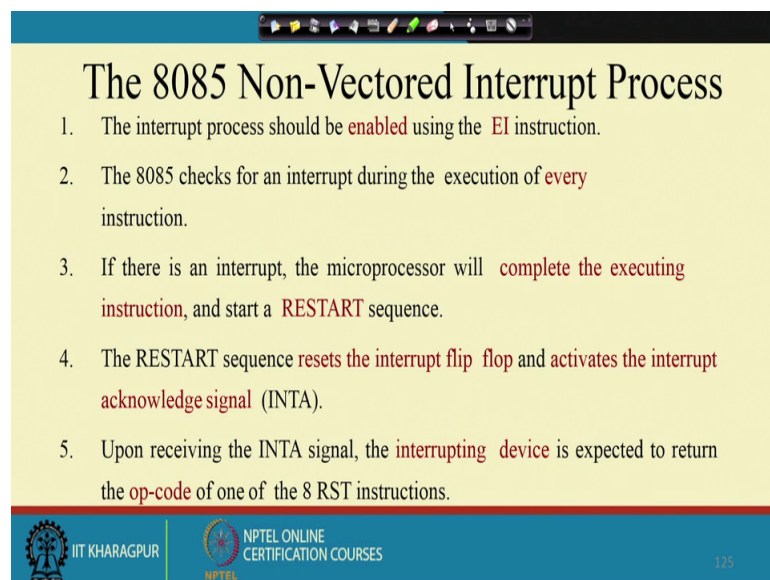


Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 58
8085 Microprocessor (Contd.)



In our last class we started discussing on 8085 non vectored interrupt process. So, as we know that in case of non vectored interrupt, the interrupting device it should provide the ISR addresses (Refer Time: 00:25) address for the interrupt service routine.

(Refer Slide Time: 00:26)



The 8085 Non-Vectored Interrupt Process

1. The interrupt process should be **enabled** using the **EI** instruction.
2. The 8085 checks for an interrupt during the execution of **every** instruction.
3. If there is an interrupt, the microprocessor will **complete the executing instruction**, and start a **RESTART** sequence.
4. The RESTART sequence **resets the interrupt flip flop** and **activates the interrupt acknowledge signal (INTA)**.
5. Upon receiving the INTA signal, the **interrupting device** is expected to return the **op-code** of one of the 8 RST instructions.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES

125

Now, for their pre requisite for this Non-Vectored Interrupt Process to be initiated is that the interrupt process should be enabled using the EI instruction. So, somewhere in the beginning, so when the 8085 processor is reset, this all the interrupts are enabled.

However, due to due to the course of some program execution maybe the interrupts have been disabled. So, through some DI instruction, so in that case, these interrupt will not be sensed. So, it is required that before this non vectored interrupt it can sent given interrupt to the processor, the process should be enabled by means of an EI instruction. 8085 checks for an interrupt during the execution of every instruction.

So, as I said it is the last, but one clock cycle of an instruction at which the interrupt status is checked. So, that way, it is at the end of that instruction, so it will check whether

an interrupt has occurred or not. And if an interrupt has occurred, then instead of executing the next instruction in the sequence the processor goes to the interrupt service routine. If there is an interrupt, the microprocessor will complete executing the current instruction and start a restart sequence. So, restart sequence, it is basically a sequence of operations by which the service routine can be started.

So, first of all, it resets the interrupt flip flop. So, there we will see later that there is an interrupt enable flip flop. So, that flip flop is reset in the beginning of the restart sequence. And also, you remember that there was a pin of 8085 called interrupt acknowledge. So, INTA which is an active low signal and this is activated. So, interrupt acknowledge line is made low. So, from the device an interrupt signal has come and this interrupt acknowledge signal is generated by the processor and then it goes to the device.

So, device will now understand that my interrupt has been acknowledged. So, I have to provide the service routine address. So, upon receiving the interrupt acknowledge signal, the interrupting device is expected to return the op code of one of the 8 RST instructions. So, RST is the restart instruction. So, it takes along with it one value which is a 3 bit value. And so, 3 bit value means it can go from 0 to 7. So, 8 possible RST instructions are there starting with RST 0 going up to RST 7 as you will see subsequently.

(Refer Slide Time: 03:04)

The 8085 Non-Vectored Interrupt Process

- When the microprocessor executes the RST instruction received from the device, it saves the address of the next instruction on the stack and jumps to the appropriate entry in the IVT.
- The IVT entry must redirect the microprocessor to the actual service routine.
- The service routine must include the instruction EI to re-enable the interrupt process.
- At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

The diagram shows a vertical box representing an IVT entry. At the top, it is labeled 'JMP 8070'. An arrow points from this label to a box containing '8070'. Below this, there is a box containing 'EI'. At the bottom of the diagram, there is a box containing 'RET'. An arrow points from the 'RET' box back to the 'JMP 8070' label, indicating the return path.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, the process the device somehow sends an RST instruction to the microprocessor. So, microprocessor now executes the RST instruction received from the device on the data

bus lines. It saves the address of the next instruction on the stack and jumps to the appropriate entry in the interrupt vector table. So, the appropriate entry is determined by the RST instruction that it is executing. So, based on the value of n in the RST n instruction, so it goes to a particular location in the IVT, Interrupt Vector Table and it starts executing from that point.

