

## **Computer Organization and Architecture: A Pedagogical Aspect.**

**Prof. Jatindra Kr. Deka**

**Dr. Santosh Biswas**

**Dr. Arnab Sarkar**

**Department of Computer Science & Engineering  
Indian Institute of Technology, Guwahati**

### **Lecture – 34 Interrupt Driven I/O**

Hello everybody, welcome back to the online course on Computer Organization and Architecture. Now we are in a module input output subsystem. So, in our last class we have discussed about the issues related to a input output, why IO module is required and we have seen there are three ways of transfer information; one is your programmed IO, second one is your interrupt driven IO and third one is your DMA. In last class we have briefly discuss about the programmed IO.

Now, in this unit we are going to discuss about the interrupt driven IO. So, what are the objective of this particular unit? So, further I have stated three objective.

(Refer Slide Time: 01:10).

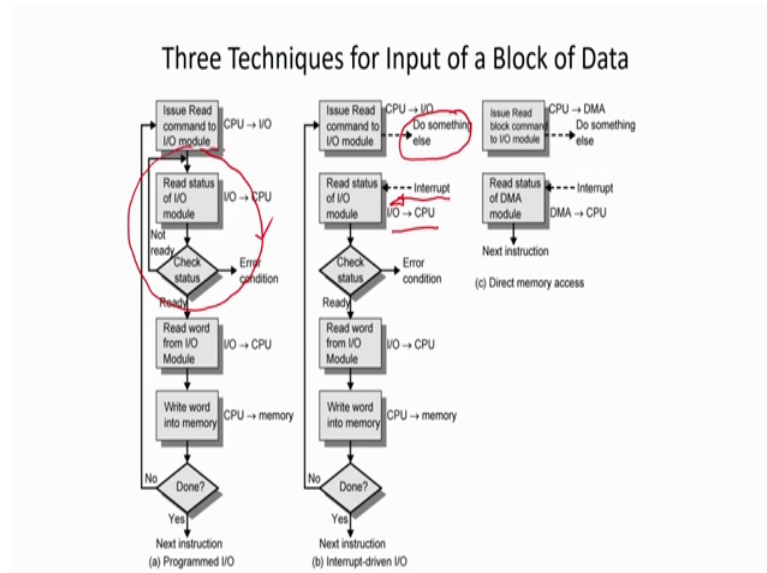
#### **Module: Input/Output Subsystem**

- Unit-2: Interrupt Driven I/O
- Unit Objectives:
  - Objective-1: Discuss the need of Interrupt driven I/O transfer. (Comprehension)
  - Objective-2: Specify the control signal needed for interrupt driven I/O transfer and their use. (Analysis)
  - Objective-3: Explain the design issues of Interrupt driven I/O transfer. (Design)

Objective 1; discuss the need of interrupt driven IO transfer. This will be done in comprehension level. Objective 2 specify the control signal needed for interrupt driven IO transfer and their use. So, it will be in the analysis level and objective 3 explain the design issues of interrupt driven IO transfer; so, it will be in the level design. So,

basically we are going to see how to design an interrupt driven IO. So, already I have mentioned that there are three ways of transfer formation; one is your programmed IO.

(Refer Slide Time: 01:45)



So, in case of programmed IO, already we have discussed that we have some problem with this particular portion that processor is going to set continuously, whether device is ready or not. If it is not ready then it will be in this particular loop and your wastage of time. So, we say that processor read in ideal step doing nothing.

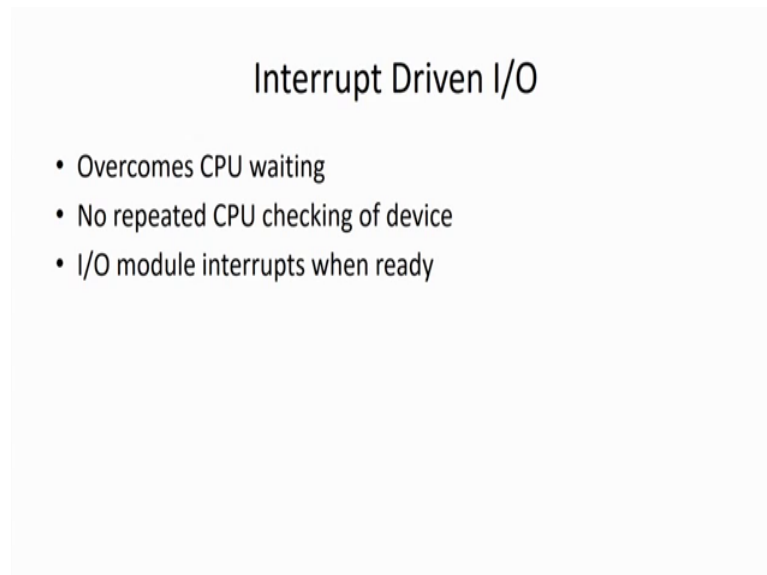
So, one way to look into that particular issue and how we can remove this particular unnecessary waiting, where CPU time is wasted. So, for that from program driven programmed IO we are coming to interrupt driven IO. So, in interrupt driven IO what we are basically doing, we are trying to remove or we have removed this particular busy waiting or idle cycle. So, in that particular case what will happen? Processor will request for IO transfer and after requesting it now, processor can do some other work if really processor than do it, because again there are some issues related to this particular point.

But the processor can carry out some other work, then processor can carry out that particular work, then IO module is going to make the data ready for transfer and; that means, it will look for that status of the device, it will collect the in for level information. And once everything is set, everything is ready then IO module is going to give an interrupt signal to the processor.

