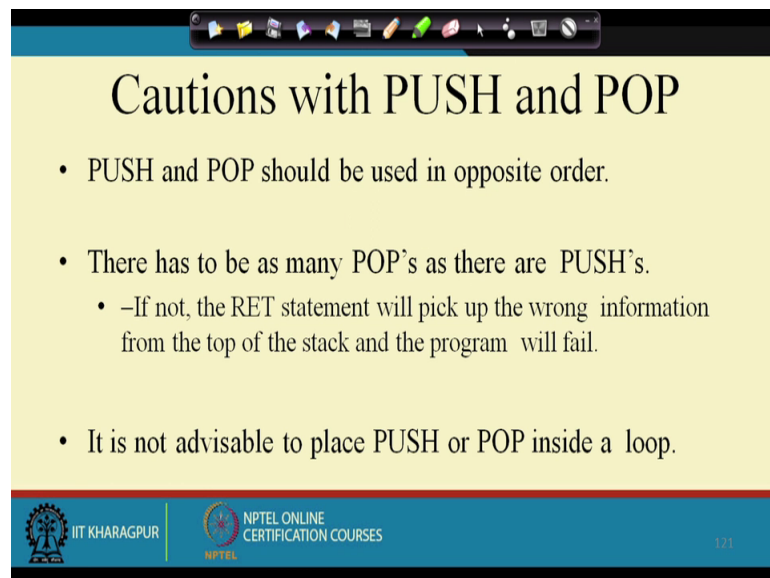


**Microprocessors and Microcontrollers**  
**Prof. Santanu Chattopadhyay**  
**Department of E & EC Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 16**  
**8085 Microprocessors (Contd.)**

Also there has to be as many pops as there are pushes.

(Refer Slide Time: 00:27)



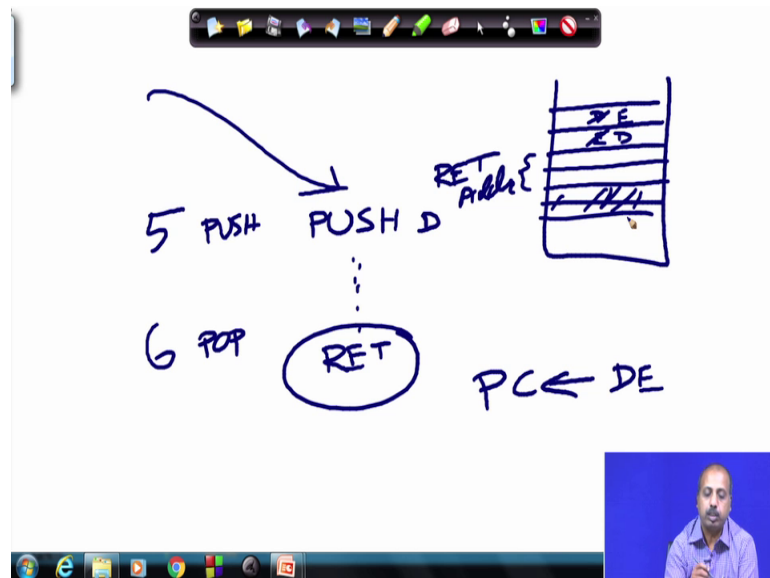
**Cautions with PUSH and POP**

- PUSH and POP should be used in opposite order.
- There has to be as many POP's as there are PUSH's.
  - –If not, the RET statement will pick up the wrong information from the top of the stack and the program will fail.
- It is not advisable to place PUSH or POP inside a loop.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Otherwise what will happen? So, there will be a mismatch and in that case the return address picked up by this return instruction will be in problem. So, let us try to see like how this thing can happen say.

(Refer Slide Time: 00:37)



So, as you know that when I am going to subroutine, so this stack top. So, if this is the stack top. So, it contains the return address. So, these two locations they will have the return address.

Now, after that what at the beginning, I have done say one push. So, these push say D. So, the d pair has been pushed here. So, this is your E register this is the d sorry this is the d register this is the E register that is there now at the end. So, if we forget to pop the D and we directly put a return instruction here, then in the execution of return instruction; what will happen? So, it will try to take out the content of the top to top most locations from stack and put them into the pc.