Now, the IVT entry, so it must redirect the microprocessor to the actual service routine. Actually what happens is that in that; so all the 8 RST services that we have. So, they are clubbed together. So, we do. So, if we start writing the interrupt service routine from that point onward in the IVT, then it will not have space for the for the interrupt the interrupt service routines for other interrupts.

So, what is done? We normally put a jump instruction there, so that the processor executes the jump instruction and somewhere later the ISR maybe there. So, it is like this that suppose upon getting the interrupt the, so it comes to this particular location in the IVT. So, if your actual interrupt service routine is located from say location say 8000 onwards. So, this is the actual ISR. So, what it does is, it put puts a jump instruction here jump 8000.

So, upon getting the interrupt the processor comes to this point and execute the jump instruction. As a result it goes to the ISR and executes the Interrupt Service Routine. So, that is how this I that that is what is meant here the IVT entry must redirect the processor to the actual service routine. Now, the service routine must include the EI instruction because as soon as an interrupt occurs in the restart sequence the, the all the interrupts are disabled. And so, the very first thing that the interrupt service routine should do is to use the instruction EI to re-enable the interrupt process.

So, if it is not re enabled, then a subsequent interrupts will not be sensed by the processor. So, that may be a problem. Of course, if the user wants that I the my interrupt service routine should not be interrupted further then may not put the EI instruction at the beginning rather maybe towards the end of the interrupt service routine, this EI instruction is kept. So, at the end of the service routine, there must be one return instruction that returns the execution to where the program was interrupted.

So, every interrupt service routine should end with a return instruction and upon getting this return (Refer Time: 06:00), the processor will take out the return address from the stack and it will return to that location.

(Refer Slide Time: 06:08)

The 8085 Non-Vectored Interrupt Process

- The 8085 recognizes 8 RESTART instructions: RST0 - RST7.
 - each of these would send the execution to a predetermined hard-wired memory location:

Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

Handwritten notes: RST (n), n*8

Here is actually that restarts that IVT interrupt vector table. So, we have got this as I said that there are 8 possible non vectored interrupt RST instructions, RST 0 through RST 7 and each of these should send the execution to a predetermined hard wired location.

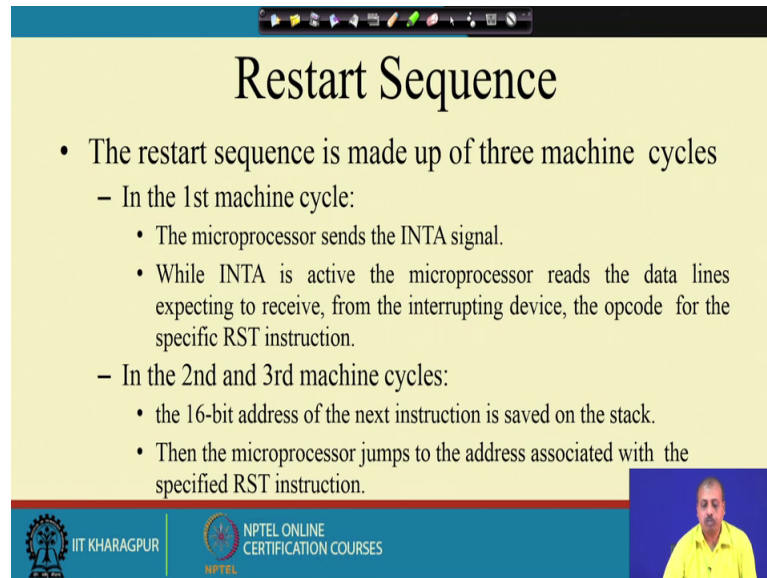
So, if the instruction is RST n, so what the processor does is that this n is multiplied by 8. And whatever be the result, so it jumps to that particular location. For example, this RST 0 it is transferring the control to the memory address 0, then RST 1, it will transfer the control to memory address 8, RST 2 will transfer to memory address 16 like that.

So, you can see that the RST 0 instruction is equivalent to this call 0000 hex. Similarly, RST 1 is equivalent to call 0008 hex. So, like that we have got this RST instructions they are equivalent to this call instructions. So, so you can understand that between RST 0 and RST 1, we have got only 7 by, only 8 bytes of location are free. So, if your ISR is very short, it can hold it can be held within 8 bytes.

So, you can start immediately you can start writing the interrupt service routine from the location from that location itself. So, but normally we any meaningful ISR will have more than 8 bytes with length. So, it is not possible to hold that entire interrupt service



routine. So, that is why, we normally put a jump instruction there and jump to the actual ISR service actual ISR which is located somewhere later in the memory. So, this is how this processor this RST instructions are sensed by the processor.


