

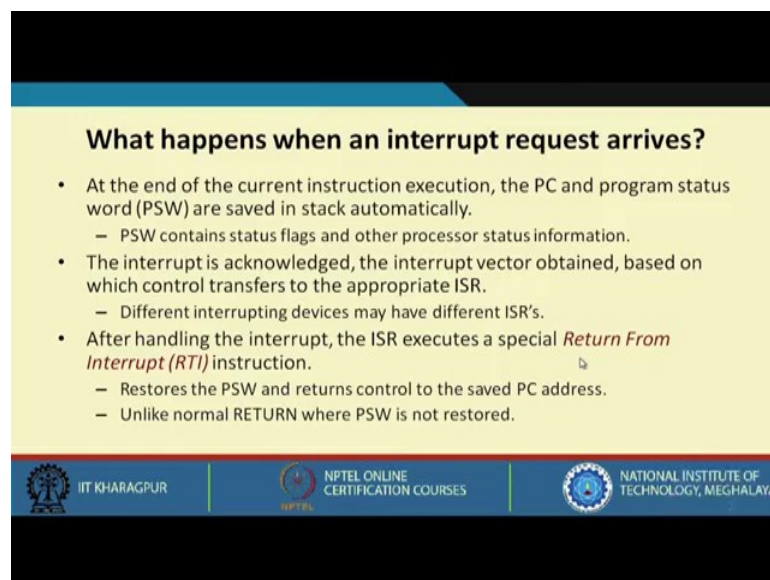
Computer Architecture and Organization
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 46
Interrupt Handling (Part I)

In our last lecture we were talking about the various IO transfer techniques specifically under the programmed IO category. If you recall we looked at synchronous method of data transmission, then you looked at asynchronous or handshaking, and finally we were discussing the concept of interrupt driven data transfer.




In case of interrupt driven data transfer what you saw was the processor is doing something, whenever the IO device is ready to transfer some data it will be interrupting the CPU by sending a interrupt request signal. The CPU when it receives the interrupt request signal will know that some IO device is now ready to transfer data, so it will be jumping to some ISR where the data transfer will take place, and after that it will again return back to the program which it was executing. So, here while the device is getting ready for transfer, the CPU is able to do something else; CPU is not getting tied up.

(Refer Slide Time: 01:48)



What happens when an interrupt request arrives?

- At the end of the current instruction execution, the PC and program status word (PSW) are saved in stack automatically.
 - PSW contains status flags and other processor status information.
- The interrupt is acknowledged, the interrupt vector obtained, based on which control transfers to the appropriate ISR.
 - Different interrupting devices may have different ISR's.
- After handling the interrupt, the ISR executes a special *Return From Interrupt (RTI)* instruction.
 - Restores the PSW and returns control to the saved PC address.
 - Unlike normal RETURN where PSW is not restored.

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

We shall be continuing our discussion on interrupt driven data transfer, in particular the topic of interrupt handling.

Let us see what exactly happens when an interrupt request reaches the CPU. The first thing that happens is that the CPU was executing some instruction when the interrupt request came. We shall be seeing that some exceptions can be there, but typically the instruction which was being executed will first complete and only after that the interrupt request will be acknowledged.

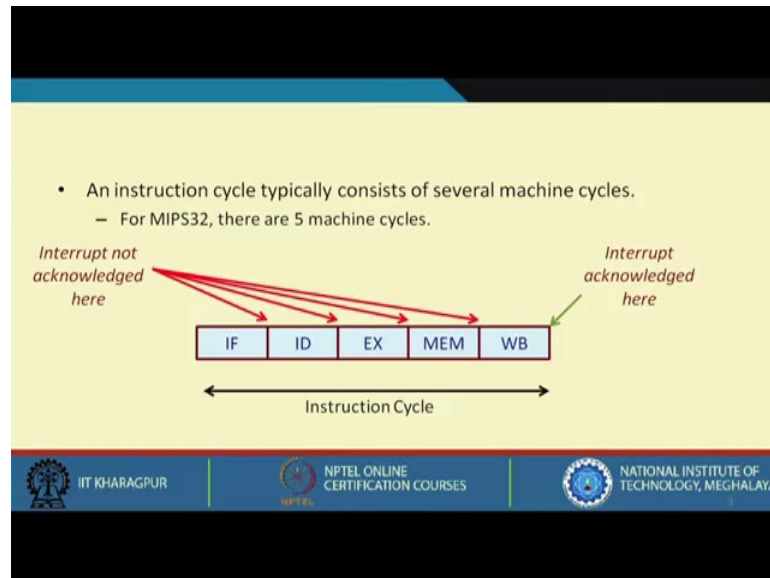
So, what we are saying is that at the end of the current instruction execution, during the interrupt acknowledge cycle the program counter and the program status word or PSW are saved in a stack. You may recall that for normal subroutine or function call and return only the return address; that means, the value of the PC is saved in stack, but here we are saving not only the PC, but also some status information, which depends on the processor, the ISA.

