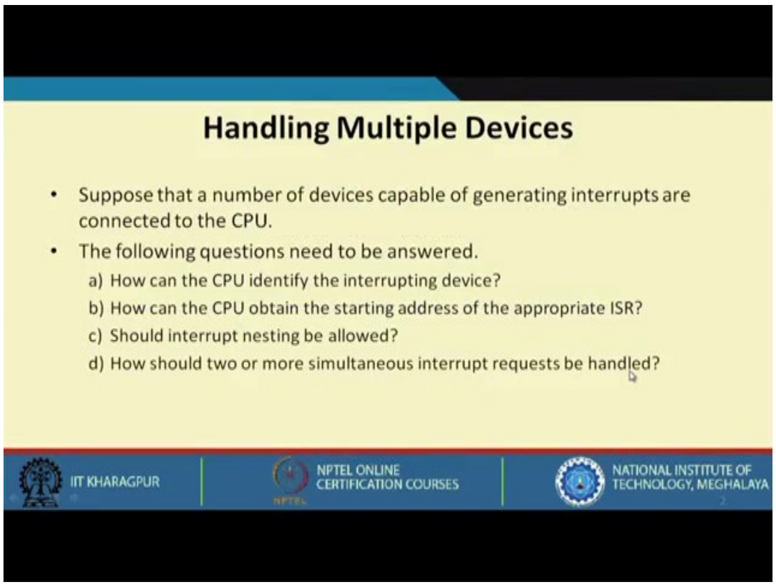
Computer Architecture and Organization

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

Lecture - 47
Interrupt Handling (Part II)

We continue with our discussion on Interrupt Handling. In this lecture, we shall first be looking at some issues that occur whenever multiple devices are allowed to send interrupt signals to the processor. So, this is the Part II of Interrupt Handling.

(Refer Slide Time: 00:40)



Handling Multiple Devices

- Suppose that a number of devices capable of generating interrupts are connected to the CPU.
- The following questions need to be answered.
  a) How can the CPU identify the interrupting device?
  b) How can the CPU obtain the starting address of the appropriate ISR?
  c) Should interrupt nesting be allowed?
  d) How should two or more simultaneous interrupt requests be handled?

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

We start with this problem of handling multiple devices. We have already talked about this earlier in our last lecture, but there are something else we shall be discussing here.

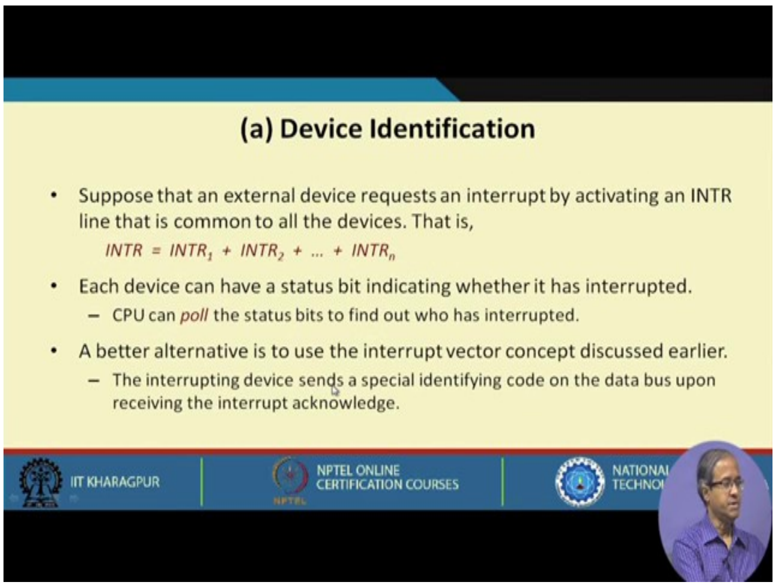
Here we assume because there are multiple devices a number of devices can potentially generate interrupts to the CPU. Now, there are four questions we need to answer. We need to have solutions to all four of these. Some of these we have already answered earlier. First is how can the CPU identifies the interrupting device? Well, earlier we suggested one method that is using the interrupt vector concept. The interrupting device itself can send an interrupt vector through which the CPU can identify who has interrupted.

This is the way in which the device itself is identifying that well I have interrupted. So, CPU can know that.

The second important question is how can the CPU obtain the starting address of the ISR? Once the CPU has identified the device, this is easy because the address of the ISR can be stored in a table. After knowing which device has interrupted, the CPU can consult the table, get the address and can jump to that appropriate address.

The third question is should we allow interrupt nesting to happen because we saw earlier that interrupt nesting can create some problems, or we should be disabling the interrupts while interrupts have been processed? And lastly, for simultaneous interrupt requests, how we should handle them?

