Unit-5 Input/output (I/O)

Peripheral Devices :⇒

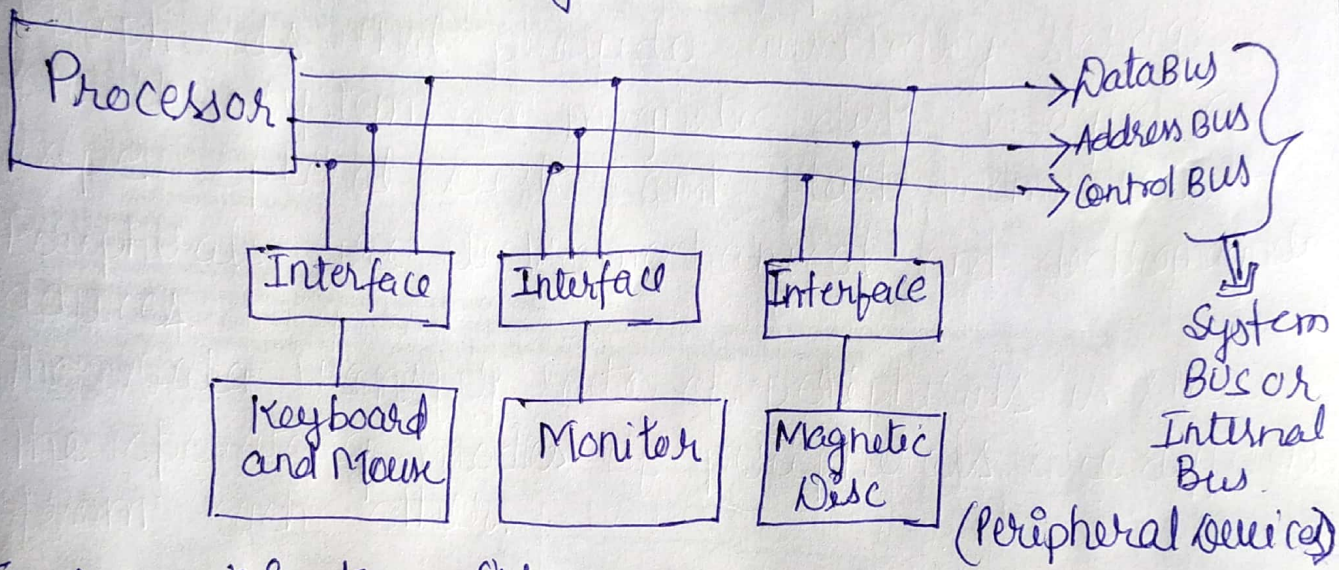
⇒ Input or output devices attached to the computer are also called peripheral devices
⇒ Among the most common peripherals are Keyboard, mouse, Monitor, printer, scanner.
⇒ Peripherals that provide auxiliary storage for the system are magnetic disk, magnetic tape, optical disk and flash memories
⇒ Peripherals are electromechanical and electromagnetic devices
⇒ The Data transfer rate of peripherals is slow.
⇒ The operating modes of peripherals are different from each other.

Input-output Interface :⇒

⇒ A hardware unit that plays an important role between the CPU and I/O devices for input and output transfers is known I/O interface
⇒ The need of I/O interface arieses because the differences between CPU and I/O devices are:
(i) Peripheral are electromechanical and electromagnetic devices but CPU and memory are electronic devices Therefore, conversion of signal values may be required.
(ii) The data transfer rate of peripherals is slower than the CPU. So a synchronization is needed
(iii) Data codes and format in peripherals differ from the word format in the CPU and memory.

⇒ The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

⇒ To resolve these differences Input/output (I/O) interface or I/O controller is included with each and every peripheral device.



⇒ To communicate with a particular device, the processor places a device address on the address bus.

⇒ When the interface detects its own device address it activates the path between the Bus line and device. At the same time processor provides a function code in the control line also called I/O commands.

⇒ There are four types of I/O commands that the interface may receive.

(i) Control command ⇒ It activate the peripheral device and inform it what to do.

(ii) Status command ⇒ used to test various status condition like peripheral is busy.

(iii) Output data command ⇒ It causes the interface to respond by transferring data from the Bus into its Register.

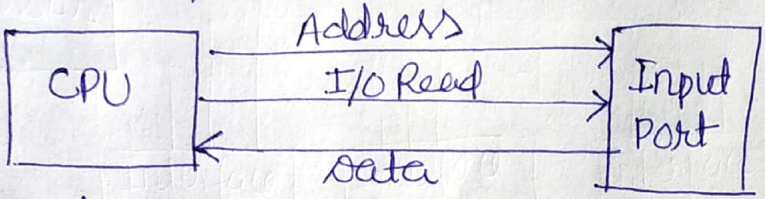
(iv) Input data command ⇒ Interface receive data from I/O device and place it in it's buffer register.

I/O Ports ⇒ An I/O port is a program addressable hardware unit through which the CPU can transfer information. Basically these are registers.

⇒ Different ports are alloted distinct addresses which are used while communicating with them.

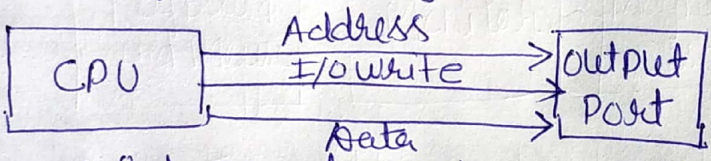
⇒ An input port supplies data to the bus whereas an output port accepts data from the bus.

⇒ An input port gives data when it receives its address and I/O Read signal.



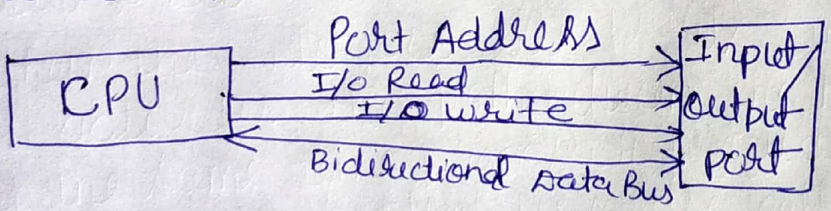
Reading from an input port

⇒ An output port receives data when it receives its address and I/O write signal.



Writing to an output port

⇒ A port which sends data as well as accept data is known as an input/output port (I/O port). with same address.



~~Memory Mapp~~

Isolated I/O versus Memory Mapped I/O

⇒ As a CPU needs to communicate with the various memory and input-output devices (I/O).

⇒ Data between the processor and these devices flow with the help of system Bus. There are three ways in which system Bus can be allotted to them:

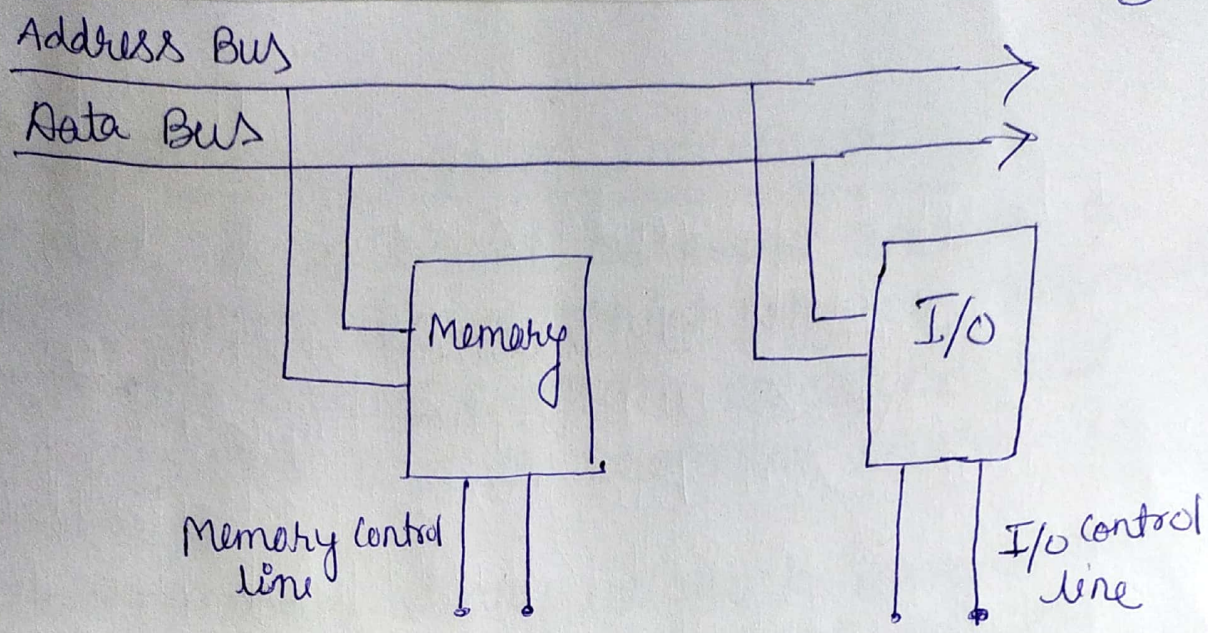
1. Separate set of address, control and data bus to I/O and memory.
2. Have common bus (data and address) for I/O and memory but separate control lines.
3. Have common bus (data, address and control) for I/O and memory.

In first case it is simple because both have different set of address space and instruction but require more buses.

Isolated I/O ⇒ In Isolated I/O common bus is only for (data and address) for I/O and memory but separate read and write control lines for I/O.

⇒ When CPU decode instruction then if data is for I/O then it places the address on the address line and set I/O read or write control line on due to which data transfer occurs between CPU and I/O.

