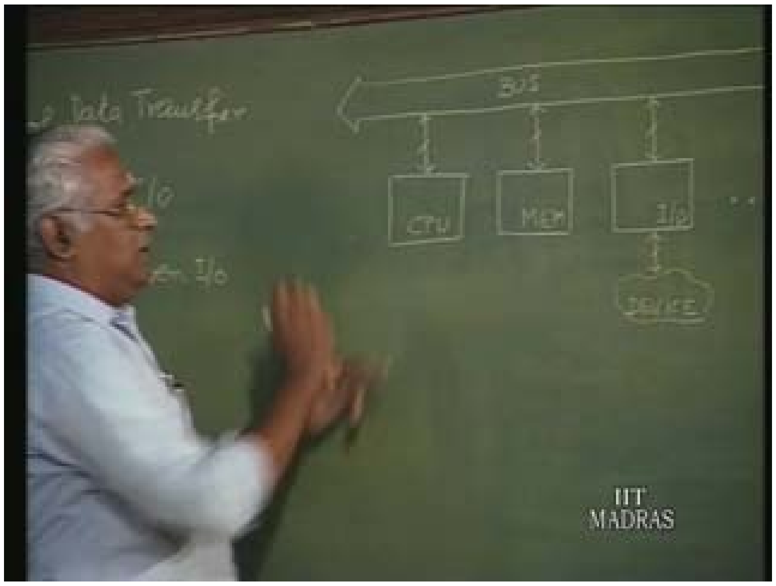
Computer Organization

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

Part – III
Input/output
Lecture – 26
Dma: Direct Memory Access

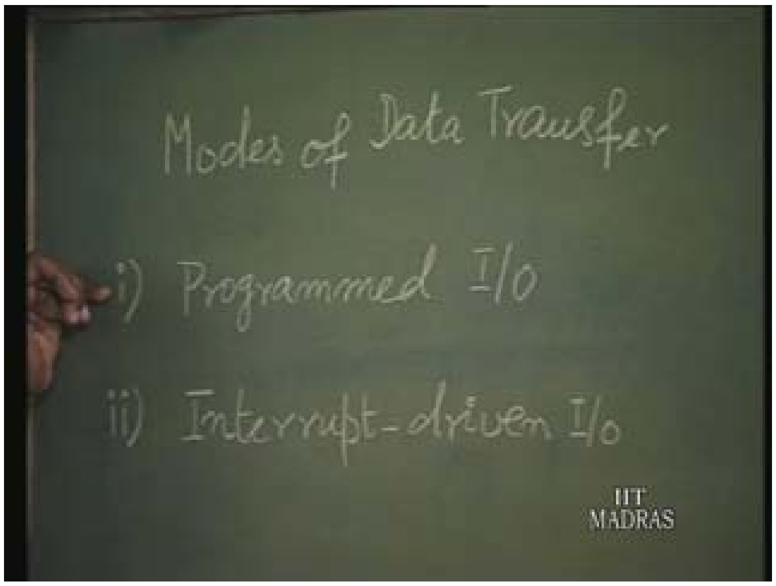
In the two previous lectures, I introduced two modes of data transfer, essentially with regard to the CPU–I/O interaction. The first one in which the CPU takes the initiative is called the programmed I/O; that is, the CPU runs the program and effects the data transfer and that is entirely up to the CPU, which means it runs it as per the given program.

(Refer Slide Time: 01:21)



The sequence in which the different devices will have to be checked for their readiness, etc., will all be programmed – one can, by changing the program, alter the sequence and so on. The problem with that particular thing was because it is the CPU initiating, as an when the device is ready, the CPU may not be aware of it and hence it may not scan that particular one; that is check the status of the device and know that it is ready. So there is a possibility that the data could be lost. We moved on to the second one, in which what we said was checking whether the device is ready or not is left to the CPU, but as soon as the device is ready it will indicate to the CPU that it is ready by generating what is known as an interrupt signal.

(Refer Slide Time: 01:33)


Modes of Data Transfer

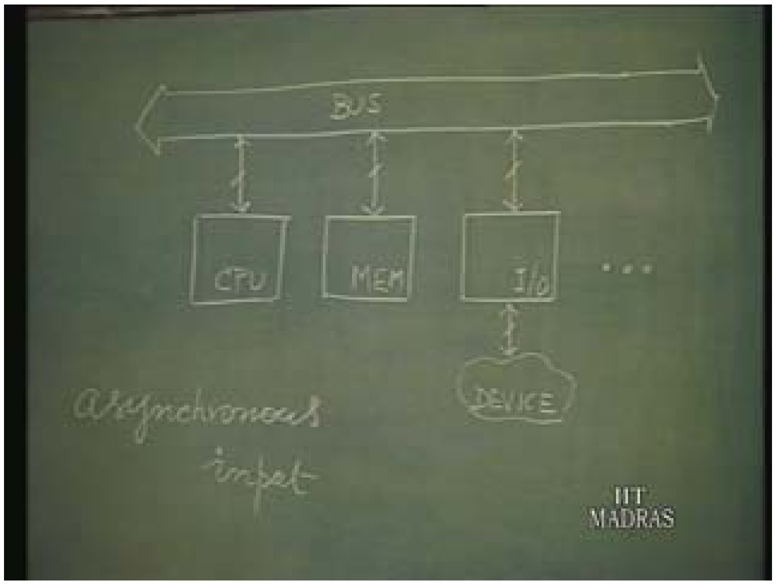
i) Programmed I/O

ii) Interrupt-driven I/o

This is in fact an interrupt driven input/output scheme, meaning the CPU does the data transfer as in the previous case but checking whether the device is ready or not is left to the CPU. The device itself informs when it is ready. In the previous lecture we were looking into the details of how this particular thing works. Essentially it is something like switching from a main program or a main routine to a sub-routine. A CPU executes a main program and then there is a call to a sub-routine; so it jumps to the sub-routine. Similarly in this particular case also, the CPU is involved in some processing. We can call it main processing or a main program and then, subsequently, when the device interrupts the CPU, the attention is turned away from the main program to the service program meant for that particular device. So here also we talk about a switch from main program to the device interrupt service program; this is generally called an interrupt service routine. In that particular process there is some delay because after the device has been serviced its CPU must switch back to the main program.