Typically in computer systems there are status flags. PSW will contain status flags and some other processor status information regarding which level of privilege it is working on, and so on and so forth. Now you may recall in MIPS32 processor we do not have any such status flags. So, saving of the PSW becomes much simpler there.

After this is done the interrupt is acknowledged, which means the interrupt acknowledge signal is activated and the external interrupting device or the IO module can supply the interrupt vector. The interrupt vector will identify the device that has interrupted; using that information you can compute the address of the interrupt service routine and transfer control to that.

Different devices may be having different service routines. You need to get the address of the correct service routine and then jump to that. Now after the interrupt handling is completed the interrupt service routine executes a return, but this is a special kind of a return instruction not a normal return instruction that you use to return from a subroutine or a function where only typically the PC is popped from the stack and is loaded into PC. But here you have to restore not only the value of the PC, but also the program status word which was also saved. So, this is a special kind of instruction, let us say RTI. This will restore the PSW and return control to the saved PC address. As I had said this is slightly different from the normal return instruction where PSW is not saved and returned, only the PC is restored.

(Refer Slide Time: 05:23)



Now, let us look at an instruction cycle. In MIPS32 you recall the instruction cycle consist of five steps, IF ID EX MEM and WB. These are typically called machine cycles. An instruction cycle comprises of typically more than one machine cycles. These machine cycles are executed in order to complete the execution of an instruction.

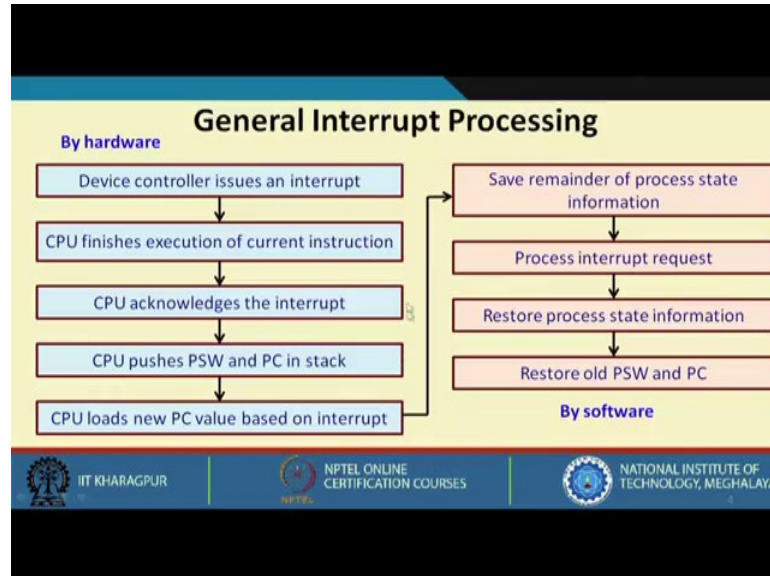
When the instruction execution is complete only then interrupts are acknowledged, but in between the machine cycles interrupts are not acknowledged even if the interrupt may appear earlier. But the processor will wait till the instruction execution is complete, and only then the interrupt system will get acknowledged.

Now, in this context you may recall one thing that we talked about the machine cycles and instruction cycle, and said that interrupts will be acknowledged only at the end of the instruction cycle not in between, but there is another method of IO transfer which we have not discussed yet that is direct memory access.