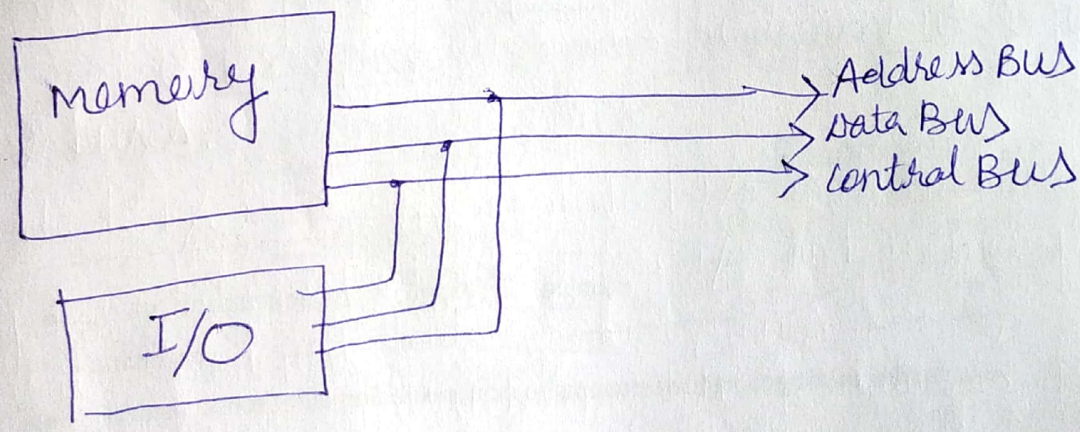
⇒ In it different read/write instruction for both I/O and memory.



Memory Mapped I/O ⇒

⇒ In this case every bus is common due to which the same set of instructions work for memory and I/O.

⇒ Here we manipulate I/O same as memory and both have same address space, due to which addressing capability of memory become less ~~because~~ because some part is occupied by the I/O.

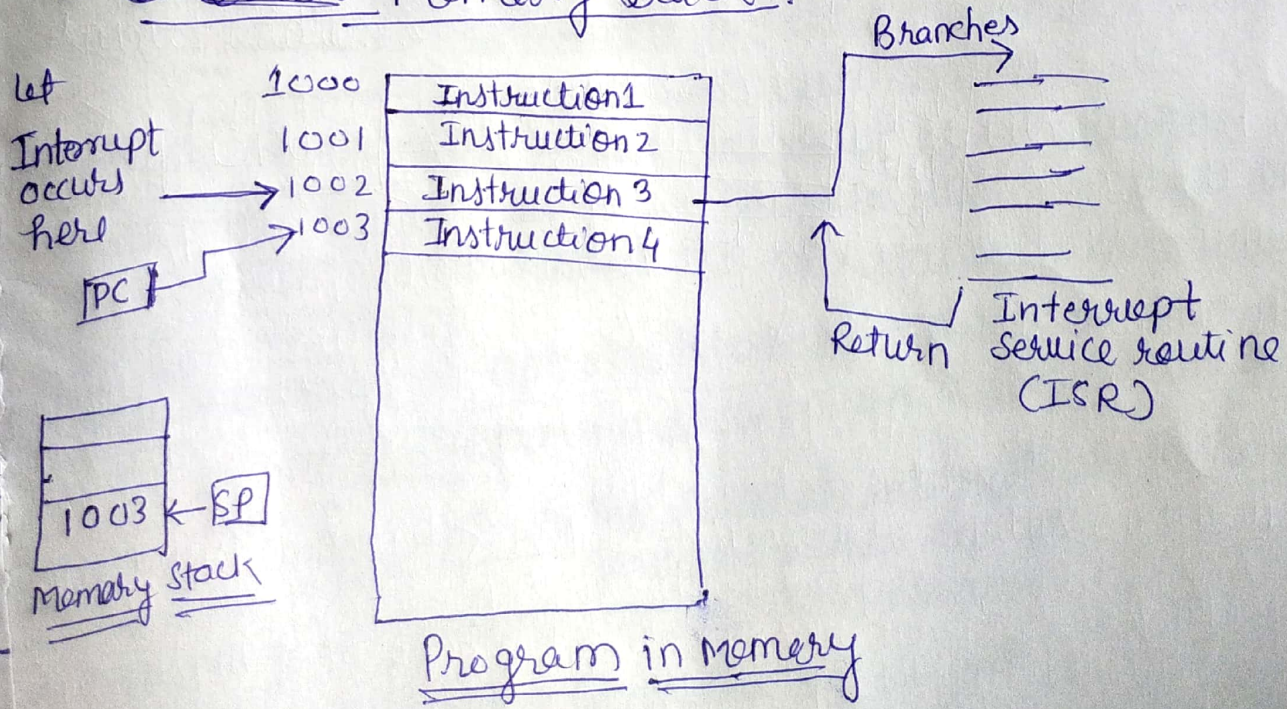


⇒ Interrupt or Program Interrupt

⇒ Interrupt is a signal or event inside a computer system, due to which the CPU temporarily suspends the current program execution and starts executing another program related to the interrupt.

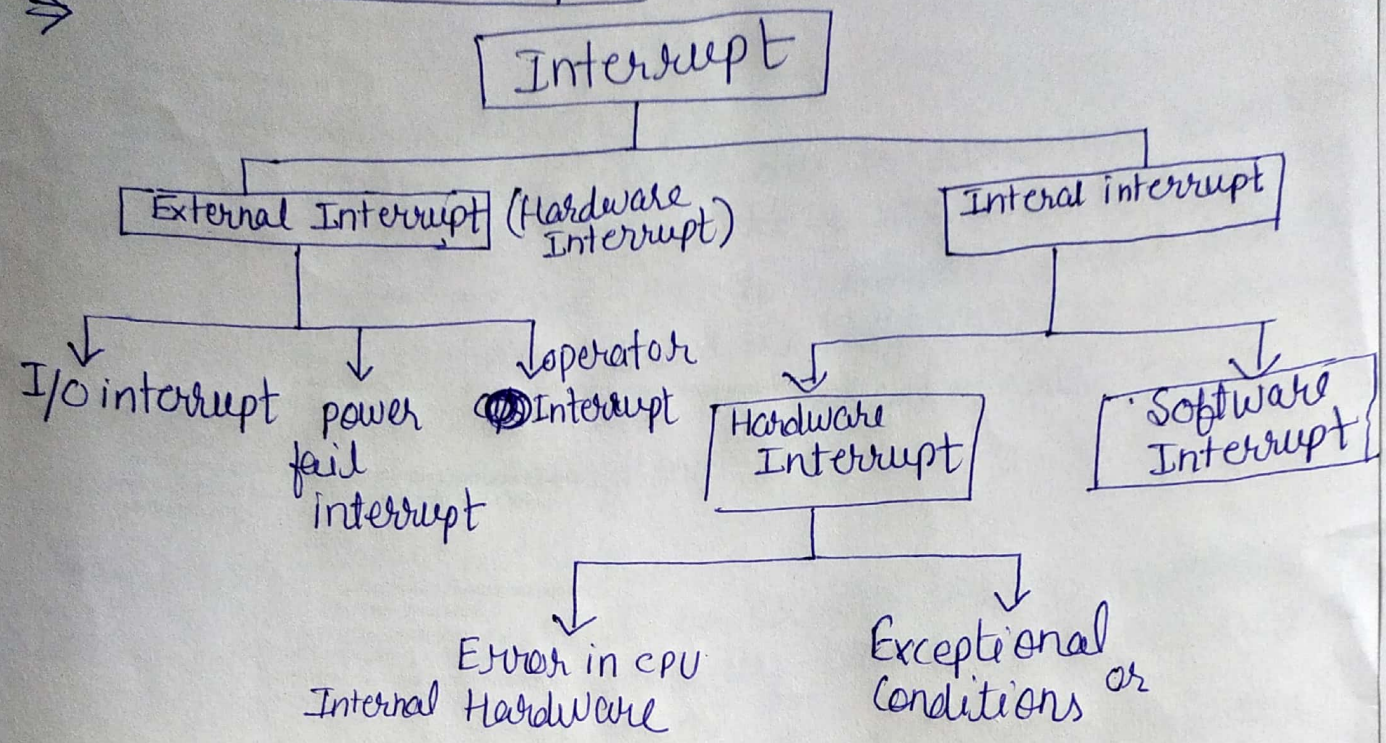
⇒ The CPU's activity at any instant is instruction processing. on receiving an interrupt, the CPU should execute another program which will service the interrupt. this program is known as Interrupt service routine (ISR).

⇒ On completing the ISR, the CPU should usually, continue previously interrupted program from the exact place where it left on receiving the interrupt Hence before taking ISR, the CPU stores the next instruction address from program counter (PC) to ~~Stack~~ Memory stack.



Types of Interrupt

⇒



External Interrupt ⇒ The external interrupts are generated by hardware circuits outside the CPU.
⇒ These are basically hardware interrupts generated by the circuits. These are:

ⓐ I/O interrupt :- Generated by an I/O interface in order to indicate to the CPU that it is ready for data transfer with the CPU.

ⓑ Power fail interrupt :- This interrupt is an advance information that power is going to fail in a short while. The ISR can in turn store the CPU status in some battery backed memory.

ⓒ ~~operator interrupt~~ ⇒ operator interrupt :- This is an interrupt to the operating system in order to perform a specific action

Internal Interrupt ⇒ The internal interrupts are the interrupts generated within the CPU.
⇒ These may be hardware or software interrupt.

Hardware Interrupt ⇒ Generated by circuit.

Error in CPU internal Hardware :

⇒ This interrupt is a result of detection of some hardware malfunction such as data failure within the CPU.

Exceptional condition :

⇒ These interrupts are caused due to abnormal conditions encountered by the CPU during the execution of the program. These are not due to hardware errors but due to special situations caused by programs. The various program exceptional cases are illegal opcode, stack overflow, instruction format violation.

Software Interrupt :

⇒ The software interrupt is created by the program so that the CPU will temporarily branch from the current program to another program.

⇒ INT (interrupt) instruction is used for generating the interrupt.

⇒ CPU access the corresponding location in memory in which the start address of ISR (subprogram) is stored.

⇒ Switching from a CPU user mode to the supervisor mode or system mode (when CPU executing a program that is part of the operating system). The CPU is normally in the user mode when executing user programs.

Modes of Transfer

"The ways by which the data can be transferred from I/O devices to the memory is known as Modes of transfer".