(Refer Slide Time: 02:49)

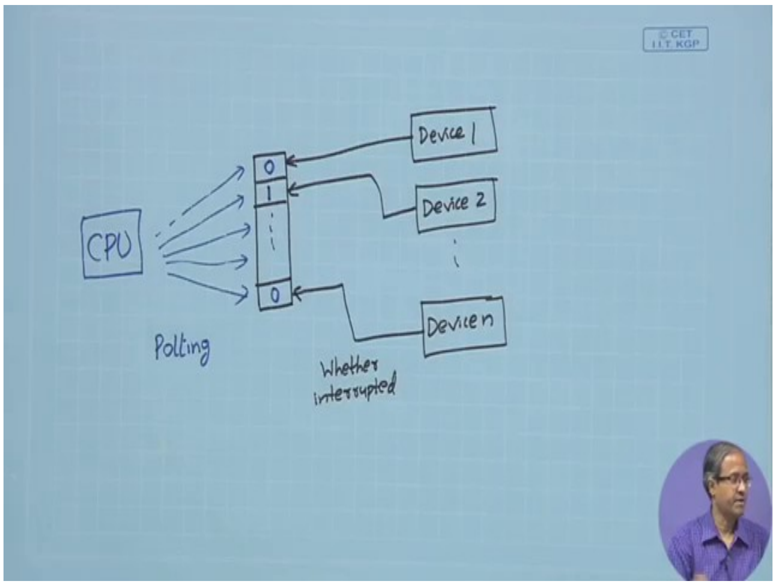


(a) Device Identification

- Suppose that an external device requests an interrupt by activating an INTR line that is common to all the devices. That is,
  $INTR = INTR_1 + INTR_2 + \dots + INTR_n$
- Each device can have a status bit indicating whether it has interrupted.
  - CPU can *poll* the status bits to find out who has interrupted.
- A better alternative is to use the interrupt vector concept discussed earlier.
  - The interrupting device sends a special identifying code on the data bus upon receiving the interrupt acknowledge.

Let us look at these one by one. First device identification. This has already been talked about earlier. The first thing is that there are several external devices which may be having different interrupt signals INTR1, INTR2 up to INTRn. So, if anyone of them generates an interrupt, you can make a logical connection and you can generate the consolidated interrupt request which will be sent to the CPU. So, CPU will get interrupted.

(Refer Slide Time: 03:36)

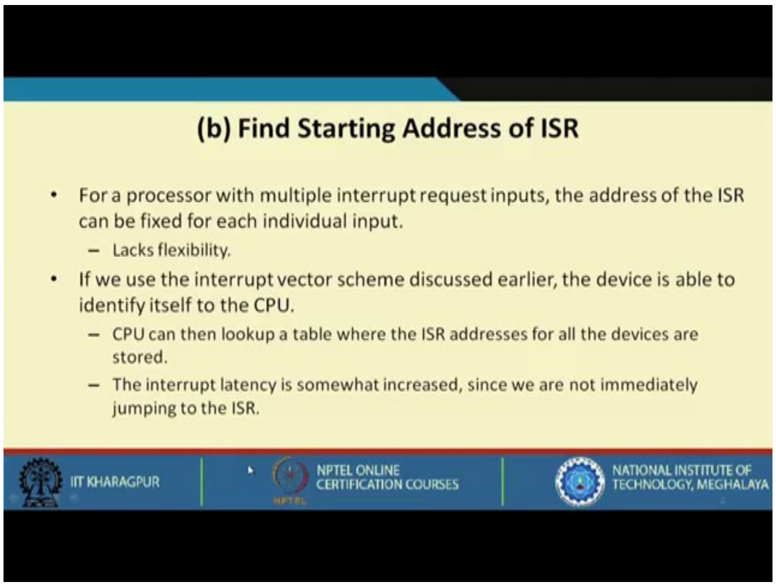


Now, in addition each device can have a status bit indicating whether it has interrupted. What we are saying is something like this. Let us say we have the different devices device1, device2 and devicen. Let us say there is a port, an input port, where each of the device can set one of the bits.

Now, what these bits will indicate? These will indicate whether interrupted or not. Let us say device 2 has interrupted. So, device 2 will be setting this particular bit to 1, and all other bits will remain 0. So, what the CPU will do? CPU can read this word and can check bits one by one which of them is 1. It can identify which of the bits is 1. This process is sometimes called polling. Polling means scanning one by one to find out which device has sent the interrupt.

This is exactly what is mentioned here in the slide. Each device can have a status bit indicating whether it has interrupted, and CPU can go on polling those bits one by one to find out that which device had interrupted. A better alternative will be to use interrupt vector concept that we talked about earlier, because when the CPU is doing polling, some additional time will be required because there will be program which will be running which will be shifting the bits one by one and will be checking which bit is 1, which bit is 0. That will take some time, but if the device itself sends an identifying code, the interrupt vector on the data bus, CPU can immediately read that and directly find out which device has interrupted. There is no need of any polling. So, this is a better alternative.