DMA is one method using which the IO device can directly transfer some data to memory or back without intervention of the CPU. Now here there is slight difference in terms of the handling of the DMA requests. Whenever a device wants to make such DMA transfer, we can stop the processor not only at the end of the instruction cycle, but in between also. At the end of a machine cycle we can stop, at the end of IF we can stop, at the end of ID or EX or MEM we can stop. This is possible because while the transfer

is going on the CPU status is not changing; it is just like a pause. While the transfer is going on CPU will pause, and then it will again continue.

(Refer Slide Time: 08:08)



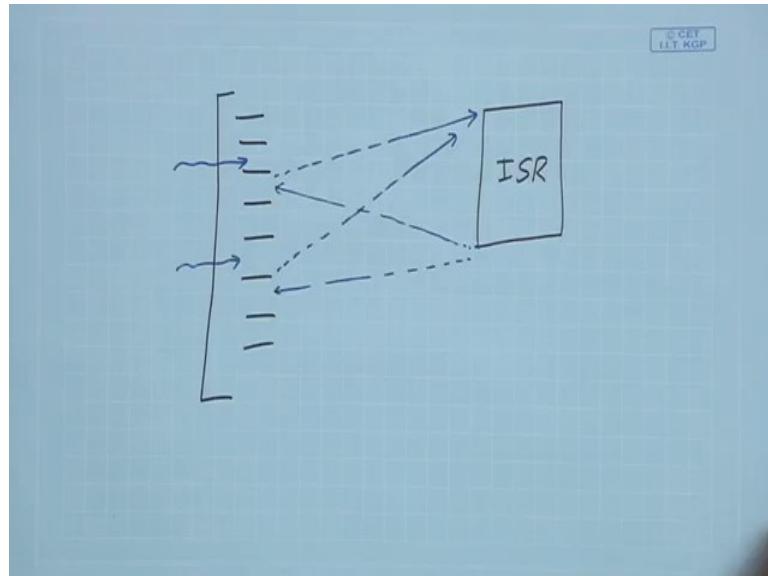
The general interrupt processing can be summarized by this flow diagram. While the portion marked in blue are performed in the hardware unit in the CPU, while the pink part are performed by software as part of the interrupt service routine. Let us see what happens. The device controller to which the IO device is connected issues an interrupt request. CPU receives that, CPU will finish the execution of the current instruction, and only after that the CPU will save the PC and the PSW, it is after that it will acknowledge the interrupt and it will save PSW and PC in the stack.

After acknowledging the interrupt as the CPU will try to know that which device has interrupted. Here the interrupt vector concept comes in, whenever interrupt acknowledge is coming the external device controller will be pushing some kind of a interrupt vector or a device ID on the data bus. CPU can read it and can identify the device.

Now, depending on the device CPU will be jumping to the appropriate interrupt service routine, which means it will loading the PC with the address of the corresponding ISR. After this is done control will jump to the ISR; this part is done by the CPU automatically whenever the interrupt comes, but here when ISR assumes control there is a program code that is written. It will be saving some other processor information, if required may be some registers it will be saving, and it will be processing the request, if

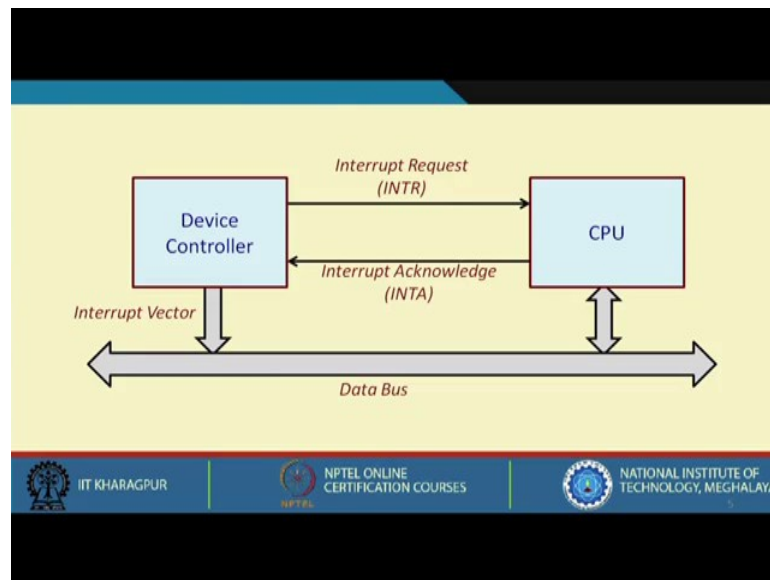
it is data transfer it will be actually carrying out the transfer, and whatever registers were saved will restore them back, and it will return back by restoring the old PSW and PC.