⇒ There are three types of Modes of transfer:

(i). Programmed I/O
(ii). Interrupt initiated I/O
(iii). Direct memory access (DMA)

* (i). Programmed I/O ⇒ In this mode of data transfer the input-output instructions are written in the form of program.
⇒ The instruction ~~written~~ written in the program basically initiate data transfer to and from a CPU register and I/O devices.
⇒ Programmed I/O requires that all I/O operations be executed under the direct control of the CPU.
⇒ CPU stays in program loop until the I/O device indicate that it is ready for data transfer.
⇒ This is time-consuming process and keeps the processor busy

(ii) ~~Interrupt~~ Interrupt Initiated I/O ⇒
⇒ In this mode of transfer CPU concentrate on some other program.
⇒ This method use the interrupt signal to inform the CPU that the data are available from I/O device and I/O flag is set to 1.
⇒ When flag is set the CPU deactivates its task to take care of I/O transfer
⇒ After the transfer has been completed, the CPU returns to continue the previous program.
⇒ Interrupt initiated I/O can be handle by priority interrupt
⇒ There are two types of priority interrupt.
① Serial ~~Pro~~ Priority interrupt or Daisy-chaining priority interrupt
② Parallel Priority interrupt.

(3) Direct Memory access (DMA)

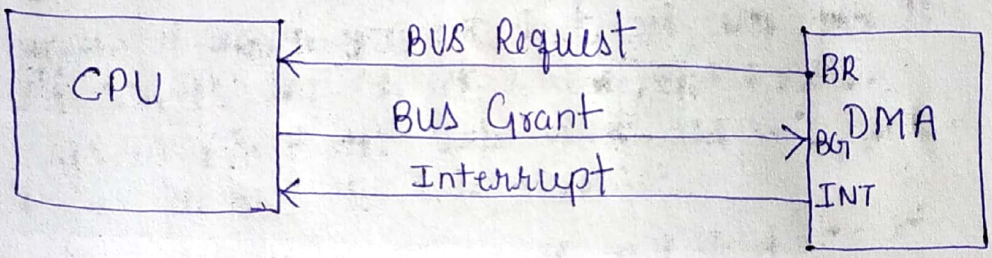
⇒ Direct memory access is a technique for moving data directly between the main memory and the I/O devices without the intervention of CPU.

⇒ The CPU initiates the transfer. It supply the starting address and number of words to be transferred to the DMA and then busy with other tasks.

⇒ The transfer can be done in two ways;

Burst transfer :- A block sequence consisting of a number of memory words can be transferred into a continuous burst called Burst transfer.

⇒ Cycle Stealing :- In this transfer technique one data word transfer at a time, after which it must return control of buses to the CPU.



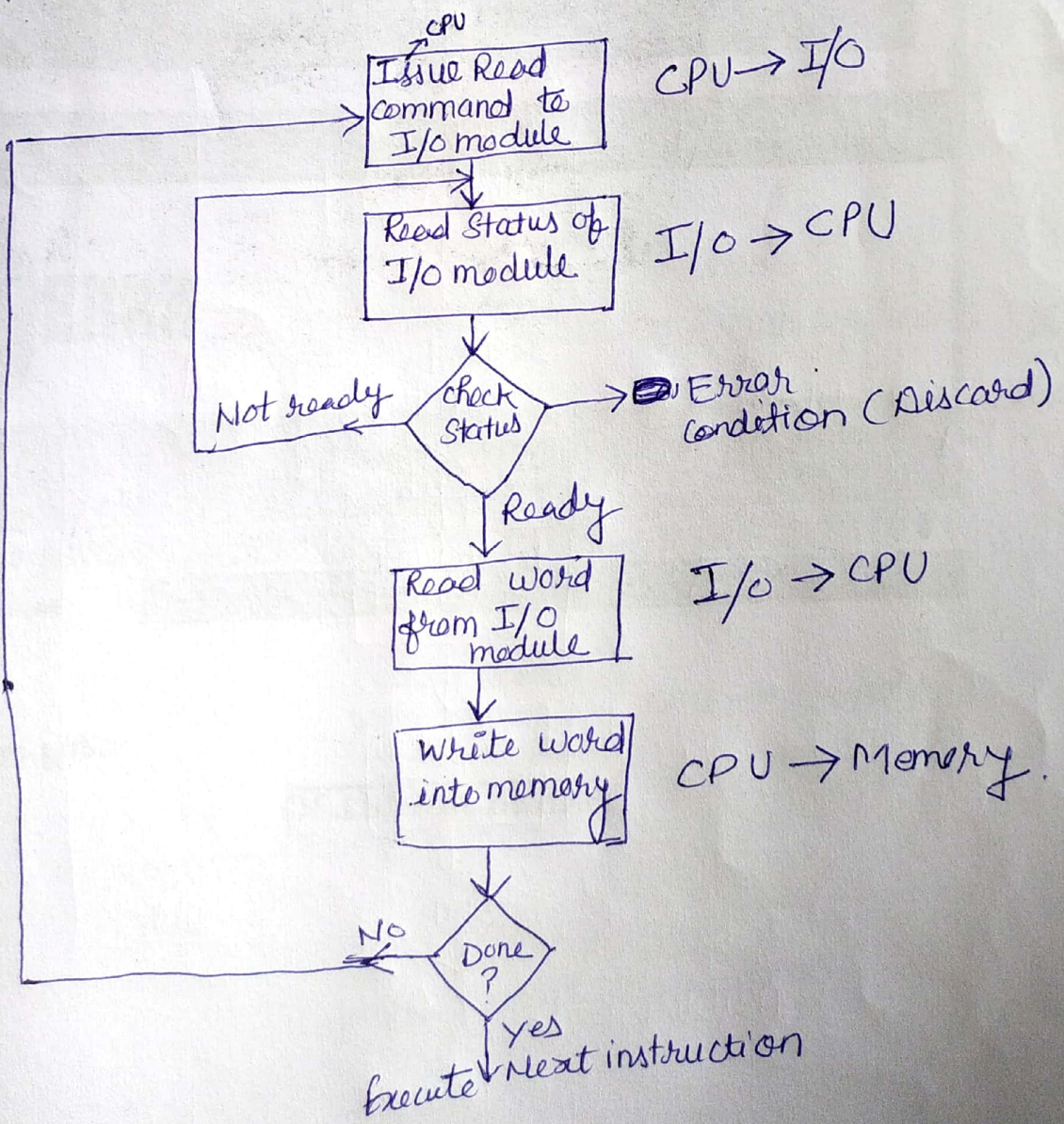
Difference between Programmed I/O and Interrupt initiated I/O ⇒

Programmed I/O ⇒

⇒ The instruction written in the form of the program basically initiate data transfer to and from a CPU register and I/O devices.

⇒ It requires all I/O operations be executed under the direct control of the CPU. CPU stays in program loop until the I/O device indicate that it is ready for data transfer.

⇒ This is time consuming process and keeps the processor busy.



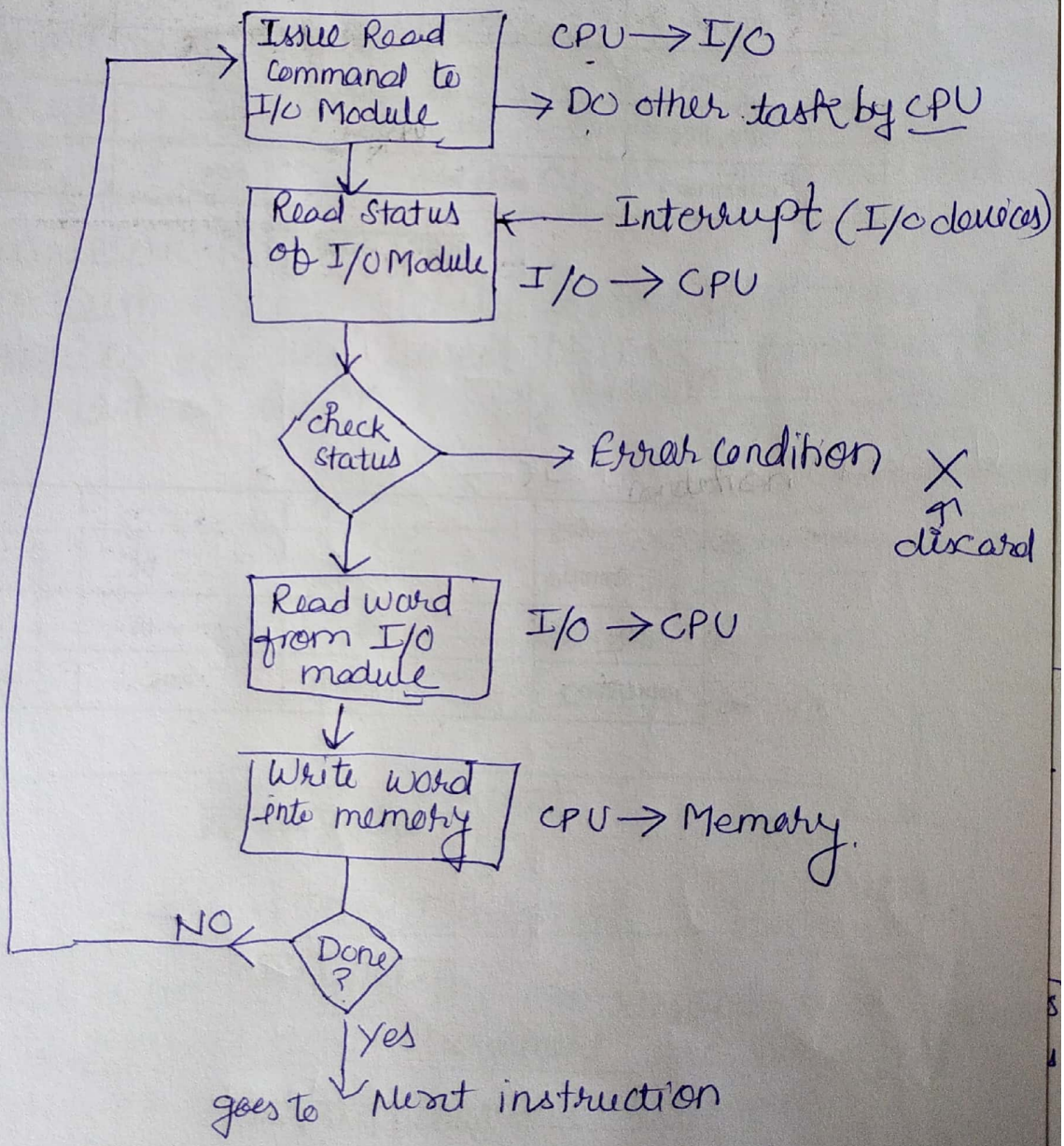
Interrupt initiated I/O

⇒ In this method CPU concentrate on execution of some other programs.

