

Unit-03/Lecture-01

Modes of Data Transfer

Programmed I/O Mode

Programmed I/O operations are the result of I/O instructions written in computer program. Each data item transfer is initiated by an instruction in the program. The I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU.

The data transfer can be synchronous or asynchronous depending upon the type and the speed of the I/O devices.

If the speeds match then synchronous data transfer is used. When there is mismatch then asynchronous data transfer is used. The transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU and memory. This method requires constant monitoring of the peripheral by the CPU. Once a data transfer is initiated the CPU is required to monitor

The interface to see when a transfer can again be made. In this method the CPU stays in a loop till the I/O unit indicates that it is ready for data transfer. This is time consuming process which can be solved by using interrupt.

In programmed I/O mode data are exchanged between the processor and the I/O module. When a processor is executing a program and encounters an instruction relating to I/O, it executes that instruction by issuing a command to that appropriate I/O module. With programmed I/O the I/O module will perform the requested action and then set the appropriate bit in the I/O status register .

The I/O module takes no further action to alert the processor (it doesn't interrupt the processor).

The I/O commands issued by the processor to the I/O module

- Test
- Control
- Read
- Write

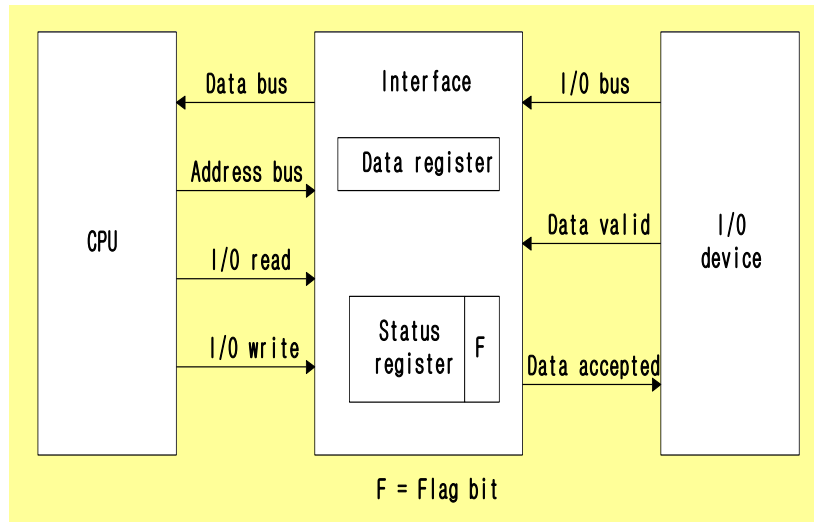


Fig 3.1 Example of Programmed I/O mode

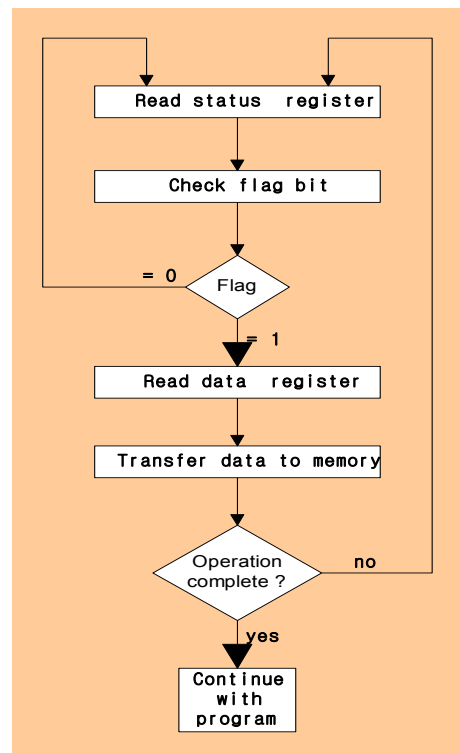


Fig 3.2 Flowchart of Programmed I/O Mode

Advantage of programmed I/O mode

- A program and processor dedicated to wait and repeatedly tests the status and for IO data transfer till the IO operation completes

Disadvantage of programmed I/O mode

- A program has to wait and repeatedly tests the status; Waiting period for an asynchronous event can be too large.
- Many I/O devices generate asynchronous events— events that occur at times that the processor cannot predict or control, but which the processor must respond to reasonably quickly to provide acceptable performance

Interrupt driven

The CPU issues commands to the I/O module then proceeds with its normal work until interrupted by I/O device on completion of its work.

For input, the device interrupts the CPU when new data has arrived and is ready to be retrieved by the system processor. The actual actions to perform depend on whether the device uses I/O ports, memory mapping.

For output, the device delivers an interrupt either when it is ready to accept new data or to acknowledge a successful data transfer. Memory-mapped and DMA-capable devices usually generate interrupts to tell the system they are done with the buffer.

Although Interrupt relieves the CPU of having to wait for the devices, but it is still inefficient in data transfer of large amount because the CPU has to transfer the data word by word between I/O module and memory.

Below are the basic operations of Interrupt:

- CPU issues read command

- I/O module gets data from peripheral whilst CPU does other work
 - I/O module interrupts CPU
 - CPU requests data
 - I/O module transfers data
-
- In this method the program issues an I/O command and then continues to execute until it is interrupted by the I/O hardware to signal the end of I/O operation.
 - Here the program enters a wait loop in which it repeatedly checks the device status. During this process the processor is not performing any useful computation.
 - There are many situations where tasks can be performed while waiting for an I/O device to be ready, to allow this the I/O device should alert the processor when it becomes ready. It can be done by sending a hardware signal called an **interrupt**.
 - The routine executed in response to an interrupt request is called **Interrupt Service Routine (ISR)**.
 - The processor first completes execution of instruction then it loads the program counter (PC) with the address of 1st instruction of ISR.

Direct Memory Access

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

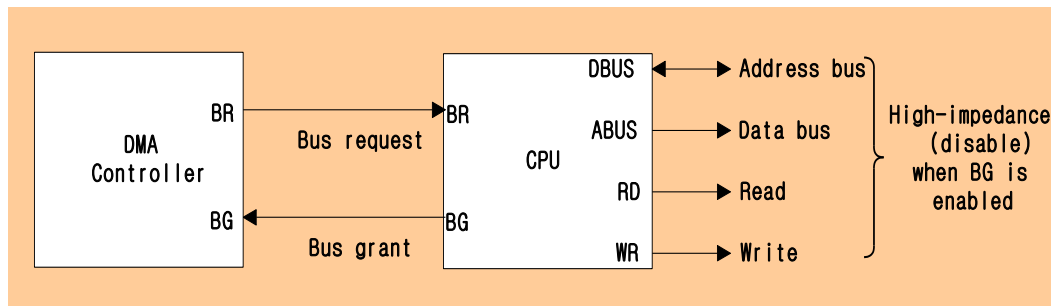


Fig 3.3 Direct Memory Access

Transfer Modes

- 1) Burst transfer : Block
- 2) Cycle stealing transfer: Byte

DMA Controller

DMA Initialization Process

- 1) Set Address register :
 - Memory address for read/write
- 2) Set Word count register :
 - The number of words to transfer
- 3) Set transfer mode :
 - read/write,
 - burst/cycle stealing,
 - I/O to I/O,
 - I/O to Memory,
 - Memory to Memory
 - Memory search
 - I/O search

