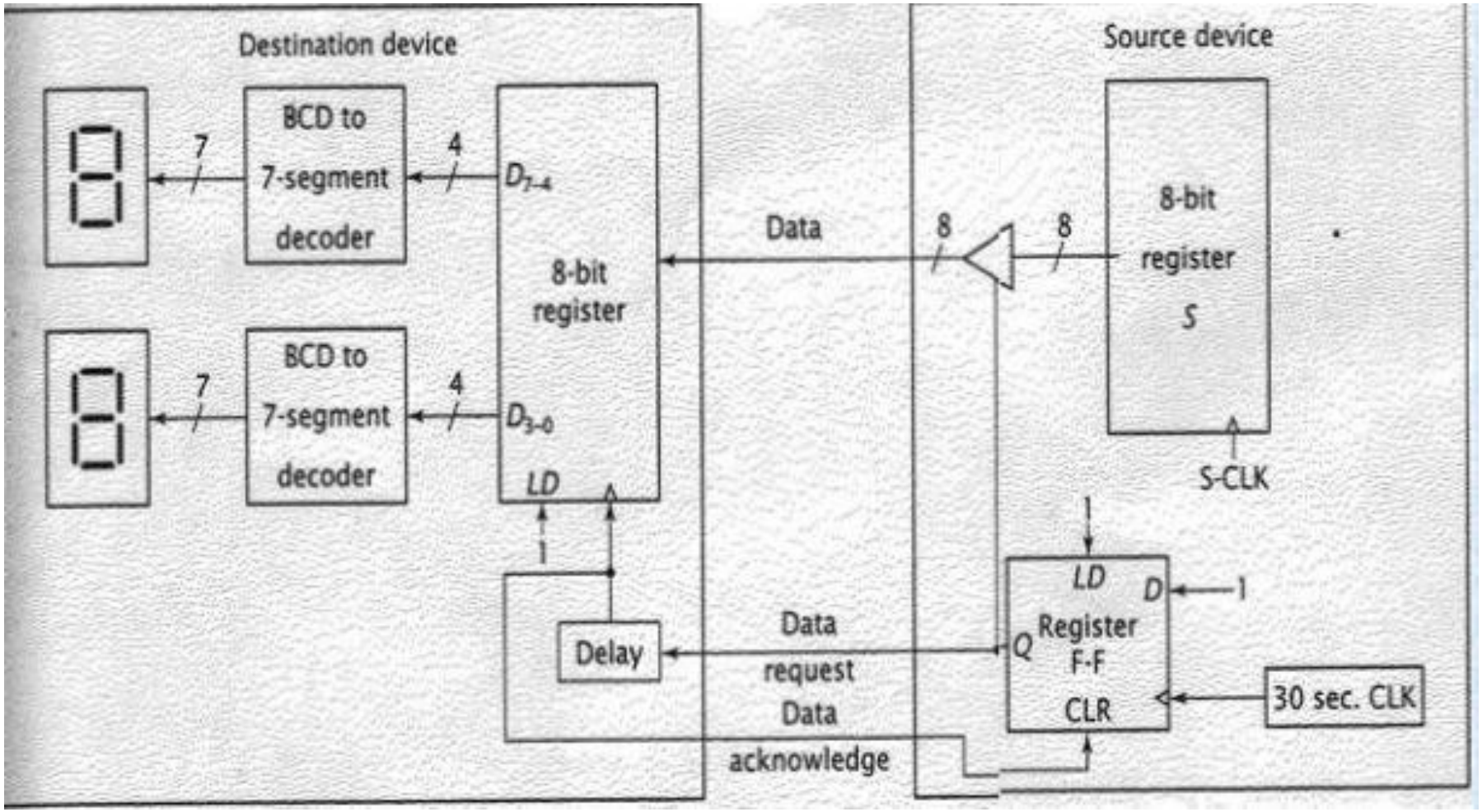
Sequence of Events



- The source unit initiates the transfer by placing the data on the bus and enabling its **data valid** signal.
- The **data accepted** signal is activated by the destination unit after it accepts the data from the bus.
- The source unit then disables its data valid signal, which invalidates the data on the bus
- The destination unit then disables its **data accepted** signal and the system goes into its initial state.

Handshake Control (3):

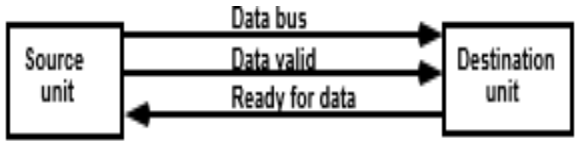
1) Source initiated transfer using handshaking: Implementation



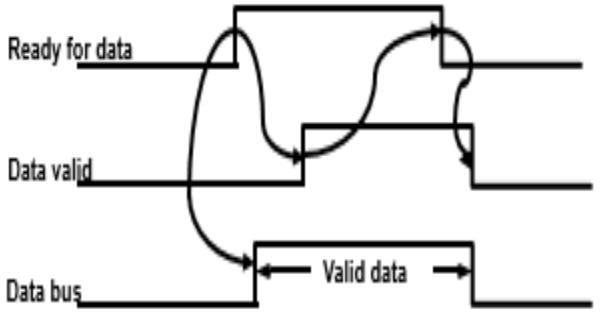
Handshake Control (4):

2) Destination initiated transfer using handshaking:

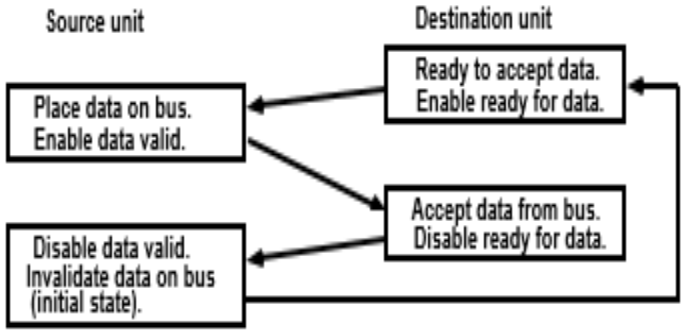
Block Diagram



Timing Diagram



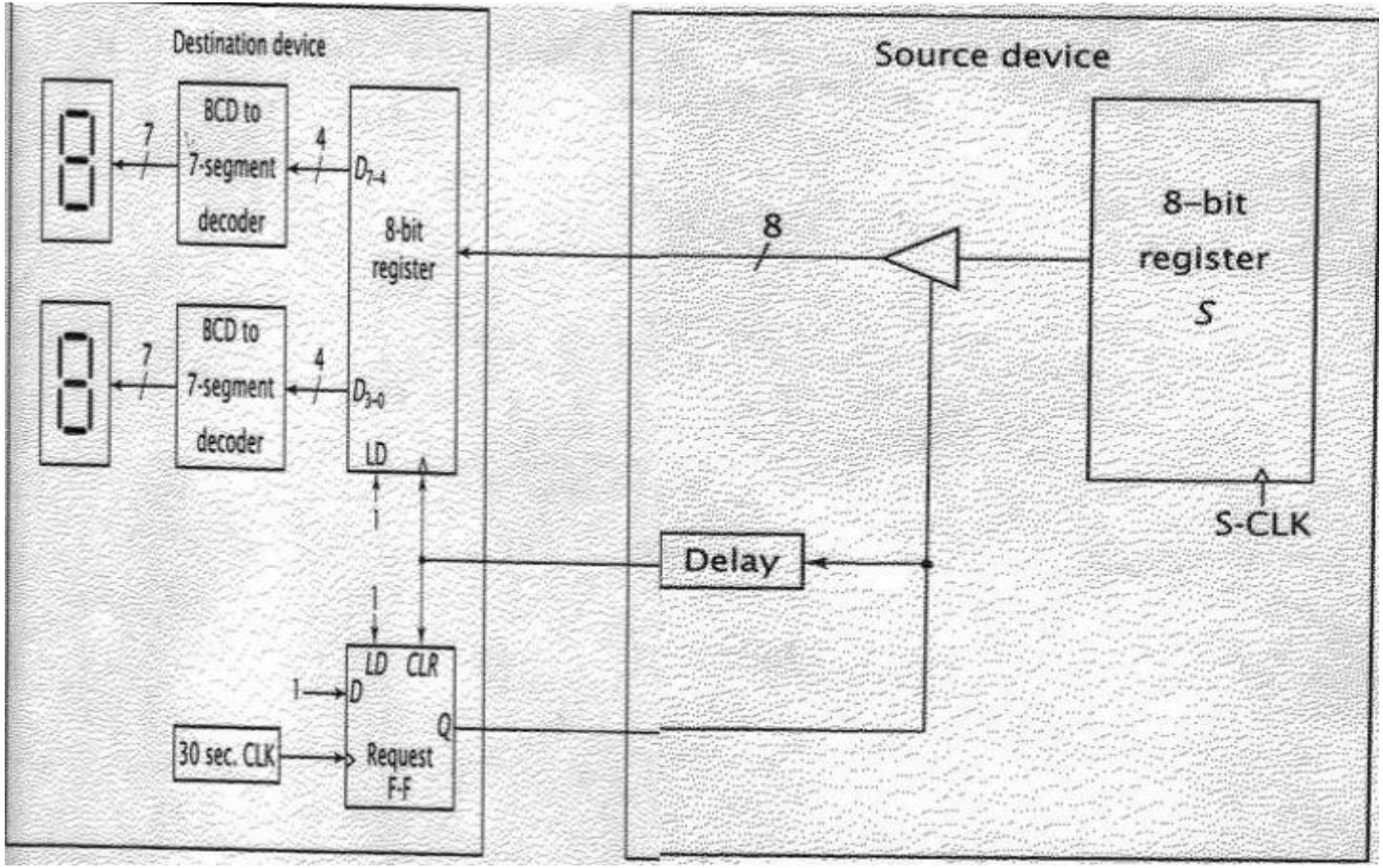
Sequence of Events



- The name of the signal generated by the destination unit has been changed to **ready for data** to reflects its new meaning.
- The source unit in this case does not place the data on the bus until it receives the **ready for data** signal from the destination unit.
- From there on, the handshaking procedure follows the same pattern as in the source initiated case.

Handshake Control (5):

2) Destination initiated transfer using handshaking: Implementation



Handshake Control (6):

Advantages of the handshaking method:

- The handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a **Timeout mechanism** which provides an alarm if the data is not completed within time.

I/O Interface:

- The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface
- The CPU is interfaced using special communication links by the peripherals connected to any computer system
- These communication links are used to resolve the differences between CPU and peripheral.
- There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.

Mode of Transfer:

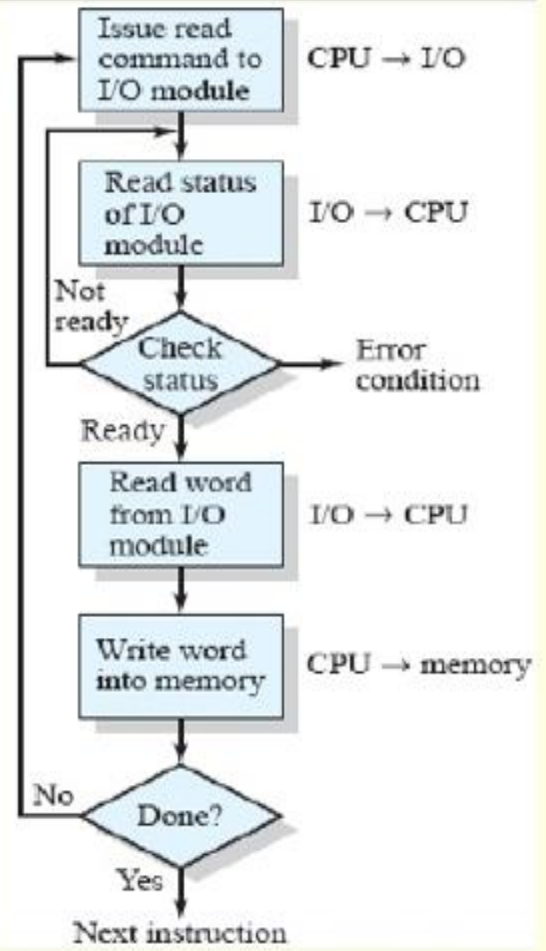
- Data transfer to and from the peripherals may be done in any of the three possible ways
  - Programmed I/O.
  - Interrupt- initiated I/O.
  - Direct memory access (DMA).

Programmed I/O:

- It is due to the result of the I/O instructions that are written in the computer program
- Each data item transfer is initiated by an instruction in the program
- Usually the transfer is from a CPU register and memory
- In this case it requires constant monitoring by the CPU of the peripheral devices.
- The main drawback of the program initiated I/O was that the CPU has to monitor the units all the times when the program is executing.

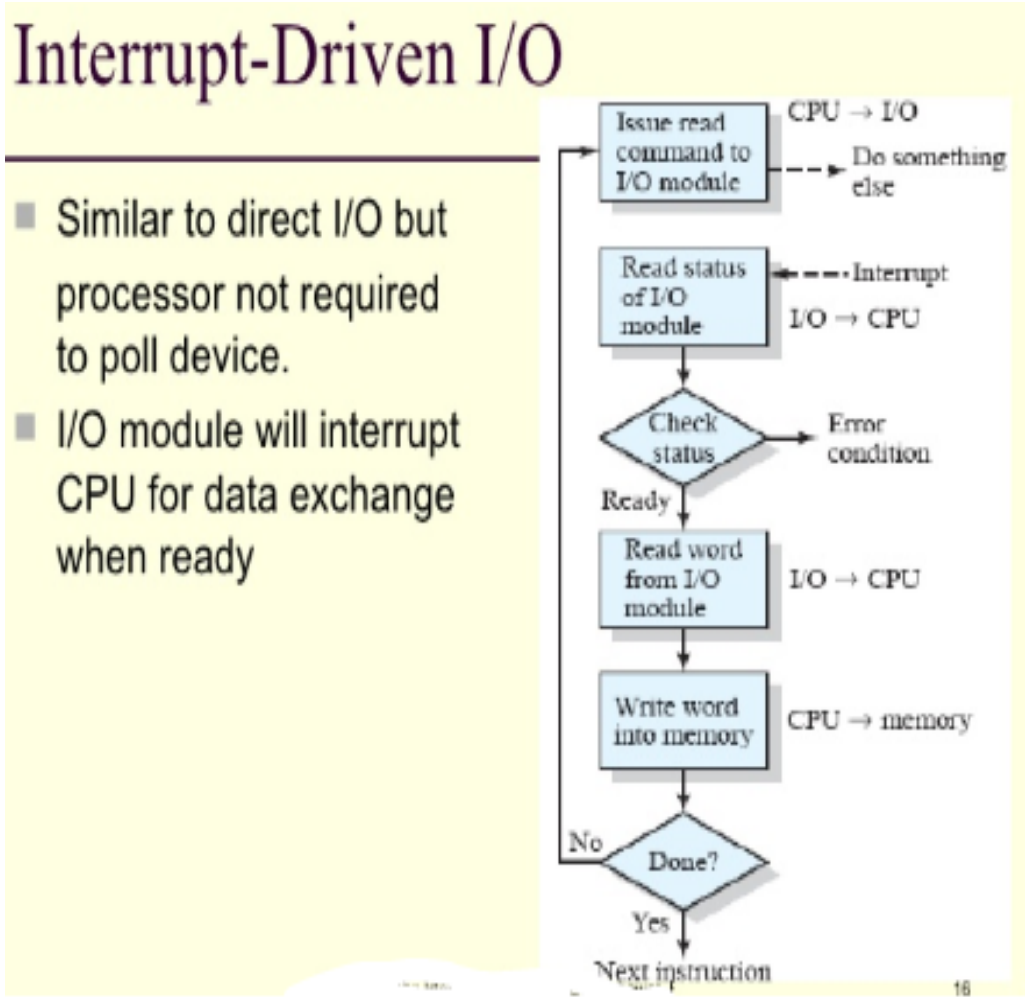
Programmed I/O

- 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



Interrupt - Initiated I/O:

- Interrupt facility an interrupt commands is used to inform the device about the start and end of transfer.
- In the meantime the CPU can proceed for any other program execution
- Whenever it is determined that the device is ready for data transfer it initiates an interrupt request signal to the computer
- Upon detection of an external interrupt signal the CPU stops momentarily the task that it was already performing, branches to the service program to process the I/O transfer, and then return to the task it was originally performing.



