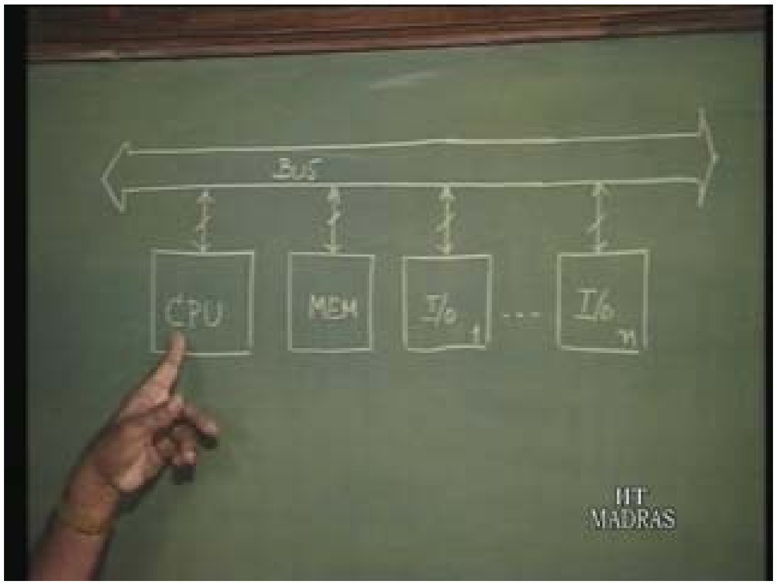
Computer Organization
Part – III

Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Input/output
Lecture – 25
Interrupt-Driven I/O

In the previous lecture, we took a look at one mode of transfer, which was essentially programmed I/O. I have also explained why it is called programmed I /O. Essentially by running a program the CPU affects an input/output and the way the CPU will deal with the device entirely depends on the program.

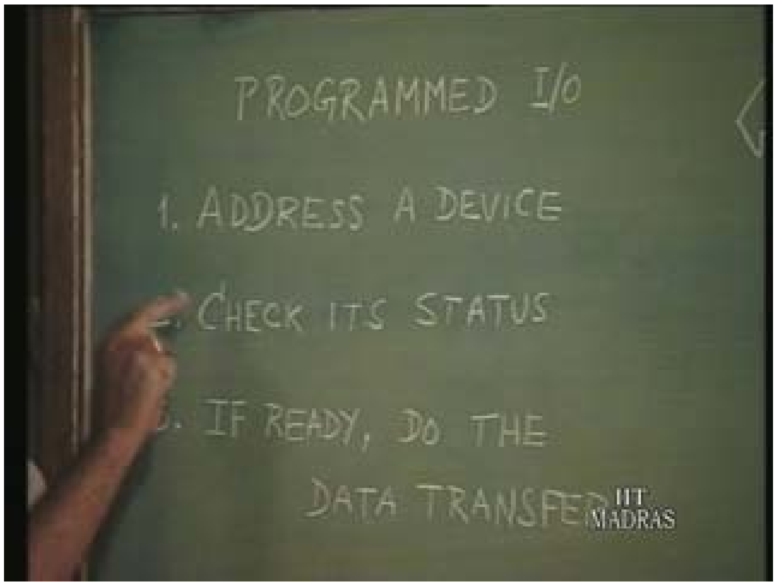
(Refer Slide Time: 01:30)



It can be changed any time that it is wanted. What was the essential thing about the programmed I/O? We saw essentially that the CPU first addresses the device, which is too general a term. We will see which part in the device it addresses because there can be many parts which can be addressed. Then essentially it addresses the device – by that we mean that select one of this devices because there can be many devices on the bus as marked here. Only the interface part of it is marked, that is, the I/O_1 up to I/O_n.

There can be any number; this CPU selects one of these devices and then it essentially first checks its status. The information about status of the device will give whether the device is ready for data transfer.

(Refer Slide Time: 01:53)

.


PROGRAMMED I/O

1. ADDRESS A DEVICE

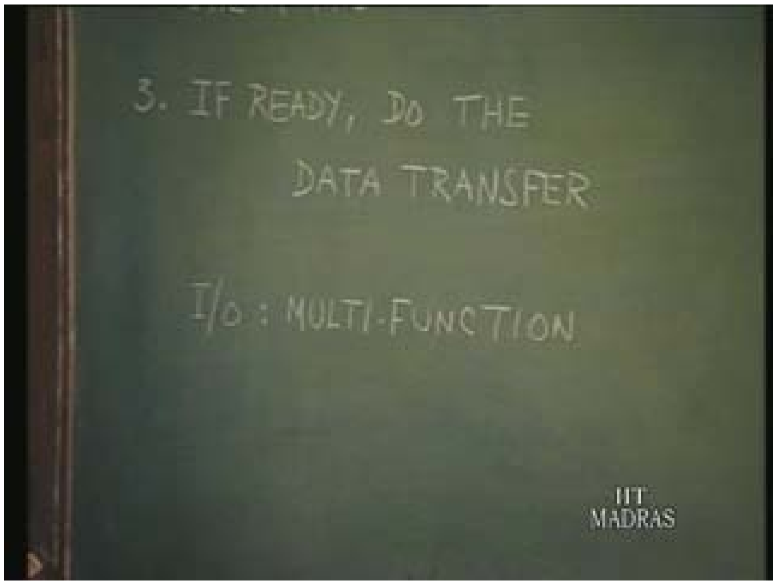
2. CHECK ITS STATUS

3. IF READY, DO THE DATA TRANSFER

If ready, the CPU will do the data transfer. This is essentially the crux of the whole thing about the programmed I/O. If it is not ready, possibly the CPU, when it finds one device is not ready, will keep checking the other devices and so basically the CPU polls among these devices. We also said programmed I/O also means device polling. Now the important thing here is the CPU takes the initiative and does it. So this is important; the CPU takes the initiative. If the CPU does not take the initiative there is not going to be any data transfer. When the CPU takes the initiative, it first has to check whether the device is ready. All these things, that is, addressing the device and then checking whether it is ready or not, do not contribute for the actual input or output because the actual input or output is with reference to the data transfer. We said data transfer essentially can be an input or output; so if it goes from CPU to I/O, it is output; and if it goes from I/O to CPU, it is called input – either way it is a transfer. So we can say that the CPU is tied up with this activity of checking; this is the first mode. Before we go into the second mode of I/O transfer, there are essentially three modes; we will see that. Before that I would like to say a few words.