It will say that now device is ready. Now we can transfer a information then IO transfer is going to happen. So, in that particular way what happens? We are eliminating this particular waiting states busy waiting step where processor is busy, but doing nothing. So, we can eliminate that and processor cannot do something else, provided it is available processor can do that particular work.

So, once this is there, then the remaining portion is same. We are going to transfer the information from IO devices to processor or from processor to IO devices and finally, we are going to come out. Now you are going to see what are the issues related to designing of this particular interrupt driven IO.

(Refer Slide Time: 03:54).



So, basically I have explained these things overcomes CPU waiting, no repeated CPU checking into device, the way we used to do in our programmed IO and IO modules interrupt when ready. So, when everything is ready then IO module interrupt the processor and says that now we are ready to transfer the information.

So, these are the basic steps that we are having, what are the advantages that we are having, when we are going for interrupt driven IO, eliminating of busy waiting. Now processor is not continuously going to check status of the device, IO module is going to give the indication through interrupt signals.

(Refer Slide Time: 04:32).

### Interrupt Driven I/O Basic Operation

- CPU issues read command
- I/O module gets data from peripheral while CPU does other work
- I/O module interrupts CPU
- CPU requests data
- I/O module transfers data

So, what are the basic operation that we have in case of interrupt driven IO. So, we just say that CPU issues read command. So, here I am saying that read commands, but read commands means processor is going to take information from devices. If processor is going to put information to the output devices then what will happen. Then processor issues write command, IO module gets data from peripheral, while CPU does some other work, know that we are removing their busy waiting of the processor. Now IO module is going to look into the transfer of information, it will get the data from the peripheral devices and in the meantime CPU can carry out some other work.

Io module interrupts CPU. So, when everything is ready, device is ready, IO module has collected the information that to we need to be transferred to the processor and then everything is ready, then IO module interrupt the CPU, then after getting the interrupt request, then CPU request for data. We will say what are the steps that we are having, because processor is going to the look for that particular IO devices, then wherein CPU request the data, then IO modules transfer the data and like that we are going to transfer the information from input device to the processor and similarly the write operation is same, we are going to transfer information from processor to the output devices.

(Refer Slide Time: 05:57).

### CPU Viewpoint

- Issue read command
- Do other work
- Check for interrupt at end of each instruction cycle
- If interrupted:-
  - Save context (registers)
  - Process interrupt
  - Fetch data & store

PSW

Now, from CPU viewpoint then what are the actions that we are going to do, this is read command and process, or is going to a some other work, check the interpret at a end of each instruction cycle. Now we are going to in avoid these things, because we need to look for the end of the instruction ok. Now I think you know how we are going to execute an instruction program is nothing, but a collection of instruction and we are going to execute it instruction by instruction and what we are having, what information we are having with us, we know the address of the next instruction, from where we need to fetch the next instruction.

And we are keeping this particular information in program counter, you just say what to execute one instruction. Basically I said that it may have two stage only fetch and execute or maybe we may have several stages also, because in execution phase we may have different stages; like fetching of data then carryout the operation write the result and like that. So, in all those cases you just see, we are having the information of next instruction only, we do not have those intermediate information where we are currently, what stage we are doing, whether we are fetching a data or we are executing the instruction or carryout the operation or we are writing the result.

So, these things are not available with us. We are having only the information of the next instructs that is why processor is going to look for the interrupt at the end of this particular instruction cycles. So, once complete the instruction is over then only we can

give the service for the interrupted devices, because in between we do not have any checkpoint or any monitoring. We know only starting of the instruction and end of the instruction.

So, instruction cycle must be completed then only you can look for interrupt. Now when we are going to give service to a interrupt then what does it means? We have to perform some operation. Again those operation can be treated as an collection of instruction which is a basically nothing, but again another separate computer program. So, we are going to execute that program and that program is basically known as your interrupt service routine for different devices, we are having different interrupt service routine.

So, basically we are going to stop the execution of the current program after completion of the current instruction and we are going to run the interrupt service routine ok. So, that is why we are going to have point the interrupt service routine. Now I think you can correlate this situation with a function call. I think we have discussed that issues when we are going to discuss about the instruction set design and how we are going to implement it.

So, when we are going to a execute a sub function or a sub routine then what will happen, which temporarily suspend the execution of the main program. When we are going to suspend the execution of the main program, we have to store the processor status and where we are storing the processor status, basically we are storing it in a system stack. So, we keep all those information in the system stacks those that is why we are saying that if interrupted save context and we are going to save context, means basically we are going to set the contents of the register, because those register will be used again by the interrupt service routine if these are general purpose register.

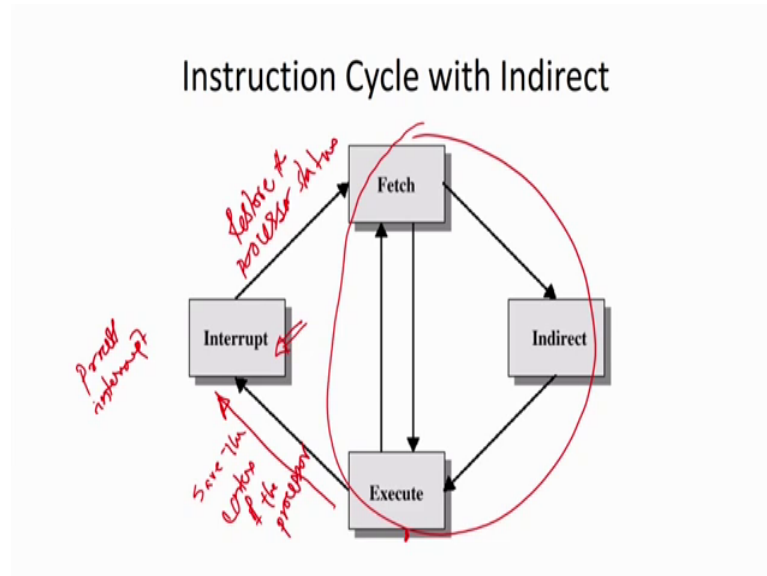
Secondly, we have to store the content of the program counter even, because after completion of the interrupt service routine we have to come back to the current program that we are executing. So, we must know where to come back. So, that is why we are going to store the contents of the program counter example. Again we are going to store the PSW program status word. So, PSW is nothing, but the contents of the flag bits, because if we are performing one operation depending on the result of operation, we may have to take some decision.