Interrupt (1):

- 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:

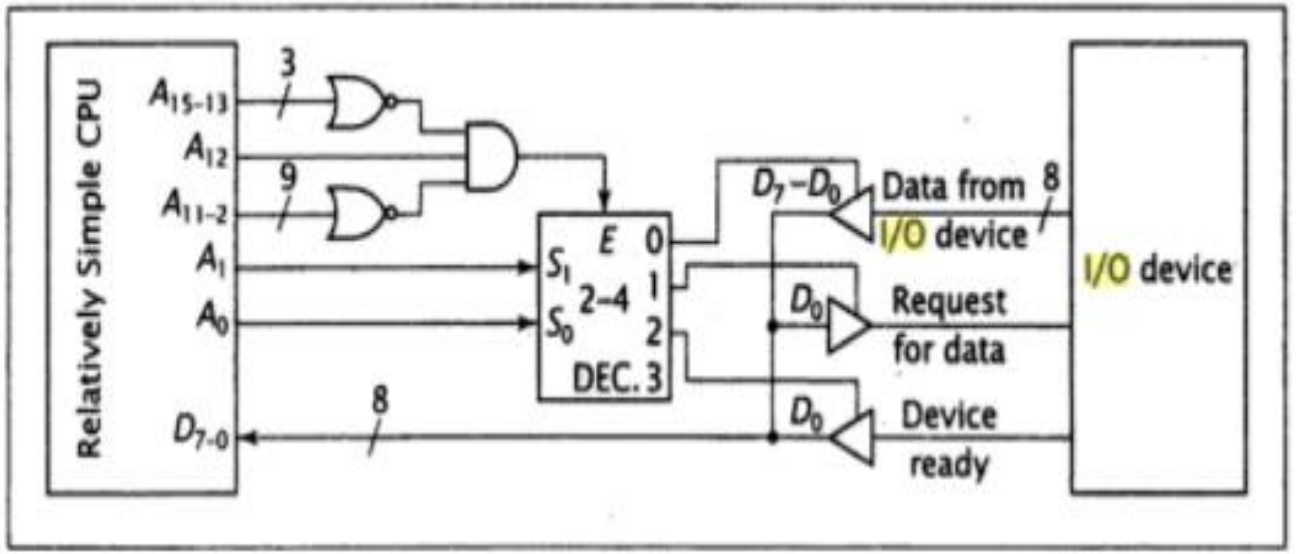
- Method to alleviate the problem of I/O devices with variable delays is called polling.
- In polling, the CPU sends a request to transfer data to an I/O device.
- I/O devices processes the request and sets a device-ready signal when it is ready to transfer.
- The CPU reads signal; if it is high, it 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.

Interrupt (2)

Transferring data between CPU and I/O devices:……

- 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.

Hardware to implement an I/O port that uses polling



Interrupt (3):

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.

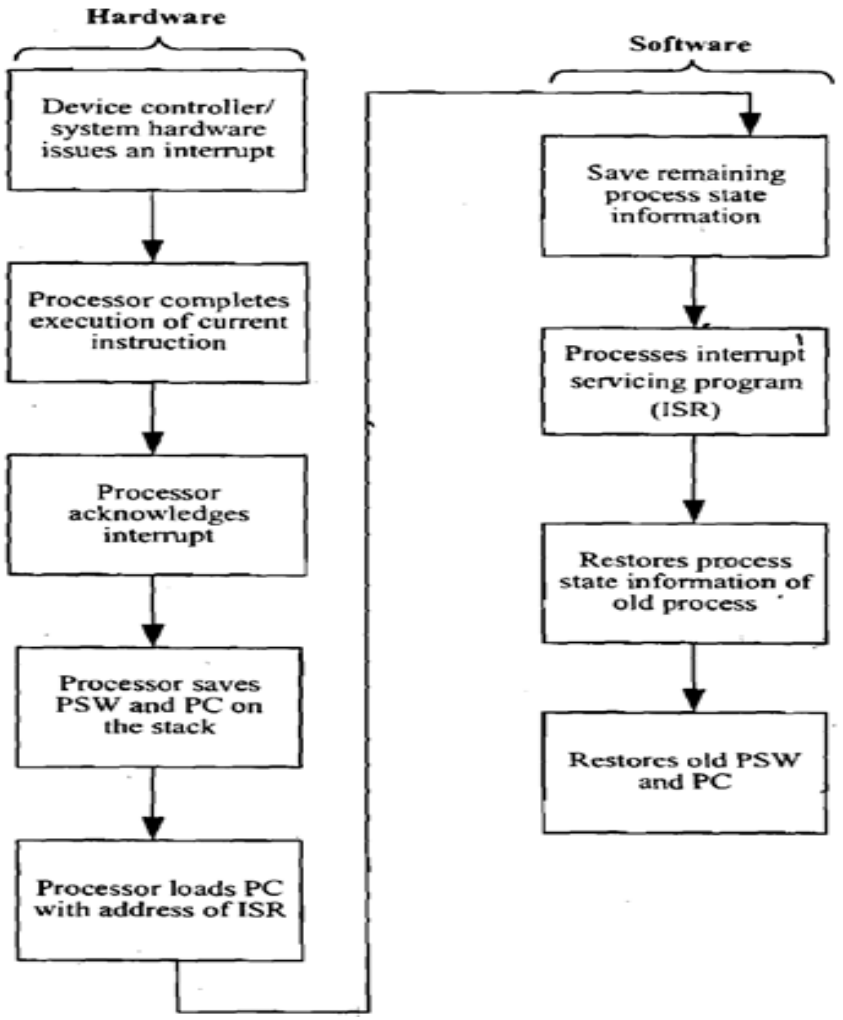
3) Software Interrupt:

- 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 (4):

Interrupt Processing (1):

- The device issues an interrupt signal to processor.
- Processor completes execution of current instruction before responding to interrupt.
- Processor tests for interrupts and sends an acknowledgement signal to device that issued the interrupt.
- The minimum information needed to be stored for task being currently executed before CPU starts executing interrupt routine (using its registers) are:
  - Status of processor that is contained in register known as program status word (PSW), and
  - Location of next instruction to be executed, of currently executing program that is contained in program counter (PC).



Interrupt (5):

Interrupt Processing (2):

- Processor now loads PC with entry location of interrupt-handling program which will respond to this interrupting condition. Once PC has been loaded, processor proceeds to execute next instruction, which is the next instruction cycle that begins with an instruction fetch. Since the instruction fetch is determined by contents of the PC, result is that control is transferred to interrupt-handler program. The execution results in the subsequent operations.
- The contents of processor registers are also needed to be saved on stack which are used by called Interrupt Servicing Routine since these registers may be modified by interrupt-handler.
- Interrupt handler next processes interrupt
- When interrupt processing is finish, saved register values are retrieved from stack and restored to registers
- Final step is to restore values of PSW and PC from stack. Consequently the instruction to be executed will be from previously interrupted program.

