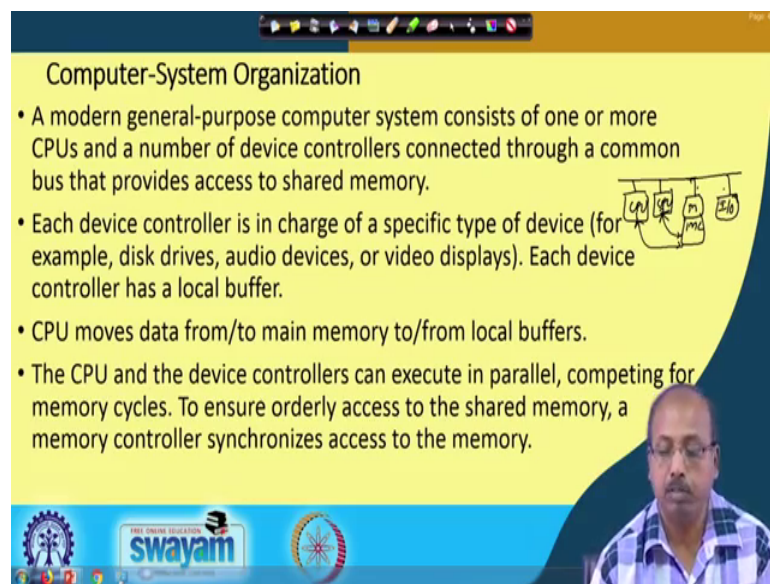


Operating System Fundamentals
Prof. Santanu Chattopadhyay
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture – 03
Introduction (Contd.)

Next, we will look into the computer system organization. So, how are the components of a computer system are organized?

(Refer Slide Time: 00:35)



The slide is titled "Computer-System Organization" and contains the following text:

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

A small diagram to the right of the text shows a central horizontal bus line. Above the bus are boxes labeled 'CPU', 'SP', 'M', and 'IO'. Below the bus are boxes labeled 'CPU' and 'M'. Arrows indicate data flow between the bus and these components.

The slide also features a video inset of a man in a checkered shirt in the bottom right corner and logos for "swayam" and "IIT Kharagpur" at the bottom.

So, a modern general purpose computer system, it consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory. So what we mean by this is something like that. So we have got up we can have one or more CPUs like it is. So, ideally the type of interface that we have is there is a CPU and there is a memory chip and there are IO devices that are connected.

Now this type of systems the basic problem is that at one point of time CPU can do only one job. So, I cannot think of parallelism in terms of programs being executed by the system. So, if there are even if there are multiple users. So I cannot execute their jobs parallelly. I have to do some sort of time sharing I maybe I do user one job for sometime, then user two job for sometime like that.

So, that way I can operate in a time multiplexed fashion, but that way performance or the throughput of the system will be low because the user jobs will take more time for completion. Now, think of a situation where I have got 2 CPUs instead of 1 CPU. So, there I have got 2 CPUs, then at least the 2 CPUs can do 2 jobs simultaneously.

So, there the number of programs that these CPUs can execute is just doubled of the actual or previous one. Now apart from that we have got a number of devices connected. So this IO device if then this memory so all of them are connected. Now you see this CPU there are two CPUs; CPU 1 and CPU 2. So, they need to talk to the memory.

So, when they send some request for some content of some memory location. So, 2 CPUs maybe requesting for content of two memory location now how to get the data ok? So, how to how like a so, memory will get confused because now are normally we know in a memory chip, we have got address bus, data bus and control bus and those values coming.

So, memory will be responding with the content of the corresponding location. Now, thus now in this case I need to have two address buses; two data buses and two control buses between the CPU 1 memory and CPU 2 memory. So, that makes it more difficult. So, you should have something which is additional which we call the memory controller or in short, I write it as MC, the memory controller should be there. So, all these CPU request that are coming should be coming to the memory controller and the memory controller will in turn decides like how to provide service for these requests from their individual CPUs and the data made available to them.

