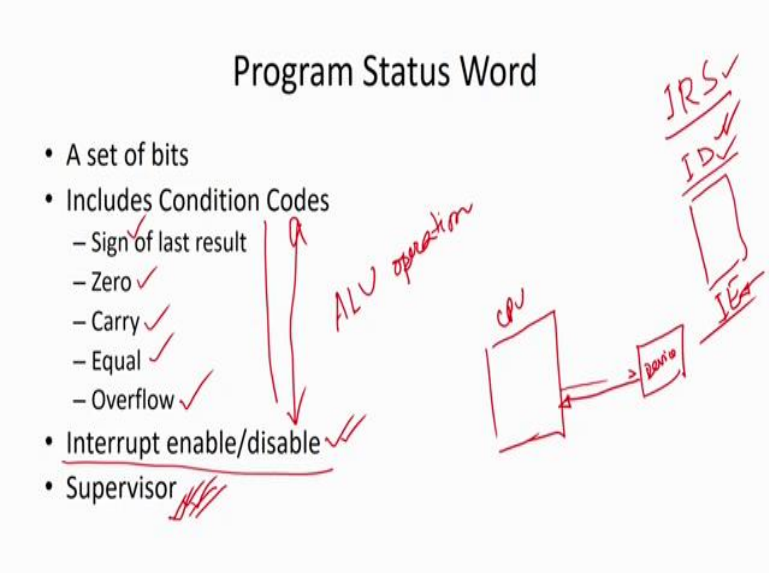
(Refer Slide Time: 29:11).



I think we have mentioned or we have discussed all those particular bits along with that, we may have some other bits also. So, these are the bits basically affected by some ALU operation. So, programmer cannot set or reset those particular bits ok. These flag bits will be always affected by the result of an ALU, but along with that we are having some flags also. So, one of the flag bits is your interrupt enable and disable. So, in that particular case when we are going to look for interrupt enable and disable.

So, what basically we have said, this is the processor and this is a device say CPU and say device. So, device is giving an interrupt. So, whenever is interrupt is coming now processor, what processor will do? It will complete the execution of the current instruction and going to give service to the devices by indicating with another signal call, say acknowledgment signal.

Now, what will happen if whenever you are doing, if some device is going to interrupt? We are bound to give the service to the interrupted devices, but if processor is engaged in some important work, high priority work, just say in case of your this thing, say when we are looking for the aircraft control. So, in the control unit then what will happen? We are monitoring the aircraft and accordingly we have to schedule the remaining aircraft.

So, when it is doing that particular job processor need not be interrupted by any other devices, because if it is going to give the service to that interrupted devices that during that period something else may happen. So, in that particular case what will happen? We may have a provision to say whether we will allow interrupt or we will disallow interrupt. So, for that we

are having a flag bit called interrupt enable. So, if we set it, then it says that we are enabling interrupt; that means, during the execution of a particular program, any devices can interrupt the processor, if we set it to interrupt disable then what will happen?.

We are setting, we are disabling the interrupt and in that particular case what will happen if interrupt comes, then processor is not going to give the service to the interrupted devices, it will first complete the current program, after that only it will look for that particular interrupt devices. So, this is the way we can control it also whether interrupt will be allowed or not, but there is a risk, there is a problem say you have written an interrupt service routine ok and your first instruction is a interrupt disable. We have disabled the interrupt and you have written your program.

So, what will happen after completion of the interrupt service routine, it will come to the main program, you have disabled the interrupt at that particular point. So, it remains disabled. So, after that processor is not going to give service to the interrupt, because it is already disabled. So, responsibility lies with the programmer who is going to write the interrupt service routine. If he writes interrupt disable after completion of the interrupt service routine, we should enable it also interrupt enable. So, this is basically I am talking about an interrupt service routine.

So, while you are going to write your program also, when you are going to develop the software if you feel that if you think that processor should not be interrupted while executing this particular program. So, at the beginning you can disable it, but before completion, before coming out from this particular software you should enable the interrupt; otherwise for remaining span the interrupt will remain as disabled. So, processor is not going to give service to any of the devices. So, this responsibility lies with the programmer to enable it and disable it.

