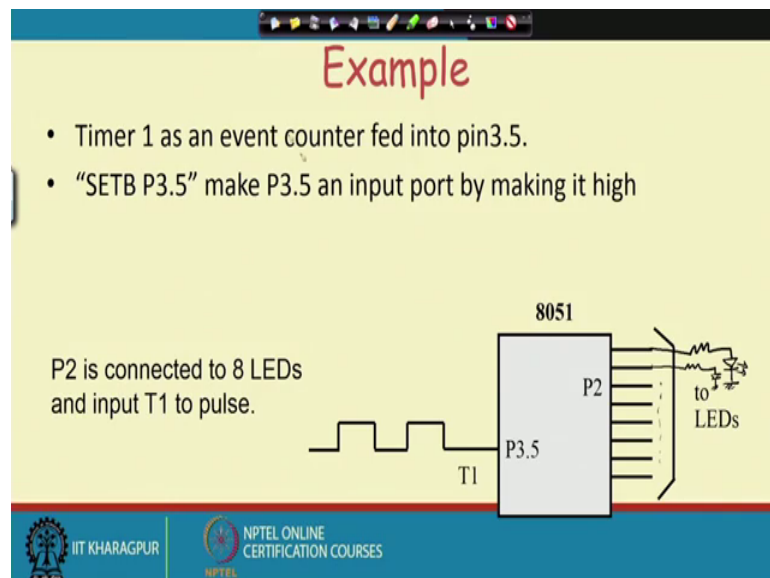


Microprocessors and Microcontrollers
Prof. Santanu Chattopadhyay
Department of E & EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 34
8051 Microcontroller (Contd.)

So, the program that we have seen here the timer 1 is acting as an event counter.

(Refer Slide Time: 00:23)



So, this is pin 3.5 so that is T 1 P. So, there I am feeding the pulses that are coming and then this set b as we have said that set b 3.5 will make this 4 3.5 as input port by making it high, then for our display purpose we can say that that I can have 8 LEDs connected on this port P 2 lines.

(Refer Slide Time: 00:48)

Example

Assuming that clock pulses are fed into pin T1, write a program for counter 1 in mode 2 to count the pulses and display the state of the TL 1 count on P2.

```
MOV TMOD,#01100000B ;mode 2, counter 1
MOV TH1,#0
SETB P3.5 ;make T1 input port
AGAIN:SETB TR1 ;start
BACK: MOV A,TL1
MOV P2,A ;display in P2
JNB TF1,Back ;overflow
CLR TR1 ;stop
CLR TF1 ;make TF=0
SJMP AGAIN ;keep doing it
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, you have seen that you are outputting this count of TL 1 through a register on to this P 2. So, we can do that here. So, we can connect a number of LEDs here. So, you can connect something like this, maybe put a register then we put 1 led here, put a led here. So, this way for each of them we can have a current limiting resistance followed by some LED ok.

So, that way for every point we can connect 1. So, 8 LED, 8 such LEDs can be connected. So, it will have a nice display of the pattern of number of pulses that had received in this T 1 and since we are we cannot display more a number more than 8 bit white. So, in port P 2 it has got only 8 lines. So, we have got 8 LEDs so that is fine. So, we can just read reload this counter with 0, after there is an overflow to effect. So, next we will look into another example.

(Refer Slide Time: 01:56)

Example

Assume that a **1-Hz frequency pulse** is connected to input pin 3.4. Write a program to display counter 0 on an LCD. Set the initial value of TH0 to -60.

Solution:

Note that on the first round, it starts from 0 and counts 256 events, since on RESET, TLO=0. To solve this problem, load TH0 with -60 at the beginning of the program.

The diagram shows an 8051 microcontroller. A 1 Hz clock pulse is connected to pin T0. Pin P3.4 is also connected to the clock pulse. The P1 port is connected to an LCD. The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker.

So, like this so we assume that a 1 hertz frequency pulse is connected to pin 3.4 and we want to write a program to display counter 0 on an LCD.

So, this whatever be the counter 0 value we want to put it on the LCD. So, initial value of TH0 we set as minus 60 and from that point onwards. So, it will be continued. So, in the first round it starts with 0 and counts the 256 events and since on reset TL 0 is 0. So, what we have to do it. So, we do not want that. So, we want to reload we want to load this TH 0 with minus 60 at the beginning of the program. So, let us have a look at the program.

(Refer Slide Time: 02:46).

```
Example

ACALL LCD_SETUP ;initialize the LCD
MOV TMOD,#00000110B ;Counter 0,mode2
MOV TH0,#-60
SETB P3.4 ;make T0 as input
AGAIN: SETB TR0 ;starts the counter
BACK: MOV A,TL0 ;every 60 events
ACALL CONV ;convert in R2,R3,R4
ACALL DISPLY ;display on LCD
JNB TF0,BACK ;loop if TF0=0
CLR TR0 ;stop
CLR TF0
SJMP AGAIN
```

So, first this LCD setup parts initializing LCD. So, this a call LCD setup. So, that is not a part of our program. So, we assume that there is an LCD setup routine. So, which will be doing this LCD setup part, some control valves have be set to LCD and all that.