(Refer Slide Time: 06:18)



(b) Find Starting Address of ISR

- For a processor with multiple interrupt request inputs, the address of the ISR can be fixed for each individual input.
  - Lacks flexibility.
- If we use the interrupt vector scheme discussed earlier, the device is able to identify itself to the CPU.
  - CPU can then lookup a table where the ISR addresses for all the devices are stored.
  - The interrupt latency is somewhat increased, since we are not immediately jumping to the ISR.

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

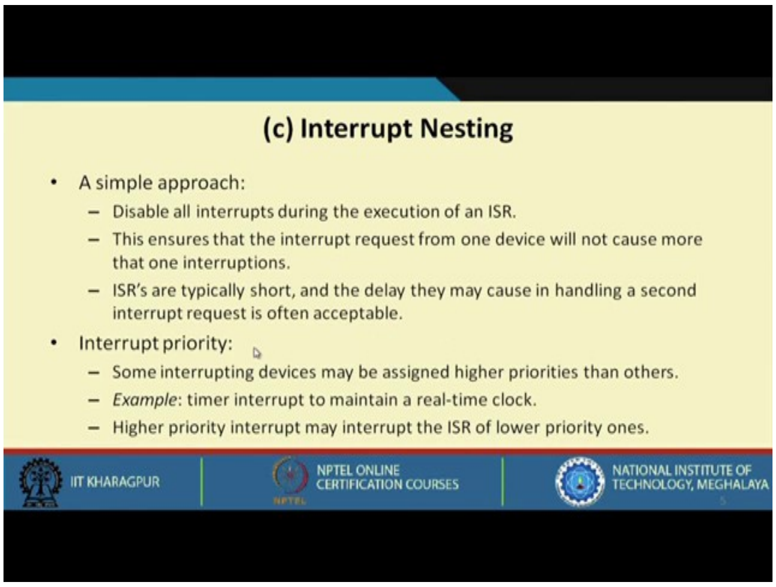
The second question is to find starting address of ISR. Here there can be multiple ways. There can be processors where multiple interrupt requests pins are there. This you would see mostly in some older processors, but in most of the modern processors, you do not see many. You will see that there was multiple interrupt input lines and each of these input interrupt request lines was associated with a particular address of ISR, which means that if an interrupt comes over this line 1, it will jump to a particular address. If an interrupt comes on line 2, it will jump to another particular address.

So, when you write your ISR, you will have to write those ISRs in those particular addresses. This was one of the concepts which was used in some of the processors. For multiple interrupt request lines the address of the ISR can be fixed for each individual input by hardware. If an interrupt comes on a particular line, the control will jump to a particular address. So, ISR will have to be loaded there, but this method lacks flexibility because if there are four interrupt inputs, you can have only four ISRs, but in general the number of devices you are interfacing can be more than four also. So, flexibility is less in this approach.

Again if you use the interrupt vectors scheme, this is much more flexible. There can be many devices which are connected to the CPU. Whoever has interrupted will send the corresponding interrupt vector on the data bus. So, there is no restriction as to how many devices can be connected. It can be 2, 5, 10, or 100. The interrupting device will have the responsibility of sending the interrupt vector on the data bus, and CPU can read it and uniquely identify the device who has sent the interrupt.

Once the CPU has identified the device, there can be a look up table where all the ISR addresses will be stored. The CPU can do a look up in that table and find out the corresponding ISR address, and then it will be jumping to the corresponding ISR. The only difficulty here is that you need to do some processing, searching a table and so on. So, a little more time is required for interrupt handling. This we are calling as the interrupt latency; the delay between an interrupt request signal is arriving and when the interrupt acknowledge is finally going out. This delay will be increased a little bit because here you need to do a little more processing to find out the address of the ISR.

(Refer Slide Time: 09:44)



(c) Interrupt Nesting

- A simple approach:
  - Disable all interrupts during the execution of an ISR.
  - This ensures that the interrupt request from one device will not cause more that one interruptions.
  - ISR's are typically short, and the delay they may cause in handling a second interrupt request is often acceptable.
- Interrupt priority:
  - Some interrupting devices may be assigned higher priorities than others.
  - *Example*: timer interrupt to maintain a real-time clock.
  - Higher priority interrupt may interrupt the ISR of lower priority ones.