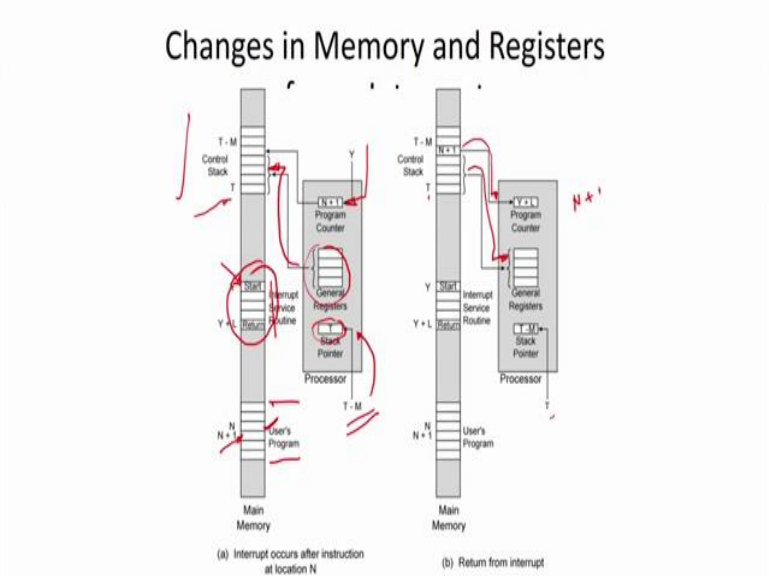
So that means, this can be set and reset by a programmer, but other bits cannot be set and reset by a programmer that will be set and reset by ALU operation. Like that similarly we are having one particular flag bits which is the supervisor mode. So, if you are working with a UNIX system or Linux system, you may be knowing that we are having different kind of user; one is your root user, all of you know about it.

So, if you are having a privilege of root user you can do many more system operation, you can do many more things, you can set or reset many more things, but if you are in, you are in the user mode then only you can work with the processor which is relevant to you only, you cannot

touch any other system parameters. So, for that also what will happen? We can have a flag bits and that flag bit will be set to either supervisor mode or non-supervisor mode. So, for root generally we set it as a supervisor mode.

So, when you login as a root then what will happen? You are having in the supervisor mode. Now we can change many more system parameter, but if you login as a user then you are not in supervisor or you are not having the supervisor mode. So, you can carry out only the work or the privilege assigned to you only you cannot change any of the system parameters. So, such type of flag bits are also there which will be, which can be set by the programmer. So, some bits are set by the programmer and some bits cannot be set by the programmer. So, combination of all those things are known as my program status word.

(Refer Slide Time: 34:58).



So, this is a simple example, just say how, what will happen when we are going to perform the interrupt service routine, so it says that this is my user program ok. Currently the values of program counter is $N + 1$. So, what does it means; that means, we are executing this particular instruction that is available in this particular memory location N ok. Now, at that time while you are executing this thing, you just see that some interrupts has arrived, then what processor will do?.

Processor will complete the execution of this instruction after it is going to give the service to the interrupted devices; that means, it is going to execute the corresponding interrupt service routine. So, before going to start that interrupt service routine what it needs to do. Say, basically

it shows like that at that time that value of the stack pointer is T; that means, this is my stack. So, stack pointer is pointing at that particular point. So, when it receives an interrupt what it does? First it is going to store those particular general purpose registers.

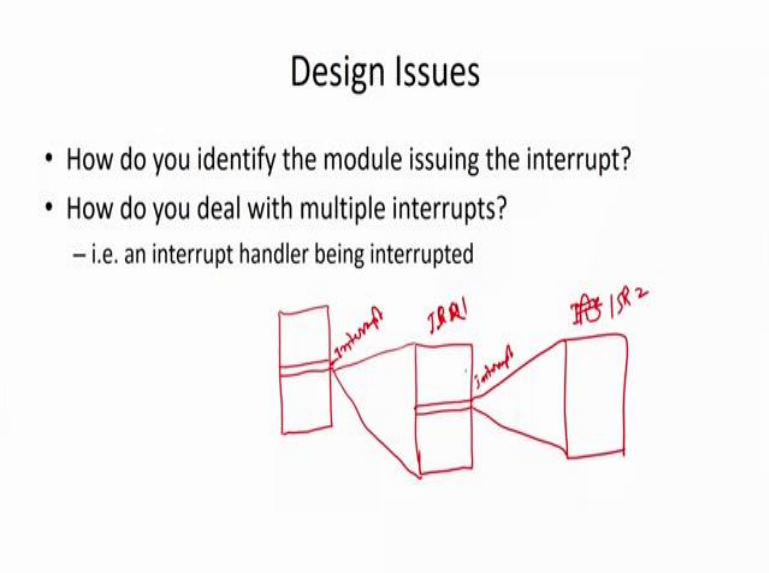
So, we are pushing it to the control stack, after pushing this things to the control stack, then we are pushing this particular program counter value to the stack. Now you just see that stack is now from T and we are putting some information. Now values of the stack pointer that top of the stack is becoming the address $T - M$. So, this $T - M$ is put into the stack pointer, because after completion of the service routine we have to pop out from this particular point $T - M$ to T. So, once we put those things then what will happen. Now this is the interrupt service routine, it started from Y and going to $Y + L$. So, the program counter values will be loaded with this particular address Y.

So, now, we are set. Now what will happen when we are going to fetch the next instruction, then what will happen? We are going to fetch the instruction from this particular memory location Y. So, like that now we are going to execute the program that is available over here. Now, once it is done, once we complete this particular interrupt service routine that return instruction is coming, then we have to restore the value. Now what we are doing? You just see that first this is the top of the stack, we push, we pop it now we are popping it to the $Y - L$, all the registers value that we have stored we again pop it out and in accordingly it is going to push it over here.