The CPU is doing something; the device, when it is ready, is going to indicate an interrupt. The processor's activity is interrupted then it is going to attend to the interrupt service for the particular device, whichever is interrupting. At the end of that interrupt service the processor must continue with whatever it was doing earlier; that is, it is something like switching back from the sub-routine to the main routine. So the necessary information for that must have been stored earlier, prior to switching over to the other context. That is going to cause some delay; that is number one. Number two is that the CPU is running at its own speed and the device is going at its own speed. There is no synchronization between these two; these two are two independent things, except when there is some transfer. Even then, the synchronization will be with reference to the interface and not the device as such. So we say that the device will interrupt the processor and that interrupt is generally characterized as asynchronous input; asynchronous because it is not synchronous with the processor's main clock. So interrupt is in fact an asynchronous input to the processor and when can the CPU be interrupted?

(Refer Slide Time: 05:37)

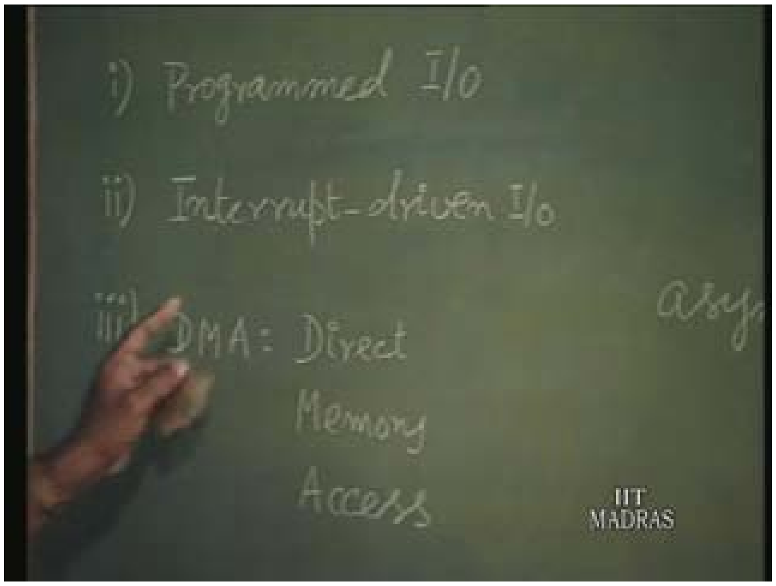


We saw this also in the previous lecture. We saw that the CPU should complete the current instruction cycle and only then prepare for the context switch, which means there is some more delay. That is, in the current instruction cycle itself, there is some delay because it has to be completed. So we saw that there is some delay in this; the CPU turning its attention away from the main thing to the specific device. Now what if the device happens to be very fast? That is, it cannot wait for the delay mainly because it is too fast and it keeps generating the data very fast. Then there is the possibility that the data may be lost, in which case, the CPU is involved in the data transfer and the CPU must continue with whatever it was doing before the interrupt after the interrupt service, so we have no other choice really.

The instruction cycle must be completed; so this is the minimum thing, in which case we say that if the device is too fast, the CPU's delay from switching from the main to the interrupt cannot be tolerated. If that is so, then the only other way is the CPU just cannot be involved in that. In other words, the CPU must be cut off from the main activity mainly because what we say is the CPU is too slow for this particular device. The CPU cannot handle the situation; CPU is incapable of doing it in which case, the device indicates to the CPU – of course, this is meaningful for the fast device – now there are two things. Generally a fast device will be generating a bulk of data mainly because it is too fast; it is meaningful to keep generating the data at a very fast state. It is possible that to initiate, that is, the initial period when we first check whether the device is ready, it may take a long time. We are not bothered about that. Once the device is ready, it is something like a machine gun – the data come keeps coming, but to load the machine gun and then to aim, all these things will take the same amount of time, whether it is an ordinary gun or machine gun.

The situation is the same here too. Now when the device is very fast and it is generating a bulk of data, it is meaningful to allow the device to put the data in the memory directly. So we are coming directly, means the CPU is not involved; something else must come because the CPU is always considered the master of the bus. If the CPU is cut off, something else must master mind the whole process. So moving on to the third one, in which the CPU is replaced by some controller; the precise method of the mode of transfer is briefly called DMA. As the name suggests, it is direct memory access; that is, this DMA stands for Direct Memory Access. That is, we can say the I/O device is going to access the memory directly; the CPU is not involved in it.

