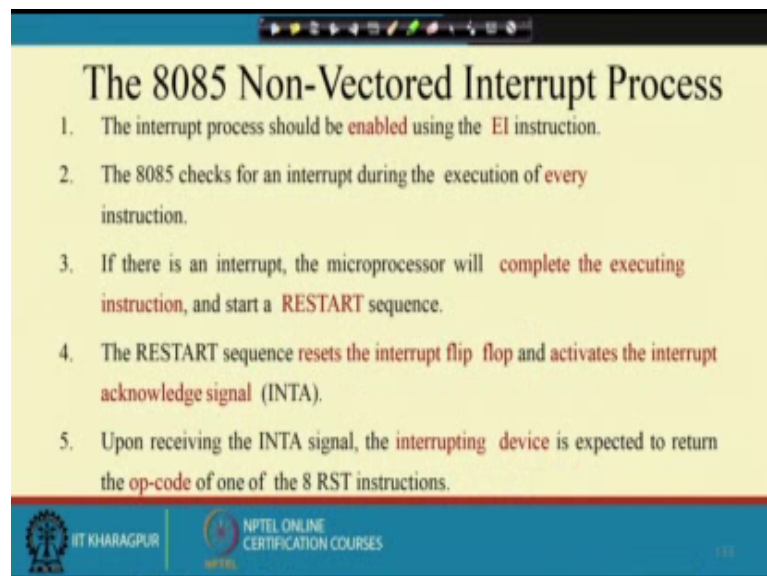


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture – 17
8085 Microprocessors (Contd.)

Next we will see the interrupts process like when an interrupt occurs then how this that interrupt is serviced in 8085, both non vectored interrupts and the vectored interrupts. So, we will first look into the non vector interrupt.

(Refer Slide Time: 00:30)



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.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The non vectored interrupts process is like this. First of all the process should be enabled using the enable interrupt instruction. So, if this enable interrupt is not executed by the processor then the processor will not be able to receive any of the interrupts.

In fact, when the processor is reset this enable interrupt, this is they all the interrupts are enabled and for the purpose of different programs. So, it may so happen that the enable the interrupts have been disabled through the DI instruction. So, if this EI is not done then this interrupt will not reach the processor. So, the first thing to ensure is that the interrupts are enabled using the EI instruction.

Now, as we know that interrupts can occur at any point of time during execution of instructions by the processor. So, what the processor does is irrespective of the point at

which the interrupt has occurred. So, interrupts will be checked whether the processor will check for the interrupt at the end of every instruction. So, typically in the last machine cycle the interrupt is checked whether any interrupt has occurred or not. So, we will see after a few lectures that how to decide like for how much time this interrupt pin should be activated to get an interrupt. So, we can understand that the interrupt should be activated for at least a complete duration of one instruction. So, if we look into the instruction set whatever be the largest instruction execution time. So, the interrupt must be enabled for that much of time to get the interrupt detected.

So, if the interrupt occurs then what the system does the microprocessor will first complete executing the current instruction. So, this is very important thing to note. It is not that interrupt has occurred and the processor will leave the execution in the middle of middle of an instruction and go to the interrupt service routine it does not happen like that. The first thing it does is the current instruction that the processor is executing is completed and once the instruction is completed then it will start a restart sequence, how that interrupt service routine can be started so that is the restart sequence.

So, first of all this restart sequence will reset the interrupt flip flop this is the first thing that is done. So, the interrupt enable flip flop that we have seen. So, that will be deactivated at the beginning of this restart operation and it activates the interrupt acknowledge signal. So, this non vectored interrupt in 8085 INTR is the non vectored interrupt line. So, when that interrupt is detected the processor will first reset the interrupt acknowledge, interrupt flip flop and then it will activate the interrupt acknowledge signal INTA. As we know that INTA is an active low signal, this will be made low to make it active.

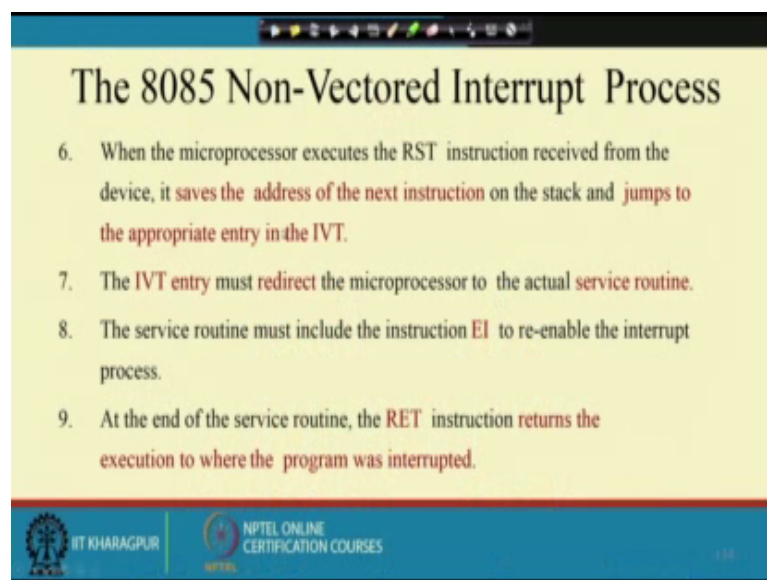
Now, the protocol is like this that interrupt acknowledge line should be connected somehow to the device which has generated the interrupt and on getting this interrupt acknowledgment the device should provide some address from which the interrupt service routine has been located in the memory.

