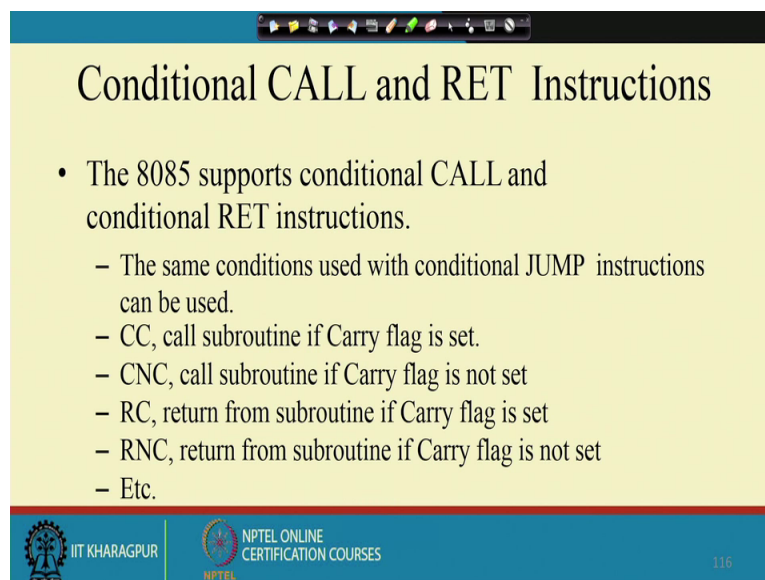


Digital Circuits
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 57
8085 Microprocessor (Contd.)



So, conditional call so far whatever call and return we have seen; so that where unconditional. So, the call instruction will definitely take the control to the subroutine and return instruction will definitely take you back from the subroutine.

(Refer Slide Time: 00:13)



Conditional CALL and RET Instructions

- The 8085 supports conditional CALL and conditional RET instructions.
 - The same conditions used with conditional JUMP instructions can be used.
 - CC, call subroutine if Carry flag is set.
 - CNC, call subroutine if Carry flag is not set
 - RC, return from subroutine if Carry flag is set
 - RNC, return from subroutine if Carry flag is not set
 - Etc.

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES 116

Sometimes we need some conditional call and return instructions like the CC instruction. So, call subroutine if carry flag is set similarly, CNC is call subroutine if carry flag is not set so like that we can have some conditional calls. Similarly, RC is return from subroutine if carry flag is set and RNC return from subroutine if carry flag is not set then we have got this 0 flag so CZ call if 0 flag is set, so like that we can have. So, all these condition codes that you are that are allowed with this conditional branch instructions so, they are all allowed with this conditional, call instructions also.

But they are much less practiced compared to these branch instructions, because that makes the code complex and difficult to follow while reading. So, that is. So, normally what is done is that we do some jump some conditional branch checking, and then put

the call instruction on top of that ok. We try to make it as straight line as possible ok. So, that way it has to be done.

(Refer Slide Time: 01:32)

A Proper Subroutine

- According to Software Engineering practices, a proper subroutine:
 - Is only entered with a CALL and exited with an RET
 - Has a single entry point
 - Do not use a CALL statement to jump into different points of the same subroutine.
 - Has a single exit point
 - There should be one return statement from any subroutine.
- Following these rules, there should not be any confusion with PUSH and POP usage.

The diagram shows a vertical rectangle representing a subroutine's memory space. The top is labeled '2000' and the bottom '3000'. A curved arrow starts from the top right and points back to the top left, indicating a return. Below the rectangle, two call instructions are listed: 'CALL 2000' and 'CALL 2500'. The 'CALL 2500' instruction is positioned below the 'CALL 2000' instruction, illustrating a jump to a non-entry point within the subroutine.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how to write a subroutine? Like what should be the, what are the general guidelines for writing a subroutine. So, proper subroutine is only entered with a call and exited with an ret instruction. So, that is it has got a single entry point, so do not use a call statement to jump into different points of the same subroutine should has it is ok.

So, what we mean is the guideline is like this see assembly language programs. So, since that control is entirely in the hand of the programmer. So, if this is my subroutine and supposes my subroutine starts at location say 2000, it starts at location 2000. Now we should issue a instructions like call 2000 so that is fine some. So, this subroutine supposes it spans from 2000 to 3000. Now it is highly possible that I put a call instruction for location 2500.

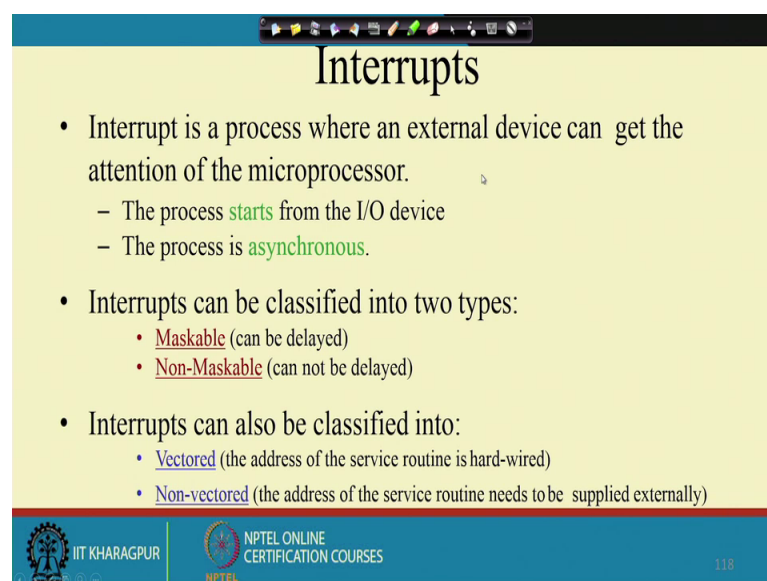
That is possible because 2500 is somewhere in between so the call this particular call will start from this point onwards. So, processor does not differentiate between whether this is a subroutine or not, but the processor will not differentiate between whether this is a subroutine or not; so it will always take you from this address onwards. So, 25 so if you whatever address you mention so it will take you to that address and it will go from there, but this so it is, but as you can understand that if you have got several such entry

