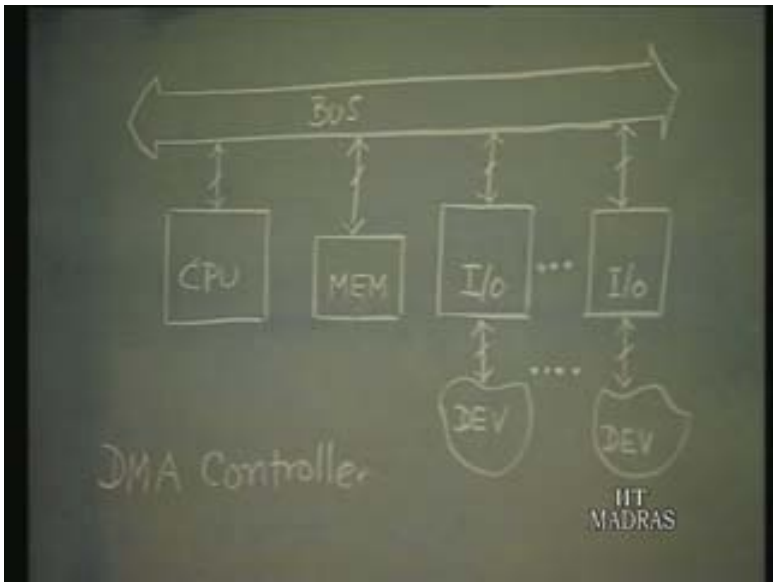
Computer Organization
Part – III
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Input/output
Lecture – 28
Evolution of I/O

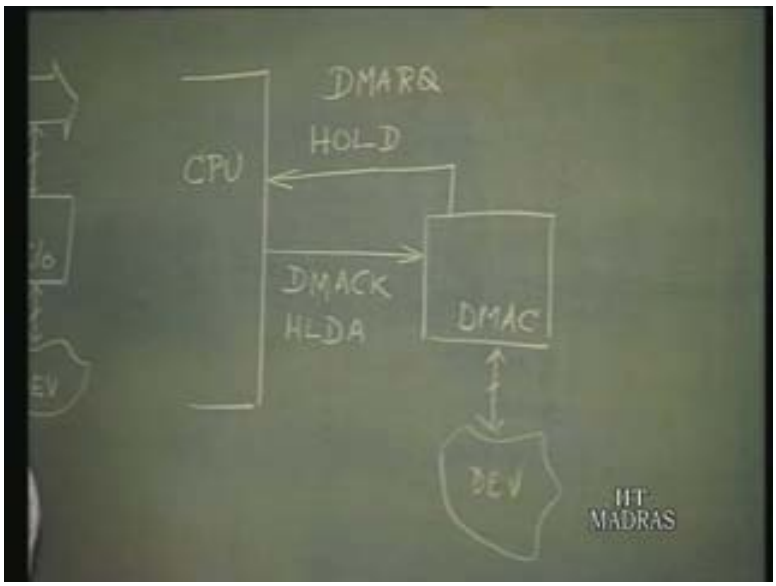
Continuing with the discussion on device service routine before we take an integrated look at evaluation of I/O we will start today's coverage with the one that is still pending; that is, the DMA or direct memory access.

(Refer Slide Time: 02:38)



We had earlier seen what sort of program segment or software is needed – in fact it is only a small segment, two lines or three lines – for programmed I/O on interrupt service. Now let us take a look at DMA. We know that in the case of DMA the processor is not involved and it is, in fact, cut off from the system. So the device will directly access the memory. Now in the case of input, the device will access and then it will be putting the data in the memory. In the case of output device from memory, the data will be routed out towards the device. So the I/O interface of the controller that takes care of the device interface will, in that particular one, be essentially the direct memory access controller or just DMA controller. So we had seen that the processor is not involved; in the case of programmed I/O processor is very much involved; in the case of interrupt processor is asked to be involved; and in the case of DMA, the processor will be informed of the arrival of a very fast device and very fast data transfer. So the device, when it is ready, will inform the CPU obviously over the bus lines. So we will just take a restricted view of some of these signal lines.

(Refer Slide Time: 05:05)