Interrupt (6):

Interrupt hardware and priority (1):

Vectored Interrupt:

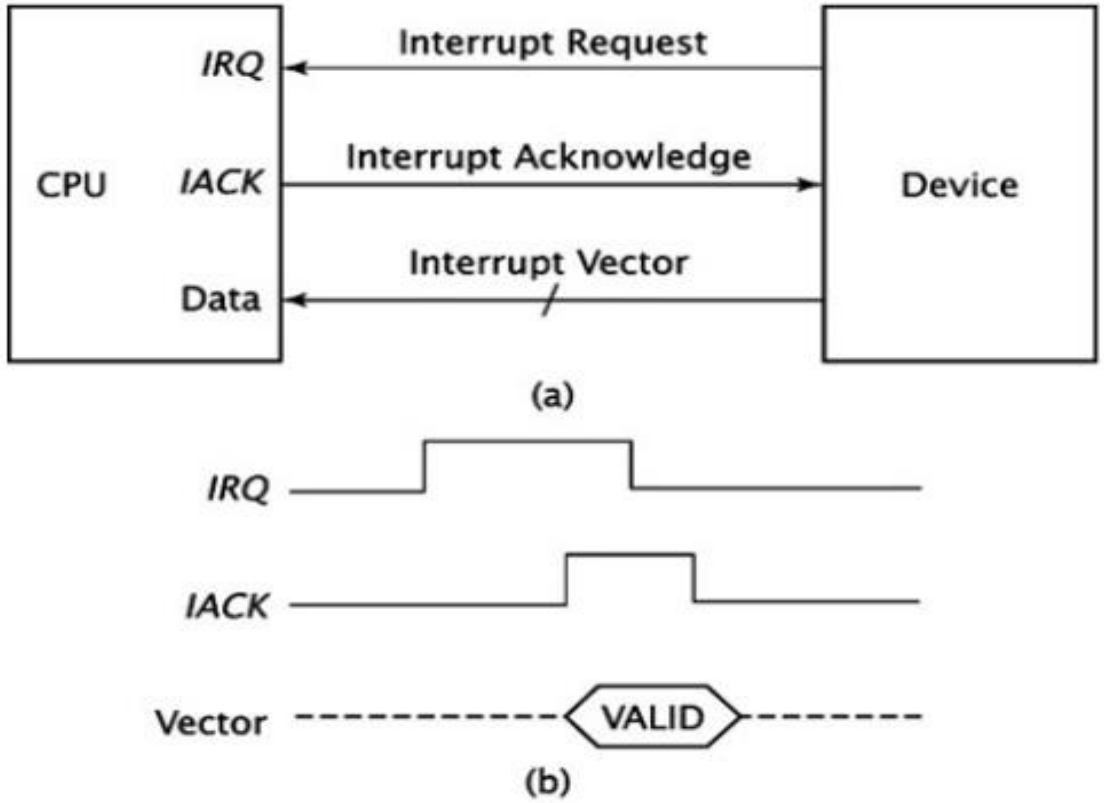


Fig: Hardware and timing for a vectored interrupt for a single device

- 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

Interrupt (7):

Interrupt hardware and priority (2):

Non-Vectored Interrupt:

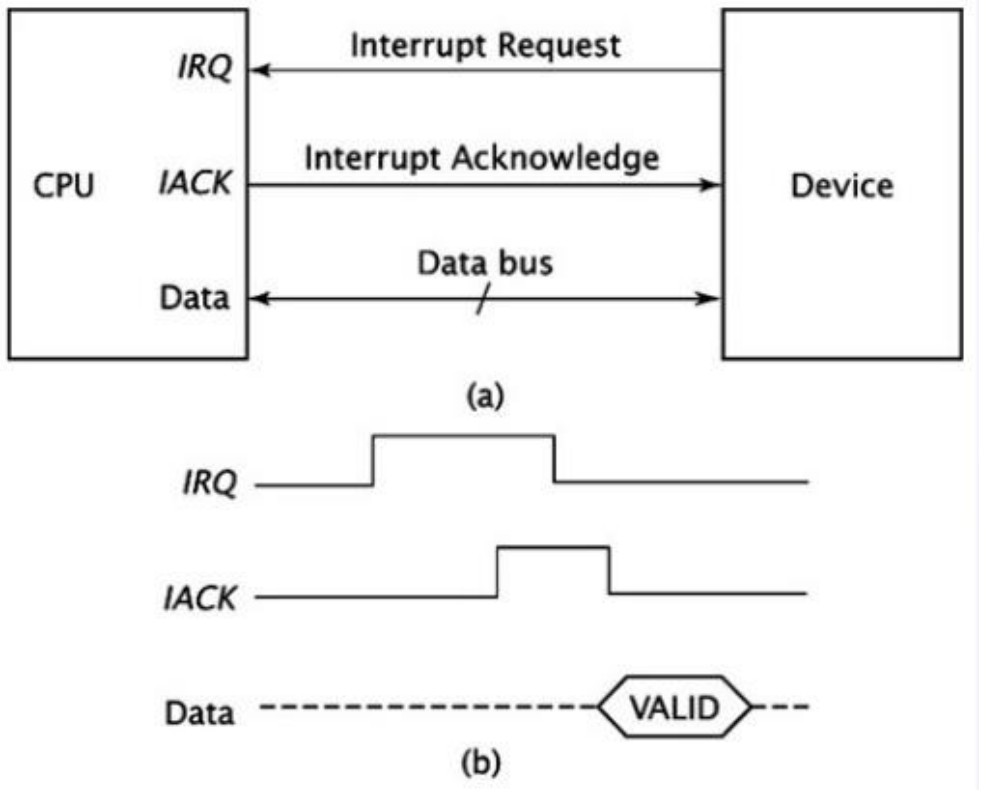


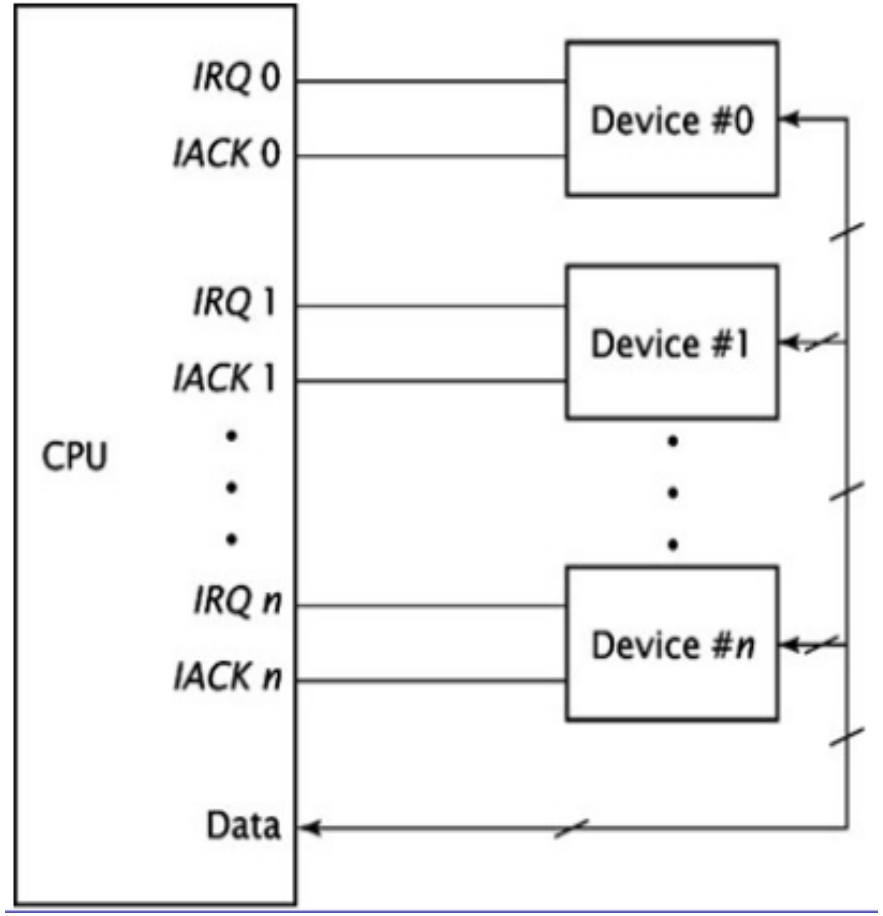
Fig: Hardware and timing for a non- vectored interrupt for a single device

- 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.

