

**Computer Architecture and Organization**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 45**  
**Data Transfer Techniques**




If we recall in our last lecture we talked about how we can select the IO device interfaces in the form of the IO ports for carrying out input and output for selecting the devices. In this lecture and the next few lectures we shall be talking about in some detail how the actual data transfer takes place between the IO devices and the processor and what are the different variations that are possible.

So, the topic of today's lecture is data transfer techniques.

(Refer Slide Time: 01:03)

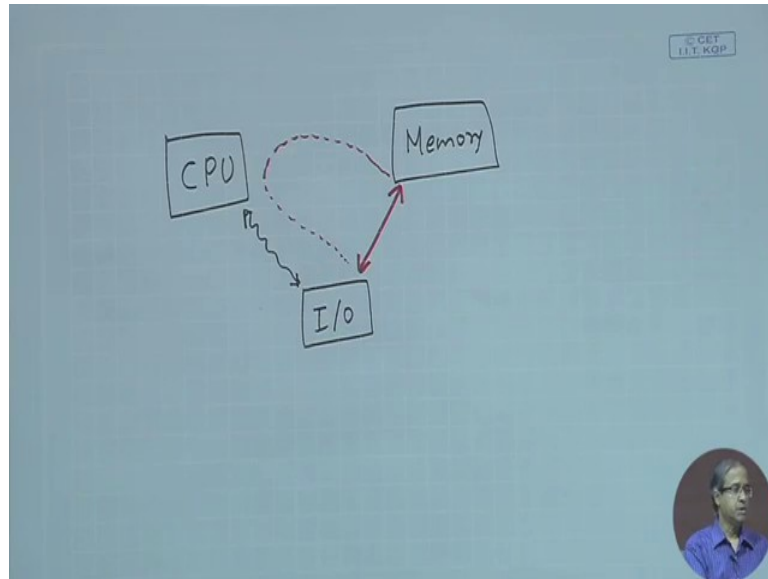
**Data Transfer Techniques**

1. **Programmed**: CPU executes a program that transfers data between I/O device and memory.
  - a) Synchronous
  - b) Asynchronous
  - c) Interrupt-driven
2. **Direct Memory Access (DMA)**: An external controller directly transfers data between I/O device and memory without CPU intervention.

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES |  NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Broadly speaking when we talk about data transfer techniques we basically talk about a scenario like this; we have the CPU, we have the memory, and we have the IO devices.

(Refer Slide Time: 01:12)



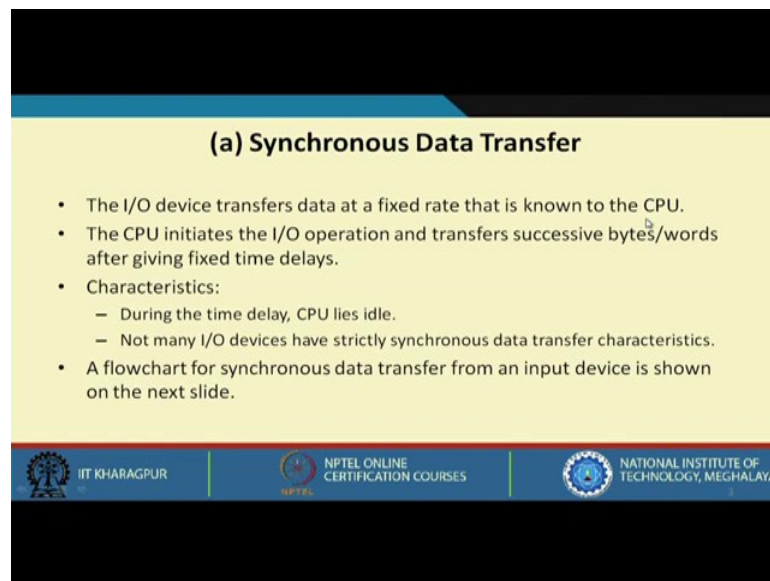
When we talk about data transfer you may be tempted to think that we are actually thinking about data transfer between CPU and IO, but it is not exactly that. Data transfer on input output transfer actually takes place between IO and memory devices. Because suppose it is an input device. the CPU after reading it will write into memory, and if it is an output device CPU will be reading some data from the memory and write it into that, because CPU is just a mediator, whatever data it is transferring that will be there in memory.

