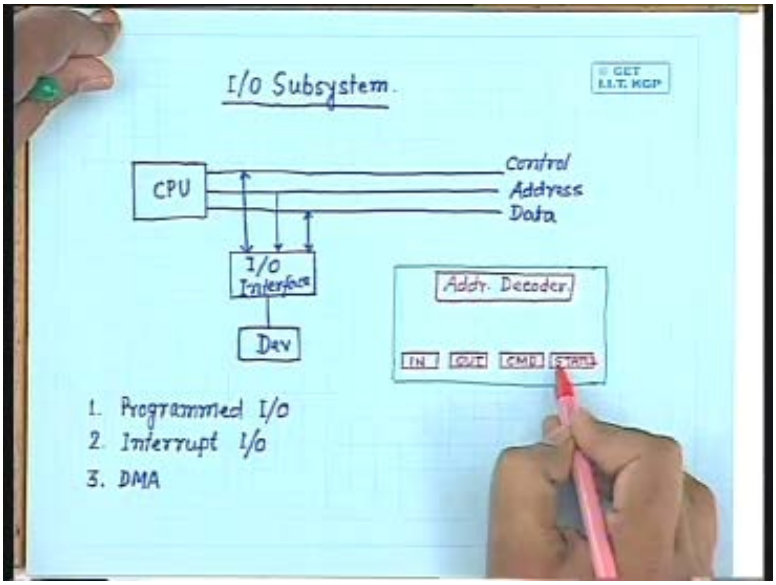
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic & Electrical Communication Engineering
Indian Institute of Technology Kharagpur
Lecture No. # 27
I/O Subsystem Organization

Today we are going to discuss about what is called an input output or I/O subsystem. You know that in any computer system, a number of input output devices including say keyboard, video monitor, printer then again the hard disc is also considered as an input output or I/O system.

(Refer Slide Time: 00:01:12 min)



Now when we talk about this I/O subsystem, it is the unit which interacts with the input output device. So if you look at any computer system as we have seen earlier that in a computer system you have at the heart of the computer system which is the CPU and the CPU is connected with a number of input output systems in addition to the main memory. So I will have a number of buses few of the lines taken together will be called control bus then address bus and data bus. So suppose this represents the control bus, second set of lines represents the address bus and the third set of line may represent the data bus.

So whenever you connect to any input output device, you have to have an I/O interface unit. This is what we will call as input output interface and this input output interface actually connect with the input output device and as we said that this device can be of various forms. So actually this input output interface, this is what interfaces the device with the CPU. So input output device has two interface unit will have the connection with the control bus. It will also have connection with the address bus and it will have the connection with the data bus, so the connection will be like this. Now when we talk about an I/O subsystem, we can have three different kinds of I/O. The first kind of I/O is called programmed I/O that means the input output device is under direct control of the CPU, so any software which has to be executed to access the input output device

will be executed by the CPU. I/O operation will be initiated by the CPU, it will also be terminated by the CPU. The second kind of I/O that we can have is called interrupt I/O. In case of interrupt I/O whenever an I/O operation is to be initiated, the CPU simply informs the I/O interface to start the I/O operation. At the end of the I/O operation the I/O interface will interrupt the CPU informing that I/O operation is complete. Now whatever the CPU has to do with that data either received from the I/O or sent to the I/O that the CPU can do now and the third kind of I/O is what is called DMA or direct memory access and in this case the data is transferred between the input output device and the main memory directly by the DMA unit. The CPU does not come into picture at all.