(Refer Slide Time: 08:11)



Restart Sequence

- The restart sequence is made up of three machine cycles
 - In the 1st machine cycle:
 - The microprocessor sends the INTA signal.
 - While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
 - In the 2nd and 3rd machine cycles:
 - the 16-bit address of the next instruction is saved on the stack.
 - Then the microprocessor jumps to the address associated with the specified RST instruction.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES



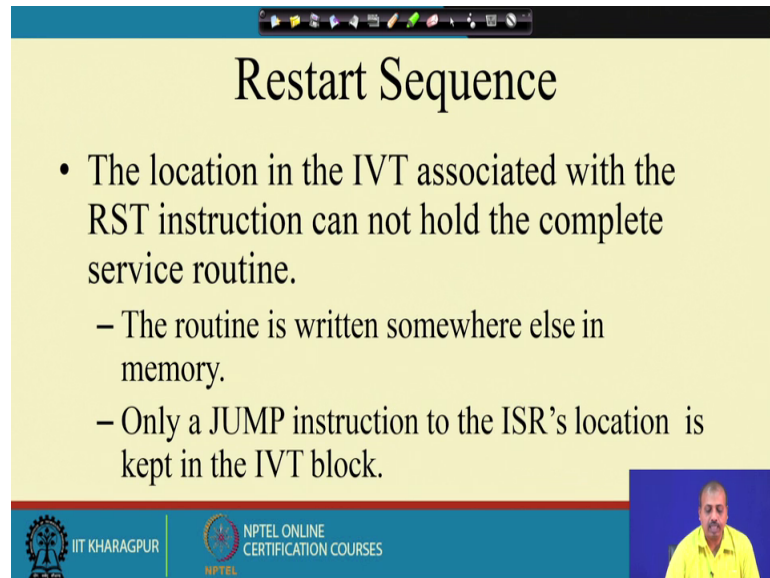
So, coming back to the restart sequence, so restart sequence it is made up of three machine cycles. So, that it is actually the execution of the RST instruction. So, how it does in the first machine cycle, the microprocessor sends the INTA signal and while this INTA is active the microprocessor reads the data lines expecting to receive from the interrupting device, the opcode for the specific RST instruction. So, RST n is a single byte instruction out of in which the 3 bits are reserved for the value of n and rest of the bits identify the opcode RST.

So, what it says is that in the first machine cycle. So, maybe, maybe the processor is executing some instruction and the end of it finds that there is one interrupt ok. So, RST n there is an interrupt on the INTA line. So, in the next machine cycle next instruction, after completing the current instruction; in the next machine cycle, it will activate the INTA signal. Getting the INTA signal, we said that the device is expected to put the address of the ISR onto the data bus which is in terms of the of the one RST instruction.

So, after putting INTA signal onto the to the device, the processor will read the data bus lines because it is expecting that the device has put one RST instruction onto the data bus lines. So, after getting that, in the second and third machine cycles, the next return

address is saved onto the stack the 16 bit return address for the next instruction is saved onto the stack. And depending upon this value of n in the RST n instruction, the processor jumps to the address associated with the specified RST instruction. For example, if it is RST 2, it will jump to the location 16. So, like that it will jump to the corresponding location and start executing program from there.

(Refer Slide Time: 10:10)



The slide is titled "Restart Sequence" and contains the following text:

- The location in the IVT associated with the RST instruction can not hold the complete service routine.
 - The routine is written somewhere else in memory.
 - Only a JUMP instruction to the ISR's location is kept in the IVT block.


The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a presenter in a yellow shirt.

The location in the IVT associated with the RST instruction cannot hold the complete service routine as I said that service routine, we have got only 8 bytes free for holding this interrupt service routine at the location of the IVT. So, actual routine is written somewhere else in memory and only a jump instruction to the ISRs location is kept in the IVT block. So, normally this is the standard procedure for writing the interrupt service routine of any processor. So, in the IVT we keep a jump instruction and from there, it jumps to a particular some address and in that address the actual ISR is located.

(Refer Slide Time: 10:49)

Hardware Generation of RST Opcode

- How does the external device produce the opcode for the appropriate RST instruction?
 - The opcode is simply a collection of bits.
 - So, the device needs to set the bits of the data bus to the appropriate value in response to an INTA signal.



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the question is how and how a device can generate the opcode RST opcode ok. So, this is a very important thing because device is for some for some special purpose, but it has to do this additional job if it wants to be interface with 8085 using this INTR line. So, how it does the, as we know, that any opcode is nothing but 1 8 bit collection of bits ok. So, device needs to set the bits of the data bus to the appropriate value in response to a to one INTA signal.

So, this is what has to be done.

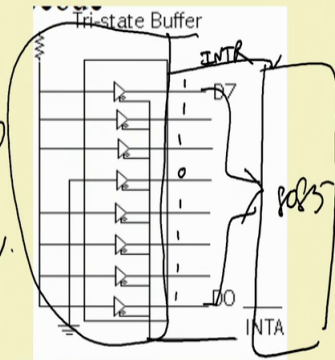
(Refer Slide Time: 11:26)

Hardware Generation of RST Opcode

RST 5's opcode is EF =

D	D
7	6
5	4
3	2
1	0
1	1
1	1
1	1

5x8=40 Dev.



