Computer Organization and Architecture: A Pedagogical Aspect.

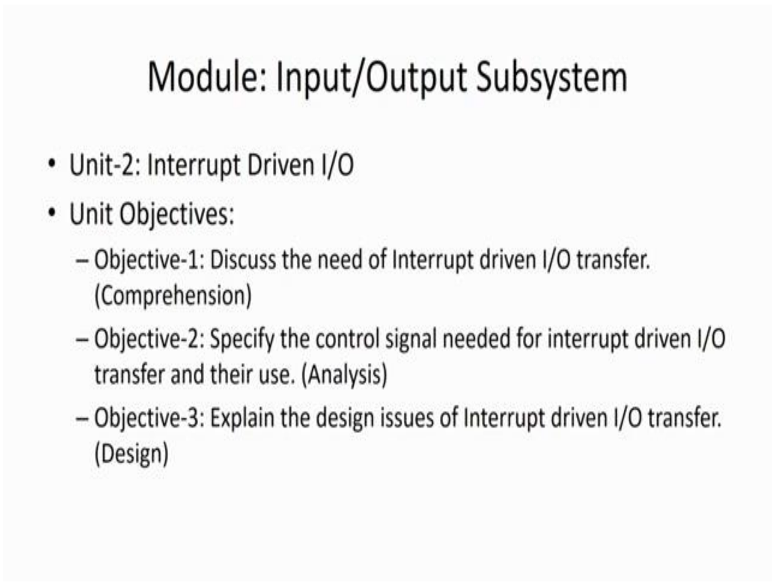
Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

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 input output, why I/O module is required and we have seen there are three ways of transfer information; one is your programmed I/O, second one is your interrupt driven I/O and third one is your DMA. In last class we have briefly discuss about the programmed I/O.

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

(Refer Slide Time: 01:10).

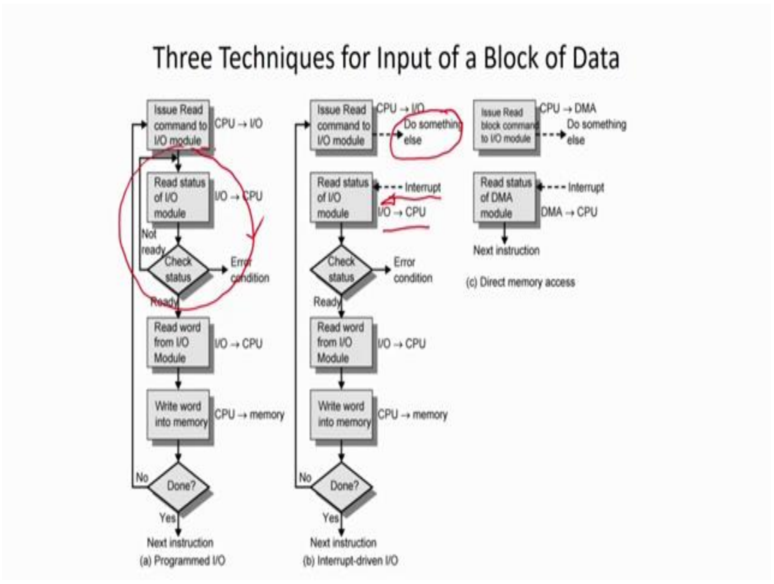


Module: Input/Output Subsystem

- 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 I/O transfer. This will be done in comprehension level. Objective 2: specify the control signal needed for interrupt driven I/O transfer and their use. So, it will be in the analysis level and objective 3: explain the design issues of interrupt driven I/O transfer; so, it will be in the level design. So, basically we are going to see how to design an interrupt driven I/O. So, already I have mentioned that there are three ways of transfer information one is your programmed I/O.

(Refer Slide Time: 01:45)



So, in case of programmed I/O, already we have discussed that we have some problem with this particular portion that processor is going to check 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 is in ideal state doing nothing.