Now, as a system designer we will know like in which, at for what memory location the interrupt service routine for a particular device has been loaded. So, the ISR address should be provided by the device. How is it provided? It is provided by means of some

RST instructions. So, there are 8 RST instructions and any of these, these are each of these RST instruction is of the format RST n where n is a 3 bit number and then this RST n is a single byte instruction.

So, what happens is that upon getting this interrupt processor enters into an interrupt acknowledge cycle. In this interrupt acknowledge machine cycle this interrupt acknowledge pin is made low and it is expected that on the data bus the opcode for one of the RST n instructions will be available and upon getting that particular opcode. So, processor will decide to which address it should go and accordingly it will go to the corresponding address.

(Refer Slide Time: 05:05)



The slide is titled "The 8085 Non-Vectored Interrupt Process". It contains four numbered points:

6. 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.
7. The IVT entry must redirect the microprocessor to the actual service routine.
8. The service routine must include the instruction EI to re-enable the interrupt process.
9. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

The slide footer includes the IIT KHARAGPUR logo and the text "NPTEL ONLINE CERTIFICATION COURSES".

So, when the processor executes this RST instruction received from the device it saves the address of the next instruction in the stack. So, this is the now, it is the execution of the RST instructions upon getting the RST instruction the first thing is that it will be saving the return address into the stack. So, return address is saved into the stack and it the processor will jump to the appropriate interrupts, appropriate entry in the interrupt vector table.

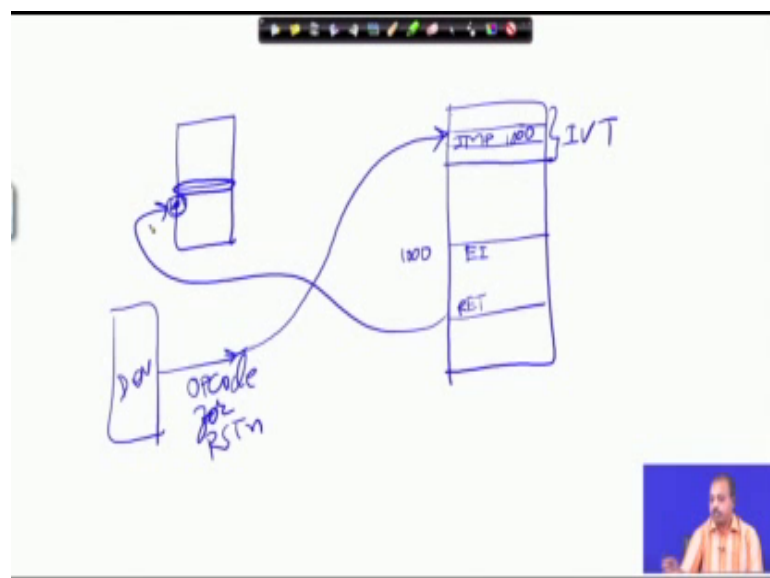
And this IVT entry, it has to redirect the processor to the actual service routine. So, as I said that each of this entire vector table either location like, so 8 such locations will constitute to one of these RST codes and whatever be the value, based on that it will reach a particular location in the memory in the page 0 0 2 f f in that page it will reach

that particular location and then it will start executing from that point. So, if your interrupt service routine is located from our different address we should put a jump instruction there. So, that now, the system jumps to the processor jumps to that particular location.

So, that is the responsibility of the IVT until enter factor table entry. So, normally it is we put a jump instruction there to go to the actual service routine. One thing that the service routine should do is that it should enable the interrupts using EI instruction because as soon as this interrupt occurs the when we go to the interrupt acknowledge cycle this or the interrupt gets disabled the INTR gets disabled. So, it is important that the interrupt service routine it performs on enable interrupts to re enable the interrupts process otherwise the future interrupts will not be accepted.

And at the end of the service routine the rate instruction should be there and this rate instruction will return the execution to the point where the program was interrupted. So, as we have seen that when in the INTS I go in the RST opcode is getting executed. So, system saves the return address into the stack and on getting the rate instruction that address is retrieved and that address will be loaded into the program counter. So, that the processor will return to the instruction which it has the instruction just after the one at which the interrupt was received. So, that way it continues.