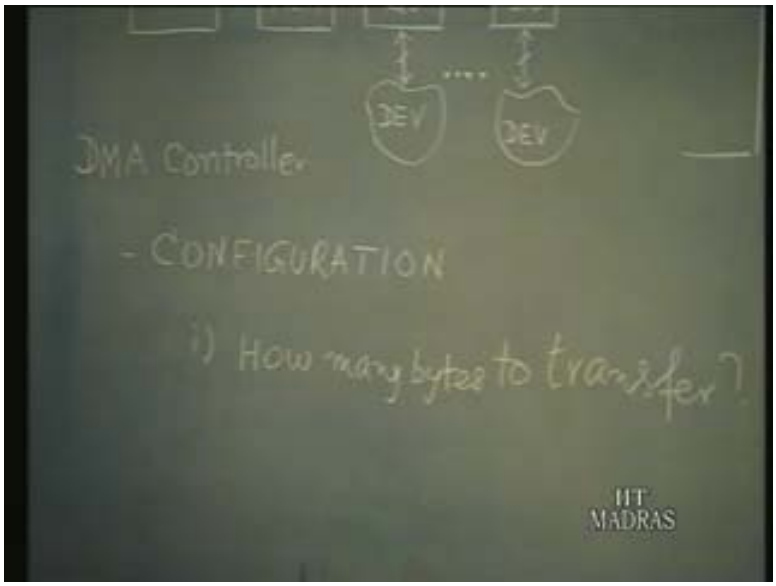


I will keep switching between this and that. So there is going to be some request, very much like interrupt. In the case of interrupt, when the device is ready, the device interrupts the CPU and the CPU is involved. Now in the case of the DMA, the CPU will similarly be interrupted, though it is not called interrupt. It will just say keep off; in other words, hold off from the entire scene. This is one name for the DMA request; hold is nothing but a DMA request signal. In some places it is called just hold. So the device which is ready is in fact as I said will be DMA controller and the I/O interface, so this will generate when this particular device is ready and the CPU will have to relinquish the bus. That is the CPU may be doing something. So it has to be allowed to be completed up to some point and then the response from there will come. In other words this may be called a DMA acknowledgment. In some cases, they would say hold acknowledged; if you use hold as the input then this may be called hold acknowledge.

We just have different names for the same thing. So a request goes to the CPU and the CPU responds; it is similar to what we had in interrupt. In the case of interrupt, interrupt request went to the CPU and then the CPU sent an interrupt acknowledge, except that this is fast and so the subsequent one is not going to be involved. That is, the CPU is not going to be involved – that is the difference between these two; of course, there are minor differences. Hold is another input to the CPU, which comes over the bus and hold acknowledge is another output from the CPU, which goes over the bus and it must go to this. Now like we had seen in the case of interrupt, here also there can be more than one device requesting but generally it may not be because what we are talking about is very specialized situations when you have very high speed device and so on. It is possible that there may not be as many. So one has to also bear in mind that there can be more than one DMA device – that possibility is very much there. There can be more then one DMA device. Now if this is a situation, what is the kind of programming that is involved in this; that is, the device service routine? After all, the CPU is not involved in the service at all; the CPU is cut off from the bus. So it is the DMA controller, which takes over the role of CPU and masters and sits as the master of the bus and then regulates. So before the CPU can relinquish, if at all, something must be sent to the DMA controller.

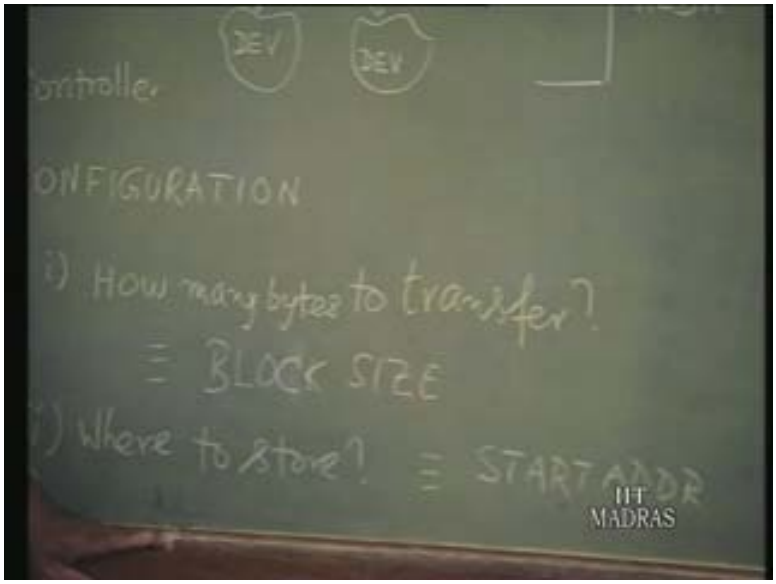
Something must be sent to the DMA controller before the CPU relinquishes the bus because once the whole input goes and then CPU relinquishes the bus, there is nothing the CPU can do. So you can see that this particular device service routine will be subsequently run by the DMA controller, not by the processor because processor is out of the picture. But then, the CPU as the general master of the bus, must have indicated to the DMA controller earlier. You can expect possibly 1 KB of data or 10 KB of data, and when that comes, it has to store it, starting from say location 1000 or 10000 or whatever it is, some address. We have seen this. Essentially, this particular device service routine is mostly during the configuration phase. We have always talked about two phases during IO: one is configuration phase and the other is actual data transfer phase. So during configuration, these two things must be done; that is, we had already mentioned this earlier. First, how many bytes to transfer, assuming it is only bytes how many bytes to transfer.

(Refer Slide Time: 08:47)



In other words, what is the block size – it is the same thing as saying what the size of the block of data is. Where exactly it is to be stored – in the memory or where to read from the memory? I always say store mainly because I always have input device in mind. On the other hand, if you are thinking of output device, obviously it will be read from memory; memory is involved in storing and we are talking about reading from the memory and outputting into the device, in the case of the output device. In other words we can also say what exactly is the starting address of this particular block? That is, from where is the first byte to be stored? Sometimes we may say start address or the start address of the first byte and then the extent of a block size – how far it will extend.

(Refer Slide Time: 09:57)



These things must be stored in the DMA controller and that will be done by the CPU during configuration phase. Now we always said I/O consists of configuration phase and data transfer. Since the CPU is not involved in the data transfer, CPU's job is over with the configuration phase because that is all the CPU is involved in. As part of the configuration, the software segment would be something like we will follow the format we followed earlier MOV destination comma source – that is the format.

(Refer Slide Time: 12:40)

So we will say MOV must go to the DMA controller; I am assuming a command register is there in the DMA. A command register is there and specifically within the command register I may have to move to the part which is going to store the block size.