So, that so, similarly for each and every device that we have in the system so, they will have their own behavior and they need their own controller and fortunately or unfortunately this devices that we have in a computer system. So, they are not uniform. So, all of them cannot have same type of interface.

For example, the keyboard and mouse and the disk; so, they are not having similar type of behavior. So, keyboard the user is pressing some key. On the other hand and in that case key code or the text of the key should reach the computer. On the other hand, when the user moves a mouse, the mouse coordinates should reach the computer. Similarly, when you are transferring some data from a disk, then it is actually the data that has to go from the disk to this computer system and the amount the volume makes the difference.

So, keyboard also whenever we are typing a pressing some key, the key code is reaching the computer and when you are in a disk also so when some transfer is done, then there also some data is being transferred. But the nature is different. So, in case of disk it is a voluminous data; whereas, in case of keyboard. So, it is not that a big volume.

So, it is a small amount of data. So, that way these devices they are of different nature ok. So, each of them has got its own controller. So the overall diagram that we can have is maybe like this, we have got one or more CPUs and a number of device controllers connected through a common bus. So, the situation is like this.

(Refer Slide Time: 05:17)

Computer-System Organization

- A modern general-purpose computer system consists of one or more CPUs and a number of device controllers connected through a common bus that provides access to shared memory.
- Each device controller is in charge of a specific type of device (for example, disk drives, audio devices, or video displays). Each device controller has a local buffer.
- CPU moves data from/to main memory to/from local buffers.
- The CPU and the device controllers can execute in parallel, competing for memory cycles. To ensure orderly access to the shared memory, a memory controller synchronizes access to the memory.

So, you got a bus from which I can have a number of CPUs say this is C 1 and C 2 are two CPUs and then, we have got a number of device controller. So, we call it DC 1, device controller 1; DC 2 device controller 2. To device controller 1, the actual device D 1 is connected; to device controller 2, the actual device D 2 is connected.

So, individual devices can be memory chips maybe disk drives, maybe keyboards, maybe mouse; any device that you can attach to the system. So, this interface has to be provided, connected to a common bus that provides access to shared resource. So, this memory chip is also there as I said; now the memory locations need to be accessed. So, that way there the memory has access has to be provided.

So, each device controller is in charge of a specific type of device. For example, disk drives, audio devices, video displays and each device controller has a local buffer. So as I said for each device there is a device controller. Now, very often what happens is that for a number of devices, they are of same type.

For example, I may have say there 3 disk drives in my system. I may have say a number of audio devices in my system. So for them having separate device controllers may not be necessary. So I can have a single device controller that can control all the audio devices or similarly, single device controller that can control all the video devices or disk drives so like that.

So, that way I can have device controllers of a specific type and each device controller will have a local buffer so that any transfer that is going to take place between the computer system and the device driver or the device controller. So, they can be buffered there.

So, as I was telling previously that when say one program is printing something on to the printer, another program also wants to print. Now, this print data cannot be allowed to go to the printer, but we can keep it in some local buffer and the say for the second program, the content may be kept in a local buffer. And then, later on when the printer is free that data may be printed. So that is why we should have some local buffer available with individual controllers.

CPU moves data from or to main memory towards from these local buffers. So, CPU does not view each of these devices in a separate fashion. So, they just that the CPUs job ends by transferring the input output data to and from the buffer within the device controller. So we have got if this is the device controller.

So, within this device controller, we have got the buffer. So this is the buffer and with this buffer with this device controller the actual device is connected. So, the CPU does not send the data to the device directly, but it sends to this buffer and from this buffer the device controller will be writing to the device. So we have got this thing that each device CPU moves data from to main memory to from local buffers.