⇒ Interrupt signal is used to inform the CPU that the data are available from I/O device.

⇒ After the transfer has been completed, the CPU returns to continue the previous program.

⇒ Interrupt initiated I/O can be handle by pricrity interrupt.



Priority Interrupt

⇒ In Priority interrupt when two devices interrupt the CPU at the same time, the device with highest priority is serviced first.

⇒ Devices with high-speed transfers such as magnetic disks are given high priority and devices with slow speed transfers such as keyboard are given low priority

⇒ There are two types of Priority interrupt

(I). Serial priority interrupt or Daisy chaining Priority

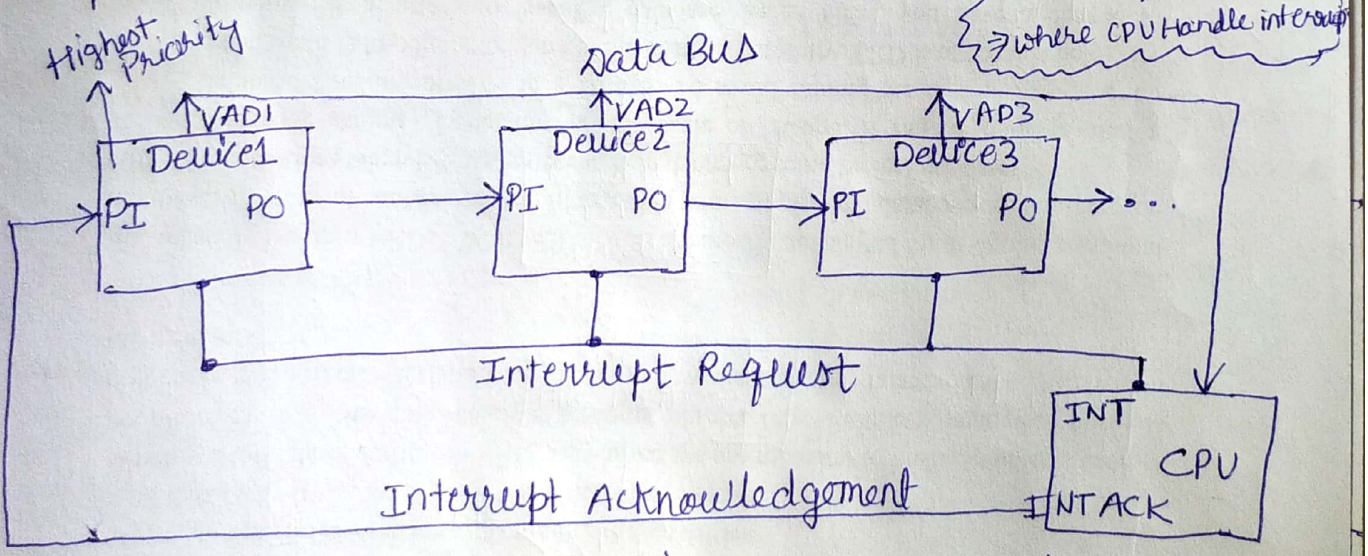
(II) Parallel priority interrupt.

(I) Serial Periority or Daisy chaining Priority interrupt

⇒ It consist of a serial connection of all devices that request an interrupt.

⇒ The devices with highest priority is place first, then lower priority devices upto the lowest priority device which is placed at last as shown in fig below

VAD → Vectered Address
⇒ where CPU Handle interrupt



⇒ Whenever there is an interrupt the CPU responds to the request by enabling the interrupt acknowledgement line. This signal is received by device 1 at its Priority In (PI) input.

⇒ If device 1 is not requesting an interrupt the signal is passed to next device through Priority Out (PO) output by putting PO=1.

⇒ If device has to send data then it place 0 in the PO output block

block the signal for the next device.
⇒ So, the device with PI=1 and PO=0 is the one that is requesting the interrupt.

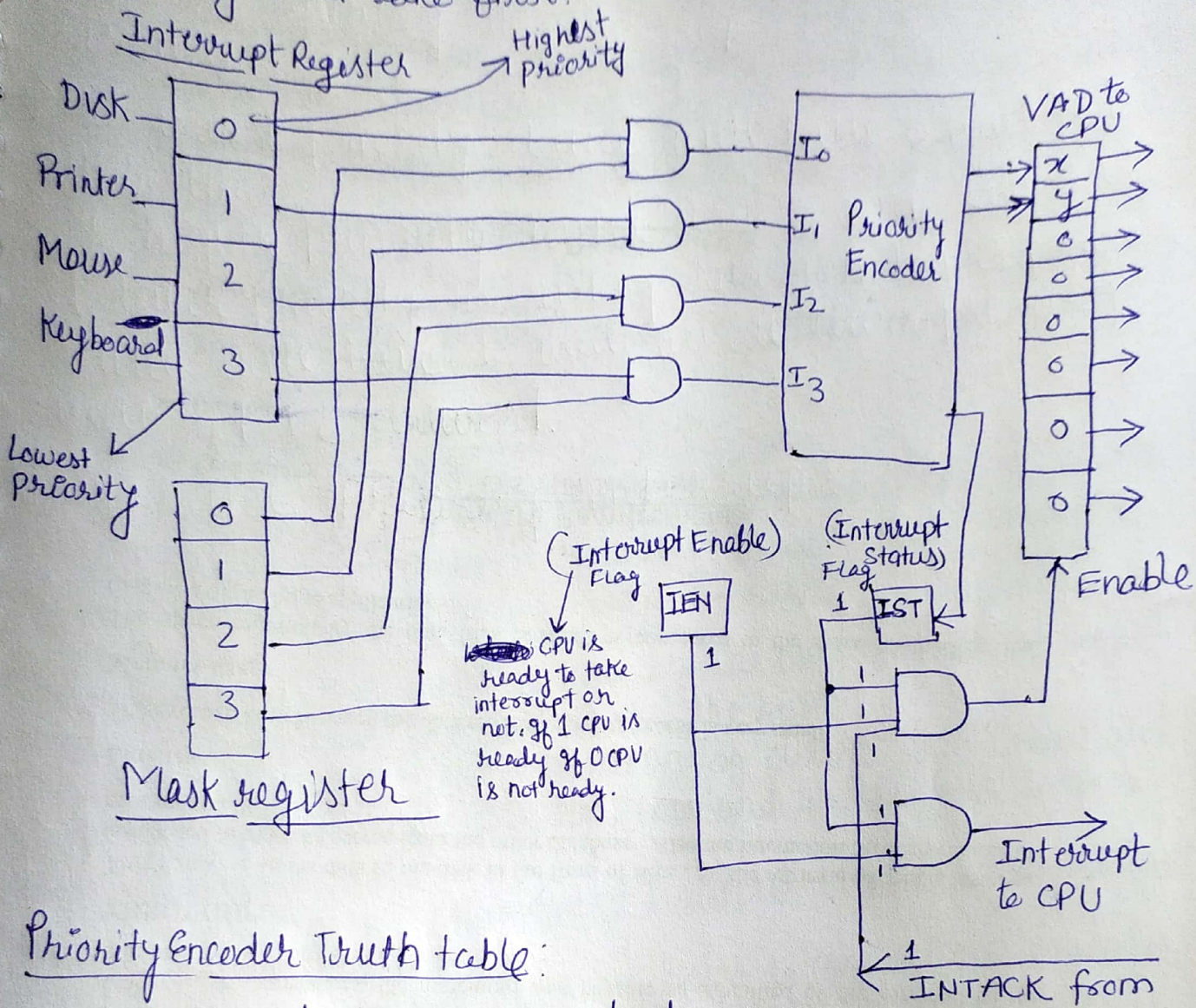
Disadvantage ⇒
⇒ The lowest priority device will have to wait for more time.
⇒ If higher priority device have to send data and if lower priority is sending data then higher priority device will have to wait till all lower priority will not be serviced.

(11) Parallel Priority Interrupt ⇒

⇒ In it any device can interrupt the CPU. and if higher priority device have to send data and if lower priority is sending data then higher priority device can interrupt the CPU.
⇒ An interrupt is recognized only if AND gate output will be 1. for 0 output of AND gate there will be NO interrupt.
⇒ Let All devices generate an interrupt at the same time so there output bit will be 1. and mask register has 1 input at the start. So AND gate output of all devices will generate 1. But since Disk has highest priority so $I_0 = 1$ and other output of AND gate for I_1, I_2 and I_3 input will be don't care. and output value of x and y will be 00 for 0 device. which will be the vector address for CPU.
⇒ Now, let second highest priority device make an interrupt signal in this case first highest priority device put 0 as input to AND gate and Masking result will give 0. and second highest priority device will have 1 and mask value is also 1 so AND gate output will be 1. and other lower priority devices interrupt will be don't care so when $I_0 = 0$ & $I_1 = 1$ then I_1 interrupt will be selected and x & y values will be 0 & 1. This will work in similar manner as Priority encoder truth table next page.

Priority Encoder ⇒ The priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more input inputs arrive at the same time. The input having highest priority will be serve first.

arrive at the same time, The input having highest priority will take first.



Priority Encoder Truth table:

Inputs				Outputs		
I_0	I_1	I_2	I_3	x	y	IST
1	x	x	x	0	0	1
0	1	x	x	0	1	1
0	0	1	x	1	0	1
0	0	0	1	1	1	1
0	0	0	0	x	x	0

when IEN=1 & IST=1 CPU gets interrupt signal and send Interrupt Acknowledgment value=1. and receives the Vector Address (which tells the CPU where to handle the interrupt) from Interrupt device.

⇒ If input I_0 has highest priority, so regardless of the values of other inputs when this input is 1, the output generates an output xy=00. I_1 has next priority level. The output is 01 if I_1=1 provided that I_0=0.