Under the data transfer technique the first broad approach can be called as programmed data transfer. Programmed means as the name implies CPU will be executing a program that transfers data between IO device and memory, meaning suppose it is an input device. Let us say we are using memory mapped kind of device interfacing, then CPU will have to execute a program that will be loading some data from the memory from that device, and it will be storing it into some memory location. Suppose there are 100 data this has to be done in a loop 100 times. So, there will be a program that will be running, and will be actually reading the data from the input device and will be writing the data into the memory.

Under programmed IO again there can be three variations, synchronous, asynchronous or interrupt driven. We shall be looking into the detail of this later. The other alternative is direct memory access or DMA; well here the CPU has very minimal intervention. CPU

does not execute any program, rather there is a separate external controller that will directly transfer data between IO device and memory without disturbing the CPU; CPU may be continuing with whatever it was doing. So, it will be directly transferring the data between the IO device and the memory. This is what DMA is; external controller directly transfers data between IO device and memory without CPU intervention.

(Refer Slide Time: 04:46)



**(a) Synchronous Data Transfer**

- The I/O device transfers data at a fixed rate that is known to the CPU.
- The CPU initiates the I/O operation and transfers successive bytes/words after giving fixed time delays.
- Characteristics:
  - During the time delay, CPU lies idle.
  - Not many I/O devices have strictly synchronous data transfer characteristics.
- A flowchart for synchronous data transfer from an input device is shown on the next slide.

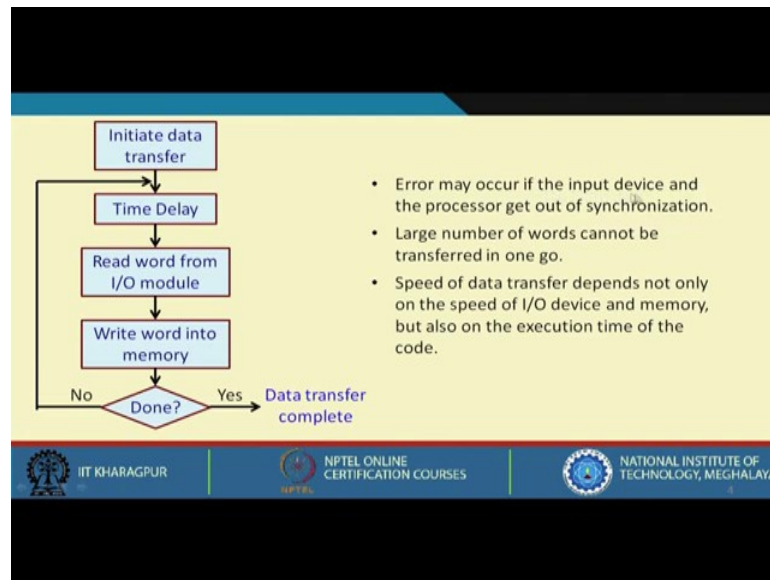
Logos: IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Let us look into these methods one by one. Synchronous data transfer ... as this name implies that there is some kind of a transfer going on, there is a sender, there is a receiver and both of them are synchronized. Synchronized roughly means both of them knows what is the speed of data transfer. Suppose I know some data is coming to me, I know what is the speed with which it is coming to me. So, I can pick up the data one by one with that speed.

This is what is mentioned here --- the IO device transfers data at a fixed rate that is also known to the CPU. So, here the CPU will be initiating the IO operation whenever it wants to, and after it does the CPU will know that the data will get transferred at fixed rate. Giving some fixed time delay the CPU can read or write the bytes or words one by one. Now there is some characteristic in this method that while the CPU is waiting with the time delay, it does not have anything else to do. CPU will lie idle; this is one major problem that you are having leading to very poor CPU utilization.

Second thing is that you will find very few IO devices have this kind of synchronous behavior. The data always coming at fixed speed is very rare you know. So, you will not find many devices like this.