Interrupt (8):

Prioritizing multiple interrupts (1):

Extension of Non-Vectored Interrupt:

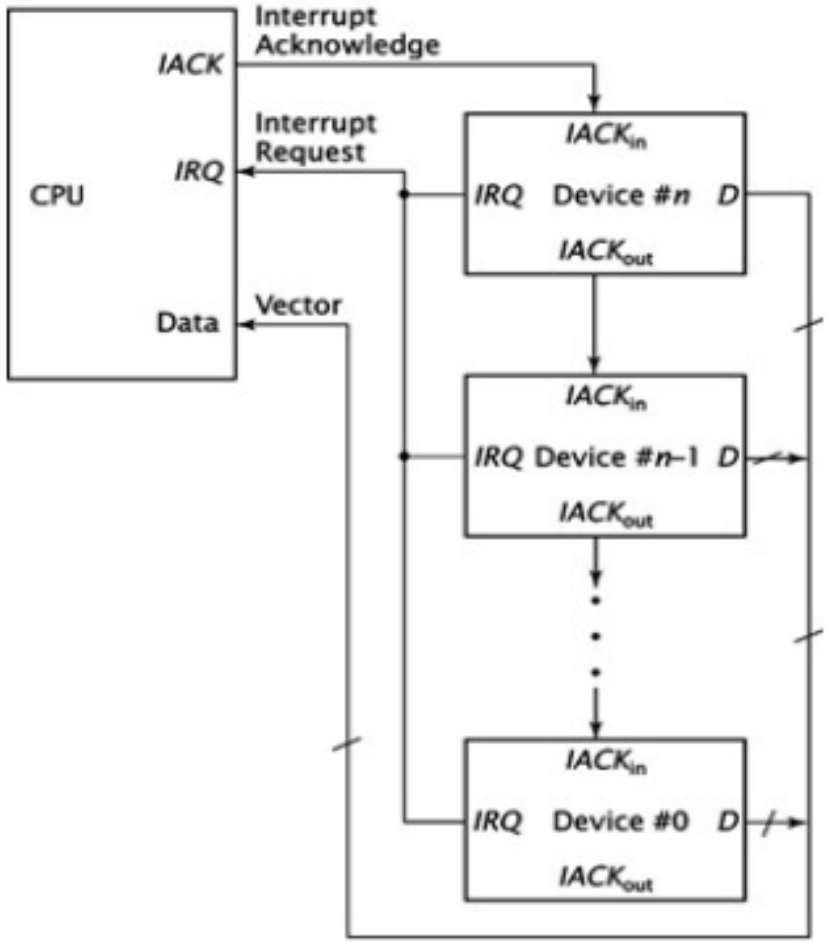


- IRQ_n has the highest priority and IRQ_0 has the lowest.
- If more than one device requests an interrupt, the CPU acknowledges and serves the interrupt with the highest priority first.
- This method works well when there are only a few IRQ/IACK pairs.
- As the number of interrupt increases, the number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

Interrupt (9):

Prioritizing multiple interrupts (2):

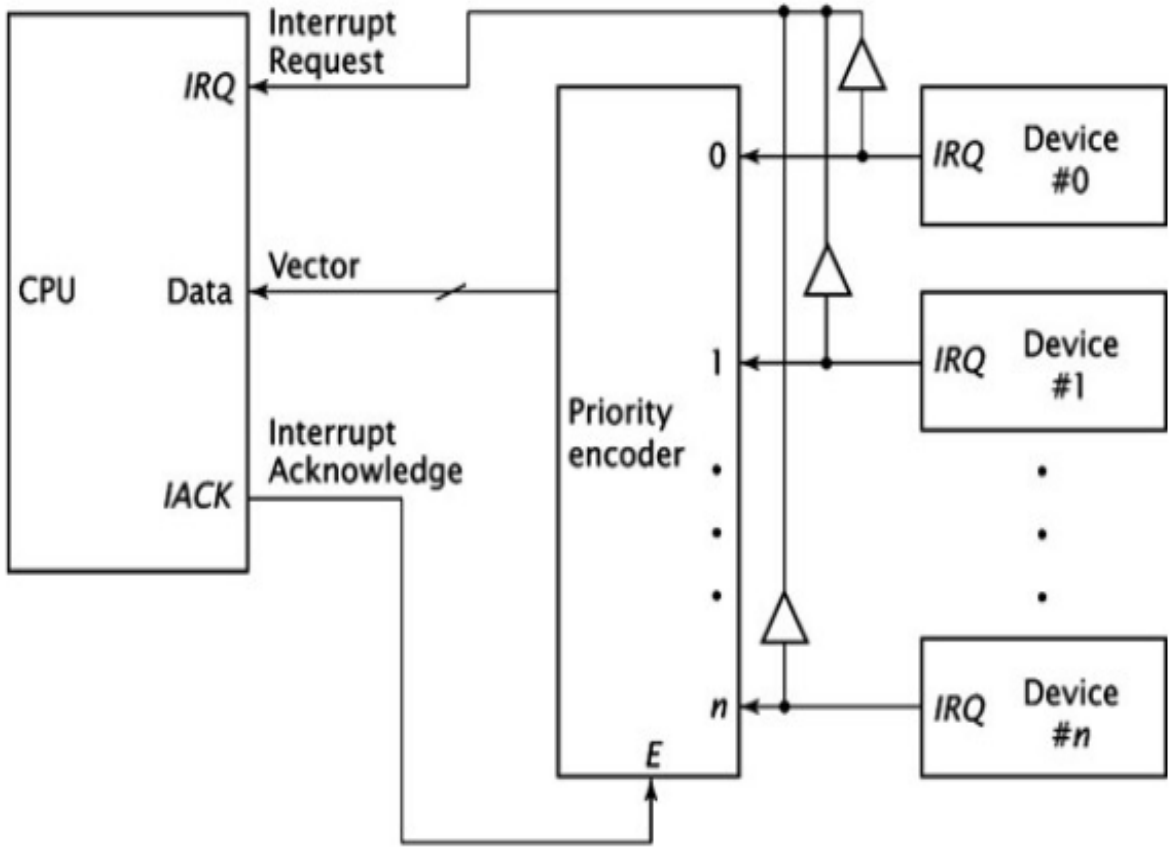
Daisy Chaining:



- 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.

Interrupt (10):

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 (1)

- The data transfer between a fast storage media such as magnetic disk and memory unit is limited by the speed of the CPU.
- Thus we can allow the peripherals directly communicate with each other using the memory buses, removing the intervention of the CPU. This type of data transfer technique is known as DMA or direct memory access
- During DMA the CPU is idle and it has no control over the memory buses.
- The DMA controller takes over the buses to manage the transfer directly between the I/O devices and the memory unit
- **Bus Request:** It is used by the DMA controller to request the CPU to relinquish the control of the buses.
- **Bus Grant:** It is activated by the CPU to inform the external DMA controller that the buses are in high impedance state and the requesting DMA can take control of the buses. Once the DMA has taken the control of the buses it transfers the data.