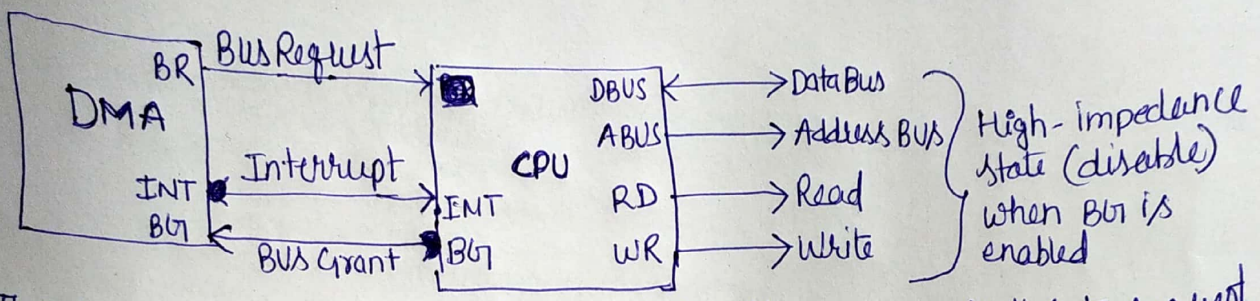
⇒ The interrupt status IST is set only when one or more inputs are equal to 1. If all inputs are 0, IST is cleared to 0. This is because vector address is not transferred to CPU when IST=0.

DMA (Direct Memory Access)

⇒ "Direct memory access is a technique for moving data directly between the main memory and the I/O devices without the intervention of CPU"

⇒ During DMA transfer, the CPU is idle and has no control of memory buses.

⇒ A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.



⇒ The BUS Request (BR) input is used by the DMA controller to request the CPU to release the control of the buses.

⇒ Then, CPU terminates the execution of current instruction and places the address bus, data bus, read and write signals into a high-impedance state.

⇒ The CPU activates BG (Bus grant) output to inform the DMA that the buses are disable.

⇒ Now, DMA takes control of these buses to conduct I/O transfer to and from memory without intervention of CPU.

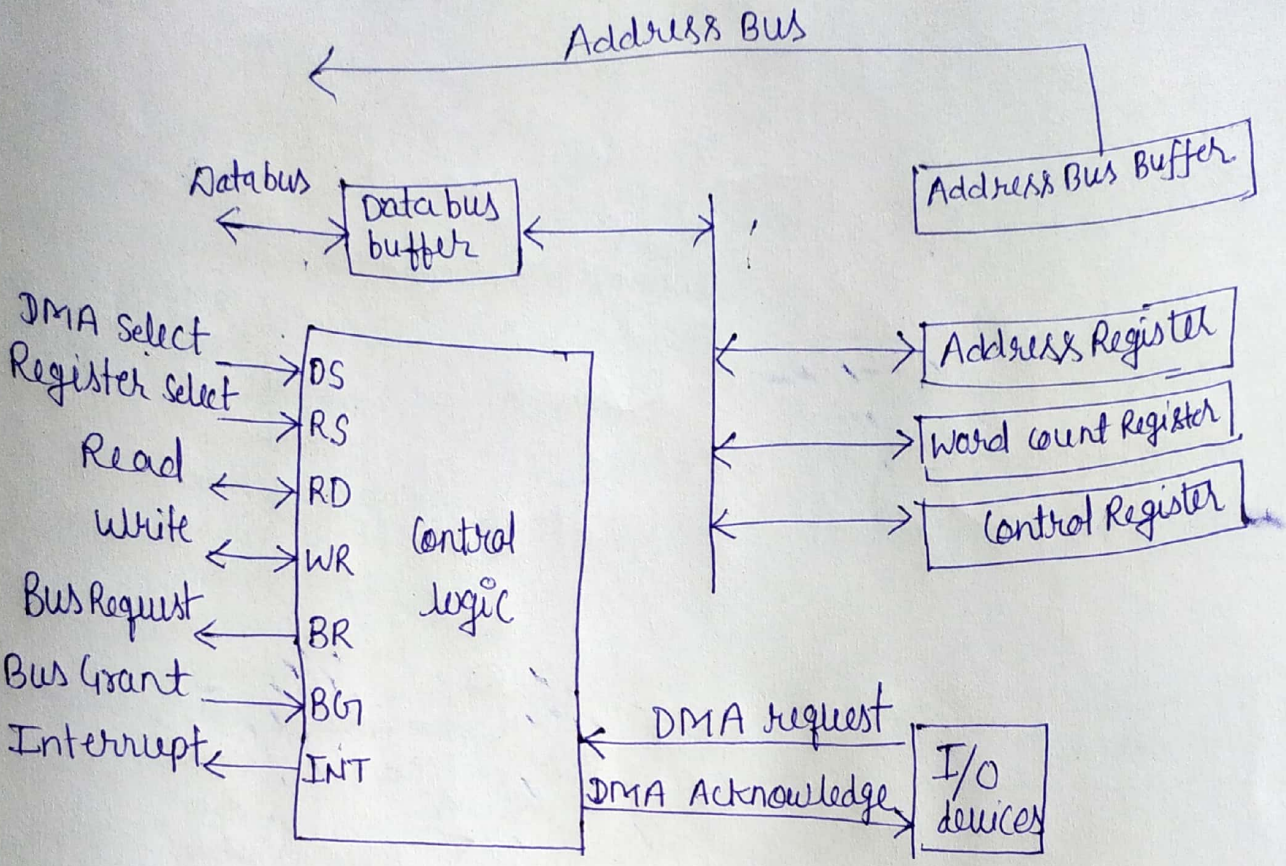
DMA controller

⇒ DMA controller needs an address register, a word count register and control register for direct communication with memory.

⇒ The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs.

⇒ When BG (Bus grant) is 0, The CPU communicate with DMA registers through the data bus buffer to read from or write to the DMA registers.

⇒ When BG = 1, the CPU releases the buses for DMA and now DMA directly communicate with memory by putting the address in address bus and activating the Read or Write



Block diagram of DMA controller

⇒ DMA Controller has three Registers:

① Address Register ⇒ Contains the address to specify the desired location in memory and address bits goes into the address bus through address bus buffer.

② Word count register ⇒ It holds the number of words to be transferred to memory. This register decremented by one after each word transferred into memory till zero.

③ Control register ⇒ It specifies ~~whether~~ whether it is read or write operation.

DMA Transfer

⇒ CPU communicates with the DMA with address and data bus.

⇒ CPU Initializes the DMA through data bus

⇒ When I/O device send DMA request, the DMA controller activates the BR (Bus Request) line, then CPU activates the BG line.

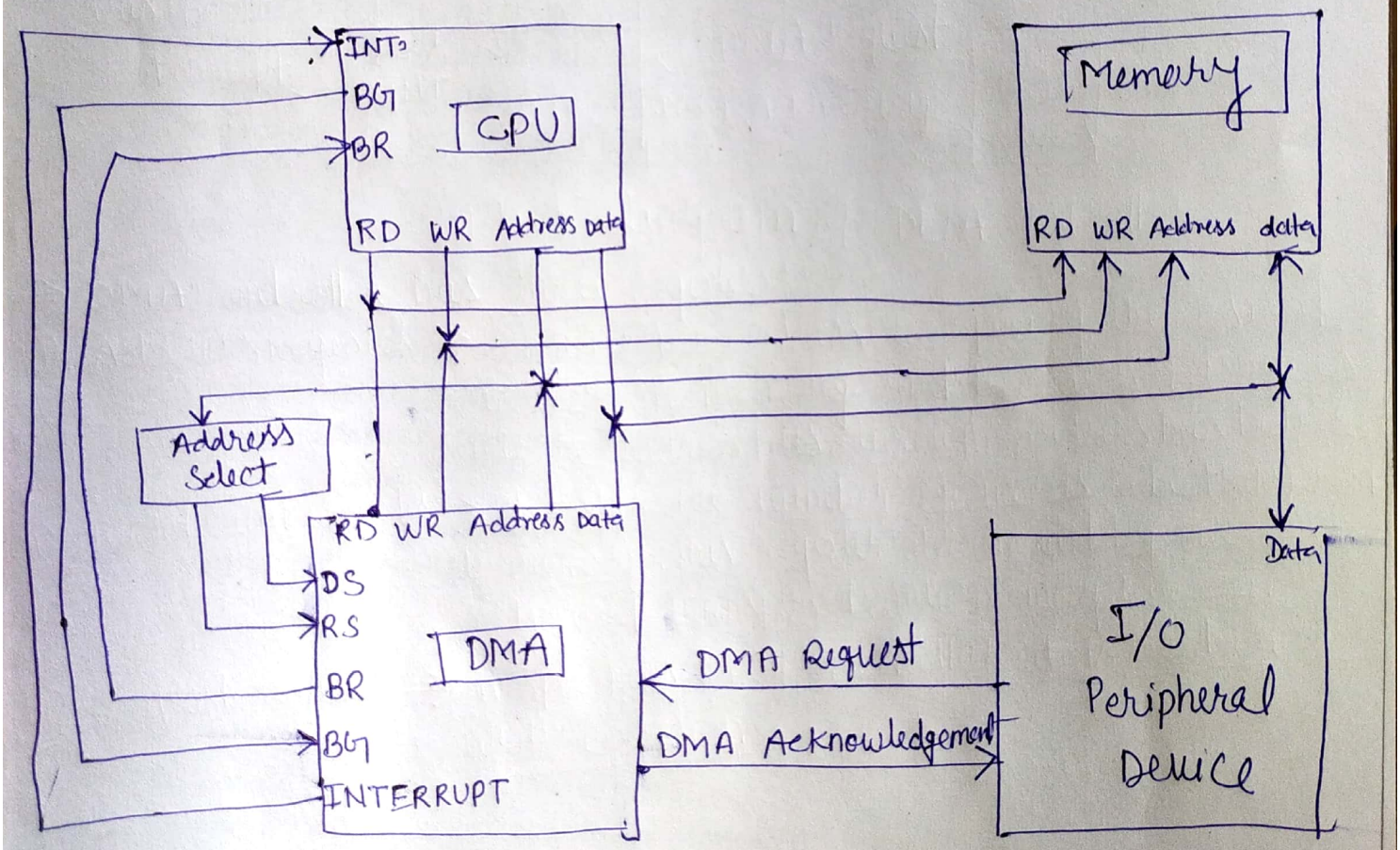
⇒ Then the DMA sends a DMA acknowledge to the device by putting

current value of its address register into address bus and initiates the RD (Read) or WR (write) signals.

⇒ After receiving the signal the I/O device puts a word in the data bus (for write) or receives a word from the data bus for (read).