Basically recall some of what we had seen; we said there would be a status register and one of the bits would be checking whether the device is ready. There can be many more bits in that particular status register – that is one aspect. We also said there will be a data buffer, another register. Essentially the data buffer assembles the data so that at the bus transfer rate, the data transfer can be affected, so that the CPU will be utilized to the maximum. We also saw an example of a 1-bit data production and an 8-bit data transfer between CPU and I/O. How about if the CPU wants to define the function of an I/O? What do you mean by that? In this particular one, the CPU checks the status of the device. Suppose CPU can dictate to the device what it should do, CPU, for instance, can configure or define some function of I/O in case the I/O is a multifunction or a multipurpose. Suppose the particular I/O is the multifunction interface.

(Refer Slide Time: 06:20)

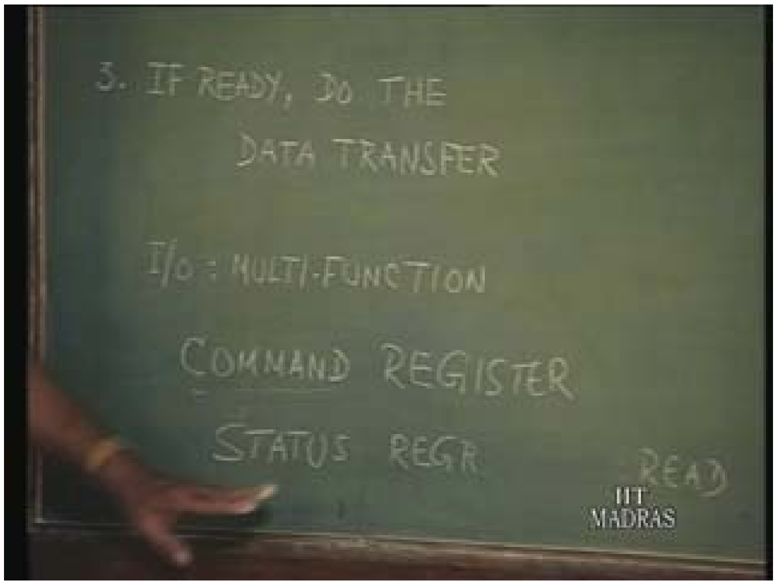


3. IF READY, DO THE DATA TRANSFER

I/O : MULTI-FUNCTION

We will assume a simple thing. That is, an I/O is either capable of transferring 4 bits or 8 bits or 16 bits at a time or possibly something like let us say one part of this; I/O will be configured as an input and another part will be configured as an output. Now all these things would mean the I/O is essentially a multifunction, in which case the CPU can send some code to the I/O and then define what that particular I/O will be doing. That is possible by having status register and data buffer in the interface circuitry – do you remember we saw in the previous lecture all these as part of interface circuitry? Similarly an interface circuitry can have a command register added to it, so that the CPU can issue an appropriate command to tell which of the multi-functions that particular device will carry out at a given time. Then you can see that the status register, which we saw earlier, is essentially for reading the information from the interface whereas a command register is a register into which the CPU can write in, that is, the status register is essentially for reading the information from that.

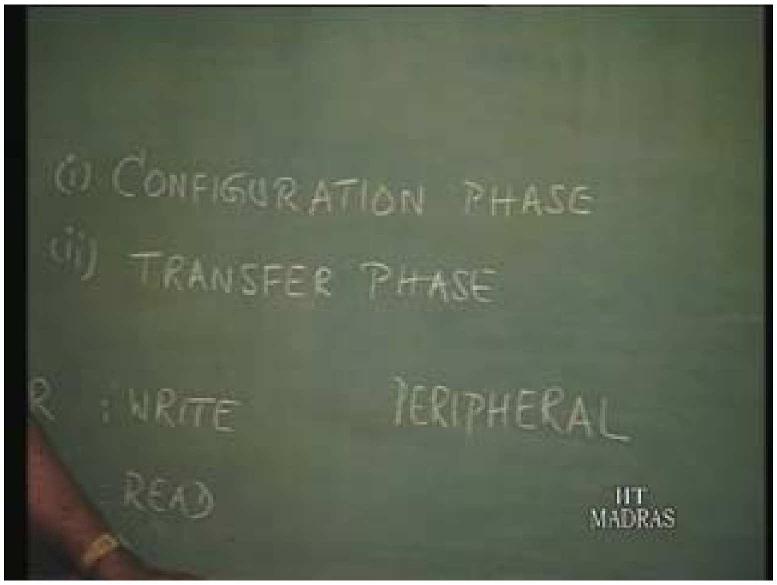
(Refer Slide Time: 08:31)



Sometimes you may have only read only a function from that register. The command register is essentially for writing into. What we mean by that is the CPU writes the command into the register and the CPU reads the status information from the register. Since the CPU and the interface circuitry will either involve in read or write only, some times what happens is this function will be combined and there may be a single register because when CPU addresses, it issues a read command; then it could be reading the status. If the CPU issues a write command, obviously it means CPU is writing some command into it so that physically this can be combined and that can be single register also.