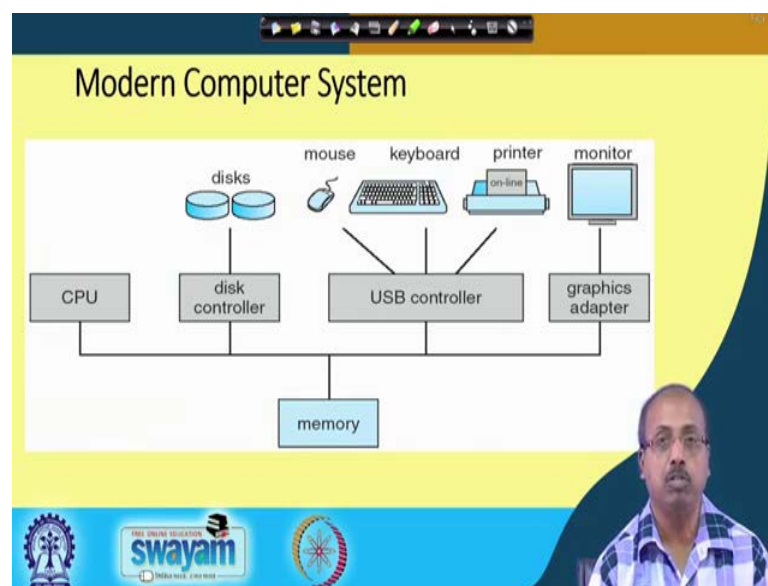
The CPU and device controllers can execute in parallel. So this is one good thing like when the device when the printer device controller is printing something on to the

printer. So CPU is free. So, CPU can do some other computation. So, they are executing in parallel. So as long as they do not need to access main memory, there is no problem.

But at some point of time it may so happen that both of them need to access main memory. So they will start competing for the memory cycle. This is more common if you have got multiple processes multiple CPUs. So, they want to access memory and there will be a fighting between them for getting access to the memory.

So, to ensure orderly access of the shared memory, a memory controller has to synchronize the access to the memory. So, we need some sort of memory controller which will be making this access pattern proper and it is not that one process request for a memory location content and it is given to some other process that should not happen. So all these are controlled by the memory controller.

(Refer Slide Time: 09:50)



So we also have got this different other devices like say this is a typical situation of a modern computer. We have got the CPU and memory. So, these were there in the traditional computer and we can have a number of IO devices like disk, mouse, keyboard, printer, monitor etcetera. Out of this may be in any most of the modern computer systems.

So they have got so this mouse keyboard printer. So they are all USB driven devices and we can have some USB controller ok. So, that these all these interfaces become standard

interface. Though, we have also have got USB disk, but so there are many disk which are not connected via USB because of speed limitations and all.

So, there I can have separate some other type of disk and the corresponding disk controller should be present. Similarly, the monitors can also be USB driven, but the type of feature that the monitor has in terms of resolution, in terms of graphics capabilities, so this basic USB may not be able to provide it. As a result we keep separate graphics adaptor that connects to the monitor.

So, when this CPU looks into the system. So apart from the memory, so for all other devices it looks in a very in a homogeneous fashion. So whether it is a disk write or a writing on the printer or writing on the monitor CPU issues similar type command to the corresponding controller. Now it is the controller's responsibility to translate it into the appropriate command for the underlying device and get it executed by the underlying device.

(Refer Slide Time: 11:37)

Computer Startup

- **Bootstrap program** is loaded at power-up or reboot
 - Typically stored in ROM or EPROM, generally known as **firmware**
 - Initializes all aspects of system
 - Loads operating system kernel and starts execution

Processor → FCK-0000

swayam THE ONLINE SOLUTION

So modern computer systems they will look something like this. So, any computer system what happens, when we start it up? So we switch on the computer, then what happens? So typically what happens is that there is a bootstrap program which is loaded at power up or reboot. So what happens is that if this is the CPU, in the basic architecture that you have. So, there I said that there is a memory chip connected here and there can

be multiple memory chips and then a part of the memory is called Read Only Memory or ROM and another part is called RAM or Random Access Memory.