Now accordingly that's these values will be reduced. Now top of the stack becomes T. Now we are going to put this particular T to over here. So, when I am bringing this information, you just see now that program counter value is becoming $N + 1$. Now, we are going to fetch the instruction that is in this memory location $N + 1$. So, this is the way that we are giving service to the interrupted devices. We are storing the context current context of the processor into the system stack execute the interrupt service routine.

After completion of the interrupt service routine restore back the processor status and start executing the program from the next instruction. Now this is the way that we are handling the interrupt ok. Now, I think it is clear to us, how we are handling the interrupt and how we are going to give service to the interrupted devices by running an interrupt service routine. Now what are the design issues that we have? So, some of the issues that I have identified over here.

(Refer Slide Time: 38:38).



How do we identify the module using the interrupt? Because already I am saying that there may be several I/O module. In every I/O module we may connect several devices. So, we are going to work with a particular device, it will come to a particular I/O module. So, how to identify that particular module which is receiving the interrupt, how do we deal with the multiple interrupts. Now say this is my main program ok.

So, I am executing this particular instruction at that point 1 interrupt is coming; that means, I am running this particular interrupt service routine. So, from here we are coming to this particular interrupt service routine, when I am running this interrupt service routine when say some more interrupt is coming over here say 1. So, another interrupt is coming over here.

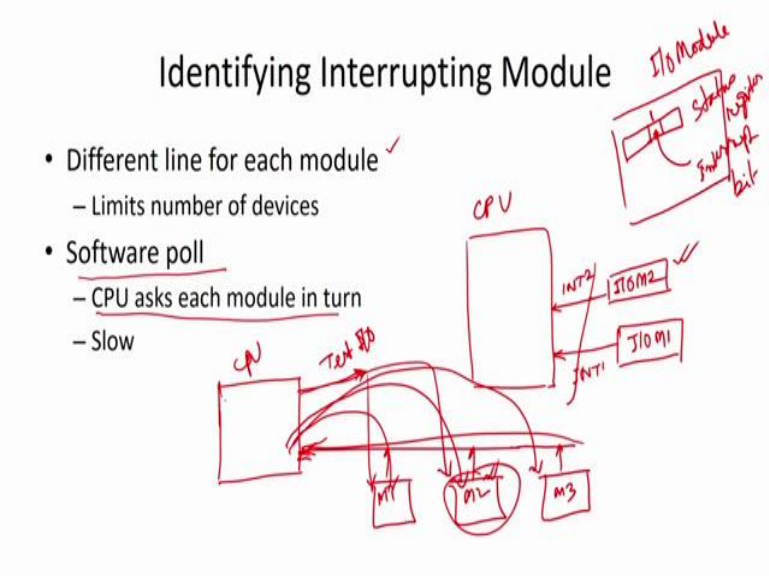
So, now what to do, whether can I go for interrupt service routine 2, sorry interrupt service routine 2 whether I should execute this particular interrupt, then only you should go for this thing. So, these are the issues that we are having. So, how we are going to handle this particular interrupt. So, this is one issue or secondly, we may complete this particular interrupt, then you can go to this. So, basically it depends on the priorities that we are going to set with the interrupt ok.

So, we will see these particular issues. So, these are the issues. Now how to identify the module using a interrupt. This is the addressing scheme that already we have discussed when we discussed about the programmed I/O and we have seen that how we are going to connect the I/O devices, we are having two way to do it; one is your memory mapped I/O and second one

is your isolated I/O. So, for all the modules we are having, an address, but if several modules are connecting, so several addresses are there. How we will be knowing which module has given the interrupt. So, you have to identify the appropriate module.

So, this is the issues how to identify it, but we know the addressing scheme. Through addressing scheme we can give address to each and every I/O module

(Refer Slide Time: 40:53).



So, there are different ways to do it to write, define. Identifying the interrupt module, 1 of issue is your define lines for its module. So, basically what will happen, say this is my processor, I am saying that I am having an line through which I can give interrupts, so this may be your, say I/O module 1. So, if interrupt is coming through this particular interrupt line, interrupt line 1 then I will be knowing that I have to give service to the interrupt module 1 I/O module 1.

So, similarly I can have another interrupt line where I am connecting, say I/O module M_2. Second module is connected to the interrupt line 2. So, if interrupt is coming through this particular interrupt line 2, then we know that it is coming from this I/O module 2. So, in that way we can resolve this issue, but how many lines you are going to provide, we don't know the how many devices we are going to connect. Now while you are designing the processor, we do not know where we are going to use it, depending on the use of the processor we are going to connect several devices.