(Refer Slide Time: 07:51)



So, if we just, if we look into the process more clearly, so I can say that if this is the program at which the interrupt has occurred say when the instruction. So, this particular instruction was getting executed at that time interrupt has occurred. So, the processor will come to this. So, on RST code will be generated from the device if this is your this is the device. So, somehow this device will produce an opcode for RST opcode, for RST n instruction and this will redirect the processor to some address. So, this is, this will redirect if this is the full memory.

So, the first part is the IVT interpreter table it will take the processor to a particular location in the IVT and in that location I can have a jump instruction jump 1000 say. So, that my I actual ISR is loaded from 1000. And while this opcode RST one is executed the return address that is this address has been saved into the stack. So, this address is saved into the stack. Now, from this jump 1000, it will (Refer Time: 08:57) come here. So, one important thing that it should do here is the enable interrupt otherwise the interrupts will be disabled and finally, there should be a rate instruction for return.

So, when it gets the rate instruction then from the stack these address, these address will be retrieved from the stack and the program counter will be loaded with this so that my control comes back to this point. So, this is the sequence in which this interrupt will be executed.

(Refer Slide Time: 09:34)

The slide is titled "The 8085 Non-Vectored Interrupt Process". It contains a list of 9 steps describing the interrupt process. The steps are:

6. 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.
7. The IVT entry must redirect the microprocessor to the actual service routine.
8. The service routine must include the instruction EI to re-enable the interrupt process.
9. At the end of the service routine, the RET instruction returns the execution to where the program was interrupted.

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, in case of 8085, there are 8 such restart instruction the RST instruction that I said. So, there are 8 such instructions named RST 0, through 7 and each of this would send the execution to a particular predetermined hardware memory address.

(Refer Slide Time: 09:38)

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:

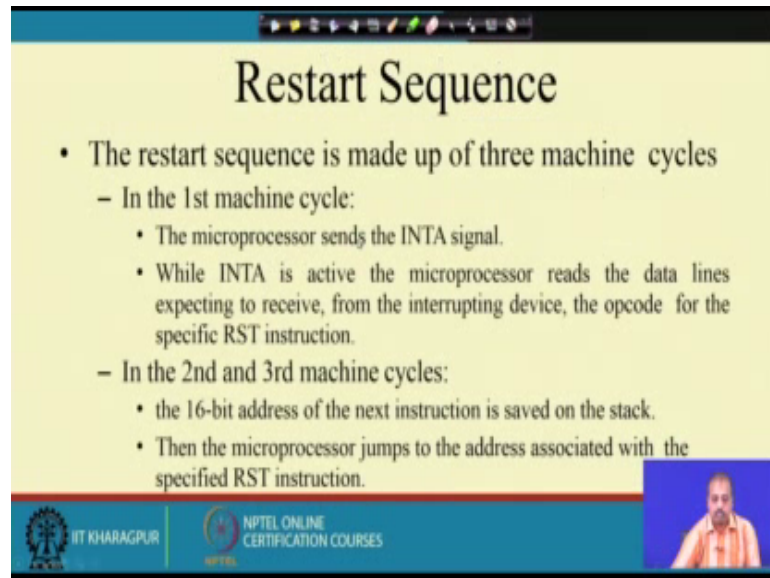
Restart Instruction	Equivalent to
RST0	CALL 0000H
RST1	CALL 0008H
RST2	CALL 0010H
RST3	CALL 0018H
RST4	CALL 0020H
RST5	CALL 0028H
RST6	CALL 0030H
RST7	CALL 0038H

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, RST 0 is equivalent to call 0 0 0 0. So, it will send the control to the location 0 also it is same as pressing the reset button activating the reset pin of the processor or this RST 0. So, it will transfer the control to the location 0 0 0 0 then RST 1 is 0 0 0 8. So, RST 2 is 0 0 1 0 that is 16 in decimal. So, you see that what is happening is that in general if the instruction is RST n if the instruction is RST n. So, this n is multiplied by 8 and whatever be the value. So, processor will be jumping to that particular address. So, if it is a RST 4 then 4 into 8 is 32. So, it will jump to the address 0 0 2 0 x.

So, that way at that location the interrupt service routine should exist. So, it may be a jump instruction to the actual routine or if the routine is very small as you see that between two successive RST locations. So, we have got 8 byte space. So, if my interrupt service routine is very small then I can hold it in 8 bytes, so that may be sufficient or if it is not. So, then we have should put a jump instruction and go there. So, next we will have this, sequence is like this. So, restart sequence it has got 3 machine cycles in the first machine cycle which is known as interrupt acknowledge cycle or INTA cycle.