So move to this command register; well, whatever is the size of the block assuming that I will have to really bother about the syntax of that. So I will just say number 1000, which means 1000 bytes. This is the syntax we just follow arbitrarily; MOV number 1000 to block. Basically this would tell the DMA controller that up to 1000 bytes can be transferred and the other part – start address – so MOV start address; move this start address to the same command register, but to the part, which will hold the address. Essentially you can see that the CPU is writing something into the command register. When you write something in to the command register, the CPU is actually issuing a command and defining certain parameters. As far as the CPU is concerned, essentially these are the two instructions basically in preparing for DMA transfer; that is all. The DMA controller must do the rest of it. So in the case of the DMA, a study of the DMA controller is more important.

For instance, we know that like any other interface chip, this controller will have a command register, status register and so on and so forth. There may be other things also; for instance we may be having commercially available DMA controller chip, which can take care of four devices at a time. Generally these things will be called channels; that is, a four-channel DMA controller; that means basically I have just shown one device here; instead there will be four. When any of these four devices puts up a request, rather it is not a device request; it will be like a block. So for four devices connected over four channels, there must be four sets of command registers; some of them may be common. But then certainly block size of the starting address will all be different; so there must be four sets and then possibly there will be four blocks. For instance the device may store 1000 here.

Let us see; 1000 means basically the DMA controller will be having a buffer size of 1000, in which case while the actual transfer takes place with the memory, this is cut off. The DMA controller will be involved with the memory here. So there in the actual DMA transfer from the buffer, it will go to the memory in the case of input device. But on the other hand, if the device is very fast then there will not be any problem. This it will keep collecting the data and then passing on to the memory. So any way these are all some variations. When you have four channels, we have come across the situation earlier. Let us say if there are four devices interrupting, then we talk about priority. The same issue comes here too: you have to locate the priority among the four channels. Normally this will be built in, saying channel 1 is of higher priority than channel 2, which is of higher priority than channel 3 and so on and so forth. But then, one can change the priority also. We had talked about this earlier in the case of interrupt – that is called software definable priority. These are all certain additional things.

Now we get back to the program that is needed. So the DMA controller is set up by the CPU with the issue of these bits of information and that is part of the whole show. Afterwards when the device is ready, the DMA controller will indicate to the CPU its readiness and then at a convenient time, the CPU will stop processing and will generate the acknowledge; on seeing the acknowledgement, the device will start doing the data transfer between the device and the memory; that is why it is also called direct memory access; the memory is accessed directly for the DMA. This is as far as the data transfer is concerned. In the data transfer, this is the one which is going to do it and it will be very fast; as I said earlier, it will be something like a machine gun. May be one or two clocks will be enough; whereas here one may have to interrupt the CPU; the processor may have to execute a few instructions. After a few instructions, it may have to check the status and the readiness of the device as well as any error committed during transfer, etc.

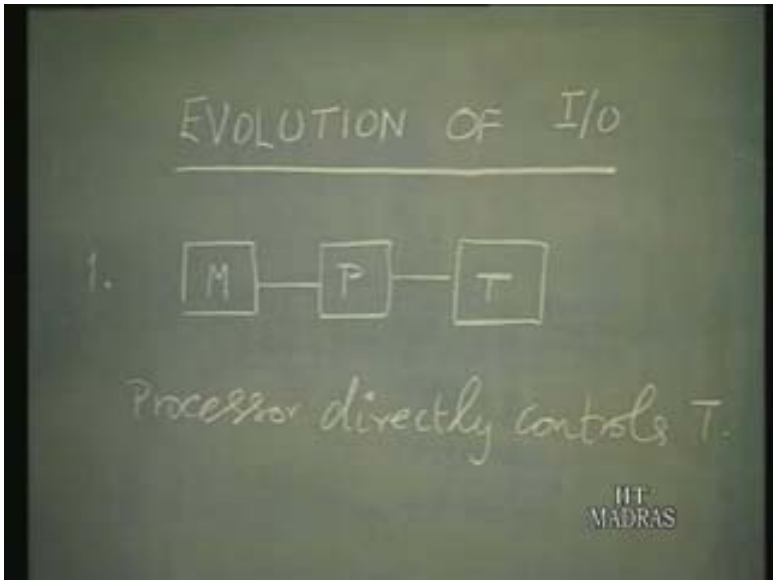
There is no room for such errors here. In case there is an error, the whole block is going to be erroneous. So we place the address and send the data; place the next address and send the data – it goes on this way. Starting from some address it adds on, so it is +1; +1; and so on; it goes fast – that is why it is used for very fast devices; fast in the sense that the CPU cannot other wise have handled. That is why now you can see that the DMA controller or the I/O interface takes on the role of a processor; there is no master because the CPU is not minding the bus. It is not mastering the bus so CPU is also not minding the data transfer. This is in fact a very good point; the next thing we discuss is the I/O evolution. As I said, this is a nice point in time to trace the evolution of I/O.

Let us get back to our original diagram. You have the CPU as the master of the bus, then you have the memory and the I/O. We had taken a look in our first part of these lectures at the processor and in the second part at the memory and then interaction of the memory and the processor. Now, in the ongoing third part, we just looked at the essential or basic three modes of data transfer, which involves the I/O and we noted that programmed I/O is interrupt driven and then the DMA. Progressively you can see the CPU was involved in programmed I/O, the CPU was involved even in checking whether the device was ready in programmed I/O. In the interrupt driven case CPU was involved but then only when the device indicates that it is ready, and then, in the third one the CPU was involved only to the extent of shutting itself off; that is, when the device says it is ready.