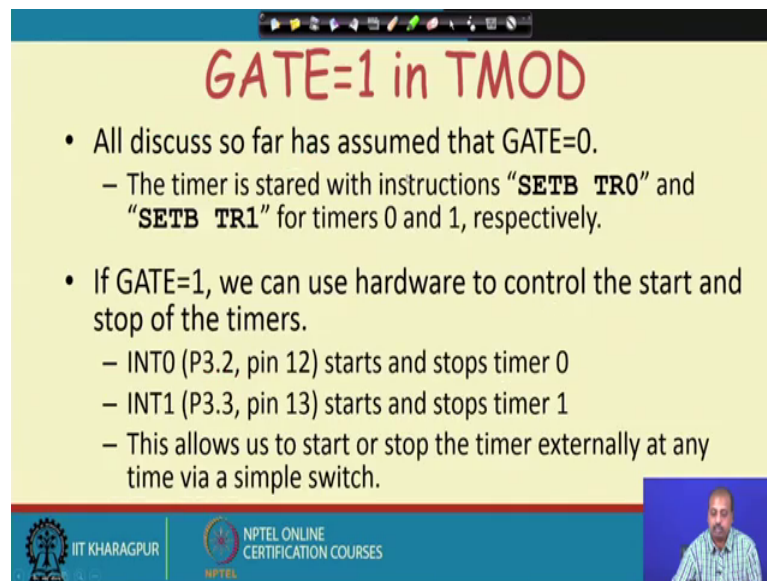
So, assuming that, it is already done by this LCD set up routine. So, we initialize counter 0 in mode 2. So, again the same thing. So, this lower order 4 bits. So, they will tell you like the mode setting and so, it is time it is a counter now and it is mode 2. So, TH0 is initialized to minus 60 and set B, P 3.4. So, this will make this T0 as input just like in the previous example we were using set B, P 3.5.

So, here we have to use set B, P 3.4 to set T0 as input. Then set B, TR0 it will start the timer, it will start the counter and move A comma TL0. So, after every 60 seconds so, we have started, we have loaded T0 with minus 60. So, after 60 seconds it will be after 60, after every 60 such events. So, there will be the timer will overflow. So, so move A comma TL0. So, then we convert this A call convert, it will convert in some register R2, R3, R4.

So, again this routine is not a part of our discussion because this is some display routines. So, that uses this register R2, R3, R4 to maybe to hold this hour, minute and second part of the display. So, that is not shown here and then we check this TH0, if the TH0 is overflowing then we go to back; that means, we have 1 loop is over. So, TL0 value will be again. So, if it is not bit. So, if it is not yet over then this TL0 value again be go to

going to A and then this converter will convert the value into this hour, minute, second display and this display routine will be called and when this bit is set then . So, now, this TR0 has to be stop the timer has to be stopped and this overflow flag has to be reset. So, that is done and going to start that timer. So, this way we can have a program for this purpose.

(Refer Slide Time: 05:19)



GATE=1 in TMOD

- All discuss so far has assumed that GATE=0.
 - The timer is started with instructions “SETB TR0” and “SETB TR1” for timers 0 and 1, respectively.
- If GATE=1, we can use hardware to control the start and stop of the timers.
 - INT0 (P3.2, pin 12) starts and stops timer 0
 - INT1 (P3.3, pin 13) starts and stops timer 1
 - This allows us to start or stop the timer externally at any time via a simple switch.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next we consider the situation where gate is equal to 1 in the T mode register. So, whatever we have discussed so far, their gate was equal to 0. So nothing, so externally there were no control. So, everything was controlled by the internal TR bit. So, we have used set B, TR0 and set B, TR1 for timer 0 and 1. Now if gate is equal to 1 we can use the hardware to control the start and stop of the timers as well. So, that is many times that is useful and they are done by this INT 0 and INT 1 pin.

So, INT 0 is pin number 12 of port number 2 and pin number INT 1 is pin number 13 of what is bit number 3. So, they can control the timer operation. So, if this gate bit is set to 1 then for timer 0, whenever this INT 0 is activated, then only the timer will operate and if it is deactivated then the timer will stop. So, that way we can have control both externally and internally for this timers.

(Refer Slide Time: 06:29)

GATE (external control)

- Timer 0 must be turned on by “**SETB TR0**”
- If GATE=1 count up if
 - INTO input is high
 - TR0=1
- If GATE=0 count up if
 - TR0=1

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, for turning on the timers for the software side for internal turn on. So, you have to use this instruction set B, TR0. So, that remains. Apart from that we can have this external gate equal to 1. So, if gate equal to 1, it will count up if INT 0 input is high and TR0 is equal to 1. So, if these two conditions are satisfied, then only the upcounting will take place for gate equal to 1 and for gate equal to 0 it will be only the software control. So, only TR0 will be equal to 1 will be sufficient. So, this we have seen

(Refer Slide Time: 07:12).