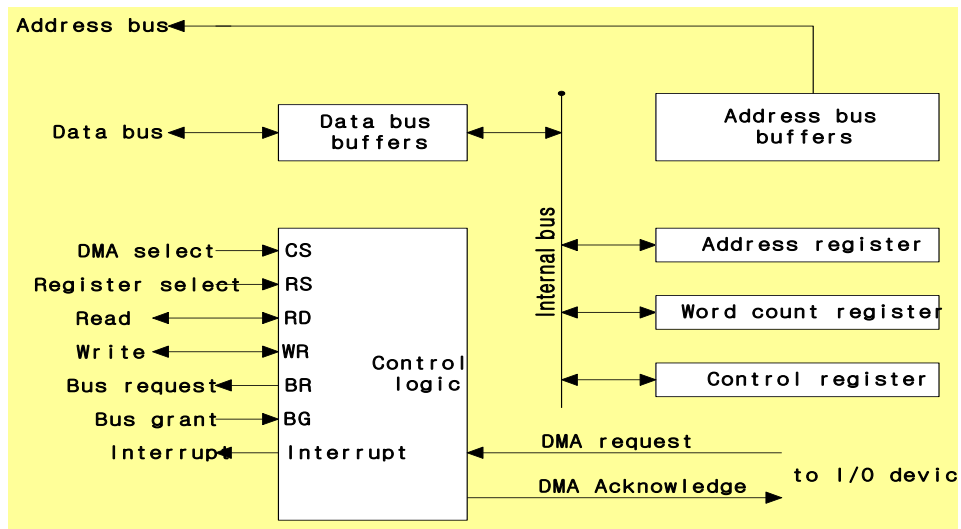


Fig 3.4

	RGPV QUESTIONS	Year	Marks
Q.1	Explain the following modes of data transfer : (i) Program controlled (ii) Interrupt driven (iii) Direct memory access	June 2011	10
Q.2	Explain with suitable example the working principle of DMA controller.	June 2012	7
Q.3	Write three modes of data transfer and explain any one of them.	June 2014	7
Q.4	What is DMA ? Describe how DMA is used to transfer data from peripherals.	June 2013	7
Q.5	Explain the drawbacks in programmed I/O and interrupt driven I/O	June 2013	7

Unit-03/Lecture-02

Interrupt structure

Interrupt structure refers to the precedence of interrupts. Hardware events that cause interrupts are assigned CPU interrupt levels. The CPU can disable interrupts of a certain level and below, thus allowing an important interrupt to preempt an interrupt of lower priority, but not vice-versa. Most machine architectures do not allow the software to reconfigure the interrupt structure. If the precedence of interrupts can be reconfigured, though, make sure that the clock that is used for task-switching has its interrupts serviced at a high priority so that lower priority events (such as disk access) are not capable of disabling the preemptive scheduling. The serial port should also have a high priority so that high baud rates can be reliably used.

The precedence of interrupts is important because disabling interrupts is a common way to provide mutual exclusion in the kernel. For example, a routine that modifies tty structures (those that handle input and output to a terminal) will call `spltty()` (for Set Processor Level TTY) just before a critical section and later will reset the level to what it was before `spltty()` was called. On most machines, this call will disable any interrupt that would call a function that handles the tty structures and, in addition, will disable any interrupts with a priority lower than that of ```tty```. An interrupt structure not well suited for UNIX might cause relatively long interrupt code (*e.g.*, disk access or sound driver processing) to block more frequent interrupts (*e.g.*, timer, keyboard, or serial port). On the Macintosh port, heavy use of the hard disk would cause the system time to be off by several hours a day, and scrolling of the display, being in a tty device, would preempt the keyboard routine and cause keystrokes to be lost. On A/UX, Apple's System V UNIX for the Macintosh, a beep causes serial port data to be lost.

TRAP: TRAP is a nonmaskable vectored interrupt. It can interrupt the μ once the power is on. Most μ interrupt inputs are level sensitive; however, some are edge sensitive and others are both edge and level sensitive. The TRAP input is both edge sensitive and level sensitive interrupt. It means that TRAP makes a low-to-high transition and remains high until it is acknowledged. The positive edge of the TRAP signal will set the D flip-flop. Because of the AND gate, however, the final TRAP also depends on a sustained high-level TRAP input. This is why the TRAP is both edge and level sensitive; this also avoids false triggering caused by noise and transients.

For example, suppose the 8085 is midway through an instruction cycle with another 300 ns to completion. If a 300 ns noise spike hits the TRAP input, it will be edge triggered but not level triggered. The 8085 is still working on the current instruction. Since the TRAP input is both edge and level sensitive, the 8085 avoids responding to false TRAPs.

Since the TRAP input has the highest priority, it is used for catastrophic events such as power failure, parity errors, and other events that require immediate attention. In the case of brief power failure, it may be possible to save critical data with parity errors; the data may be resampled or corrected before going on.

Whenever TRAP comes, the μ completes the current instruction, pushes the program counter on the stack, and branches to fixed location 0054 H. Once the 8085 recognizes a TRAP interrupt, it will send a high TRAP ACKNOWLEDGE bit to the TRAP F/F, thus clearing the F/F so that even if TRAP is high, it is not recognized only if it goes low, then high and remains high. The TRAP F/F is also cleared when the μ is being reset during which it goes low and clears the F/F.

RST 7.5, 6.5, & 5.5:

These are maskable vectored interrupts. These interrupts can be enabled or disabled through software. RST 7.5 has the highest priority among these & RST 5.5 has the lowest priority. RST 7.5 control signal input is a rising edge sensitive interrupt; whenever LOW-to-HIGH transition occurs, it can interrupt the μ . This LO-to-HI transition is registered in the second D-F/F. The output of this F/F is labelled I 7.5. Whenever the other inputs are high, the μ recognizes this interrupt; this request is remembered until (1) the 8085 responds to the interrupt (when interrupt is acknowledged, it sends an acknowledgment to the D-F/F), (2) or until the request is RESET by SIM instruction (R 7.5 bit is made high through SIM instruction and the F/F can be cleared), or (3) or until the user makes use of these facilities.

p is being reset ie signal becomes LOW whenever RST 7.5 is recognized, control is transferred to 003cH RST 6.5 & 5.5 are also vectored masked interrupts and are HIGH level sensitive interrupt control signal input. These are directly connected to AND gate the signal at these inputs must be maintained until the interrupt is acknowledged. Whenever RST 6.5 is recognized, the control is transferred to 0034 H & whenever RST5.5 is recognized, the control is transferred to 002CH. The signals I7.5,6.5 & 5.5 are called pending interrupts. The signal IE (bottom F/F) is called interrupt enable flag, it must be high to active The AND gates, Also notice the M7.5,M6.5 & M5.5 signals, they must be low to enable the AND gates. e.g. to activate RST7.5 interrupt, I7.5 must be high, M7.5 must be low and IE must be high.

INTR: INTR is a maskable interrupt. A HIGH level on this pin interrupt the μ p. the interrupt signal input INTR is not affected by SIM instruction only INTE F/F must be SET to 1 before this interrupt comes. Thus, in 8085 we can write the LOGIC expression for the LOGIC variable VALID INT.

$$\text{VALID INT} = \text{TRAP} + \text{INTE}, [\text{INTR} + \text{R75}.$$

Other than the F/F in the above, there is one more F/F in the μ p called INTA F/F. this is used only for internal operation by the μ p when first power is ON; their F/F is RESET by control signal. Thereafter by the p, it always RESETs the INTE F/F and then SETs the INTA F/F before further action.