(Refer Slide Time: 06:57)



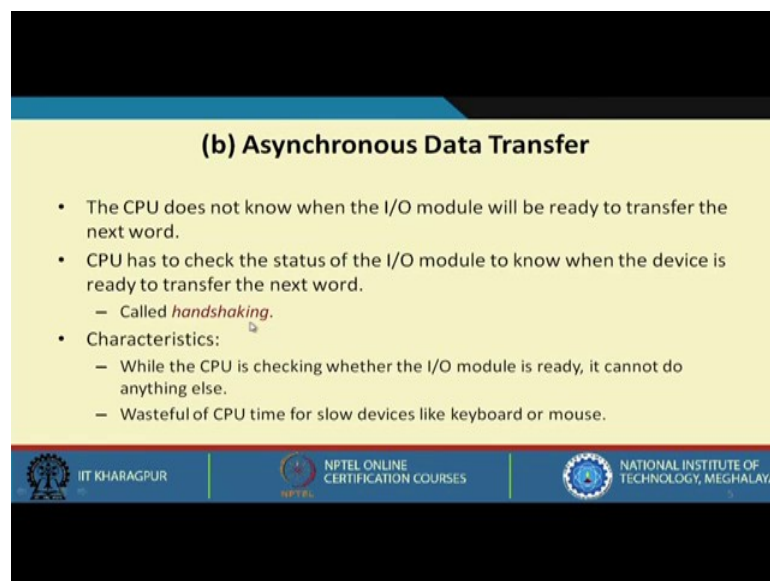
How this works I am showing it in a flowchart form here. The steps are roughly like this. The CPU initiates the data transfer, after initiating CPU will know that now the data will be coming. Suppose it is an input device; data will be coming at a fixed rate. So, CPU will give that amount of time delay, and it will read the next word from the IO device, it will write that word into memory, then it will check whether all the words have been transferred or not. If not it will again go back, again wait for a time delay, and read the next word. This will repeat and if it is done, the CPU will know that data transfer is complete it will continue with the next step.

Now, you see here the CPU is reading the data after giving fixed time delays because it knows the rate. If there is a mismatch in the speed, say due to some problem the IO device is not able to send the data at that fixed rate, then obviously CPU will be starting to read the wrong data because CPU is expecting the data to come every time  $t$ , but if the device is a little late, it may be reading the old data.

So, some error may occur if the input device and the processor get out of synch. This is one problem; because of the same reason large number of words cannot be transferred in one go, because chance of the two devices getting out of synchronization will be high.

And third thing is that you see there is a program that is executing. You are reading from some IO module, and writing into memory. If we think of the speed of data transfer, what is the maximum speed you can achieve? The maximum speed of data transfer does not depend only on the speed of IO device and memory, but also in the execution time of this code. This is true for all the programmed I/O techniques; the basic code where you are reading a word from I/O module and writing into memory, this will take some time and this will limit the maximum data transfer rate.

(Refer Slide Time: 09:39)



**(b) Asynchronous Data Transfer**

- The CPU does not know when the I/O module will be ready to transfer the next word.
- CPU has to check the status of the I/O module to know when the device is ready to transfer the next word.
  - Called *handshaking*.
- Characteristics:
  - While the CPU is checking whether the I/O module is ready, it cannot do anything else.
  - Wasteful of CPU time for slow devices like keyboard or mouse.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, let us move on to the next method asynchronous data transfer that is more practical, because you will not find many devices in real life that are strictly synchronous in nature. So, in asynchronous what are the characteristics?

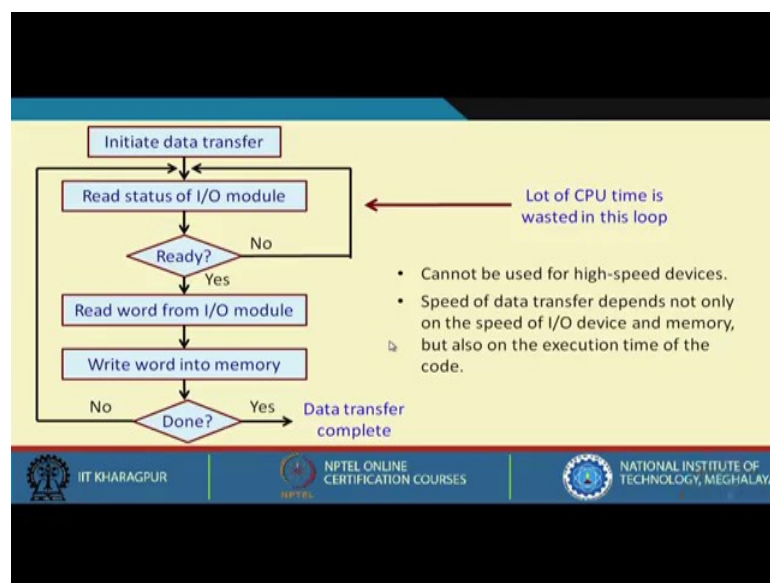
Here it is assumed that the CPU does not know or need not know when the IO module will be sending the next data. So, CPU does not know beforehand when the next data will be coming; next millisecond or may be after one hour. So, the CPU has to continuously check the status of the IO device whether it is ready. It is continuously asking the device are you ready are you ready are you ready, and as soon as the IO device says yes I am ready, the data transfer takes place.