Timer Special Function Register

TIMER SFR	PURPOSE	ADDRESS	BIT-ADDRESSABLE
TCON	Control	88H	Yes
TMOD	Mode	89H	No
TL0	Timer 0 low-byte	8AH	No
TL1	Timer 1 low-byte	8BH	No
TH0	Timer 0 high-byte	8CH	No
TH1	Timer 1 high-byte	8DH	No
T2CON*	Timer 2 control	8EH	Yes
RCAP2L*	Timer 2 low-byte capture	8FH	No
RCAP2H*	Timer 2 high-byte capture	90H	No
TL2*	Timer 2 low-byte	91H	No
TH2*	Timer 2 high-byte	92H	No

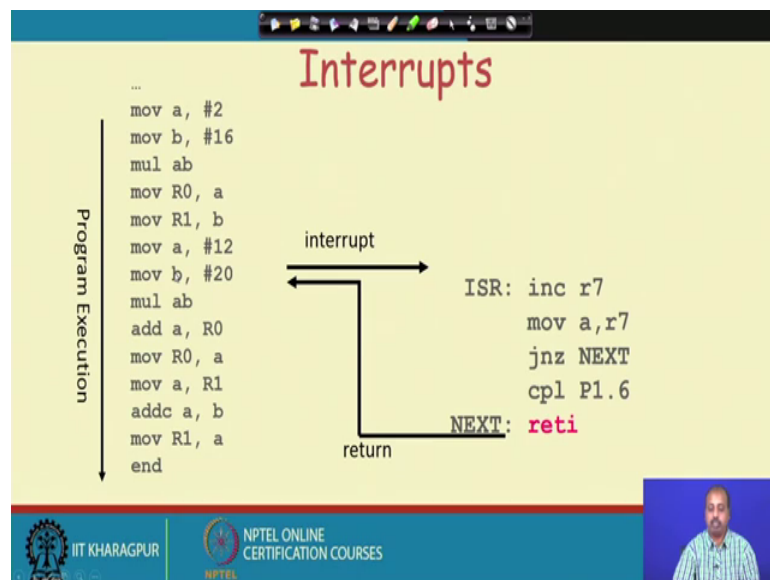
*8032/8052 only

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are many special function registers with respect to timers. So, this TCON is 88H it is bit addressable and as we see that the last digit is 8. So, it is that why it is bit addressable. So, whenever you have got this last digit as 8 or 0 the register is bit addressable in case of 8 0 5 1 then T mod registers it is the mode control. So, its address is 89 and it is not bit addressable.

So, this TL0, TL1, TH0, TH1 so, these are all this timer values, low byte and high byte and they are not bit addressable. Then we have got this T2CON . So, T2CON is for 8 0 5 2 and we have got some low byte captures. See these are some timer tools modes, some special modes are there. So, for that purpose these registers are used.

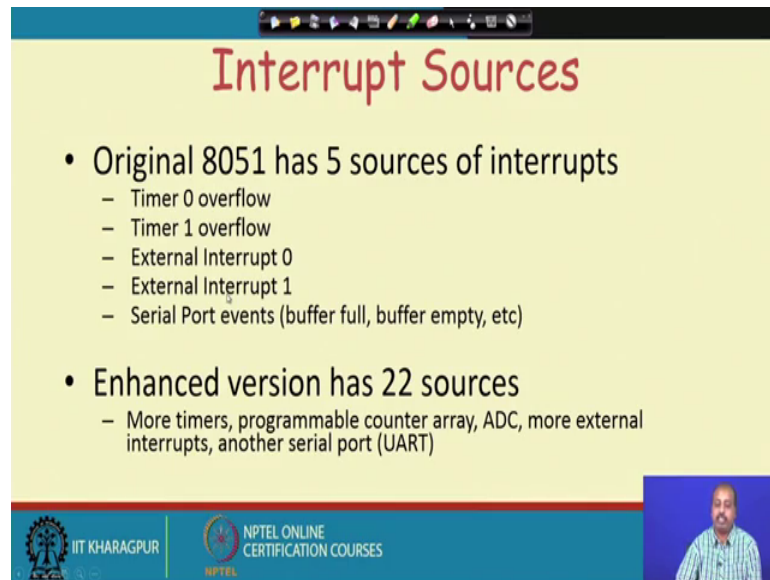
(Refer Slide Time: 08:06)



Next we will look into the interrupt. So, in case of 8 0 8 5, we have seen interrupts and then we have seen that there are different types of interrupts that that are there in 8 0 8 5, RST interrupt then INTR trap and things like that. So, in case of 8 0 5 1, we have got 2 interrupt clients. So, INT 0 and INT 1 as the external interrupts plus there are some other interrupts that they are actually internal interrupt. So, we will look into them. So, this diagram is very generic. So, if the interrupt occurs at this point. So, it will be jumping over to the ISR the corresponding interrupt service routine. It will finish up this ISR and this return I return from interrupt. So, these instructions so it will take it back to the instruction from where it was interrupted.