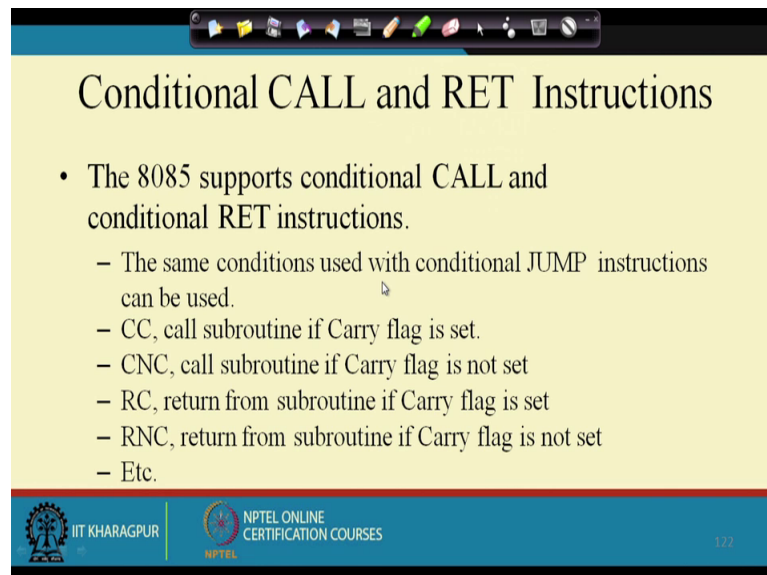
So, as a result when this return is executed, the PC register will get the original content of the de pair, and which is definitely not the return address. So, that way the program will return to some arbitrary location. So, that is why you said that, this is the other caution that this number of push and pop must match. If there is a mismatch then or it may so happen that I have done say 5 pushes at the beginning and 6 pops at the be at the end.

So, if I do 5 pushes and 6 pops, and then what will happen is that this return address has also been popped out from the stack. So, the next location whatever is there. So, that will be a some garbage in value and that value will be loaded into the PC, when we are trying

to return. So, this push and pop must match number of push and pops their order and these things should match.

So, we have to be careful in that sense. So, these are the things that push pop should be used in opposite order, and number of push and pop must match and it is not advisable to place this push and pop inside a loop, because it may so happen that a loop may exist via different ways, and it is not it may not be ensured that in all the exits same number of push and pops are encountered. So, that way the return at the stack content may become erroneous and this return address may be popped out wrongly.

(Refer Slide Time: 03:13)



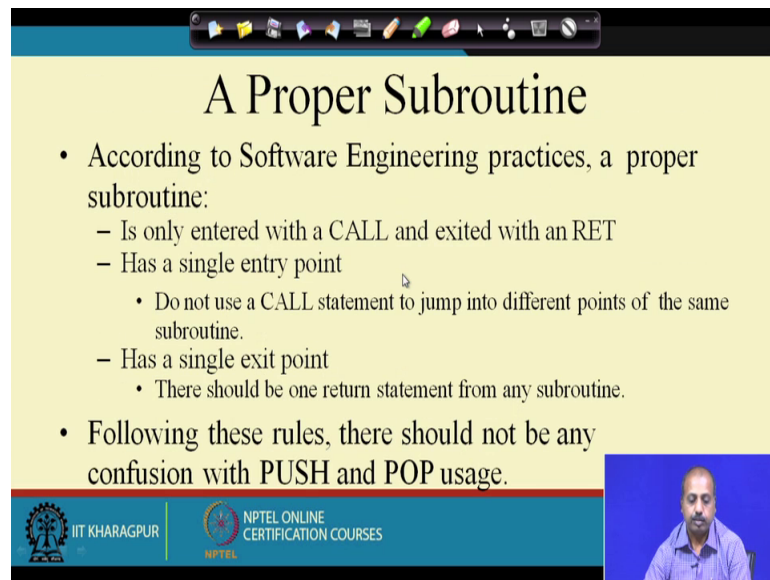
The slide is titled "Conditional CALL and RET Instructions". It contains a bulleted list of instructions supported by the 8085 microprocessor. The list includes: "The 8085 supports conditional CALL and conditional RET instructions." followed by sub-points: "The same conditions used with conditional JUMP 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", and "Etc.". The slide footer includes the IIT Kharagpur logo and the text "NPTEL ONLINE CERTIFICATION COURSES".

- 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.

So, there is another version of this call and return instruction, which are known as conditional call. So, like just like conditional jump. So, you have got conditional call; like we have got the cc instruction. So, this will call the subroutine if the carry flag is set. So, call subroutine if carry flag is set then CNC, it will call the subroutine if carry flag is not set. So, we have got similarly we can have say CZ, call if the zero flag is set CNZ call if the zero flag is not set, this way we can have many variants like then similarly for the return also we can have RC. So, return if carry flag is set, and RNC return if carry flag is not set.

So, this way we can have different variants whatever combinations we have called for jump instruction all those combinations can be there for the call instruction as well.