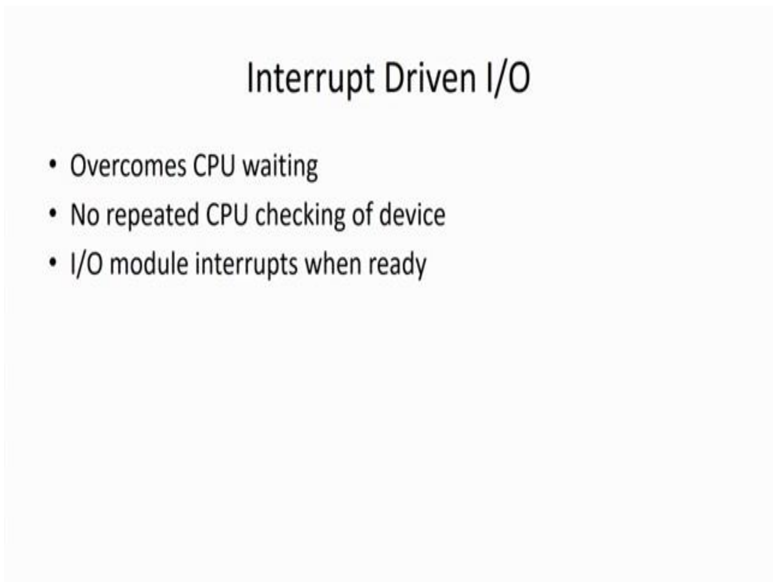
So, one way to look into that particular issue and how we can remove this particular unnecessary waiting, where CPU time is wasted. So, for that from program driven programmed I/O we are coming to interrupt driven I/O. So, in interrupt driven I/O what we are basically doing, we are trying to remove or we have removed this particular busy waiting or idle cycle. So, in that particular case what will happen? Processor will request for I/O transfer and after requesting it now, processor can do some other work if really processor can 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 I/O 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 information. And once everything is set, everything is ready then I/O module is going to give an interrupt signal to the processor.

It will say that now device is ready. Now we can transfer information then I/O transfer is going to happen. So, in that particular way what happens? We are eliminating this particular waiting states busy waiting state where processor is busy, but doing nothing. So, we can eliminate that and processor can now 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 I/O devices to processor or from processor to I/O devices and finally then we are going to come out. Now you are going to see what are the issues related to designing of this particular interrupt driven I/O.

(Refer Slide Time: 03:54).



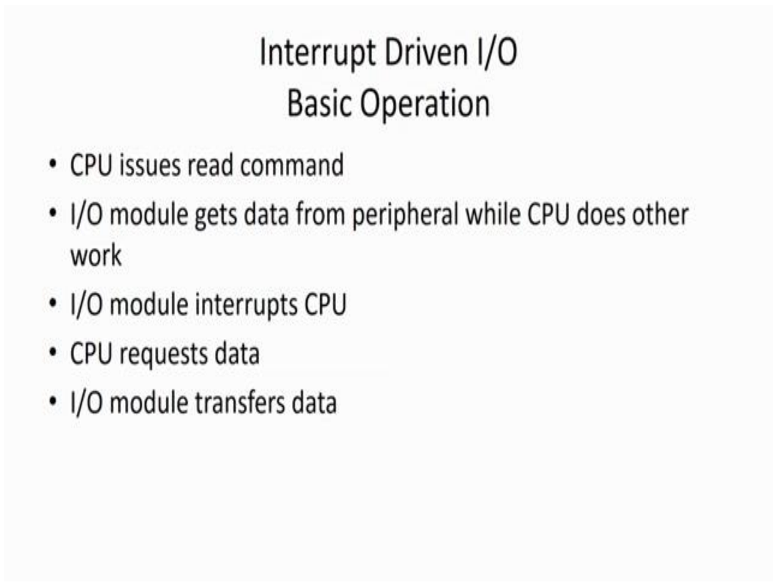
Interrupt Driven I/O

- Overcomes CPU waiting
- No repeated CPU checking of device
- I/O module interrupts when ready

So, basically I have explained these things overcomes CPU waiting, no repeated CPU checking to device, the way we used to do in our programmed I/O and I/O modules interrupt when ready. So, when everything is ready then I/O 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 I/O, eliminating of busy waiting. Now processor is not continuously going to check status of the device, I/O module is going to give the indication through interrupt signals.

(Refer Slide Time: 04:32).



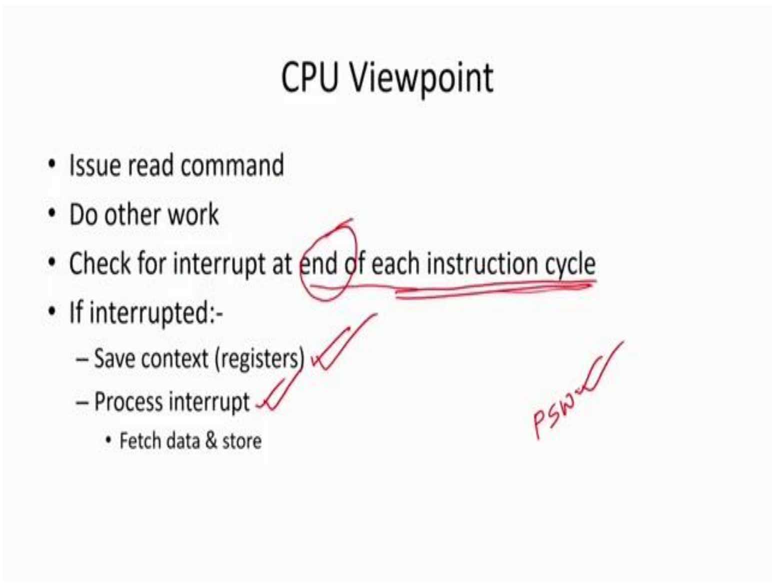
Interrupt Driven I/O
Basic Operation

- 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 I/O. 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, I/O module gets data from peripheral, while CPU does some other work, now that we are removing their busy waiting of the processor. Now I/O 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.