Now, read only memory; the point is that the content is not lost even if the CPU power is off. So, it is there. So, what this CPU does? So, if you look into any processor any microprocessor, so you will find that it tells that if you reset or power on this microprocessor its program counter or pc gets some specific value. For example if you are looking into 8085. So it gets the value all 0. So, that way depending on the processor, so this manufacturer they will tell what is the value of this of this program counter when the system resets and as you know that when so if I in microprocessor if I switch on or if I press the reset button, program counter will get this value.

And then, at the next cycle what it does is that it puts the content of that program counter onto the address bus and whatever be the content of the corresponding memory location, it gets loaded into the CPU and that is taken as an instruction and executed and then the content of the next memory location the PC value the program counter value is updated and that is put onto the address bus.

So that it gets the content of the next memory location; until and unless the current instruction is a jump instruction. So that way the process goes on. So, this operating system designers they have to use this particular facility for locating the operating system and making the system to execute the part.

So the since the operating system is often very large. So it is not possible to hold the entire operating system into this read only memory. So what is done is that only the very small part of it is loaded into is kept into this read only memory. So which is called bootstrap program. The purpose of this program is to load the So, this computer system got the disk connected to it. So, here the actual operating system is loaded here.

So what this bootstrap program will do? It will be copying this operating system from here and put it into the RAM, when the system reboots. So that is or the system boots when the system reboots or the power is switched on.

So this part of the program which is this part of the operating system which is kept in ROM for EPROM is called firmware. Because this is not fully hardware; but it is not fully software also, because I am not going to change it. So this is this is something

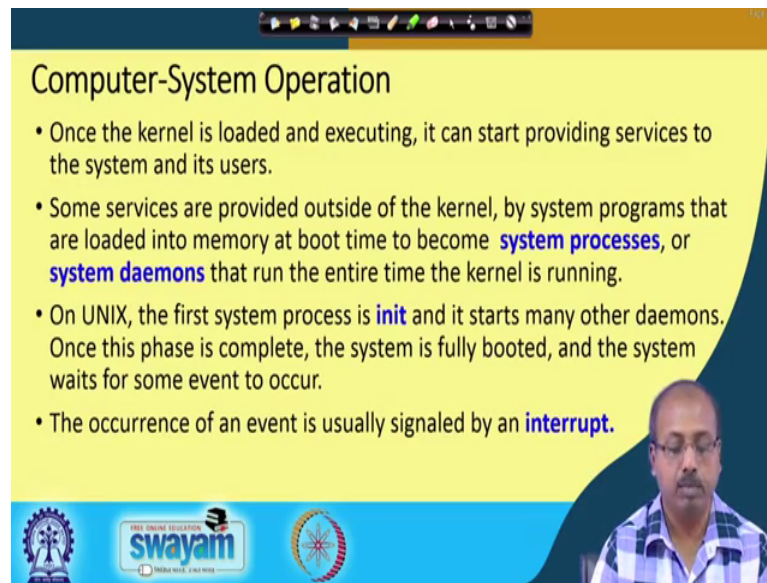
intermediary between hardware and software and that is why it is called a firmware. So the basic responsibility of this bootstrapping or firmware is to initialize all aspects of the system. So it will be initialize all the devices that you have the system and then, then it will not operating system kernel and start execution.

So, then it will be as I said that the operating system is kept in as a fixed location in the disk and from there, it will be loading the operating system, put it into ram and then, from that point onwards system is fully functional and it starts operating accordingly. So, this is the computer startup process or the booting process. So many times you may find, you might have notice this error that this computer the operating system could not locate the boot device. The reason that the further bootstrap ROM has said whatever address is given for the disk may be that disk is corrupted. So, it could not reach that particular point or the disk may not be inserted into the system.