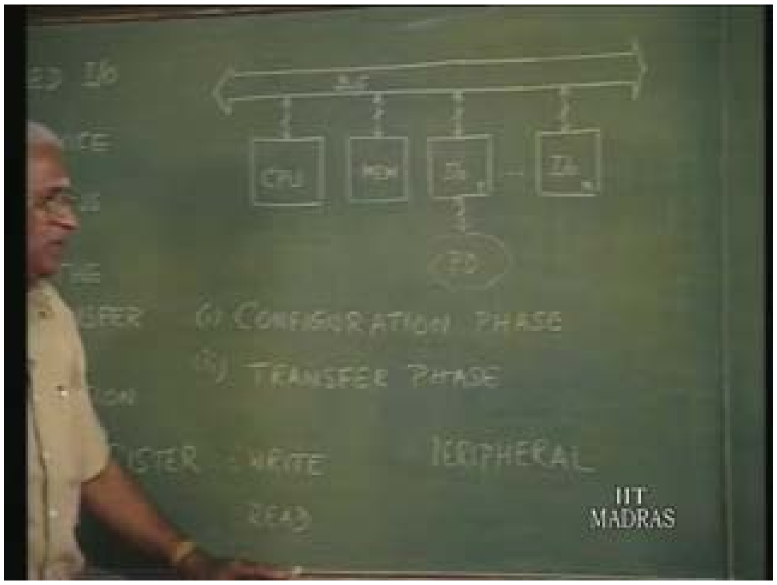
Now if all of an I/O function can be specifically defined, if the I/O supports, we say that the CPU configures that particular I/O. So this phase is called a configuration phase. The CPU addresses the I/O and then defines or configures the particular I/O. It says the function it will be doing. Essentially the configuration is by writing a specific command mode into that register. Now, once the configuration is done, then the next phase is only data transfer. So essentially in any I/O, especially if the I/O is of multi-function type, we have these two phases in mode and in this phase for any I/O device there will be a specific interface chip. Generally that particular chip is called a peripheral chip. For instance, you may have come across the term micro-computer peripheral.

(Refer Slide Time: 11:01)



What is a micro-computer peripheral? Essentially it is nothing but the interface circuitry in a micro-computer system, which interfaces with the bus of the micro-computer system and the device on this side. That is nothing but a peripheral device that separates and this is called peripheral. In fact if you come across the term peripheral, it means only the interface the interface circuitry, the chip part of it, not this; to denote this generally we would say peripheral device, whereas this is called peripheral.

(Refer Slide Time: 11:29)



So the CPU will first configure the peripheral, which means the peripheral chip, by writing a command into it, would, just like we saw in the case of status register in which ready it is in 1 bit, similarly in the case of command register there may be many bits and setting or resetting these bits is precisely what is done during configuration phase. And then once you configure, whenever the device is ready the CPU will do the transfer. This is somewhat related to the input output; so I thought I would mention it before I go on to the second mode of data transfer. What is that one? In the first one, the CPU takes the initiative and checks whether it is ready, and so there is going to be some delay. It is possible that the device was ready for quite some time; it has been waiting and the CPU is not really checking it. So there is a possibility because when does the CPU check? The CPU checks as per the given program which it runs.

There is a program; according to that program only, it will go and check whether the device is ready. So there is a possibility that the device is ready and the CPU does not know mainly because it has not gone and checked; that is why we say CPU must take the initiative and do it. It is up to the system designer concerned to make sure that the low device is idling; the CPU is fully utilized; memory is fully utilized, etc. But then there is always a situation which cannot be foreseen. So the other thing in which the wait period is such that device is ready but CPU is not checking it; there is a delay; or we may say the device is waiting for a transfer. To reduce that, we have the second mode of data transfer and that is called an interrupt driven technique, interrupt driven mode or just interrupt driven I/O.

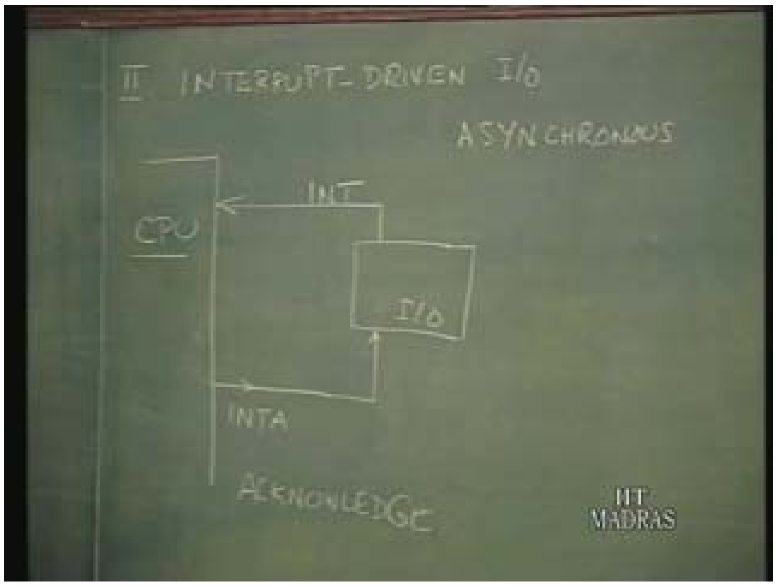
(Refer Slide Time: 14:20)

II INTERRUPT-DRIVEN I/O

As you would have guessed by now, the interrupt is that the device, when it is ready, that is, when the I/O is ready, immediately informs the CPU because possibly CPU is doing something else. So it will be interrupting the CPU and informing of its readiness. So in this particular one, we can say the device takes the initiative and informs the CPU. For this purpose there must be a special line, interrupt line. T hat must be part of this bus. So we may say that CPU gets – only one part of the bus I am showing – the information. It is called INT or interrupts input. So I/O, that is, the device, is ready to indicate that the I/O interface is generating and interrupts CPU.

Now what was CPU doing and what will be its response on receiving this interrupt input? Interrupt is an input signal to the CPU and it is generated when the device is ready. What is the relationship between the CPU clock and device? They are two independent things. So we say that interrupt is an asynchronous signal, meaning it is not synchronized with the clock of the system. Any time the device is ready, the device will generate the signal and there is no synchronization between the clock of this and when this will be generated, any time it can come. Now if CPU is busy doing something, what do you mean by the CPU being busy? The CPU is executing some instruction. That is, it is going though the instruction cycle; what does an instruction cycle consist of? It consists of machine cycle; the machine cycle consists of states. So half way through an instruction cycle, we can say that any time during an instruction cycle the interrupt input can come, it can arrive. When should the CPU be permitted to get interrupted? Because if we just allow the CPU to be interrupted, whatever the CPU is doing may be lost. The CPU cannot respond immediately. We will come to that a little later. On receiving the interrupt request, that is interrupt input, the CPU at some convenient time will generate an acknowledge; generally it is called an interrupt acknowledge signal. So that is interrupt acknowledge signal. This essentially means the CPU has taken note of the incoming interrupt.