I/O module interrupts CPU. So, when everything is ready, device is ready, I/O module has collected the information that to be need to be transferred to the processor and then everything is ready, then I/O module interrupts the CPU, then after getting the interrupt request, then CPU requests for data. We will see what are the steps that we are having, because processor is going to now look for that particular I/O devices, then when CPU requests the data, then I/O 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

Now, from CPU viewpoint then what are the actions that we are going to do, issues read command and processor is going to some other work, check the interrupt at the end of each instruction cycle. Now we are going to elaborate 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 see 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 carry out 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 don’t 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 carry out 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 instruction that’s why processor is going to look for the interrupt at the end of this particular instruction cycle. So, once complete the instruction is over then only we can give the service for the interrupted devices, because in between we don’t 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 the interrupt then what does it means? We have to perform some operation. Again those operation can be treated as a collection of instructions 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's why we are going to have run 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.

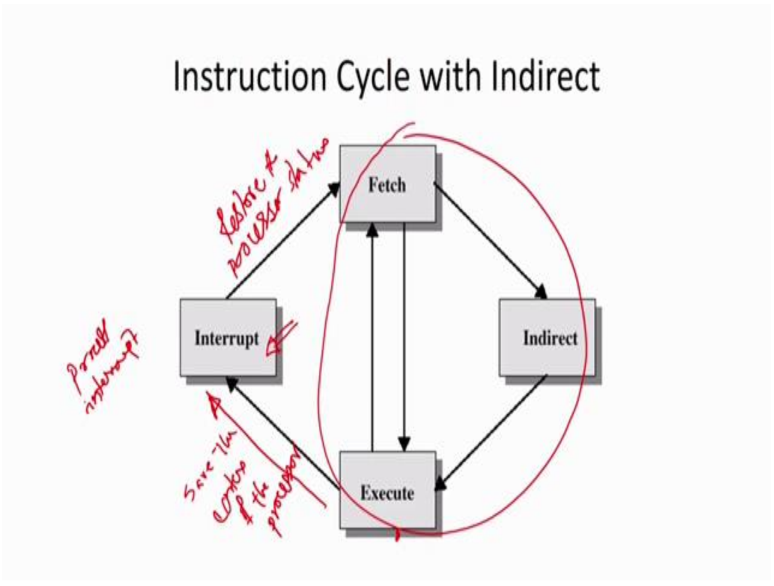
So, when we are going to execute a sub function or a sub routine then what will happen, we temporarily suspend the execution of the main program. When we are going to suspend the execution of the main program, we have to store the processor status and where we are storing the processor status, basically we are storing it in a system stack. So, we keep all those information in the system stacks those that's 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's why we are going to store the contents of the program counter even. 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 won't store it and while executing the interrupt service routine their flags bit maybe changed, it may be disturbed. So, we are not going to get the initial status. So, that's why you need to save this particular PSW also, program status word. So, this is basically a context switching. We are switching the context from main program to the interrupt service routine. We are saving the context of the main program or the currently executing program in the system stack, then we are going to load the program counter with the 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, a program from where we have given the interrupt or given 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 viewpoint 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).



Instruction Cycle with Indirect

Fetch
Indirect
Execute
Interrupt
Restore the processor status
Process interrupt
Save the status of the processor

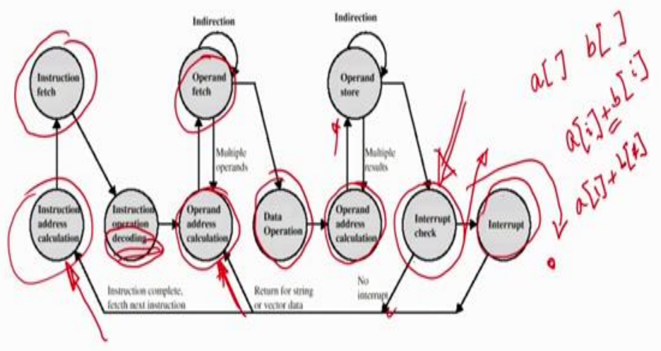
So, this is basically I have already explained that this is fetch, then execute and execute may have several stages or. Secondly, in some other instruction it needs indirect cycle, because we need to fetch the instruction. So, while we are performing one operation we are basically doing this cycle. So, when executing a program if interrupt arrives so we are getting an interrupt from the interrupted devices or for I/O 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 post 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, indirect as well as with interrupt.