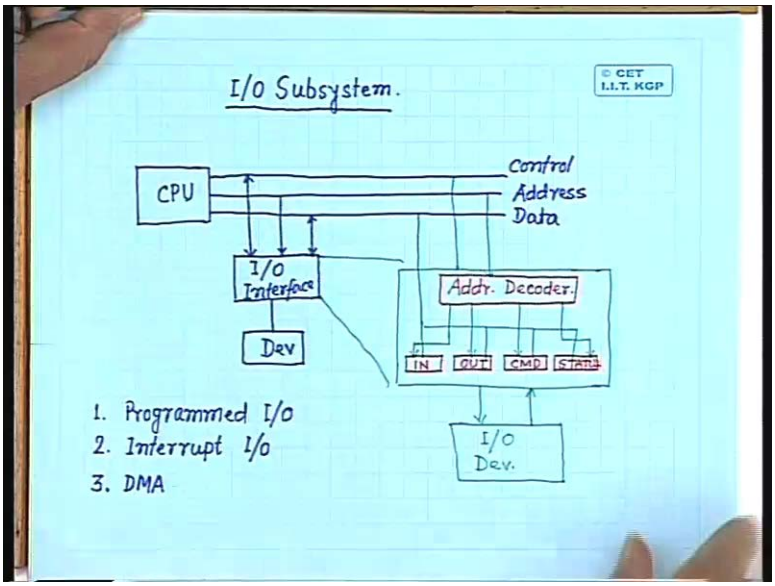
So if I go for this programmed I/O in that case what will be the nature of the I/O interface unit? If I expand this I/O interface unit, the I/O interface unit will have a number of components. The first component will be that since every I/O device will have an unique address in the system, so I have to have an unit called address decoder. So in this I/O interface unit, I will have an unit called address decoder and you might be knowing that there are some other units which are known as say in register, out register. Then I have to have what is called a command word and I have to have another one called status word. So in this I/O interface unit if I go for the programmed I/O kind of configuration, the I/O interface unit will have an address decoder, it will have an in register so that for any inputting operation the data which is transferred from the input device is stored in the in register then the CPU can read the data from the in register.

Similarly for an output device when a data is to be transmitted to the out or stored in the output device or is to be sent to the output device, the CPU will write data into out register then from the out register it will go to the device. Common register is mostly used to configure the I/O operation that is at what speed the I/O should operate, what should be the word length or how many bits will contain the data, what will be the error correcting message all those informations will be configured in the command register. The status register will inform the status of the I/O device. So these are the kind of things which you might have used in your microprocessor course say for example if you want to connect an USART that is universal synchronous asynchronous receiver transmitter so in that case what we have to do is we have program what is the baud rate that is at which rate the data is to be transmitted.

You also have to program that how many start bits or how many stop bits you want to have, what is the data length whether a character will consist of 7 bits or a character will consist of 8 bits, all those informations are to be stored or to be written into the command register and then only the I/O interface will act accordingly. Similarly the status register will be used, suppose the CPU wants to read a data from an input device, so firstly by configuring the command register, it configures that in which way the data communication will take place then the status register will tell that whether the data which is to be read is available in the in register or not that is whether this device is ready with the data. So that information will come from the status resistor and from the status register whenever the CPU finds that the data is ready in the in register, the CPU can read the data from in register. So for doing this after sending a command to the command resistor, what the CPU has to do is the CPU has to remain in a loop always checking the condition of the status register whether the data is ready or not. So only when it finds that the data is ready, it reads the data from the in register and comes out.

So this is why this kind of I/O operations is called program I/O because all the input output operations are directly done under the direct supervision of the CPU through some program. Now each of this in register, out register, command register or status resistor they will have different addresses that means I have to have some connection from the address decoder to the in register. I have to have a connection from the address decoder to the out register, I have to have a connection from the address decoder to the command register, I also have to have a connection from the address decoder to the address register so that I can have unique address for each of these registers. In some of the I/O interfaces you will find that the in and out registers they have the same address.

(Refer Slide Time: 14:12)



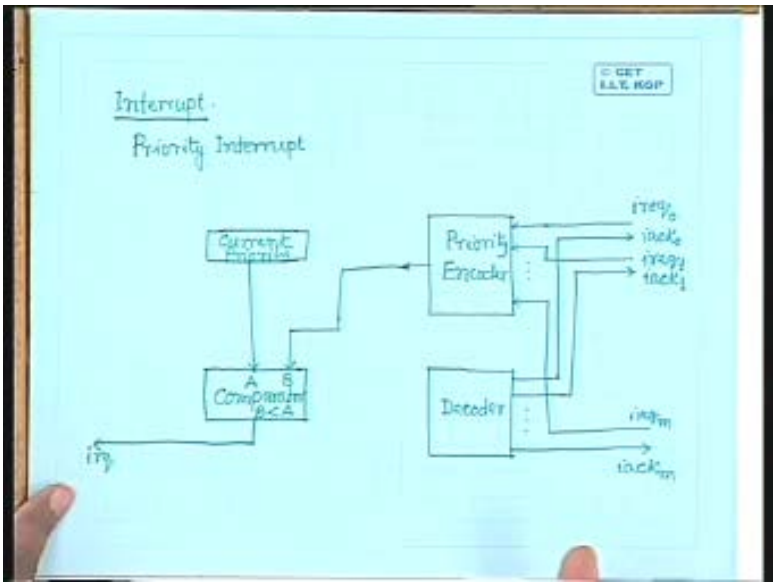
Similarly command and status registers they have the same address. There the idea is suppose in out registers they have some address say AA, AA hexadecimal so with that AA address if I want to write anything, the data will always come in the in register. If I want to read anything from A address, the data will be read from the out register sorry it is opposite. Similarly here suppose this has an address of AB both command register and status register, so with address AB if anything is written into this it will be written into command register. If anything is read by the CPU with address AB then the information will be read from the status register that means these registers are unidirectional. To one register you can only write, another register can only be read. Yes, may not be. I am taking a specific case, there are a number of situations when the I/O interface unit will have much more number of registers than this.