The CPU goes off the bus and then the I/O controller, specifically the DMA controller itself, takes over. Now you can see that the sophistication of I/O has been increasing. Earlier for the programmed I/O, it was enough if the CPU just checked that; this was a passive one; it only responded to the processor. Then in the interrupt case, it became active; in the sense that when the device says it is ready, it will actively inform the CPU that it is ready. And in the third case, we saw that it became really more active: it not only informs CPU but it also takes over the function of CPU. So this in fact is why I said the evolution of I/O is a nice thing we can talk about at this point in time, mainly because we might say that the various stages in the evolution of I/O have contributed over years, over decades, in fact to the present state of art of computing. Now let us take a look at it how is it.

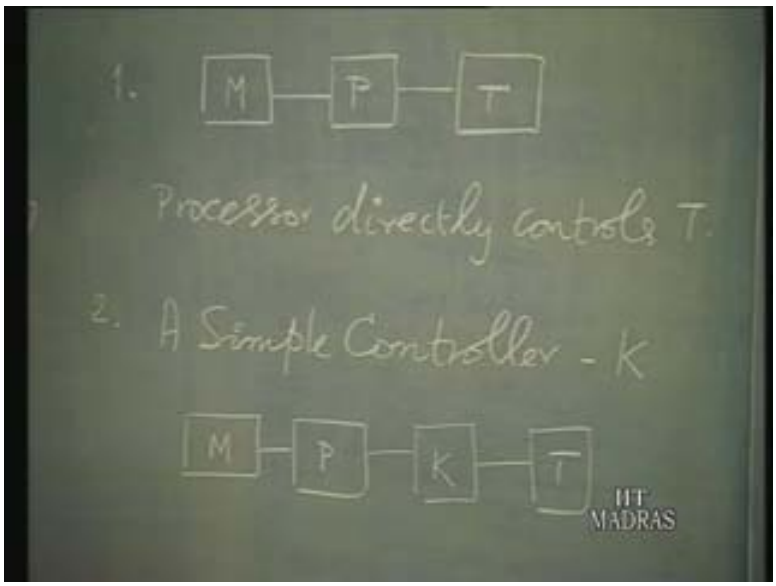
There are something like about eight or nine stages of evolution. I will trace part of it – in the first stage, there is the pre-computer historic age or just dawn of computer age. Those days it is always processor, that is, the CPU, the memory and the I/O. Essentially what is I/O? I/O consists of a few terminals or, in process control application, it will consist of a few transducers, which change some of the analog signals into digital and then communicate. Either way, we can call this T for transducer or terminal or whatever; so you can see that essentially P, M, and T – these are the three main elements of any computing system. In the earlier days, the processor was directly involved with the T. How? It was somewhat like this: the processor was not only interacting with the memory but the processor was also involved with the I/O; for instance, in programmed I/O, that is what we are seeing. So in the first stage, we may say the processor directly controls T. This cannot be avoided – processor interacting with memory – anyway, why should we avoid? Both of them are anyway working at electronic speeds; there is no problem.

(Refer Slide Time: 23:46)



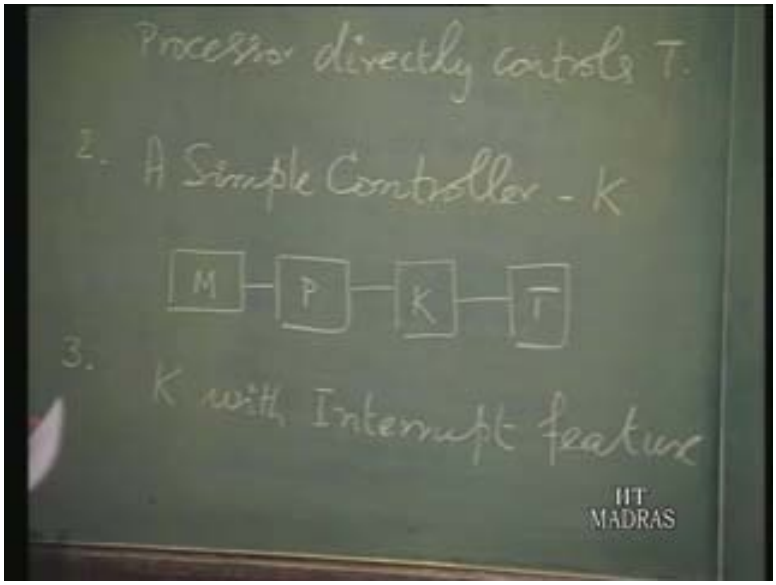
But generally the problem here is because the processor works at electronic speed and mostly this will be at the set speed [24:08]. So the first stage is one in which the processor directly controls T or even in the case of programmed I/O that is what the processor does; P is for processor. The processor checks whether the terminal is ready or device is ready and then affect the data transfer; so it directly controls the T. In the second one, a simple controller gets introduced. As the processor is not directly involved with T, a simple controller is introduced.