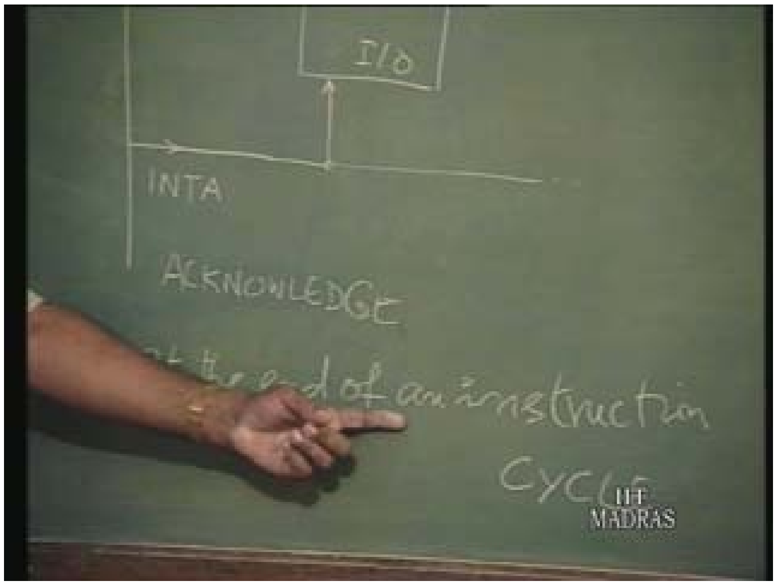
(Refer slide time: 18:16)



So this particular thing should go to the device. Now this INT input and INTA output are actually two signals on the bus; one is an INT input, which goes to the CPU and another one interrupt output, which comes from the CPU. Now these really go over the bus, means they are being watched or noticed by all the devices; it is in public domain. Any device can notice that, is it not? Now suppose two devices generate interrupt, one simple way is you can always have two interrupt input signals and connect one device each, but then how long you can go on multiplying? We will come to these various things a little later. So we will just take that there is only one interrupt input signal being processor and then there is one interrupt acknowledge from the processor. So these actually go over the bus.

This is not only going to one device but it also goes to other devices because it is part of the bus signal. An interrupt request comes under the device when CPU goes and checks; when the device is ready the interrupt input comes. So in other words, the device is taking the initiative and the CPU will be interrupted at an appropriate time, the most appropriate time. When is it? It is at the end of an instruction cycle because at the end of an instruction cycle, that particular end of an instruction cycle would have been executed, and all the necessary information to continue in the next instruction will all be available with the processor.

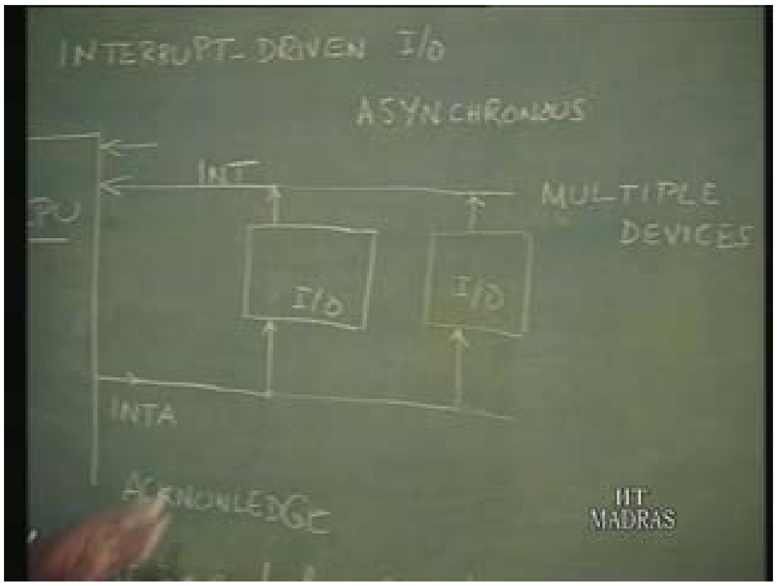
(Refer Slide Time: 20:28)



So really speaking, the interrupt will be acknowledged at the end of an instruction cycle; this is one convenient time. So generally we say that this kind of acknowledging interrupt and a few more things are part of what we may call as service, meaning the CPU keeps looking for these things in anticipation of servicing of some device. This is in fact a service phase. As part of the service phase, it will check whether there is an interrupt which may have come during an instruction cycle for which an appropriate acknowledge must be generated at the end of it.

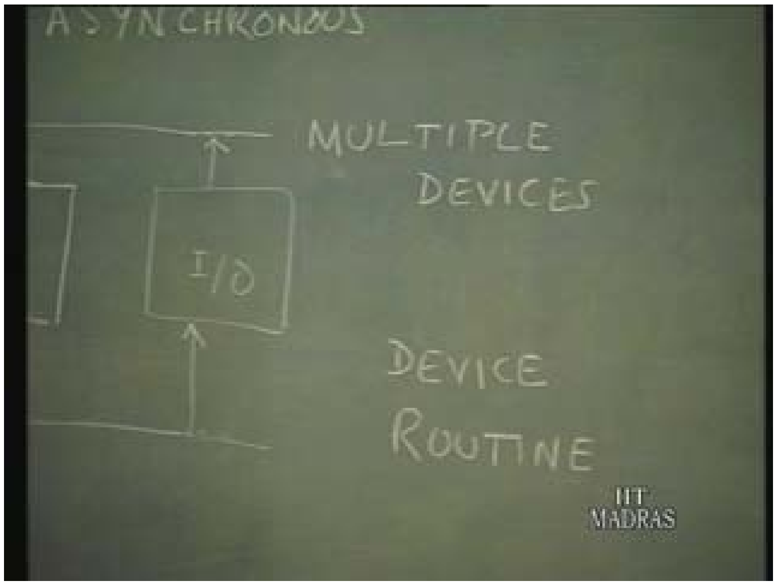
That is the interrupt driven I/O. Now if there is only one device which interrupts, there is no problem. But if there are multiple devices, suppose we have got multiple devices which is a very valid situation and if you just have only one interrupt input, then essentially what we mean is we have another I/O, which also makes use of the same interrupt input. What is the relation between these two? These two devices will be independent. Just as we said earlier when the CPU is doing something the device may be ready and the moment it is ready it will interrupt.