(Refer Slide Time: 09:28)


i) Programmed I/O

ii) Interrupt-driven I/O

iii) DMA: Direct Memory Access

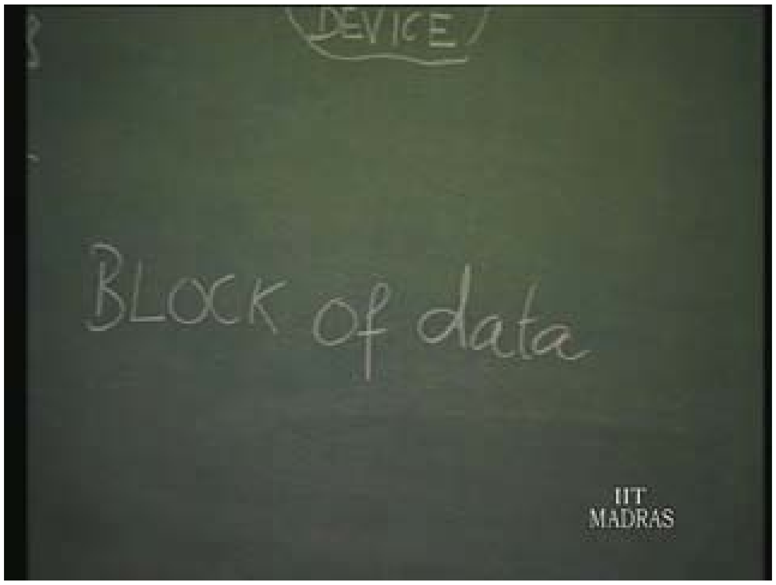
asy
IIT MADRAS

It is going to access it directly and then store data. Now obviously what we are assuming is the memory is fast enough to respond to the device request. The memory is any way going to work at electronic speed. Now we said CPU is not involved; then what else is, because there must be something, which is the master of the bus. For this purpose, there is a specific chip or we may say circuit, which is called a DMA controller. That is, this DMA controller is going to take the place of CPU. Earlier the CPU was involved; now the CPU is cut off from the bus and instead, the DMA controller will take over. It is going to play the same role as CPU, but CPU is a very general purpose processor, whereas the DMA controller is a special purpose processor. This is specifically for special purpose and the purpose essentially is to see that the data that is generated by the fast device is routed to the memory. Remember I said we generally deal with bulk data generally; we talk about that as a block of data.

(Refer Slide Time: 10:52)

A block of data could be anything like 1 k, 2 k or 5 k size or even less. The CPU is cut off from the bus and a DMA controller will take over and that particular one is going to sit on this bus necessarily.

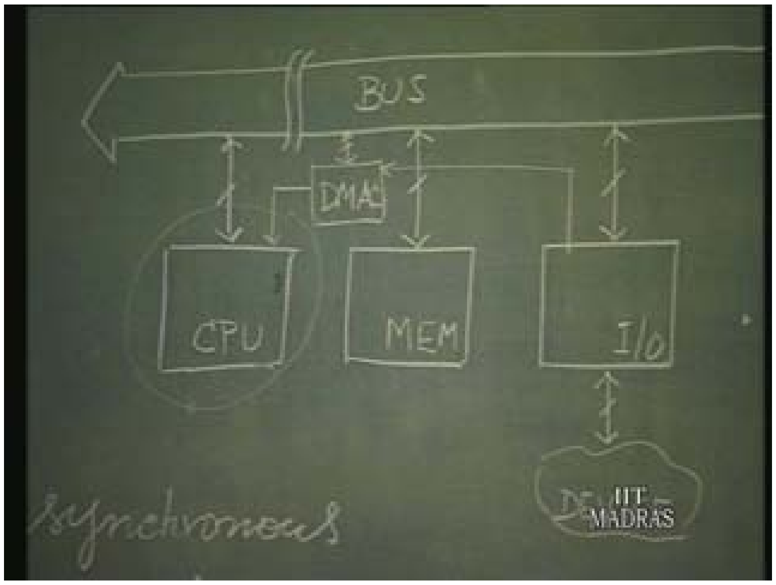
(Refer Slide Time: 11:17)



Now you can see I have to introduce the DMA controller here – what is the best way? I can make use of the same set of signal lines that are there in the bus. I think I will draw another sketch possibly because some of this address and address control data and all these lines can be used because the bus is not in use by the CPU as such. The CPU will be cut off but the bus is still available. Now assuming the bus can accept this data transfer and can respond to the speed at which the data is coming, it can cater for this particular transfer and then it can be used.

Otherwise of course, a separate bus line can be used; but generally it is not so. The DMA controller is going to replace the CPU and sit on the bus and so I will just put it as DMA controller is introduced.

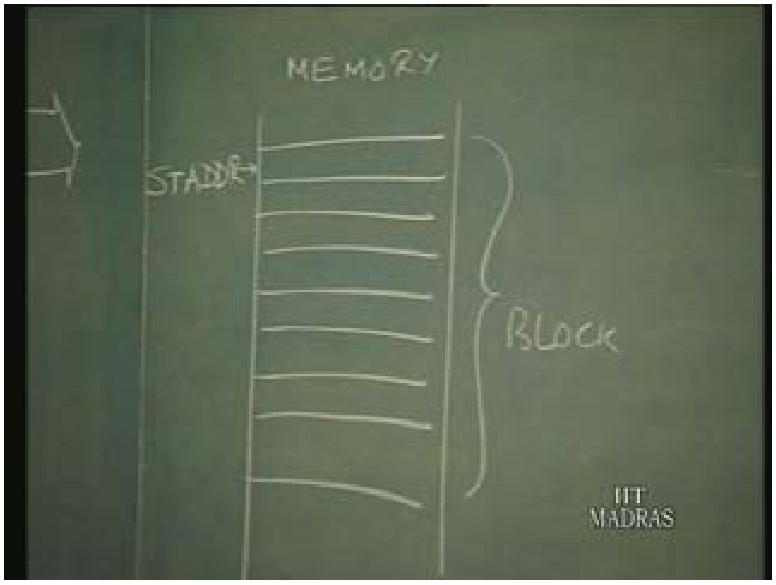
(Refer Slide Time: 13:29)