This I/O interface unit can have an internal memory say for example if I want to design an I/O interface for interacting with the key board, for interacting with the video unit in such cases the I/O interface unit will have some internal memory in addition to these registers. So this is just a particular example, a very simple example the system can be complicated further. Not only that, the I/O interface unit can be a processor itself, this itself can be a sequential machine. So in some cases if our I/O operation is very very complex in that case such simple interface may not be sufficient. So for every I/O operation the task can be given to the I/O interface unit or I/O

subsystem, the subsystem will take care of the task in its entirety, it will not depend upon the CPU. In such cases its I/O interface unit has to be a processor by itself. In many cases such units are called I/O channels, I/O channel.

DMA is a kind of such channel, I will come to that. So in this case this in out registers, command registers and status registers through this you can control all the input output operations. So obviously this address decoder it will have a connection from the address bus. All these registers in registers, out registers, command register and status register they will have connection with the data bus so I can put it this way and at the same time, the control signals which come, this control signals also propagate to these registers because among the control signals we have the read signal, write signal and all those things so whenever a read operation is to be performed this in register has to be activated. Whenever an output operation is to be performed or write operation is to be performed either the out register or the command register will be activated depending upon what address comes from the address decoder. So if you expand this I/O interface unit, it will look like this and here I will have the I/O device and this kind of interface unit is mostly suitable for programmed I/O kind of operation.

(Refer Slide Time: 00:14:17 min)



Now as I said that there is another kind of I/O operation which we call as interrupt I/O. In case of interrupt I/O what is assumed is whenever some device needs some service, so in this case the serve was the initiated by the CPU, it was terminated by the CPU. In case of interrupt I/O we assume that whenever a device needs some service, the device interrupts the CPU. Now the actions or the services for different devices are different. The service needed by a hard disc will be different from the service needed by a keyboard or the service needed by a keyboard will be different from the service needed by an output device like printer. So in the CPU, what the CPU will do is on getting an interrupt it will identify that which interrupt it is and following that identification it will execute a program which is called an interrupt service student and it is this interrupt service student which meets the requirement of the device.

So whenever any I/O device needs some service from the CPU, it is the responsibility of the I/O device to put the request, service request in the form of an interrupt. So I can have two different kinds of interrupts. One kind of interrupt is called a priority interrupt which you have done with a 8085 microprocessor where you might be knowing that there are different types of interrupts 7.5, 6.5, 5.5 trap and all these things and there is another interrupt which is called INTR. All these interrupt lines have got different priorities, trap has got the highest priority, INTR has got the lowest priority or among the vectored interrupts trap has got the highest priority and RST 5.5 has got the lowest priority.

See if I go for this priority interrupts what are the units that we need? Firstly because a number of devices can put the interrupts simultaneously to the CPU, if the interrupts are not simultaneous then I don't have any problem. If only one device puts an interrupt at a say, at a time then that interrupt can immediately be acknowledged and serviced but the problem comes when more than one devices interrupt simultaneously. In that case a decision has to be made that out of all these devices which device has to be serviced first. I have to set some interrupt priority level and to do this what I need is a priority encoder so this is what is known as a priority encoder.

Priority encoder has got a number of input interrupt lines; these lines have got different priority levels. Now out of all these lines if more than one line's are active simultaneously then the line which has got highest priority among them will be selected and passed to the output of the priority encoder. Now along with this, what is needed is because now I have a number of devices connected together and many of them can give interrupt simultaneously, when the interrupt is acknowledged by the CPU then the device whose interrupt is acknowledged that the device must know so that the device can start operation. So along with this priority encoder, we also have to have another unit, the reverse unit which we call as a decoder so this is a simple decoder.