⇒ The DMA increment its address register and decrements the word count register after each word that is transferred

⇒ If word count register reaches to zero, DMA stops and disables its burst request signals and inform the CPU through an interrupt.



DMA Transfer

IOP (Input-output Processor) or I/O channel

⇒ "An input-output processor (IOP) is a processor that is dedicated to handle only input and output operations.

⇒ The block diagram of the computer having CPU and IOP both shown below.

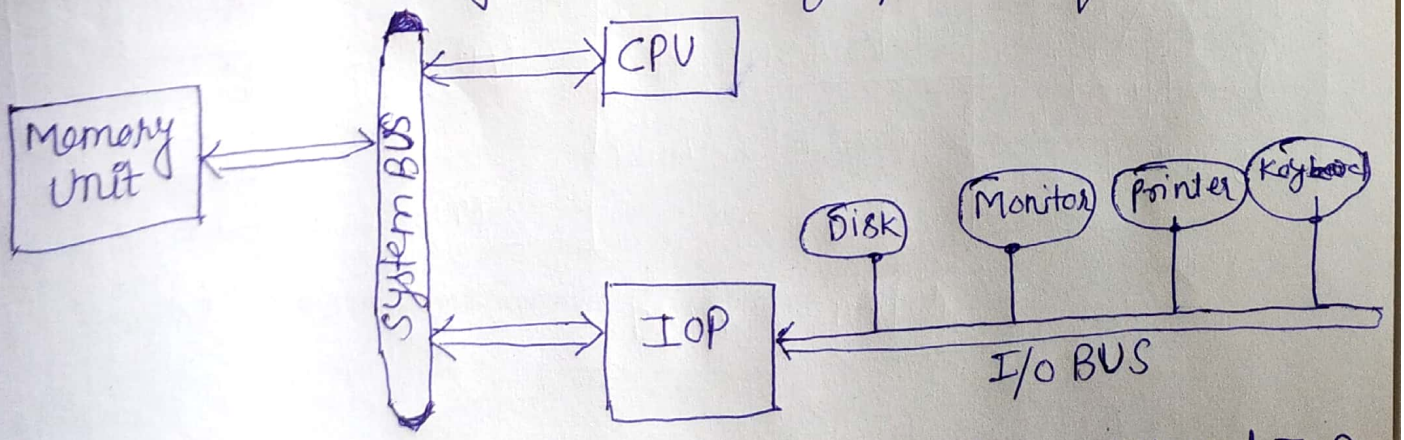
⇒ IOP provides a path for transfer data between various I/O devices and memory. CPU only initiates the I/O program and then IOP operates independently of CPU and transfer data from I/O to memory.

⇒ The CPU sends instructions to test I/O status conditions

⇒ The IOP responds by placing a status word at the memory location specified by CPU.

⇒ When a I/O operation is desired, the CPU informs the IOP about the location where to find the I/O program and then leaves the transfer details to the IOP.

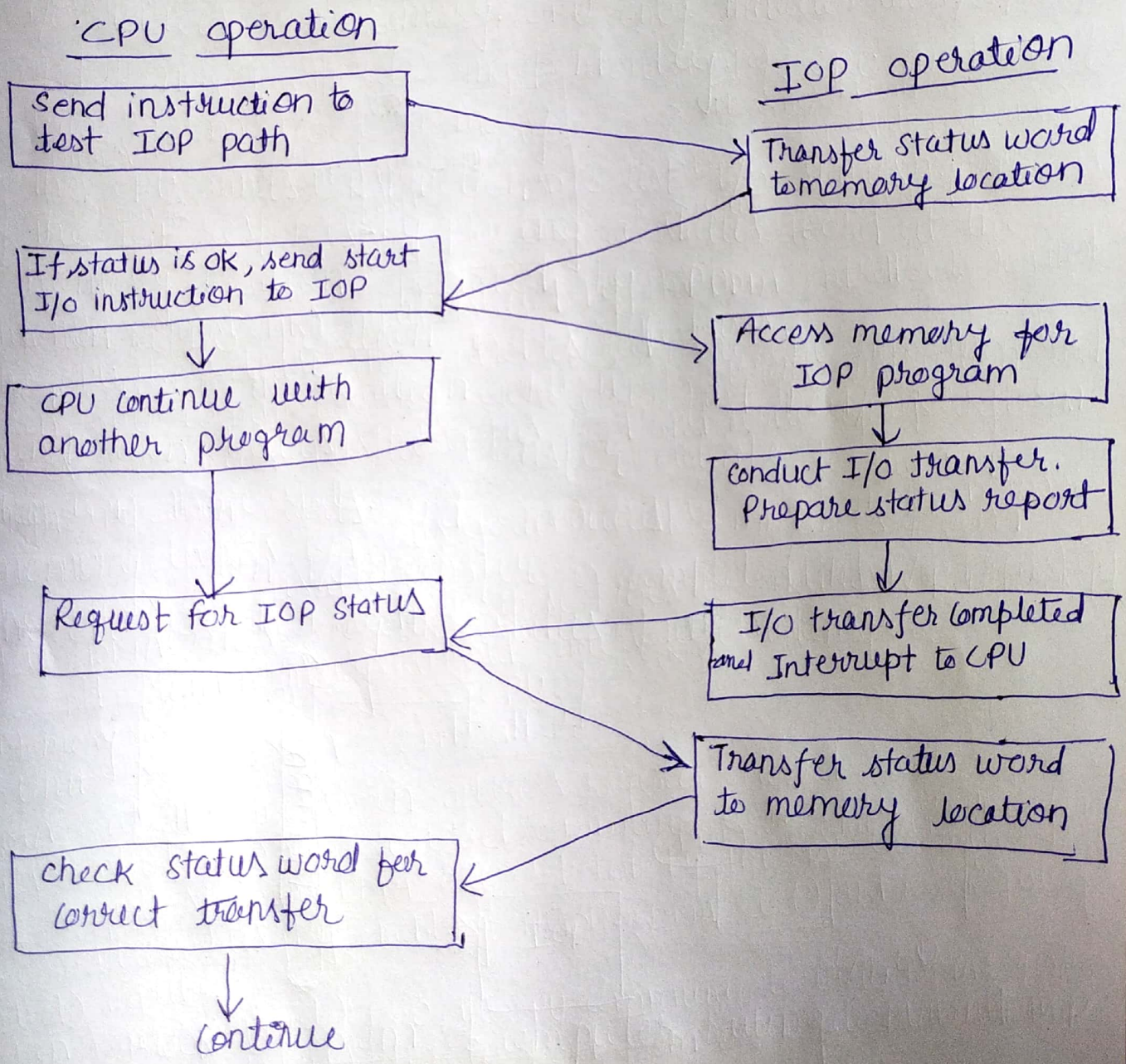
⇒ The IOP then reads and executes commands from memory. When I/O transfer is completed the IOP interrupt the CPU for checking the status of I/O transfer.



Block diagram of Computer having CPU and IOP

CPU-IOP communication

⇒ The communication starts when CPU sends an instruction to IOP to test path.
⇒ IOP sends a status-word to memory for the CPU to check the status.
⇒ The CPU then decides and sends an instruction to IOP to start I/O transfer.
⇒ This instruction also contain an memory address to tell the IOP where to find the program
⇒ Complete CPU-IOP communication is shown below.



CPU-IOP communication

Asynchronous data transfer

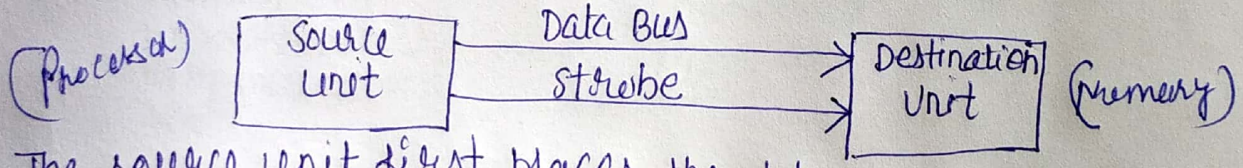
⇒ Asynchronous data transfer is a system in which different parts of computer works in different clock.
⇒ Since, it is not possible to maintain the same clock between processor, memory and I/O devices because processor is faster than memory and I/O devices, and memory is faster than I/O devices.
⇒ There are two methods for asynchronous data transfer
① Strobe control ② Handshaking.

① Strobe control ⇒

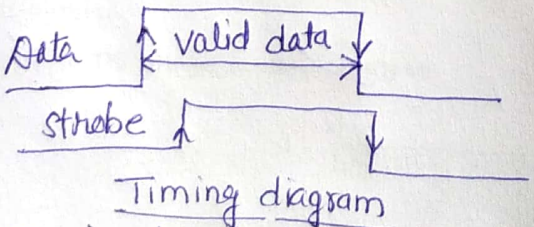
⇒ This method use a single control line called strobe. It may be source-initiated or destination-initiated

Source-Initiated Strobe for data transfer ⇒

⇒ The strobe line is initiated by source as shown in fig



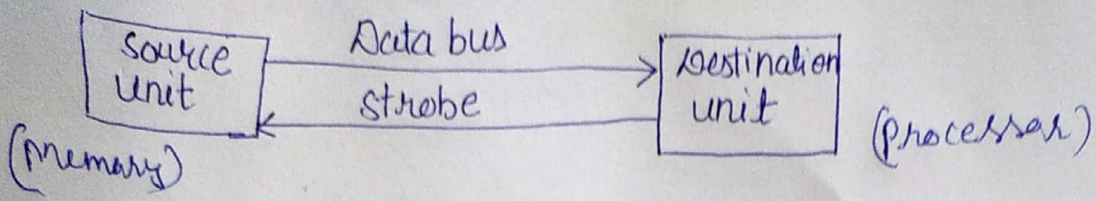
⇒ The source unit first places the data on the bus. After a small delay to ensure that the data settle to a steady value, the source activates the strobe signal.
⇒ The information on the ~~bus~~ data bus and strobe control signal remain in the active state for a sufficient period of time to allow the destination unit to receive the data.

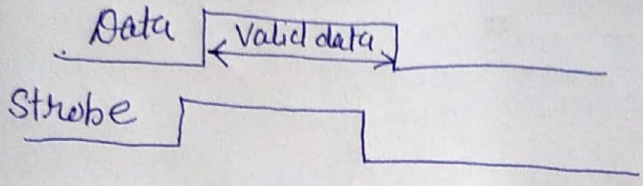


Example is data write operation by processor in memory.