(Refer Slide Time: 10:58)



Now the point to note is that suppose this was a program which was executing. There are many instructions and let us say this is my ISR. Now you cannot predict before exactly where the interrupt will come; may be the interrupt will arrive here, then after the execution of this instruction you will be jumping to the ISR. It will finish and then it will return back. May be the interrupt has come here, then after execution of this control will jump it will get executed and it will be returning here.

(Refer Slide Time: 11:49)



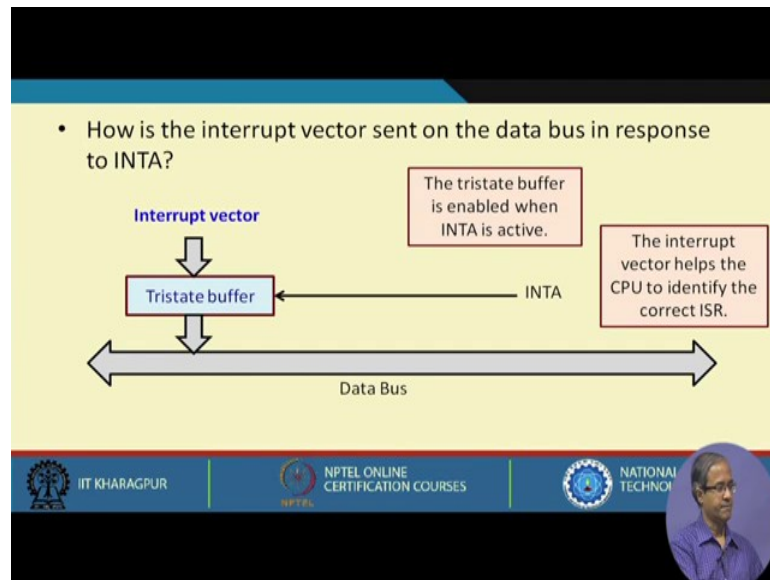
So, it can actually jump from anywhere in the program depending on where the interrupt request has actually come. This is the picture that shows the CPU, the device controller and the bus. The device controller will be sending an interrupt request to the CPU, CPU at the end of the current instruction execution will be sending back an interrupt acknowledge.

(Refer Slide Time: 12:25)

- The steps:
 - a) Device controller sends INTR to the CPU.
 - b) CPU finishes the current instruction and sends back INTA.
 - c) Device controller sends interrupt vector (or number) over data bus.
 - d) CPU reads the interrupt vector, and identifies the device.

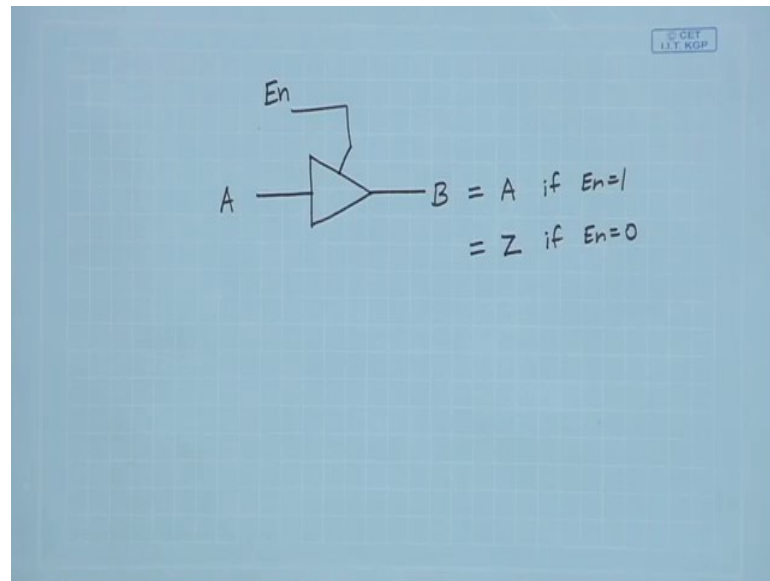
In response to that, device controller will be pushing an interrupt vector on the data bus. CPU will be reading that interrupt vector and will be able to identify which device has interrupted.

