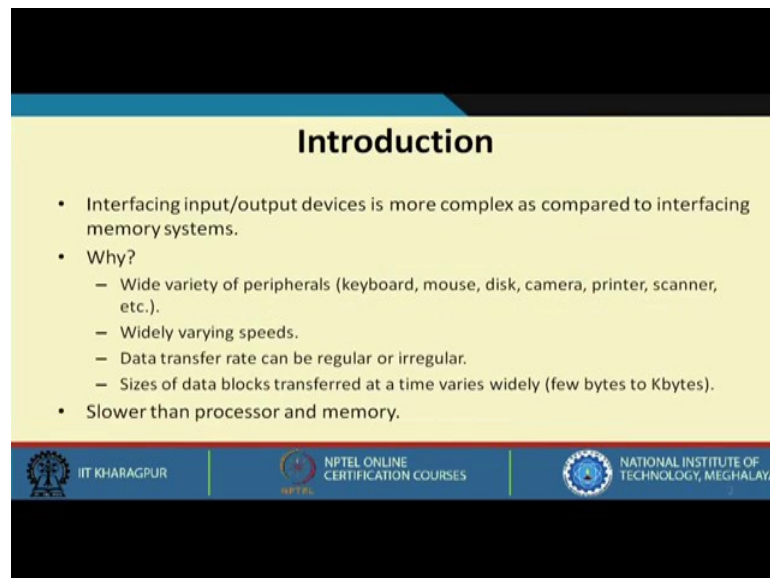


Computer Architecture and Organization
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 44
Input - Output Organization

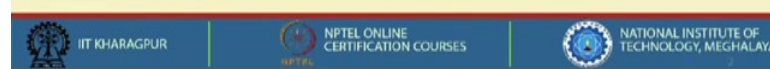
In this lecture we start our discussion on Input Output Organization. Now, we shall be starting to talk about how input output devices can be interfaced to the computer system and what are the different ways in which you can transfer data to and from the input output devices.

(Refer Slide Time: 00:47)



Introduction

- Interfacing input/output devices is more complex as compared to interfacing memory systems.
- Why?
 - Wide variety of peripherals (keyboard, mouse, disk, camera, printer, scanner, etc.).
 - Widely varying speeds.
 - Data transfer rate can be regular or irregular.
 - Sizes of data blocks transferred at a time varies widely (few bytes to Kbytes).
- Slower than processor and memory.



Let us see why this input output interfacing is so important. The first thing to note is that interfacing I/O devices is more complex as compared to interfacing memory systems. When we talk about interfacing memory systems which you saw earlier, there can be static RAM there can be dynamic RAM, well static RAMs are faster than dynamic RAM you know, but when you interface the speeds of the memory systems are known to us, they are not that widely varying. You see normally static RAMs are used to build the cache memory, dynamic RAMs are used to build the main memory.

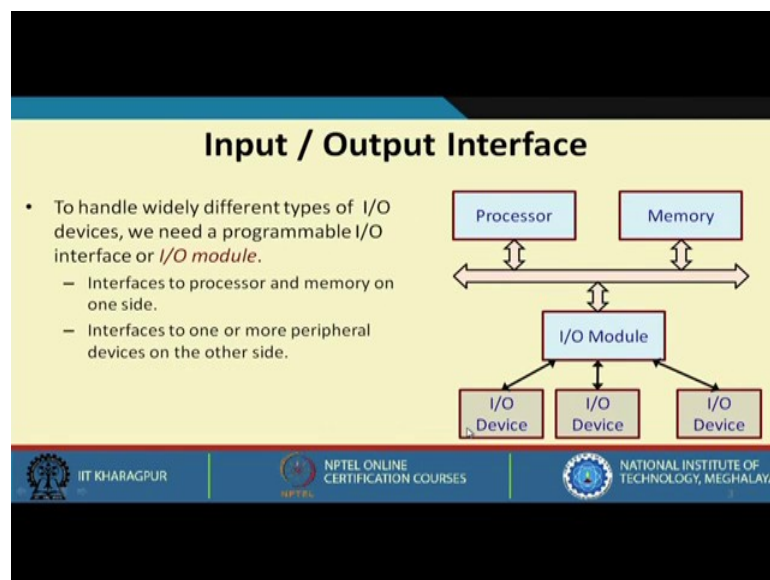
So, when you are designing the interface for the main memory, it is all dynamic RAMs, their speeds are all very uniform. The way CPU transfers data to and from memory are also very similar. But you think of the I/O devices we have very slow devices like a