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

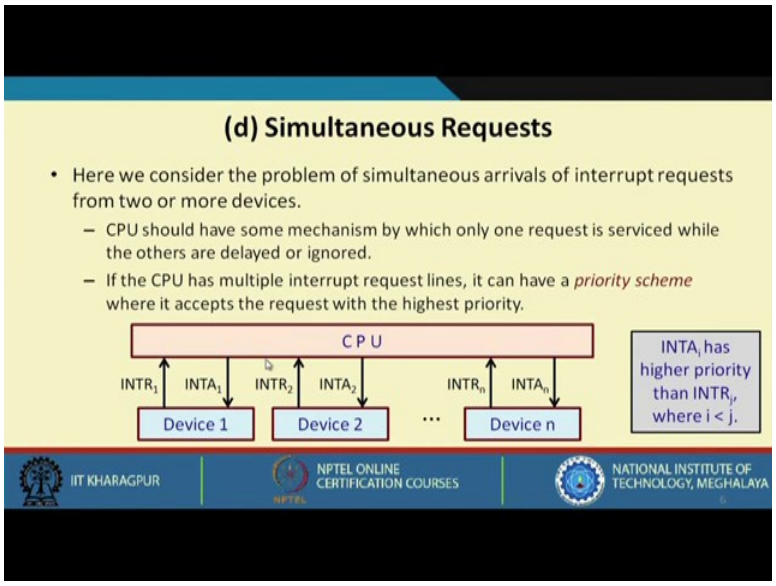
Here we are not immediately jumping to the ISR. We are searching for the address first and then jump. Now, regarding interrupt nesting, we told earlier that we can have a simple approach of disabling all interrupts while the execution of an ISR is going on. This is of course simple. This will ensure that whenever an interrupt comes, there will not be any more interruptions. The ISR will be executing to completion, it will come back, and then if another interrupt comes, there will be another interruption.

There will be at most one interruption per interrupt request, but the point to note is that for most of the devices, the ISRs are typically very short. Just transfer a few bytes or words of data that is the task of the ISR. So, the total time taken for the ISR to finish or execute is typically very small. So, you can afford to use this approach that you can disable the interrupt because you know that ISR will be finished very quickly even if you have disabled. So, after finishing the interrupt can be handled which, has come after that. So, this will not be that much of a problem.

A better method may be to use interrupt priority, with some interrupting devices assigned higher priority than the others. The idea is that here you are not ruling away interrupt nesting all together. Here you are saying that you are allowed to do interrupt nesting, but only higher priority interrupts can interrupt the ISR of a lower priority interrupt. Suppose there is a timer which is generating an interrupt every 1 millisecond, which you are using to maintain a real time clock which is very accurate.

You cannot afford to delay this timer interrupt because this real time clock is a very important thing. Many subsystems of your computer may be using this real time clock for various purposes. When this real time clock interrupt comes, if the CPU sees some other ISR is currently being executed, it will interrupt that because this real time clock interrupt is a very high priority interrupt. A high priority interrupt can interrupt the ISR of lower priority devices or interrupts. This is the concept.

(Refer Slide Time: 12:54)



(d) Simultaneous Requests

- Here we consider the problem of simultaneous arrivals of interrupt requests from two or more devices.
  - CPU should have some mechanism by which only one request is serviced while the others are delayed or ignored.
  - If the CPU has multiple interrupt request lines, it can have a *priority scheme* where it accepts the request with the highest priority.

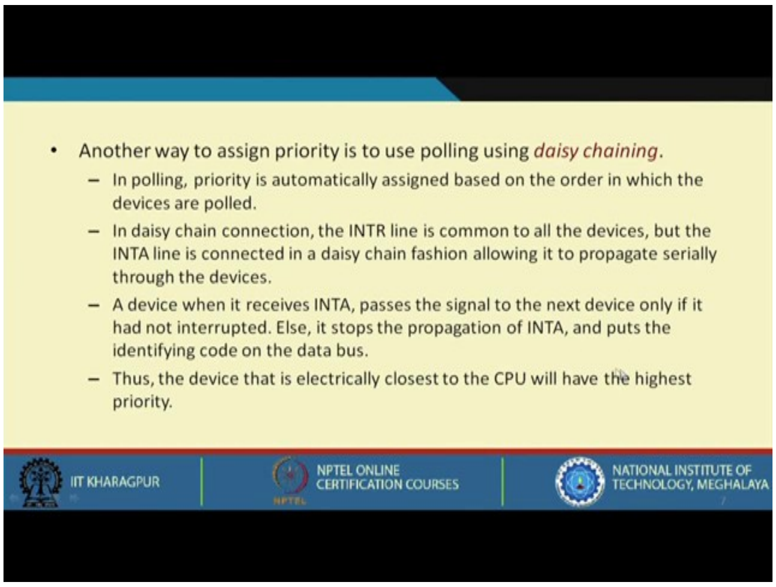
CPU
$INTR_1$
$INTA_1$
$INTR_2$
$INTA_2$
$INTR_n$
$INTA_n$
Device 1
Device 2
…
Device n
$INTA_i$ has higher priority than $INTR_j$, where $i < j$.
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

By using interrupt priority, you can have an arrangement where high priority interrupts can interrupt the ISR of lower priority ones. Lastly comes simultaneous requests. Here we are assuming that simultaneous interrupt requests can arrive from two or more devices. Obviously CPU should have some mechanism to resolve that.