Unit-03/Lecture-03

I/O INTERFACE

In computing, input/output or I/O (or informally, io or IO) is the communication between an information processing system (such as a computer) and the outside world, possibly a human or another information processing system. Inputs are the signals or data received by the system and outputs are the signals or data sent from it. The term can also be used as part of an action; to "perform I/O" is to perform an input or output operation. I/O devices are used by a human (or other system) to communicate with a computer. For instance, a keyboard or mouse is an input device for a computer, while monitors and printers are output devices. Devices for communication between computers, such as modems and network cards, typically perform both input and output operations.

Note that the designation of a device as either input or output depends on perspective. Mice and keyboards take physical movements that the human user outputs and convert them into input signals that a computer can understand; the output from these devices is the computer's input. Similarly, printers and monitors take signals that a computer outputs as input, and they convert these signals into a representation that human users can understand. From the human user's perspective, the process of reading or seeing these representations is receiving input; this type of interaction between computers and humans is studied in the field of human–computer interaction.

In computer architecture, the combination of the CPU and main memory, to which the CPU can read or write directly using individual instructions, is considered the brain of a computer. Any transfer of information to or from the CPU/memory combo, for example by reading data from a disk drive, is considered I/O. The CPU and its supporting circuitry may provide memory-mapped I/O that is used in low-level computer programming, such as in the implementation of device drivers, or may provide access to I/O channels. An I/O algorithm is one designed to exploit locality and perform efficiently when exchanging data with a secondary storage device, such as a disk drive.

An I/O interface is required whenever the I/O device is driven by the processor. The interface must have

necessary logic to interpret the device address generated by the processor. Handshaking should be implemented by the interface using appropriate commands (like BUSY, READY, and WAIT), and the processor can communicate with an I/O device through the interface. If different data formats are being exchanged, the interface must be able to convert serial data to parallel form and vice-versa. There must be provision for generating interrupts and the corresponding type numbers for further processing by the processor if required.

A computer that uses memory-mapped I/O accesses hardware by reading and writing to specific memory locations, using the same assembly language instructions that computer would normally use to access memory.

Higher-level implementation

Higher-level operating system and programming facilities employ separate, more abstract I/O concepts and primitives. For example, most operating systems provide application programs with the concept of files. The C and C++ programming languages, and operating systems in the Unix family, traditionally abstract files and devices as streams, which can be read or written, or sometimes both. The C standard library provides functions for manipulating streams for input and output.

In the context of the ALGOL 68 programming language, the input and output facilities are collectively referred to as transput. The ALGOL 68 transput library recognizes the following standard files/devices:

An alternative to special primitive functions is the I/O monad, which permits programs to just describe I/O, and the actions are carried out outside the program. This is notable because the I/O functions would introduce side-effects to any programming language, but this allows purely functional programming to be practical.

Channel I/O

Channel I/O requires the use of instructions that are specifically designed to perform I/O operations. The I/O instructions address the channel or the channel and device; the channel asynchronously accesses all other required addressing and control information. This is similar to DMA, but more flexible.

Port-mapped I/O

Port-mapped I/O also requires the use of special I/O instructions. Typically one or more ports are assigned to the device, each with a special purpose. The port numbers are in a separate address space from that used by normal instructions.