Tri-state Buffer

8085

INTA

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

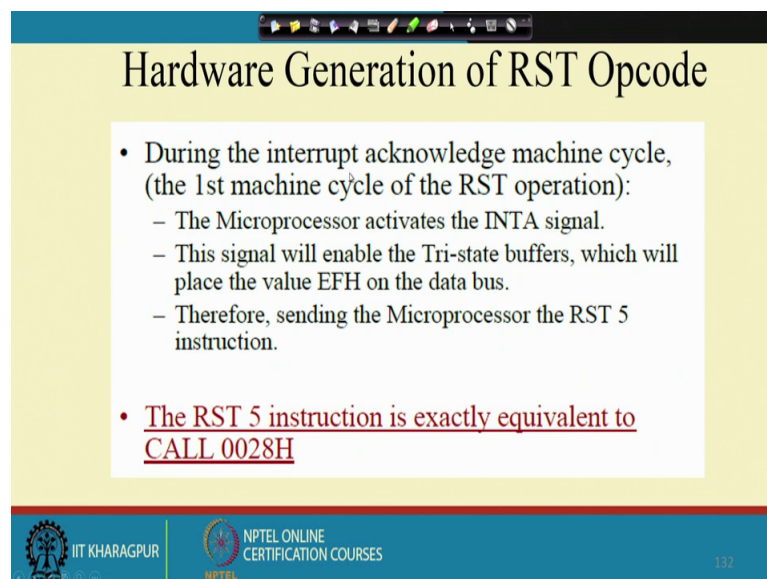
This is the strategy suppose. So, this is suppose this device this is a device and if this device has got a tri state buffer which is an 8 bit bus and the. So, this side I have got the microprocessor. So, this side we have the microprocessor, this side we have got that 8085 and this D 0 to D 7, they are actually forming the data bus.

So, this is the data bus line for the 8085. Now, when this interrupt signal came to the processor the INTR signal came to the processor, the processor has activated this INTA bar line and this INTA bar line has been connected to a set of tri state buffers. So, when this line is 0 all this buffers are enabled. As a result, if you look into this connection pattern. So, these bits are all 1. So, and this bit is 0 and this bits are again all 1. So, if you look into the 8085 manual, you will find that this particular bit pattern 1 1 1 0 1 1 1 1. So, this corresponds to this RST 5 instruction, ok.

So, what happens is, when the device gives an interrupt and the processor gives interrupt acknowledgement, so, this tri state buffers within the device are enabled. So, that this is inside the device this whole thing is inside the device. So, when these interrupt acknowledgement comes. So, the device can very easily put the data bus on to the data bus this bit pattern 1 1 1 0 1 1 1 1 as a result, 1 8085 will read it will understand that this is a RST 5 instruction and it will jump to the location 5 into 8 equal to 40.

So, it will go to the location 40 and start executing from there. And as I said, most probably there will be a jump instruction to the actual routine at the location 40.

(Refer Slide Time: 13:38)



Hardware Generation of RST Opcode

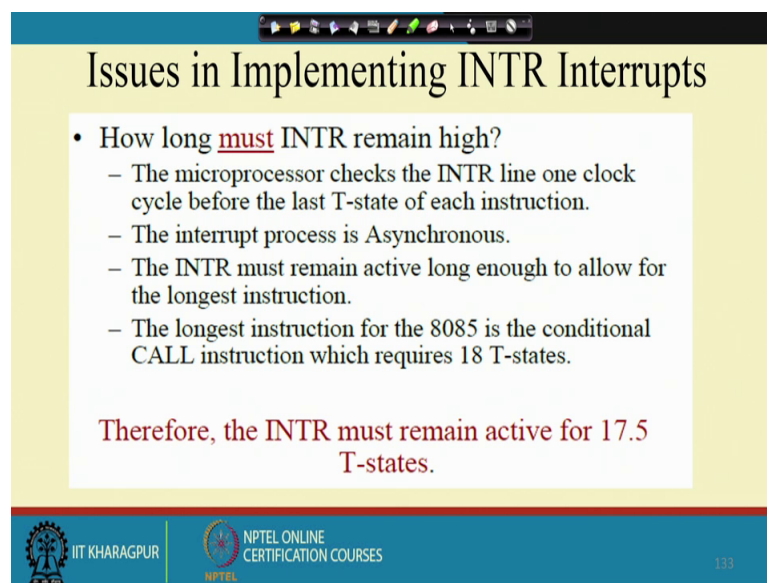
- During the interrupt acknowledge machine cycle, (the 1st machine cycle of the RST operation):
 - The Microprocessor activates the INTA signal.
 - This signal will enable the Tri-state buffers, which will place the value EFH on the data bus.
 - Therefore, sending the Microprocessor the RST 5 instruction.
- The RST 5 instruction is exactly equivalent to CALL 0028H

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 132

So, during the interrupt acknowledge machine cycle, the microprocessor activates the INTA signal the signal will enable the tri state buffers which will place the value EFH onto the data bus. Therefore, the sending the microprocessor RST 5 instruction; so this is the standard procedure by which this interrupt INTR line and the device gets the corresponding, device produces the corresponding RST instruction for doing the for the processor to start the interrupt service routine.

Now, you see that this is one of the approach by which a device can generate this RST instruction. So, there can be many other ways by which it can be done. So, this is just an example ok. So, some device may be more intelligent and it can do it in a better fashion, but this is one of the very basic technique where you do not need anything more than a few tri state buffers to generate the RST instruction.