But if we will not store it and while executing the interrupt service routine their flags bit maybe stands, it may be disturbed. So, we are not going to get the initial status. So, that is why you need to set this particular PSW, also program status word. So, this is basically a context switching. Now we switching a context from main program to the interrupt service routine. We are showing the context of the main program or the currently executing program in the system stack, then we are going to load the program, then write a starting address of the interrupt service routine and we are going to give the service to the interrupt; that means, we are going to process the interrupt.

So, when we are going to process the interrupt, depending on the interrupt service routine, depending on the nature of the devices, we are going to fetch data or we are going to store it. So, if it is an input devices we are going to fetch it and we are going to keep it in our memory. If it is an output device then we are going to take the information from memory and going to put it in the output device. So, process interrupt means, basically transferring the information. Once the transfer is over then what will happen.

again we will come back to the main program, but a program from where we have given the interrupt or giving the service to the interrupted devices. So, again we have to restore the information; that means, again we are going to bring the information from system state to the processor; that means, we are going to restore all the registers value. We are going to restore the program status word and along with that we are going to bring the program counter values also, then we will be knowing from where we need to start the execution. So, from CPU view point we are going to say that these are the parts that we need to do, when we are going to give a service for interrupt.

(Refer Slide Time: 11:48).



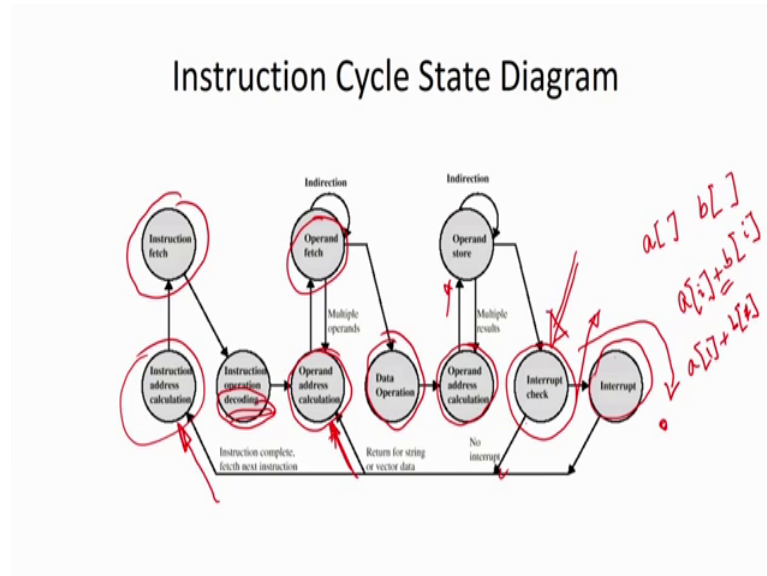
So, this is basically I have already explained that this is fetches, then execute and execute map several stages or. Secondly, in some other instruction it needs indirect cycle, because we need to fetch the instruction. So, while we are performing one operation we are basically doing this cycle. So, when executing a program if interrupt arrives so we are taking an interrupt from the interrupted devices or for IO module. Then what will happen after completion of the instruction. Before fetching the next instruction we will go for the interrupted devices or we are going to execute the interrupt service routine.

At that particular point first of all we have to retain the processor status, we are going to pose all the information to the system stack, then we are going to execute the interrupt service routine to give the service for that interrupt and after completion of that particular interrupt service routine; that means, when data transfer is over, then we are going to restore back the processor status; that means, we are going to pick up the information from system stack and going to put into the relevant registers.

So, these are the things that we are going to do basically; one is your. So, basically it is a context switching ah, basically is save the context of the processor after saving a context, then process the interrupt, then we have to run the interrupt service routine, then after completion of these things restore the processor status; that means, we are going to get back the processor status from system stack and we are going to restore everything in the appropriate register. So, this is the, in indirect cycle, it indirect as well as with interrupt.



(Refer Slide Time: 13:40).



Now, that what are the state diagram, we may have a slightly elaborate diagram. So, in that particular case what will happen. When we are going to execute a program or execute an instruction, first we have to get the instruction address calculation, first they have to say where from we have to fetch the instruction. So, this is basically nothing, but we are going to get the information from program counter, then we are going to fetch the instruction. After fetching the instruction we have to decode the instruction, after decoding the instruction we will be knowing whether there is any indirect cycle or not basically, whether we have to fetch some more data's or not.

So, even indirect cycle is there then what will happen, we have to calculate the operand address, fetch the operands and if we need to fetch more operand then it will be in this particular loop and after completion of this particular indirection, then we will go for the operation data operation. We perform the data operation and finally, when we get the result, then we can sometimes we need to perform the operand address calculation; that means, in which memory location, we are going to store our result. So, after getting this particular address, we are going to store the result or store the operand.