The first approach can be a simple priority scheme which we talked about earlier using a interrupt priority controller. Conceptually there is a CPU, there can be multiple devices; so each of them having a request and an acknowledgement line.

If more than one requests are coming together, the device or the interrupt input that has the higher priority will be acknowledged, while the others will be temporarily ignored. It will be handled later. Here we are assuming, INTRi will be having higher priority than INTRj, if i < j. So, INTR1 will be having higher priority than INTR2, INTR2 will be having higher priority than INTR3 and so on.

(Refer Slide Time: 14:19)



- Another way to assign priority is to use polling using *daisy chaining*.
  - In polling, priority is automatically assigned based on the order in which the devices are polled.
  - In daisy chain connection, the INTR line is common to all the devices, but the INTA line is connected in a daisy chain fashion allowing it to propagate serially through the devices.
  - A device when it receives INTA, passes the signal to the next device only if it had not interrupted. Else, it stops the propagation of INTA, and puts the identifying code on the data bus.
  - Thus, the device that is electrically closest to the CPU will have the highest priority.

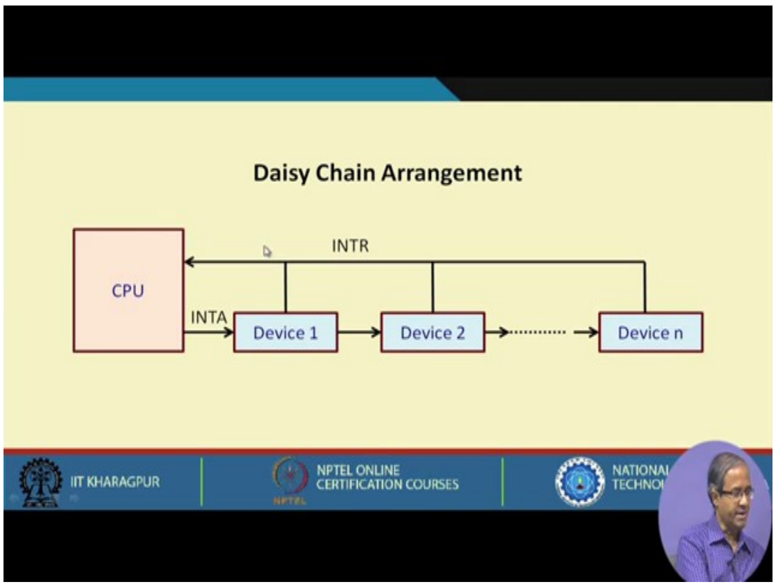
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

There is another way to assign priority. Instead of fixed fixing, there is another way in which this priority is implemented in practice. This is a method called daisy chaining. The idea of this daisy chaining, we will be explaining with an example. The concept is fairly simple. In the method of polling you are checking the devices one by one. Now, there you can regard that priority of the devices are automatically assigned based on the order in which you are checking or you are polling.

Suppose there are 8 devices and you are checking the status of the devices one by one. Now, the order in which you are checking the devices that will determine the priority; the first device that you are checking well, of course we checked first. If it has interrupted, it will be handled first. So, it is having the higher priority. So, if the first device has not interrupted, then only you are going to the next that is having a lower priority. So, the order in which you are checking that implicitly defines the priority.

For daisy chain connection, the interrupt request line is common to all the devices, but the interrupt acknowledge line is connected in a particular fashion. We will be showing this. This is called a daisy chain which propagates serially through the device.

(Refer Slide Time: 16:08)