(Refer Slide Time: 04:08)



## A Proper Subroutine

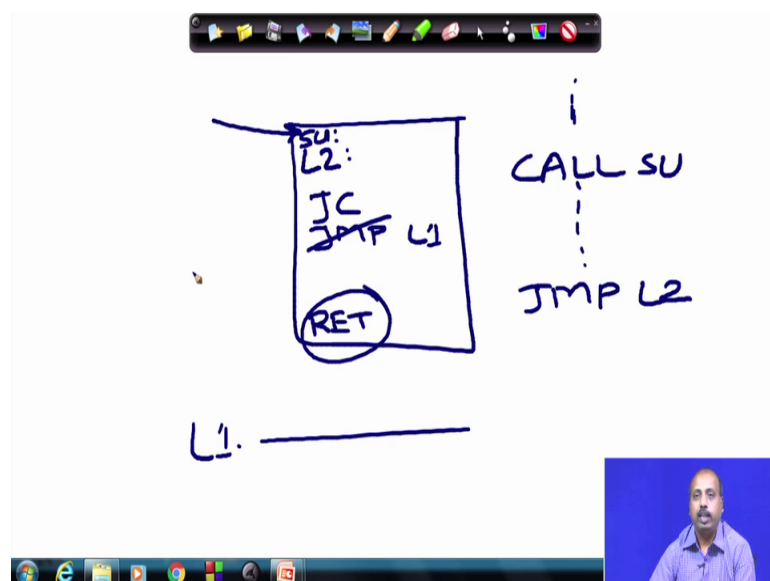
- According to Software Engineering practices, a proper subroutine:
  - Is only entered with a CALL and exited with an RET
  - Has a single entry point
    - Do not use a CALL statement to jump into different points of the same subroutine.
  - Has a single exit point
    - There should be one return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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

However as a general design practice, if you are looking it from a software engineering angle. So, if proper subroutine should have some important properties. The first one is that it is the is only entered with a call and exited with an with a return instruction. So, the subroutine should have a call instruction and a. So, it should be entered via a call and return via a ret. So, that is it has got a single entry point and it has got a single exit point.

So, if we do not follow this, then what will happen is that say I am writing a subroutine.

(Refer Slide Time: 04:52)



The diagram illustrates a subroutine structure. On the left, a box represents the subroutine body with the following labels: 'SU:' at the top left, 'L2:' below it, 'JC' and 'JMP L1' in the middle, and 'RET' circled at the bottom. To the right, a vertical dashed line indicates the flow: 'CALL SU' at the top, followed by 'JMP L2' at the bottom. Below the box, 'L1.' is written with a horizontal line extending to the right. The entire diagram is drawn in blue ink on a white background.

So, this is my subroutine and there is no harm like say instead of some from somewhere here I put a jump instruction, jump to address L 1, where L 1 is somewhere outside the subroutine L 1 is somewhere here. But actual the return is here, but in some conditional execution of the code it jumps from this L 1 maybe instead of a simple jump, it may be a jump on carry or something like that some conditional jump and it jumps out of the subroutine.

Now, if this thing is done then the measured difficulty that, you have is that now the stack content becomes inconsistent because the stack will still have the return value stored in the in its top. So, that is. So, or somehow we have to ensure that the stack pointer value is modified so that the return value gets discarded. Or if this is a level say L2 in my program in my sub program or subroutine then from outside. So, you can have something like. So, normal if this if the first entry has got the level say subroutine sub say suppose the name of thus it has got the name su, then what will happen is that you will get normal exit into entry into this subroutine is like call su.

So, it will be calling the subroutine su and their putting going there. But as such in this this is the main program. So, there is nobody is restricting you to write a statement like say jump L2. So, we will jump in inside the subroutine. So, instead of going via this proper entry point, you can jump inside the subroutine. And if you do that then the difficulty is that the return address is not saved in the stack. So, when the ret instruction will be executed in in future. So, ret will not be able to return to your proper address, until and unless you take enough care to put the return address into the stack separately ok.

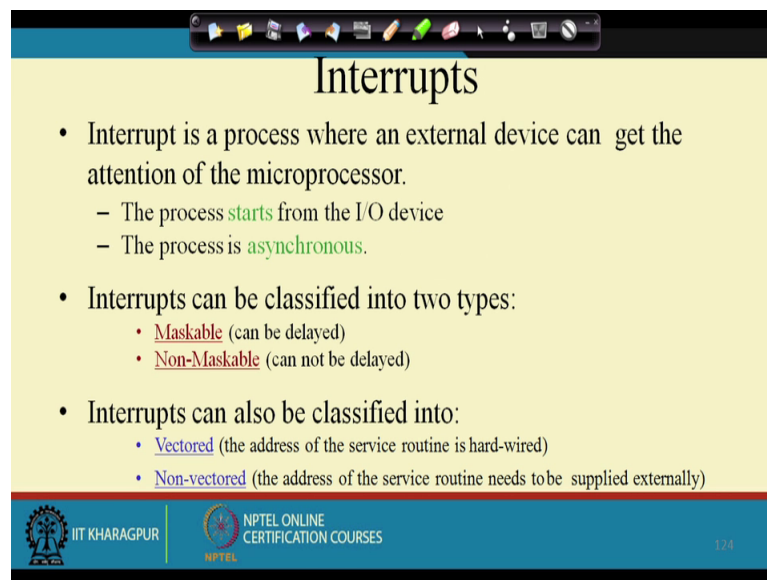
So, these are not a very good programming practice. So, ideally you should have a single entry point and single exit point in the subroutine, and the subroutine should be entered through that entry point only and it should be exited in that entry was through that entry point only. So, this is what is said here that according to software engineering practice, a subroutine is only entered with a call and exited with a with an RET, and has a single entry point. So, do not use a call statement to jump into different points of the same subroutine; so ok.