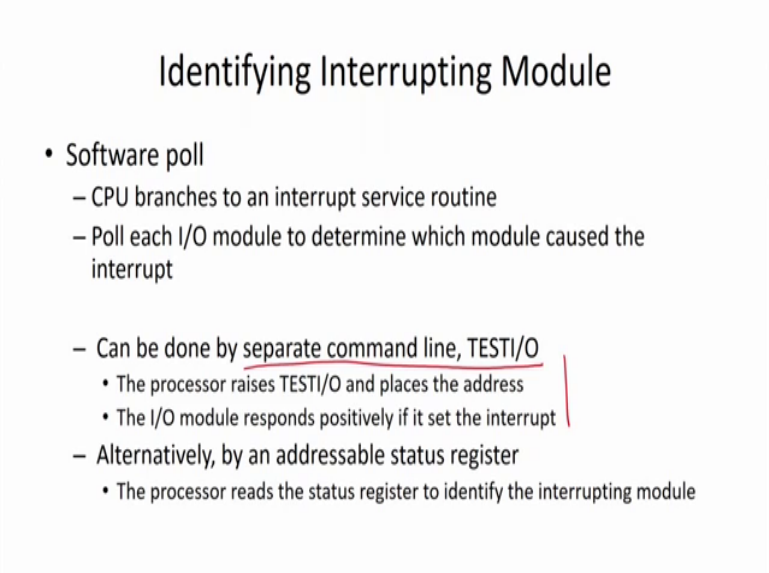
So, during design issues we cannot simply give the number of lines. We cannot freeze it, because any number may not be sufficient enough ok. So, for that what will happen? We have to look for a generic solution, so that that processor can be used for any situation, any number of devices can be used. So, for that one of the method is called software poll. Here we are going to use some software routine, CPU asks each module in turn ok. So, in that particular case what will happen? So, very simple way I can say that this is the software poll. Now processor is going to run a routine, software routine to identify the I/O module which has given this particular interrupt, you just see.

After completion of the current instruction, we know that some interrupt request is there. Now processor is going to give service to the interrupted devices. Now if you are conducting several interrupted devices. Now how to identify which module has given this particular interrupt. So, for that before running the device service routine, it will run another software which may be a part of my operating system which is known as your software poll. So, in that particular case, it says that CPU asks each module in turn ok. Now in that particular case situation may be something like that, this is my processor ok, through this particular interrupt line we are connecting many more devices or many more I/O module; say this is module 1, module 2, module 3.

So, any module can give interrupt. Now when processor is going to say that, now it has got an interrupt. Now it have to identify which module has given it. So, I am saying that in software poll is going to run a software. So, in that particular case what will happen? Now CPU branches to the interrupt service routine. In that particular case we are having the software, poll each I/O module to determine which module caused the interrupt. So, first it will going to see check this particular module ok, that service routine or say the software poll routine, whether interrupt is coming from this particular module or not.

If it is not coming, then it will be going to check for the next module ok. If it is not coming from it, then processor is going to check for the third module. So, this is the way it can look for it.

(Refer Slide Time: 44:47).



So, it is going to poll each and every module I/O module or each and every device to check which one has given the interrupt ok. So, this is the way it will identify and once it will identify that this particular module is giving the interrupt, accordingly it will place the addresses of this particular module, and going to work with this particular module. So, now, how to do these things. So, or doing this things we are having define a process, one it says that it can be done by separate command line TEST I/O. So, we may have one command line or one control line through that particular control line, it is going to give TEST I/O. So, the processor raises the TEST I/O and puts the addresses.

So, what will happen we can have a control signal called TEST I/O and it will raise these things and along with that, it will give the address of this particular I/O module. Then if this particular module has generated the interrupt, then what will happen, accordingly it will respond to the processor that; yes, it has done it. If it is not doing it then it is not going to respond say not done it, then what will happen. This signal will go to the next module along with the appropriate address of this particular module and this module will also behave like that. If generated the interrupt, then it will respond to the processor. This is one way of doing it, which is known as your separate command line called TEST I/O.

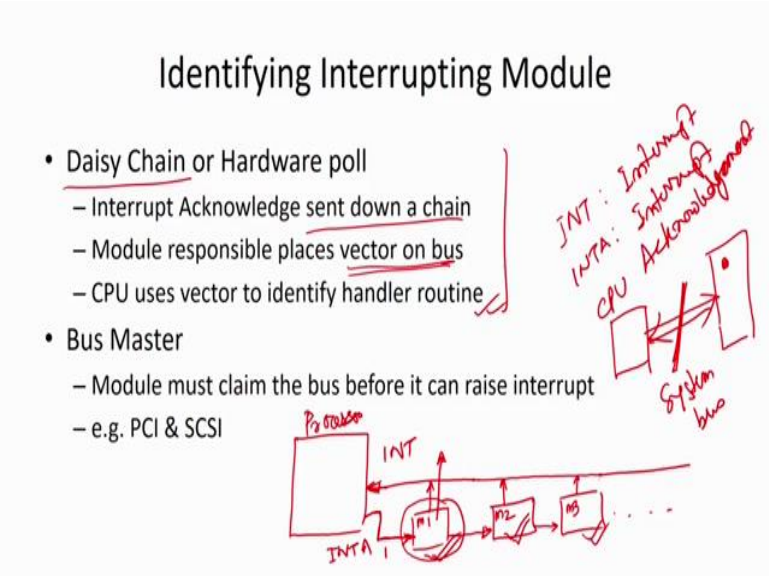
The processor raises the TEST I/O and places the address the I/O module respond positively if said the interrupt. So, this is one way, second way is done it by addressable status register. So, basically what will happen? Already I said that for I/O module, if I am having an I/O module

then we are having called as status register ok. In last class I think I have explained it that we are having a status register. So, here we may have 1 bit position, maybe which is known as your interrupt bit ok.