Destination-initiated strobe for data transfer ⇒

⇒ In this case, the destination unit activates the strobe signal, informing the source to provide the data. The source unit responds by placing the data on data bus.





Timing diagram

ex ⇒ Data read operation by CPU from memory.

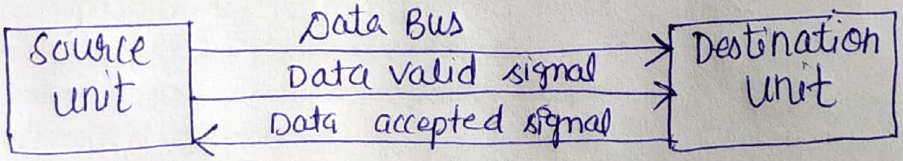
Disadvantage of Strobe control

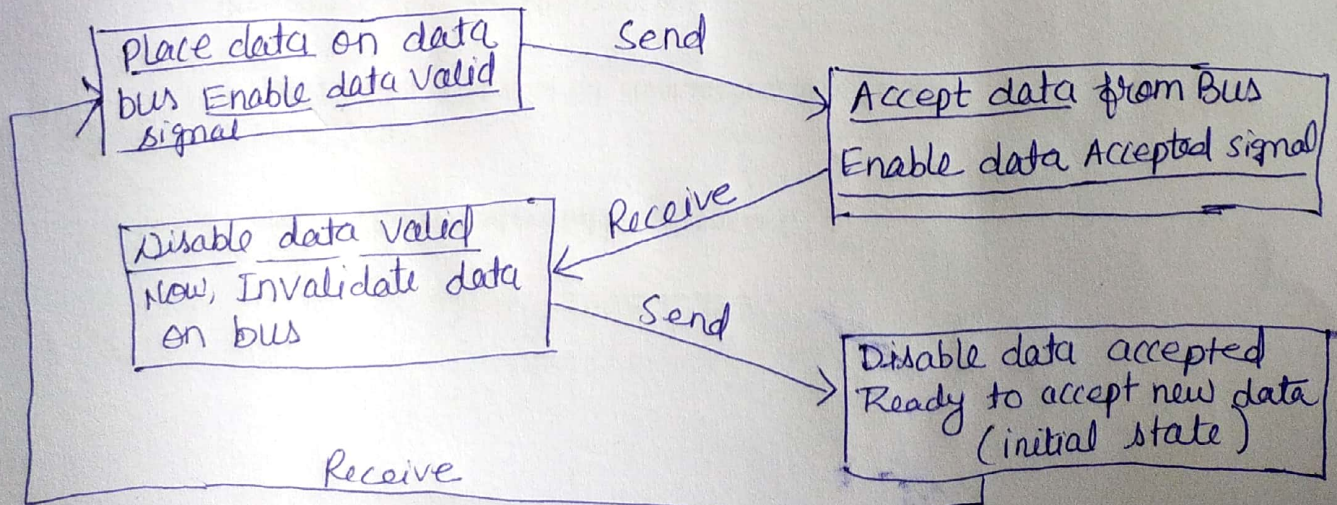
⇒ The source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data. Similarly the same case when destination unit initiate

② Handshaking ⇒

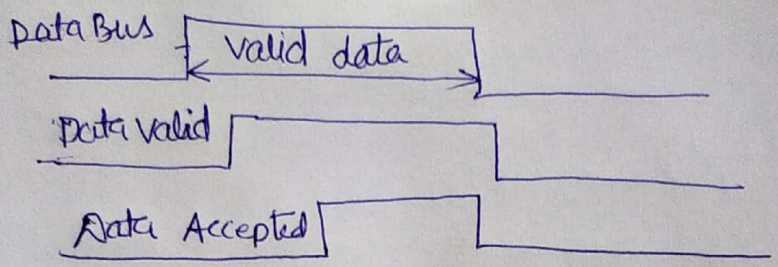
⇒ The disadvantage of strobe method can be solve by Handshaking method. This method include a second control signal line that provide a reply to the unit that initiate the transfer. Handshaking is of two types:

ⓐ Source-initiated Handshaking ⇒



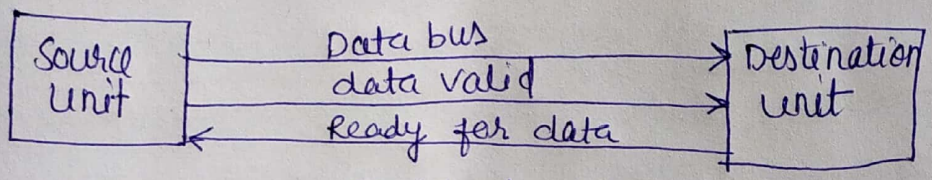


Sequence of Events

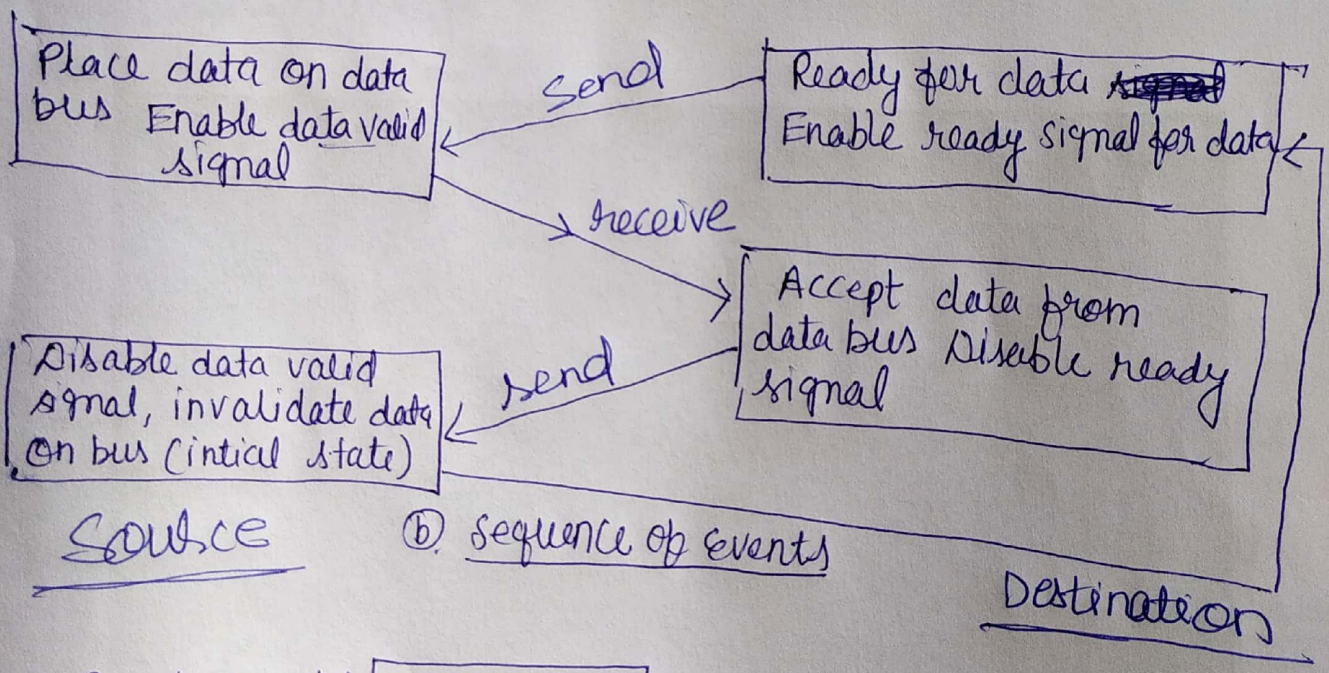


Timing Diagram

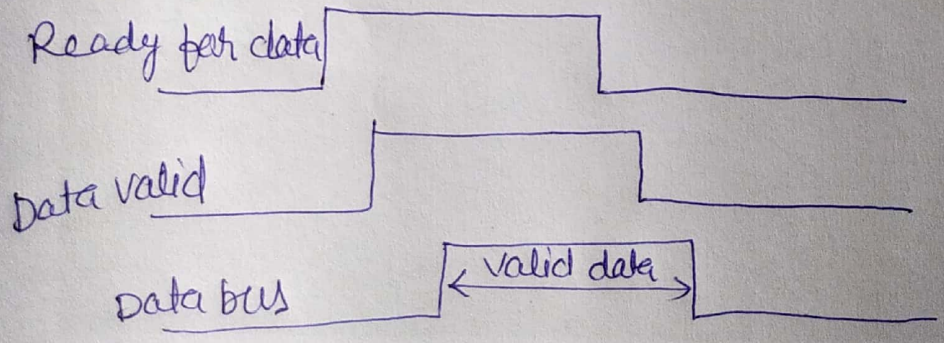
(b) Destination-initiated Handshaking



(a) block diagram



(b) Sequence of events



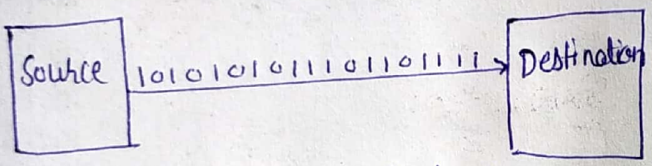
(c) Timing diagram

Serial communication or transmission

⇒ In serial communication each bit in the message is sent in sequence one at a time.
⇒ Serial transmission can be of two types
① Synchronous serial transmission
② Asynchronous serial transmission

① Synchronous Serial transmission

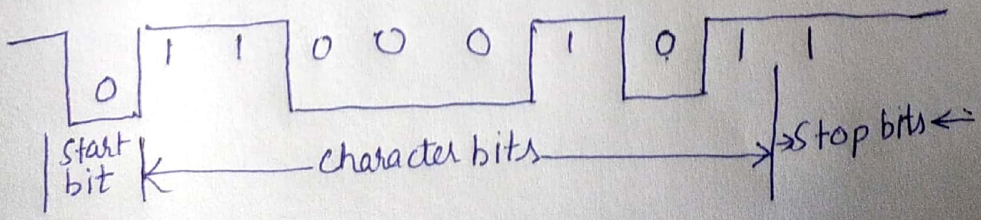
⇒ In synchronous serial transmission, the two units (source and destination) share a common clock and bits are transmitted continuously at the rate dictated by the clock.