So, or you can have you can you should also have a single exit point, should not be there should be only one return statement from any subroutine. So, if you have got multiple

return statements. So, that makes the readability of the program difficult. So, we should try to redesign that subroutine so that there is only one exit point. So, that will help in making the program easily understandable.

So, following these rules, there should not be any confusion with push and pop usages also. So, push and pop. So, their number should match and all that and so, if we follow all these rules then we will be able to design good subroutine.

(Refer Slide Time: 08:25)



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

- 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)

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 124.

Next we will go into another very important concept in the processor design, which are known as interrupts. So, so far whatever we have discussed in this course, there we have got the processor, and processor it gets the next instruction from memory executes it.

So, there is no way by which we can tell the processor that something has happened in the outside world, and something emergency has to be done, something exceptional has to be done. Like say if there is if I am having a program that controls the operation of a plant, then it controls the conveyor belt etcetera, etcetera. So, all those parts are routine. So, they are going on at some regular intervals of time. So, my processor is doing a generating control signals in through some programs in that way.

But suppose there is a fire that is detected there is a smoke detector. So, smoke detector detects, if some fire or the some smoke or there is a possibility of fire. So, now, whatever the processor was doing, it was controlling the conveyor belt. So, it should be told

immediately that something some fire handling routine has to be executed, which will activate the fire tenders or if it will send a call to the local fire brigade office like that. So, that has to be done.

So, that is basically the interrupt. And in our day to day life also suppose I am reading a book sitting in my room. So, that is the normal process or some person comes and presses the bell door bell. So, that is an interrupt. So, I have to stop the reading the book I have to go open find out what to what that person wants and all that, do all those operations and then again after some time when that interaction with the person is over I will come back and read the book.

So, similarly, if in the processor also the interrupts will be like that. So, when such situation occurs that particular service has to be done or particular routine has to be executed by the processor and once that processing is over, the processor should resume whatever it was doing. So, interrupt is a process where an external device can get the attention of the micro processor. So, normally is. So, this interrupt has to be initiated by the IO device. So, processor will not initiate it, otherwise what like if I have connected a keyboard with a processor. Now normally this keyboard is operated by or by human beings, they are much much slower compared to these electronic components like processors.

Now, if the processor is operating like this that it continually scans whether any keybo key has been placed by the user and then takes some action. Then a major part of processors time will be spent just to do this polling like if any key has been pressed or finding out that, A better design is like this that whenever the user presses a key the processor will be informed that a key has been pressed, and then accordingly the processor will try to understand which key has been pressed and take some action on that basis.

So, this way these interrupts are actually originating from the IO device. So, IO device they will send an interrupt to the processor, and this whole process is asynchronous. So, asynchronous means it is not it is not synchronized with the cp the processor clock. So, it is not mandatory that this interrupt should occur at the clock edges and things like that. So, it is totally asynchronous, it can come at any point of time. So, whether it will be answered by the processor or not that depends on many other things as we see it slowly,

but the occurrence is asynchronous. There can be two types of interrupts some of the interrupts are called maskable and some of them are called non-maskable.

So, maskable means there is interrupts they can be delayed. So, you can I can say that for the time being, I do not want to get interrupted from this source for example, as I going back to the human being reading a book and the door bell rings. So, I if I want that I should not be disturbed at this time, and then I can disable the doorbell switch. So, that even if that door bell is pressed the bell will not ring.

So, that way we can make that that door bell a maskable interrupt. On the other hand if there is a power failure or if there is some say again the smoke detector that detects fire, that that sort of things they are they be they are non-maskable. So, we should not be able to mod modify them. So, we should not be able to delay them. So, that should be taken care of immediately. So, that way in a processor also depending upon the type of activity the type of IO devices that you connect to the system, some of the devices are so critical that we cannot stop them from interrupting the processor, and for some of them we may we they are not that are not of that much importance. So, when the processor feels that it can accept interrupt from those devices, it will unmask those devices those interrupts and then are then only the interrupts can be sent from those devices otherwise not.