You can see the CPU is wasting its time just asking the status of the device; are you ready are you ready. During this process the CPU is doing nothing useful, it is simply checking the status. CPU has to check the status of the IO module because CPU does not

know when the next data is coming; it will continuously have to check the status of the IO module and this process is called handshaking. So, CPU and the IO module are doing some kind of handshaking by virtue of which it comes to know when the device is ready to transfer.

While the CPU is checking the status of the IO module, it cannot do anything else it is just waiting. This is obviously wasteful of CPU time, particularly for slow devices like keyboard or mouse, because for the entire period this CPU is waiting to check the status.

(Refer Slide Time: 12:01)



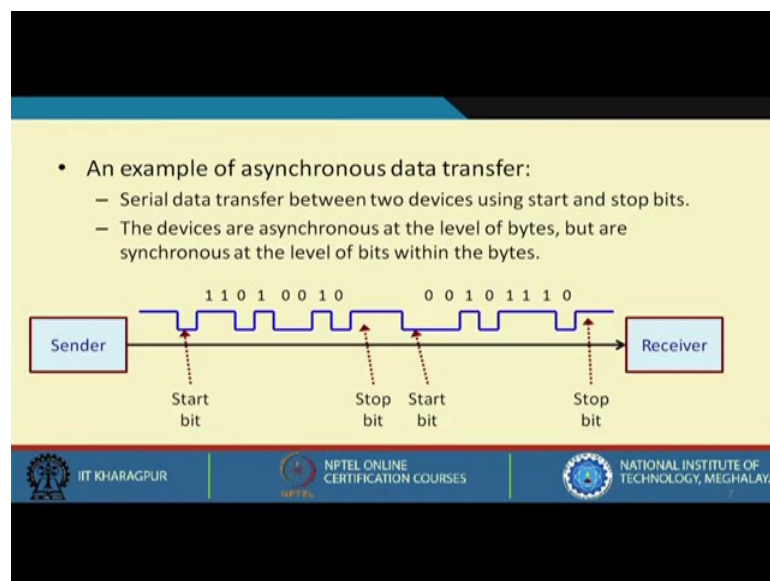
In terms of a flowchart the asynchronous data transfer works as follows. CPU will initiate data transfer, then CPU will read the status of the IO module to check whether it is ready or not; if it is not ready it will go on repeating this, it will continually check the status until it is ready. Now, as soon as it is ready it can read the word from the IO module and write the word in memory; the data transfer can be done and then it checks whether all the data have been transferred, if not it again goes back and starts this process, and if it is yes the transfer is complete and it proceeds with the next step.

Now, in this loop lot of CPU time is wasted because during this time the CPU is not doing anything useful just checking the status of the IO module whether it is ready. Clearly this method cannot be used for high speed devices and just like the previous method the speed of transfer depends on the execution time of the code and not only on the speed of IO device and memory.

So, synchronous and asynchronous data transfer methods we have seen they are quite similar, in one case the CPU does not wait for the IO device because CPU knows when the next data will be available. Suppose you take an analogy. I have asked you to do some work I know that you will take 10 minutes. I just come after ten minutes and take it from you, I do not ask you anything. And in asynchronous mode, I give you a task, and I continuously come every ten seconds and ask you whether you have done, you have done, you have done. I am continually engaged here; this is the main difference between these two.