So, it was interrupted at this point. So, the process at jump after finishing this mov b hash 10 insta, hash twenty instructions. So, after completing this instruction the processor will come back and start this multiply ab this instruction.

(Refer Slide Time: 09:16).



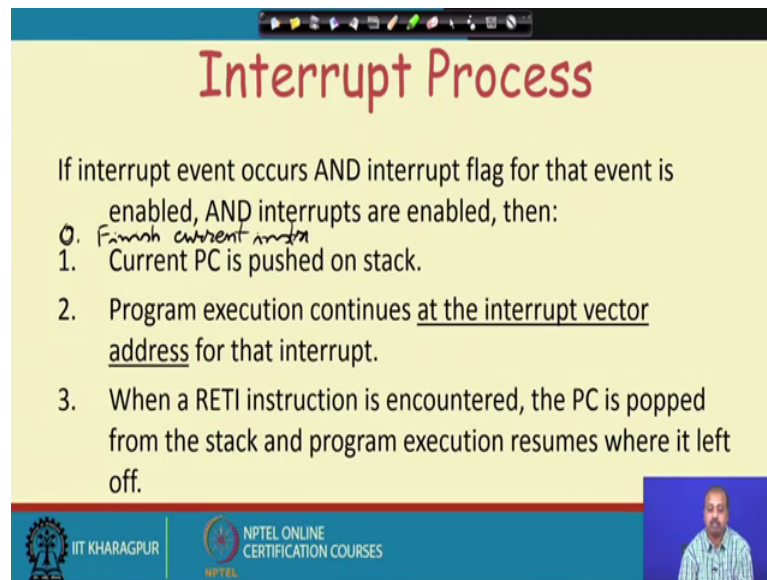
The slide is titled "Interrupt Sources" in red text. It contains two main bullet points. The first bullet point is "Original 8051 has 5 sources of interrupts" with a sub-list: "Timer 0 overflow", "Timer 1 overflow", "External Interrupt 0", "External Interrupt 1", and "Serial Port events (buffer full, buffer empty, etc)". The second bullet point is "Enhanced version has 22 sources" with a sub-list: "More timers, programmable counter array, ADC, more external interrupts, another serial port (UART)". The slide footer includes the IIT Kharagpur logo and the NPTEL Online Certification Courses logo. A small video inset of a speaker is visible in the bottom right corner.

- Original 8051 has 5 sources of interrupts
 - Timer 0 overflow
 - Timer 1 overflow
 - External Interrupt 0
 - External Interrupt 1
 - Serial Port events (buffer full, buffer empty, etc)
- Enhanced version has 22 sources
 - More timers, programmable counter array, ADC, more external interrupts, another serial port (UART)

So, in case of 8 0 5 1 there are 5 sources of interrupts. So, they are called a timer 0, timer 0 overflow, timer 1 overflow, then external interrupt 0, external interrupt 1 and serial port interrupts. So, these are the various interrupts that we have. So, timer 0, timer 1, we have seen that TF0 and TF1 bit. So, as I said that you can either (Refer Time: 09:44) in a program or you can have you can have a interrupt generated when the overflow has occurred

There are some enhanced versions of 8 0 5 1 that has got 22 different sources. So, they so, the advanced version of 8 0 5 1 that has got more timers, programmable counter that is ADC and more external interrupt and then serial interrupts. So, they are there. So, but the basic 8 0 5 1 has got only 5 interrupts. So, we will try to understand how these 5 interrupts will operate.

(Refer Slide Time: 10:18)



The slide is titled "Interrupt Process" in a large, red, serif font. Below the title, there is a paragraph: "If interrupt event occurs AND interrupt flag for that event is enabled, AND interrupts are enabled, then:". This is followed by a list of steps: "0. Finish current instruction", "1. Current PC is pushed on stack.", "2. Program execution continues at the interrupt vector address for that interrupt.", and "3. When a RETI instruction is encountered, the PC is popped from the stack and program execution resumes where it left off." The slide has a yellow background and a blue footer. The footer contains the IIT Kharagpur logo, the NPTEL logo, and the text "NPTEL ONLINE CERTIFICATION COURSES". A small video inset of a man is visible in the bottom right corner of the slide.