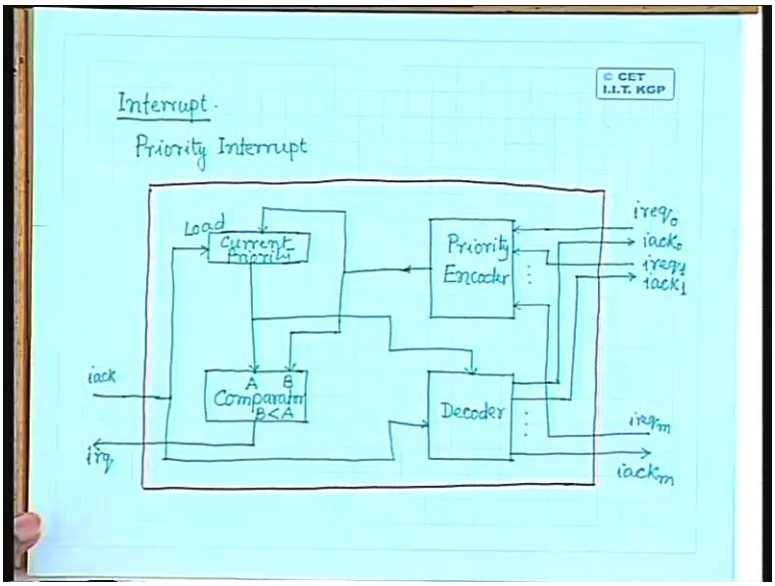
The devices will give an interrupt suppose this is, I name this as interrupt request line and this is the interrupt request line zero. When an acknowledgment comes, the acknowledgment should also reach the device and the acknowledgment will be generated by the decoder unit, so this is interrupt acknowledgment zero. Similarly we will have interrupt request one, interrupt acknowledgment one which will also be generated by the decoder will go to the device which is connected to this interrupt request one line, so this is interrupt acknowledgment one.

Similarly if there are say m number of devices then this will be interrupt request m and similarly an interrupt acknowledgment m will be generated by the decoder. Now before an interrupt is actually entertain by the CPU, the CPU has to compare what is the level of the interrupt which is coming from a device with respect to the interrupt level or priority level of a task which is under execution. So if the CPU is executing a task of which the interrupt level is say 5, now while that task has been executed if another device whose interrupt level is say 2 puts an interrupt then if the priority of interrupt 2 is less than the priority interrupt 5 then this new interrupt will not be accepted. Whereas if the priority of 2 is greater than the priority of interrupt level 5 then the new interrupt will be accepted, so I have to have some memory element which will tell me that what is the current priority level of a job which is under execution. So this gives you the current priority level. I have to compare this current priority level with the new priority with which an interrupt has come. So I must need a comparator, one of the inputs to the comparator will be from the current priority level and the other input to this comparator will be from the new

interrupt priority level that has come. So this is a comparator let me call this input as A, this input as B.

So I will have an output which will actually give the interrupt request to the CPU. So this is the actual interrupt request to the CPU and I will generate this interrupt request following some logic. So my logic will be that if a priority level low indicates a high priority, I can have different types of logic. Suppose at the input I have 8 devices, so I can have priority levels from 0 to 7. I can assume that a priority level zero is the maximum priority or highest priority or I can also assume that a priority level 7 is the highest priority. So accordingly this comparator has to be set. So if I assume that the lowest priority level indicates the priority is actually high that means whenever B is less than A then only this interrupt will be generated. So it will be an output whether B is less than A or not. So if B is less than A, then only you are generating an interrupt. So obviously this interrupt has to be accepted by the CPU and when the interrupt is accepted by the CPU in turn the CPU will give an interrupt acknowledgment.

(Refer Slide Time: 24:00)



On getting an interrupt acknowledgment that means the CPU is now going to take the new task. So the current priority level which was set in this current priority register that has to be changed, it has to get this new priority level. So along with coming to this comparator, this priority encoder output should go to the current priority level register and this has to be loaded. So I have to have a load input to this and this has to be loaded whenever an interrupt acknowledgment comes, so interrupt acknowledgment will be given by the CPU. This interrupt acknowledgment will give a load input to the current priority level, when this current priority can be loaded into the current priority register and at the same time whenever this acknowledgement comes, it also has to give a signal to the decoder, decoder makes use of this current priority to generate, to activate one of the decoder output lines.