(Refer Slide Time: 14:40)



Issues in Implementing INTR Interrupts

- How long **must** INTR remain high?
 - The microprocessor checks the INTR line one clock cycle before the last T-state of each instruction.
 - The interrupt process is Asynchronous.
 - The INTR must remain active long enough to allow for the longest instruction.
 - The longest instruction for the 8085 is the conditional CALL instruction which requires 18 T-states.

Therefore, the INTR must remain active for 17.5 T-states.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 133

Next question is, suppose the INTR line has been activated, so, a device wants to interrupt the processor microprocessor and it has raise the INTR signal.

Now, how long this INTR should remain high now as I said that the microprocessor checks the INTR line, 1 clock cycle before the last t state of each instruction. So, whenever a microprocessor whenever microprocessor executing some instruction, when it comes to the last but one t state. So, at that time it will try to see whether this I interrupt, interrupt has come or not. The interrupt process is asynchronous. So, it can

come at any point of time. So, naturally this interrupt must remain active long enough to allow the longest to allow for the longest instruction.

So, that to be on the safe side, if you want that your interrupt should always be sensed by the microprocessor, then it has to be done in such a fashion that even for the longest instruction, the interrupt line is active till the last but one clock cycle. In case of 8085, the longest execution time is for conditional call instruction which requires 18 T-states. So, to be sensed properly the INTR pin must remain active for 17.5 T-states ok

So, now you can understand that if I have got a 2 megahertz clock frequency than that is 17.5 divided by 2 megahertz. So, that, so much of time this interrupt line must be high. So, if the pulse is shorter than that if the interrupt line is not high for that much time it may. So, happen that the processor misses that point and it is not sensed by the processor.

So, this is the guideline for raising the INTR signal.

(Refer Slide Time: 16:32)

The slide is titled "Issues in Implementing INTR Interrupts". It contains a bulleted list of points:

- How long can the INTR remain high?
 - The INTR line must be deactivated before the EI is executed. Otherwise, the microprocessor will be interrupted again.
 - The worst case situation is when EI is the first instruction in the ISR.
 - Once the microprocessor starts to respond to an INTR interrupt, INTA becomes active (=0).

Below the list, a red text box states: "Therefore, INTR should be turned off as soon as the INTA signal is received." The slide footer includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo. A small video inset in the bottom right corner shows a man in a yellow shirt speaking.

And how long can the INTR remain high? So, that is once it is made high how long can it remain high? The INTR line must be deactivated before the EI signal is executed. So, what, so as because as soon as this interrupt occurs. So, the processor automatically disables the interrupt and before enabling the interrupts again, so you should deactivate the INTR signal otherwise the microprocessor will find that again another interrupt has

come. So, it will take it the same interrupt. So, it will be sensed twice. So, it will be taken as two different interrupts.

Now, in case of worst, in the worst case situation the in the ISR the very first instruction may be the EI instruction. So, once the microprocessor starts to respond to INTR interrupt, INTA becomes active. So, that way because this EI is the first instruction, so it becomes active. So, INTR should be turned off as soon as the INTA signal is received as soon as the processor gets the as soon as the device gets the interrupt acknowledge signal. It should withdraw the INTR signal. Otherwise there is a chance that in the interrupt service routine. This enable interrupt will be the very fast operation as a result the processor will sense the interrupt once more ok.

So, this is the guideline. So, it should be, so when the INTR when the to interrupt the processor properly, device should be ready to keep the interrupt line high for 17.5 clock cycles and as soon as the interrupt acknowledge signal is received, it should deactivate the INTR signal.

(Refer Slide Time: 18:10)

The slide is titled "Issues in Implementing INTR Interrupts". It contains a bulleted list of questions and answers. The first question is "Can the microprocessor be interrupted again before the completion of the ISR?". The answer is "only if you allow it to". A second question is "If the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done." To the right of the text is a hand-drawn diagram of a vertical rectangle with a wavy line on the left side, representing an interrupt signal. The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset in the bottom right corner shows a man in a yellow shirt speaking.

Issues in Implementing INTR Interrupts

- Can the microprocessor be interrupted again before the completion of the ISR?
 - As soon as the 1st interrupt arrives, all maskable interrupts are disabled.
 - They will only be enabled after the execution of the EI instruction.

Therefore, the answer is: "only if you allow it to".
If the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done.