points inside the subroutine then, the branching becomes this understanding the program becomes difficult.

So, this is exactly what is done here the done by the programmer. So, programmer has to be careful, and has to take into consideration that I should not branch into there the location inside a program. So, do not use a call statement to jump into different points of the same subroutine. So, that you should not do and you should similarly you should have a single exit point. So, somewhere here in the program I put an statement in the subroutine I put a statement like jump to say 1500; that takes it outside the subroutine. So, we should not do that because then we have got some exit so which is not really very valid one. So, there should be only one return statement from any subroutine.

So, these are the guides from the software engineering angles so as per as the processor instruction execution is concerned so these are the not at all any restriction. So, the programmer is absolutely free to do all these things, and assembly language programs also they have got the highest level of freedom, so they can write programs in in any in any style that they feel like ok. So, just similarly this if push and pop we have seen that there are some rules like push and pop should be done in the reverse order and all. So, those rules if they are followed properly, so that will avoid any confusion that can arise due to these push pop instructions so that that is there.

(Refer Slide Time: 04:47)



Interrupts

- Interrupt is a process where an external device can get the attention of the microprocessor.
 - The process **starts** from the I/O device
 - The process is **asynchronous**.
- Interrupts can be classified into two types:
 - **Maskable** (can be delayed)
 - **Non-Maskable** (can not be delayed)
- Interrupts can also be classified into:
 - **Vectored** (the address of the service routine is hard-wired)
 - **Non-vectored** (the address of the service routine needs to be supplied externally)

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL | 118

Next will be looking into another very important concept which is known as Interrupt; interrupt is a process where an external device can get the attention of the microprocessor. So, in our day to day life also interrupt is very common, like say suppose we are sitting in a room and reading a book and then all of a sudden the door bell rings. So, we have to go and see like what is attention give attention to why the door bell has ring somebody has come or not etcetera.

Now, the first thing that we should do is that; we should keep a note like a like the which page of the book I was reading. So, that I can come back and start reading from that. So that is the first point; then even before that maybe we were reading one line we were in the middle of a line. So, we first finish of that line and then a note down the page number maybe by means of a bookmark or so, we keep a note now the page that we are reading. And then go to attend the door call and then after doing whatever, after talking to the person who has arrived and then maybe we again come back and start reading the book.

So, that whole thing is an interrupt process. So, the similarly the processor when it is in it is normal execution what it is doing? So, it is just getting the next instruction from memory and executing it. After that it is again getting the next instruction and executing it. So, there is no way by which I can tell the processor see something emergency has happened. So, you need to look into this part and do the do the operation. So, so that until and unless the processor is asking for some input output type of operation. So, we are not getting any chance to tell the processor that; something extraordinary has occurred and you have to take care of this you have to do something for this.

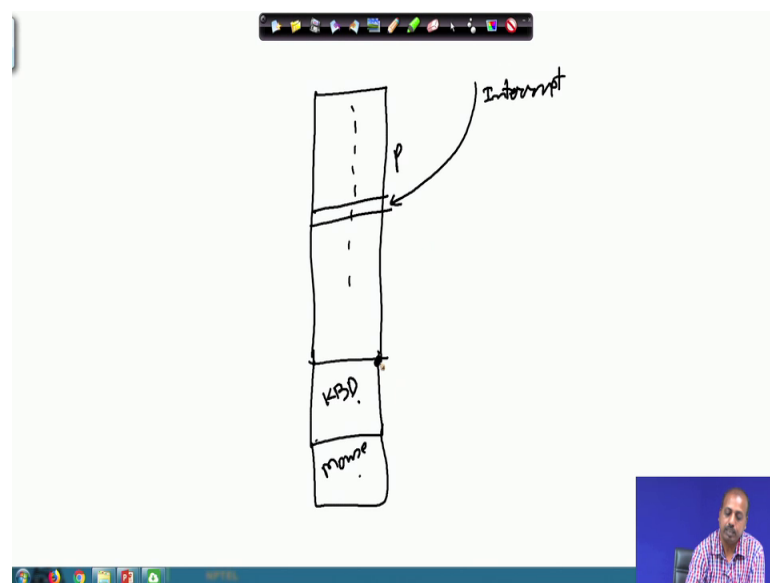
So, this is the process of interrupt. So, this is a interrupt is a process where an external device can get the attention of the microprocessor. The process starts from the IO device and the process is asynchronous. So, it starts from the IO device means the IO device will tell the microprocessor that; there is something emergency that has occurred and it has to it has to process that operation. And if it is asynchronous because when this interrupt will come; so there is no such time there is no such fixed timing for that. So, so processor works in a synchronous fashion so it works at the, with the clocks and or the as some clock frequency and all, but this interrupt can come at any time.