So, previously when we have got in particularly the lightweight systems, where we have to boot it using some external disk or say over a network like that. So, that is not done. So, now I have got the concept of this thin client and all where this system is there in the server and this thin client it has to load the operating system from the server. So here the bootstrap or the this ROM; the EPROM or EPROM that I am talking about. So, it will have some routine so that it can access over the network the server that that you that contains operating system.

So, there it is, here bootstrap operation is not copying from disk, but copying over the network. So that way the meaning of this bootstrap operation may change as you go from one system to another system. But essentially it loads the important part of the operating system and starts executing operating system.

(Refer Slide Time: 17:14)



Computer-System Operation

- Once the kernel is loaded and executing, it can start providing services to the system and its users.
- Some services are provided outside of the kernel, by system programs that are loaded into memory at boot time to become **system processes**, or **system daemons** that run the entire time the kernel is running.
- On UNIX, the first system process is **init** and it starts many other daemons. Once this phase is complete, the system is fully booted, and the system waits for some event to occur.
- The occurrence of an event is usually signaled by an **interrupt**.

The slide also features a small video inset of a man in a checkered shirt in the bottom right corner, and logos for 'swayam' and 'THE ONLINE EDUCATION' at the bottom.

So we have next we will be looking into the computer system operation. So once the kernel is loaded and executing, it can start providing services to the system and its users. Now the kernel has been loaded. So any operating system can be thought of as a set of services as from the from the users view point. So you can think of an operating system as if it provides me a set of services.

Like it allows me to execute a program, it allows me to open a file, it allows me to read from the file or write onto the file or closing the file. So like that any operating system can be viewed as a set of services. The some services are provide outside of the kernel by the system programs that are loaded into memory at boot time to become system processes or system daemons that run the entire time the kernel is running

So, some of some of the services they need to modify the system resources. So as like the a new process is created. So process table has to be updated. A new file is opened. So, that way a file has to be updated file table has to be updated. So like that the kernel has to be updated. So that is they are called daemons system daemons and there are certain services that are provided by the system. So, they are provided separately like say compilation job or so, so like that. So, there they maybe system processes.

So, in on UNIX operating system, the first system process is the process whose name is init. So if you look into any UNIX operating system and try to see what are the processes running in the system now at any point of time, you will find a process whose name is

init. So this is the first process that is created in the UNIX operating system. So this when the system boots so, this is when the system boots this is the first process that is created and its starts other daemons. So and then once this phase is complete, the system has fully booted and the system waits for some event to occur.

So, normally if you boot a UNIX operating system, you will you will we may find this type of messages on the screen the init and all and then, we can find that several demands are getting started. So, these daemons starting is the done by the init process and then when it is completely booted. So, then it comes up with a login message and it waits for some user to login into the system and ask the system to do something.

So that way this system will wait for some event to occur and when that event occurs. So, it is operating system is inform that is that is event does occurred and accordingly, the system takes care of that. The occurrences of an event is signaled by an interrupt. So this interrupt maybe a hardware interrupt, maybe a software interrupt. So, we will see that that signifies the occurrence of an event and accordingly, the system starts doing their operation.

(Refer Slide Time: 20:11)

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

- There are two types of interrupts:
 - **Hardware** -- a device may trigger an interrupt by sending a signal to the CPU, usually by way of the system bus.
 - **Software** -- a program may trigger an interrupt by executing a special operation called a **system call**.
- A software-generated interrupt (sometimes called **trap** or **exception**) is caused either by an error (e.g., divide by zero) or a user request (e.g., an I/O request).
- An operating system is **interrupt driven**.

Handwritten note: $z = x/y, y=0$

The slide also features a video inset of a man in a checkered shirt in the bottom right corner and logos for "THE ENGINEER EDUCATION SWAYAM" and "MOOC" at the bottom.

So there are two types of interrupts as I said one is hardware interrupt and another is software interrupt. A hardware interrupt is a device may trigger and interrupt by sending a signal to the CPU usually by way of the system bus. So it is like this suppose I have got a keyboard connected to my system.