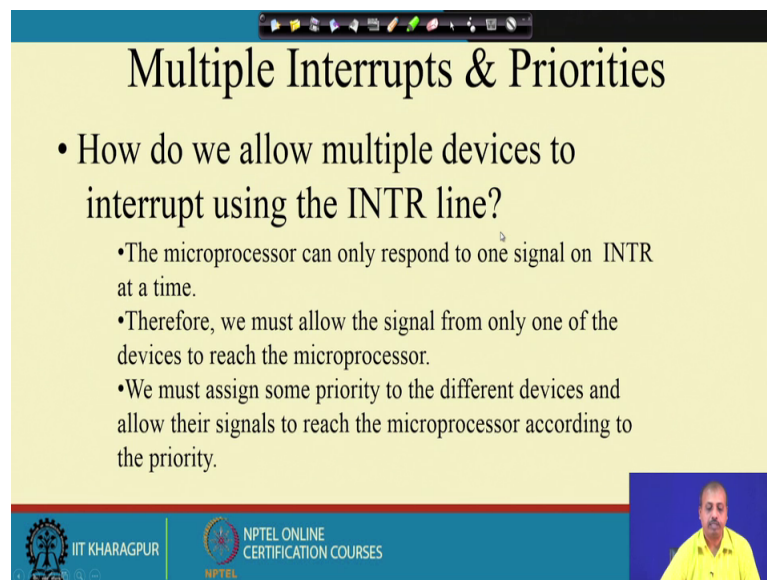
Issues in implementing INTR interrupts, so like can the microprocessor be interrupted again before completion of the ISR. So now, the question is dependent on the style in which the ISR has been written. As soon as the first interrupt arrives, all maskable interrupts are disabled and INTR being a maskable interrupts. So, it also gets disabled and they are enabled only after execution of the EI instruction.

So, this is specified by the processor designer. So, 8085 designer, so they have they have told that this is the guide line. Now, the point is that if you are, if you allow the processor to be interrupted again like if you are, if this is the ISR that I am I have written and my EI is put here itself, now there is a very high chance that while I am still in the ISR another interrupt comes from the device.

So, that way it is again another interrupt will be sensed by the processor and it will again come to restart the ISR. So, that can happen or in the other case, you can put the EI at the bottom of this interrupt service routine as a result this will not be interrupted again. So, if it is put at the end, then this interrupts will not be coming again till the first one is over. That is why the answer is only if you allow it to if the EI instruction is placed early in the ISR, other interrupt may occur before the ISR is done. So, that that is the problem.

So, if you do not want that, you should put this EI instruction at the bottom. But again putting at the bottom has got some other consequence because you can that will also differ other more important instruction interrupts as well.

(Refer Slide Time: 20:04)



The slide is titled "Multiple Interrupts & Priorities" and contains the following text:

- How do we allow multiple devices to interrupt using the INTR line?
 - The microprocessor can only respond to one signal on INTR at a time.
 - Therefore, we must allow the signal from only one of the devices to reach the microprocessor.
 - We must assign some priority to the different devices and allow their signals to reach the microprocessor according to the priority.

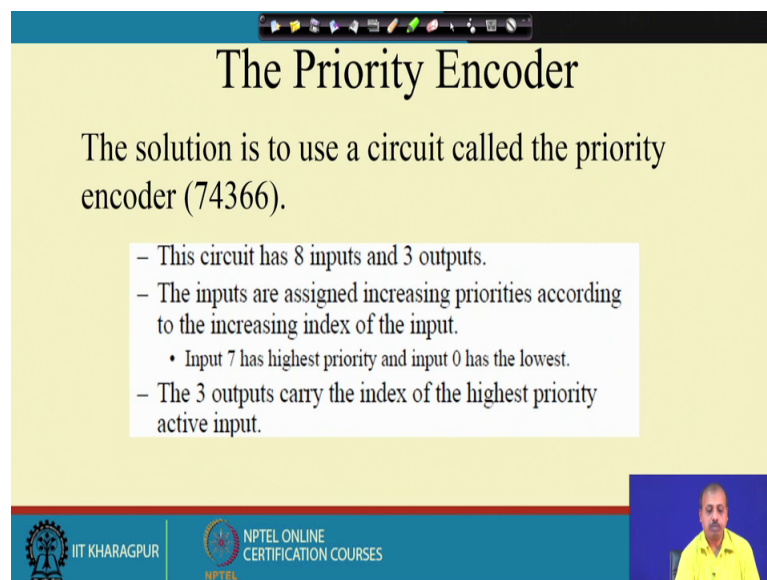
The slide also features logos for IIT Kharagpur and NPTEL Online Certification Courses, and a small video inset of a speaker in the bottom right corner.

So, if you have got multiple interrupts, then which interrupt should be having higher priority? So, that is one problem. Now, how do you allow multiple devices to interrupt using the INTR line? So, I have got a single interrupt line. Now if I have got a number of devices that can that can give interrupt to the, that can use this non vectored interrupt to the 8085 processor.

Now, how to do this? Now, microprocessor can respond to only one signal on the INTR at a time. So, it can since it is a uniprocessor system, so, it will be responding to only one device request. So, we must allow the signal from only one of the devices to reach the microprocessor. So, that is important. So, somehow we have to do that that processor does not get interrupt from multiple sources, because there is ultimately there is only one pin in the microprocessor.

So, you must assign some priority to different devices and allow their signals to reach the microprocessor according to the priority. So, this is the issue that is we have to somehow prioritize the devices and some device which has got higher priority should be able to interrupt the microprocessor before other low priority devices are interrupting it.