(Refer Slide Time: 25:25)



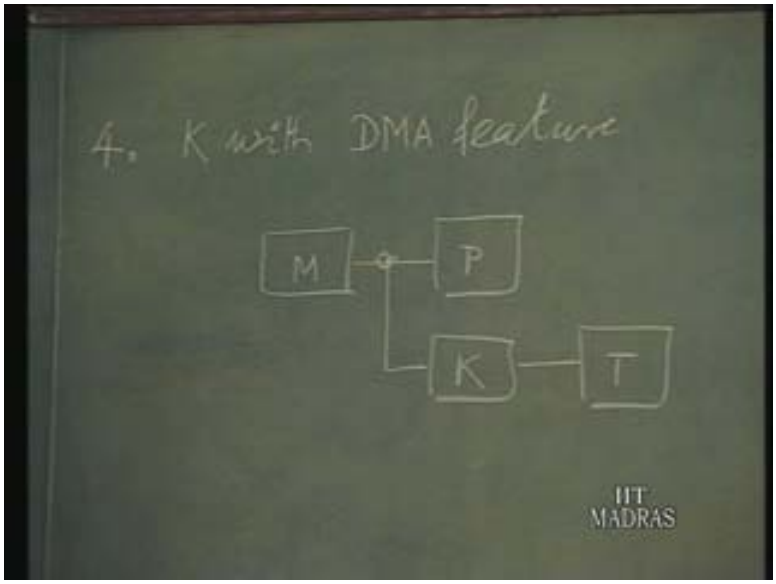
We will just call it K; in the sense that we have the memory; of course, that cannot be done away with, and we have the processor and between the processor and the terminal or the transducer, we have a simple controller if this processor is a very general purpose processor. This is specifically as far as the I/O part is concerned, but it will be very simple one. Now we can say this way, this I/O interface can be considered as the K. That is, in other words, originally this I/O interface possibly was part of the processor itself, with some extra circuitry. Now you are slowly separately that. You can consider that as K. So now you can see that the idiosyncrasies of this device can be hidden here, and at this end, only some standard protocols can be established. So in this way, a simple controller K is introduced. In the third stage of evolution, we can talk about the simple controller, which becomes a little more sophisticated – let us say, the controller K with interrupt feature. Now you must be able to see that earlier the processor was checking with the controller and seeing whether the controller says that the device is ready – that is how it was doing.

(Refer Slide Time: 26:58)



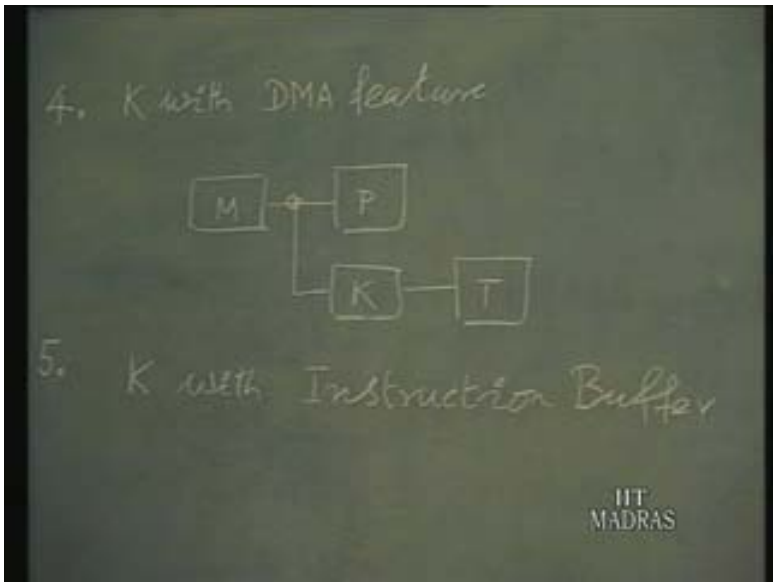
Now the controller becomes little more sophisticated in that when the device is ready when T is ready when the terminal is ready, the controller informs the processor through the interrupt. So this is the third stage of development. Then the fourth may be you are able to guess yourself what that is. The controller K with a simple controller programmed I/O; now interrupt for the DMA. Here we have K with DMA feature – here the controller not only informs the processor when it is ready, but the controller can also access. How is it possible? What can it access? It can access the memory. We have to slightly modify the memory M and the way the processor deals with it. We have the controller K, which helps T access the memory, if necessary, directly. Earlier, in the case of interrupt also, when the device is ready the controller will inform the processor, then the processor executes the interrupt service routine and moves the data from the device to the memory if it is input device.

(Refer Slide Time: 28:38)



So the processor was involved even in this case whereas in this case, the processor can be removed and the controller can directly access the memory so that between device and memory the data transfer can take place; depending on input output the direction will change. What is the next one? The next one would be this K controller becoming little more sophisticated or complicated with some more complex features.

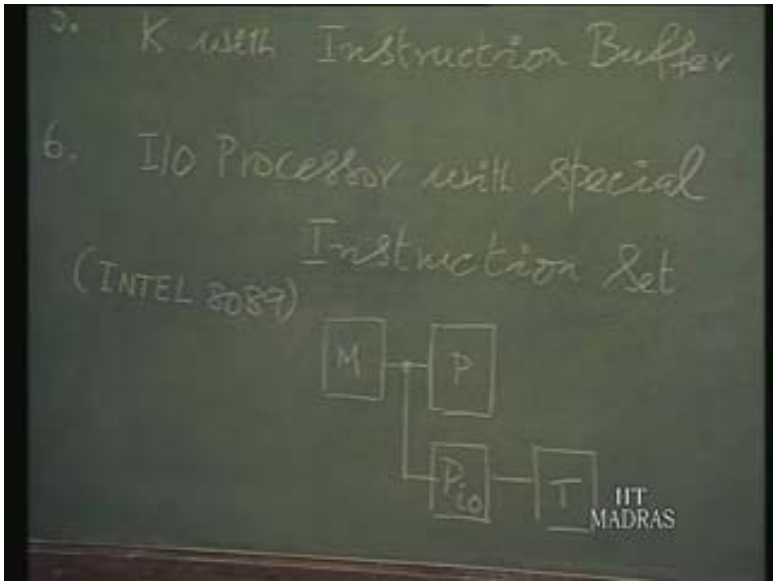
(Refer Slide Time: 30:56)



First it was the processor, then, with the introduction of the controller, we keep seeing that more and more features are added.