Some of this interrupts are maskable interrupts because you may we can make the processor, you can say that it will wait for some time and some of them and non

maskable interrupt so they cannot be delay. In terms of our day to day life, so suppose this micro oven whistle goes; that means, the say the food is ready. So, you know that even if you keep the food there for some time, nothing is going to happen ok. So, you can it is it is a maskable interrupt so we can do it later it can be delayed. And somebody the making a door pressing the doorbell maybe we need to attend it immediately. So, that has to be done immediately so that is non maskable.

Then interrupts can also be classified into vectored interrupt and non-vectored interrupts. So, vectored interrupt means the address of the service routine is hard wired. So, the processor knows the address of the subroutine to be executed, when this interrupt occurs. And the non-vectored interrupts so address of the subroutines need to be supplied externally. So, what we mean by this vectoring maybe we can explain further so it is like this.

(Refer Slide Time: 08:39)



See so also this is the, if this is suppose this is the program that the processor was executing. So, this is the program P that the processor was executing when the interrupt occurred. So, when the processor was at this particular line, the interrupt has occurred so then interrupt has come.

Now, what the processor will do? So, processor has to process this interrupt, and processing this interrupt means again executing some routine. So, what is done; depending upon this type of interrupt maybe this interrupt is a to to that the user has

pressed a key and that has generated an interrupt or user has moved the mouse and that has generated interrupt. So, what the processor is supposed to do; is to read the character that has pressed or read the current mouse location coordinate of the current mouse location for some purpose. So, that so the for those operations that is, reading the key where the content of the key or reading the mouse position etcetera.

So, there is there are routine. So, this may be the keyboard routine, which will be which will be stored which will be reading the keyboard. This may be the mouse movement detection routine.

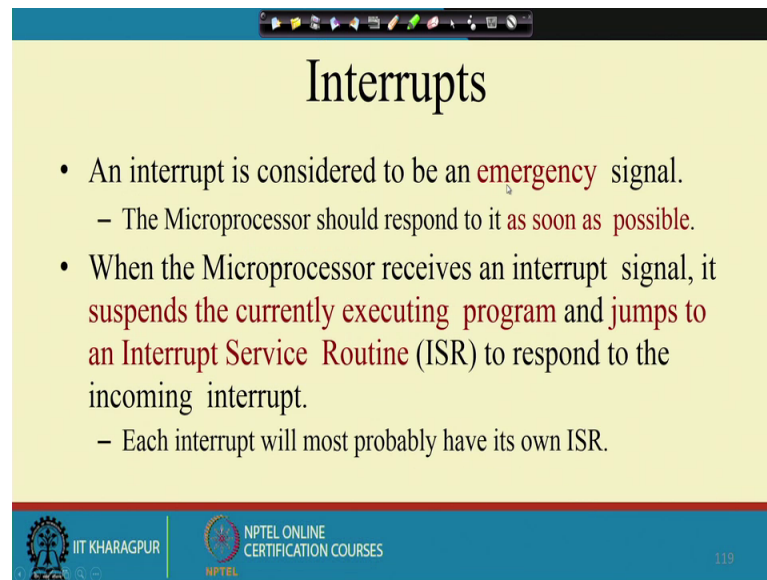
So, if this interrupt is a keyboard interrupt so this these routines are already loaded into the memory. So, one possibility is that the processor already knows that for the keyboard interrupt so, what is the address; from which the, from where the program be executed. So, when this keyboard interrupt comes, the processor immediately branches to this address so that is one possibility.

And the other possibility is that the processor does not know where to go. So, in that case in that case the device that has generated the interrupt should tell the processor that; what is the address of the service routine the corresponding service routine, maybe for the mouse it does not the processor does not know like where to go, if the mouse interrupt occurs

In that case the processor should the, the this address of this mouse routine should be provided by the mouse device itself. So, it should tell the processor this is the address. So, once the mouse device tells the processor that this is the address, then the processor will go to that particular address. So, this is the difference between the vectored interrupt and non-vectored interrupt. So, in case of vectored interrupt, the processor knows the service address service routine address where to go, and in case of non-vectored interrupt so processor does not know the service routine where to go. So, in that in the second case the processor has to get the address from the device.



So, this is the difference between this vectored and non-vectored interrupt. So, in case of vectored interrupt the address of service routine is hard wired, so it is it is fixed by the designers, and in case of non-vectored interrupt; so this needs to be supplied externally by the device.

(Refer Slide Time: 11:31)



Interrupts

- An interrupt is considered to be an **emergency** signal.
 - The Microprocessor should respond to it **as soon as possible**.
- When the Microprocessor receives an interrupt signal, it **suspends the currently executing program and jumps to an Interrupt Service Routine (ISR)** to respond to the incoming interrupt.
 - Each interrupt will most probably have its own ISR.