And if we need to store more data then it will be in this particular flow, because if sometimes we may work with the vector data also. So, one operand store is over then the next, it is the completion of the instruction, then we are coming to this particular step. In this particular step what will happen. We are going to check whether any interrupts are

pending or not. If interrupts are pending then what will happen. We will go over here, we will give the service to the interrupted devices and if no interrupt is coming then straight away we will coming over here and we will go for next address calculation.

But in between I am having one more arrow. So, in that particular case, this is basically if we are working with the vector data or maybe saying a array, you just see that if I am going to add two arrays, then what will happen. We are talking about the same operation addition only that was a difference. So, I am having an array a and array b, if I am performing say  $a_i$  plus  $b_i$  then what will happen. This operation is same, I am going to perform this addition operation. So, after performing  $a_1$  plus  $b_1$  then what will happen? Instead of going for, going to look next instruction, new instruction what will happen? We know that we have to perform this instruction itself.

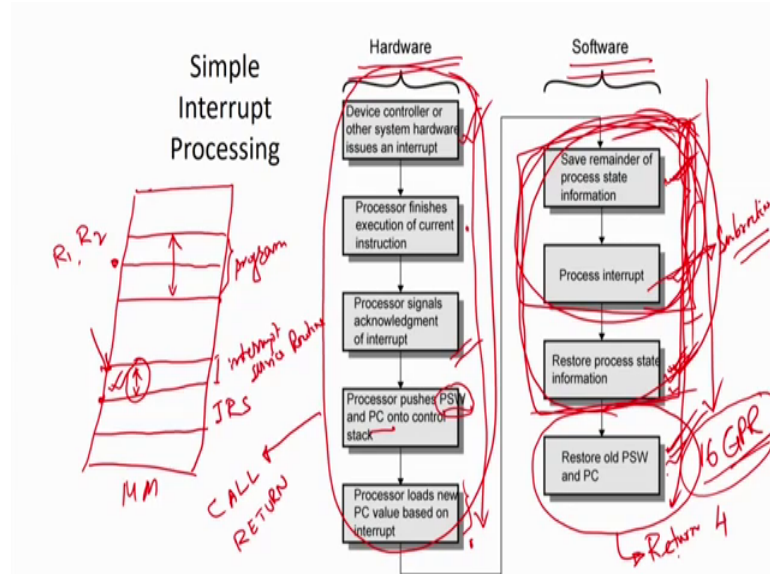
So, you have to only get the data. So, we are straightway coming to operand address calculation; that means, after 1 we will go to  $a_2$  and  $b_2$ ,  $a_3$  and  $b_3$ . So, this is basically single instruction multiple data. If we are going to perform the same operation on multiple data, then we may design the processor in such a way that instead of fetching the instruction straightaway, you can go for fetching the data, you just see what happens after completion of one instruction we are going to check for the interrupt, if there is no interrupt then what will happen? We will go back to the current program execution.

But if any interrupt is pending then what will happen? We will go for this particular interrupt step; that means, we are going to give service to the interrupted devices in that particular case, we are going to save the status of the processor, we give the service to the interrupted devices, then we are going to restore the status of the processor and we come back to the program itself. So, we know that once we restore the status we know from where we have to fetch the next instruction and accordingly we are going to execute the program itself.

So, this is the complete instruction cycle state diagram. So, these are the states ah, these are the states that we have during instruction education. So, this is the elaborate view, some of the instruction may not have all the states, but some of the instruction may have all the states. It depends on the type of the instruction and after decoding the instruction we will be knowing what are the state that we have for execution of this particular

instruction. This is the complete instruction cycle state diagram and these are the step that we have when we are going to execute an instruction.

(Refer Slide Time: 18:08).



Now, already I have mentioned that how we are going to give the service to the interrupt. This is very much similar to the function call and return. So, what basically we are doing. So, basically what are the steps we are having device controller or other system hardware issues an interrupt. So, you are getting an interrupt processor is getting an interrupt. Now say processor finishes the execution of the current instruction. Now we must complete the current instruction, because we do not have any mechanism to keep the information that what stage we are executing currently.

So, that information that such several information we cannot returning that side, you are going to complete the current instruction after completion of the current instruction. We know from which memory location we need to fetch the next instruction. So, at that particular point you can stop it. So, after completion of the current instruction what will happen? Processor signals acknowledgement of interrupt. Now processor give an acknowledgment signal to the interrupted devices or to the interrupted IO module, which says that now processor is free to give the service to the interrupted devices.

Now, processors stall at that time. So, at that particular time processor process the PSW and PC on to the control stack, already I have mentioned that. We are having a control stack, what are the basic information that I need to store; one is your program counter,

because it will give me the information from where I need to fetch the next instruction of the current executing program. So, basically we are going to give a service to the interrupted devices we have to retaining.

So, you are going to retain this particular information we are going to push it to the system stack along with that. We are going to store that PSW also program status word, already I have mentioned that these are nothing, but this is nothing, but the collection of all flag bits and the basically the flag bit is set or reset depending on the ALU operation. So, we have to (Refer Time: 20:12) thing, because when we are going to come back to the main program then what will happens depending on the status, we may have to some text, some other decision. So, we are going to stack.

then processor loads the new PC value, based on the interrupt. Now I am saying that for every interrupt we are having an interrupt service routine. So, basically now we see if this is my main memory and say currently we are executing this particular program, depending on the these things that service routine say that maybe your, this may be 1, this may be your another interrupt service routine. So, we must know the starting address of this particular service routine.

So, we are going to load the value of the program counter with the starting address of this particular interrupt service routine; that means, we are setting which program we need to execute. Now processor is going to execute this particular program after completion of this particular program. We will come back to this particular point wherever say your interrupt is coming at that particular point. So, we will come back to this particular point. So, we are setting it then along with that, before going to execute these things we may have to some more information.