So obviously, the next one will be that the controller has come with not only direct memory accessing capability because just a few minutes back we were discussing the case of DMA, where the processor informed the controller the size of data and the place to store. Just as per that, the access will be taking place and the data transfer will be taking place. Suppose the controller has a few more instructions, can execute a few more instructions, then slowly it can take the role of the processor itself. So now you can say controller K has some kind of an instruction buffer; in fact that instruction can as well be issued by the processor prior to the access. During DMA, a simple data transfer takes place, and before the DMA takes place the processor issues some commands; rather we can say issues some instructions. Now in this particular case, the processor issues quite a few more instructions and the controller has the instruction buffer and during the particular access it will execute those things. This is a little more complex situation. In the next stage this particular controller becomes almost like this processor, because earlier we were doing DMA then we said it executes some instructions – we did not elaborate on what instructions – it will be capable of executing some more instructions. Now let us say it is really executing some instructions of its own. That is, the processor may have its own in-built instruction. We can call this particular thing as the controller becoming an I/O processor. So we can say it is an I/O processor with special instruction set; that is, it has its own instructions.

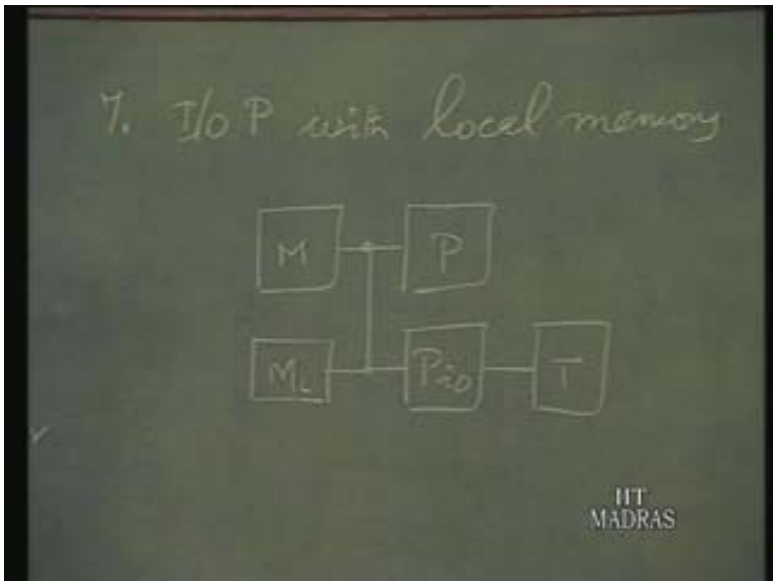
(Refer Slide Time: 33:46)



In fact, chronologically speaking, this has happened in the case of Intel 8089 and its I/O processor. This is a classic example of Intel when its 8086 came as the main processor, this K became an I/O processor; that is, K became more complicated and thus an I/O processor. Intel 8089 is a classic example of this. so now what is this? guess change some what like this a little what memory then you have the processor that is the general purpose processor let us say eight zero eight six if it were Intel then this becomes an IO processor; that is, a controller becoming an I/O processor involving in data transfer. This is a general purpose processor; we can call this also channel processor. In the case of mainframe, this channel processor is very much in use and these things were realized in the case of microprocessor also. That is how it came. So the processor is essentially concerned only with the main processing and as far as I/O is concerned, it will just churn out some data and hand over saying what to do and then the I/O processor will take care of the rest.

The situation is not very much different from here except that here the processor continues to remain the master of the bus, whereas here it is something like the I/O processing function has been delegated to this and it takes care of it. In this seventh stage what do you expect? The controller was becoming more and more complicated; then finally controller became the I/O processor and in fact it is very much sitting like a challenge to the main processor itself though its function is restricted to the I/O. So what would the next one be? When we look at these developments, what comes to your mind? Some more resources are provided for this processor; that is, the I/O processor or I/O P comes with its own local memory in the sense that the main memory is there, and then you have the main processor. Earlier you had the I/O processor, which was essentially dealing with the terminals. Now we have a local memory, that is, I/O processor has its own memory.

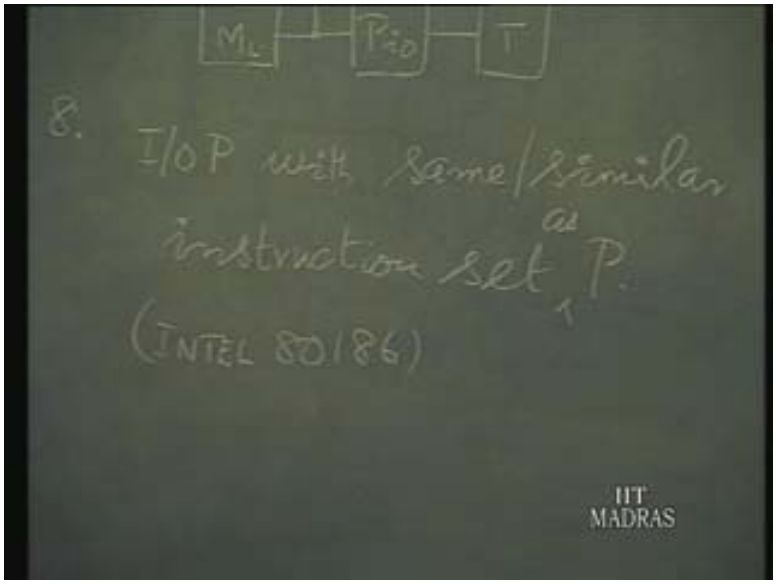
(Refer Slide Time: 36:18)