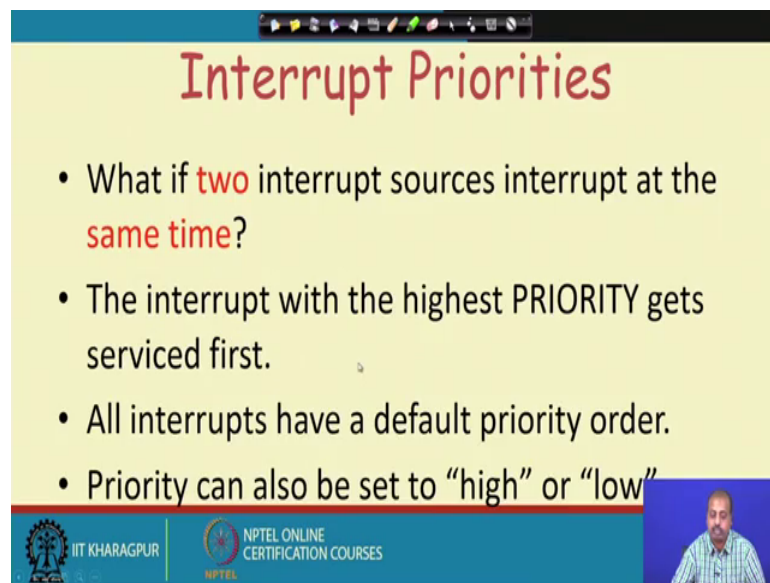
So, if the interrupt event occurs and interrupt flags for that event is enabled and interrupts are enabled, then the interrupt will be recognized. So, from the first of all the interrupt should be enabled. As we have seen for 8085 also, the all the interruptions should be enabled and we can have masking of interrupts here also the same thing that the interrupt flag for that event must be enabled for the mask, for the masking purpose because the interrupt is not masked off and the interrupt has really occurred in the void ok

So, if all these things happen then only an interrupt is considered to have occurred and it will be taken care of by the processor how is it taken care of. So, first the current program counter value is pushed on to stack and before that you can write down step number 0 which is basically finishing the current instruction. So, you can write another step as step number 0 here.

So, which says that finish current instruction, the instruction that it was executed and then only it will be going into the interrupt service routine. So, after that after finishing the current instruction, so it will be the program counter value into the stack and the program execution will continue at the interrupt vector address for that interrupt. So, in case of 8051 all the interrupts are vectored interrupts. So, there is nothing like non vectored interrupt like INTR in 8085 we had. Here all the interrupts are vectored interrupts. So, their addresses are fixed.

So, execution will start at a fixed address and then after some time, when this interrupt service routine is over, then this program counter this data instruction is encountered and then the program counter is popped out from the stack and the program execution will resume from the point where it was left off. So, that is the thing. So, this is the standard interrupt operation flow. So, that is true for other processes also and the only thing is that in case of 8051 the interrupts are all vectored interrupts.

(Refer Slide Time: 12:41).



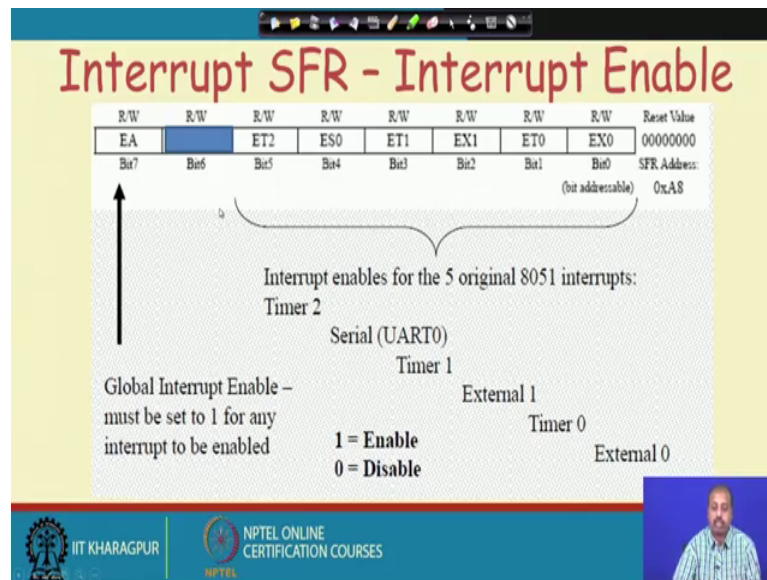
The slide is titled "Interrupt Priorities" in a red, serif font. It contains a list of four bullet points. The first bullet point is "What if two interrupt sources interrupt at the same time?". The second is "The interrupt with the highest PRIORITY gets serviced first.". The third is "All interrupts have a default priority order.". The fourth is "Priority can also be set to 'high' or 'low'". At the bottom left, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES. At the bottom right, there is a small video inset showing a man speaking.

- What if **two** interrupt sources interrupt at the **same time**?
- The interrupt with the highest PRIORITY gets serviced first.
- All interrupts have a default priority order.
- Priority can also be set to "high" or "low"

Next we look into the interrupt priorities like what if 2 interrupt sources interrupt at the same time in case of 80, 85. So, there is a priority setting and we have seen that the trap has got the highest priority and then that is non maskable interrupt in 7.5, 6.5, 5.5. So, they have got the priorities in that order.