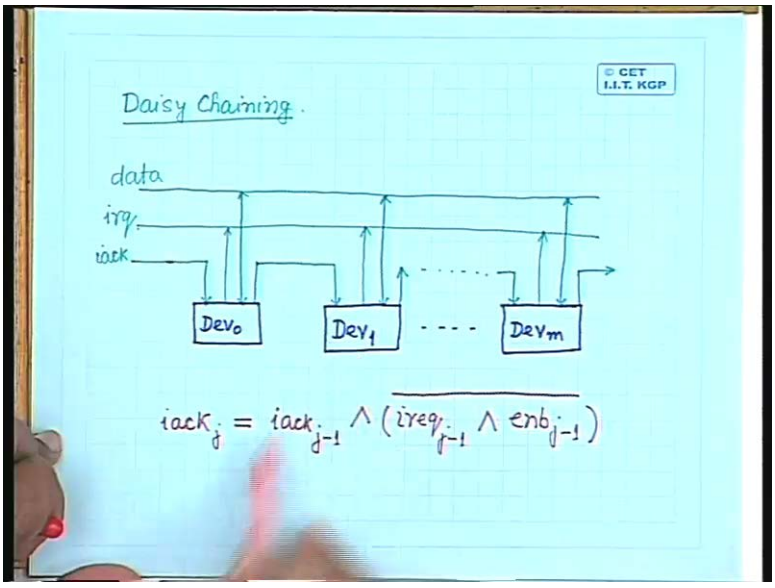
So whenever this acknowledgment comes from the CPU that means the new interrupt is being accepted, when this new interrupt is accepted, the current priority is changed. The new priority

value along with this interrupt acknowledgment signal that comes to the decoder and accordingly the decoder generates an interrupt acknowledgment signal which goes to the device which is being selected. So this is the total scheme of an interrupt controller, if I want to use the priority interrupt scheme. So this whole thing is the priority interrupt controller. Yes. That is what I have assumed, it can be reverse also. For this case interrupt level zero is of highest priority. I can have the reverse also, I can make interrupt level 7 to be the highest priority but in this case this comparator output will be changed instead of B less than A, I have to make it B greater than A. (Conversation between Professor and Student: Refer Slide Time: 26:20). That has to be done by the device itself, device controller has to take care of that.

Say in the device controller, the device controller will put an interrupt request then it has to wait for an interrupt acknowledgment, until and unless it gets the interrupt acknowledgment the interrupt request line should be kept high that has to be taken care of by the device controller in this configuration, correct. However in case of 8085, such a type of thing is implemented in 8085. So if two interrupt comes say RST 7.5 and RST 5.5 simultaneously then RST 5.5 goes into an internal register. So after 7.5 is serviced, 5.5 will also be serviced. There are additional interrupt controller chips also. Yes, some question from this side.

This is the interrupt acknowledgment, the other one is the current priority level. So the logic is because this decoder has to activate one of the decoder outputs. Which decoder output has to be activated, that depends upon this priority which has been accepted. So that is why it needs this input as well as this interrupt acknowledgment both are needed to generate an output signal high. The other kind of interrupt which can be used is what is called a Daisy chaining.

(Refer Slide Time: 00:28:16 min)



In case of Daisy chaining we don't have such a complicated circuit. Daisy chaining concept is very simple, say I have a set of data lines which are connected to the CPU, I have a single interrupt request line. So these are the data lines or data bus and this is the interrupt request line, in turn an interrupt acknowledgment will come from the CPU. I will connect a number of