So, save the remainder of the processor state information. So, what are the processor state information, we are say in this particular program. When we are executing this particular program, we might have used some general purpose register like that R 1 and R 2 we are having some valid result. So, when I am going to execute this particular subroutine, then what will happen? Again we have to use those particular temporal stories those registers. So, in that particular case what will happen that values will be overwritten by this particular sub register routine.

So, we have to store that information also. So, after storing this information, now process the interrupt. Now we are going to execute this particular service routine and once you complete it, completion of this particular service routine, then we have to come back to that particular point. So, this is your restore the processor step; that means, we are restoring and we are coming back to that particular point. So, restoring means getting the PSW, getting the your program counter and getting the values of all the registers and we are now coming to the initial position and from that we are going to execute it.

So, this is the processor step, basically these are nothing, but the contents of the general purpose register we are going to restore it, because we have to restore it in this particular order only, because we are using a system stack, this is push and pop and stack is your last in first out, whatever we have interrupt last we have to pop out first, then I am going to put it in the appropriate register, then we are going to pop out program counter and PSW and we are going to store it.

Now you just see that here it is mentioned that this is something is written as hardware and it is software. So; that means, some portion we are going to do in the hardware level; that means, when we are going to design the instruction we are going to put everything into the processor itself.

So, up to this point we are doing it in the hardware level, again this particular point, basically, though it is written over here, you can say that this is basically we are going to do into the software level, what where inside the remainder of the state information basically that storing the general purpose register value. Now why you have shifted to the software side, say if you think that we are having a processor which say 16 general purpose register ok, in that particular case what will happen? All the 16 may be used in a main program. Again we may require all the 16 processor may be in the subroutine, also we do not know, we cannot predict anything.

So, if we are going to do everything in hardware, then what will happen we have to store the values of those particular 16 general purpose register also interrupt state; that means, we have to close all those values of the 16 register to the stack and after completion of these things, when we are going to use this particular restore it, then we have to again bring all those things to the processor. So, it is basically a very heavy instruction,

because we have to do lot of work, but many a time you may find that out of 16 general purpose register, hardly we may be using say 4 registers in this particular program.

So; that means, the values of those 4 registers are relevant, other 12 registers are irrelevant. So, what we basically do, if we are going to do it in the software level then we can check that information also, how many registers are active in this particular program segment and depending on that only we are going to store those particular register only. So, it will be reduced also, so that is why this portion when we are going to implement this portion.

Generally we are not implementing in your hardware, we generally keep it for the programmer when programmer is going to use like such type of service routine, depending on the situation in this particular interrupt service routine programmer, programmer is going to store those particular register value into the stack and after that it is going to perform the interrupt service routine.

And after completion of the interrupt service routine, since that interrupt service routine knows that it has got 4 registers fairly to the stack, it is going to pop out those particular 4 register values and store it into the processor register. So, after that that execution of this particular interrupt service routine is over, then again these things will be done in our hardware level. So, this is also will be done in your software. So, this will be done in our hardware level; that means, again we are going to restore the processor status work and program control ok.

This is the word that we can say or you can say that again, maybe in program itself we can write it, if we do not have any other instruction. So, these are the portion that we are doing in the hardware level. So, this is, you just see that in interrupt we are implementing something in the hardware and we are implementing something into the software, just to have a balance. If we are going to do everything in a hardware then what will happen? This particular interrupt handling situation will be a complex one and it will be a an heavy instructions. So, that is why something we are doing into the hardware and something we are doing into the software.

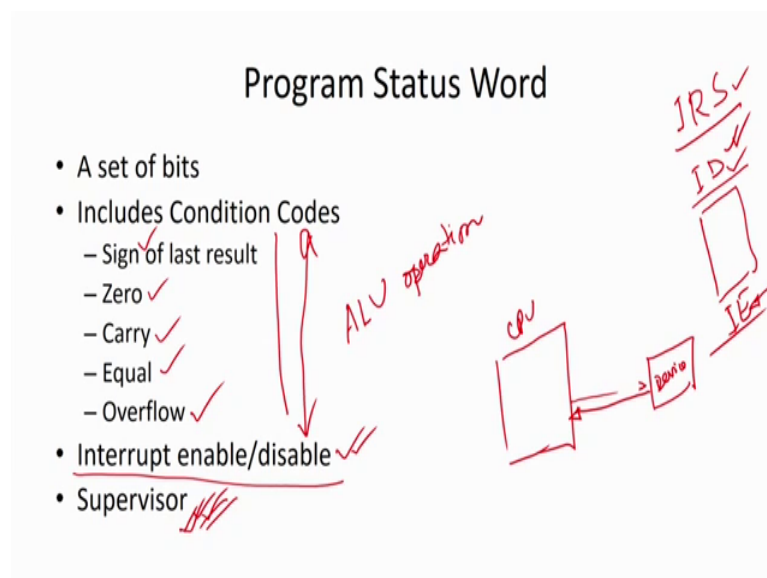
So, you just see that I think similar scenario, we are doing in the function call or subroutine call. So, in that particular case what will happen, this person here instead of process of interrupt, this is basically your subroutine process, execute the subroutine and

this entire portion will be done into the software that storing the values of general purpose register and restoring the general purpose register, after completion of this particular subroutine and this portion will be done with another instruction which is known as a return instruction and now instruction set.