(Refer Slide Time: 11:22)



Restart Sequence

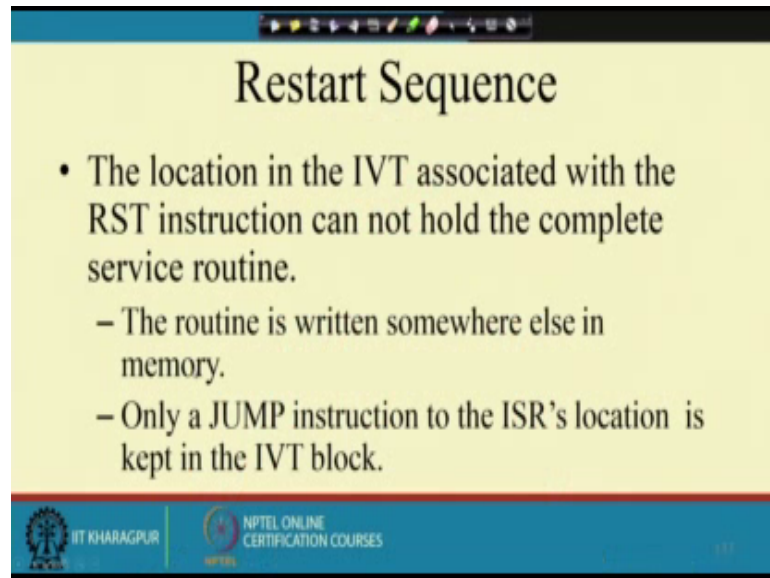
- The restart sequence is made up of three machine cycles
 - In the 1st machine cycle:
 - The microprocessor sends the INTA signal.
 - While INTA is active the microprocessor reads the data lines expecting to receive, from the interrupting device, the opcode for the specific RST instruction.
 - In the 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, the microprocessor sends the INTA signal. So, INTA line is made low and when the INT is active the processor will read the data lines and it will expect that on the data bus there will be the opcode for a specific RST instruction. As I said corresponding to the device. So, we have to map it onto one of these RST and then that device should put that RST opcode onto the data bus. So, this is the first machine cycle or INTA interrupt acknowledge cycle.

The second and third machine cycles what will happen is that the 16-bit address of the next instruction will be saved on to the stack that is the PC high and PC low those two values will be saved onto the stack and then the processor will jump to the address to which to the of the specified RST instruction. So, this is the whole sequence of operation that is done in the restart sequence. So, this way the processor will be going to the interrupting to the interrupt service routine.

(Refer Slide Time: 12:39)



Restart Sequence

- 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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES


Now, the location for the IVT it may not hold the complete service routine as I said. So, the routine may be written somewhere else and we can have a jump instruction at the ISR location to take here that may be kept in the IVT block. As we have already said that this interrupt table it may have a jump instruction only for the actual service routine.

Now, how to generate this RST opcode. So, that is a challenge, apparently it seems that how a device will generate the RST end instruction code. So, this is very simple in the fact that the designers they have made this RST coding in such a fashion that way the opcode, the core design of RST in such a fashion that it is quite easy to generate the RST opcode. So, we will see how it is doing it.

(Refer Slide Time: 13:28)

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.




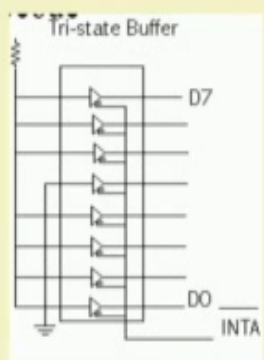
As we know that is nothing, but one 8 bit pattern that 8 bit pattern has to be generated on that response to the INTS signal. So, this is how we can we see that the RST opcode is generated.

(Refer Slide Time: 13:36)

Hardware Generation of RST Opcode

RST 5's opcode is EF =

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

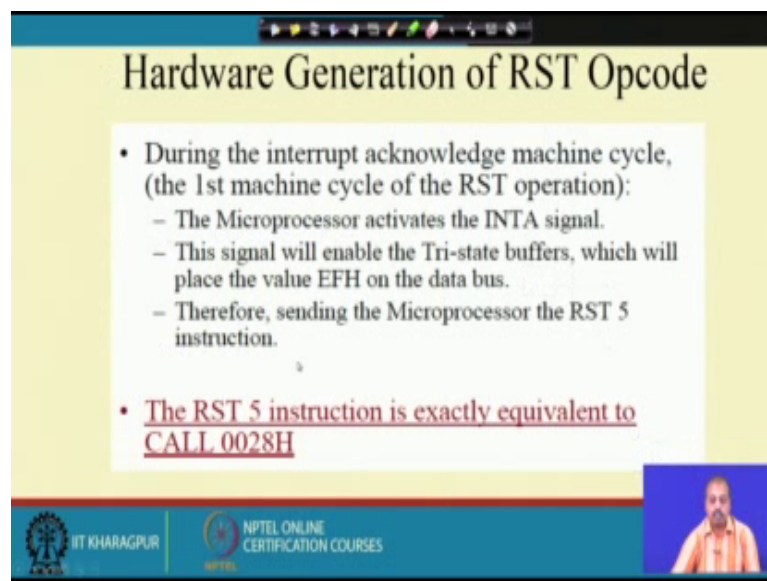


So, for example, suppose we are looking into the RST 5 instruction. So, in the RST instruction opcode, if you look into the bits then these bits are fixed like say D 7 that D 6 then the D 7 D 6 then actually this bit number 5 4 and 3, so the 3 4 and 5. So, they will contain the actual number that the opcode that are n number that we have the rest of the

bits are all 1. So, this is, 3 bit number 3 4 and 5. So, they are they will hold the pattern one 0 one and rest of the bits are all one. So, you see that we can have a simple device, you can have a simple tri-state buffer like this where for this particular instruction you see only the bit 4 has to be 0. So, bit 4 is made 0 and all other bits are tied high.