(Refer Slide Time: 12:55)



Now, the question is how is the interrupt vector sent to the data bus in response to INTA? We have said that whenever this INTA signal is coming, the device will be putting the interrupt vector on the data bus, but how does it happen automatically? It is very simple. Here actually what we require is the tristate buffer. On the input side of the tristate buffer we have the interrupt vector, the output of which is connected to the data bus and the tristate buffer is enabled by the interrupt acknowledgement signal.

(Refer Slide Time: 13:46)



You recall what is a tristate buffer. A tristate buffer is a circuit which has an enable. Suppose I have an input A and the output B. If this buffer is enabled, then B will be equal to A, but if it is not enabled then the output will be electrically disconnected or in the high impedance state. So, this tristate buffer can either connect or disconnect a signal from a destination point; in this case the destination point is the data bus.

Whenever INTA is active the interrupt vector will appear on the data bus. When it is not active, tristate buffer will be making the output lines in high impedance state. The interrupt vector which is pushed on the data bus can be read by the CPU. This will help the CPU to identify which device had interrupted, and accordingly it can identify the correct interrupt service routine.

(Refer Slide Time: 15:20)

Multiple Devices Interrupting the CPU

- A common solution is to use a priority interrupt controller.
 - The interrupt controller interacts with CPU on one side and multiple devices on the other side.
 - For simultaneous interrupt requests, interrupt priority is defined.
 - The interrupt controller is responsible for sending the interrupt vector to CPU.

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

Now, let us consider the scenario where multiple devices can interrupt the CPU. Let us look at a scenario like this. Suppose there are several devices connected on this side. Each of these devices will be having an interrupt request and interrupt acknowledge; let us say there are 4 such sets.

When there are multiple devices that need to interrupt the CPU, one common solution is to have a device which is called a priority interrupt controller. A priority interrupt controller works exactly like shown in this diagram. On one side it will be having multiple sets of interrupt request and acknowledge lines through which various device controllers can be connected, but on this side of the CPU there is a single interrupt line and a single interrupt acknowledge line. The interrupt controller interacts with CPU on one side and multiple devices on the other side.

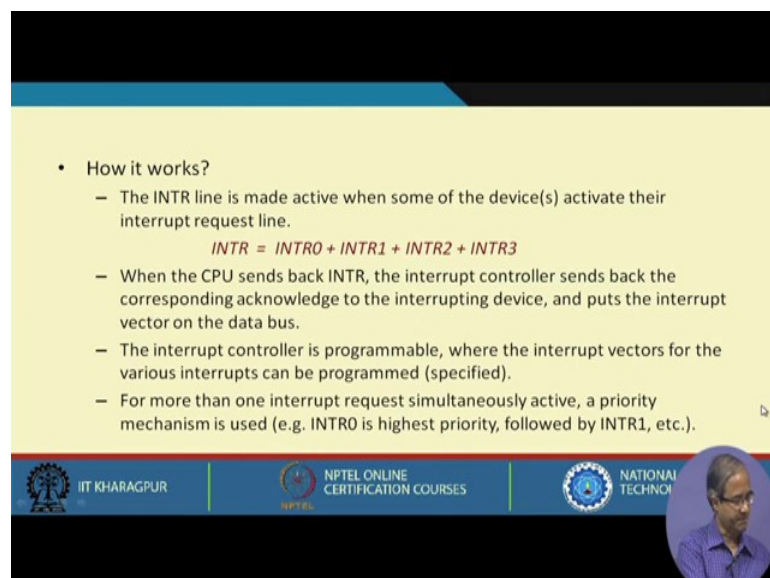
Here the idea is whenever any one of the devices will be sending an interrupt request, the priority interrupt controller will automatically generate an interrupt request to the CPU, and when the CPU sends back the acknowledgement the interrupt controller knows that which device has sent interrupt. Whenever acknowledgment comes, this interrupt controller will be putting the appropriate interrupt vector on the data bus corresponding to that interrupting device.