Now let us move to something that is more practical, but before that let us see some examples of this asynchronous method. This asynchronous method or handshaking that we have talked about, this can be used not only for data transfer between CPU and some IO device, but also for data communication purposes.

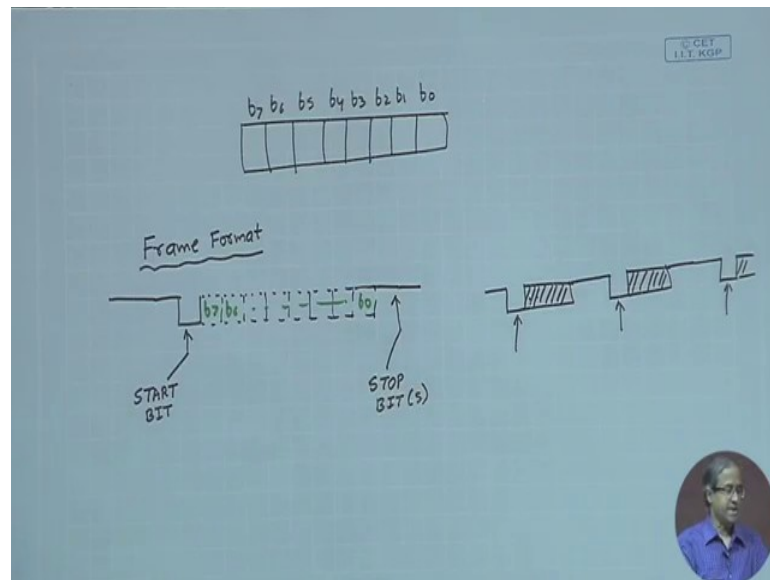
(Refer Slide Time: 15:00)



This example that we are showing is an example of asynchronous data transfer. Let me explain what is the meaning. There is a sender, there is a receiver --- sender will be sending a sequence of bytes to the receiver sequentially. We are assuming that the devices are asynchronous at the level of bytes, meaning the receiver does not know when the next byte will come; it can come immediately, it can come after five minutes, it can come after 10 minutes.

So, it is asynchronous at the level of the bytes, but once a byte is coming the bits within the bytes are synchronous. Because once a byte has started to come, within that byte the bits are coming with fixed delays that are synchronous. This is a very realistic assumption in communication systems.

(Refer Slide Time: 16:23)



Now, let us see how this data transfer is taking place. Suppose I have to transmit a byte; there are 8 bits, let us call them  $b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0$ . Now we define something called a frame format. What is a frame format? What we are saying is that this signal that we are transmitting will go zero at the beginning, and this zero is called the start bit. So, whenever this signal goes zero, the receiver will know that a new byte is coming. Once it has done this, the eight bits can be transmitted one after the other. So, here the 8 bits are transmitting let us say  $b_7$  first, then  $b_6 \dots$  up to  $b_0$ , and at the end I am leaving the line high for 1 or 2 cycles, I am calling this as stop bit, or stop bits.

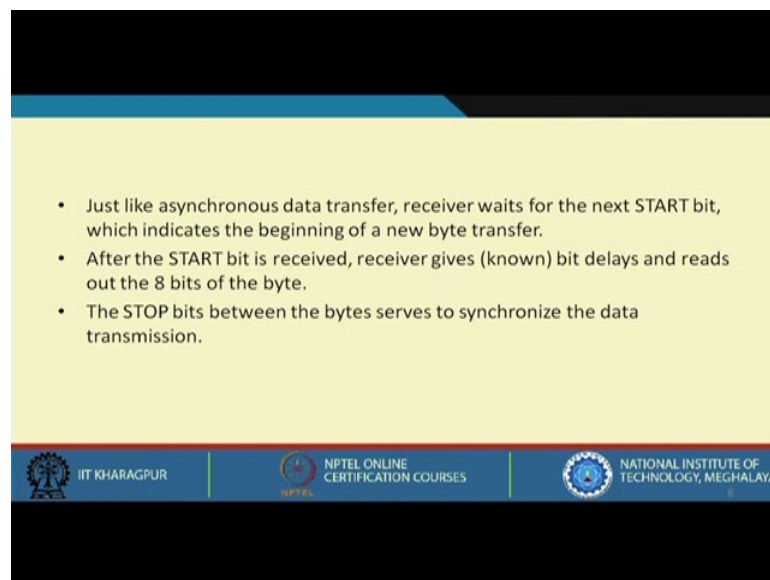
When I am transmitting a number of such bytes there will be a start bit, there will be a stop bit. How will it look like? It will begin with a start bit, then the bits of the byte will come, then the stop bit, then again a start bit will come then again the byte then again the stop bit, and so on. The receiver will always be looking for the start bits again. So, once a start bit is found, it will know that immediately the next 8 bits of the data bits are coming in fixed intervals of time. So, after some fixed delays it can read them out.