So, in case of 8051 also so, we have got the concept of priority. The interrupt with the highest priority will get service first and all interrupts survey default priority order. So, priority can also be set to high or low. So, that can also be done. So, these are the interrupt priority.

(Refer Slide Time: 13:22)



So, there are some special function registers that are useful for these interrupts. The first special purpose register that we will look into is called Interrupt enable. So, this interrupt enable, so it has got 7 bits, 8 bits out of that this bit number 6 is not meaningful. So, it is not documented like what is the purpose of this bit number 6. So, that is not documented. So, we so, far as far as the user is concerned. So, you cannot do anything with this particular bit.

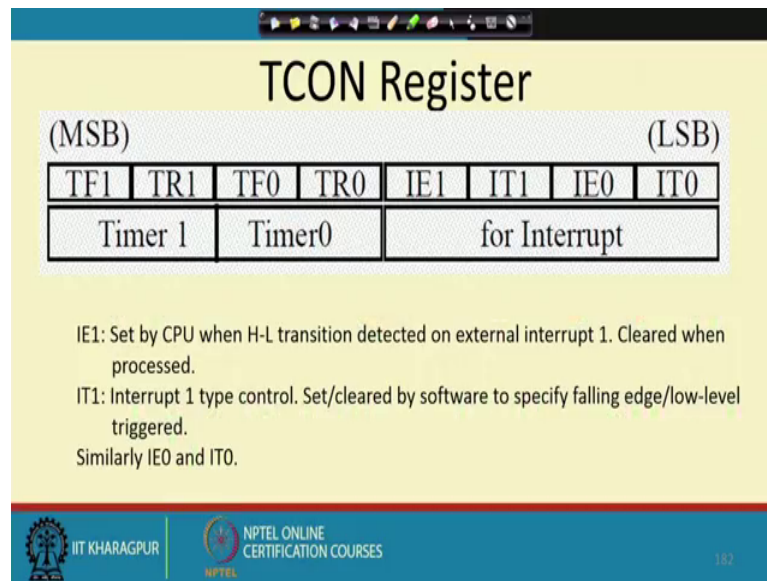
So, this first bit, bit number 7 is the enable interrupt. So, this is the enable this is for enabling all the interrupts in the system. So, it is a global interrupt enabled and it must be set to 1 for any interrupt to be enabled. So, this EA must be set to equal to 1 for this enable and if A is 0, then all the interrupts in the system they are disabled. Then this bit number 5 is the ET2. So, this is the timer interrupt. So, we have got this timer interrupt 2, then this we have got this. So, that is true for timer 2. So, timer 2, 8 0 5 1 we do not have timer 2. So, that is in 8 0 5 2 you will have, but this bit is there in 8 0 5 1 also.

Then we have got ES0. So, this is the serial interrupt or serial communication interrupt. So, we have, so for serial communication we have got both byte received and byte sent, but both of them are controlled by this A 0 by a single interrupt. So, there is both receive and transmit they are they generate the same interrupt ES0. Then we have got ET1. So, ET1 is for timer 1 for. So, this is a then EX1 for external interrupt 1, then you have got ET0 for timer 0 and EX0 for external interrupt 0 and there is also a reset value. So, reset

value for this interrupt enable register is all 0. So, you see that initially all the interrupts disabled and this interrupt enabled register is a 8H address is a 8H.

So, for any reasonable system design that allows the interrupt to be occurring in the outside world; so, we should set this EA bit to 1. So, this may be 1 important step in the whole operation of the system that the EA bit should be turned on; otherwise none of the interrupts will be sensed by the processor.

(Refer Slide Time: 16:12)



So, that is the TCON register that we have seen previously it comes once, when we are going into this interrupt discussion. So, we have seen this higher order nibble. These four bit, so they are dedicated for timers timer 1 and timer 0 they are overflow flag and the run flag.

So, those 2 are bits, but this lower order 4 bits. So, they are actually IE1, IT1, IE0 and IT0. So, IE1, so this is set by the CPU, when high to low transition detected on external interrupt 1. So, this will be, so when there is a high to low transition on this interrupt 1 then this IE1 bit will be set and accordingly this IE0 bit will be set if there is a high to low transition detected by the processor on the interrupt on the INT 0 line. So, there are two interrupt pins INT 0 and INT 1 and whenever it sees a high to low transition on those lines, these pins are these bits are set to their set to proper values, they are set to 1 and then we have go IT1. So, IT1 is the interrupt 1 type control. So, if it is it is set or cleared by software to specify the falling edge or low level triggered.

So, if it is, so you can control this set this IT1 bit. So, if you set this IT1 bit to 1, then it will be a falling edge triggered flip, a falling edge triggered sensing. So, when the interrupt line goes from high to low, it will be taken to be interrupting the system and if it is 0, then when the INT 1 line is INT 1 line is low, it will be taken as interrupt for the processor. So, you have to be careful in setting these interrupt types. So, depending upon the device that we are connecting to this to the system so, we may like to have this interrupts of type this H triggered or level triggered, but both are low, both so, H triggering also high to low edge and this level triggering is also this low level triggered and similarly we have got these IE0 and IT0 bits for the INT 0 interrupt.