keyboard or a mouse where the computer has no idea when the next input will be given, as a user you can press the next key immediately or you can even press it after one hour. But when you are interfacing a disk or a some kind of high speed I/O devices, then the data might be coming at a much faster data, e.g. scanner or camera. So, the data transfer speed varies widely from one device to another, and their characteristics are also widely different.

That is what is mentioned here. The main reason is the wide variety of peripherals that are existing starting from keyboard and mouse very slow ones, disk camera the faster ones, printer is also not very fast, it comes in between scanner. There are so many devices, they are widely varying in speeds. Not only that the data transfer characteristics can be regular or irregular; like when you think of a camera which is continuously sending you data, the data transfer is in some way regular.

But for the other kind of device like scanner, its scans one page then it stops, then after a while it scans another page then stops. So, it is irregular in some sense, and the size of the data block depends very much on the type of the I/O device.

(Refer Slide Time: 04:33)



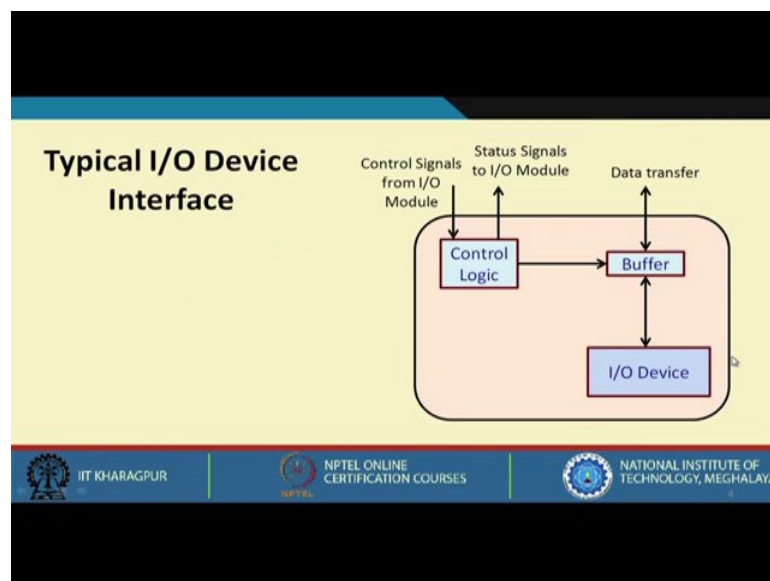
Size of data block transferred can vary widely, it can be a few bytes it can even be few kilobytes. Let us say a disk it can transfer a sector or a block (collection of sectors). In general I/O devices are slower than both processor and memory.

Just a schematic diagram we are showing here which tells you something about the input output interface.

You see here we have the main processor bus that connects the processor with the memory. This is the most important and the highest speed components in the system, then we have the interface called I/O module. You need a separate I/O module; you cannot afford to connect all the I/O devices directly to this bus, because of the reason that you mentioned these I/O devices have so widely varying characteristics. This I/O module in some way can take care of these variations, and it can present a uniform interface to this bus.

The I/O module on one side interfaces to the processor and memory through this bus, and on the other side it interfaces to one or more peripheral or I/O devices.