So, interrupt controller is responsible for sending the correct interrupt vector to the CPU, but just one thing. If suppose two or more interrupt lines are activated simultaneously

then what will happen? Then this priority comes into the picture. Let us say you define a priority where this line 0 has higher priority than 1, 1 has higher priority than 2, and 2 has higher priority than 3. So, if interrupt request 0 and interrupt request 2 are activated simultaneously, then this interrupt request 0 will be processed and 2 will be ignored for the time being.

Later on, after finishing the processing of interrupt request 0, if the interrupt request 2 is still active it will be processed and handled. This is a typical way using which multiple interrupts can be handled.

(Refer Slide Time: 18:34)



• How it works?

- The INTR line is made active when some of the device(s) activate their interrupt request line.

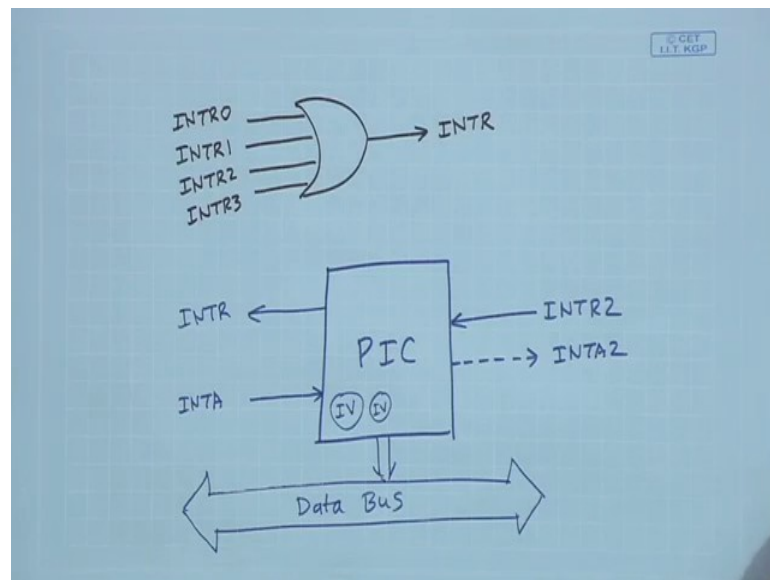
$$INTR = INTR0 + INTR1 + INTR2 + INTR3$$

- When the CPU sends back INTR, the interrupt controller sends back the corresponding acknowledge to the interrupting device, and puts the interrupt vector on the data bus.
- The interrupt controller is programmable, where the interrupt vectors for the various interrupts can be programmed (specified).
- For more than one interrupt request simultaneously active, a priority mechanism is used (e.g. INTR0 is highest priority, followed by INTR1, etc.).

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL TECHNICAL INSTITUTE OF ADVANCED STUDIES

The interrupt line as I had said is activated when some of the devices activate their interrupt request line.

(Refer Slide Time: 18:55)



How it can be implemented? You use a simple 4 input OR gate, where the 4 interrupt request lines are connected to the 4 inputs and the output will be generating the consolidated interrupt request signal to the CPU. Then the CPU sends back interrupt acknowledge INTA. When the CPU sends back INTA, the interrupt controller will send back the corresponding acknowledge to the interrupting device and puts the interrupt vector on the data bus.

Suppose I have the priority interrupt controller here. Let us suppose device number 2 has interrupted. In response the CPU got this INTR. Later on when the CPU generates the acknowledgement, the PIC will be generating the corresponding acknowledgement for this device 2, INTA 2, this will be done automatically. It also has another responsibility; it is also connected to the data bus. Whenever this interrupt acknowledge is sent by the CPU, the corresponding interrupt vector for this particular device will be pushed on the data bus, so that this CPU can read the value and know which of the devices had actually interrupted. This is the second step and the third point to note is that the interrupt controller can be storing the interrupt vectors for the different devices.