So, for subroutine call we are having one instruction for call, the calling the inform a subroutine and one instruction is return to return from the subroutine. So, in that particular case, this is the portion that we are going to do it in call, this portion will be done in the software and this portion will be done in the return instruction. So, the necessary same execution of interrupt and execution of sub routine, but for your subroutine call, we are calling it with the instruction call and returning it form with the instruction returned, but in case of interrupt with an control signal, we are initiating this process and after completion of this things we will go back to this particular point.

So, this is the simple inter processing and these are attacks we need to do while performing the interrupt. Now what is the program status here? Already I have mentioned that these are nothing, but a set of bits and this set of bits basically includes result of the last instruction that we have executed on, may be your sign bit, 0 bit, carry bit, equal bit and overflow bit.

(Refer Slide Time: 29:11).



I think we have mention 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 already affected by the result of an ALU, but along with that we are having some fraction. So, one of the flag bits is your interrupt enable and disable. So, in that particular case 1, 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 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 another signal call, say acknowledgment signal.

Now, what will happen if whenever you are doing, if some devices is going to interrupt. We are bound to give the service to the interrupted devices, but the processor is engaged in some important work, high priority work, just say in case of your distance, say when we are looking for the air cup 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 the, during that period something else may happen. So, in that particular case what will happen. We may have a provision to say whether I 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 dispatches can interrupt the processor, if we set it to interrupt disable then what will happen?.

We are setting in, we are disabling the interrupt and in that particular case what will happen if interrupt comes, then processor is not going to give the surface to the interrupted devices, it will first complete the current program, after that only it will look for that particular interrupt process. 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 and your pass the instruction itself 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 a interrupt service routine.

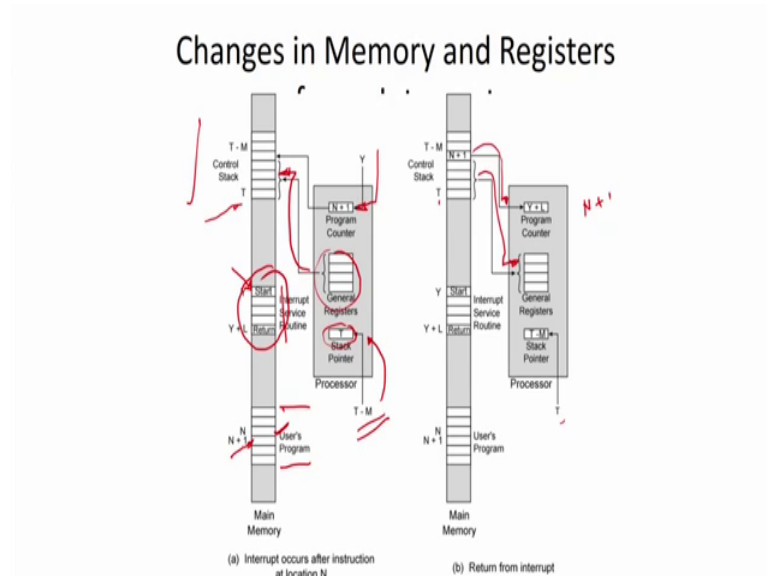
So, while you are going to write your program also, when you are going to develop the software if you fill, that is if you think that processor should not be interrupted while executing this particular program. So, at the beginning you have, can disable it, but before completion, before coming out from 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 the 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 route user, all of you know about it.

So, if you are having a privilege of group 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, further 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 route, generally we set it as a supervisor mode.

So, when you login as a root then what will happen. You are having either supervisor mode. Now we can sense many more system parameter, but if you login as an 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 send 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 simple example, just say how, what will happen when we are going to perform the interrupt service routine, so its a set. This is my user program ok. Currently the values of program counter is N plus 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, that before going to start that interrupt service routine what it needs to do. So, basically it shows like that at that time that is value of the stack pointer is T; that means, this is my stack. So, stack pointer is pointing at that particular point. So, the, 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 minus M. So, this T minus M is put into the stack pointer, because after completion of the service routine we have to pop up from this particular point T minus 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 plus 1. 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 minus 1, all the registers value that we have stored we again, we pop it out and in accordingly it is going to push it over here.

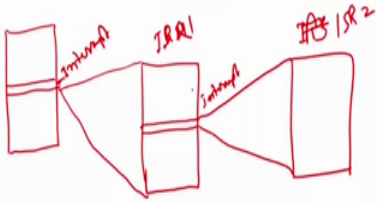
Now accordingly that is these values will be reduced. Now top of the stack becomes T. Now we are going to put this particular T to over here. So, when I am bringing this information, you just see now that program counter value is becoming N plus 1. Now, we are going to fetch the instruction that is in this memory location N plus 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).

### Design Issues

- How do you identify the module issuing the interrupt?
- How do you deal with multiple interrupts?
  - i.e. an interrupt handler being interrupted



How do we identify the module using the interrupt, because already I am saying that there maybe several IO module. In every IO module we may connect several devices. So, we are going to worked it a particular device, it will come to a particular IO 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 once has some more interrupt is coming over here; say 1. So, another interrupt is coming over here.