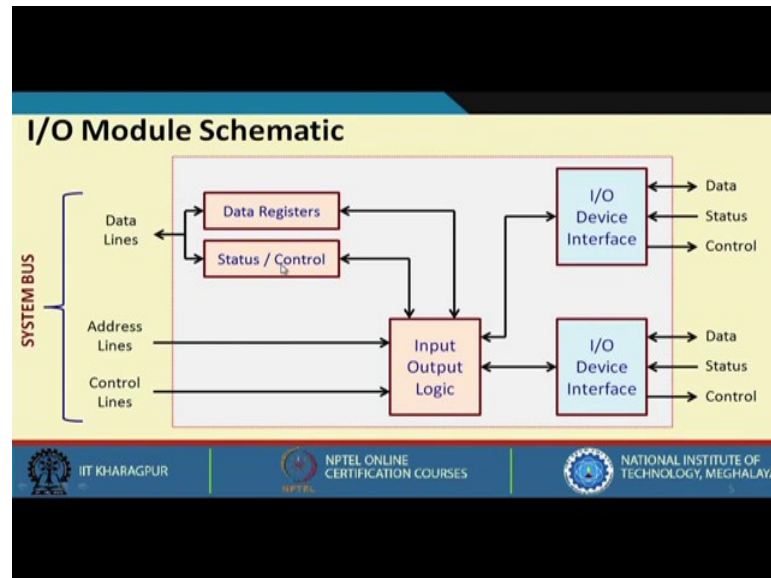
(Refer Slide Time: 05:55)



In typical I/O device interface shown in this diagram, when you say this I/O device is connected to the I/O module, exactly how the interface is like. See the I/O module signals are here. The I O module can send some control signals, it can receive some status signals, or it can do some data transfer. In the previous diagram these arrows which I show will carry all these three kinds of signals; control signals, status signals and data transfer.

Now, inside the I/O device interface in addition to the I/O device we have a buffer which temporarily stores the data that is been transferred, and there is a control logic, which controls the buffer and there will be other circuitry also. This status signal will tell you whether the I/O device is ready, whether the next data word or data byte is ready, and so on and so forth. This is a typical I/O device interface.

(Refer Slide Time: 07:06)



Now, the I/O module if you look at ,overall it looks like this. On one side you have the I/O devices. For each of the I/O devices if you again look at the previous diagram, three kind of signals are required: control, status and data. Each of the I/O device interface has control, status and data. And other side you have the system bus, where there is an address bus, data bus and also the control bus. The address bus will provide you with the address of the I/O device, which I/O device you are trying reading or writing. So, the I/O logic will select that appropriate device and the control lines will tell you exactly what is the operation being carried out.

Depending on that the data line will be carrying either some data or some status or control. The CPU may want to control the device, send some control signal, or it may want to read the status of the device. This is during the configuration phase and during the data transfer phase the same line will be sending or receiving data.

(Refer Slide Time: 08:36)

Typical Steps During I/O

- Processor requests the I/O Module for device status.
- I/O Module returns the status to the processor.
- If the device is ready, processor requests data transfer.
- I/O Module gets data from device (say, input device).
- I/O Module transfers data to the processor.
- Processor stores the data in memory.

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

This is roughly how the I/O module looks like. Now, typical steps during the I/O operation. The first step can be that the processor requests the I/O module for device status, because the processor wants to do some kind of I/O. The processor is not sure whether the device is ready to carry out the operation.

After sometime the I/O module will return back the status to the processor, whether the device is ready or not. Once the processor knows the status of the I/O device, if it is ready then the processor initiates the data transfer, read or write. Suppose it is a input device. Now, this input data transfer will be initiated and I/O module will be getting the data from the device, this data will be sent to the processor, which will store the data in memory. This process will repeat. This is how typically the data transfer takes place between the I/O module and the I/O devices via the processor.

So, you see the processor needs to periodically check the status of the I/O device, that is very important. Whether the device is ready; if it is ready only then I will try to read or write, otherwise I will wait because you have to understand the device is much slower than cpu. cpu may be requesting very fast, but the I/O device does not have that speed, it is much slower. So, cpu will have to wait till the I/O device is ready, only then it can read or write the data.