(Refer Slide Time: 22:14)



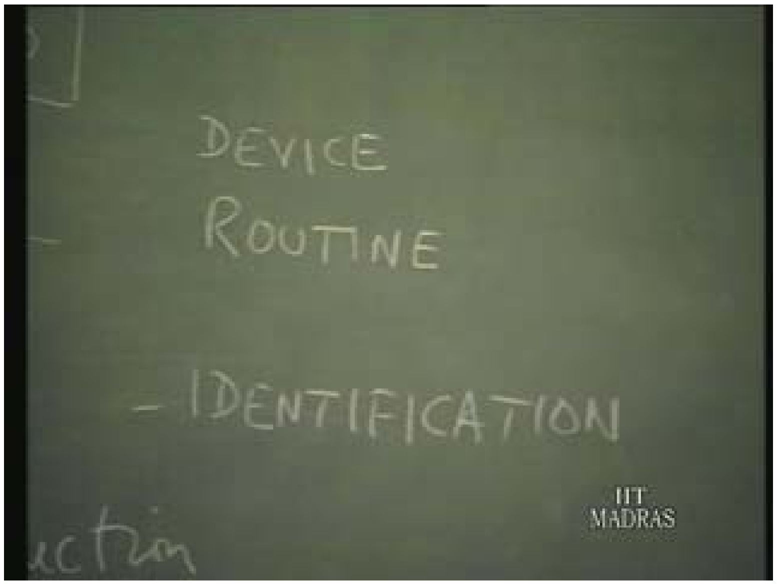
It is also possible in the same way that these two devices being independent may generate the interrupt when they are ready and it is also possible that simultaneously they are ready and both of them are putting a request and this device may be a printer and that device may be a plotter. Whatever be the device and I/O devices being very much different in the configuration function, the way the signal must be sent etc., that is, the form in which the data must be sent to these, the CPU will be having different device routines. For multiple devices, that is, for different devices, it will have different device routines.

(Refer Slide Time: 23:25)



Essentially since these device routines are executed in response to an interrupt, generally these are also called interrupt routine or device interrupt service routine and so on. They are interrupt routines, meaning the CPU was carrying out some program execution; we will call it a main program; that main program has been interrupted and a device must be serviced because the device says it is ready. And this device routine would be different from this device routine. In other words the CPU must know which device has interrupted because these routines are different; device routines are different. The CPU must choose the appropriate routine and execute them, that is, it has to stop the main program and go to the appropriate program associated with that. What does it mean? It means essentially the interrupting device must identify itself. So it is not enough if the device interrupts; the interrupting device must identify itself.

(Refer Slide Time: 24:53)

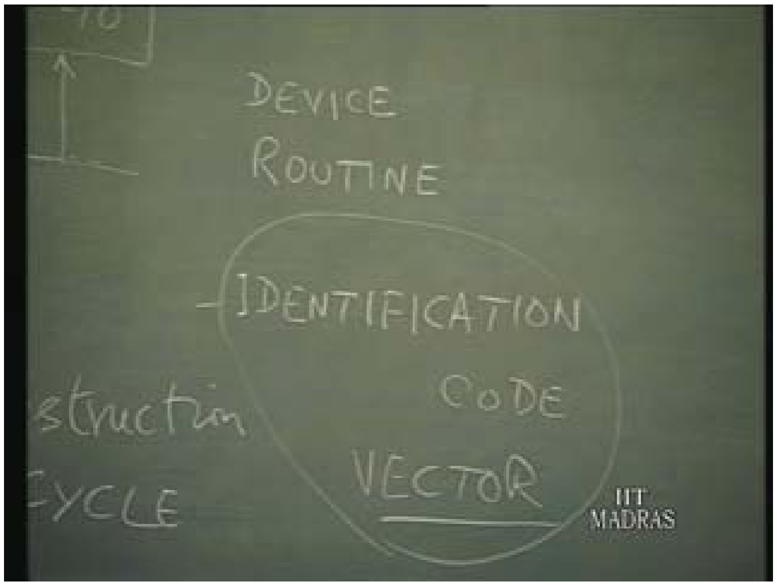

DEVICE
ROUTINE
- IDENTIFICATION
IIT MADRAS

That is, there must be an identification phase. How can identification be done? There are two ways. One is: the moment the CPU gets an interrupt input, at the end of the instruction cycle it can acknowledge and then while acknowledging, the CPU can check these devices, whether they are ready. As we earlier did with programmed I/O, we do it in a similar way, except that the CPU is not checking on its own; the CPU is responding to an interrupt; that is the difference. Otherwise the CPU can check while generating the interrupt acknowledge, that is, while generating the acknowledge, whether this device is ready or this device is ready and so on is one possibility. The other possibility is that the CPU generates an interrupt acknowledge and it is up to the device which interrupted the CPU; let us not worry for the moment when more than one device has interrupted. Suppose only one of the devices is interrupted, then on receiving the interrupted acknowledge the device which has interrupted the CPU can generate an identification code. Generally the identification code is called a vector; what is it? Essentially this particular thing is going to point to where its device service routine is available.