Now if we just look at this part, the bottom part, does it remind you of something? What is this? It is the same thing – so it is very much like natural biological cell growth, you know, some cell dividing and then growing by itself. In fact, now you can call this whole thing I/O; for instance the bus may be called an I/O bus. So you have the main processor, main memory I/O processor and I/O or local memory, local to that. This memory and this memory are not going to be very much different, in the sense that the processor can churn out a bulk block of data and can carry out some processing and transfer to this memory. Then the processor deals with this and goes on; so this acts very much like a buffer now. And whenever this one has done and this memory is empty or whatever was there has been already input or outputted, then the processor will start filling it.

So the I/O processor has local memory. When we talk about local memory as I said this whole thing can be considered as sitting on I/O bus and this as the main bus. That is how even buses have grown we will have some bus for very fast thing and then some low speed bus for I/O. So bus evaluation is also very much involved with this. In the eighth step what do you expect? The I/O processor is becoming very powerful, so actually that is what it is. The I/O P or the I/O processors have the same or similar instruction set as the main processor.

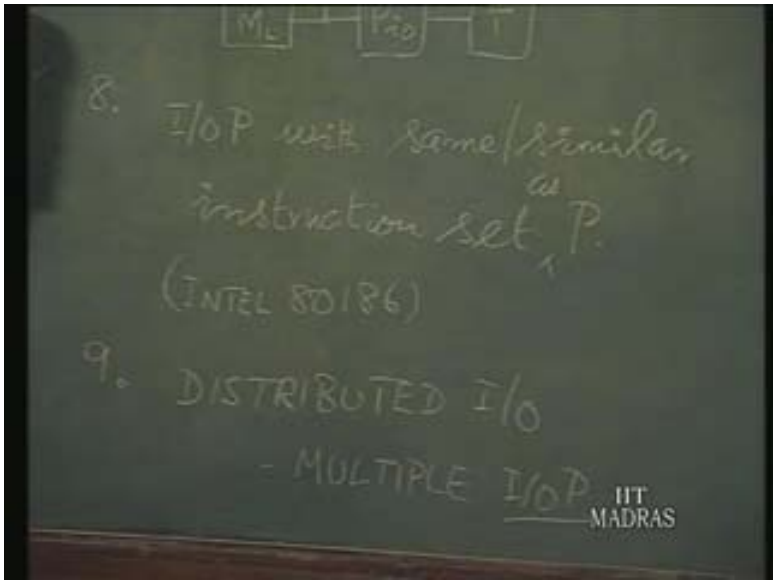
(Refer Slide Time: 39:35)



Is it just an imagination or did it really happen? It had happened; in fact, in the case of Intel 8089, it had happened. In the case of Intel family first 8089 came and this particular one is nothing but Intel 80816. Although that particular processor was as a general purpose processor, it became quite useful because of certain functionalities as an I/O processor. So even to this day, you will find 80186 based device controller boards, which will take care of multiple devices and so on. Now there is no difference between this and this. For instance you can have an 80286 or 386 or 486 memory and then here 80186 will be sitting; it may not be very powerful. It certainly is powerful from I/O point of view. Now we have traced eight stages – what do you think is possibly the next one, one which we may put as the ongoing or current development?

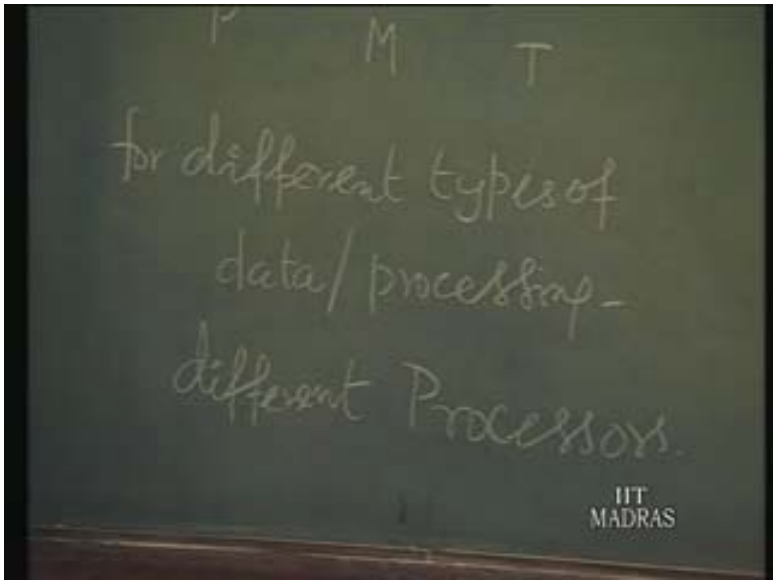
In the ninth stage the main processor was giving bus to a specialized I/O processor. We can have one processor, which can handle one type of data; a graphics processor for graphics; we have a special processor for handling graphics; similarly if you have audio data, have another processor; if you have video data, have another processor; if you want one type of graphics, you have one processor; you have another type of graphics, have another processor, and so on and so forth. So what has happened is in other words we can talk about the ongoing thing or what is still evolving is something like distributed I/O. That is, you are not having one I/O processor, but you are having multiple I/O processors. When I say multiple I/O processors, it is not just n types of the same processor.