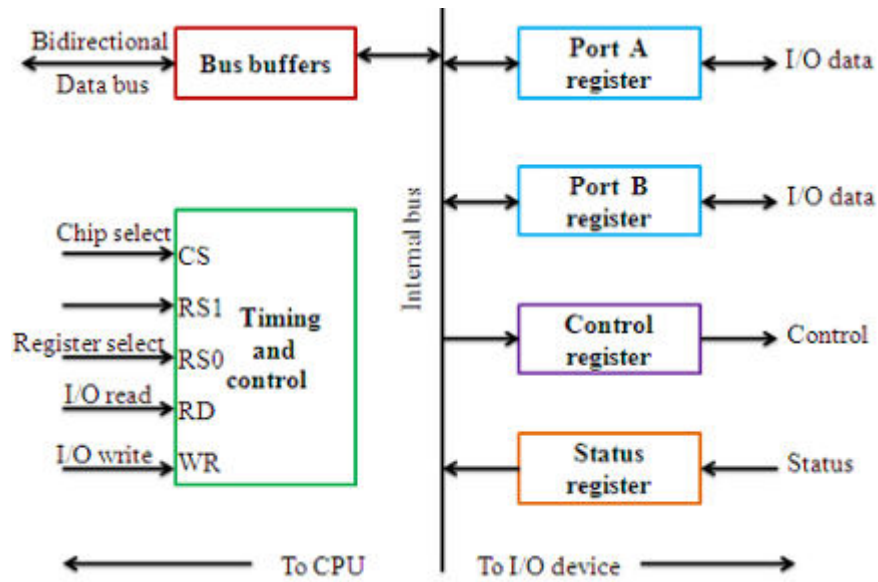


Fig 3.6

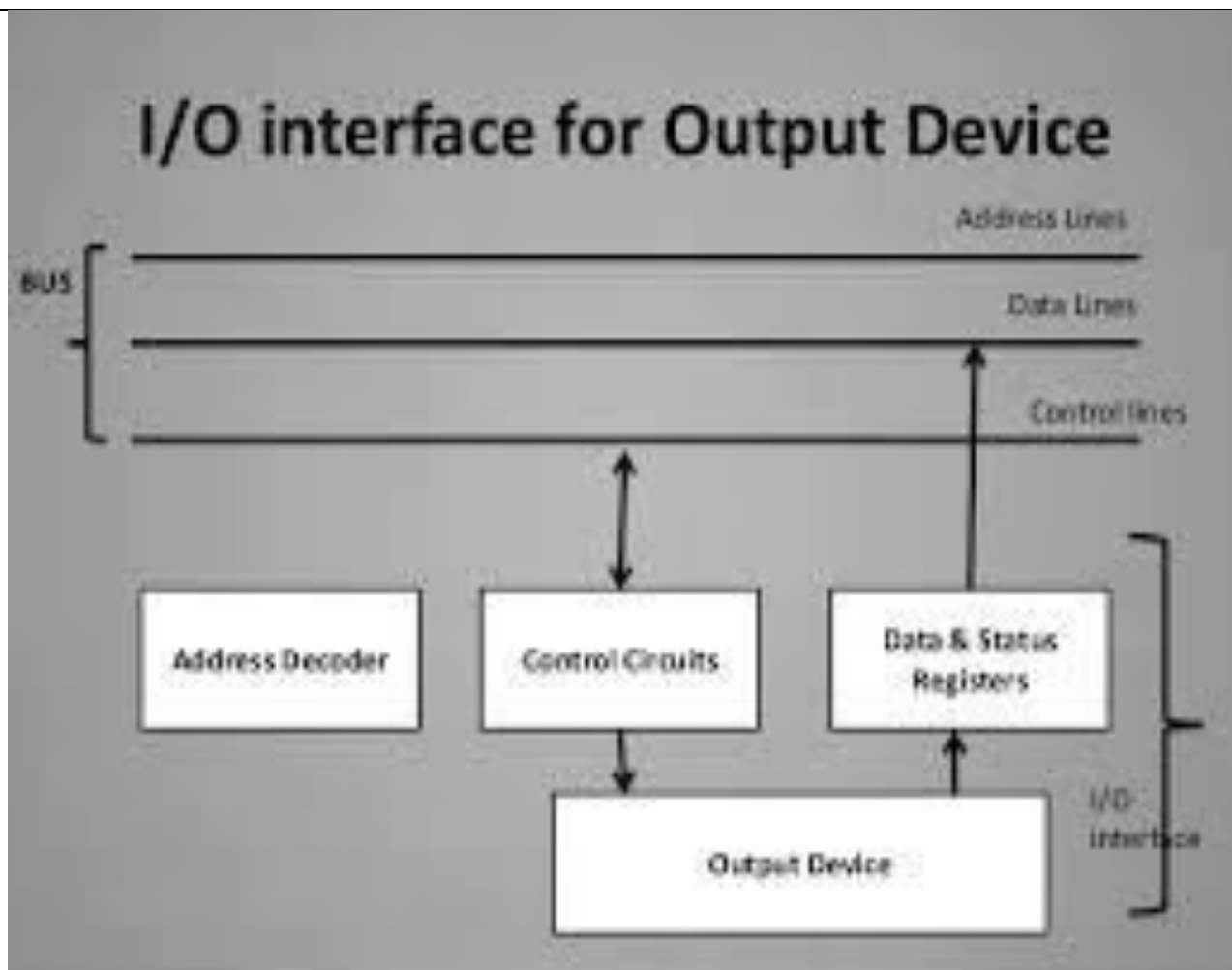


Fig 3.7

	RGPV QUESTIONS	Year	Marks
Q.1	Differentiate between the following: Isolated and memory mapped I/o	June 2011	5

Unit-03/Lecture-04

Synchronous Data Transfer

In a digital system, the internal operations are synchronized by means of clock pulses supplied by a common pulse generator. In a computer, CPU and an I/O interface are designed independently of each other. If the registers in the interface share a common clock with the CPU registers, the data transfer between two units is said to be synchronous.

Asynchronous Data Transfer

In a computer system, CPU and an I/O interface are designed independently of each other. When internal timing in each unit is independent from the other and when registers in interface and registers of CPU use its own private clock, in that case the two units are said to be asynchronous to each other. CPU and I/O device must coordinate for data transfers.

Asynchronous data transfer is a way of transferring data from the various peripherals to CPU in an asynchronous manner. Peripherals and CPU have different speeds and different codes and formats so we employ interfaces between them to communicate. They communicate by transferring of information and this transfer can be in accordance to a single clock or multiple separate clocks. Following a single CPU clock makes its synchronous transfer of data while following separate different clocks for each peripheral device is termed as asynchronous transfer.

In general asynchronous data transfer is followed. The internal operations in a digital system are synchronised with the help of a clock pulse supplied by a common pulse generator. Clock pulses are applied to all registers within a unit. All data transfers between internal registers occur simultaneously during the occurrence of a single clock pulse. Two units, let's suppose CPU and I/O interface are designed independently. If the registers in

the interface share a common clock with the CPU registers then the transfer between the two units is said to be synchronous. Synchronisation means following a common clock in simple terms. But in most cases, the internal timing in each unit is independent from the other as they use its own private clock for internal registers. In such cases the two units are said to be asynchronous to each other. This approach is presently used widely in most of the computers around the world.

For more help log on to Transtutors.com. We also provide assignment help and homework help on wide range of topics. We have experts of varied subjects with good experience. We cover subjects from school to graduate level of almost every discipline.

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communication units to indicate the time at which they send the data. It is similar to timestamp of data at which it is send to other register. There are two ways to achieve this.

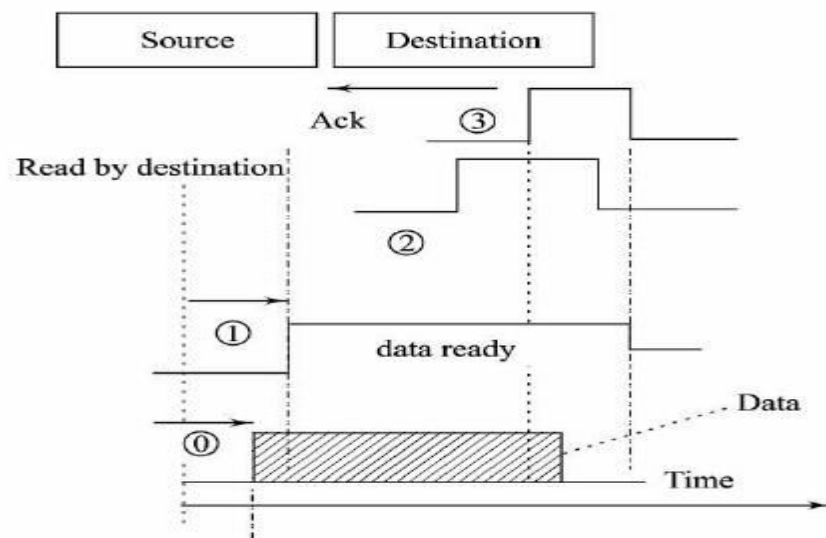


Fig 3.8

Methods used in Asynchronous data transfer

Strobe Control :

This is one way of transfer i.e. by means of strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.

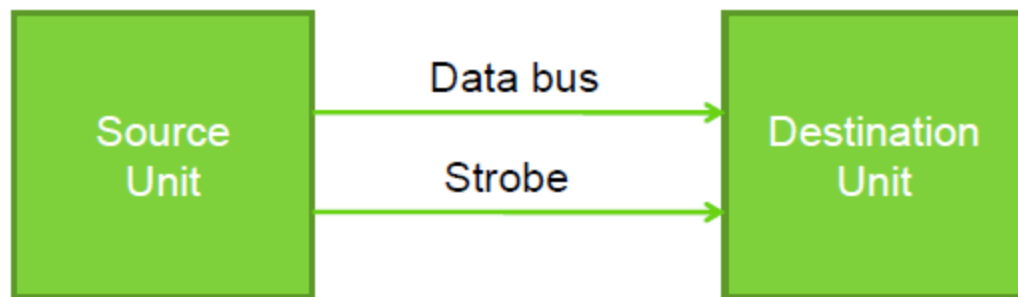


Fig 3.9

Handshaking :

This method is used to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data.

Handshaking is similar to the common phenomena of handshaking where we see to it that the person we are trying to handshake is initially free, and then only we approach towards him/her and set up an agreement by handshaking. In the similar way the data is transferred onto the bus with a control signal indicating the presence of data on bus. When the receiving unit receives the data item it responds back sending another control signal to acknowledge the receipt.

For assignment help and homework help log on to Transtutors.com. Our experts will help you in every domain of your subject and get your work done on time and as per your specification. The strobe pulse and handshaking method of asynchronous data transfer are not at all restricted to I/O transfers. In fact they are extensively used for numerous transfers in two independent units of different clocks only.