Exactly this thing is shown in the slide. The sender is transmitting two bytes one by one.



The start bit is a way of indicating that a new byte is coming. So, the receiver continuously checks whether the start bit is coming. Once it is found that it will simply read the next 8 bits of the byte. This is a very classical example where asynchronous IO is used.

(Refer Slide Time: 20:41)



- Just like asynchronous data transfer, receiver waits for the next START bit, which indicates the beginning of a new byte transfer.
- After the START bit is received, receiver gives (known) bit delays and reads out the 8 bits of the byte.
- The STOP bits between the bytes serves to synchronize the data transmission.

This is what I have just said. The receiver will wait for the next start bit that will indicate the beginning of the next byte transfer and once the start bits is received, the receiver knows the data rate, it will give appropriate bit delays and read the 8 bits. And the stop bits between the two bytes serve the purpose of synchronization, because if you do not give the stop bits then one byte and the next byte might be joined together and the receiver might get confused. So, it is always good to give some gap in between, and again resynchronize at the next start bit. So, sender and receiver will get resynchronized.

(Refer Slide Time: 21:32)

**(c) Interrupt-Driven Data Transfer**

- The CPU initiates the data transfer and proceeds to perform *some other task*.
- When the I/O module is ready for data transfer, it informs the CPU by activating a signal (called *interrupt request*).
- The CPU suspends the task it was doing, services the request (that is, carries out the data transfer), and returns back to the task it was doing.
- Characteristics:
  - CPU time is not wasted while checking the status of the I/O module.
  - CPU time is required only during data transfer, plus some overheads for transferring and returning control.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

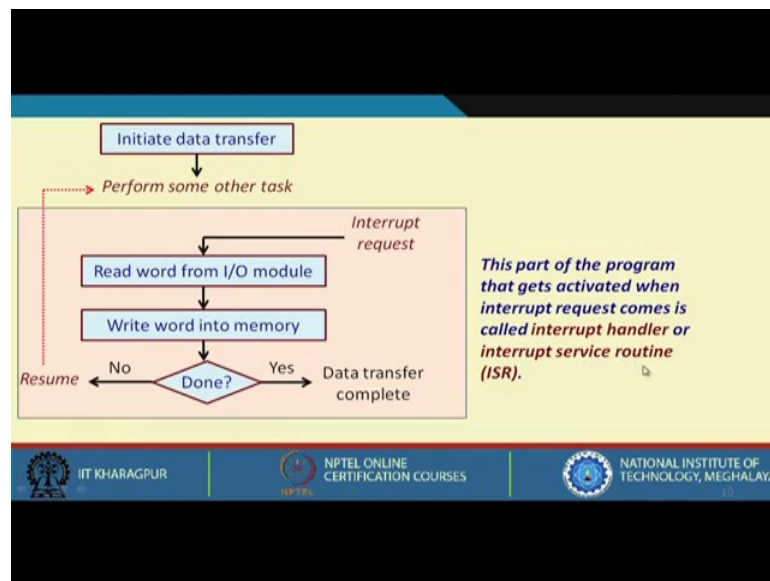
The third and most important kind of data transfer technique that we discuss is called interrupt driven data transfer. Let me first tell you the analogy. Say I come and give you a task, I ask you do this, but I am not asking you continuously that whether you have finished whether you have finished. I give you a task, I go back and I start doing my own work; something else I am doing. You finish your task and as soon as you are done, you come to me and tell me that you have finished. Whatever I was doing I will keep it, I will see what you have done, then I will again come and resume whatever I was doing. This is the basic idea behind interrupt driven data transfer or interrupts in general.