(Refer Slide Time: 10:38)

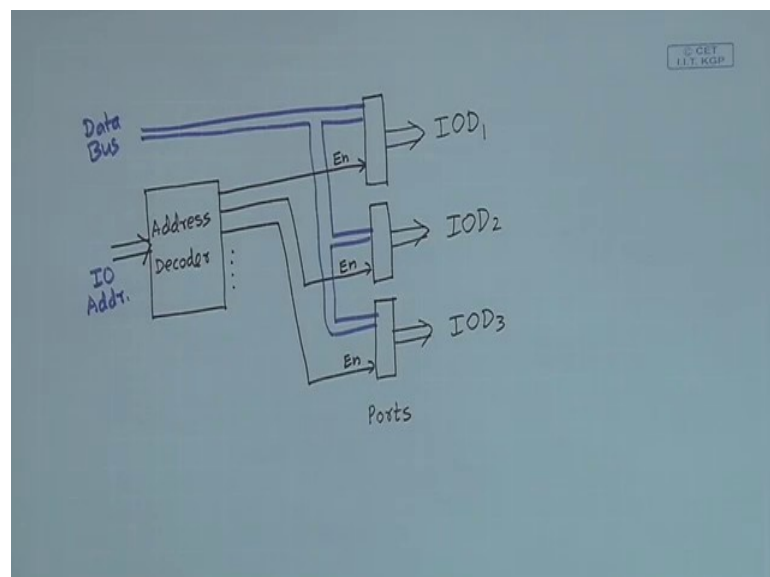
How are I/O devices typically interfaced?

- Through input and output ports.
- Output port:
 - Basically a PIPO register that is enabled when a particular output device address is given.
 - The register inputs are connected to the data bus, and the register outputs are connected to the output device.
- Input port:
 - Basically a parallel tristate bus driver that is enabled when a particular Input device address is given.
 - The driver outputs are connected to the data bus, while the inputs are connected to the input device.

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

Now, let see how the I/O devices are typically interfaced. The interface is through something called input or output ports. Now what is an output port? I will give an example in next slide. Output port is nothing but a parallel-in parallel-out register which is enabled when a particular output device address is given. The idea is suppose I have several I/O devices.

(Refer Slide Time: 11:21)



Let us say all are output devices. I will be having several I/O ports. On this side I am connecting the I/O devices, this is I/O device 1, I/O device 2, I/O device 3, and I am

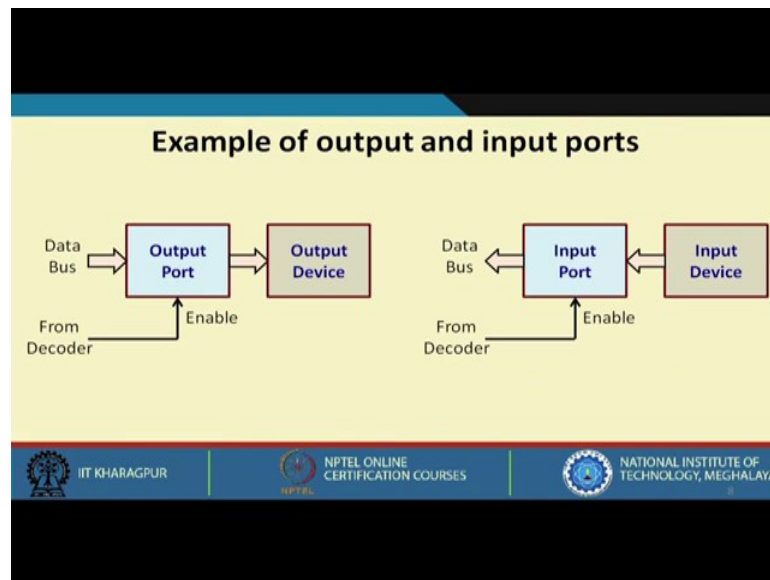
selecting one of these ports at a time. How I am selecting? There will be some kind of an address decoder just like you used an address decoder for selecting memory modules, this is something very similar to that. Some address of an I/O device will be applied, the address decoder output will be selecting one of these I/O ports, these are the enable lines.