	RGPV QUESTIONS	Year	Marks
Q.1	what do you mean by synchronous and asynchronous data transfer ? explain handshaking method of asynchronous data transfer	June 2012	7
Q.2	Differentiate between the following – Synchronous and asynchronous modes of serial data transfer	June ,Dec 2011	7

Unit-03/Lecture-05

I/O processor

The concept of I/O processor is an extension of the concept of DMA. The I/O processor can execute specialized I/O program residing in the memory without intervention of the CPU. Thus, CPU only needs to specify a sequence of I/O activity to I/O processor. The I/O processor then executes the necessary I/O instructions which are required for the task; and interrupts the CPU only after the entire sequence of I/O activity as specified by CPU have been completed. An advanced I/O processor can have its own memory, enabling a large set of I/O devices to be controlled without much involvement from the CPU. Thus, an I/O processor has the additional ability to execute I/O instructions which provide it a complete control on I/O operations. Thus, I/O processors are much more powerful than DMA which provides only a limited control of I/O device. For example, if an I/O device is busy then DMA will only interrupt the CPU and will inform the CPU again when the device is free while I/O device and once it has found to be free go ahead with I/O and when I/O finishes, communicate it to the CPU. The I/O processor is termed as channel in IMB machines.

In computer systems which have IOPs the CPU normally do not execute I/O data transfer instructions. I/O instructions are stored in memory and are executed by IOPs. The IOP can be provided with the direct access to the memory and can control the system bus. An IOP can execute a sequence of data transfer instructions involving different memory regions and different devices without intervention of the CPU.

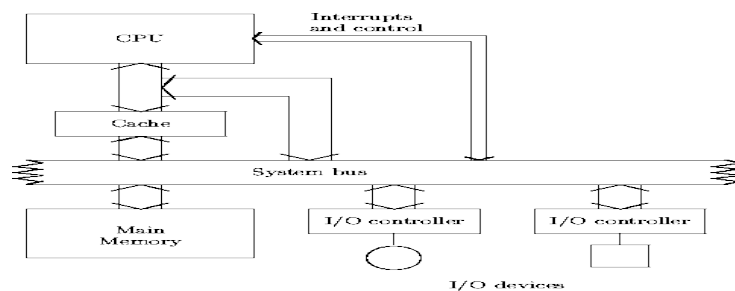


Fig 3.10

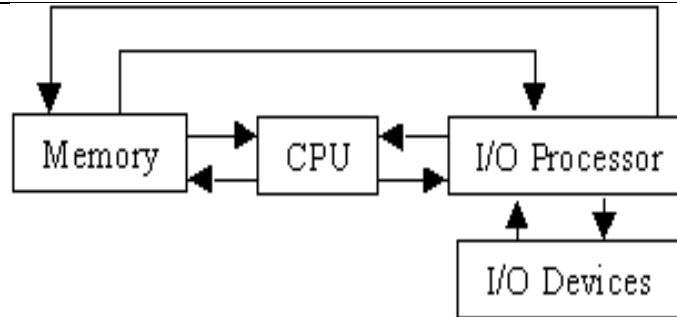


Fig 3.11

For those computers that have an I/O processor, the physical organization of I/O is similar to the other major functional areas: CPU and memory. I/O processors can vary from many pcb's that makeup a module/unit to a single pcb. Larger mainframe computers use the modular arrangement: multiple components on multiple pcb's that comprise one or more modules or units. Mini- and microcomputers use chassis or assemblies, cages or racks, and motherboard/backplane arrangements. Minis and micros use multiple components on one pcb or groups of pcb's (usually not more than seven) to form the I/O processor. The I/O processor controls the transfer of information between the computer's main memory and the external equipments. I/O processors are packaged two different ways: (1) IOC/IOA modules or multiple IOC/IOA pcb's, and (2) I/O pcb's. Regardless of the setup, computers with an I/O processor will use some sort of controller to regulate the signals in the I/O processor itself (includes IOC/IOA setup) and memory.

Unit-03/Lecture-06

8085 interrupt structure

There are five interrupt input TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR. TRAP is a nonmaskable interrupt, that is, it cannot be disabled by an instruction RST 7.5, 6.5, 5.5 and INTR are maskable interrupt i.e. they can be enabled or disabled by software. The 8085 A interrupt structure is shown in Fig. 6.1

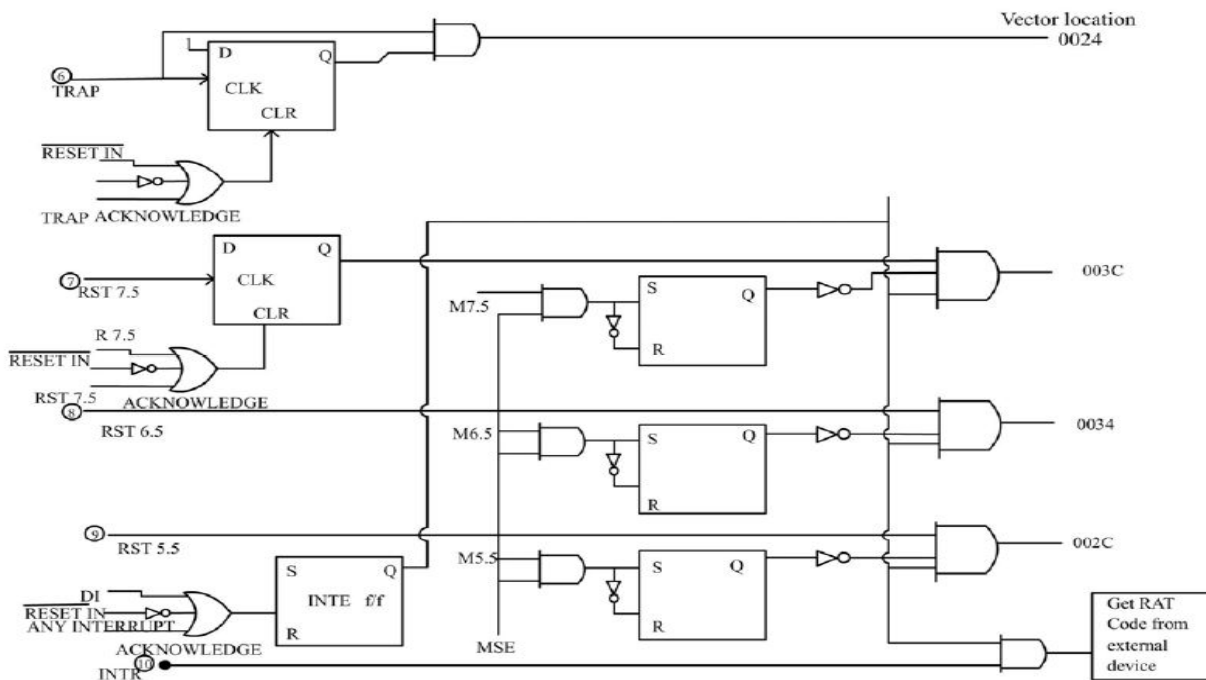


Fig 3.12

INTE F/F :

When the power is ON for the first time, signal goes Low. if resets the 8085. It also resets the INTE F/F. so that the entire interrupt structure is disabled. The INTE F/F can be SET or RESET using instructions. When INTE F/F is reset, except for TRAP no other interrupt signal can interrupt the μ P.

TRAP:

TRAP is a nonmaskable vectored interrupt. It can interrupt the μ P once the power is on. Most μ P interrupt inputs are level sensitive; however, some are edge sensitive and others are both edge and level sensitive. The TRAP input is both edge sensitive and level sensitive interrupt. It means that TRAP makes a low to high transition and remains high until it is acknowledged. The positive edge of the

TRAP signal will set the D flip flop. Because of the AND gate, however, the final TRAP also depends on a sustained high level TRAP input. This is why the TRAP is both edge and level sensitive. This also avoids false triggering caused by noise and transients.