② Asynchronous Serial transmission

⇒ In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.
⇒ In this technique, each character consists of three parts
⇒ a start bit is always a 0 and is used to indicate the beginning of a character.
⇒ The character bits
⇒ The stop bits is alway 1. After the character bits are transmitted one or two stop bits are sent.

ex ⇒



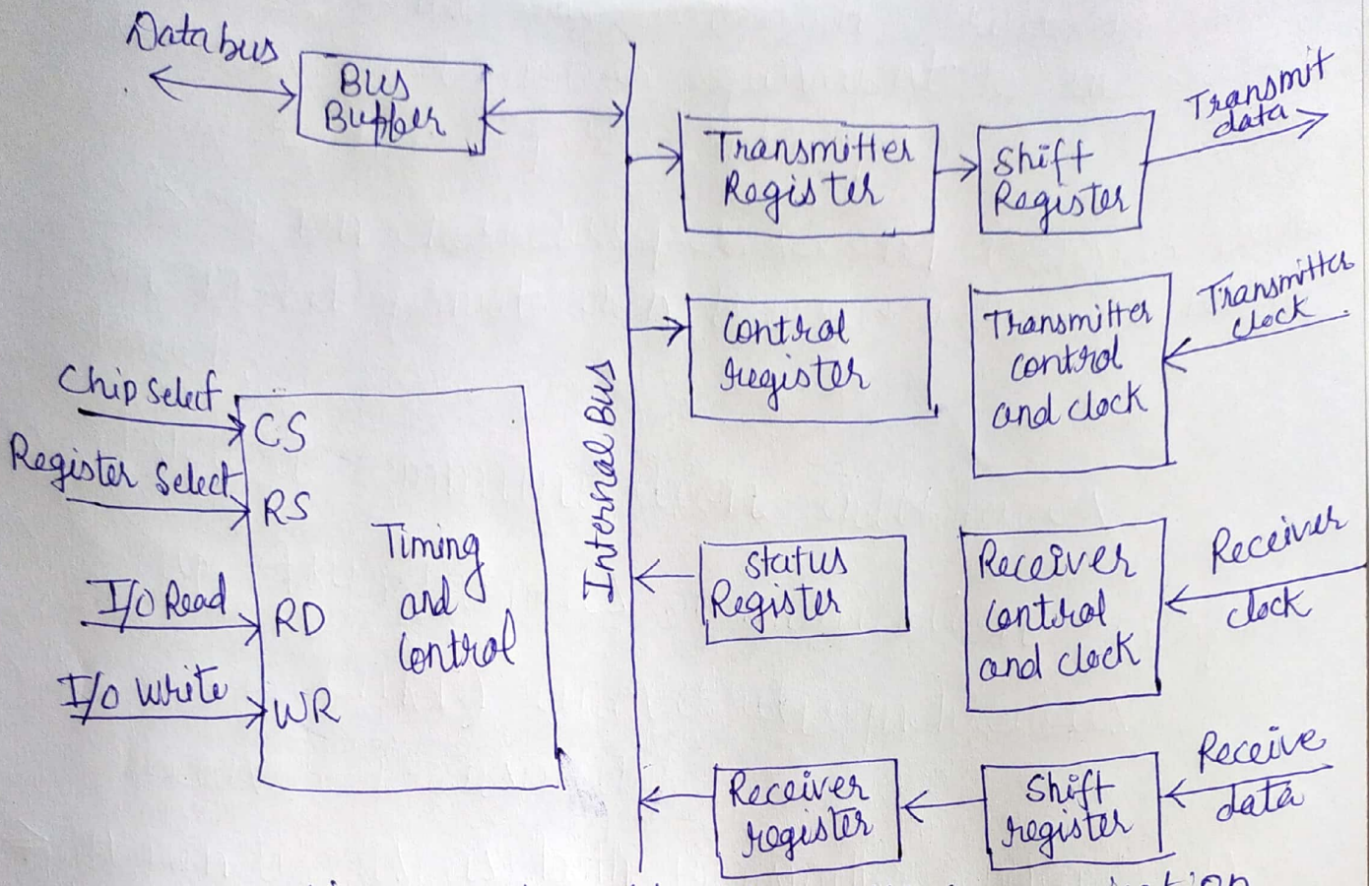
Baud rate ⇒ The rate at which serial information is transmitted and is equivalent to the data transfer in bits per second (bps).

Ex ⇒ If Ten characters per second transfer with an 11-bit format then transfer rate will be 110 baud.

Standard Communication interfaces

Asynchronous communication interface or (UART) (Universal asynchronous receiver-transmitter)

⇒ It functions as both a transmitter and a receiver.
⇒ The Transmitter register accept a data byte from the CPU through the data bus. This byte is transferred to a shift register for serial transmission as shown in fig next page.
⇒ The receiver portion receives serial information into an other shift register, and when a complete data by is received it is transferred to the receiver register. The CPU can select the receiver register to read the byte through the data bus.
⇒ The CPU can read the status register to ~~chek~~ check the status of the flag bits and to find out if any errors have occured.
⇒ The chip select (CS) and ~~Read~~ Read (RD), Write (WR) control lines communicate with CPU.
⇒ The chip select (CS) input is used to select the interface through the address bus.
⇒ The Register Select (RS) is used with the Read (RD) and Write (WR). Controls. Two registers are Read-only and two are write-only as shown in truth table.



Block diagram of a Asynchronous communication interface

CS	RS	Operation	Register Selected
0	X	X	None; data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status Register

Table

⇒ The operation of (UART) interface is initialized by the CPU by sending a byte to the control register. The initialization procedure places the interface in a specific mode of operation as it defines certain parameters such as the baud rate to use, how many bits are in each character, How many stop bit are appended.

⇒ Two bits in the status register are used as flags. One bit is used to indicate whether the transmitter register is Empty and another bit is used to indicate whether the receiver register is FULL

Transmitter ⇒ The CPU reads the status register and checks the flag to see if the transmitter register is empty.
⇒ If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.
⇒ The first bit in the transmitter shift register is set to 0 to generate a start bit.
⇒ The character is transferred in parallel from the transmitter register to the shift register and stop bits are appended into shift register. The transmitter register is then marked empty.
⇒ The character can now be transmitted one bit at a time by shifting the data in the shift register at the specified baud rate.
⇒ Now, the CPU can transfer another character to the transmitter register after checking the flag in the status register.

Receiver ⇒
⇒ The receiver data input is in the 1-state when the line is idle. The receiver control monitors the receive data line for a 0 signal to detect of a start bit.
⇒ Once a start bit has been detected, the incoming bits of the character are shifted into the shift register.
⇒ After receiving the data bits, the interface checks for the ~~parity and~~ stop bits.
⇒ The character without start and stop bits is then transferred in parallel from shift register to the receiver register.
⇒ The flag in the status register is set to indicate that the receiver register is full.
⇒ The CPU reads the status register and checks the flag and if set, it reads the data from the receiver register.

Different Modes of Serial Communication

① Simplex mode ⇒

⇒ It allows transmission of data in only one direction i.e data travel in only one direction.

Ex⇒ radio and television broadcasting.

② Half-duplex ⇒

⇒ It allows transmission of data in both directions but data can be transmitted in only one direction at a time. A pair of wires is needed for this mode.

③ Full-duplex ⇒

⇒ It allows transmission of data in both directions simultaneously.

Q 11.12 ⇒ How many character per second can be transmitted over a 1200-baud line in each of the following modes? (Assume a character code of eight bits)

(a) Synchronous serial transmission
(b) Asynchronous serial transmission with two stop bits
(c) Asynchronous serial transmission with one stop bits.

Soln:- line = 1200-baud
each character code = 8-bits

(a) No. of characters per second can be transmitted in Synchronous serial transmission $= \frac{1200}{8}$
$= 150$ cps Ans
(Character per second)

(b) In Asynchronous serial transmission there will be one start bit also and given two stop bits.
So total bits in character code $= 1+8+2 = 11$
So, No. of character per second can be transmitted $= \frac{1200}{11}$ CPS Ans

(c) one start bit and one stop bit $= 1+8+1 = 10$
So, No. of character per second can transmitted $= \frac{1200}{10} = 120$ CPS Ans

~~(a) How many characters per second can be transmitted over a~~

(a) 11.13: Information is inserted into a FIFO buffer at a rate of m bytes per second. The information is deleted at a rate of n byte per second. The maximum capacity of the buffer is K bytes.
(a) How long does it take for an empty buffer to fill up when $m > n$?
(b) How long does it take for a full buffer to empty when $m < n$?
(c) Is the FIFO buffer needed if $m = n$?

Soln ⇒ given,

Information inserted into a FIFO buffer = m bytes/sec

Information deleted at a rate = n bytes/sec

Maximum capacity of FIFO buffer = K bytes

(a) Time to take for an empty buffer to fill up when $m > n$

$$= \frac{\text{total capacity of buffer}}{m-n}$$

$$= \frac{K}{m-n} \quad \text{Ans}$$

(b) Time to take for a full buffer to empty when $m < n$

$$= \frac{\text{total capacity of buffer}}{n-m}$$

$$= \frac{K}{n-m} \quad \text{Ans}$$

(c) No Need of FIFO buffer is $m = n$ Ans

(2) (11-29) Same A DMA controller transfer 16-bit words to memory using cycle stealing. The words are assembled from a device that transmits characters at a rate of 2400 character per second. The CPU is fetching and executing instructions at an average rate of 1 million instruction per second. By how much will the CPU be slowed down because of the DMA transfer?

Soln ⇒ given,

⇒ A DMA controller transfer 16-bit words to memory using cycle stealing

⇒ character transmission rate = 2400 cps

⇒ CPU Fetch and execute instruction at an average rate = 1 million instruction/sec

∴ 2400 character transfer = 1 sec

1 character " $= \frac{1}{2400}$ sec $= \frac{1}{2400} \times 10^6$ μsec ($1 \text{ sec} = 10^6$ μsec)

$= 416.6$ μsec

For two character (16-bit) $= 16 \times 416.6$

$= 833.3$ μsec

CPU slowed down $= \frac{1}{833.3} \times 100 = 0.12\%$