So, that is one type of classification. Another way of classifying interrupts is by vectored and non-vectored. Like as I have said that whenever an interrupt occurs. So, what is done is that the interrupt has to be serviced. Now how the processor will service interrupts? So, processor should execute a piece of code which is pre decided, then this processor will be like as I said that whenever this smoke detected sense and interrupt that a smoke has been detected, the code for initiating the fire a fire tenders that should start for activating the fire tenders that should start.

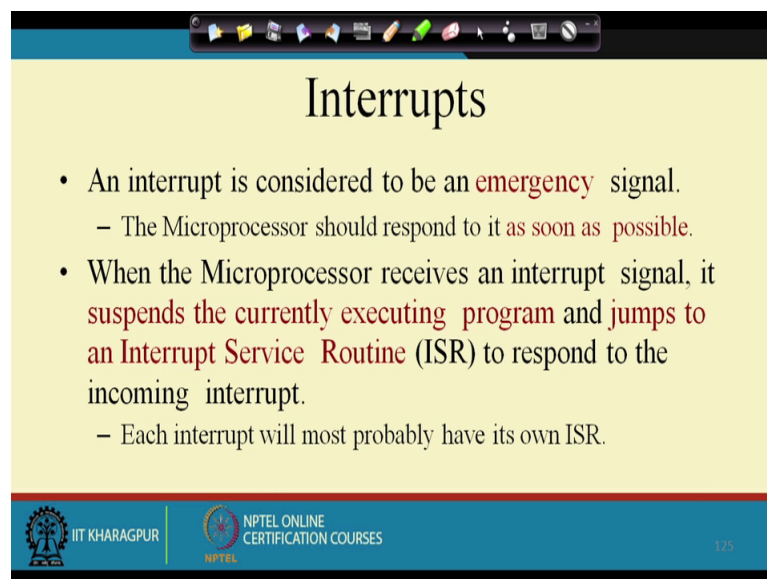
Similarly, the code for starting for the informing the fire brigade that should start; so that way we can say that those are the addresses for the ISR these are called interrupt service routine or ISR in short. So, these ISRs are to be loaded into the memory they are already available in the memory. So, before the system starts the programmer has to ensure that the interrupt service routine for all the interrupts are already in place in the memory and for some of the interrupts these address the address of the ISR is fixed ok.



So, there are some fixed location where the corresponding ISR should be loaded. So, whenever that interrupt occurs, the processor need not check for the ISR address. So, it knows where to go for the ISR. So, it immediately goes to that address. Other possibility is that they are non-vectored interrupt that is in this case when the interrupts will has occurred the processor will ask the device to tell the interrupt service routine address.

So, processor in some way it will provide the interrupt service routine address, and the processor will jump to that particular interrupt service routine. So, this way there can be two types of interrupts; one is the vectored interrupts where the address of the service routine is hard wired. So, it is known to the processor it is fixed by the processor designers, on the other hand there are non vectored interrupts where the address of the service routine needs to be supplied externally, from the outside world this interrupts address should be supplied.

(Refer Slide Time: 16:17)



## 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 | 125

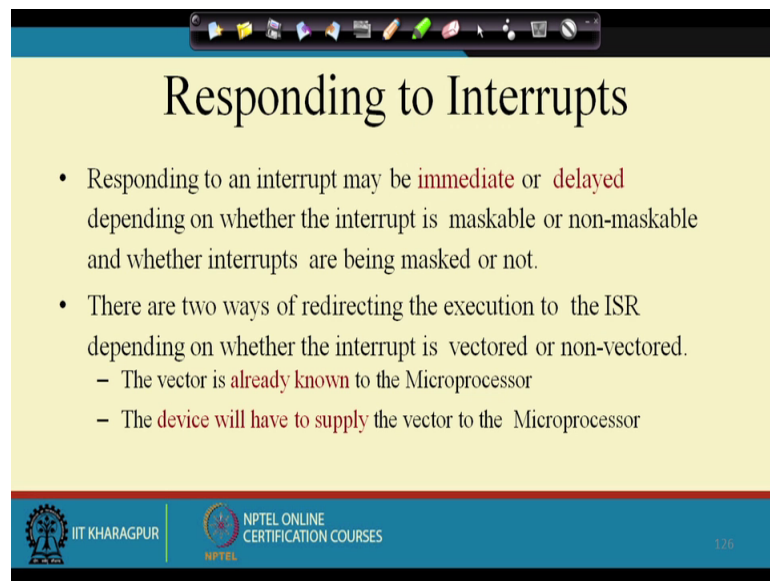
So, this is an emergency signal. So, microprocessor should respond to it as soon as possible. Because that is why this has been put as an interrupt, we should be able to respond to that interrupts as soon as possible. So, when the microprocessor receives an interrupt signal, it suspends the currently executing program and jumps to the interrupt service routine to respond to the incoming interrupt.