For example, suppose the 8085 is midway through an instruction cycle with another 5 to completion. If a 300 ns noise spike hits the TRAP input, it will be edge triggered but not level triggered. The working on the current 8085 is still working on the TRAP input is both edge and level sensitive. The 8085 avoids responding to false TRAPs.

Since the TRAP input has the highest priority, it is used for catastrophic events such as power failure, parity errors, and other events that require immediate attention. In the case of brief power failure, it may be possible to save critical data with parity errors; the data may be resampled or corrected before going on. Whenever TRAP comes, μ P completes the current instruction, pushes the program counter on the stack and branches to fixed location 0054 H. Once the 8085A μ P recognizes a TRAP interrupt, it will send a high TRAP ACKNOWLEDGE bit to the TRAP F/F, thus clearing the F/F so that even if TRAP is high, it is not recognized only if it goes low, then high and remains high. The TRAP F/F is also cleared when μ P is being reset during which it goes low and clears the F/F.

RST 7.5, 6.5, & 5.5:

These are maskable vectored interrupts. These interrupts can be enabled or disabled through software. RST 7.5 has the highest priority among these & RST 5.5 has the lowest priority. RST 7.5 control signal input is rising edge

sensitive interrupt

whenever LOW to HIGH instruction occurs, it can interrupt the p. this LO to HI transaction is registered in second D-F/F. The output of this F/F is labelled I 7.5. whenever the other input are high the p recognize this interrupt this request is remembered until (1). the 8085 A responds to the interrupt (when interrupt is acknowledged, it sends a high RST 7.5 ACKNOWLEDGE bit to the clear input this clears it for future interrupts of D F/F. (2). or until the request is RESET by SIM instruction (R 7.5 bit is made high through SIM instruction and the F/F can be cleared). 2f is for the user to make use of these facilities. (3). or until the p is being reset ie signal becomes LOW whenever RST 7.5 is recognized, control is transferred to 003CH RST 6.5 & 5.5 are also vectored masked interrupts and are HIGH level sensitive interrupt control signal input. These are directly connected to AND gate the signal at these inputs must be maintained until the interrupt is acknowledged. Whenever RST 6.5 is recognized,

the control is transferred to 0034 H & whenever RST 5.5 is recognized, the control is transferred to 002CH. The signals I 7.5, 6.5 & 5.5 are called pending interrupts. the signal IE (bottom F/F) is called interrupt enable flag, it must be high to active the AND gates, Also notice the M 7.5, M 6.5 & M 5.5 signals, they must be low to enable the AND gates. e.g. to activate RST 7.5 interrupt, I 7.5 must be high, M 7.5 must be low and IE must be high. The interrupt enable F/F can be set or reset through software then F/F can be set using EI instruction. EI stands for enable interrupt whenever EI is executed, it produces a high EI bit and sets the INTE F/F and produces a high IE output.

This f/F can be reset in three ways.

1). When the power is on for the first time or signal goes low, it resets the INTE F/F so that that entire interrupt structure is disabled. When INTE F/F is reset except for TRAP no other interrupt can interrupt the p.

2). The INTE F/F can be reset using DI instruction. DI stands for disable interrupt when executed it produces a high DI bit & clear the INTE F/F

3). When the 8085 recognize an interrupt, it produces a high ANY INTERRUPT ACKNOWLEDGE bit. this disables

the interrupts.

INTR:

INTR is a maskable interrupt. A HIGH level on this pin interrupt the p.the interrupt signal input INTR is not affected by SIM instruction only INTE F/F must be SET to 1 before this interrupt comes. Thus, in 8085 we can write the LOGIC expression for the LOGIC variable VALID INT.

VALID INT = TRAP + INTE,[INTR+R75. +RST 6.5. +RST 5.5.]

	RGPV QUESTIONS	Year	Marks
Q.1	Classify instruction set of 8085	June 2014	2
Q.2	Explain the internal architecture of 8085 with a neat block diagram	June 2012	10
Q.3	Explain the different techniques used for interfacing I/O devices with 8085 processor .State the merits and demerits of each	Dec 2010	10
Q.4	Explain in brief the instruction set of 8085 microprocessor (Give only types of Instruction)	Dec 2011	7

Unit-03/Lecture-07

8085 INSTRUCTION SET

INSTRUCTION DETAILS DATA TRANSFER INSTRUCTIONS

Opcode	Operand	Description
Copy from source to destination		
MOV	Rd, Rs M, Rs Rd, M	This instruction copies the contents of the source register into the destination register; the contents of the source register are not altered. If one of the operands is a memory location, its location is specified by the contents of the HL registers. Example: MOV B, C or MOV B, M
Move immediate 8-bit		
MVI	Rd, data M, data	The 8-bit data is stored in the destination register or memory. If the operand is a memory location, its location is specified by the contents of the HL registers. Example: MVI B, 57 or MVI M, 57
Load accumulator		
LDA	16-bit address	The contents of a memory location, specified by a 16-bit address in the operand, are copied to the accumulator. The contents of the source are not altered. Example: LDA 2034 or LDA XYZ
Load accumulator indirect		
LDAX	B/D Reg. pair	The contents of the designated register pair point to a memory location. This instruction copies the contents of that memory location into the accumulator. The contents of either the register pair or the memory location are not altered. Example: LDAX B
Load register pair immediate		
LXI	Reg. pair, 16-bit data	The instruction loads 16-bit data in the register pair designated in the operand. Example: LXI H, 2034
Load H and L registers direct		
LHLD	16-bit address	The instruction copies the contents of the memory location pointed out by the 16-bit address into register L and copies the contents of the next memory location into register H. The contents of source memory locations are not altered. Example: LHLD 2040

Store accumulator direct
STA 16-bit address

The contents of the accumulator are copied into the memory location specified by the operand. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: STA 4350 or STA XYZ

Store accumulator indirect
STAX Reg. pair

The contents of the accumulator are copied into the memory location specified by the contents of the operand (register pair). The contents of the accumulator are not altered.
Example: STAX B

Store H and L registers direct
SHLD 16-bit address

The contents of register L are stored into the memory location specified by the 16-bit address in the operand and the contents of H register are stored into the next memory location by incrementing the operand. The contents of registers HL are not altered. This is a 3-byte instruction, the second byte specifies the low-order address and the third byte specifies the high-order address.
Example: SHLD 2470

Exchange H and L with D and E
XCHG none

The contents of register H are exchanged with the contents of register D, and the contents of register L are exchanged with the contents of register E.
Example: XCHG

Copy H and L registers to the stack pointer
SPHL none

The instruction loads the contents of the H and L registers into the stack pointer register, the contents of the H register provide the high-order address and the contents of the L register provide the low-order address. The contents of the H and L registers are not altered.
Example: SPHL

Exchange H and L with top of stack
XTHL none

The contents of the L register are exchanged with the stack location pointed out by the contents of the stack pointer register. The contents of the H register are exchanged with the next stack location (SP+1); however, the contents of the stack pointer register are not altered.
Example: XTHL

Push register pair onto stack

PUSH Reg. pair

The contents of the register pair designated in the operand are copied onto the stack in the following sequence. The stack pointer register is decremented and the contents of the high-order register (B, D, H, A) are copied into that location. The stack pointer register is decremented again and the contents of the low-order register (C, E, L, flags) are copied to that location.