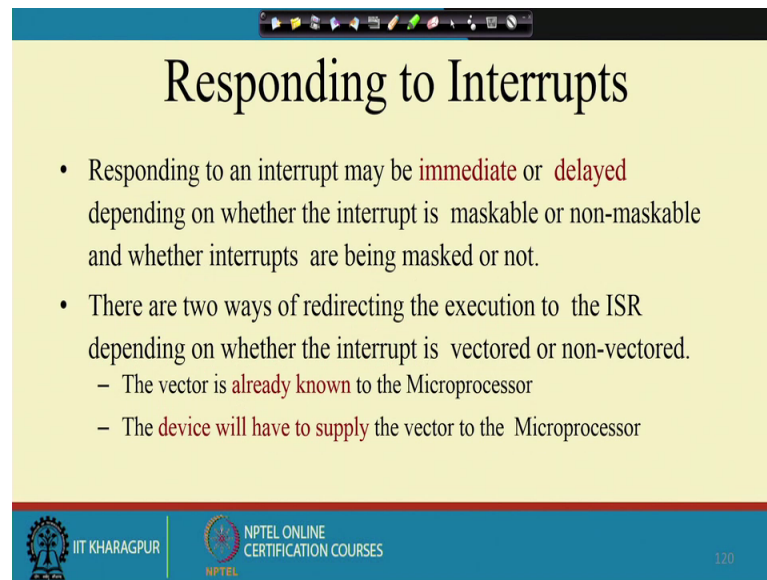
 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES

119

So, in terms of an interrupt, so it can be considered to be an emergency signal and microprocessor should respond to it as soon as possible, so as soon as possible it should be responded to. When the microprocessor receives the interrupt signal it suspends the currently executing program jumps to the interrupt service routine to or it is interrupt service routine so this is also known as ISR in short, to respond to the incoming interrupt and each interrupt most will most probably have it is own ISR.

So, some of in some cases some of the interrupts we grouped together to have a common ISR, but in general each interrupt should is of different group, as a result it should have different types, so it should have a different service routine.

(Refer Slide Time: 12:21)



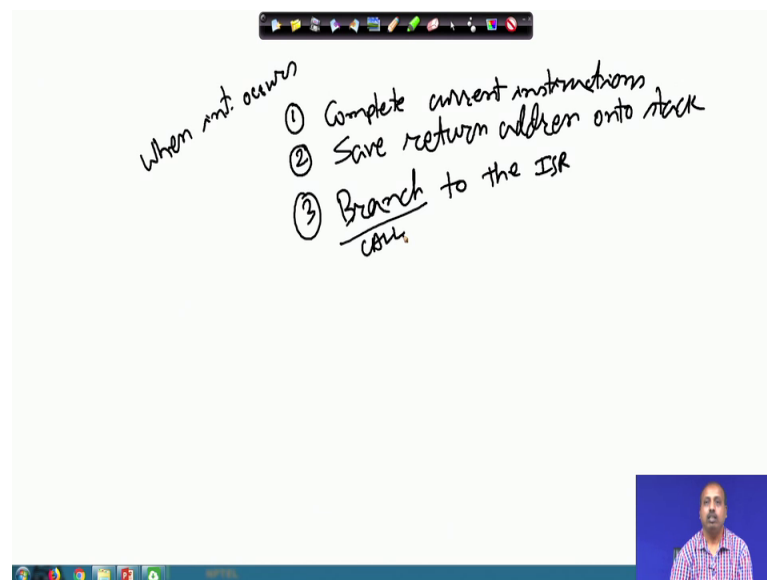
Responding to Interrupts

- Responding to an interrupt may be **immediate** or **delayed** depending on whether the interrupt is maskable or non-maskable and whether interrupts are being masked or not.
- There are two ways of redirecting the execution to the ISR depending on whether the interrupt is vectored or non-vectored.
 - The vector is **already known** to the Microprocessor
 - The **device will have to supply** the vector to the Microprocessor

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 120

So, if you try to look in to the spaces, the steps that it should take for responding to an interrupt. So, the steps that are needed for responding to an interrupt.


(Refer Slide Time: 12:32)



When int. occurs

- ① Complete current instructions
- ② Save return address onto stack
- ③ Branch to the ISR

ANI



The first thing that the processor does is that; so when interrupt occurs the first thing that it should do is to complete the current instruction.

As you know that every instruction may be spanning over number of clock cycles depending upon the type of instructions so it may span over a number of clock cycles. So, it should complete the current instruction. After completing the current instruction,

the next thing that it should do is to save the return address, save the return address onto step.

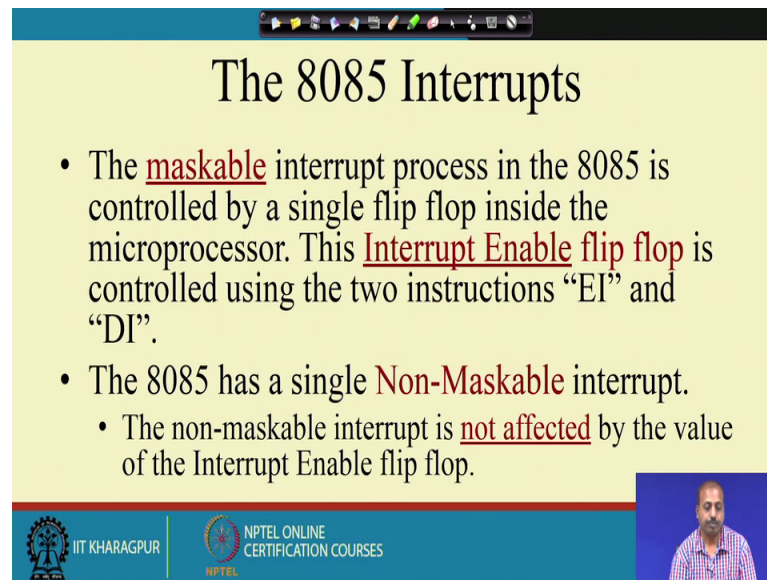
And then it will branch to the to the interrupt service routine. Now for in case of vectored interrupt the interrupt service routine address is known so that case; branch to external this interrupt service routine is easy or this it is basically a call basically a call instruction. So, this program counter values will be saved onto the stack and it will be going to the interrupt service routine, but in case of non-vectored interrupt, so this ISR address is not known.