Daisy Chain Arrangement

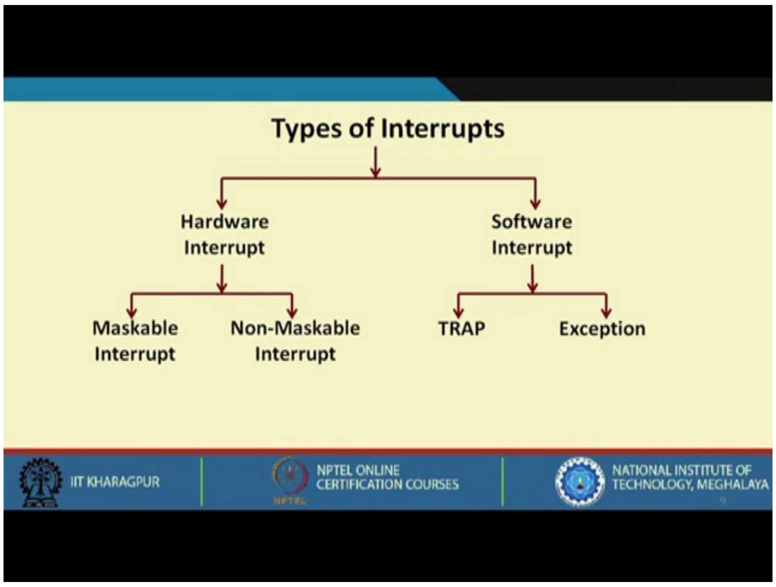
All the devices having the interrupt request are all connected together. Any device interrupting will be generating an INTR, but INTA is not connected directly to all the devices. INTA is connecting to device 1, device 1 is generating an INTA signal to device 2, device 2 is generating an INTA signal to device 3, and so on.

Now, the way it happens is like this. After an interrupt is generated, this interrupt could have come from any of the devices. The CPU generates an acknowledgement and it first comes to device 1. If device 1 had interrupted, then it will send the interrupt vector on the data bus and it will stop this INTA from propagating any further, but if device 1 had not interrupted, and then it will allow this INTA to go forward to device 2. Device 2 will do the same thing. It will check whether it has interrupted or not. If yes, then it will put the interrupt vector on the data bus and if not, it will allow INTA to propagate to the next device and so on.

The order in which the device is connected will define the priority. The device which is nearest to the CPU will be having the highest priority, and the device farthest will be having the lowest priority. So, a device when it receives INTA, passes the signal to the next device only if it had not interrupted. Else if it had interrupted, it will stop the propagation of INTA and will put the interrupt vector or the identifying code on the data bus.

So, as I had said the device that is electrically closest to the CPU will have the highest priority. This is how daisy chain works.

(Refer Slide Time: 18:29)



Types of Interrupts

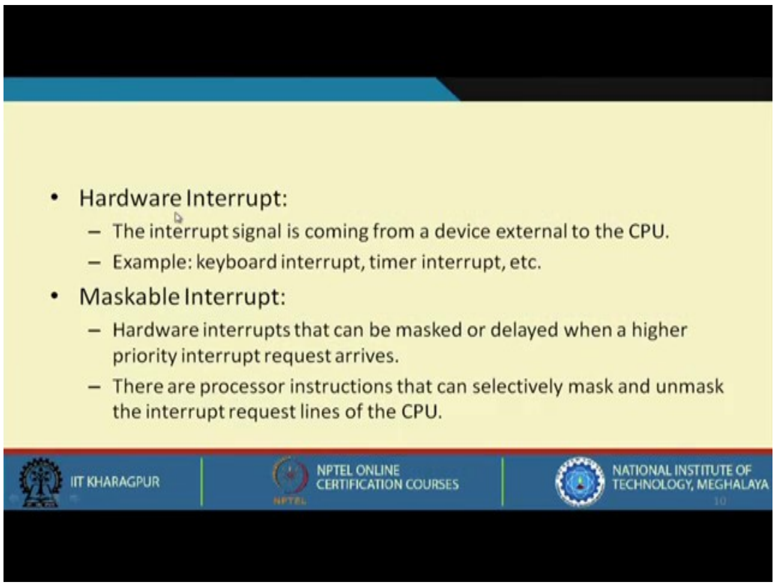
$$\begin{tikzcd}[column sep=small] & & \text{Types of Interrupts} \arrow[dl] \arrow[dr] & & \\ & \text{Hardware Interrupt} \arrow[dl] \arrow[d] & & \text{Software Interrupt} \arrow[d] \arrow[dr] & \\ \text{Maskable Interrupt} & \text{Non-Maskable Interrupt} & & \text{TRAP} & \text{Exception} \end{tikzcd}$$

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

Now, let us look at the different types of interrupt. This diagram shows you the different types. Broadly speaking you can classify interrupts as hardware interrupt and software interrupt. Well, the name comes from the source. Hardware interrupt is something which is generated by the hardware, and software interrupt is something which is generated because of the execution of an instruction.

That is why it is called a software interrupt and hardware interrupt again in turn can be either maskable or non-maskable, and under software interrupt, you can make a categorization, you can have something called trap or you can have something called exception. Let us look at these nomenclatures what these are actually.