Essentially the whole exercise for this CPU is to know which device service routing must be executed and that service routine is different for different devices. So this identification code, which is generally called a vector, points to device service routine one in this case or the device service routine two in this case and so on and so forth. Because it comes through interrupt, they are also called interrupt service routines or just interrupt routines. This is one way. So a device may generate and put an interrupt request on a common input, and CPU may go ahead and poll among the device which is ready and then service it, or the CPU generates an inter acknowledge and the I/O device identifies itself. Now more than one device is interrupted because there is certainly going to be some delay between the time when interrupt has been generated and interrupt acknowledge comes because only at the end of the instruction cycle the acknowledgement will come. So there is going to be some delay.

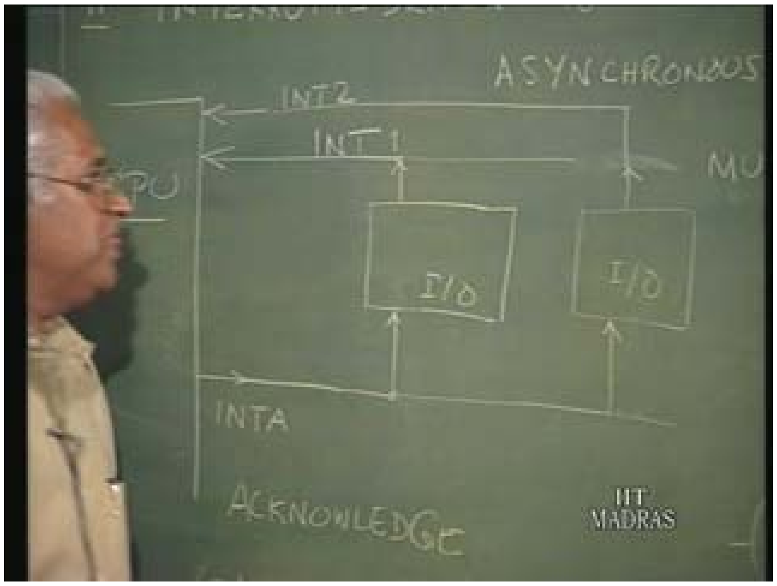
During that period if another device has been ready or let us say two devices simultaneously are ready the situation is same. So first an interrupt has been generated and before interrupt acknowledge can be generated by the CPU, the end of the cycle of another device is ready. Suppose two devices are ready; what can be done? If you adopt the second scheme in which the device is identified on interrupt acknowledge, and if both the devices which are ready will identify, then there will be problem. So there must be some way in which we can resolve among these various devices. So we talk about what is known as priority among these devices, that is, you have to assign some priority among these devices, so that, let us say, this device has higher priority and this device has a lower priority, in which case the higher priority device will be having service first. The lower priority will have to wait; one can keep changing this priority. Now where exactly will the identification code or vector be placed? Essentially this is a code which should identify the start address of the device routine. So the best place for this particular code is generated by the respective devices. So the I/O device places the code on the data bus and CPU reads it in.

(Refer Slide Time: 29:46)



Essentially it is a code which gives the start address of the device service routine; so that particular routine can be started and executed. This is one thing. Another scheme is whatever I mentioned earlier – you can have multiple interrupt inputs. Now here I have shown two different interrupt inputs, in which case possibly I remove this. If I connect this to this, there is no problem when INT 1 is generated, the CPU knows this device is interrupted; when INT 2 is generated, the CPU knows this device has been generated.

(Refer Slide Time: 31:00)

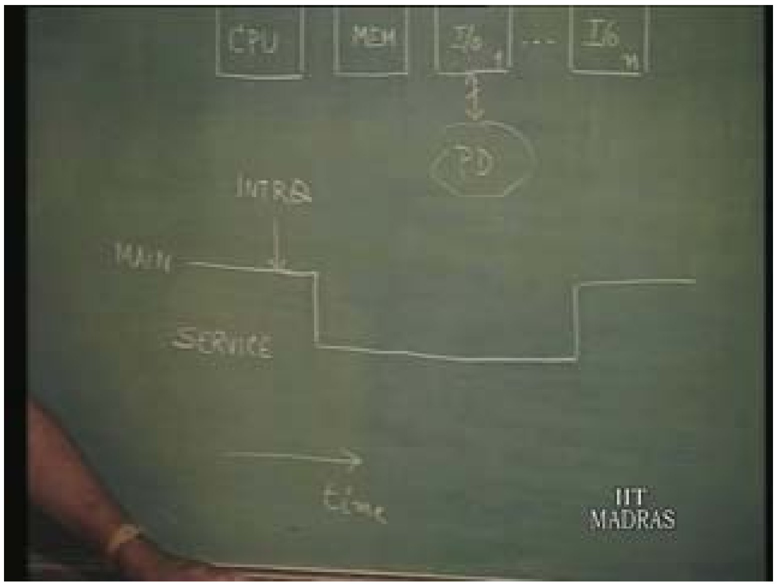


Now you can see that by having multiple interrupt inputs, essentially I have hard wired the information because when INT 1 is generated, this device is identified immediately because there is only one device, at least in the particular scheme. It is again possible that both INT 1 and INT 2 are generated simultaneously because these two devices are independent. So then we must come to this priority part of it. Among these between these two inputs, INT 2 and INT 1, the priority must be defined. That is not a problem. So now I can hard wire the information about this code. The code essentially has start address of the service routine. I can hard wire it by having uniquely one device for one; I mean connecting only one device for one interrupt line. Then it is known by the CPU or alternately, through the software, the device can identify itself the start address of its service routine.

Now as you can see in this particular one, the overhead to some extent is minimized; more important thing is the moment the device is ready, the CPU can be informed and the CPU by itself does not go and spend some time. This way the overheads are reduced but then, in the practical situation, you cannot have many interrupt inputs. So there are overheads involved, because the device must identify and CPU must know which service routine is there and, to that extent, the overhead that is there. But then the response to any interrupt request of a device, which indicates its readiness with the data or its readiness for accepting the data, that particular response can be certainly vastly improved. By having this particular one, we are automatically talking about other necessities such as priority and so on, because multiple interrupts have multiple priorities.