(Refer Slide Time: 41:47)



Each one can be a different one. So that is what we are having even now. For instance we call them accelerators. We talk about graphics accelerator; we talk about video accelerator – what are they? These accelerator boards are essentially going to have a special type of processor; for instance, in the case of DSP handling, digital signal processing, we have special DSP processors. So for handling special signals, you have special processors; that is what we are talking about. So we have multiple I/O processors for different types of data.

(Refer Slide Time: 43:19)

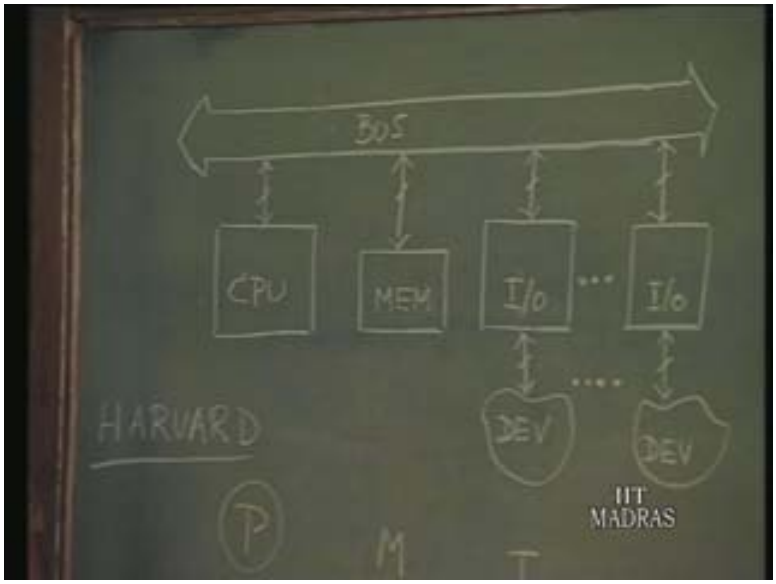


Let us put it in very general way – different types of data require different types of processing. You have different processors, each with its own capability and capacity.

Now trace back – we started with this and we said at block diagram level we can talk about three types in general. Right at the beginning itself we knew that we had to deal with different types of devices, and then, while discussing memory, we had also come across different types of memory. Now what are we talking about? We are talking about different types of processors. What is the next step? In fact, that is also ongoing. We started with one memory and one processor; one memory and many devices, and then many memories and now many processors. In other words, the situation is multiple processor in general and then, when you distribute the function of this processing, with reference to the specific processing need, like text processing, graphics processing, audio processing or the type of data like video data or audio data, if you have special types of processors that directly handle the signal, in which case we talk about digital signal processors or DSPs. What are DSPs? DSPs are also processors.

They have the same capability. So now slowly from one of each, we have gone into n of each; in other words, multiples. When things have evolved like this here, we have traced specifically with reference to I/O, but earlier we have also talked about different types of memory. We were not talking about the evolution and all in general. If one of each has to go on as n of each, that is, multiples, what about this bus? The bus also can do that. Similarly we have seen it already. Here in this particular one, we were talking about a general high speed bus and a special low speed I/O bus. So we had one bus here and another bus here. In the case of DSPs, I am talking about digital signal processor; you would have noted that there will be a special instruction bus and there will be a special data bus because instruction and data both go from the memory, and once when a data is being processed, it is possible to fetch the next instruction and keep it ready. Generally we refer to it as pipe line. We keep it ready in anticipation that that is the next instruction that will be used; it may or may not be used. So you paralyze the activity. To support these multiple buses sometimes even this will be split into instruction memory and data memory. I think I had indicated that earlier and then I also said when you have that, what is known as hardware architecture comes.

(Refer Slide Time: 47:22)



That is the one which has instruction bus and data bus separately and the memory connected to that appropriately: the instruction memory and data memory.

So now you can see that slowly things are growing more and more complicated at each subsystem level and hence it is becoming very complicated at the system levels too. In fact we have not gone into details of instruction fetching. We just said instruction is fetched but actually when instruction is fetched and then when it is being processed it is possible to fetch the next instruction and if execution of one set of data takes some time, it is possible to break down those things and then carry out lot of parallel activity. Some of it we will see in the later lectures. For now you can just see how really this evolution has taken place and in fact we have historically two instances as I was mentioning in the case of I/O. You should also be able to really understand how a single thing has become multiple things and all just because of semiconductor technology evolution.

Essentially that is because today if one make is available, the next year accordingly the applications also make this demand. What was possibly a restriction in the size of the program breaks down because more memory is available and when more memory is available then there is more demand and then the processing need also goes up; in this way, it keeps on growing. The purpose of all this is that when you have different types of processors for different types of data or signals, the tendency is to become natural. For instance, if you say that man is a natural, is part of nature, he has all the natural processing capabilities – for instance he has eyes specifically for seeing, he has ears specifically for audition, and then of course mouth and other related systems for producing the speech and so on; in other words different senses, and for different senses we have different transducers. Similarly here tomorrow we may have general purpose processor or a speech processor for feeling something; there may be a processor for smelling and so on and so forth. So the tendency is really for the computing system to go as close as possible to natural processing system, that is, the human level – that is how the trend is. In the next lecture, we will take a look at some of these I/O devices because we have to complete that part and then come back to this over all situations.