Example: PUSH B or PUSH A

Pop off stack to register pair

POP Reg. pair

The contents of the memory location pointed out by the stack pointer register are copied to the low-order register (C, E, L, status flags) of the operand. The stack pointer is incremented by 1 and the contents of that memory location are copied to the high-order register (B, D, H, A) of the operand. The stack pointer register is again incremented by 1.

Example: POP H or POP A

Output data from accumulator to a port with 8-bit address

OUT 8-bit port address

The contents of the accumulator are copied into the I/O port specified by the operand.

Example: OUT 87

Input data to accumulator from a port with 8-bit address

IN 8-bit port address

The contents of the input port designated in the operand are read and loaded into the accumulator.

Example: IN 82

ARITHMETIC INSTRUCTIONS

Opcode	Operand	Description
Add register or memory to accumulator		
ADD	R M	The contents of the operand (register or memory) are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADD B or ADD M
Add register to accumulator with carry		
ADC	R M	The contents of the operand (register or memory) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the addition. Example: ADC B or ADC M
Add immediate to accumulator		
ADI	8-bit data	The 8-bit data (operand) is added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ADI 45
Add immediate to accumulator with carry		
ACI	8-bit data	The 8-bit data (operand) and the Carry flag are added to the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the addition. Example: ACI 45
Add register pair to H and L registers		
DAD	Reg. pair	The 16-bit contents of the specified register pair are added to the contents of the HL register and the sum is stored in the HL register. The contents of the source register pair are not altered. If the result is larger than 16 bits, the CY flag is set. No other flags are affected. Example: DAD H

Subtract register or memory from accumulator

SUB R
 M

The contents of the operand (register or memory) are subtracted from the contents of the accumulator, and the result is stored in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.
Example: SUB B or SUB M

Subtract source and borrow from accumulator

SBB R
 M

The contents of the operand (register or memory) and the Borrow flag are subtracted from the contents of the accumulator and the result is placed in the accumulator. If the operand is a memory location, its location is specified by the contents of the HL registers. All flags are modified to reflect the result of the subtraction.
Example: SBB B or SBB M

Subtract immediate from accumulator

SUI 8-bit data

The 8-bit data (operand) is subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.
Example: SUI 45

Subtract immediate from accumulator with borrow

SBI 8-bit data

The 8-bit data (operand) and the Borrow flag are subtracted from the contents of the accumulator and the result is stored in the accumulator. All flags are modified to reflect the result of the subtraction.
Example: SBI 45

Increment register or memory by 1

INR R
 M

The contents of the designated register or memory) are incremented by 1 and the result is stored in the same place. If the operand is a memory location, its location is specified by the contents of the HL registers.
Example: INR B or INR M

Increment register pair by 1

INX R

The contents of the designated register pair are incremented by 1 and the result is stored in the same place.
Example: INX H

Unit-03/Lecture-08

Data Transfer:

There are two primary data transfer methods in computers: serial and parallel. The distinction between the two used to be more important because common peripherals used one type or the other. Early printers, for example, used a parallel cable for faster data transmission. As Universal Serial Bus (USB) connectors have become faster with each new generation, serial data transfer has become the default option for connecting peripherals to computers.

When data moves between components within a computer, it's transmitted over a connection that is generically referred to as an "interface." A peripheral cable, like the USB cable connecting an external hard drive, is an example of such an interface. Depending on the interface, data can travel in one or both directions between devices. When data is only sent in one direction, with the receiving device signaling back whenever it receives data, the connection is called "serial" because data only goes out one step at a time. When data travels in both directions between devices, the connection is called "parallel" because there are multiple streams going back and forth.

Serial Transmission :

In telecommunication and computer science, serial communication is the process of sending data one bit at a time, sequentially, over a communication channel or computer bus. This is in contrast to parallel communication, where several bits are sent as a whole, on a link with several parallel channels.

Serial communication is used for all long-haul communication and most computer networks, where the cost of cable and synchronization difficulties make parallel communication impractical.

In a serial connection, the data are sent one bit at a time over the transmission channel. However, since most processors process data in parallel, the transmitter needs to transform incoming parallel data into serial data and the receiver needs to do the opposite.

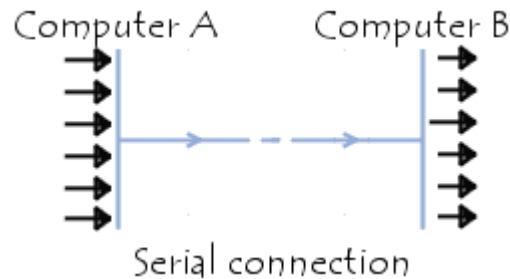


Fig 3.12 Serial Connection

Parallel Transmission

Parallel connection means simultaneous transmission of N bits. These bits are sent simultaneously over N different channels (a channel being, for example, a wire, a cable or any other physical medium). The parallel connection on PC-type computers generally requires 10 wires.

These channels may be:

N physical lines: in which case each bit is sent on a physical line (which is why parallel cables are made up of several wires in a ribbon cable)

one physical line divided into several sub-channels by dividing up the bandwidth. In this case, each bit is sent at a different frequency.

Since the conductive wires are close to each other in the ribbon cable, interference can occur (particularly at high speeds) and degrade the signal quality...

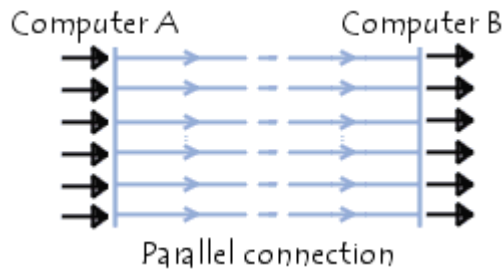


Fig3.13 Parallel Connection

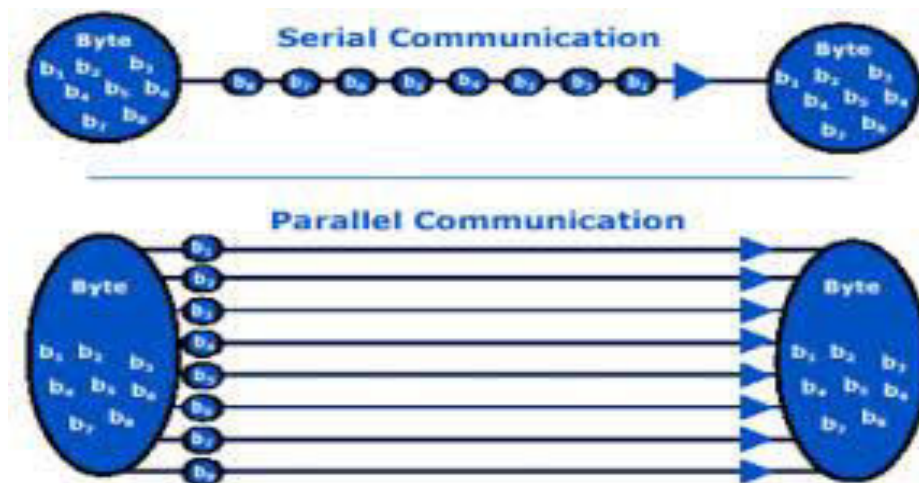
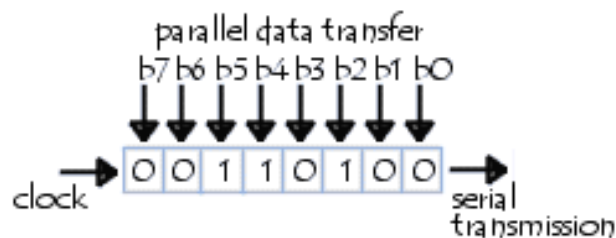
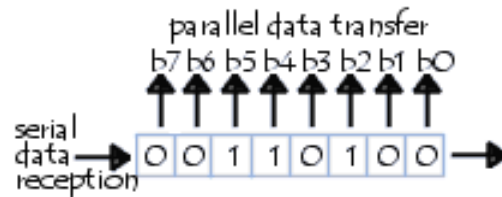


Fig 3.14

- The parallel-serial transformation is performed using a shift register. The shift register, working together with a clock, will shift the register (containing all of the data presented in parallel) by one position to the left, and then transmit the most significant bit (the leftmost one) and so on.