(Refer Slide Time: 19:27)



- Hardware Interrupt:
  - The interrupt signal is coming from a device external to the CPU.
  - Example: keyboard interrupt, timer interrupt, etc.
- Maskable Interrupt:
  - Hardware interrupts that can be masked or delayed when a higher priority interrupt request arrives.
  - There are processor instructions that can selectively mask and unmask the interrupt request lines of the CPU.

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

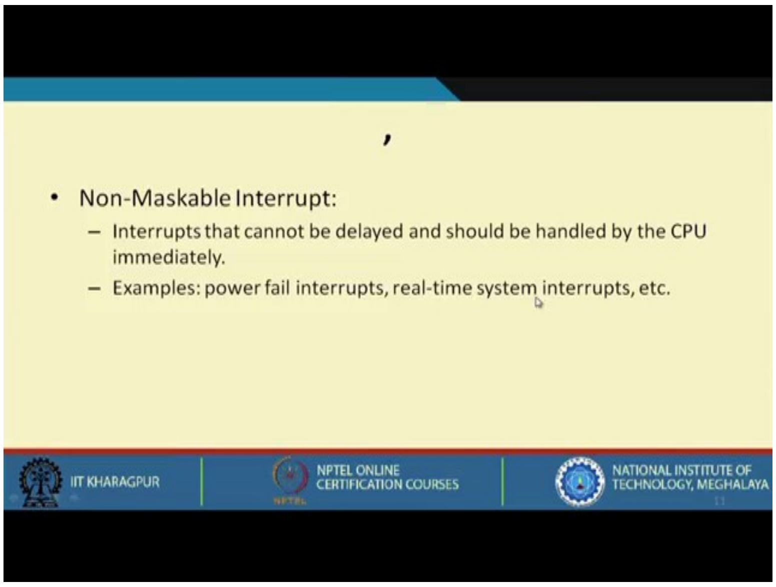
Coming to hardware interrupt the interrupt signal is generated by the hardware. It is coming from a device which is external to the CPU from some hardware unit. It is coming from memory, coming from IO device; some examples are keyboard interrupt, timer interrupt, etc. Suppose I am maintaining the time of the day, the tick is coming every millisecond. So, whenever a tick comes, you increment the value of the clock. These are examples of hardware interrupt.

Now, maskable interrupt means well you are allowed to mask some of the interrupts. Mask means temporarily stop the handling of those interrupts. This means masking. That means, you are allowed to delay the handling or the processing of the interrupts. These kind of interrupts are called maskable interrupts.

The hardware interrupts that can be masked or delayed when a higher priority interrupt request arrives are called maskable interrupts. Suppose there are multiple interrupts. You are handling one of them. If a higher priority interrupt comes and you can afford to delay the current interrupt which has been handled, then we say that it is a maskable interrupt. You are able to mask or pause the handling of that interrupt temporarily.

In systems where this kind of a maskable interrupts are allowed, there are instructions using which you can mask or unmask the various interrupts.

(Refer Slide Time: 21:30)



- Non-Maskable Interrupt:
  - Interrupts that cannot be delayed and should be handled by the CPU immediately.
  - Examples: power fail interrupts, real-time system interrupts, etc.

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

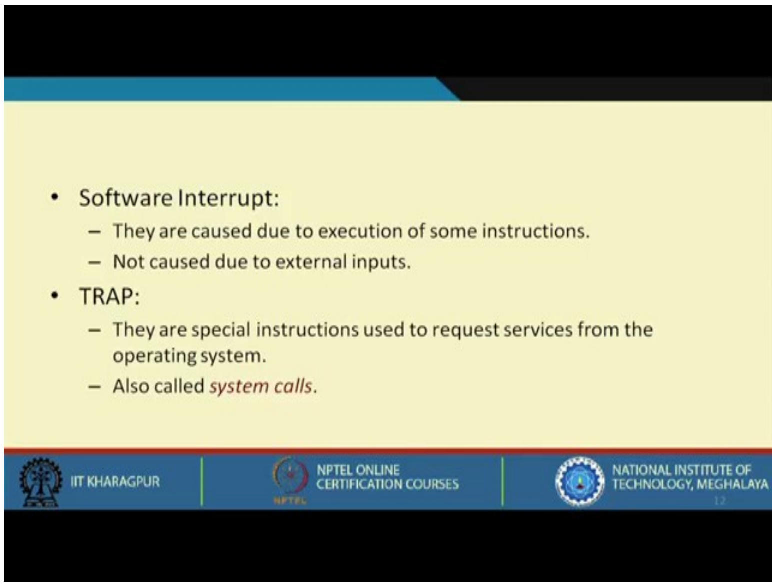
Non-mask able interrupt is one where the interrupts cannot be delayed. They should be handled by the CPU immediately. Some examples are some kind of power fail interrupts. You detect that the power supply of your computer is failing; the voltage level is going down. May be there is a power failure. So, if there is an interrupt which comes, now I may save some of my important data to a non-volatile part of the memory. This is a very important or high priority interrupt.