Now if the keyboard has to be operated the user has to press some key at some point of time and now you will see that the computer system that we have or the processor that I have so that is an electronic device. This keyboard is a mechanical device and the person who is operating it in most of the time is a human being. So human being has got an inherent speed limitation in which he or she can enter can place the characters on a keyboard.

So the electronic part or the processor part is much faster compared to this human being. Now, if the system is designed like this that the processor will check whether the user has pressed some key or not and it waits for that then it becomes clumsy (Refer Time: 21:23) because the user the processor is waiting unnecessarily for a long amount of time.

So one solution is that the processor maybe doing something and periodically, it comes back to check whether the user has pressed any key or not. But then, the difficulty is that users the response that the user will see is not very much deterministic.

So it will see sometimes that as soon as the key is pressed it is responded to because the processor has checked for the status of keyboard very soon or some point it may be the response is not that fast because the processor was busy doing something else and it checked that keyboard status after a long time. So, that can happen.

Now, the other solution to this is as and when the user presses some key, the CPU is informed that a key has been placed and the processor takes care of that. So, immediately the processor suspends whatever it was doing and it comes to the user process, it comes to the operation by which it can read the content of the keyboard. Now, this way since the user cannot type at a very high speed, so the user will not be able to generate a large number of interrupts very fast. So the overall system works very fine.

Similarly when we have to transfer some data from the secondary storage to the main memory or to the processor, then also the same thing secondary storage in most of the time is an electromechanical device. So, that way it takes time to read something or an optical device that take some time to get the data from transferred and all.

So the processor or the CPU should not wait for that much of time. So, when this processing is over this, when the disk controller has done the transfer; then it may tell the CPU at the transfer is over. So, you can take the content from the location. So, that way

we can have some um interrupt coming from the disk controller. So this is this is one type of situation.

Other type of situation is that a software may generate some interrupt, it is a program may trigger and interest by executing a special operation called a system call. Like while executing the program maybe a program has given a statement like to output something on to the printer. So that requires services on the operating system as I as we discussed previously the printer is not given simultaneously to more than one process. So, that way it is a system call.

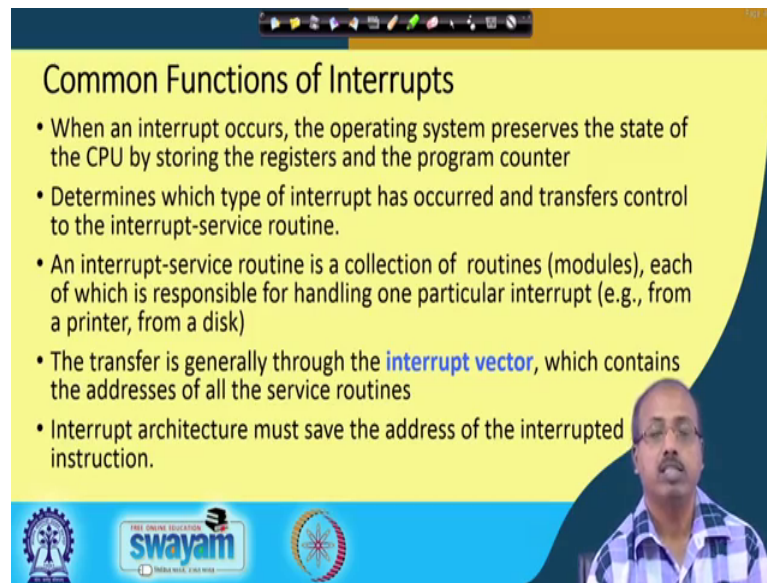
Similarly a program it wants to create another it wants the start another program. So, that way it has to get a service from the operating system. So, these are the interrupts generated by the software. A software generated interrupt is caused either by an error or user request like IO request. This is other source of software interrupt that we have is the some error that has occurred like say I have got say x I have written statement like z equal to x divided by y .