Whatever I have said in the terminology of computers it is like this. The CPU initiates the data transfer and proceeds to perform some other task. It is not sitting idle, it is doing some other task. When the IO module is ready for data transfer, it informs the CPU by activating a signal called interrupt request; like someone is coming to the CPU and telling that well I am ready.

When the interrupt request comes, the CPU that was doing some other task will suspend that task temporarily, service the request -- whatever interrupt request is coming. In this case it is for a data transfer, so it will do the data transfer and it will return back to that some other task it was doing. It is similar to a subroutine or a function call and return; whenever interrupt comes I go somewhere, I finish it and then again I come back.

Some of the characteristics here are that the CPU time is not wasted while checking the status of the IO module, because we need not check the status. The device itself will come and tell me that I have finished. CPU time is required only to carry out the data transfer plus of course, some overhead for transferring and returning of the controls. Stopping the process, going there and again coming back this extra overhead is there, but in addition to that the CPU time can be utilized in a much better way. CPU can do something more fruitful, some other task.

(Refer Slide Time: 24:36)



Pictorially let us see how it looks like CPU initiates the data transfer and after initiating the data transfer CPU is performing some other task. The device when it is ready, the IO interface will send an interrupt request to the CPU, interrupt request will tell the CPU that I am finished. Then CPU will temporarily suspend whatever it was doing, it will read word from IO module, write or do the data transfer, check whether all the words have been transferred, if it is yes done - if no it will resume this other task whatever it was doing and it will be waiting for the next interrupt request to come. Next interrupt request means another data transfer fine.

The part of the code that gets activated whenever interrupt request comes, in this case a simple data transfer, is called interrupt handler or interrupt service routine (ISR). Here since we are more concerned about IO transfer, we are talking about interrupts for IO, but interrupt is a general concept. Interrupts can come for other reasons also, like

interrupt may be coming from a timer in a time sharing operating system, interrupt may be coming from internal to the processor like there is a division by zero, or there is a power failure; the power is falling there is a sensor; there are so many sources of interrupts.

Depending on that exactly what you do inside your interrupt handler or ISR may be different, but here because we are concerned only for IO transfers. So, the ISR's sole tasks will be to transfer the data.

(Refer Slide Time: 27:12)

**Some Features of Interrupt-Driven Data Transfer**

- How is ISS different from a normal subroutine or function?
  - A function is called from well-defined places in the calling program.
    - Only the relevant registers need to be saved on entry to the function, and restored before return.
  - The ISS can get invoked from *anywhere* in the program that was executing.
    - Depends on when the interrupt request signal arrived.
    - So potentially all the registers that are used in the ISS needs to be saved and restored.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Some features of interrupt driven data transfer are as follows. There is some similarity with function or a subroutine call; when an interrupt request comes you stop what you are doing, you go somewhere else, and then again come back. In a function call also you do something like that. You go to a function, do it and come back, but there is one difference. For a function call you actually make a function call from somewhere in the program. You know that in a program from this place are making a function call. But here you do not know when the interrupt request will come; it can come here, it can come here, it can come here. So, while your program is executing your interrupt request can come potentially anywhere. Wherever it comes you will have to stop, you will have to go there, finish it and then again come back.

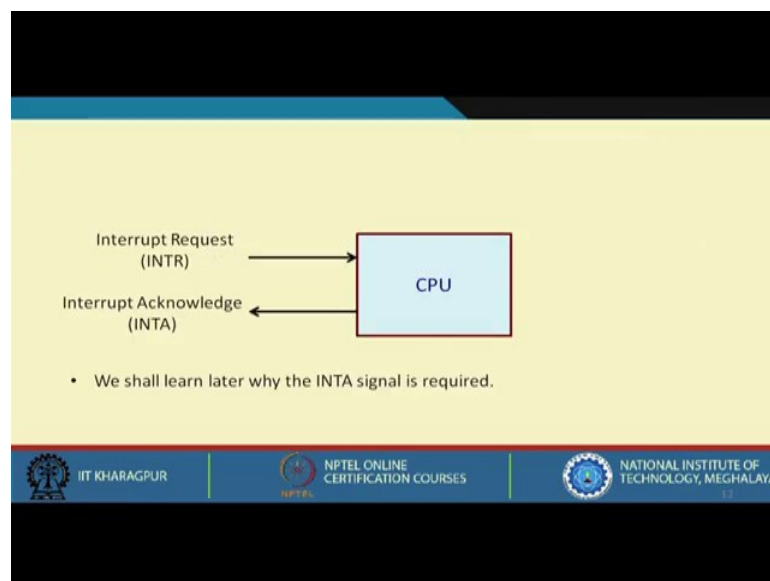
Just as I said for a function it is called from well defined places in the calling program that is actually coded and only the relevant registers need to be saved. Whatever registers

you are using you need to save only those registers and return, because the function might be destroying some of the registers your main program or the calling program was using. So, you should make sure that those registers are not destroyed.