One of the I/O port will be enabled and on the other side you have the data bus connection. Roughly speaking an output port looks like this. Data bus is there, this is your I/O address, the address of the I/O device. The address decoder decodes the address, it will be selecting one of the ports, and the port which is selected the data on the data bus will get stored there, and the I/O device will be getting that if it is an output device. So, this is about the output port.

The output port are basically a register that is enabled when the particular output device is given. Now you may ask why you need a register ? We need a register because the I/O devices are much slower. The cpu will be writing something into the port, it will be stored in that register, and the device can take its own sweet time to take the data from that register. But if the CPU just writes it and then withdraws, the device may not be able to read the data within that short time, that is why we need a register to store the data.

On the other hand for input devices you can have input ports. For an input port you do not need a register, you need basically a tristate bus driver. Just a set of drivers with tristate control which will be enabled when a particular input device address is given, using that same kind of an address decoder.

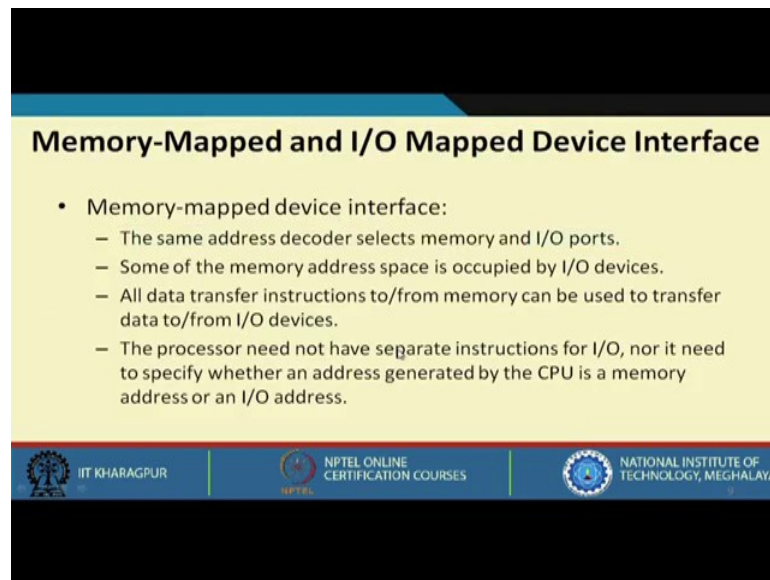
(Refer Slide Time: 14:48)



The driver outputs are connected to the data bus, inputs are connected to the input device. For an output port the diagram I showed this is for one device. There is one output port, which is connected to one output device, and from the address decoder the enable of the output port is activated. For an input port it is similar, but it is not a register it is just a tristate buffer. Input device is sending the data and address, decoder again is decoding, and enabling it. When it is enabling this data will be available on the data bus because it is a buffer.

You see for an input port you do not need a register, a tristate buffer is sufficient. Why? Because the I/O devices are much slower. So, whenever the I/O devices are ready, the input device will be providing the data to the input side of the input port, and whenever cpu wants to read it, it will be enabling the tristate buffer. The data will come on the data bus it will read from the data bus. So, the device need not store it in a register. The buffers are much simpler to build than register, that is why you can save some hardware.

(Refer Slide Time: 16:13)



Memory-Mapped and I/O Mapped Device Interface

- Memory-mapped device interface:
 - The same address decoder selects memory and I/O ports.
 - Some of the memory address space is occupied by I/O devices.
 - All data transfer instructions to/from memory can be used to transfer data to/from I/O devices.
 - The processor need not have separate instructions for I/O, nor it need to specify whether an address generated by the CPU is a memory address or an I/O address.