The interrupt controller is in some sense programmable, programmable means before you are using it you can specify that these will be the interrupt vectors for the 4 devices. The CPU can initialize some internal registers within that PIC, where the values of the

interrupt vectors can be programmed or specified. If more than one requests are activated simultaneously we can use a priority mechanism.

Let us say by convention interrupt 0 will be having the highest priority followed by interrupt 1 followed by interrupt 2 and then interrupt 3 .

(Refer Slide Time: 22:14)

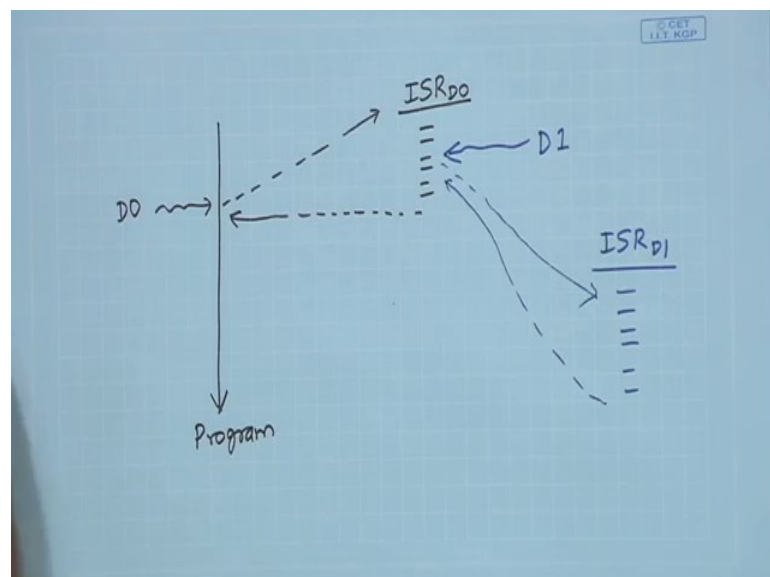
How is interrupt nesting handled?

- Consider the scenario:
 - a) A device D0 has interrupted and the CPU is executing the ISR for D0.
 - b) In the mean time, another device D1 has interrupted.
- Two possible scenarios here:
 - D1 will interrupt the ISR for D0, get processed first, and then the ISR for D0 will be resumed. → *CREATES PROBLEM FOR MULTI NESTING*
 - Disable the interrupt system automatically whenever an interrupt is acknowledged so that handling of nested interrupts is not required.

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

Another issue is nesting of interrupts. Let us consider a scenario where a device D0 had sent an interrupt to the CPU, and the CPU is currently executing the corresponding ISR. While this ISR is being executed, let us assume that another device D1 has interrupted.

(Refer Slide Time: 22:47)



What I am saying is that some program was executing in between there was an interrupt from device D0. Because of this interrupt the control had transferred to the ISR corresponding to device D0. After finishing it is supposed to come back like this and resume execution.

But what we assuming is that while this ISR D0 was executing there is another interrupt which has arrived from D1. So, now you may argue that there is also an ISR for device D1. So, should we stop this go here process this and then come back and resume here. This is called nesting that before something is finished you are going to somewhere else. This is nesting of interrupt requests; before the handling of interrupt request D0 is finished, another interrupt has come. For nesting there can be two possible scenarios.

The first one is what I have just now illustrated. D1 can interrupt the interrupt service routine for D0, get processed first and then the ISR of D0 will be resumed. So, when the D1 interrupt comes, the ISR of D1 will be processed first, then it will come back, and ISR of D0 will be resumed, and then it will be finished. This will involve nesting of interrupts and theoretically means any arbitrary number of such nestings can happen because interrupts may appear one after another. Some important interrupt request that came earlier might get delayed for its processing.

The other thing is that you can disable the interrupt system automatically whenever an interrupt is acknowledged, so that handling of nested interrupt is not required. Because I mean if you disable the interrupt system, then if some interrupt request comes it will not be considered, it will be ignored. This is another solution.