But in contrast the ISS or ISR can get invoked from anywhere in the program that was executing --- this anywhere means it depends on exactly where the interrupt signal arrived. So, potentially all the registers that are used in the service routine needs to be saved and restored.

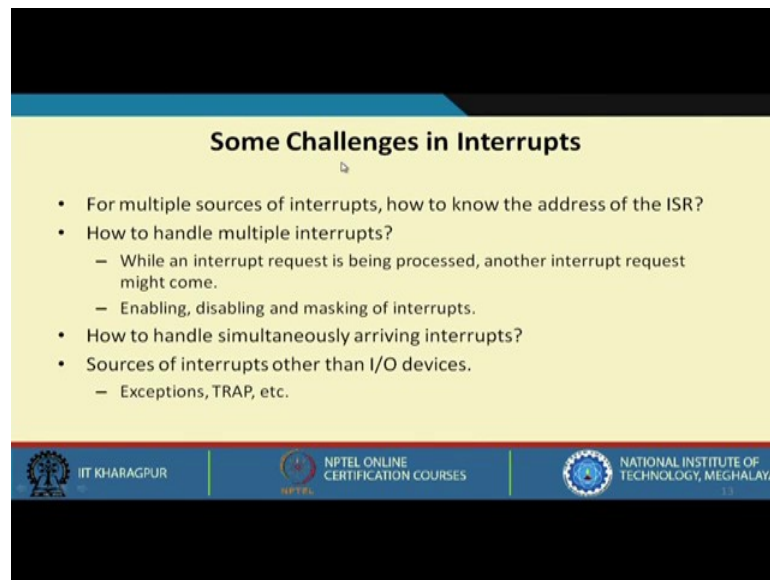
What I mean to say is that because you do not know where in the program your interrupt might come, and in the program you may be requiring a large number of registers in different parts, so potentially you may have to save and restore all those registers. The overhead of saving and restoring registers in interrupt can be much higher.

(Refer Slide Time: 29:51)



Pictorially it looks like this. This is a processor, the interrupt request comes from some device or some external source. Processor generates an interrupt acknowledge and why this is required we shall be discussing later, because so far we have not talked about interrupt acknowledge. Now there are some challenges in interrupts, and these we shall be dealing with in some detail in our subsequent lectures.

(Refer Slide Time: 30:22)



**Some Challenges in Interrupts**

- For multiple sources of interrupts, how to know the address of the ISR?
- How to handle multiple interrupts?
  - While an interrupt request is being processed, another interrupt request might come.
  - Enabling, disabling and masking of interrupts.
- How to handle simultaneously arriving interrupts?
- Sources of interrupts other than I/O devices.
  - Exceptions, TRAP, etc.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

First thing is that if there are multiple devices interrupting, how to know who has interrupted and how to know what will be the address of the interrupt service routine. Because the service routine for the disk may be different from the service routine of the keyboard, which may be different from the service routine for the mouse, so I have to know which device has interrupted and where to go and how to know this.

Secondly how to handle multiple interrupts. What I am trying to say is that while an interrupt request is being processed let us say another interrupt request has come, so what do you do?

There are many alternatives we shall be discussing. This may require enabling, disabling or masking of the interrupt system. Next there can be simultaneously arriving interrupts, so how to handle. Here you have to do something regarding prioritization of the interrupts; you have to define interrupt priorities. And lastly there can be interrupts that can come from some other device, not only IO devices, like exceptions, etc.

With this we come to the end of this lecture. In our next lecture we shall be continuing with our discussion on interrupt handling and interrupt processing. We will look into the different details exactly what are the issues and what are the possible solutions we can use there.

Thank you.