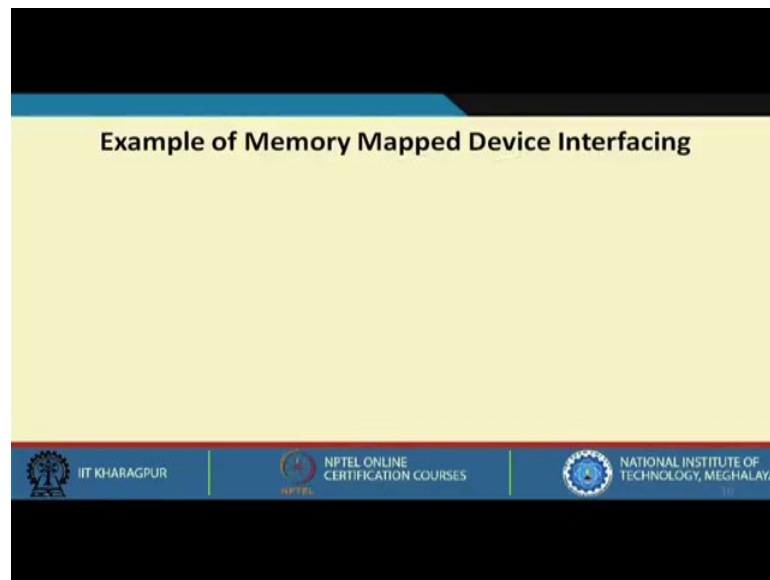
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Now, let us look at two alternate ways of connecting the I/O devices to the address and data buses. There is something called memory-mapped device interface, there is something called I/O-mapped device interface. What is memory mapped device interface? Here the same address decoder is selecting both memory and the I/O ports and there is no separate decoder for memory and I/O devices. The same single decoder is there and some of the memory address space is eaten up by the I/O devices. I/O devices occupy some of the memory addresses, and you cannot use the full capacity of memory here, some of the memory space is occupied by the I/O devices.

And for data transfer from the input devices or the output devices on the processor, you do not have separate instructions. You treat the I/O devices as memory. For example, in the MIPS32 architecture you can use load and store instructions to load data from the memory or store data to the memory. In a same way if we use a memory address that represents an I/O device, you can load data from that address means I am reading the data, or you can store data into that address means I am writing the data to the device.

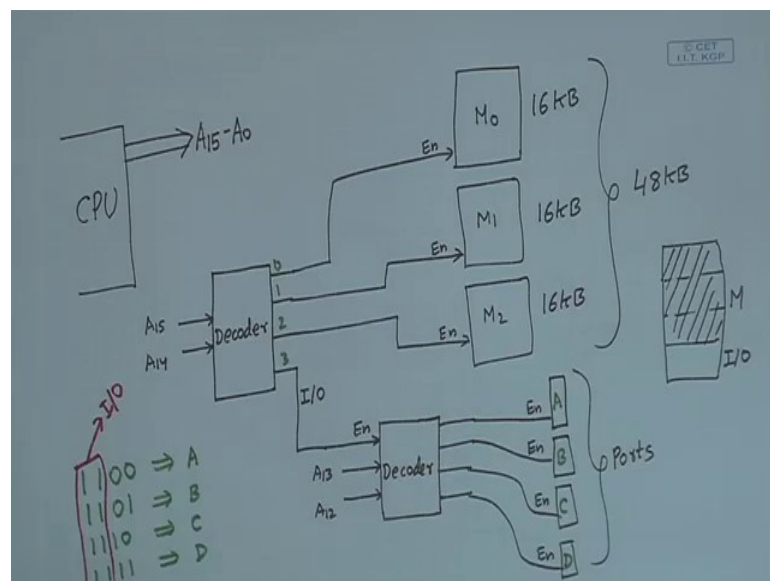
So, reading data from the input device or writing data into the output device you can do using the same load and store instructions. Here as I had said the processor need not have separate instructions for I/O, the processor need not specify whether the address is for memory or I/O, because the decoder will take care of it.