DMA transfer uses two signal

- HOLD
- HLDA

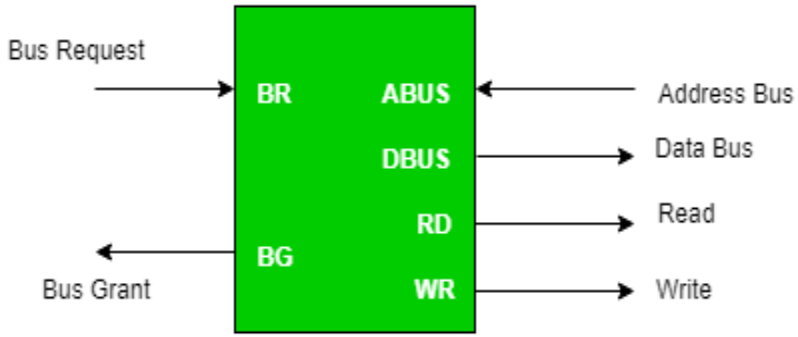
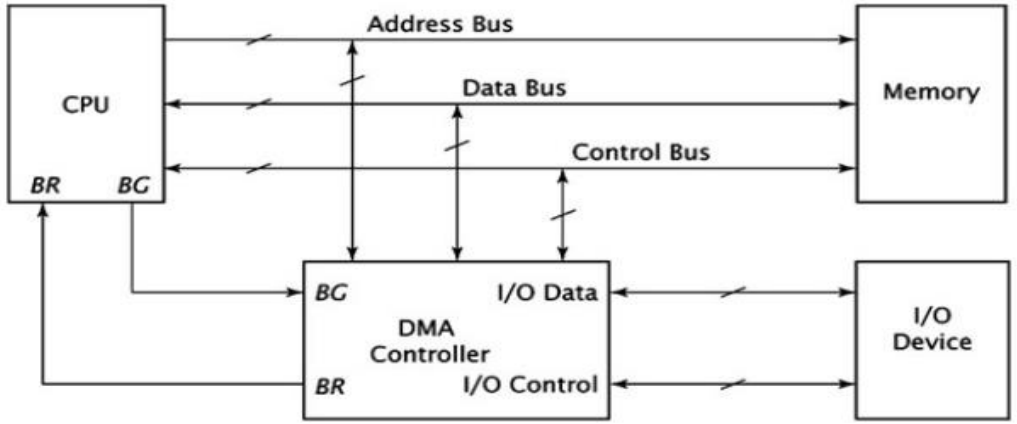
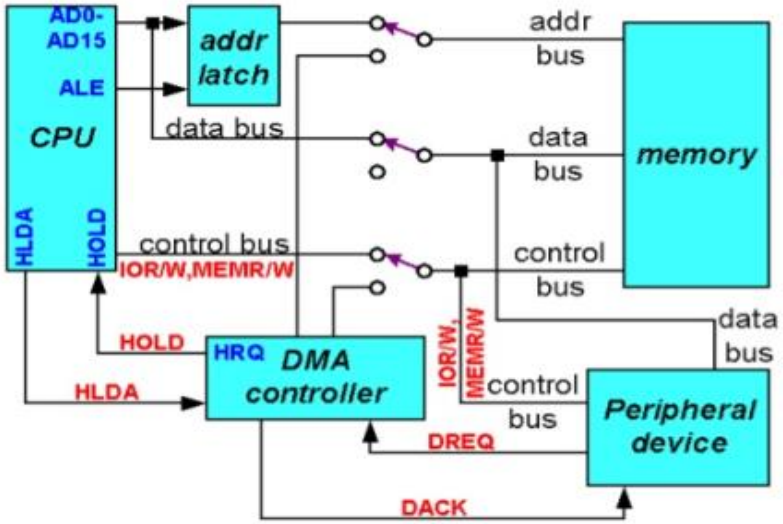


Figure - CPU Bus Signals for DMA Transfer

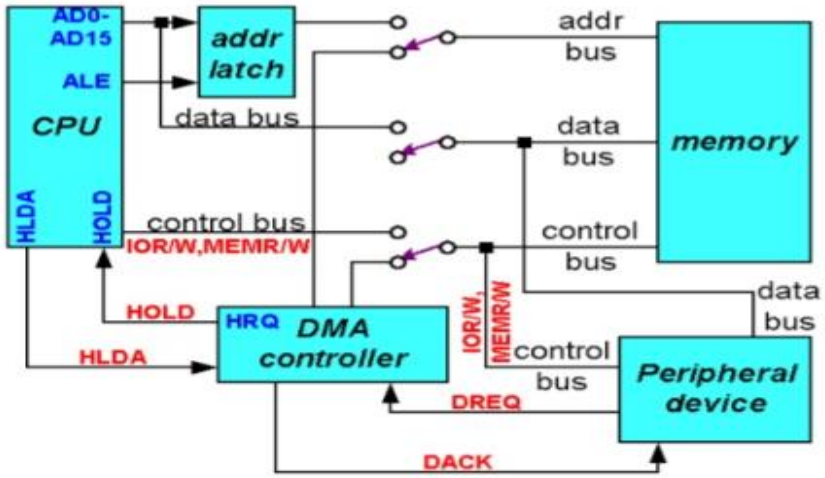
Direct Memory Access (2)



CPU having the control over the bus:



When DMA operates:



Direct Memory Access (3)

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.

Direct Memory Access (4)

DMA transfer modes (1):

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 transfer 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 de-asserts 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.

Direct Memory Access (5)

DMA transfer modes (2):

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 (1)

- 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.

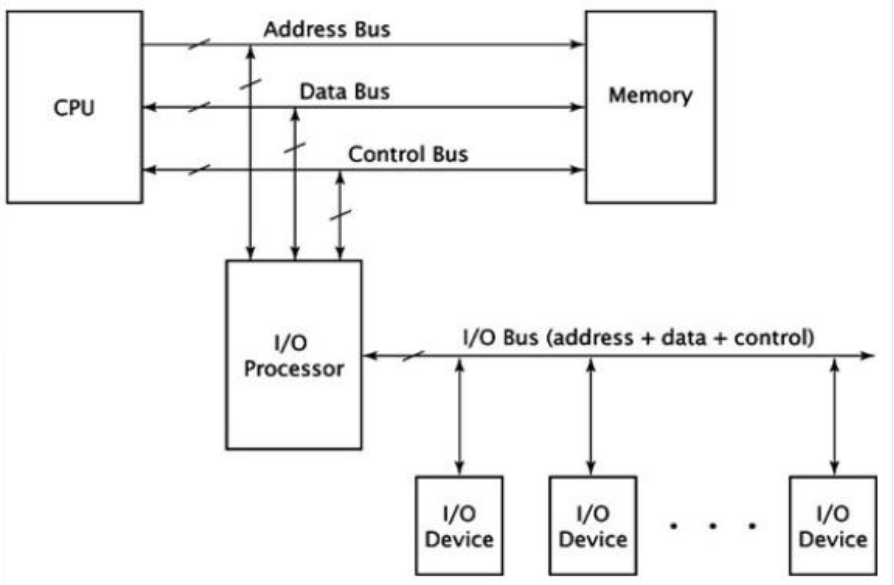


Figure: system configuration incorporating an I/O processor

I/O Processors (2)

- 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 (1)

- 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.

Serial Communication (2)

Basics of serial communication (1)

- 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 = 34.2µs.]

Serial Communication (3)

Basics of serial communication (2)

- It then waits 1 bit time and reads a data bit of the line, repeating this process for 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, $1^1/_2$ or 2 stop bits are used.