(Refer Slide Time: 18:34)

The slide, titled "Interrupt Priorities", displays the IP register structure and lists priority settings. The IP register is shown as a row of bits: --, --, PT2, PS, PT1, PX1, PTO, PX0. The PT1 and PX1 bits are circled in blue. Handwritten annotations show the priority order: INT1 > INT0 > TF0 > TF1 > RI + TI. A list of bullet points explains the bits and provides assembly code examples.

- Default priority: $INT0 > TF0 > INT1 > TF1 > RI + TI$
- To alter priority set IP register
- PT2: Priority of timer 2 interrupt
- PS: Priority for serial port interrupt, and so on.
- "MOV IP, #00000100B" sets INT1 to have the highest priority
- "MOV IP, #00001100B" sets priorities as $INT1 > TF1 > INT0 > TF0 > RI + TI$

If you look into the priorities, we have got. So, this is the default priority. So, INT 0 has the highest priority followed by TF0 timer that is timer 0, followed by interrupt 1, followed by timer 1. Then this RI plus TI so, this is received interrupt and transmit interrupt. So, these are for serial communication. So, they are by a single interrupt. So, that is why we have we can write it as RI plus TI, as if this 1 interrupt is generated. So, this is the default priority. So, this immediately tells you that if a number of interrupts occur simultaneously, then the processor will first respond to INT 0 and if INT 0 is not there, it will respond to TF0, TF0 is not there it will respond to INT1 . So, it will go like that, but unlike 8 0 8 5. So, here you can alter the priority setting. So, you can you can say like this, that we have got this PT2. So, this is the 1 register which is called IP

register. So, interrupt priority register the first 2 bits do not have any do not have any functionality, then this PT2 is the priority of timer 2 interrupt.

So, PS is the priority of timer serial port interrupts. So, PT1 is the priority of timer 1 interrupt. So, then this PX1 is the priority of external interrupt 1, PT0 is the priority of timer interrupt 0 like that. Now, if you set 1 bit to any of those bits to, that interrupt will assume the highest priority. For example, if you look into this comment. So, MOV IP, this instruction, MOV IP comma 00000100B, the binary pattern so, 1 this PX1 this bit is 1 and rest of the bits are 0. So, in that case INT 1 will assume the highest priority. So, this order will now change and it will be something like this. So, once you do this thing this setting. So, my INT 1 will have the highest priority INT 1, followed by INT 0, then TF0, then INT 1, sorry then INT 1 is already done, the TF0, then this TF1 and then this RI plus TI.

So, the order will change. So, order will change. So, that we can have this thing it will be representing the new order. So, you can change the priorities and so, that interrupt will assume the highest priority and rest of them will follow the previous order. So, naturally it raises the question that can we have this, can we have more than 1 bit set to 1. So, if you do it like this then these 2 interrupts. So, 1 1, I am looking for. So, this interrupt and this interrupt, they are set to 1 and rest are all 0 so; that means, that these two interrupts. So, they will acquire priorities higher than the remaining ones. So, your interrupt INT 1, sorry this timer 1 and this external interrupt 1 to they will assume the highest priority, followed by others and within them within these two the priority will be decided by the default priority order.

So, if you consult this relation. So, you see that this INT. So, between INT 1 and timer 1, INT 1 has got the higher priority. So, as a result if you set it like this then INT 1 will have the highest priority then it will be followed by this TF1. So, like this. So, this relationship will come. So, INT 1 will have the highest priority followed by TF1 and then that will be followed by rest of the interrupts in the same order. So, so INT 0, TF0 then RI plus TI, they are the remaining interrupts. So, they will be following in the previous order, only thing is that these two interrupts they will have priorities higher than others.

So, naturally you can understand that you cannot set these interrupt priorities arbitrarily. So, there are some values, some of the interrupt combinations are feasible, but you

cannot set all of them simultaneously, but still that is a good option, like we can change the default interrupt priorities and this is particularly true because in a system when you are designing the microcontroller based some embedded application.

So, it may so happen that you need to take interrupt from different sources and at different times of program or different phases of program. So, you may require the priorities to be different though, so, instead of connecting the changing the hardware connections. So, you can just change this interrupt pattern, interrupt priority pattern. So, that this changing priority will be taken care of so, that is the interrupt priority register.

(Refer Slide Time: 23:53)

Interrupt Vectors

Each interrupt has a **specific** place in **code** memory where program execution (interrupt service routine) begins.

Reset:	0000h	✓
External Interrupt 0:	0003h	↓
Timer 0 overflow:	000Bh	↓
External Interrupt 1:	0013h	
Timer 1 overflow:	001Bh	
Serial :	0023h	
Timer 2 overflow (8052+)	002bh	

Note: that there are only 8 memory locations between vectors.