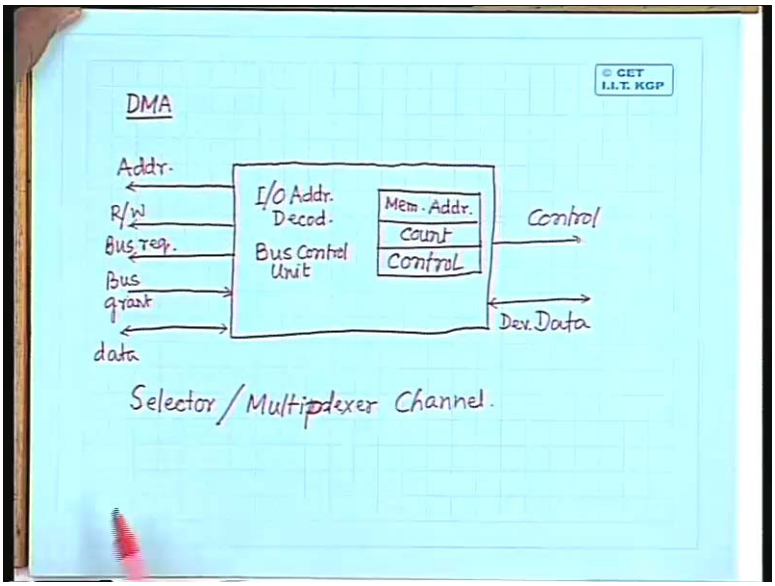
devices on the system. See this is device number 0, I have device number 1 like this, I will have see device number m. The Daisy chaining concept is whenever a device puts an interrupt, all the interrupts are connected to the same interrupt request line. So I can have some wired or kind of connection. All the devices are connected to the same data bus and this is usual, there is nothing special about it. What is special is the way the interrupt acknowledgment signal is connected. What is done is whenever the CPU gives an interrupt acknowledgment, the acknowledgment goes through the first device. From the first device, the first device gives an interrupt acknowledgment output signal, this output is connected to the interrupt acknowledgment input of the next device and this way it continues. The device m will get the interrupt acknowledgment from device m minus 1.

Similarly device m will give an interrupt acknowledgment out signal which will go to device m plus 1. So this way it continues. For any device say device j the logic is like this, interrupt acknowledgment of j will be active provided you get an interrupt acknowledgment out from device j minus 1 and interrupt request of j minus 1 and enable j minus 1, this is not true. So the concept is every device will have an enable signal, so device can put the request only when the corresponding device is enabled. Now because the acknowledgment signal moves from one device to another device in the form of a chain, so we find that this device one can get an acknowledgment signal from device zero only if there was an interrupt to the CPU. The CPU gives an interrupt acknowledgment signal, first is it reaches the device zero but for device zero the condition is something like this, enable of device zero and interrupt request of device zero inward of this if this is true.

This means two things either the device was not enabled or the device was enabled but it did not put the interrupt. Only in this case the acknowledgment signal will reach device one from device zero and the same logic follows. Acknowledgment signal will reach device two from device one if this is true for device one. Similarly device one can generate an output acknowledgment if it gets an input acknowledgment and the input acknowledgment can come from device zero if this is true for devices zero. So we find that the acknowledgment signal flows from one device to another device in the form of a chain. So that is why it is called a Daisy chaining priority interrupt scheme and here of course some priority is in built because the device which is nearest to the CPU has the highest priority. Isn't it? Suppose both device zero and device one both of them put interrupts simultaneously. Then because device zero is enabled and it has put the request so ireq and enable and these two invert it it becomes zero that means device one does not get the acknowledgment signal.

So device zero can start the operation but device one cannot start the operation. So some priority is already in built in the scheme that is a device which is nearest to the CPU will get the highest priority, device which is farthest from the CPU will get the lowest priority. You have to short this. No, actually these adapters are like that. If you connect it, it will be through this, if you disconnect it in that case that will be shorted, that is how the adapters are made. In fact this kind of scheme was used in, have you seen the earlier HP machines, HPIB called HP Hewlett Packard interface bus or something like this. HPIB was using this kind of scheme, that is any number of devices you just connect one device to the other device by a… (Refer Slide Time: 00:35:10) that's all. You can connect any number of devices on the system.

(Refer Slide Time: 00:35:22 min)



Coming to the other kind of I/O operation that is DMA or direct memory access. Now in this earlier scheme whether we go for programmed I/O or an interrupt I/O or basic operation is to transfer the data from the memory to a device or getting the data from a device storing it into memory. In case of a programmed I/O, the CPU itself will take the initiative to read a data from a memory, write that into an output device or read a data from an input device and write it into memory. That means the data has to be first read by the CPU then only it has to be given to the proper destination, transferred to the proper destination and that is through both in case of programmed I/O as well as interrupt I/O. In case of programmed I/O the initiative is taken by the CPU, in case of interrupt I/O the interrupt signal tells that when that action has to be performed but the action is done by the CPU. In case of DMA the concept is slightly different. Whenever you have to transfer some data, may be from an input device to the memory or from memory to the I/O device in that case the CPU does not come into picture.