So, when this I/O module will set this particular interrupt, then what will happen? It will set this particular bit to 1; that means, it will indicate that interrupt has been raised by this particular module. So, in that particular case what will happen? This processor is going to check this particular status bits, say 1 says that one interrupt that processes has got an interrupt. Then it will complete the execution of the current instruction and now it is going to look for the giving service to the interrupted devices.

Now, it is going to check this particular bit if this is set, then it will be a processor will be knowing that this module has given the interrupt and accordingly processor will set the address of this module and going to carry out the work. If it is 0, then this is going to check the next module and accordingly it will go this way. So, it is going from module to a module, just to check it whether who has raised the interrupt. So, this is another way to resolve it. So, these are the two ways we can resolve it. We can see which devices or which I/O module has raised the interrupt and once we can identify it then what will happen? We are going to run the appropriate service routine, interrupt service routine for that particular device.

(Refer Slide Time: 48:09).



So, this is basically say we are doing into the software levels. So, another one we are having an hardware level and which is known as your hardware poll. It is similar to that particular

polling only, software polling only, but it is done in the hardware level. So, you don't have any service routine. So, how we are doing it? You just see say this is the processor, now say this is the interrupt line ok. So, through this particular interrupt line module will be connected. So, this is the module 1, module 2 and module 3 like that ok.

Now, what will happen when processor is going to give the getting an interrupt, then it should give the service to the processor. So, most of the processor is having one line called interrupt acknowledgement. So, basically this is the terminology we use INT basically look for, says that this is the line interrupt line and INTA is your interrupt acknowledgment ok. So, these are the two lines. So now, when processor is going to service to the interrupted devices, then it will set this particular interrupt acknowledgement to one, it says that now processor is ready to give the service.

Now, when it will come to the first module, then what will happen? If that module has raised the interrupt then it will capture this particular interrupt acknowledgement and accordingly it will place it in function, maybe address of this particular device to indicate that this module 1 has raised the interrupt. If module 1 has not raised the interrupt then what will happen? It will pass this interrupt acknowledgement to the next devices. So, now, next devices also act accordingly. If it has raised the interrupt then it will give the indication to the processor along with its address. If it has not devices, then it will pass the interrupt acknowledgement to M3.

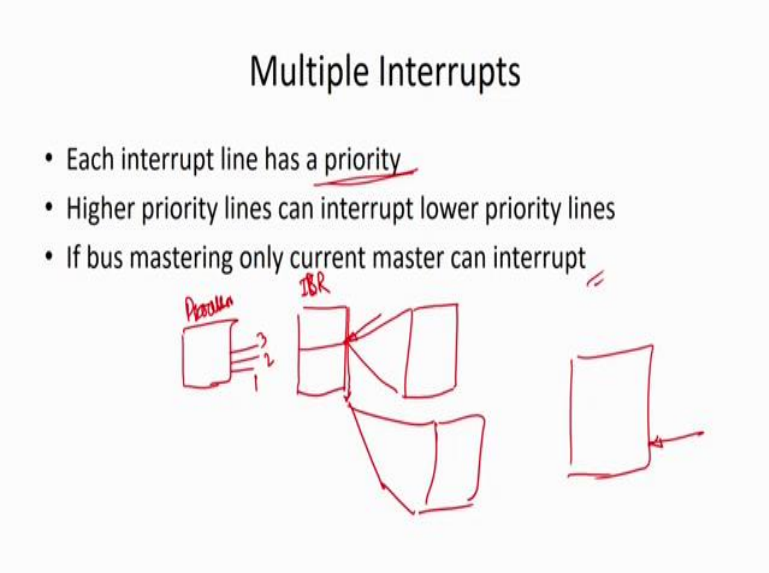
So, this is you just see that we are implementing these things in the hardware. So, that's why you are talking about it is the hardware poll or the name given is your daisy chain, because devices are connected in a same fashion one after another in one chain itself. So, this is the things that we are having the interrupt acknowledgment sent down the chain. So, this is the chain. So, it is sent on the chain, module responsible places vector on bus. This vector basically nothing but a address of this particular devices to indicate that; yes it has done it, CPU uses vector to identify the handle Routine.

So, depending on those particular vector what will happen? Now processor is going to identify which service routine it needs to execute and accordingly it will set the program counter value. So, this is one way and another way is called bus master. So, we are having a bus master. So, basically what will happen? This is the processor, it is connected to say all the devices through this particular system bus. So, in case of bus master what will happen? We are going to implement one more devices called bus master. So, in that particular case what will happen,

whichever devices is going to give an interrupt first of all, it is going to acquire the bus; that means, that bus will be used by that particular devices only.