Now any fast device is going to send its request to DMA controller and the DMA controller will indicate to the CPU that there is a device which is ready. Specifically we can say this is also part of the bus only; or let me put in this way. We can also consider this I/O is the DMA controller. This is specifically shown here because it is a very general purpose I/O. This DMA controller is very much a part of the interface as you can see. So the device is going to put in its request to the controller; the controller will indicate to the CPU; and the CPU will cut itself off from the bus and indicate to the controller, so that the DMA controller, in place of CPU, will set and monitor the transfer from the device to the memory or memory to device, depending on input or output.

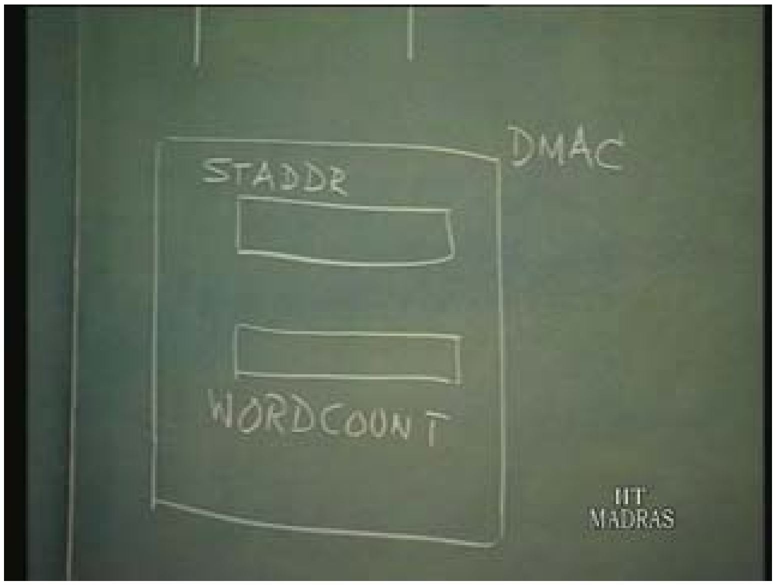
What is important is that it is a block of data. Now when we deal with block of data, what is generally done is we have the memory organized. So we talk about some block of data, whatever be the number of bytes. So you can say that starting from this address, let us call this as start address, starting from this particular location for this length of block, continuously data is going to be stored. Because the data is going to come as a block and also the DMA controller as you can see is having a very special role, it is only going to place the address of memory location and then generate the read or write signal appropriately. This address, read or write, depends on input/output; then once that is done, the next address plus 1; then read or write that address plus 1, read or write – it is going very fast in that fashion. So really it will take only about one or two clock cycles.

(Refer Slide Time: 15:03)



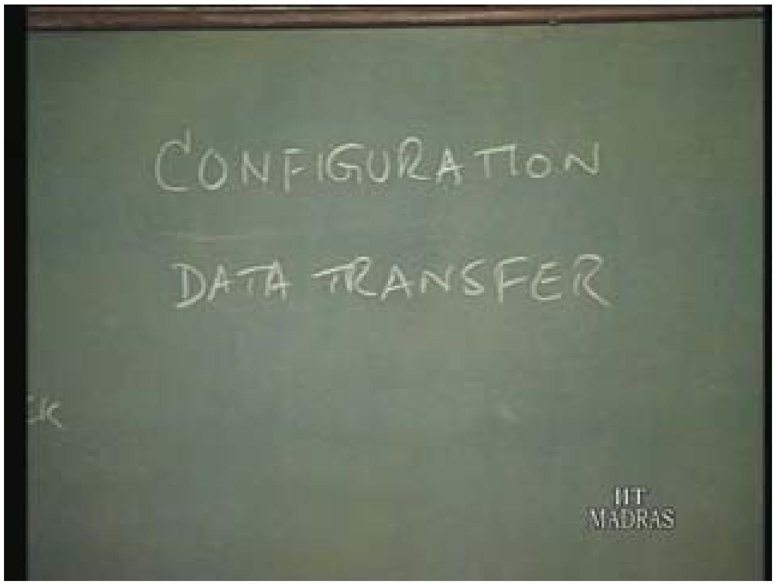
For normal read or write cycle for CPU, it may take many clocks. The DMA controller will do it very fast mainly because it is a hardware solution meant for the fast transfer; it has no other job to do. As part of the DMA controller, we have been talking about the hardware associated with the I/O interface. As you can see because of this particular function, the hardware part of the DMA controller will have to include at least two registers – one which is going to indicate the start address but from where? In a memory the data is going to be stored as block size if it is input. Generally that particular thing is called a word county register. The number of words that have to be transferred is called a word counter register.

(Refer Slide Time: 17:15)



So you can see that in the interface, a start address register is there and the word count register is needed and these will be essentially stored before any DMA transfer can take place because it is not that the moment device indicates that it wants transfer these things can be prepared. These things must have been prepared earlier, meaning well before the DMA transfer can start, even before the device can indicate that it wants a DMA transfer through this request, well before that, these things must have been prepared. So if you recall I was earlier talking about two phases: one called chip configuration phase and another one the data transfer phase.