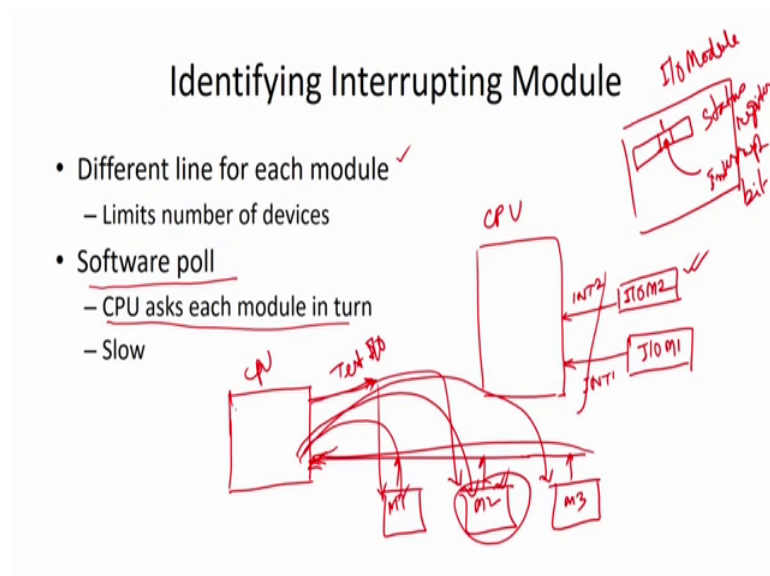
So, now what to do, whether can I go for interrupt service routine to, sorry interrupt service routine to whether I should execute this particular pro interrupt, then only you should go for this thing. So, these are the issues that we are having. So, how we are going to handle this particular interrupt. 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 check with the interrupt ok.

So, we will see this 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 discuss about the programmed IO and you have seen that how we are going to

connect the IO devices, we are having two way to do it; one is your memory mapped IO and second one is your isolated IO. So, for all the modules we are having, connect this, but if so while 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 IO module

(Refer Slide Time: 40:53).



So, there are different ways to do it to write, define. I did not define 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 IO 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 IO module 1.

So, similarly I can have another interrupt line where I am connecting, say IO 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 IO module 2. So, in that way we can resolve this issue, but how many lines you are going to provide, we do not 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 their number of lines. you 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 an routine, software routine to identify the IO module which has given this particular interrupt you just say.

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 is 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 works 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 IO 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 in. So, I am saying that in software poll is going to run an 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, its IO module to determine which module cause 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 is 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).

## Identifying Interrupting Module

- Software poll
  - CPU branches to an interrupt service routine
  - Poll each I/O module to determine which module caused the interrupt
- Can be done by separate command line, TESTI/O
  - The processor raises TESTI/O and places the address
  - The I/O module responds positively if it set the interrupt
- Alternatively, by an addressable status register
  - The processor reads the status register to identify the interrupting module

So, it is going to poll each and every module IO 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 given the interrupt, accordingly it will place the addresses of this particular module, I am 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 IO. So, we may have one command line or one control line through that particular control line, it is going to give test IO. So, the processor raises the test IO and puts the addresses.

So, what will happen we can have a control signal call test IO and it will raise these things and along with that, it will give the address of this particular IO 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 and it is not going to response and not done it, then what will happen. This signal will go to the next module along with that, prepare the 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 IO.

The processor raises the test IO and places the address the IO module respond positively if said the interrupt. So, this is one way, second way is done it by addressable status

transistor. So, basically what will happen? Already I said that for IO module, if I am having an IO 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.

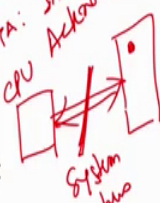
So, when this IO module will set this particular interrupt, then what will happen. It will set this particular bit to 1; that means, it will indicate that interrupt has been raised by this particular module. So, in that particular case what will happen? This processor is going to check this particular 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 interpreted devices.

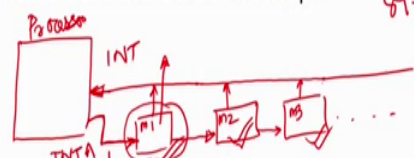
Now, it is going to set this particular bit if this is set, then it will be a processor will be knowing that this module has given the interrupt and accordingly processors will set the address of this module. I am going to carry out the work if it is 0, then this is going to check the next module and accordingly it will go this way. So, it is going from module to a module, just to check it whether who has raise 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 IO module has raise 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).

### Identifying Interrupting Module

- Daisy Chain or Hardware poll
  - Interrupt Acknowledge sent down a chain
  - Module responsible places vector on bus
  - CPU uses vector to identify handler routine
- Bus Master
  - Module must claim the bus before it can raise interrupt
  - e.g. PCI & SCSI

*JINT : Interrupt*  
*INTA : Interrupt Acknowledge*  






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 do not 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, 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 call interrupt acknowledgement. So, basically this is the terminology we use INT basically look for, says that this is the line interrupt line and INPI is your interrupt acknowledgment ok. So, these are the two line. So, now, one 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 us.

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 raise the interrupt. If module 1 has not raise the interpret then what will happen? It will pass this interpret acknowledgement to the next devices. So, now, next devices also act accordingly. If it has raise the interrupt then it will give the indication to the processor along with his address. If it has not devices, then it will pass the interrupt acknowledgement to M 3.

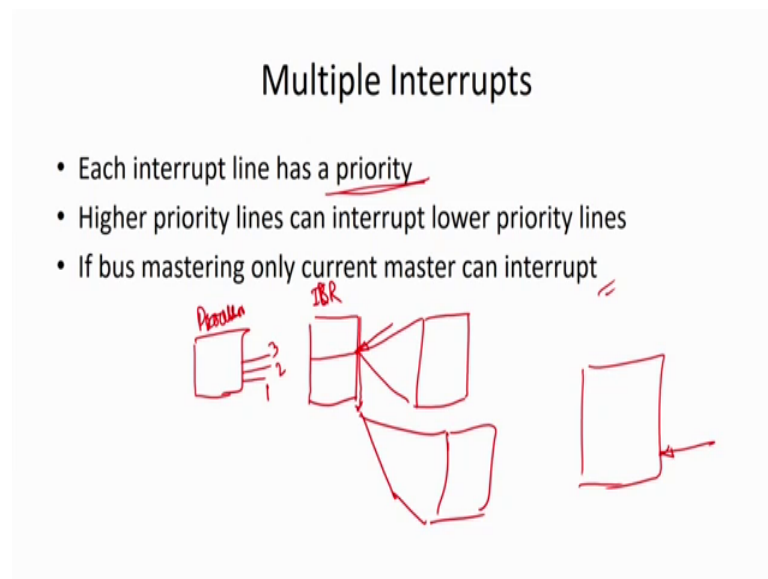
So, this is you just see that we are implementing these things in the hardware. So, that is why you are talking about it. It is the hardware poll or the name given is your decision daisy chain, because devices are connected in a same fashion one after another in once in itself. So, this is the things that within the interrupt acknowledgment sent down the chain. So, this is the chain. So, it is sent down 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 acquired a bus; that means, that bus will be used by that particular devices only.