So, as I said that every interrupt has got a corresponding interrupt service routine. So, the processor it was doing something it was executing some program, all on a sudden the

interrupt has occurred. So, the processor should have to suspend the program that it was currently executing, and go to that interrupt service routine. Execute their interrupt service routine and once the interrupt service routine is over, then it should resume the original operation.

So, that is how this sequence of operation should take place.

(Refer Slide Time: 17:15)



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

- 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

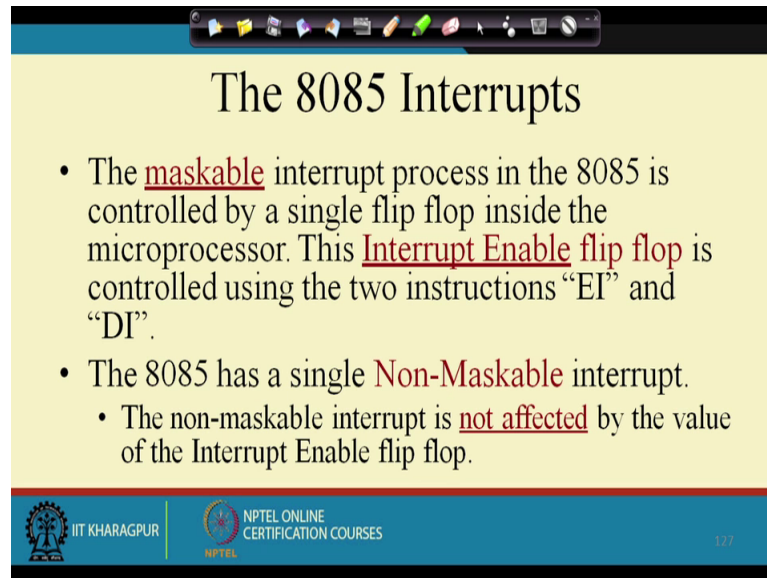
The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and the number 126.

So, the time at which you respond to the interrupts. So, that is determined by or it may be immediate or may be delayed. So, if it is if the interrupt is maskable or the interrupt is non-maskable depending on that, this inter response time may vary and. So, there are if it is immediate if the response if it is not masks the interrupt is not masked, then it is expected that the responses will be immediate.

If the interrupt is maskable then the processor may find that interrupts some time later when it tries to remove the mask for the interrupt, and then at that time it find it may find that some interrupt is waiting, and it will go and go to the interrupt service routine at that times. So, that way whether it is masked or not. So, based on that the interrupt service may be immediate or it may be delayed. So, there are two ways for redirecting this ISR to the ISR or depending upon the vectored or non-vectored, that we have already seen that if an interrupt is a vectored interrupt, then the interrupt service routine address is known to the microprocessor. So, it can immediately go to that interrupt service routine.

On the other hand if it is a non vectored interrupt, then the interrupt service routine is not known to the microprocessor. So, the device will have to supply. So, at least the microprocessor has to wait for some time for the device to come up with the interrupt service routine address. So, that way some cycles will be spent in getting the interrupt service routine address, and that way the response will be delayed by that much time.

(Refer Slide Time: 18:57)



### The 8085 Interrupts

- 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.

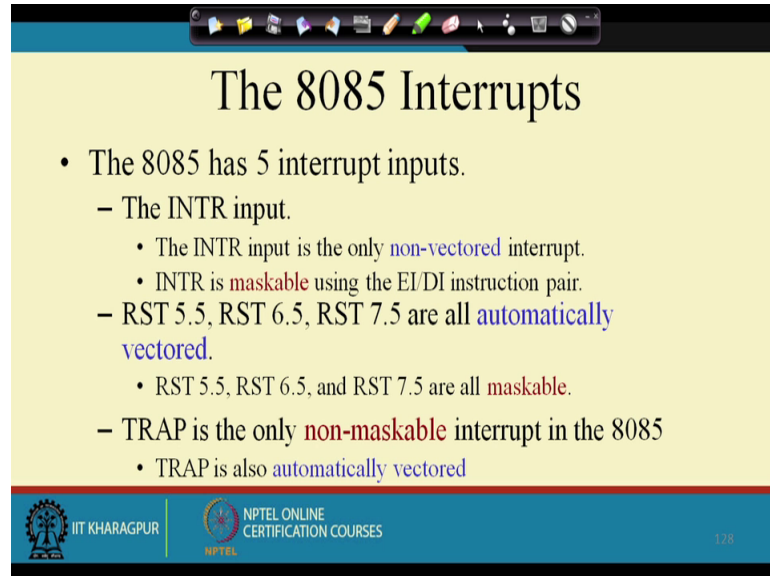
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 177

So, this discussion is true for any processor. So, 8085 is one example. So, they will see; what are the types of interrupts that 8085 has, but in general any processor will have maskable interrupt non-maskable interrupt then it will have this vectored interrupt, non vectored interrupt like that. So, in case of 8085; so there are a number of interrupts some of which are maskable, some of which are non-maskable and these maskable interrupts are controlled by one flip flop, which is known as interrupt enable flip flop. So, this interrupt enable flip flop is not directly accessible by the user, but the user can use the instruction like EI and DI to enable and disable this interrupt enable and disable the interrupt.