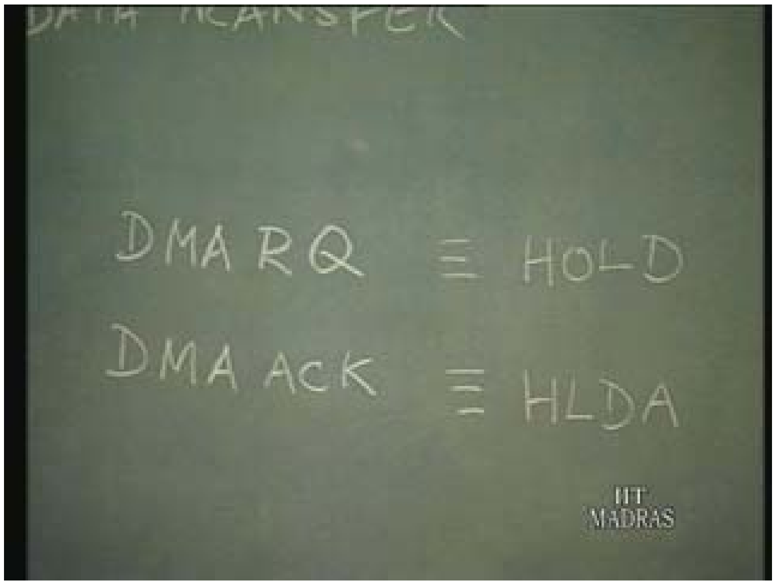
(Refer Slide Time: 18:37)



So when the actual data is transferred, DMA controller is going to take the place of CPU. Now well before the DMA process can start during the configuration phase, the CPU must have loaded these two bits of information into the DMA controller. The start address and word count must have been thought of a priori. The device will request DMA transfer; we had earlier talked about when the interrupt comes and when the interrupt is acknowledged. Similarly here we have to discuss that when the DMA request comes and when the DMA request is acknowledged. As soon as CPU relinquishes the bus, that is, releases the bus for the DMA activity, the DMA controller should know from which address and how much of data is coming. So these will be part of this. In addition to this you have the earlier things like status register and whatever we had talked about. Specifically for DMA you need this; status register will be there; there may be some other one, say a command register; and then we talked about data buffer in general.

It is like this: before the CPU proceeds with the program, it has to configure all the I/Os appropriately; specifically in the case of the DMA thing it must have loaded these two bits of information – when we say bits I hope you understand it is these two pieces of information in the DMA controller – and then it proceeds with its own program. When the device which requests the DMA transfer is ready, it is going to indicate. Now how does it do it? Generally this signal that goes to DMA controller from the device is called a DMA request. DMA request is a signal which is an input to the CPU finally it will have to go to the CPU. The DMA request has to go to the CPU and in response to that, CPU will generate what is known as an acknowledge signal. That is the DMA acknowledge signal.

(Refer Slide Time: 22:03)

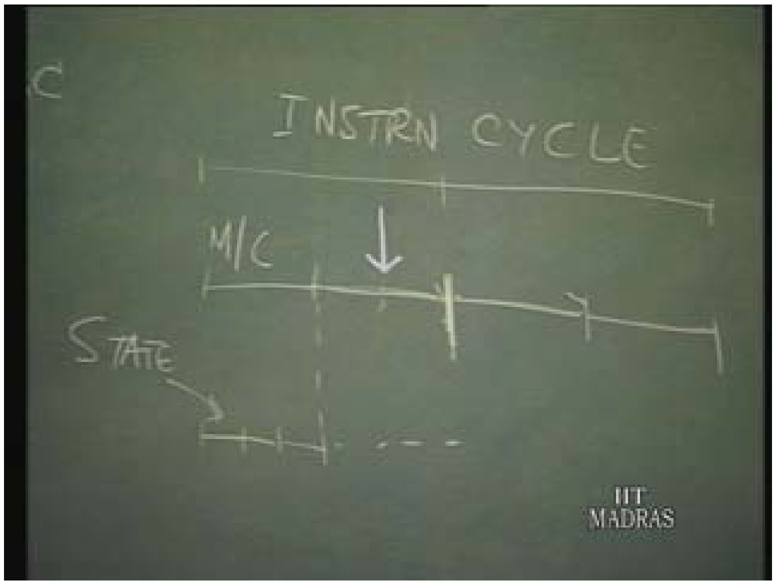


DMA RQ ≡ HOLD

DMA ACK ≡ HLDA

In respect of some processors, this thing is called hold. Why is it called hold? We will see, and this is called hold acknowledge. You put someone on hold, waiting, and here you find CPU is put on hold. CPU will be waiting until this DMA transfer is completed. That is why the name hold has come. The fast device also requires block transfer. Of course that also needs to be fast. Then it puts in a request and through the controller the request will go to the CPU. Finally it is an input in the CPU. Now as before, the DMA request as we saw in the other one interrupts; the DMA request is also an asynchronous input, it is not synchronized.