So, first of all devices or I/O module is going to acquire a bus, whichever is the I/O module is acquiring the bus only that I/O module or that device can raise the interrupt. So, it is not unique, only one device can raise the interrupt who has got the bus, because through bus master we are going to get the bus actually control of the bus. So, this system bus will be having a control to only one devices and with the help of these things we are going to identify which module is going to give the interrupt. So, these are the ways that we are having.

(Refer Slide Time: 52:35).



Now, how to handle multiple interrupts. So, in that particular case 1 issue is like that we can have several interrupt lines, but this is limited. So, in that particular case what will happen? Now we have to handle this things. Now already I said that when one interrupts come, we are giving a service to one interrupted interrupt service routine at that time another interrupt may come. So, now, what decision you have to take. So, basically for each and every devices we are going to give a priority and generally it says that higher priority device cannot be interrupted by lower priority device.

So, in a bus mastering only current master can interrupt ok. This is also another issue we have, so we are assigning a priority. So, if processor is giving a service to a devices which is having higher priority than that lower priority interrupt will remain pending. So, in that particular case what will happen? This is the processor. So, this is say one interrupt service routine. So,

interrupt may come over here ok. We will see if this interrupt is coming from a higher priority devices then what will happen? Then we are going for the interrupt service routine of that higher priority devices.

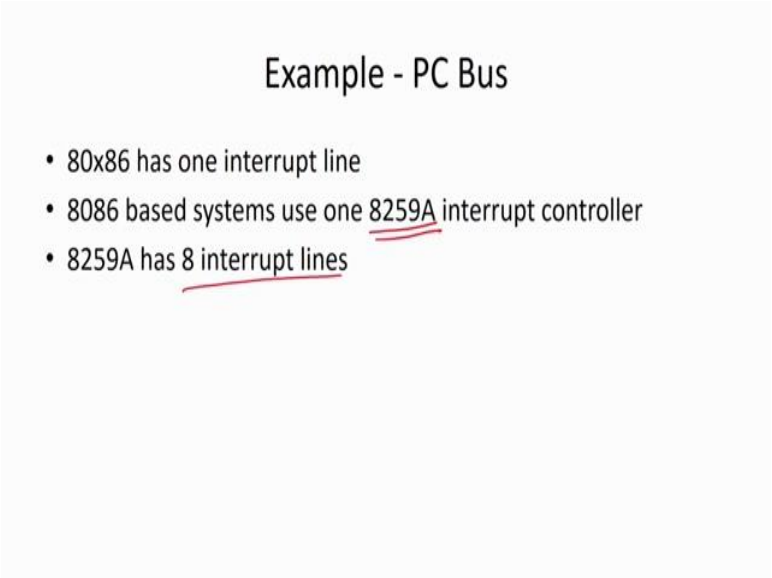
If the interrupt is coming from a lower priority device than what will happen. Then we will come, complete this particular interrupt after that only we are going to give the service to that particular interrupt service routine. So, this is the way we are going to handle it, either immediately you can give it if the priority is higher or otherwise we first complete the interrupt first interrupt service routine first, then only we are going for the next interrupted devices. Now how to issue these things, how to handle this particular interrupt.

So, if we are having multiple line then what will happen? Each line can have a priority, say this is a higher priority line one, this is the lower priority line and this is the least priority line. So, those devices connected to the higher priority line will be always given the preference. On the other hand say if I am having only one line and all the devices are connected to it, then what will happen? In that particular case that we have to give service to the higher priority device.

Now, you consider about a software poll then what will happen? We are going to write the routine, software poll routine in such a way that first it is going to look check the status of the higher priority devices. If it is not interrupting then we will go for the lower priority devices. Similarly in case of your daisy chain, then higher priority devices will be electrically nearer to the processors. So, in this particular diagram you just see that first $M1$ is going to get the interrupts ah, acknowledgment if it is not doing it, then it will give the pass the interrupt acknowledgement to the next module.