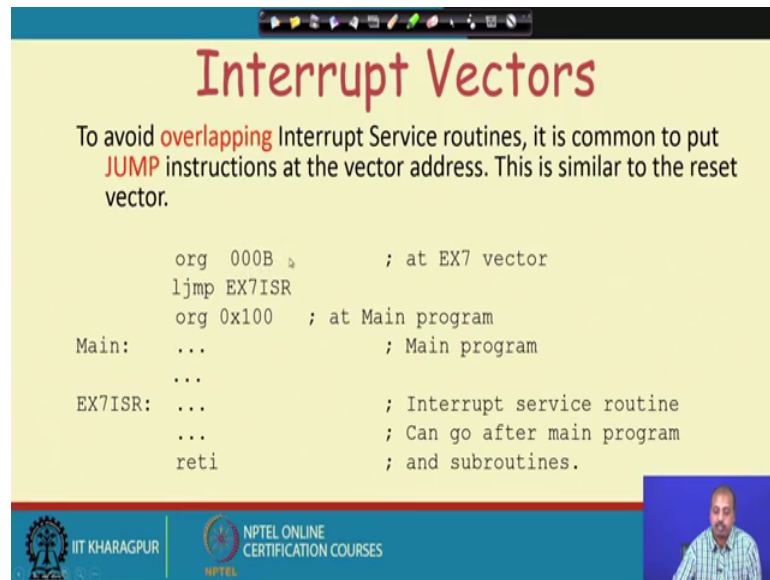
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Next, we look into the interrupt vectors. So, interrupt vectors. So, as I said that in case of 8051 all interrupts are vectored interrupt. So, 8051. So, 8085 it had got only a few vectored interrupts and INTR are all non vectors. So, here in case of 8051 this vectored interrupt their addresses are fixed like this reset. So, it will take you to the address 0000. So, this address is fixed then this INT 0, external interrupt 0. So, this address is 0003 timer 0 is 000B. So, you see if you look into these two locations, 3 and B. So, you have got only 8 memory locations in between.

So, if you are writing the interrupt service routine for external interrupts 0. So, if you find that this 8 bytes are sufficient, then for holding your interrupts it is fine. If it is not you have to take care of. So, that you know if you feel some sort of jump instructions and all that, as we see as we have seen in 8085 as well. Now, so these are the interrupt

vector location. So, so this serial interrupt which has the highest address 0023h and this timer 2 overflow is that is for 8 0 5 2. So, that is 002b. So, that way we can have different interrupt addresses the vector addresses. So, this is fixed by the designer.

(Refer Slide Time: 25:27)



The slide is titled "Interrupt Vectors" in a large, red, serif font. Below the title, there is a paragraph of text: "To avoid overlapping Interrupt Service routines, it is common to put JUMP instructions at the vector address. This is similar to the reset vector." Below this text is a block of assembly code. The code is as follows:

```
org 000B ; at EX7 vector
ljmp EX7ISR
org 0x100 ; at Main program
Main: ... ; Main program
...
EX7ISR: ... ; Interrupt service routine
... ; Can go after main program
reti ; and subroutines.
```

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


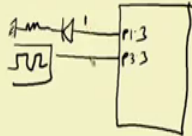
So, to avoid overlapping of interrupt service routines so, it is common to put jump instruction, like here say, we are we are putting say at origins 00. So, we are putting this ljmp, let us say EX 7 ISR. So, this is some interrupt service routine the external interrupt some service routine and this is the main program is at 100. So, from this main program starts. So, now, 000b address and there it is seeing this instruction. So, at this point I can have the code for doing the reset operation. So, that way it is the reset operation will take place.



(Refer Slide Time: 26:21).

Example Interrupt Service Routine

Pin 3.3 (INT1) is connected to a pulse generator. Write a program in which the falling edge of the pulse will send a high to P1.3, connected to a LED.

```
org 0000h
ljmp main
; ISR for hardware interrupt INT1
org 0013h
setb p1.3
mov r3, #255
back: djnz r3, back
      clr P1.3
      reti
; Main program for initialization
org 30h
main: setb tcon.2 ; make INT1 edge triggered
      mov ie, #10000100B ; enable int1
here: sjmp here
```



So, this is an example where we have got this pin 3.3, INT 1 connected to a pulse generator and we will write a program in which the following edge of the pulse will send a high bit to P 1.3 connected to LED. So, what is the situation? So, we have got this. So, we have got the situation like this that we have got this P 3.3. So, this P 3.3 is connected to a pulse generator. So, here I have got a pulse generator that can generate this type of pulses and. So, falling edge of the pulse will send high to P 1.3. So, P 1.3 is there. So, this is the P 1.3 and there I have got 1 LED connected. So, here some LED is connected here. So, if I put a 1 at this point the led will glow.

So, this program so, we can develop using this interrupt. So, this when this pulse low edge will come; so, it will generate an interrupt and that interrupt will be interrupt service routine will be putting a 1 at this point ok.