So, in that case it will go into some other it will be the it will be doing something extra to get the interrupt service address and from where it will execute after getting that address it will go to the branch to that ISR. Now so, this is the 3 things that the processor does before going to the ISR. Now if we look into; how do we respond to an interrupt. So, this maybe this maybe immediate or delayed depending upon whether the interrupt is maskable or non maskable, and whether interrupts are being masked or not.

Some of the interrupts are maskable interrupts some of them are non maskable interrupt like. In case of processors, we can say the power failure is a non maskable interrupt so, so that if there is a powerful then that is very severe thing because entire system is going to come down. So, that way that is a non maskable interrupt. And some other interrupts may be maskable so that is the user may think that I should not I do this program should not be disturbed by all these interrupts ok. So, that way you can mask out the interrupts so that; goes interrupts will not come will not disturb the program being executed.

So, this interrupt maybe maskable interrupt maybe non maskable. So, if it is a maskable interrupt then user has a choice to mask out the interrupt. So, if the interrupts are masked then the they will not be responded immediately till the user again unmask the interrupts. So, there are 2 ways of redirecting the execution are to the ISR, depending on whether the interrupt is vectored or non-vectored. So, vectored interrupt means the, the microprocessor already knows the ISR address ok, as I already said. And in case of non-vectored interrupt the device will have to supply the vector to the microprocessors so that is the non-vectored interrupt.

(Refer Slide Time: 15:59)



The slide is titled "The 8085 Interrupts" and contains the following text:

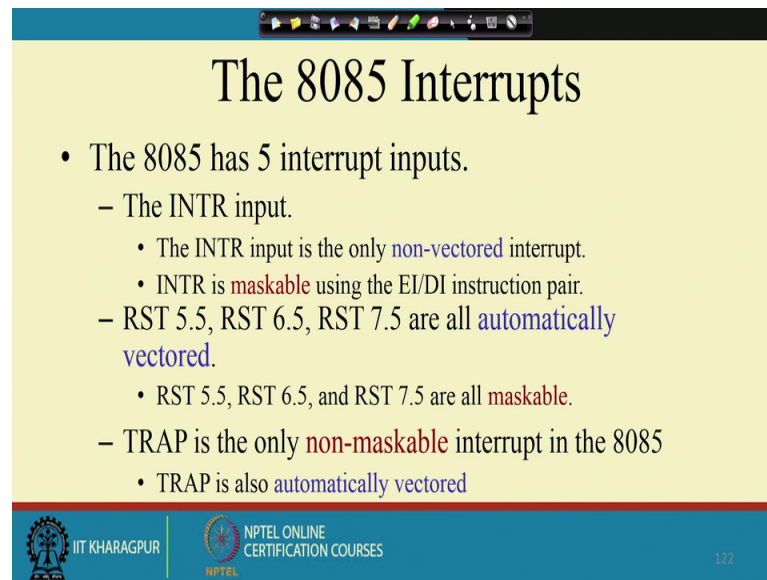
- The maskable interrupt process in the 8085 is controlled by a single flip flop inside the microprocessor. This Interrupt Enable flip flop is controlled using the two instructions “EI” and “DI”.
- The 8085 has a single Non-Maskable interrupt.
 - The non-maskable interrupt is not affected by the value of the Interrupt Enable flip flop.

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

In case of 8085 processor, the maskable interrupt process in 8085 is controlled by a single flip flop inside the microprocessor, which is known as the interrupt enable flip flop. So, this is controlled by 2 instructions, enable interrupt and disable interrupt EI and DI. So, it has got 8085 as a single non maskable interrupt and this is the so this non maskable interrupt, so this these interrupt cannot be affected by this EI and DI instructions. So, the they are always enabled so whatever program is being executed, if you if it if it finds this interrupt has occurred non maskable interrupt has occurred, that will always be serviced by the microprocessor.

So, this is the, this it has got this maskable and non maskable interrupts by and, this maskable interrupts can be masked by DI, instruction they can be unmasked by EI instruction.