(Refer Slide Time: 18:39)



Let us explain through an example. We shall work out an example.

(Refer Slide Time: 18:43)



Suppose I have some memory modules. These are three memory modules, let us call M_0 , M_1 and M_2 . Suppose on this side I have the processor which is generating a 16-bit address A_{15} to A_0 . So, the total addressable memory will be 2 to the power 16, which means 64 kilobytes.

Let us assume that the memory modules are each 16 kilobytes in size. So, that the total memory that I am connecting is 48 KB. What I am doing is I am using an address

decoder. The address decoder will be fed with the two highest orders bits of the memory address, there will be 4 outputs, the first 3 will be selecting one of these 3 memory modules, this will be the enable.

What we do with the fourth one? What we are saying is that the fourth one we are reserving for I/O, we are saying that in our total memory map if I say this is our total memory map if I divide up into 4 parts, the first three parts we are keeping for memory the remaining quarter here we are reserving for I/O. So, what you do to this I/O? Let us use another decoder here, this is also 2 to 4 decoder.

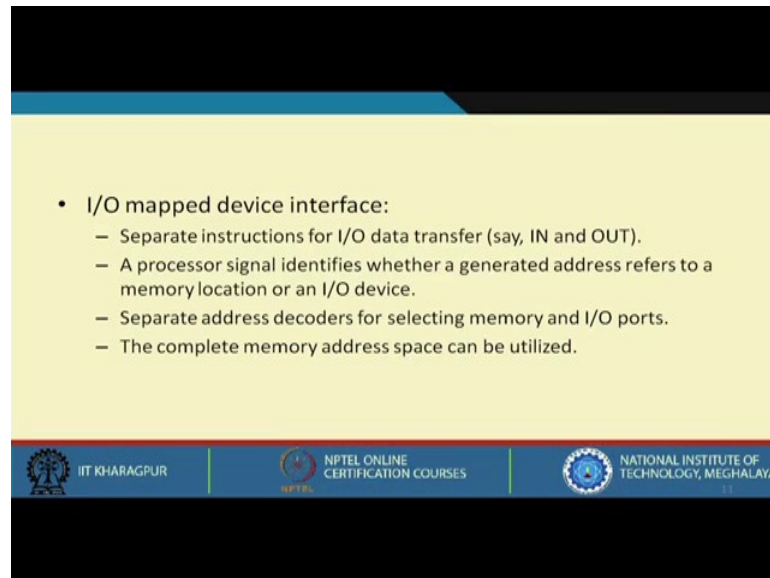
I am using the next two bits of the address, let us say A13 and A12, and this one I am using to enable this decoder. Decoder also has an enable input, this will enable the decoder. I am having 4 ports for I/O, these are the ports through which I can connect I O devices, and this decoder will be selecting one of these ports.

You see here when a particular address combination is given, then this will be selected. suppose this is your 0, 1, 2 and 3. So, when the address combination is 3, means A15 A14 should be 1 1, then 3 will be selected let us call the 4 ports as A B C D. If the next 2 bits A13 A12 are 0 0, this means port A is selected; if the first 4 bits are 1 1 0 1 then B is selected, if it is 1 1 1 0 then C is selected, if it is 1 1 1 1 then D is selected.

You see here I am using the same decoding logic which is selecting both memory and the I/O ports, just by looking at the address I can tell whether it is memory or I/O. So, if I see that my address starts with 1 1, then I can immediately say that this is an I/O address. This decoder will take care of this.

This is the example of a memory mapped I/O interfacing, where the I/O devices are considered as an extension of the memory system, and the same instructions that we used to access memory, we can use to access the I/O devices also. Just we need to remember what is the address of my input devices, what is the address of my output devices, I will have to load and store from those addresses.