Now, in the interrupt acknowledge cycle this INTA bar line will be activated as a result this tri-state buffers will get enabled and the whatever, whatever be the content on this side of the buffer they will be available here. And you see as a result all the bits excepting D 4 they will get 1, only D 4 will get a 0. So, that will be treated, this D 7 to D 0. So, if it is connected to the data bus line of the processor then it will be understood as the opcode for RST 5. So, these, any other RST instruction rest of the bits remain unaltered only these bits bit number 3 4 and 5. So, they will be changing. So, RST is 0 to RST 7 these bits will change from 0 0 0 to 1 1 1. So, that way we can really easily generate the opcode for the RST.

(Refer Slide Time: 15:32)



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

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is the during interrupt technology machine cycle the first machines microprocessor will first enable the activated INTA signal. So, this will enable the tri-state buffers and then the values that will come on the tri-state buffer which is. So, that will come on to the databus. So, we will place the value on to the databus and it will come then. So, for the RST 5 port, it is EF. So, that is fine for any other opcode there are some other any other RST some other opcode will come. So, that way this RST 5 can be

there and it RST 5 we know that it is equivalent to call 0028 H instruction. So, it will branch to the location 0028.

(Refer Slide Time: 16:21)

The slide is titled "Issues in Implementing INTR Interrupts". It contains a bulleted list of points and a concluding statement. The list includes: "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.", and "The longest instruction for the 8085 is the conditional CALL instruction which requires 18 T-states." The conclusion states: "Therefore, the INTR must remain active for 17.5 T-states." The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES".

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.

Next, we answer the question that is how long should the interrupt line remain high. So, this is a very party named question because when you are designing a device to be operating with the microprocessor and the device will operate by we will send interrupts to the processor. So, how long the device should be able to make that in trouble on high, so that, it will be sensed by the processor. So, as we know, the microprocessor will check the INTR line one clock cycle before the last T-state of each instruction.

So, if one instruction will consist of a number of machine cycles in the last machine cycle last T-state last clock state at that point it takes for the interrupt line. So, if this interrupt line becomes deactive before that then that interrupt will not be sensed by the microprocessor. So, because this interrupt process is a synchronous. So, it can occur at any point of time. So, it must remain high enough. So, that this longest instruction can be executed and in case of 85 the longest instruction is the call instruction that takes 8een clock cycles. So, so if he takes 18 clock cycles and this INT line is checked on the last clock cycle. So, for 17.5 T-states the line should the line must be high.

So, you can find out the frequency of the processor multiplied by 17.5 that will tell you where the minimum duration that for which this interrupts line must be high. So, that it will guarantee that if the interrupt occurs the processor will definitely have a look into

that. So, this way with this answers the question that how long must the interrupt line remain high.

And next question is how long should it remain high.

(Refer Slide Time: 18:03)

The slide is titled "Issues in Implementing INTR Interrupts". It contains a bulleted list of three points. The first point is "How long can the INTR remain high?". The second point states that the INTR line must be deactivated before the EI instruction is executed, otherwise the microprocessor will be interrupted again. The third point states that the worst case situation is when EI is the first instruction in the ISR. The fourth point states that once the microprocessor starts to respond to an INTR interrupt, INTA becomes active (=0). To the right of the text, there is a handwritten diagram showing a box labeled "PUSH REGISTER" with an arrow pointing to a box labeled "EI". Below the diagram, the text reads "Therefore, INTR should be turned off as soon as the INTA signal is received." The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES".

Issues in Implementing INTR Interrupts

- 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).

Therefore, INTR should be turned off as soon as the INTA signal is received.

How long can the interrupts remain high? So, can I keep it high continually? So, this question can be answered like this. The interrupt line must be deactivated before the EI instruction is executed. Now, as I said that any interrupt service routine one of the important job that it does is to exist is to put the instruction EI enable interrupts somewhere and as a standard practice what is done is that this interrupt enable line is activated just before doing anything very much useful.

So, before going into the actual routine, this is done. So, any ISR code, normally what we do is that if this is the beginning of the ISR then first we push the registers that we have whatever register this is our will be using all those registers are pushed and after that we put the EI instruction. So, if somebody thinks that I do not need the register values then this part is absent. So, EI becomes the first instruction in the interrupt service routine and when that EI is put. So, if the EI is executed all the interrupts get enabled again.