(Refer Slide Time: 16:59)



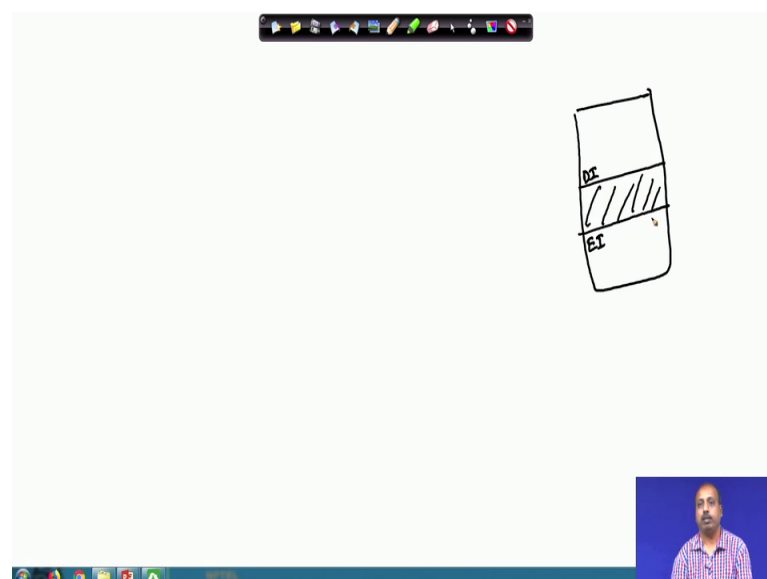
The 8085 Interrupts

- The 8085 has 5 interrupt inputs.
 - The INTR input.
 - The INTR input is the only **non-vector**ed interrupt.
 - INTR is **maskable** using the EI/DI instruction pair.
 - RST 5.5, RST 6.5, RST 7.5 are all **automatically vectored**.
 - RST 5.5, RST 6.5, and RST 7.5 are all **maskable**.
 - TRAP is the only **non-maskable** interrupt in the 8085
 - TRAP is also **automatically vectored**


IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 122

So, 8085 has got 5 interrupt inputs one of the first one is the INTR. So, INTR is the, it is a non-vectorized interrupt. In case of 80850 so there is only one non vectored interrupt, which is INTR. And the INTR is maskable using the EI and DI instruction pair. So, it can give this so if the processor thinks or the user thinks that I should not be destroyed by INTR for executing some part of my code. So, it can put a DI instruction at the beginning of that code and then EI at the bottom of the code.

(Refer Slide Time: 17:44)



A hand-drawn diagram on a whiteboard showing a vertical rectangular box. The top portion of the box is labeled 'DI' and the bottom portion is labeled 'EI'. The middle section of the box is filled with diagonal hatching lines, representing a code segment. The diagram is drawn in black ink on a white background.

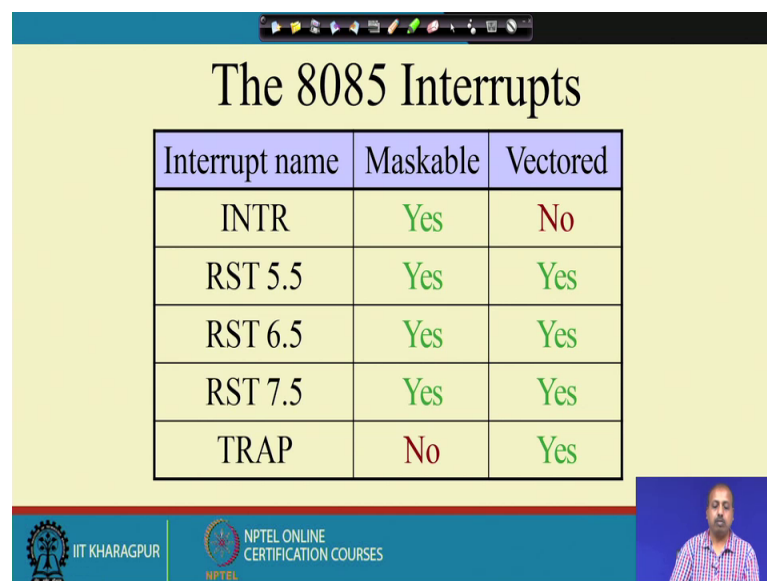


So, it is like this; suppose this is a piece of program that I want to execute, and out of this code so in this part so I do not want to get disturbed by the maskable interrupts. So, what you do; you put a DI instruction here and then EI instruction here. So, therefore, during this part of execution only the non maskable interrupts, so they can the interrupt the program; maskable interrupts so they will not be able to mask the interrupt the system.

So, in case of this INTR, so this is a non-vectored interrupt. So, this read the ISR address is not known to the processor directly and then this is also a maskable because it is the by means of this EI DI pair. There are some more instructions RST 5.5 RST 6.5 and RST 7.5 they are automatically vectored. So, they are vectored means their ISR addresses are known. So, the it will be the processor will jump to a particular address.

And all these RSTs they are maskable so all of them are maskable. And they they can the trap is the only non maskable interrupt in 8085 and trap is automatically also automatically vectored or trap is basically similar to RST 4.5.

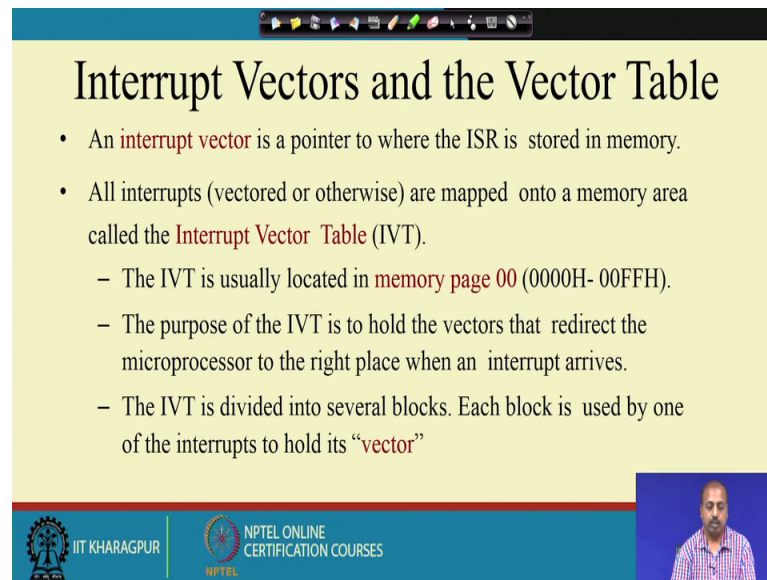
(Refer Slide Time: 19:07)