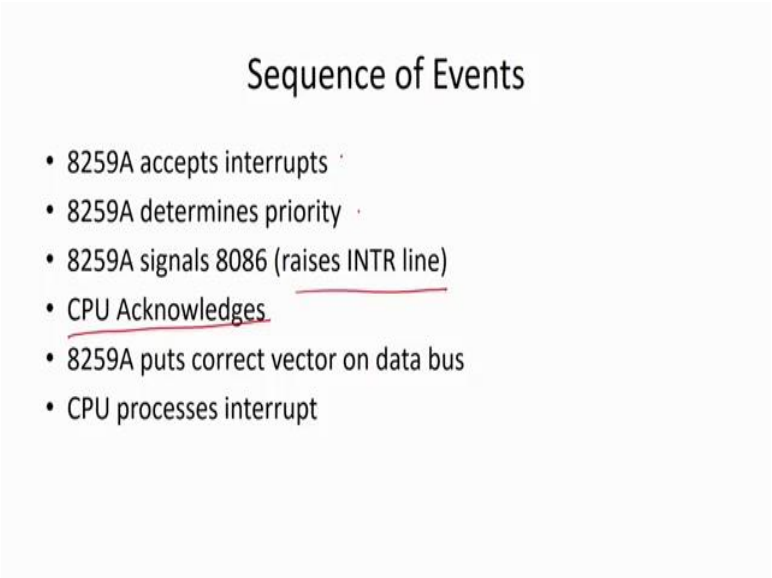
So, higher priority module will be connected first which will be electrically nearer to this particular processor. So, in this particular connection $M1$ is having the highest priority, then $M2$, then $M3$. So, this is the way we can resolve it. So; that means, while connecting the devices we have to resolve the priorities. So, this is the issues that we have while designing an interrupt and while designing the interrupt processor then we have to integrate all those things while designing the processor itself. So, there is one simple example I am giving, say we are talking about the 80x86 family.

(Refer Slide Time: 56:28).



So, when we are having that say 80486, 80386 processor. So, for that processor we are having an interrupt controller. So, 8259A is an interrupt controller. So, through that interrupt controller we can connect 8 lines, it is having 8 interrupt lines; that means, you can connect 8 devices to it ok. Again this particular 8259A can be connected in cascade fashion also. So, if we are having more devices then we can use these things in cascade to incorporate more devices. So, one simple example. So, what is the sequence of events?

(Refer Slide Time: 56:59).

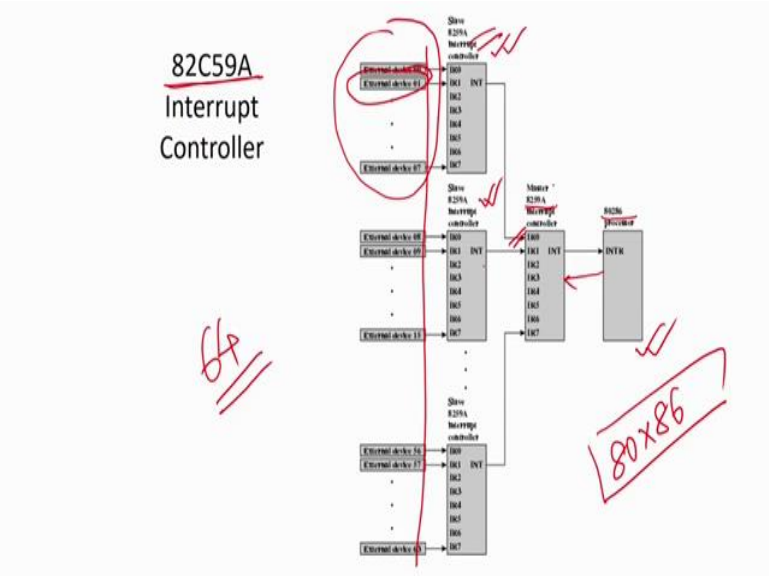


You just see I will show the example also 8259A accepts interrupts ok. So, devices are connected to 8259A, if except the interrupts from the devices 8259A determines the priority. So, again that priority, determination priority is not posed to the processor itself. So, 8259A is going to determine the priority which is having the highest priority which is having the lowest priority like that.

Once it is getting this particular resolving that particular issue, then 8259A signals 8086 through an interrupt line. So, 8086 having an INTR line interrupt line. So, once it identify the devices, once it find out the priority that it can be now service can be given to it, then 8259A give the interrupt signals to 8086 then CPU acknowledges then 8086 acknowledge for it, when it will acknowledge? after completion of the current instruction then 8259A puts the correct vector on the data bus. Now say through 8259A we are connecting 8 different devices or even we can connect more.

So, it will put the correct vector, appropriate vector; that means, every device is having some unique id or say unique address. So, it will put these things, after getting that information now CPU processes the interrupt, now what will happen when it gets the correct vector, then processor will be knowing which interrupt service routine we have to process or we need to execute. It is going to execute that particular interrupt service routine.

(Refer Slide Time: 58:39).



So, this is the way I am saying that we can connect more devices. Now say this is the processor. Now currently we are talking about 80268, it is that 8259A. So, this is working as a master, it