(Refer Slide Time: 26:13)



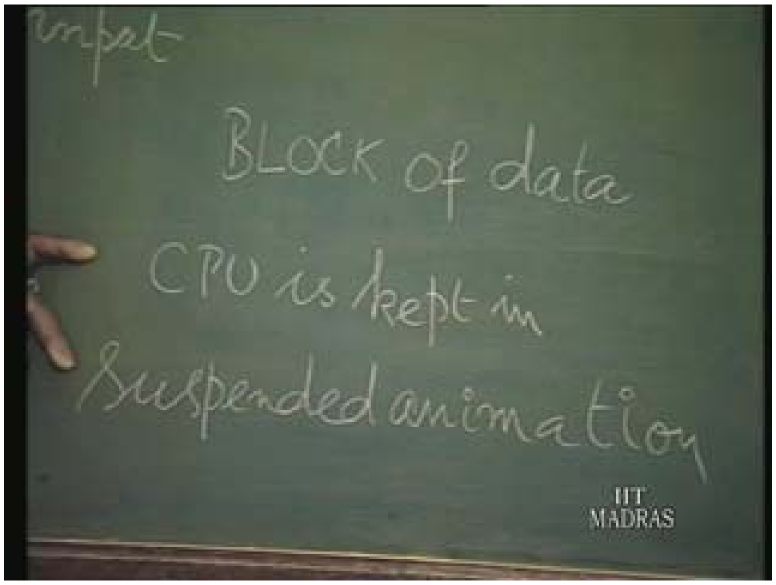
The device, when it is ready for transfer with the data, is going to indicate it is an asynchronous input; DMA is also an asynchronous input. What did we say in the case of interrupt? In the case of interrupt, we said when an interrupt request comes, the CPU will complete its current instruction cycle and that is going to cause some delay. We have an instruction cycle and that consists of quite a few machine cycles, and they consist of few states as we have indicated. Each one is the machine cycle and here each one is the state. Remember this is the very first thing we said. We said instruction cycle consists of let us say fetch and execute, and every time the CPU accesses the bus there is a machine cycle. Now the DMA request can come any time during the instruction cycle, which means it will come during any machine cycle. Let us suppose this particular thing has come, say, at this instant; that is, the DMA request has come at this instant during some state. What is to be done? If this were an interrupt request, we know that the end of the instruction cycle service will be taken up.

If it were a DMA request, that is, it is coming half way through the machine cycle. And now what is the machine cycle? Machine cycle is essentially consisting of CPU placing an address indicating a read or write or any other special control and then finishing that part. That is in the machine cycle. And then what happens in the case of DMA is that the CPU is not going to be used anymore; that means, the register contents in the stack are going to remain intact within the CPU; that is, the processor's state is not disturbed. So unlike interrupt request, in the case of the DMA request the CPU can relinquish the bus at the end of the current machine cycle because at this point, that is, the end of the machine cycle, if the CPU relinquishes the bus for the controller, the CPU still holds all the information in its register; CPU is not involved in the DMA. So at the end, when the DMA controller indicates that it has finished everything, then from the next machine cycle, it can be started – that is not much of problem. So this way you can save some time and sometimes, it need not even be the end of machine cycle because we have seen earlier that not in all the states we have bus activity.

That is, suppose in a given machine cycle the last state corresponds to incrementing a register. It is a pure CPU activity; the bus is not involved because one basic activity is what constitutes a state. Assuming the last state in a given machine cycle corresponds to incrementing of the register, it is not involved in the bus. So we can even say that in response to DMA request, normally at the end of the machine cycle the bus will be released by the CPU. It need not, really strictly, at the end of the machine cycle, because it can be one or two states earlier. The point is that the bus should not be involved any more in that particular cycle, machine cycle. So that even when the bus is released DMA controller if there is a CPU's internal activity that can still go on there is no problem. So this is the one difference between the response to a DMA request and response to an interrupt request.

So these are essentially the three modes of data transfer and, believe it or not, these are the only three modes of data transfer: One is that in which there is no way in which the CPU is involved. One is that in which the CPU is not involved in checking whether the device is ready but CPU is involved in the data transfer; the third is that in which CPU is not at all involved, it just kept aloof. So we say the CPU is kept in suspended animation; we may say that. CPU is kept in suspended animation and it is going to wait until the DMA controller releases that is, until the block of data transfer is over, the CPU will be kept in suspended animation and at the end of the block transfer the controller will release the bus to the CPU.

(Refer Slide Time: 29:17)


Input

BLOCK of data

CPU is kept in

Suspended animation

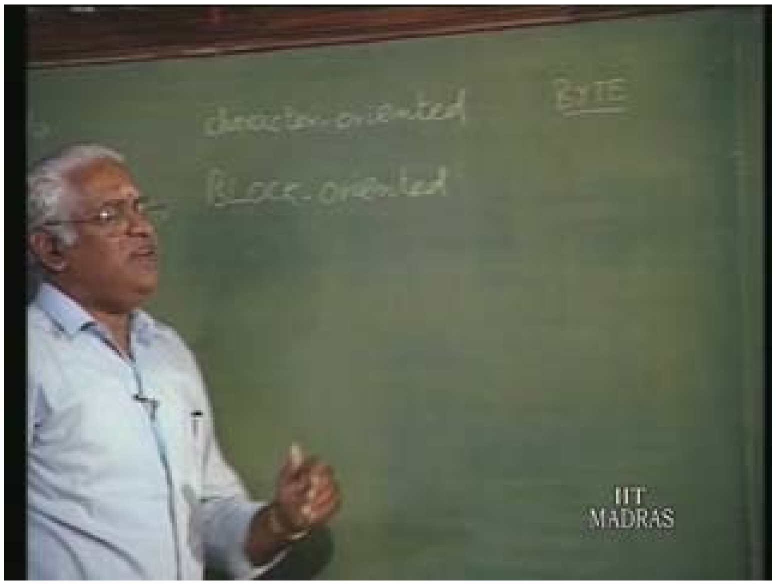
So now you can see that CPU is usually the master and in this case, instead of this master, another master takes its place. So you can see that CPU is also a processor, DMA controller is also a processor but then DMA controller is a special purpose processor meant only for data transfer, nothing else. So the third one is one in which the CPU is not at all involved; in the first one, the CPU is fully involved; in the second one CPU is involved only in data transfer; in the third one CPU is just said to keep off; it can handle the data transfer and some other faster device will come and do it and the CPU is kept in suspense. These are the three modes of data transfer.