Now, what is y is equal to 0 ok. So, at the time of program compilation, so we could not check this because we did not know what is the value of y . But the runtime, while this statement comes into execution, the hardware finds that y value is equal to 0. So the division operation cannot be carried out. So, that is a divide by 0 error.

So, what the system will do when this occurs. So, that it must transfer the control back to the operating system telling that a divided by 0 error has occurred and then, operating system should take appropriate action. So which action that is of course, different that is dependent on the operating system and the user program, but this is a special situation. So an operating system is interrupt driven. So after doing the initialization part the operating system actually waits in a loop for some interrupt to occur and when some interrupt occurs.

So, it will be telling the it will be informing the system that and interrupt has occurred and then, the appropriate action will be taken by the operating system. So this interrupt is another very important thing that these modern systems will have. So, that we can do the operations done by the operating system.

(Refer Slide Time: 25:59)



The slide is titled "Common Functions of Interrupts" and features a yellow background with a dark blue border. It contains five bullet points. At the bottom of the slide, there is a blue banner with logos for "swayam" and "THE ONLINE EDUCATION" along with a circular emblem. A small video feed of a man in a checkered shirt is visible in the bottom right corner of the slide.

Common Functions of Interrupts

- When an interrupt occurs, the operating system preserves the state of the CPU by storing the registers and the program counter
- Determines which type of interrupt has occurred and transfers control to the interrupt-service routine.
- An interrupt-service routine is a collection of routines (modules), each of which is responsible for handling one particular interrupt (e.g., from a printer, from a disk)
- The transfer is generally through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction.

Common functions of interrupts; when an interrupt occurs, the operating system preserve the state of the CPU by storing the registers and the program counter. So this is the first thing that the system does. So it remembers like what it was doing. Because some time later it has to restart the operation. So it will be saving the content of the current state of the CPU storing the registers and program counter. Determines which type of interrupt has occurred and transfers control to the interrupt service routine.

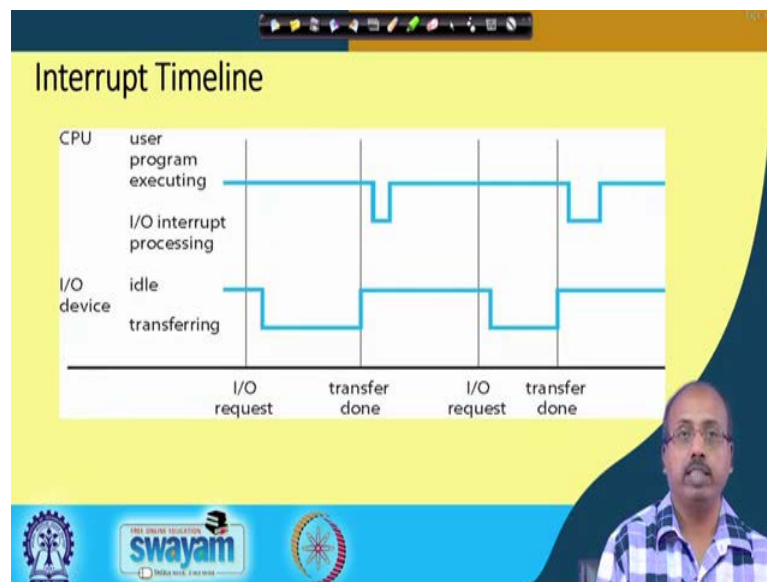
So, whether it is a maskable interrupt or non-maskable interrupt; what is the priority of the interrupt? So, based on depending many issues that you can get into the discussion can be obtained in more detail in some computer architecture classes or microprocessor classes. So I will not going to that, but after determining the type of interrupt that control will be transferred to the corresponding interrupt service routine. Interrupt service routine is nothing but a collection of modules or routines with a responsible for handling one particular interrupt.