Similarly, there can be some real time system interrupts. Real time system interrupt means some interrupt request is coming and the interrupt request has to be serviced within well-defined time limit. So, if you are not able to do that, may be your processing will be wrong or the objective for each of design the system will be lost.

Well, some examples may be medical information system where you are monitoring a patient in real time depending on some health status. You are taking some corrective actions. So, if there is something anomalous, you must take some action immediately. You think of an industrial control plant. You are sensing temperature, pressure, humidity, so many things. So, if you see that some parameter values has gone beyond the desired range, you may have to immediately take some corrective action.

Well, for a defense system you see that an enemy aircraft or a missile is heading towards you, you should immediately start the intercepting mechanism. So, these are some examples where interrupts have to be handled immediately and we have to use non-maskable interrupt for such cases. These interrupts cannot be masked or delayed. These are extremely important.

(Refer Slide Time: 23:45)



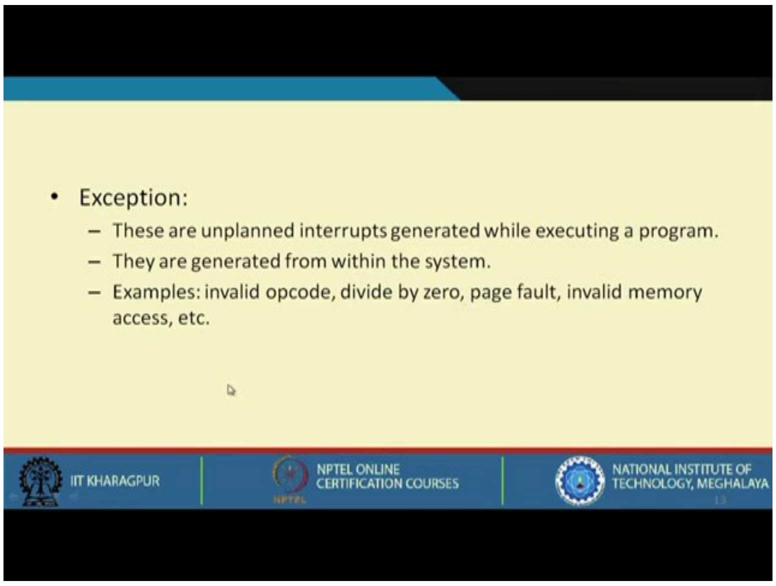
- Software Interrupt:
  - They are caused due to execution of some instructions.
  - Not caused due to external inputs.
- TRAP:
  - They are special instructions used to request services from the operating system.
  - Also called *system calls*.

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

Coming to software interrupts, they are caused due to execution of some instructions. They are not caused by some external hardware. One type is called Trap. They are also sometimes called system calls. These are nothing but special instructions which are used to request services from the operating system.

These we studying in more detail in the operating system course, but for the time being let us assume that a trap or a system call is a kind of an instruction which behaves similar to an interrupt; so when it is being executed, some instructions are saved and then, you jump to a routine. That routine gets executed and that may be part of the operating system. Then, you come back. That is what a trap is.

(Refer Slide Time: 24:45)



- Exception:
  - These are unplanned interrupts generated while executing a program.
  - They are generated from within the system.
  - Examples: invalid opcode, divide by zero, page fault, invalid memory access, etc.

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

The second kind of software interrupt is called exception. Exception is unplanned while trap is planned. That means you have an instruction and you are executing it. This is the planned call, but an exception is unplanned. They are generated within the system, but you do not know when they will come. They might be generated because of something which is beyond your control or due to some error this has happened. Like you may encounter due to some error, let say some memory error, some opcode has become invalid ,and you are trying to decode an instruction and find out that the opcode is not a valid opcode. So, you may have to interrupt immediately and tell that well there is something wrong and I have to stop my program.

Again, divide by zero, page fault, invalid memory access. You are trying to access a memory which you are not supposed to access. It is part of some other program or process. There is memory protection hardware which will prevent you from accessing that. If you try to access a memory location which is beyond your allowable limit, then a interrupt will be automatically generated.

Whenever such a thing happens, you generate something called an exception. These are also software interrupts. Later on we shall see when we look at the pipeline implementation of MIPS32 processor ISA, interrupt handling and exception handling is a real problem and we shall of course also suggest some of the approaches which we can use to handle interrupts and exceptions in MIPS32 pipeline implementation.

We have come to the end of this lecture, and in the next lecture we shall be continuing with some other IO transfer techniques, namely direct memory access which we have not yet discussed following which we shall be discussing some of the commonly used standards for data transfer, some bus standards and so on.

Thank you.