Interrupt name	Maskable	Vectored
INTR	Yes	No
RST 5.5	Yes	Yes
RST 6.5	Yes	Yes
RST 7.5	Yes	Yes
TRAP	No	Yes

So, to summarize so INTR RST 5.5 6.5 and 7.5 they are maskable interrupts and trap is a non maskable interrupt. And this then this INTR as per as vectored non vectored classification is concerned; INTR is a non-vectored interrupt and 5.5 RST 5.5 6.5 7.5 and trap so they are all vectored interrupts.

(Refer Slide Time: 19:34)



Interrupt Vectors and the Vector Table

- An **interrupt vector** is a pointer to where the ISR is stored in memory.
- All interrupts (vectored or otherwise) are mapped onto a memory area called the **Interrupt Vector Table (IVT)**.
 - The IVT is usually located in **memory page 00 (0000H- 00FFH)**.
 - The purpose of the IVT is to hold the vectors that redirect the microprocessor to the right place when an interrupt arrives.
 - The IVT is divided into several blocks. Each block is used by one of the interrupts to hold its “**vector**”

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, how this vectoring takes place ok? So, for the vectored interrupt so, there is something called the interrupt vector table or IVT. So, any processor that is supporting vectored interrupt should define some part of memory, which is called the interrupt vector table or IVT. And the specific locations in the IVT are dedicated to different vectored interrupts, and what the processor is expecting is that; in that location so you should put that the interrupt service routine should start from that location.

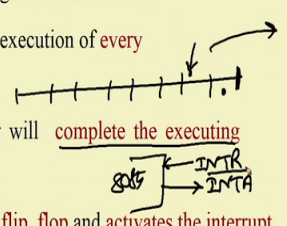
So, this interrupt an interrupt vector is a pointer to where, the ISR is stored in the memory. All interrupts vectored or otherwise are mapped onto a memory area called interrupt vector table. This these interrupt vector table is usually located in the memory range 0 0 that is 0 0 0 0 to FFF that is first 56 bytes

And the purpose of this interrupt vector table is to hold the vectors that; redirect the microprocessor to the right place when an interrupts arrives. And IVT is divided into several blocks east block is used by one of the interrupts to hold the vector, hold it is vector. So, if you have got a n number of interrupts then this range of this 256 bytes. So, it is divided into n such blocks and each block will have the, it will hold the vector for the interrupt, one vector for the interrupt.


(Refer Slide Time: 21:04)

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



So, in case of 8085 processor so, the for the non-vector interrupt process that mean; when it is not non vectored, so, how does it operates? So, this is the vectored process should be enabled using the EI instruction, the EI instruction must be executed sometime earlier by the processor ok, for the interrupts to be enabled. And the 8085 checks an interrupt during the execution of every instruction. So, what happens is that, what happens is that; so, this 8085 instruction, so that is so it is taking a number of cycles number of t states. So, if suppose this is these so these are the different t states that the processor the processor is taking for executing.

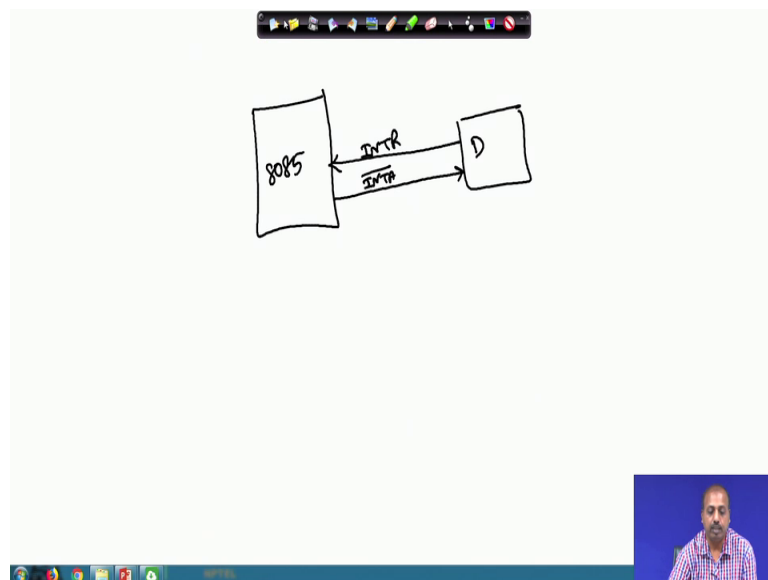
So, if this is the last t state so in these t state so it will check whether any interrupt has occurred or not. And then if the interrupt has occurred then after finishing the current instruction, so it will go to that interrupt service routine ok. So, this is the point so this last, but one clock cycle the last, but one clock cycle of the total instruction execution ok. So, in every instruction execution the last, but one clock cycle it will check the interrupts. And if it finds that some interrupt is has occurred and it needs to be serviced; then it will be branching to that ISR at routine after finishing the current instruction.