(Refer Slide Time: 31:12)

Programmed I/O

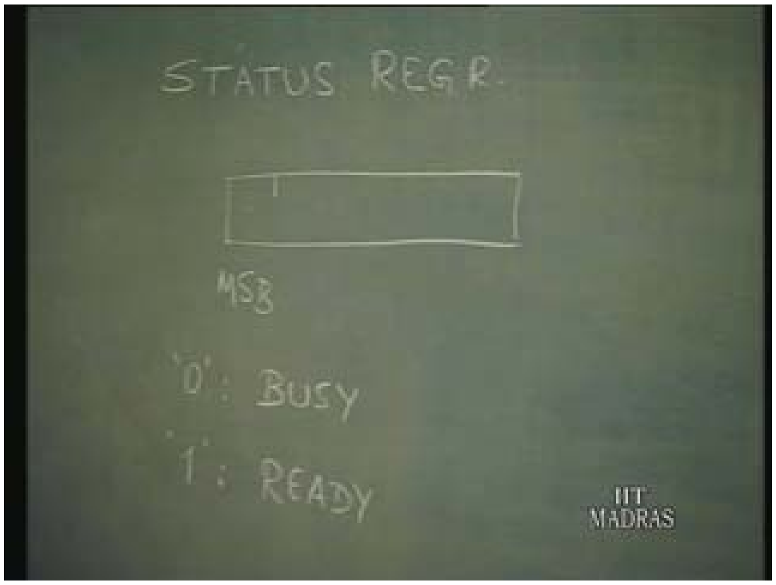
Now we will go into the details of some of these. Let us look at some details about each of these modes of data transfer. Essentially what does the programming look like? Now if we take the programmed I/O, that is, the very first one, essentially it would be first checking the status and then doing the data transfer. Now I think I will skip some of these things. Suppose you want to transfer say 5 bytes of data, and assuming the device is byte-oriented device, that is generally we talk about a device like this – may be a character-oriented or a block-oriented device – there are two types of devices, say character-oriented or block-oriented.

(Refer Slide Time: 32:00)



The block-oriented device is the kind of device that we had come across already in the DMA. In the case of character, a character we say is a to z, any number or special symbols, generally you just need only 1 byte – 1 byte or one character. So these devices will be generating byte by byte, that is, it is going to generate character by character. These are character-oriented devices. A block-oriented device is a block of data. Now let us just assume that we are dealing with the character-oriented device, which means essentially generates bytes, and let us assume that in our situation the CPU is going to look for just 5 bytes of data. So when 5 bytes of data is what is looked for, essentially the CPU is going to check whether the device is ready. if it is ready then it is going to move 1 byte and then keep track of it. So essentially a counter will be used; a counter will be set to 5, let us say, and after the transfer of every byte, the counter contents will be decremented so that when the counter contents come to 0, we know that 5 bytes have been transferred. I will just skip those details. I will not enter all those in the part of the program. So what is programmed I/O? First you have to check the device whether it is ready. We know that the device being ready can be checked by looking in to the status register and recall I also said for the status register the very first bit of whatever be the size, the MSB, is the most significant bit.

(Refer Slide Time: 34:29)



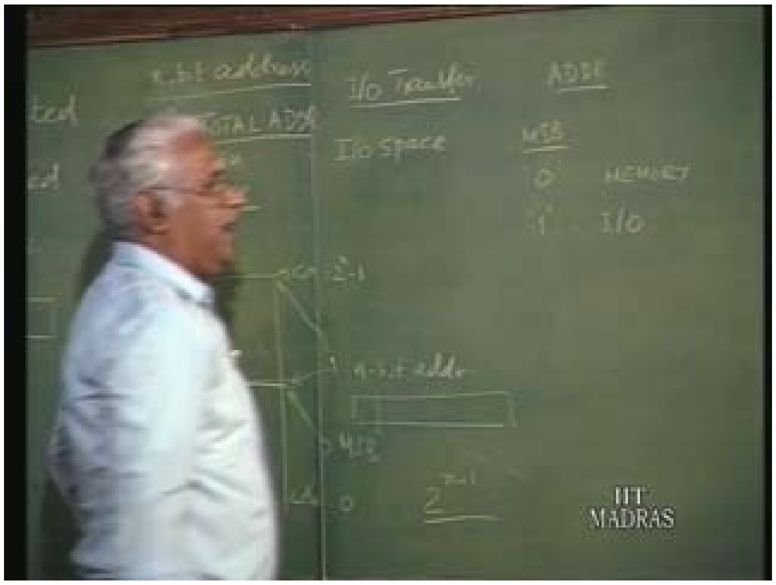
This particular bit, depending on whether it is 0 or 1, is going to indicate whether it is ready or not ready, that is, busy. Now let us assume 0 means – if this particular bit MSB or most significant bit is 0 – let us say it hardly matters how you do it. The moment the device is busy and if it is 1, the device is ready with the data; we need to know all these in our programming. Let us assume few more things. That is, this particular register must have an address. The CPU generally addresses the status register and then reads the contents. So we will just call that address very simply, mnemonically, as device status.