(Refer Slide Time: 13:40).

Instruction Cycle State Diagram


Indirection
Indirection
Instruction fetch
Operand fetch
Operand store
Multiple operands
Multiple results
Instruction address calculation
Instruction operation decoding
Operand address calculation
Data Operation
Operand address calculation
Interrupt check
Interrupt
Instruction complete, fetch next instruction
Return for string or vector data
No interrupt
a[ ] b[ ]
a[i]+b[i]
a[i]+b[i]

Now, let 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 fancy fetching the instruction we have to decode the instruction, after decoding the instruction we will be knowing whether is there any indirect cycle or not basically, whether we have to fetch some more data's or not.

So, when 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 ok 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, 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 be coming over here and we will go for next address calculation.

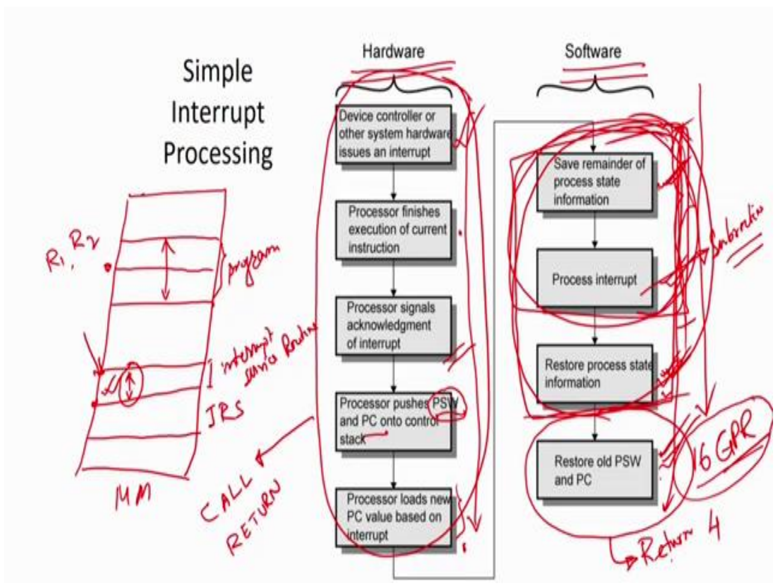
But in between I am having one more arrow. So, in that particular case, this is basically if we are working with the vector data or maybe say in an array, you just see that if I am going to add two arrays, then what will happen. We are performing the same operation addition only that was the difference. So, I am having an array a and array b, if I am performing say $a[i] + b[i]$ then what will happen. This operation is same, I am going to perform this addition operation. So, after performing $a[1] + b[1]$ 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, we 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, we 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 state; 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 execution. 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).



Simple Interrupt Processing
Hardware
Software
Device controller or other system hardware issues an interrupt
Processor finishes execution of current instruction
Processor signals acknowledgment of interrupt
Processor pushes PSW and PC onto control stack
Processor loads new PC value based on interrupt
Save remainder of process state information
Process interrupt
Restore process state information
Restore old PSW and PC
R1, R2
program
interrupt service Routine
ISR
MM
CALL
RETURN
Subroutine
16 GPR
Return
4

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, we 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 don’t have any mechanism to keep the information that what stage we are executing currently.

So, that information that such several information we cannot return in that side, we 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 I/O 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 pushes 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 retain it.

So, we 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 or 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 return this 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 take 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 you 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 save 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 temporary storage 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 entered 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 we are doing save 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 push 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’s 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 stored 4 registers values to the stack, it is going to pop out those particular 4 register values and store it into the processor register. So, after that the 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 word and program counter ok.

This is the way that we can say or you can say that again, maybe in program itself we can write it, if we don’t 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 the hardware then what will happen? This particular interrupt handling situation will be a complex one and it will be a heavy instruction. So, that’s 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 portion 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 your return instruction in your instruction set.

So, for subroutine call we are having one instruction for CALL, for calling the 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 method is same execution of interrupt and execution of subroutine, but for your subroutine call, we are calling it with the instruction CALL and returning it from with the instruction RETURN, but in case of interrupt with an control signal, we are initiating this process and after completion of these things we will go back to this particular point.

So, this is the simple interrupt processing and these are tasks we need to do while performing the interrupt. Now what is the program status word? 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, zero bit, carry bit, equal bit and overflow bit.