- The serial-parallel transformation is done in almost the same way using a shift register. The shift register shifts the register by one position to the left each time a bit is received, and then transmits the entire register in parallel when it is full:



	RGPV QUESTIONS	Year	Marks
Q.1	Define the following: Parallel versus Serial data transfer	Dec 2010	4

Unit-03/Lecture-09

Simplex Data Transmission

Simplex is one direction. A good example would be your keyboard to your CPU. The CPU never needs to send characters to the keyboard but the keyboard always send characters to the CPU. In many cases, Computers almost always send characters to printers, but printers usually never send characters to computers (there are exceptions, some printers do talk back). Simplex requires only one lane (in the case of serial).

Half-Duplex Data Transmission

Half-Duplex is like the dreaded "one lane" road you may have run into at construction sites. Only one direction will be allowed through at a time. Railroads have to deal with this scenario more often since it's cheaper to lay a single track. A dispatcher will hold a train up at one end of the single track until a train going the other direction goes through. The only example I could think of for Half-Duplex is actually a Parallel interface. Even though parallel is eight lanes, data travels through the lanes in the same direction at the same time but never in both directions at the same time. The IEEE-1284 allows printers to send messages to the computer. The printer cannot send these messages while the computer is sending characters but when the computer stops sending characters, then the printer can send messages back. It's kind of like some roads that head into downtown. In the morning, they're one way roads, allowing traffic to go into downtown. In the evening their one way roads, allowing traffic to head out of downtown. The only advantage that Half-Duplex would have is the single lane or single track is cheaper then the double lane or double track.

A half-duplex (HDX) system provides communication in both directions, but only one direction at a time (not simultaneously). Typically, once a party begins receiving a signal, it must wait for the transmitter to stop transmitting, before replying.

An example of a half-duplex system is a two-party system such as a walkie-talkie, wherein one must use "Over" or another previously designated keyword to indicate the end of transmission, and ensure that only one party transmits at a time, because both parties transmit and receive on the same frequency.

A good analogy for a half-duplex system would be a one-lane road with traffic controllers at each end, such as a two-lane bridge under re-construction. Traffic can flow in both directions, but only one direction at a time, regulated by the traffic controllers.

Half-duplex systems are usually used to conserve bandwidth, since only a single communication channel is needed, which is shared alternately between the two directions. For example, a walkie-talkie requires only a single frequency for bidirectional communication, while a cell phone, which is a full-duplex device, requires two frequencies to carry the two simultaneous voice channels, one in each direction.

In automatically run communications systems, such as two-way data-links, the time allocations for communications in a half-duplex system can be firmly controlled by the hardware. Thus, there is no waste of the channel for switching. For example, station A on one end of the data link could be allowed to transmit for exactly one second, then station B on the other end could be allowed to transmit for exactly one second, and then the cycle repeats.

Full-Duplex Data Transmission

Full-Duplex is like the ordinary two-lane highway. In some cases, where traffic is heavy enough, a railroad will decide to lay a double track to allow trains to pass in both directions. In communications, this is most common with networking. Our fiber optic hubs have two connectors on each port, one for each lane of a two-lane

roadway. Full-Duplex fiber is two cables bundled or tied together to form the two-lane roadway. In 100Base-TX, the two lanes are housed in the same jacket. RS232 was also designed to handle Full-Duplex but some of our short haul modems and converters give the user the option to go Half-Duplex or Simplex to reduce the number of conductors needed to connect between them.

A full-duplex (FDX) system, or sometimes called double-duplex, allows communication in both directions, and, unlike half-duplex, allows this to happen simultaneously. Land-line telephone networks are full-duplex, since they allow both callers to speak and be heard at the same time, with the transition from four to two wires being achieved by a hybrid coil in a telephone hybrid.

A good analogy for a full-duplex system would be a two-lane road with one lane for each direction. In full-duplex mode, transmitted data does not appear to be sent until it has been actually received and an acknowledgment was sent back by the other party.[citation needed]

Two-way radios can be designed as full-duplex systems, transmitting on one frequency and receiving on another. This is also called frequency-division duplex. Frequency-division duplex systems can be extended to farther distances using pairs of simple repeater stations, because the communications transmitted on any one frequency always travel in the same direction.

Full-duplex Ethernet connections work by making simultaneous use of two physical pairs of twisted cable (which are inside the jacket), where one pair is used for receiving packets and one pair is used for sending packets (two pairs per direction for some types of Ethernet), to a directly connected device. This effectively makes the cable itself a collision-free environment and doubles the maximum data capacity that can be supported by the connection.

There are several benefits to using full-duplex over half-duplex. Firstly, time is not wasted, since no frames need to be retransmitted, as there are no collisions. Secondly, the full data capacity is available in both directions because the send and receive functions are separated. Thirdly, stations (or nodes) do not have to wait until

others complete their transmission, since there is only one transmitter for each twisted pair.

Historically, some computer-based systems of the 1960s and 1970s required full-duplex facilities even for half-duplex operation, because their poll-and-response schemes could not tolerate the slight delays in reversing the direction of transmission in a half-duplex line.

A duplex communication system is a point-to-point system composed of two connected parties or devices that can communicate with one another in both directions. Translated literally, "duplex" means "two paths";[citation needed] thus, a duplex system has two clearly defined paths, with each path carrying information in only one direction – A to B over one path, and B to A over the other. There are two types of duplex communication systems: full-duplex and half-duplex.

In a full duplex system, both parties can communicate to the other simultaneously. An example of a full-duplex device is a telephone; the parties at both ends of a call can speak and be heard by the other party simultaneously. The earphone reproduces the speech of the remote party as the microphone transmits the speech of the local party, because there is a two-way communication channel between them, or more strictly speaking, because there are two communication paths/channels between them.

In a half-duplex system, there are still two clearly defined paths/channels, and each party can communicate to the other but not simultaneously; the communication is one direction at a time. An example of a half-duplex device is a walkie-talkie two-way radio that has a "push-to-talk" button; when the local user wants to speak to the remote person they push this button, which turns on the transmitter but turns off the receiver, so they cannot hear the remote person. To listen to the other person they release the button, which turns on the receiver but turns off the transmitter.

Duplex systems are employed in many communications networks, either to allow for a communication "two-way street" between two connected parties or to provide a "reverse path" for the monitoring and remote adjustment of equipment in the field.

Systems that do not need the duplex capability may instead use simplex communication, in which one device

transmits and the others can only "listen". Examples are broadcast radio and television, garage door openers, baby monitors, wireless microphones, and surveillance cameras. In these devices the communication is only in one direction.