So, EI will enable interrupts and DI will disable interrupts, and this EI and DI. So, they actually affect this interrupt enable flip flop and by setting EI will set it to one and DI will set it to 0 that way this EI and DI will occur. So, there is only one non-maskable interrupt in case of 8085. So, that does not get affected by this EI and DI. So, that is will see that slowly. So, non-maskable interrupt will not get affected by EI and DI only the

maskable interrupts will get. So, the interrupt enable flip flop has got effect on non-maskable interrupt and not on the maskable interrupt.

(Refer Slide Time: 20:27)



**The 8085 Interrupts**

- The 8085 has 5 interrupt inputs.
  - The INTR input.
    - The INTR input is the only **non-vectored** 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 | 128

So, there are 5 interrupt in pins. So, if you remember the pin diagram that we have shown for this 8085 processor. So, there are 5 special pins, the first one is the INTR there is a there is a pin marked as intr. So, this is one of the is the only non vectored interrupt. So, non vectored interrupt means, the interrupt address root interrupt service routine address is not known and when this interrupt occurs. So, if a device which is connected to this INTR line may send an interrupt to the processor via these INTR line and when this INTR interrupt comes since it is a non vectored interrupt.

So, the processor will go to a particular execution cycle, where it will expect the device to provide it with the ISR address and after getting that ISR address the processor will go to that ISR. So, INTR is maskable interrupt. So, this is. So, INTR is a. So, one thing is it is a non vector; another feature that we have got is that it is maskable. So, these EI DI instructions can be used for enabling and disabling interrupts.

Then this are there are three more pins RST 5.5 RST 6.5 and RST 7.5. So, they are automatically they are vectored interrupt. So, for them the ISR addresses are known to the processor. So, they the processor will blanch to a particular address when these interrupts occur, and all of them are maskable. So, this 5.5 6.5 7.5 all these interrupts are maskable interrupt.

Whereas, there is another interrupt pin which is known as trap, and this is the only non-maskable interrupt in 8085 and it is also a vectored interrupt. So, the ISR address for trap is known and it is non-maskable interrupt; so if you are designing a system with 8085 as the underlying processor. So, you can understand that you have to if you have to going to connect a number of devices to the system, then they can be connected to the INTR or they can be connected to these RST lines, and the most important of all these interrupts. So, they should be connected to that that should be connected to that trap line; typically with serious conditions like power failure and all that.

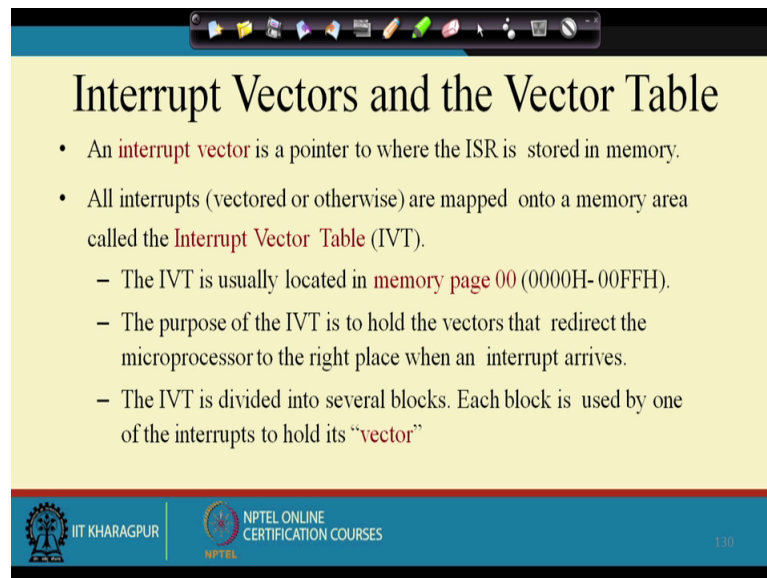
o, they are connected to the trap line. So, that this trap service. So, the power goes then the system is held on the battery. So, the system should try to save all the critical data first for the system. So, this this trap service routine may be doing that. So, it will take a note of all the critical data that are there in the system their values and all that it should copy them onto some safe locations and then it should go into a power down mode. So, that way this trap can be used for the situation which is very critical.

(Refer Slide Time: 23:35)

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

To summarize: we have got this INTR is which is maskable and it is not vectored 5.5 6.5 7.5 they are all maskable and they are all vectored as well, and this trap is a non-maskable interrupt and it is a vectored interrupt. So, the ISR address is known.