Now we will take a look at the sequence of interrupt generation, request acknowledgement and service – what it all involves and what it means in terms of the efficiency of the processor. Remember one thing: the processor is always doing something or the other, that is, either execute main program or it gets interrupted by a request and then it is going to execute interrupt service routine for a requesting device. Either way, it is just going to execute some program or other: if it is a main program the instruction cycle corresponding to that; if it's a service program the instruction cycle corresponding to that. So essentially, the processor turns its attention from one program to another program. In other words we say there is switch; there is a switch from one program execution to another program execution. Each program is supposed to create a specific context. So we talk about a context switch, meaning the CPU switches its context from main program to device service program one, or main program to device service program two. This context switch is not very much different from going from a main routine to a subroutine. Instead, here we are talking about main program – the CPU is engaged in some program and then it goes to device service program. Now diagrammatically we represent something like this. In course of time when a program is getting executed, as a program is getting executed an interrupt request comes. By this I mean here is there is a time axis.

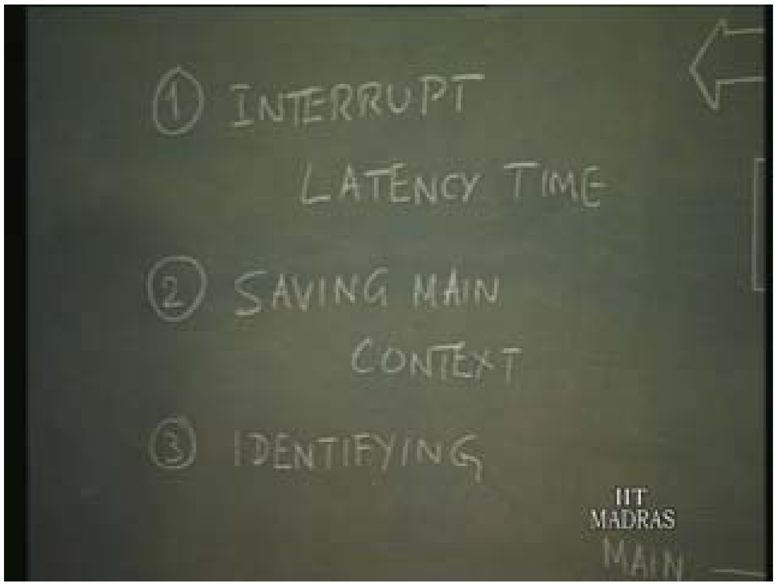
(Refer Slide Time: 37:17)



So some program is getting executed; an interrupt request comes at the end of the current instruction cycle and the CPU turns its attention in a way from the main program. So let us just say this is one related to the main execution, from this it has to go to execute a different program, that is, the device service program and that end of it comes back to the main program and resumes its main program. So the CPU is involved in executing some program and then it goes into execution of some service program, which is, servicing that device. Now you can see that different types of things are involved here. First one is the interrupt: any program execution consists of taking an instruction one by one and doing it. An instruction cycle consists of machine cycles and states as we know. So an interrupt request comes during an instruction cycle, at the end of the instruction cycle possibly.

The instruction cycle was started here, let us say, and during the instruction cycle the request comes and then at the end of the instruction cycle, when the CPU goes and sees whether there is any pending interrupt input request, it sees that there had been one and then it changes. Now subsequently, the main program must be resumed and continued. So all the information related to the main program execution must be stored. These things will be usually stored in the stack area. All these things will have been stored. So first we say that at the end of the instruction cycle, there is a switch; certainly the main from execution is stopped – that is number one. Whether the service program can be taken up or not, first it is stopped. And the first thing that will be done is the CPU will spend some time and then store all the information related to the main program execution so that it can be used later on. Some time before or after it will have to identify which service program must be taken up because there can be many devices. That is, we may say identification of device and then actually it executes the device service only for some duration and this is precisely what we may call device service. At the end of the device service, the device which interrupted is serviced. At the end of it, whatever was saved with reference to the main program will be restored so that the main program can be received safely. So you can see here that here in this particular thing, if I mark it as 1, then 1 corresponds to what we may call as interrupt latency time. That is, interrupt has come, but the CPU is not yet ready to respond; some time will have to be spent.

(Refer Slide Time: 42:22)

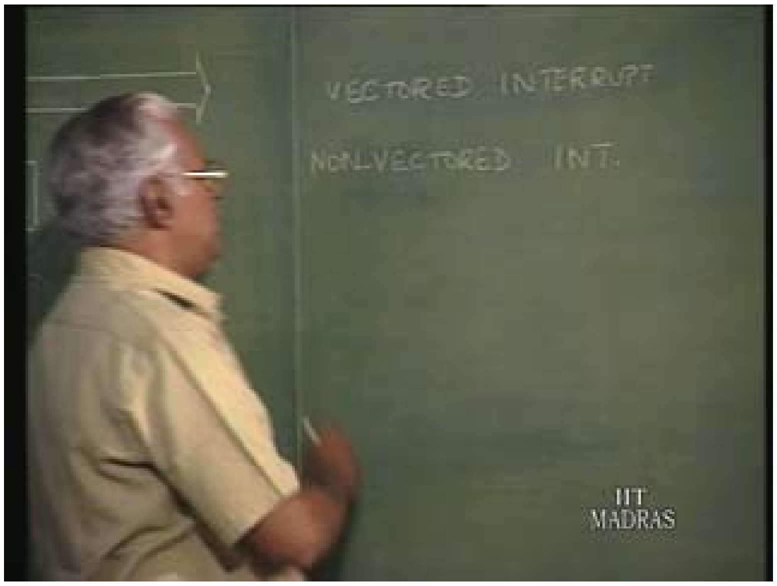

(1) INTERRUPT
LATENCY TIME
(2) SAVING MAIN
CONTEXT
(3) IDENTIFYING

IIT
MADRAS
MAIN

So this whole thing is instruction cycle time. We may call it as last instruction cycle before the contact switch not exactly the whole of this, not for this from here to here. This is what 1 must be; that is, from the time the interrupt request has come to the time context switch takes place or the end of the particular instruction to the end of the particular instruction cycle. So that is interrupt latency time. Here if we mark it as 2, then you can say that it is for saving the main context. First thing is the program counter content, which indicates at what point of the main program it stopped and then the set of registers because the registers are part of the CPU. The CPU registers may be taken up for other program. So they also have to be saved and any other status related information for the main program has to be saved. They have to be saved some where.