For example, a printer has given an interrupt that whatever printing job was given is over. Keyboard gives an interrupt that a key has been pressed or a disk gives an interrupt telling that the disk transfer is over. So, what the corresponding action should be that is known to the system designer and accordingly, the operating system designer has written the corresponding interrupt service routine. So this interrupt that comes it goes to the interrupt service routine and accordingly the appropriate action is taken. The transfer is

generally through as the interrupt vector which contains addresses of all the interrupt service routine.

So for the entire system for all the interrupts to which address it should go, which where is the actual interrupt service routine located in the main memory so that is kept in a table called interrupt vector table and from there, this interrupt vector addresses are taken and it goes to the corresponding service routine. The interrupt architecture must save address of the interrupted instructions. So this is very important because it has to restore when the interrupt is over it has to restore the operation from there.

(Refer Slide Time: 28:16)



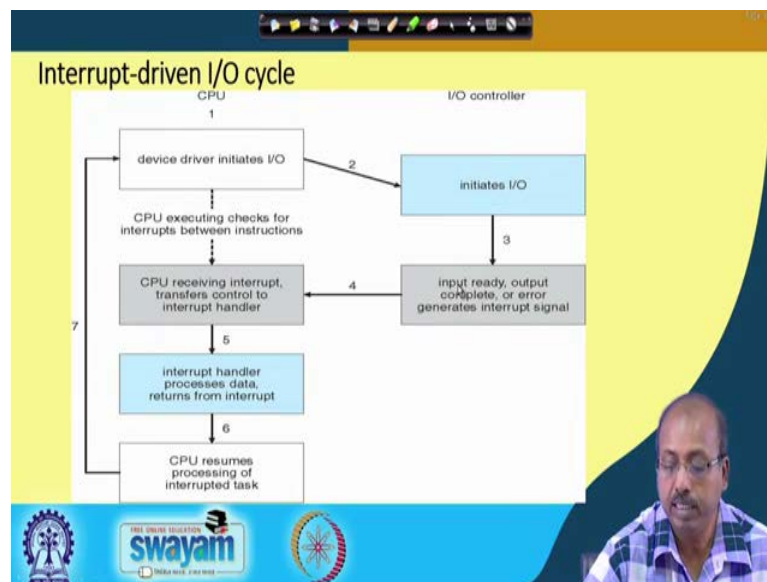
So, on the timelines we can say that for sometime the user program is executing, then after sometime the interrupt comes like for example, an IO device. So, it was told to do some transfer. So, it was idle for this much a time, then it was transferring the data. When the transfer is over, it gives an interrupt to the processor CPU telling that transfer is over.

So, the processor goes into interrupt processing cycle. So in this time it is doing the interrupt processing. Then again it is going into the service, it is going into the normal operation. So, it is going like this. Then after sometime again another IO request is made by the CPU by the program.

So, when the IO request is made the device is device is informed and the device makes a transition from idles state to the transferring state. So, again after it starts with the transfer when the transfer is over it sends an interrupt to the processor of the CPU and after sometimes CPU goes into the interrupt service routine, executes the interrupt service routine in this part. So this way it goes on.

So, transfer is initiated by the CPU after the device has done the transfer it gives an interrupt to the processor telling that transfer is over. Now, the processor goes into the interrupt service routine to decide what exactly has to be done for the particular operation.

(Refer Slide Time: 29:49)



So, this overall flow is like this. The CPU device driver initiates the IO transfer that is first step number 1. In step number 2 for the IO controller, so it initiates the IO operation. Step number 3, input ready output complete or error regenerated etcetera. So, IO operation is over at this point. So it sends an interrupt to the processor. So in between CPU is checking for executing checks for interrupts between instructions or the interrupt has occurred between two instructions or not. So those checks had done.

So when these interrupt come, so CPU receives the interrupt and transfers control to interrupt handler. In turn the interrupt handler, we process the data and returned from the interrupt and then, after that when the interrupt is over it will the CPU will resume from processing the interrupt resume to processing the interrupted task.