(Refer Slide Time: 23:55)



## 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 | NPTEL

130

Now, what are these ISR at the vector address and all that? So, how do where can we find these vectors and all that.

So, an interrupt vector is a pointer to where the ISR is stored in the memory. So, when I say I have said that 5.5 6.5. So, they are be vectored interrupt. So, vectored interrupt means that their address should be known. So, all interrupts vectored or otherwise they are mapped onto a memory area, which is known as interrupt vector table. So, they are normally a located in the page zero that is the in the first 256 locations all 0 to 00 ff hex.

So, this holds the vectors that redirect the microprocessor to the right place when the interrupt arrives. So, what happens is that when a particular interrupt occurs say 5.5 occurs. So, then if there is a fixed address to which the processor will jump to, and that address range comes in this that address comes in this range 00 to ff hex and in that part. So, you can have your ISR for this for this a 5.5. And we can accordingly execute the ISR there and it is divided into a block this IVT area there is 00 to ff it is divided into blocks and each block is controlled each block is used by the interrupts that were to hold its vector.

(Refer Slide Time: 25:29)

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.

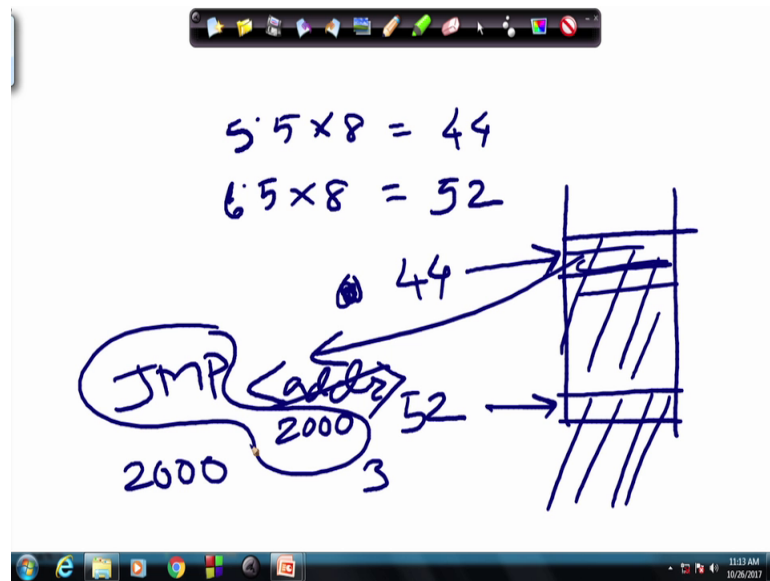
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NPTEL | 131

So, I will take an example and say that way. So, it is like this that. So, it is this interrupt vector table. So, it is like this that say this RST for this interrupt RST 5.5 has occurred. So, what the processor does is that this value is multiplied by 8. So, 5.5 into 8 so that is 24; so it will come to the memory location 24, and 24 onwards it will expect that the interrupt service routine for 5.5 to be located.

So, this after 5.5 I have 6 point five. So, 6.5 will start at 27, say that is sorry 6.5 will 5.5 will start at 5.5 into 8, 5.5 into 8 that makes it 44. So, it will start at forty four and 6.5 will start at 48. So, that way that is we have got only a few bytes for doing that.



(Refer Slide Time: 26:57)



So, 5.5 into 8; so this we this makes it 40 plus 4, 44 and say 6.5 into 8. So, this makes it 60, 48 plus four 52. So, if this 5.5 interrupts occur. So, if this is my memory. So, it will jump to the address 44; so it will jump to the address decimal 44 and if it is if a 6.5 occurs. So, it will jump to the memory location whose address is 52.

So, you see that between these two. So, 44 to 51. So, we have got 6 plus 7 8 locations. So, we have got 8 locations in between top for holding the interrupt service routine for 5.5. Similarly for 6.5 after that 7.5 will come after 8 locations. So, we have got another 8 locations to hold the interrupt service routine for 7 point for 6.5.

Now, if the interrupt service routine is very small for a particular device, then it may be contained in that 8 bytes and if it is not which is normally the case normally we do not have so small interrupts service routine that it can be held in 8 bytes. So, what we do we put a jump instruction here jump to some memory address, we put a jump instruction there so that this actual ISR for this 5.5 maybe located from memory location 2000 ok.

So, what we do we put a jump 2000 instruction here, and this requires only 3 bytes. So, the coding of this instruction requires only 3 bytes. So, that is good enough for me. So, I put or I use only three bytes here and put the code of jump 200, so that when the interrupt occurs processor will come to this point. So, it will get this instruction jump 2000. So, it will jump to the memory address 2000 and execute the ISR code there.



So, this way we can handle large ISR codes in these smaller memory locations smaller sized interrupt vector table jumps.