Then 3 – we are not very sure about specifically it what order it may come. Possibly let us say 3 is for identifying device. This identifying device indicates which service routine can be taken up. About identifying the device, I was talking about two things: one is that device interrupts and when it interrupts it identifies itself I said through a code, so at that time of generating interrupt itself, it identifies. Then we said the vector is very much a part of the interrupt request itself, and so this particular thing is generally known as vector interrupt.

(Refer Slide Time: 43:47)



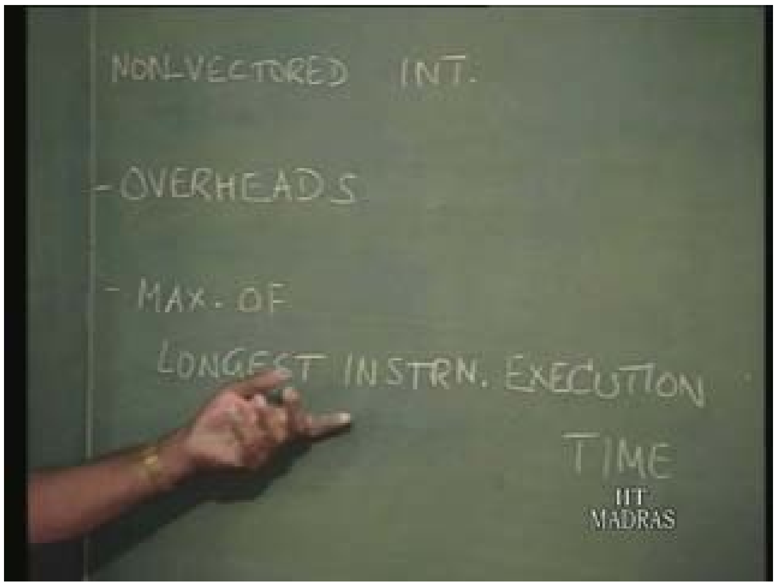
There are many ways in which it may be done. For instance by having a unique input signal line on one pin or connecting one device, when an interrupt input comes to that particular input pin, then the CPU knows where exactly the service program is for that particular device. The other one is just not known at the time of interrupt, but then later on the CPU will have to ask the device or the device may tell, in which case we call the second type of interrupt as non-vector interrupt, meaning at the time of interrupt request it is not known which is the actual start address of the device service program. There has to be some way in which the device must be identified by the CPU because only then the CPU will know which program to run. Once it knows, and then fourth is the device service – from spanning from here to here. This is the main thing as far as the interrupt service is concerned. So we can call this is an interrupt service and this interrupt service as I said is something like your subroutine.

(Refer Slide Time: 44:43)

Running a subroutine: main routine is here; interrupt service routine is here and then, at the end of this particular phase, that is, when the device has been serviced, or at the end of device interrupt service, this particular one goes back. Restore means all that has been saved so that the main program can be resumed at the end of it. So, the fifth one will be restoring main context. This context itself is used for giving enough information about the state etc., needed for running a specific program. The context switch contains of all the register etc. that must be resumed. So what ever had been saved will be put back here. At the end of it, the main program can be resumed here; so resuming the main program is as simple as this. Now you can see here that in intervals 2, 3 and 5, the CPU is not really involved in executing either this program or this program – the main program or the service program. So 2, 3 and 5 really contribute to what we can call overheads.

Overhead is basically preparing the CPU so that it can take up a service of the device. But then it cannot be avoided. Now compare this with a programmed I/O. In the programmed I/O the CPU by itself will go as per the program written because the CPU goes and checks. So it is possible that the device is ready but the CPU is not checking it; that has been avoided here. But still there is some time. Now in this second one, the interrupt driven I/O, you can see that still there is some time involved. There is a delay. We do not know; may be it is only a few nanoseconds. The best way to calculate the particular one is to see which one of those instruction cycles consumes the maximum time. That is, the worst case delay for a given processor will be number 1, which corresponds to interrupt latency time. So the worst case interrupt latency time is nothing but the maximum what is called maximum number of states, the longest instruction.

(Refer Slide Time: 48:59)


NON-VECTORED INT.

- OVERHEADS

- MAX. OF

LONGEST INSTRN. EXECUTION TIME

That is, the maximum number of states is to be multiplied by the time involved. We take the maximum or longest time that is taken, or the longest instruction execution, because certain instructions will be executed very fast. Basically we have to look at the instruction which takes the longest time. In fact this we have to say is time. So we look for that instruction which takes the longest time. This in fact corresponds to this and that some times may be even in a micro second or two or three. We do not know because it may not really be so, but it is possible that when the interrupt request comes, the misfortune may be that the longest instruction execution time is what the instruction cycle consists of. So there is going to be some delay; now the problem comes.

In case the device is very fast, normally we talk about device being slow and that is what happens: if the device is so fast that it cannot wait even for this particular latency time, what is to be done? Nothing can really be done because essentially what we mean is the processor is slow. In the case of interrupt, this is the only way it can be done. You have to switch from one context to another context, and that would involve overheads; there is no other way in which you can reduce that. So if you try to reduce, you just cannot reduce it; that is the problem. Another thing is this particular one. In case we have a device which is very fast, it means by the time you can switch from this to this if the device loses the data, then the data is lost forever. So in the case of very fast device interrupt scheme will not work and that is the reason we go for the third mode of data transfer, which we will see in the next lecture.