So basic concept is whatever operation that was to be done by the CPU is now done by a separate controller which is the DMA controller. So whenever some device puts a request to transfer some data from the device memory to the main memory, the device puts a request signal to the DMA controller. In turn DMA controller gives a signal to the CPU that it wants to perform some DMA operation. On getting that signal the CPU gets a DMA acknowledgment and what is done after the DMA acknowledgment? Whenever the CPU generates a DMA acknowledgment at the same time, the CPU releases all the buses the data bus control, bus address bus everything. Those are no more physically or logically connected to the CPU. Now this DMA controller becomes the bus master. So what it does is it reads the data from the memory, sends that to I/O device or reads the data from the I/O device sends that to memory. So all the operation which otherwise would have to be done by the CPU, now it is to be done by the DMA control.

So accordingly the DMA controller will have to have a number of registers because it has to know that which device has to be activated. Simultaneously it also has to know that which memory location is to be accessed either for reading purpose or writing purpose. So number of

units that will be present in the DMA device will be same as the number of units that you have in the CPU, more or less same it is not identical. So some of the units will be say memory address register, I have to have memory address register, I also have to have an information about what is the length of the data that has to be transferred that is count. So if I want to transfer say 100 bytes of data from the main memory to a device, in that case what I can do is I can simply increment the memory register address by one in a loop of 100 and that can be controlled by this count value. And similarly I also have to have some control unit which will give the control signals to the memory as well as the I/O device.

On the other hand I have to have an I/O address decoder because may be a number of devices are connected to the same controller, same DMA and the DMA controller has to activate one of the devices, so there has to be an I/O address decoder. In addition to this there has to be a bus control unit and in addition there will be a number of other things like in some cases whenever the data is to be transmitted by the different bytes are packed together to make a single packet and that is transmitted. All those different additional control signals can be accommodated in the DMA controller. So on one side the DMA controller will be interfaced with the memory so for that we need the address lines, we need the read write signals, we need the bus request, bus request signal which will go to the CPU following this bus request the CPU has to give a bus grant signal which will come to the DMA controller from the CPU and on getting this bus grant signal from the CPU, the DMA controller can start operation and obviously I have to have the data lines. So this is the part, this side is to interface the DMA controller with the memory and the CPU.

On the other side the DMA controller has to be interfaced with the device so that means I will have a number of control lines for controlling the I/O device I also have to have a number of data lines which will carry the device data. So now we find that the responsibility of this DMA controller will be that on one side it will be interfaced with the device, so it can get the data from the device and on the other side it is interfaced with the memory, so it can send the data to the memory. Similarly it can get the data from the memory and send the data to the output device and while performing this operation, the read and write operations by the CPU is no more needed. Got it? And because of this additional registers in this DMA controller, the operation of the DMA controller can be much faster than that in case of the CPU because in case of CPU firstly the CPU has to read the data, get it into its internal register then from the internal register it has to send the data to the destination, so two cycles are always necessary if I want to transfer the data through CPU whereas similar operation can be done in a single cycle by making use of this DMA controller.

Again when you come to the DMA controller, I can have two different options. As I said that I mean this is a kind of channel because this DMA controller itself is a processor which can work independently of the CPU. So again in this case of DMA controller or channel as we said that such kind of devices are also called channels, I can have two types of options, one is called a selector channel and other one is called a multiplexer channel. What is the selector channel and what is the multiplexer channel? Suppose I use the same DMA controller to which a number of devices are connected.

Now a selector channel, what it will do is suppose all the devices want to get some service simultaneously. Then a selector channel will select one of the devices that device will complete

its operation then only the operation of the other device will be initiated. In case of multiplexer channel, the operations of different devices are time multiplexed that means if there are say 5 devices connected, first device will operate for say 1 milli second then during second one milli second period the second device will operate, may be the operation of the first device is not yet complete. So multiplexer channel multiplexes the operations of multiple devices which are connected to the DMA controller, in case of selector channel the DMA channel the DMA controller selects one of the device, the device completes its operation then only the operation of the next device is initiated. Now which type of channel we should go for? Whether we should go for the selector channel or we should go for the multiplexer channel that depends upon device characteristics.

Say if the devices are very fast in that case I can go for selector channel because as we have said that once a device is selected, the device has to complete its operation then only the next device can start operation. So if the first device takes a small amount of time then the second device can wait until and unless the operation of the first device is complete but if the devices are very slow in that case the waiting time may not be tolerable. So if the devices are slow, we should go for the multiplexer channel. If the devices are fast enough then we can go for the selector channel. So let us take some break.