So, you can understand now, that if I do not deactivate that device the interrupts pin coming from the device interrupts signal coming from the device before this EI

instruction is executed then, it will be taken as a second interrupt. So, processor may take that interrupt as the second interrupt. So, in the worst case this is the situation that EI can be the first instruction of the ISR and once the microprocessor starts to respond to the INTR interrupt, INTA becomes active. So, this becomes INTA becomes active and then EI also gets active. So, another interrupt will be taken.

So, it is a standard practice that interrupts should be turned off as soon as the INTA signal is received. So, whenever the device receives the INTA signal from the processor it should turn off the interrupt line. So, that should be the practice to be followed.

(Refer Slide Time: 20:32)

The slide is titled "Issues in Implementing INTR Interrupts". It contains a bullet point: "Can the microprocessor be interrupted again before the completion of the ISR?". Below this are two sub-points: "As soon as the 1st interrupt arrives, all maskable interrupts are disabled." and "They will only be enabled after the execution of the EI instruction." At the bottom of the slide, it states: "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." The slide footer includes the logos of IIT Kharagpur and NPTEL Online Certification Courses.

Now, coming to the other issues regarding a INTR interrupts can the microprocessor be interrupted again before completion of the ISR. So, that is once the interrupt has occurred. So, the processor is executing the ISR can it be interrupted again before finishing the interrupt.

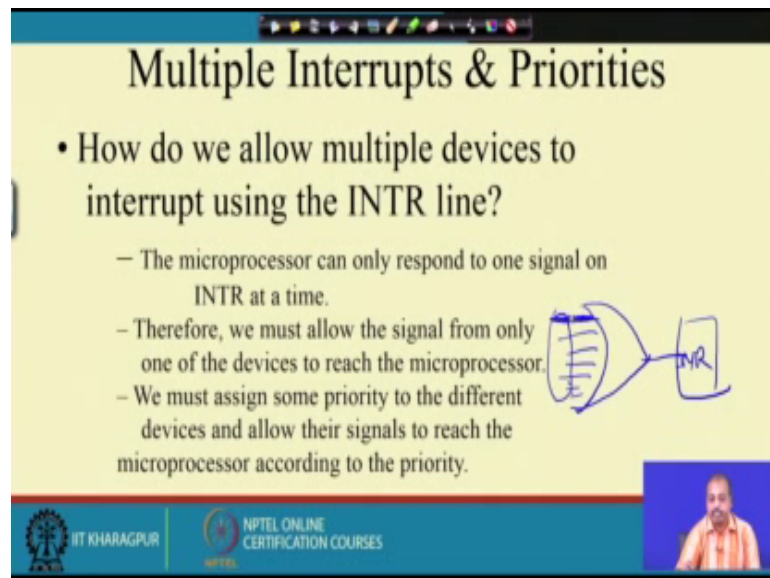
Now, the answer is yes and no simultaneously and it depends on the user because you can see that you can once the interrupt has occurred the INTR will be disabled and it will be enabled only after the EI instruction has been put. So, whenever the user feels that now I am ready to accept the next interrupt should put the EI instruction. So, if the user thinks that no more no interrupts should be coming. Now, once I am in the ISR then EI should be at the end of the ISR and if the user thinks that after at the for the very

beginning I am going to accept next interrupt then that EI should be at the beginning of the instead of the ISR.

So, this way, as soon as the first interrupt arrives all maskable interrupts are disabled. So, this is done by the processor and they are enabled by executing the EI instruction. So, now, where do you put the EI instruction is the user's job. So, if the EI instruction is placed early in the ISR other interact may occur before the ISR is done. So, that is the thing.

So, the point at which you are ready to receive the next interrupt. So, you can put the EI there.

(Refer Slide Time: 22:00)



The slide is titled "Multiple Interrupts & Priorities". It contains a bulleted list of three points:

- 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.

To the right of the text is a hand-drawn diagram of a priority encoder. It shows a multi-bit input bus on the left, a triangular symbol representing the encoder, and a single output line labeled "INTR" on the right.

At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. A small video inset in the bottom right corner shows a man speaking.

Next we answer the question that multiple interrupts and priorities like how do I have priorities like. So, I have so far whatever I have talked about. So, I have got a single interrupt line and a single device connected to the processor. Now, how can I ensure that a number of devices may be connected to the processor using the INTR line alone and there will be priorities among the devices.