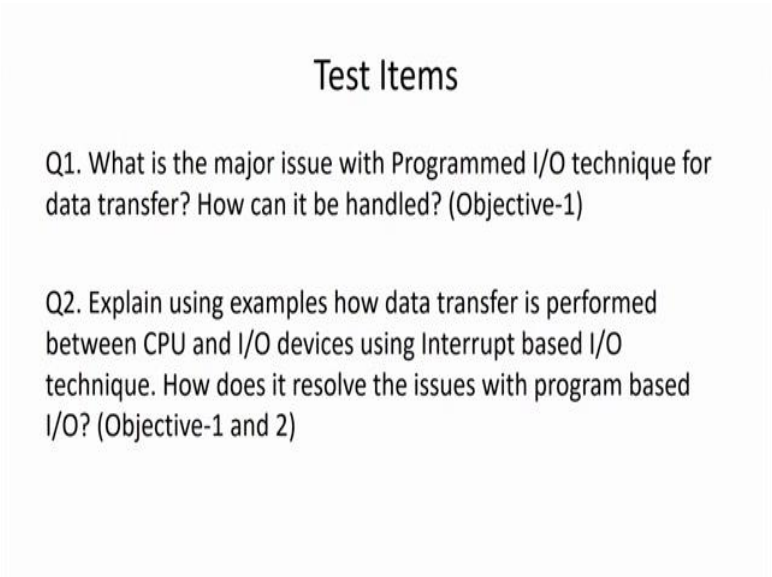
is going to give this interrupt signal. Now, say here I can connect 8 different devices from in IR0 to IR7, but instead of connecting the devices to those particular line, we are connecting another controller.

Now here we are connecting all the devices. So, in that particular case you just see that we can connect 64 devices in this particular arrangement. So, 8 devices in a first ah, when interrupt is coming from this particular interrupt controller. Basically this interrupt is related to device 0 to device 7. Again if it is coming from this particular interrupt then what will happen. Again this is related to some other 8 devices. So, like that we are having 64 devices.

Now, this controller is going to reserve from where it is going to get it. So, if it is going to get it from in the IR0; that means, it is related to those particular device 0 to 7 accordingly it will give the interrupt. When it gets the interrupt acknowledgement then it will put the, this controller will put the appropriate vectors, it will give the appropriate vector to the processor; that means, appropriate address of this particular device who has interrupted it. So, this is the way we are going to connect I/O devices to the processor.

So, it can be cascaded also, so these are specifics. Now we are talking about 80x86. So, to connect the devices in a interrupt mode we have to take help of this particular interrupt controller. For other families for other processor we have to use the corresponding interrupt controller. So, for every processor we are having an interrupt controller. So, this is the way we are connecting and we are doing data transfer with the help of your interrupt driven I/O.

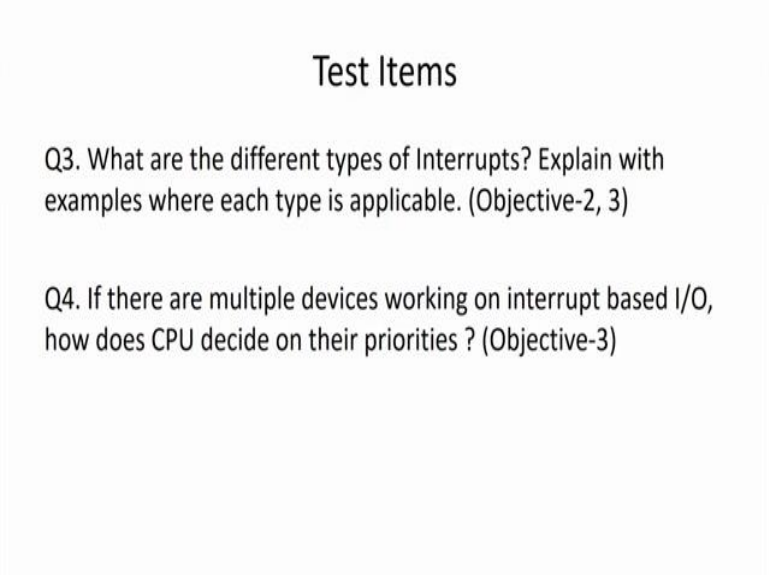
(Refer Slide Time: 60:33)



So, with this now we coming to the end. Now you just see the some test item. So, first test item talking about, saying that what is the major issues with programmed I/O technique for data transfer or how can it be handled? So, this is the objective 1, because we know the problem, what is what is the problem in programmed I/O, to handle it we are coming to interrupt driven I/O. Question 2 explain using examples how data transfer is performed between CPU and I/O devices using interrupt based I/O technique, how does it resolve the issues with program based I/O. So, this is basically objective 1 and 2.

So, basically already we have explained how we are performing the I/O transfer in interrupt driven these things and I think you know what is the difference now with respect to programmed I/O.

(Refer Slide Time: 61:32).



Test item 3, question 3 what are the different types of interrupts, explain with example where each type is applicable. So, basically you just see that what are the different type of interrupts that we may have. So, the, somethings coming from I/O devices itself, something are coming from I/O modules and all those interrupts may have different priorities. So, how we are going to handle it ok. Question 4 if there are multiple devices working on interrupt based I/O, how does CPU decide on their priorities.

So, basically here in 8086, maybe I have to say that priority has been given to your interrupt handler interrupt controller, but on the other hand if it is a daisy chain method, then what will happen, that while connecting the devices we have to resolve it. If we are using your, say

software poll then while writing the software poll routine we have to resolve it. So, these are the issues that we are having, how to handle the priorities and we have to appropriately do it. So, with this I will wind up this particular lecture.

Thank you all.