(Refer Slide Time: 21:24)



The slide is titled "The Priority Encoder" and contains the following text:

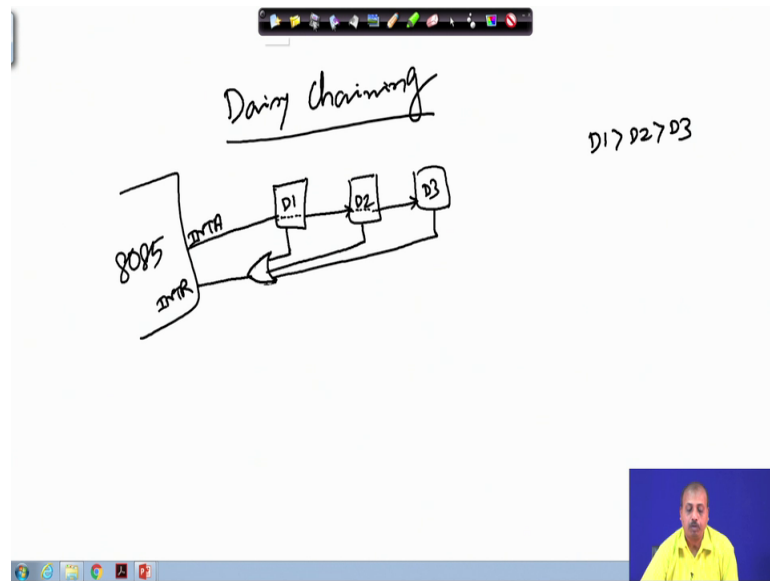
The solution is to use a circuit called the priority encoder (74366).

- This circuit has 8 inputs and 3 outputs.
- The inputs are assigned increasing priorities according to the increasing index of the input.
 - Input 7 has highest priority and input 0 has the lowest.
- The 3 outputs carry the index of the highest priority active input.

The slide footer includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo. A small video inset in the bottom right corner shows a man in a yellow shirt.

So, how to do this? So, there is a priority encoder 74366. So, I will come to that later. So, there is a simpler scheme by which you can do this which is known as daisy chaining.

(Refer Slide Time: 21:36)



So, daisy chaining policy is like this, that suppose I have got a number of devices. So, this is D 1, D 2, D 3, like that and I have got the microprocessor 8085 here.

Now, what I want is that D 1 will have the highest priority followed by D 2 followed by D 3 fine. Now how to do this thing? So, what we can do? So, we can take this interrupt line from this individual devices and pass them through an OR gate pass them through an OR gate to get and they were like connected to the interrupt line of the processor. So, this is the OR gate we connect them and connect it to the interrupt line of the micro of the microprocessor.

Now, this interrupt acknowledge line, so this is connected to the interrupt acknowledge of the first D 1 and then this is the INTA. And this INTA line, so it is passed from the device D 1 to device D 2 and from device D 2 to device D 3. It is passed like this. So, the idea is that if first device has raise the interrupt when the interrupt acknowledgement comes. So, it does not pass this interrupt acknowledge line to the output. So, it just consumes it.

So, it understands this is for me. So, it generates the corresponding RST instruction for the 8085. Now, if it happens the if it happens like this that this D 1 has not generated the interrupt, in that case this INTA line will be passed by D 1 and it will reach D 2. If D 2 had raised the interrupt, then it will use this interrupt acknowledge line to generate the next to generate the next RST instruction. Otherwise it will pass this interrupt

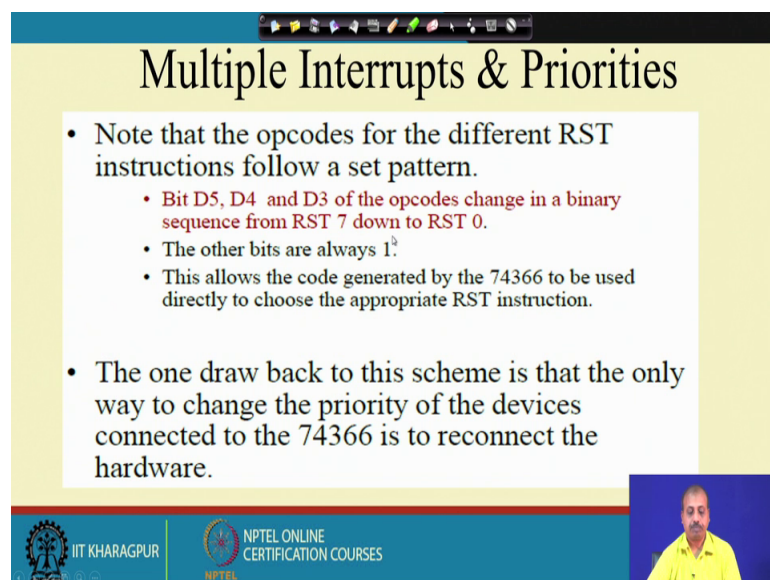
acknowledge line to the device D 3. So, this way you see the devices which are closer to the processor, they are having higher priorities compared to the devices that are further away from the processor.

So, this particular policy is known as Daisy Chaining policy. So, this is very simple to implement by means of some very simple logic. You can decide whether this devices can have some simple logic to decide whether the interrupt acknowledge line should be passed or not ok. So, this is one policy by which we can do this. The other policy that we have is by means of a Priority Encoder.