For example there may be say 8 devices connected with the microprocessor through the INT line and there will be priority. So, the devices device 1 may have the highest priority followed by device 2, device 3 like that. So, device lower priority into lower priority

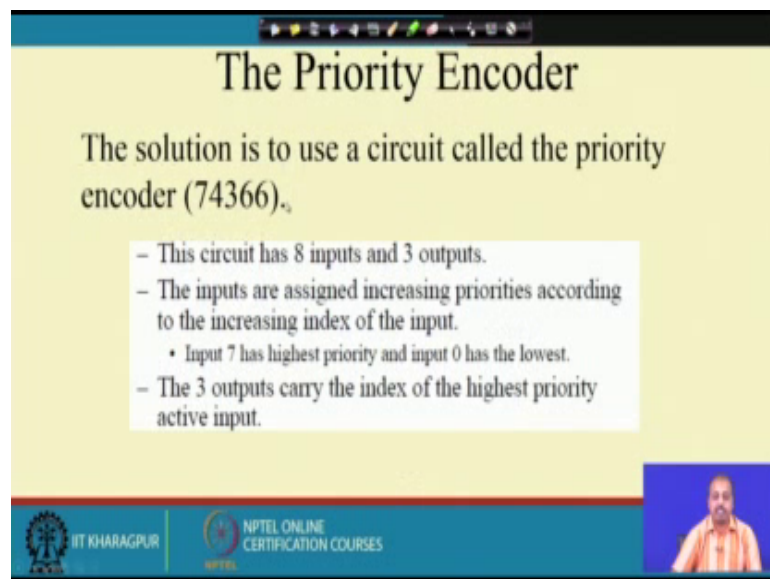
devices they will be allowed to interrupt the processor only if the higher priority devices they have not interrupted the processor.

So, typical design is like this that I have got all these interrupt lines coming from different processors put them into this or gate sort of thing and then give it to the microprocessors INTR p, but if we do this then the problem is that all these lines they are of equal priority. So, any of the interrupts coming, it will be interrupting the processor so that we may not want.

So, we may want that when these interrupt these device has interrupted. So, none of the remaining one should be allowed to interrupt. Similarly the second one will be allowed to interrupt only if the first one has not generated any interrupt. So, this is how this can be done so that we will look now.

So, the micro processor can only respond to one signal on the INTR line. So, we must allow the signal from only one device to reach the micro processor. So, there must be some priority assignment for that purpose. So, how to do this?

(Refer Slide Time: 23:54)



The Priority Encoder

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.

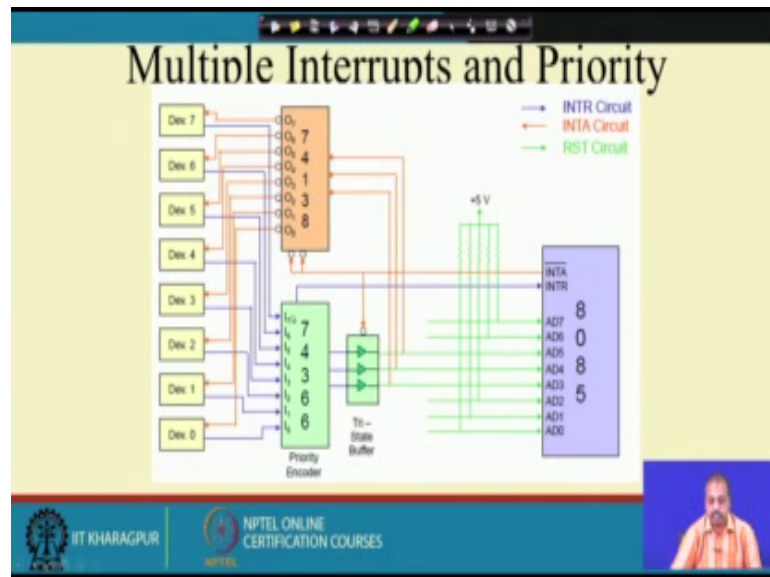
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for that purpose there is a special chip which is known as priority encoder or chip number is 74366 this particular circuit it has got 8 inputs and 3 outputs.

The inputs are there this is actually a priority encoder. So, a number of devices may be connected to this chip. So, the 8 devices can be connected to this chip and the pin to

which the device is connected will decide its priority. So, input 7 has the highest priority input 0 has the lowest priority and these 3 outputs they will carry the index of the highest priority active input. So, I think there is a diagram.

(Refer Slide Time: 24:38)



So, in this diagram, you see that 7 4 1 3 8. So, this is the priority encoder that we have. So, this device, this is actually going to this is the priority encoder, so this interrupts lines are coming from there. So, it is this I 0 to I 7. So, these are the inter planes connected here.

Now, this one this there, there is we can put these 3 output lines. So, 8 input 3 output and these 3 output lines. So, they can be put through some tri-state buffer and now, you see that if say this encoder will ensure that if I 7 has occurred. So, here you will get the pattern 1 1 1, if I 6 has occurred and I 7 has not occurred then only you will get the pattern 1 1 0 here. So, that way whichever highest priority interrupt has occurred, so that value that index will be available in these 3 bits.