(Refer Slide Time: 24:16)



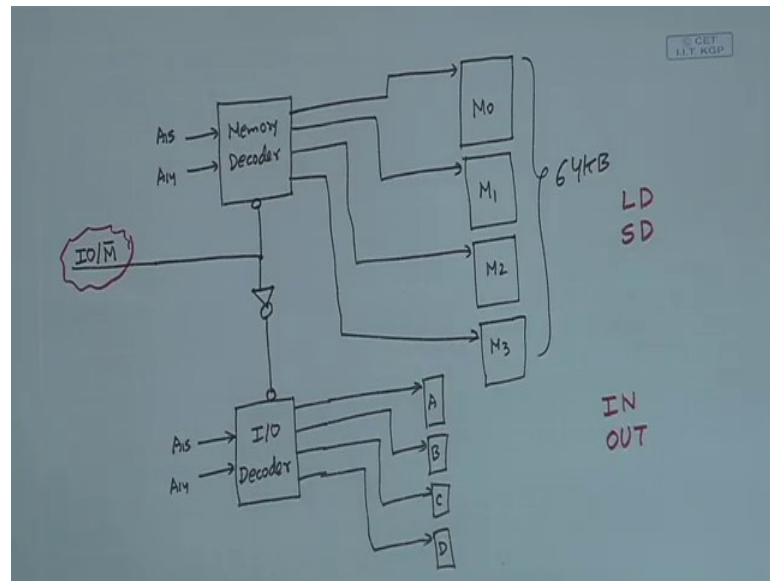
The slide features a yellow background with a blue header bar at the top. The main content is a bulleted list. At the bottom, there is a footer bar with three logos: IIT Kharagpur, NPTEL Online Certification Courses, and National Institute of Technology, Meghalaya.

- I/O mapped device interface:
 - Separate instructions for I/O data transfer (say, IN and OUT).
 - A processor signal identifies whether a generated address refers to a memory location or an I/O device.
 - Separate address decoders for selecting memory and I/O ports.
 - The complete memory address space can be utilized.

Now, let us look at the I/O mapped device interface now. It is a little different from memory mapped. Here we are saying that there are separate instructions for I/O data transfer, let us say IN and OUT. So, IN instruction will be reading some data from an input port, and OUT instruction will be writing some data into an output port.

And there will be also a processor signal which will clearly identify to the circuitry outside whether the address which is cpu is generating, is a memory address or an I/O address. Here we need to use separate address decoders for memory and I/O. The one advantage is that we can utilize the complete memory address space.

(Refer Slide Time: 25:26)



Let us quickly work out that same example we took earlier. Here we consider that I have 4 memory modules, and here we do not need to compromise; we can connect the whole memory; so 16 kilobytes. So, total I have 64 kilobytes, and I have a memory decoder where I am decoding the 2 highest bits, and this will be selecting one of the memory modules.

I assume that the cpu is generating another signal, let us say IO/M' , this signal means if the signal is 0 then it is memory address, if it is 1 it is I/O address. If it is 0 I will be enabling the memory decoder.

Now, let us see what will happen for I/O. For I/O in the same way we will be having a I/O decoder, this will be enabled when IO/M' is 1. This same signal I connect a NOT gate and connect it to the enable of this. So, here again I can connect A15 and A14 and the 4 outputs that I get, I can select one of the 4 I/O ports A B C and D.

You see here just by looking at the address you cannot say which is memory which is I/O, because same address can refer to a memory or an I O device also, but the difference is the way you are accessing memory and I/O. When you are accessing memory, let us say you are using load and store instructions, but whenever you are using I/O, you will be using IN and OUT instructions. What these instructions will do? These instructions will activate IO/M' in a proper way; if it is a load or store instruction then it will be 0, if it is an IN or OUT instruction it will be 1.

So, the appropriate decoder will be selected and the corresponding I/O port will get activated. This explains how the I/O mapped device interfacing works. In our next lecture we shall be looking at what are the different ways in which you can actually transfer data from the I/O devices to the processor, and vice versa. There are several techniques which depends on the characteristics of the I/O devices and also the environment what exactly you want to do.

We shall be looking into those methods in some detail. With this we come to the end of this lecture number 44.

Thank you.