So, first of all devices or IO module is going to acquire a bus, whichever is the IO module is acquiring the bus only that IO 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 the 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 given 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 interrupts 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 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 orders 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 m 1 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 m 1 is having the highest priority,

then m 2, then m 3. 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 80 x 86 family.

(Refer Slide Time: 56:28).

**Example - PC Bus**

- 80x86 has one interrupt line
- 8086 based systems use one 8259A interrupt controller
- 8259A has 8 interrupt lines

So, when we are having that say 8048 6803 86 processor. So, for that processor we are having an interrupt line controller. So, 8259 a is a 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 8259 a 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).

## Sequence of Events

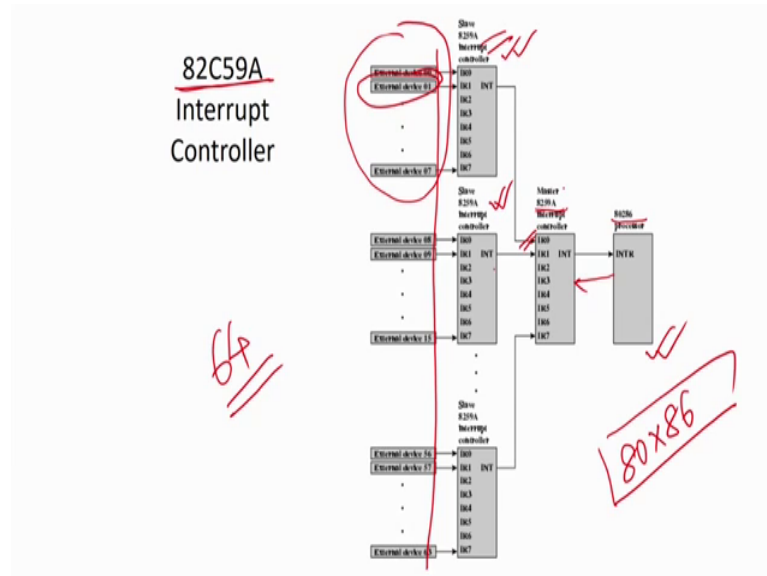
- 8259A accepts interrupts
- 8259A determines priority
- 8259A signals 8086 (raises INTR line)
- CPU Acknowledges
- 8259A puts correct vector on data bus
- CPU processes interrupt

You just say I will show the example also 8259 A except interrupts ok. So, devices are connected to 8259 A, if except the interrupts from the devices 8259 say determines the priority. So, again that priority, determination priority is not posed to the processor itself. So, 8259 A 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 8259 A 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 8259 A 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 8259 A puts the correct vector on the data bus. Now say through 8259 A we are connecting 8 different devices or even we can connect more.

So, it will put the correct vector, appropriate vector; that means, I have two devices having some unique id or say unique address. So, it will put these things after getting that information. Now say if you process the interrupt, now what will happen when it catch 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 82598. 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 IR 0 to IR 7, 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 divide 0 to device 7. Again if it is coming from this particular interrupt then what will happen. Again this related to some other 8 devices. So, like that we are having 64 devices.

Now, this controller is going to result from where it is going to get it. So, if it is going to get it from in the IR 0; that means, it is related to those particular device 0 to 7 accordingly, it will give the interrupt. When it gains the interrupt acknowledgement then it will put down, 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 IO devices to the processor.

So, it can be cascaded also, so these are specifics. Now we are talking about 80 x 86. 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 the transfer with the help of your interrupt driven io

(Refer Slide Time: 60:33)

### Test Items

Q1. What is the major issue with Programmed I/O technique for data transfer? How can it be handled? (Objective-1)

Q2. 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? (Objective-1 and 2)

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 program IO technique for data transfer or can it be handle. So, this is the objective 1, because we know the problem, what is what is a problem in program IO to handle it. We are coming to interrupt driven IO. Question 2 explain using examples how data transfer is performed between CPU and IO devices using interrupt based IO technique, how does it resolve the issues with program based IO. So, this is basically objective 1 and 2.

So, basically already we have explained how you are performing the IO transfer in interpret driven, these things and I think you know what is the difference.

(Refer Slide Time: 61:32).

### Test Items

Q3. What are the different types of Interrupts? Explain with examples where each type is applicable. (Objective-2, 3)

Q4. If there are multiple devices working on interrupt based I/O, how does CPU decide on their priorities ? (Objective-3)

Now with respect to program IO 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 difference type of interrupts that we may have. So, the, something with coming from IO devices itself, something are coming from IO modules and all those interrupts may have different priorities. So, how we are going to handles ok. Question 4 if there are multiple devices working on interrupt based IO, how does CPU decide on their priorities.

So, basically here in 8086, maybe I have say that priority has been given to your interrupt analog 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 a priorities and we have to appropriately doing. So, with this I will wind up this particular lecture.

Thank you all.