So, this 0 will be available here only if none of the interrupts from device one to device 7 has occurred. So, I only device 0 has interrupted. So, in that case you will get the pattern 0 0 0 here. So, this priority encoder it has got an interrupt line so that is connected to the interrupt line of the processor and in the interrupt acknowledge cycle. So, you see that this interrupt technology cycle. So, this will enable this tri-state buffer so that these 3 beats will be available here and they are the, this for the RST instruction you know that

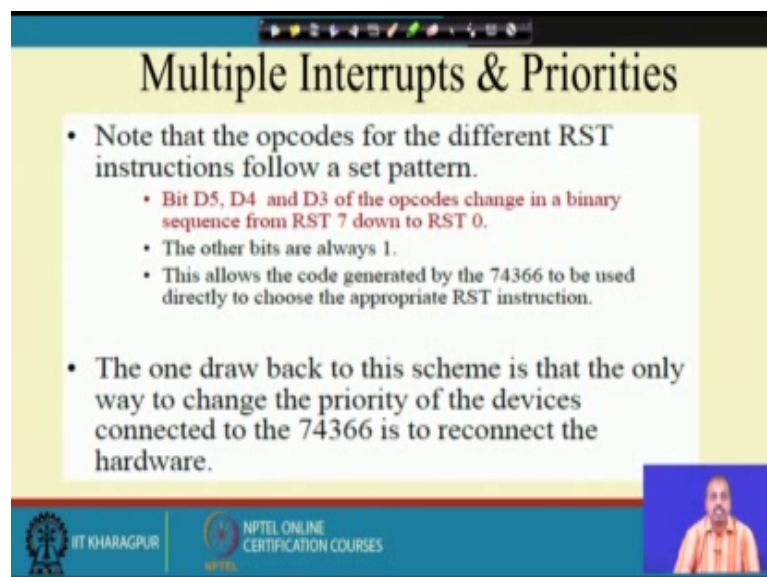
these other bits. So, they are all one is bit accepting this bit number 3 4 and 5 which can vary depending upon that RST n that co value of n. So, rest of the bits are all 1. So, that is done by this circuit tying all other bits to 1.

So, in the when the interrupt acknowledge meet in the acknowledge cycle is entered (Refer Time: 26:34) and the processor enable this INTA bar line. So, this tri-state buffer gets activated and all these in the RST n that n value will be available here. As a result the processor we will see one RST n instruction into this buffer and it will go to, it will go to the corresponding interrupt service routine.

Now, to send that how interrupt acknowledge line to the individual devices, what is done? One decoder is in employed here 7 4 1 3 8 decoder. So, here these 3 bits are connected these 3 bits from this tri-state buffer, they are connected here and whichever device has been allowed whichever device has been allowed to interrupt the system the highest priority device. So, that number is fed here. Accordingly the corresponding output will be enabled and that will go to the interrupt acknowledge for the device.

For example if say device 6 has interrupted. So, here you will get the pattern 1 1 0 and that 1 1 0 being fed here will enable this O 6 line and O 6 line will send the INTA signal to the device 6 others will not. Even if say 5 out of 6 5 4 3 all of them have interrupted only device 6 will get the interrupt acknowledge not the others. So, this way we can handle this multiple interrupts and their priorities.


(Refer Slide Time: 28:00)



Multiple Interrupts & Priorities

- 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.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Opcodes from different RST instructions they follow a pattern bit D 5 D 4 and D 3 they are they are having the opcode so that these bits will change and then that will give the code for RST and 1 1 1 is RST 7 and 0 0 0 is RST 0 other bits are always 1. So, this way we can use this decode this priority encoder for generating the RST code.

Now, the problem that we have is that the only way to change the priority of the devices is to change the order of connecting the hardware. So, you cannot change that dynamically. So, like if you want to change the priority between device 6 and device 7 you want to make device 6 higher priority and device 7 lower priority, but in that case I have to connect device 6 to I 7 and device 7 to I 6. So, physically I have to change the order, but still for the small system. So, this is fine. So, up to 8 devices, we can have this priority encoding. So, if you have got more number of devices then possibly we can have we can think about a multi level priority and coded scheme by which we can connect more than 1 7 4 3 6 6 chips and resolve interrupt priorities that way.

(Refer Slide Time: 29:30)

The 8085 Maskable/Vectored Interrupts

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).

Logo of IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

So, this on the other hand this maskable or vectored interrupt, there are 4 masked vectored interrupts in 8085 they are known as 5.5, 6.5, 7.5, they are RST 5.5, 6.5, 7.5.

And there is another trap instruction which is a trap interrupt which is know we will see later. So, all of them are maskable. So, first of all this maskable interrupts are 3, 5.5 6.5 and 7.5 they are maskable interrupt. The other interrupt that I have mentioned trap is a

non maskable interrupt and they are automatically vectored. So, RST 5.5 the vector address is 002CH, 6.5 it is 0034 and 7.5 is 003C.

So, when this interrupt occurs then the processor will jump to the location 002C then when 6.5 occurs it will go to 0034. So, you see that address is that the 002C is between RST 5 and RST 6, it is at the halfway between the two addresses for corresponding to RST 5 and 6 that why it is called RST 5.5. Similarly RST 6.5 is midway between RST 6 and RST 7 those two addresses so that is why it is 6.5 and 7.5 is after 7, 4 bytes away from 7 so that way it is 7.5. So, that is why it is the half way between the two, so it has got the name like that.