(Refer Slide Time: 25:38)

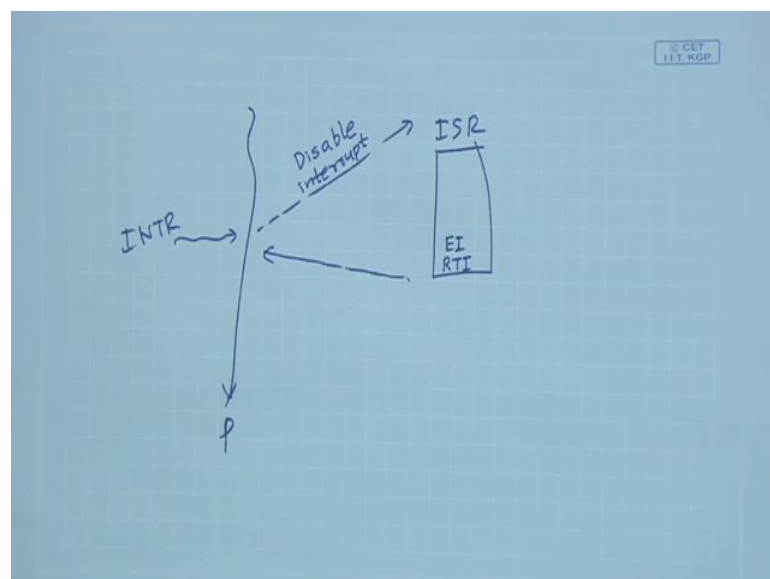
- Typical instruction set architectures have the following instructions:
 - EI : Enable interrupt
 - DI : Disable interrupt
- For the second scenario as discussed, the ISR will give an EI instruction just before RTI.
 - Some ISA combine EI and RTI in a single instruction.
- The DI instruction is sometimes used by the operating system to execute atomic code (e.g. semaphore wait and signal operations).
 - Nobody should interrupt the code while it is being executed.

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

Whenever you are processing an ISR you disable the interrupt system. So, nesting will not happen.

For this typical machine instructions are there for enabling and disabling interrupts, like EI and DI. So, for this second scenario the interrupt service routine will give an EI instruction just before return from interrupt.

(Refer Slide Time: 26:17)



Now, we are having a design where whenever an interrupt comes before jumping to the ISR interrupt is disabled. Like what I am saying is that a program was executing and

interrupt comes. Before your jumping to the ISR, before you are jumping to the ISR the hardware is automatically disabling the interrupt. So, execution of the ISR will not be interrupted. And in the ISR before the RTI instruction that is supposed to be the last instruction, there will be an explicit EI instruction so that before returning back you enabling interrupt again, so that any future interrupt if it comes should be acknowledged.

Some instruction set architecture combine EI and RTI in a single instruction. So, when you do a return, for means return from interrupt, interrupt will be automatically enabled and you do not need a separate EI.

(Refer Slide Time: 28:28)

Cases that make interrupt handling difficult

- For some interrupts, it is not possible to finish the execution of the current instruction.
 - A special RETURN instruction is required that would return and *restart* the interrupted instructions.
- Some examples:
 - a) Page fault interrupt: A memory location is being accessed that is not presently available in main memory.
 - b) Arithmetic exception: Some error has occurred during some arithmetic operation (e.g. division by zero).

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

We have seen different cases of interrupts. There are some cases that can make the interrupt handling more difficult. Like we have assumed so far that whenever an interrupt is coming we have to finish the current instruction, then only we can acknowledge the interrupt. But there are some cases where you cannot finish the current instruction that makes interrupt handling more difficult. For some kinds of interrupt it is not possible to finish the execution of the current instruction.

For such cases you need a special return instruction that would be returning from the ISR, but after returning it will be restarting the same instruction which was interrupted; not that it will be returning to the next instruction it will be restarting the interrupted instruction. An example is page fault; this happens in operating system in the memory management when a memory location is accessed that is not presently loaded in memory.

You will have to load the requested memory location page and come back and again restart the instruction, so that it can access the memory location correctly this time.

And you can also think of arithmetic exceptions like division by 0, square root of a negative number. In such cases you are not able to finish the instruction. Such cases can make interrupt handling more difficult.

With this we come to the end of this lecture. In the next lecture we shall be continuing with our discussion on interrupts, and we shall see some more issues particularly regarding multiple devices sending interrupts and the different types of interrupts.

Thank you.