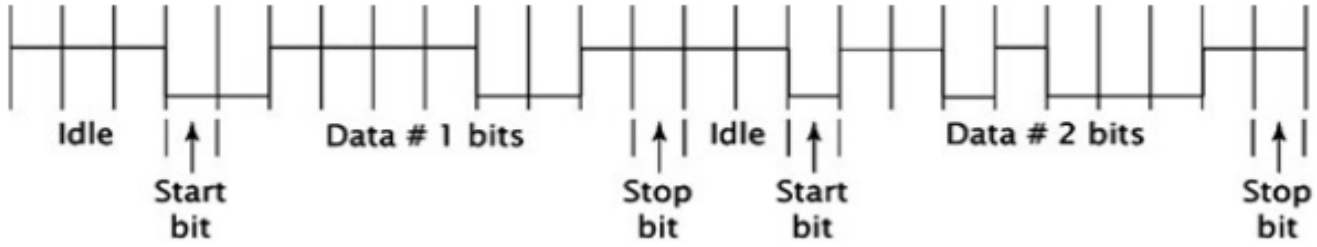


Fig: sample transmission of two byte of data

Serial Communication Standards (RS-232-C Standards) (1)

- RS-232C is an interface developed to standardize the interface between data terminal equipment (DTE) and data communication equipment (DCE) employing serial binary data exchange.
- Modem and other devices used to send serial data are called data communication equipment (DCE).
- The computers or terminals that are sending or receiving the data are called data terminal equipment (DTE).
- It specifies that DTE connector should be male and DCE connector should be female.
- It can send 20kBd for a distance of 50 ft.
- The voltage level for RS-232 are:

  A logic high or 1 or mark, -3V to -15V

  A logic low or 0 or space, +3v to +15v

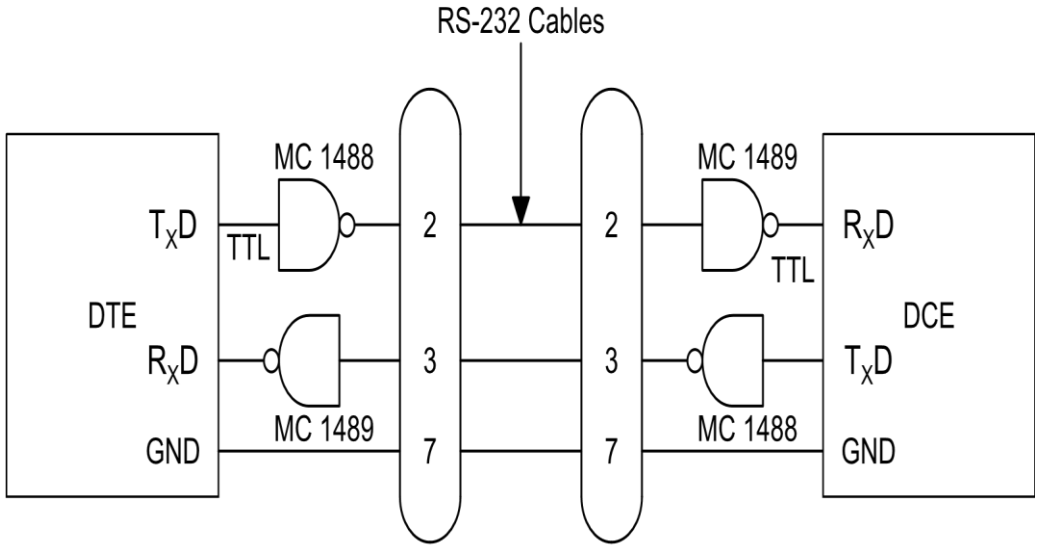


Fig: Connection of DTE and DCE through RS-232C Interface

- Mc1488 line driver converts logic 1 to -9V

Serial Communication Standards (RS-232-C Standards) (2)

RS- 232 signals used in handshaking:

Data Terminal Ready (DTR):

- After the terminal power is turned on and terminal runs any self-checks, it asserts data terminal ready (DTR') signal to tell the modem that it is ready.

Data Set Ready (DSR):

- When the MODEM is powered up and ready to transmit or receive data, it will assert data set ready (DSR') to the terminal. Under manual control or terminal control, modem then dials up the computer. If the computer is available, it will send back a specified tone.

Request to send (RTS):

- When a terminal has a character ready to send, it will assert a request-to-send (RTS') signal to the modem.

Data Carrier Detect (DCD):

- The modem will then assert its data-carrier-detect (DCD') signal to the terminal to indicate that it has established connection with the computer.

Clear to send (CTS):

- When the modem is fully ready to receive data, it asserts the clear-to-send (CTS') signal back to the terminal.

Ring indicator (RI):

- It indicates that a ring has occurred at modem. Deactivating DTR or DSR breaks the connection but RI works independently of DTR i.e. a modem may activate RI signal even if DTR is not active.

Transmitted Data (TxD):

- The terminal then sends serial data characters to the modem.

Received Data (RxD):

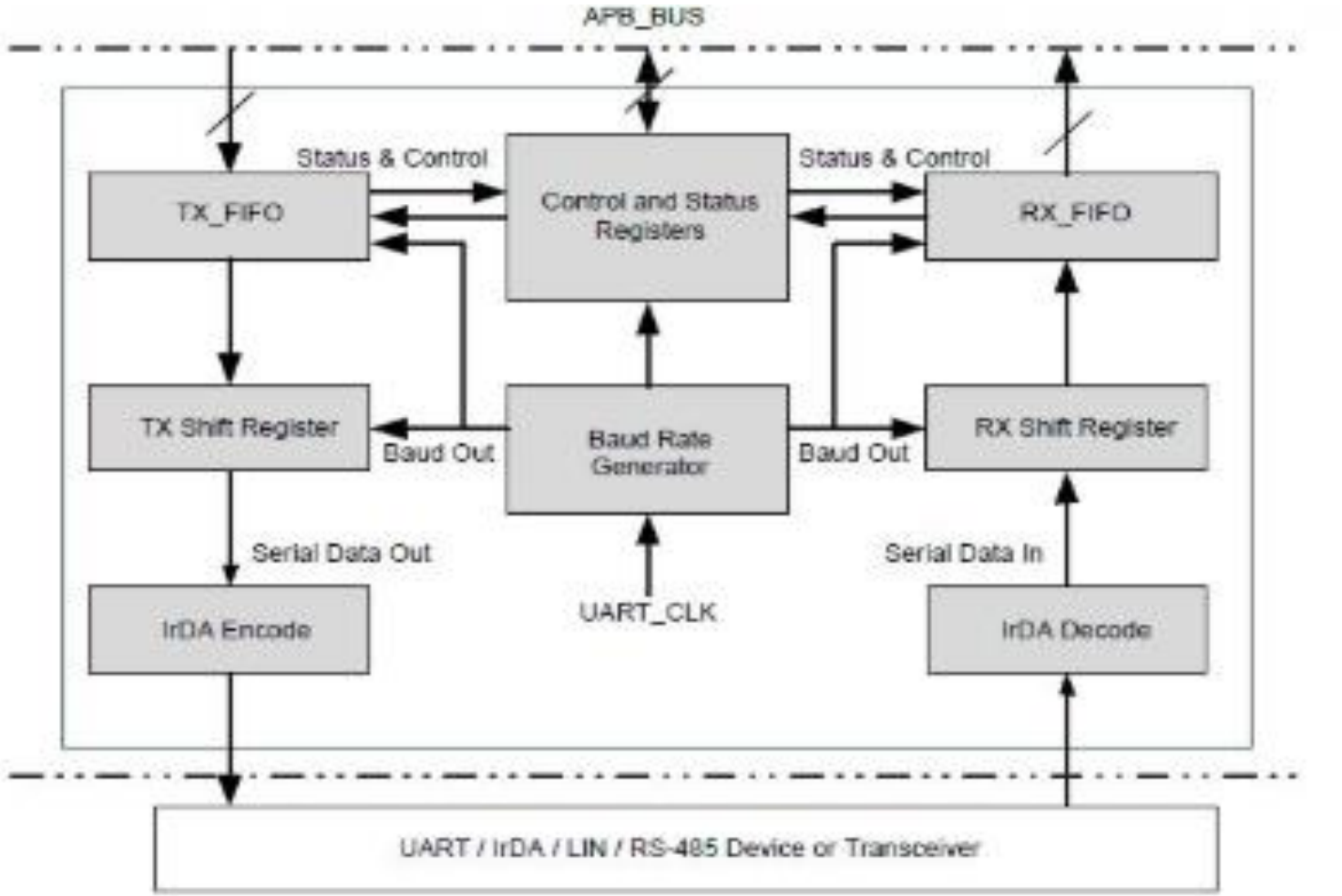
- Modem will receive data from terminal through this line.