So, this 74366 chip, so, this is a priority encoder, this circuit has got 8 inputs and 3 outputs. The inputs are assigned increasing priorities according to the increasing index of the input. So, 0 has got the lowest priority going to 7. So input 7 has the highest priority and 0 has the lowest priority. And three outputs, they carry the index of the highest priority active input. So, this is a decode encoder. So, this encodes this 8 bit input into a 3 bit output.

So, this actually identifies the index of the interrupt line that is the highest priority interrupt line that we have.

(Refer Slide Time: 25:10)



The slide is titled "Multiple Interrupts & Priorities" and contains the following text:

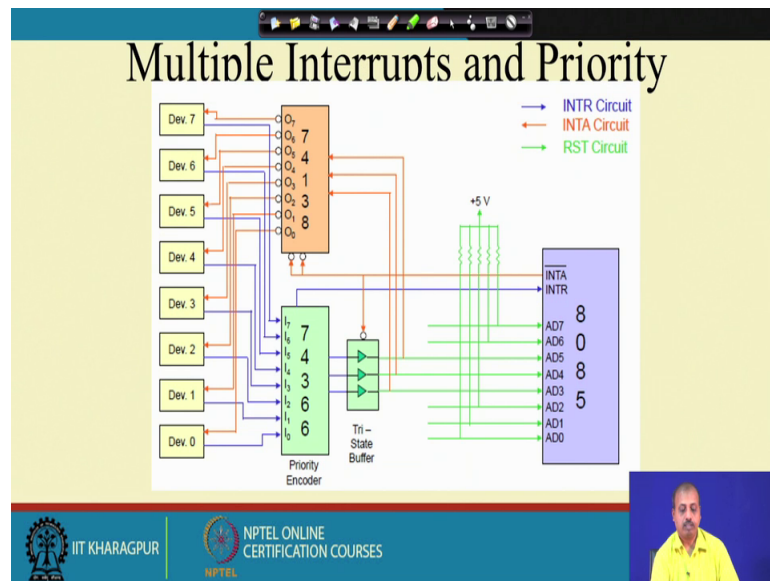
- Note that the opcodes for the different RST instructions follow a set pattern.
 - Bit D5, D4 and D3 of the opcodes change in a binary sequence from RST 7 down to RST 0.
 - The other bits are always 1.
 - This allows the code generated by the 74366 to be used directly to choose the appropriate RST instruction.
- The one draw back to this scheme is that the only way to change the priority of the devices connected to the 74366 is to reconnect the hardware.

The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

OpCodes for different RST instructions follow a particular pattern. So, if you if you look into this opcode for the RST instruction, you will find that it is an it is an 8 bit instruction

in which this bits D 5, D 4 and D 3. So, they will determine the number n in the RST n, they will determine the value of n. And the rest of the bits are always 1, rest of the all other bits are always 1. So, bits D 7, D 6 and then D 2, D 1, D 0. So, they are all 1. So, this allows a 74366 to that to generate this RST instruction directly. However, the problem that we have is that to change the priority, you have to change the connection pattern of the devices

(Refer Slide Time: 26:02)



So, we have got a connection like this. So, this is our 8085 processor. Now, this 74366, so, this is the priority encoder. So, when this device is getting this interrupts, so, this I 0, I 1, I 2 like that. So, accordingly, it this it passes through this tri state buffer when this interrupt acknowledge signal comes. So, this buffer is enabled as a result this 3 bit output. So, it is coming onto this line.

(Refer Slide Time: 26:37)

The 8085 has 4 Masked/Vectored interrupt inputs.

- RST 5.5, RST 6.5, RST 7.5
 - They are all **maskable**.
 - They are **automatically vectored** according to the following table:

Interrupt	Vector
RST 5.5	002CH
RST 6.5	0034H
RST 7.5	003CH

- The vectors for these interrupt fall in between the vectors for the RST instructions. That's why they have names like RST 5.5 (RST 5 and a half).

The slide also features the IIT Kharagpur and NPTEL logos at the bottom left and a small video inset of a presenter at the bottom right.

This 3 bit output is coming here and that connects to the data bus line D 3, D 4 and D 5 and rest of the lines. So, they are raised high ok. So, by this they are all raised high.

Now, when this and this in the 74366 has got an interrupt line; so this interrupt line gives this INTR signal. So, that is a given to a connected to the 8085 similarly INTA bar line is connected to this 74138. So, this is another 328 decoder. So, this decoder is enabled based on whichever device has generated this, whichever device has been accepted here for the highest priority one. So, that value is again you see passed here. So, as a result, the corresponding device will get the interrupt acknowledgement, some sort of interrupt acknowledgement signal.

So, we can say. So, that line will be made 0 and the rest of the lines will be made 1. Suppose this device 2 had raised the interrupt, device 2 is the highest priority device that had raised the interrupt, then what will happen, here you will get the pattern 0 1 0, and upon getting 0 1 0 here this O 2 line will be 0 and rest of the lines will be 1, so the two line being 0, so it will come to this device 2. So, device 2 sees it as an interrupt acknowledgement line as if the interrupt acknowledgement line has been passed to it. So it can accordingly reset the interrupt line and all to the rest of the things.

So, this way we can use this 74366 and 74138. So, these two encoder priority encoder and decoder in conjunction to interface multiple interrupts with different priority schemes.