So, if there is an interrupt the microprocessor will complete the execute complete executing the instruction and start a restart sequence. So, restart sequence is the interrupt, somehow the interrupt services process. One thing you must note that it always completes the current instructions. So, it is not that it leave the first the original

instruction halfway done, so, it is not like that. It finishes the current instruction and then it attends to go to the interrupt service process. It starts the restart sequence of operations by which it will be doing the interrupt servicing.

What is this restart sequence? It first resets the interrupt flag that interrupt enable flag IE so this is disabled, so this becomes DI. So, that is, so this and then it will start the one cycle which is known as an interrupt acknowledge cycle. So, in case of 8085 if you remember that there were 2 pins, one is the 1 pin was the INTR which is an interrupt line and then there was another pin going out which is INTA bar interrupt acknowledge bar. So, this actually tells that this is actually used for to that for that you to tell the device that, the interrupt has been received by the 8085 processor.

(Refer Slide Time: 24:05)



So, if this is your 8085 processor and this is the device, now this device has given an interrupt to the 8085. Now the device must be told that that is that it is interrupt has been accepted ok, then the processor is going to do the associated operation somehow this has to be told ok. So, that is the role of this INTR and INTA bar lines. And as you know that since INTR the address is the ISR address is not known, then the device should somehow supply this interrupt address the interrupt service address to the 8085 processor. So, we will see how it can be done.

Now, in case of so, upon receiving this INTA signal, the interrupting device is expected to return the op code of one of the RST instructions, one of the 8 RST instructions.

(Refer Slide Time: 25:12)

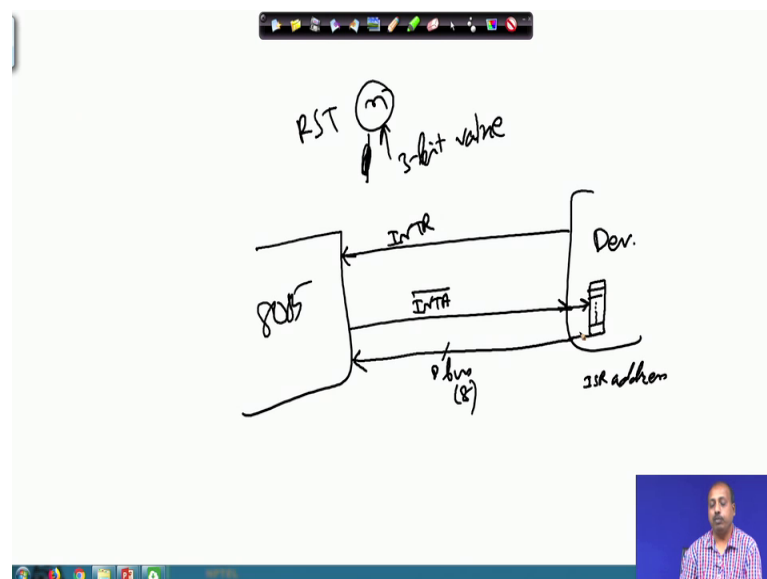
The 8085 Non-Vectored Interrupt Process

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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 126

So, so in general, this RST instructions are like this, so, this so this RST n.

(Refer Slide Time: 25:32)



So, this n can be a 3-bit value, this n can be a 3-bit value. And what the processor does is that; it multiplies this what the processor does is that it is like this, so this is the 8085 processor and this is the device.

Now, it has given the interrupt and it has put the interrupt acknowledge line. Then the processor the 8085 expects that the device will put one 8-bit code here. So, this is the data bus line, this is the data bus line of the processor and it expects that one 8-bit value

will be coming here, which will correspond to one of this RST n instructions ok. So, upon getting that; it may be that device has got some special register here one 8-bit register here, which holds the ISR address, which holds the ISR address. And this interrupt acknowledge signal when it reaches this register. So, it puts the content of those content of the register on to the data bus.

In this way, the device may have some device address register; and that address register is nothing but the address of the ISR that it is executing that, that that should be executed to get the interrupt service. So, this data bus on the data bus so it will expect that one of the RST instruction code will be put here and this 8085 processor so it will execute the RST instruction then. So, so that this RST instruction so that we actually transfers the control to some location this whatever value be, whatever be the value of this n. So, depending upon that it will jump to a predefined pre specified location and from that point onward the processor will start executing.

So, it is expected that this interrupt service routine for this device is located from that particular address. So, this way this vectored interrupt can be non vectored interrupt. So, this non vectored interrupt so this so it will be it will be doing it so, this will be yeah. So, this INTA receiving, the INTA signal this interrupting device is will put the output of one of the 8 RST instructions and upon getting this op code, this RST instruction. So, it will save the address of the next instruction in the stack and jump to the appropriate entry in the interrupt vectored table.

And this interrupt vector table entry it will it may redirect must redirect the microprocessor to the actual service routine, because the space may be small. So, in the interrupt vector table, so it may hold a branch instruction to a latter part in the memory where it has it actually loads the entire service routine. Then the service routine will be executed and the at the end of the service routine there should be one return instruction, that will return the execution to the original program at where the interrupt was received.