Data Signal Rate Detect (DSRD):

- It is used for switching different baud rate.

Universal Asynchronous Receiver/ Transmitter (UART) (1)

- A UART (Universal Asynchronous Receiver/Transmitter) is the microchip with programming that controls a computer's interface to its attached serial devices



Universal Asynchronous Receiver/ Transmitter (UART) (2)

- Asynchronous serial communication is a common function in computer system.
- As with many popular functions, manufacturers have designed special chips to perform these functions and relieve the CPU of this task.
- For asynchronous serial communication, these specialized chips are called universal asynchronous receiver/ transmitter, or UARTs.

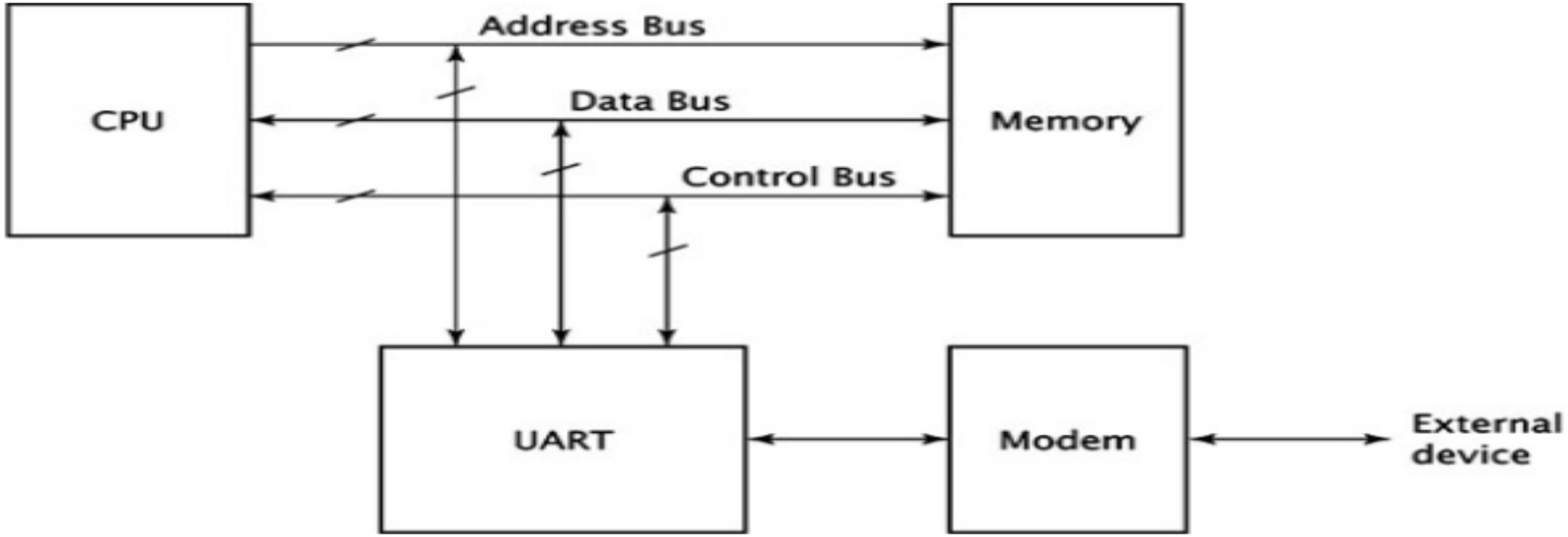


fig: A computer system incorporating a UART

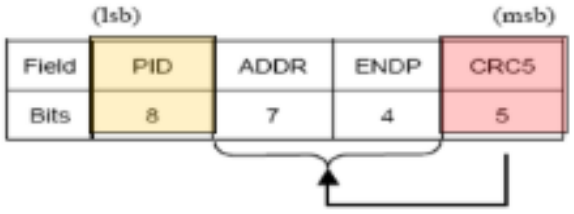
Universal Asynchronous Receiver/ Transmitter (UART) (3)

- 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).

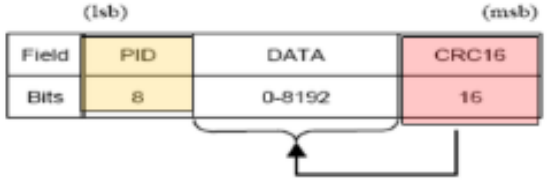
USB standards (1)

- USB stands for Universal Serial Bus
- Provides an expandable, fast, bi-directional, low-cost, hot pluggable, plug and play serial hardware interface
- Transmits data in packets
- The USB standard specifies four types of packets, which are used to communicate between a computer and its USB peripheral
- **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.

Token



Data

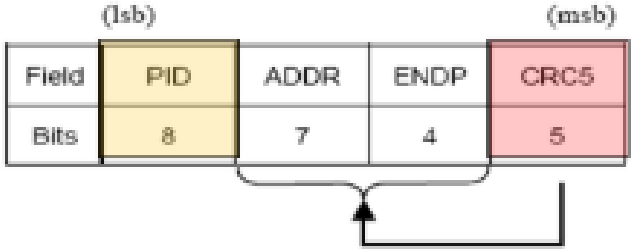


Status (Handshake)

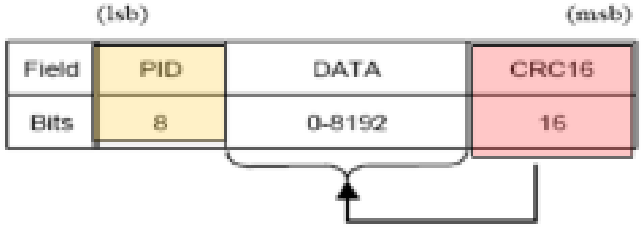
USB standards (2)

- The token packet is used to initiate data transfers.
- It specifies **address (ADDR)** of the device to send or receive data, and direction of data flow; the direction is specified in the **packet identifier (PID).** The other two fields indicate the **end of packet (ENDP)** and a **cyclic redundancy check (CRC).**
- After sending token packet to initiate transfers, the computer sends a data packet.
- Each data packet contains a **PID**, a data field and a 16-bit **CRC** field.
- After receiving this data ,the receiving device sends a handshake packet.
- This packet contains only a 8-bit **PID**.

Token



Data



Status (Handshake)

ASSIGNMENT QUESTIONS: (from old question set):

1. *What is DMA? Describe how they work with proper diagram.*
2. *What is handshaking? How are asynchronous data transfer done in CPU? Explain.*
3. *What are I/O processors? Describe how they work using suitable diagram.*
4. *How DMA controller can be incorporated in a computer system?*
5. *Differentiate serial and parallel communication? Describe 4 different modes of asynchronous data transfer.*
6. *What is an interrupt? How is an interrupt serviced? Explain software action in interrupt handling.*
7. *Define DMA? Explain DMA transfer Mode.* *Hints:Q5*
8. *Differentiate between:*
   1. *Interrupt driven I/O and Programmed I/O*
   2. *Vectored and Non vectored interrupt Hardware*
9. *Write short Notes:*
   1. *Programmed I/O*
   2. *UARTs internal configurations*
   3. *Interrupt Vector*
   4. *Interrupts and Handling Interrupts*
   5. *RS 232C standards*

Sr. No.	Factor	Serial	Parallel
1.	Number of bits transmitted at one clock pulse	One bit	*n* bits
2.	No. of lines required to transmit *n* bits	One line	*n* lines
3.	Speed of data transfer	Slow	Fast
4.	Cost of transmission	Low as one line is required	Higher as *n* lines are required.
5.	Application	Long distance communication between two computers	Short distance communication. like computer to printer.

CHAPTER COMPLETED