(Refer Slide Time: 36:29)

Device state is the address; we are just using a symbol instead of the specific address and then there are many things that we can do. In I/O transfer there is one method I think I have not talked about earlier; maybe I will explain that because it will be useful. In the I/O transfer how exactly do you split the I/O space? What do you mean by I/O space? We have a total memory space given by the address, sorry, we have say an n bit address. We will start with an n bit address; an n bit address is going to mean the total address space is going to be 2^n. So that is the size of the total address space. Now this entire total address space can be used as memory space, in which case you can have other n bits or so for the I/O to indicate the I/O space. Or this total address space 2^n, suppose 2^n is the total address space, this can also be divided into 2 exactly.

Let us say, this is the low address, this is the highest address; then what is the address of the lowest location? That will be address 0. The address of the highest will be 2^{n-1}. so the total thing is 2^n. Now if you just see n bit address, you just take the most significant bit, whatever be the size – the most significant bit. The lowest half will be indicated by this MSB being 0. MSB being 0 will be indicated by this lowest half because it is exactly half. Now this MSB being 1 will indicate this top half. So the MSB, when it is 0, will be having n−1. So 2^{n-1} is exactly half, so that will be the size; n bit address will leave this; then you have n−1 n bit address; so 2^{n-1} will be one half. You just add one more bit, then you have another 2^{n-1}. Now suppose I allot this low space for memory and this high space for I/O, what does it mean? It means all the addresses, starting with MSB 0, are memory addresses and all those addresses starting with the MSB 1 are I/O addresses.

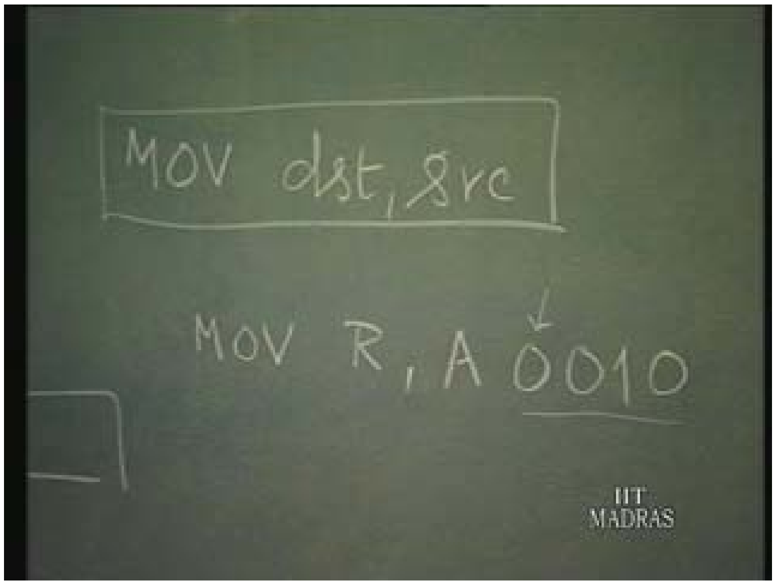
(Refer Slide Time: 39:50)



Now why do we do this arrangement? There are many reasons: one thing is that we can use one instruction. Now see this: supposing I say MOV and let me follow a format. I will first give the format. Let us say MOV destination comma source. If this is the format of the instruction, MOV is the opcode and then we are referring to two locations from where to where. Suppose I have an instruction MOV and destination let me take it as a register. Some register in the CPU – I will just call it register. Now n bit address, shall I assume just a 4-bit address for simplicity? I am going to give an address here. To indicate that it is an address, I will just use the character A; like I am using R for register I am just using A for address.

Then, I am going to give just a 4-bit address. Suppose I give a number like this 010, what is this? In this 4-bit address, this is the MSB; MSB is 0; 0 means it is memory.

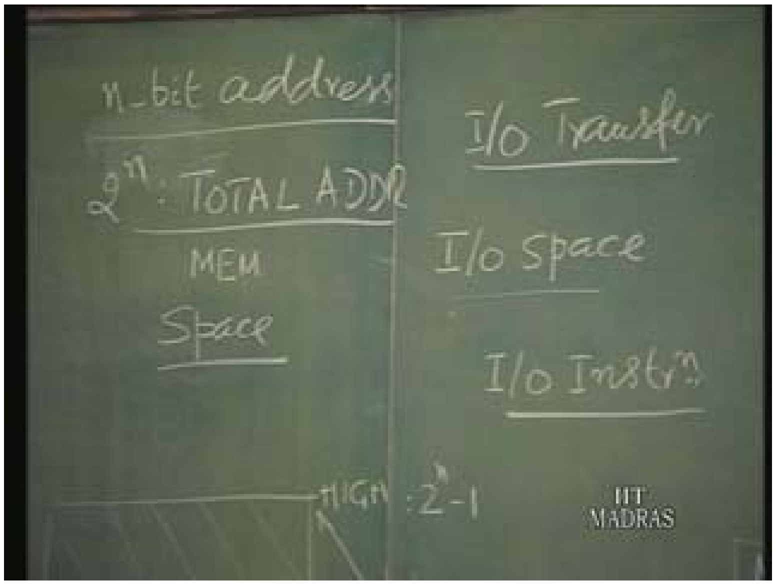
(Refer Slide Time: 41:42)


MOV dst, src
MOV R, A 0010
IIT MADRAS

So the effect of this particular one will be to move from source to destination. I am talking about all this mainly because I would develop a program and I will use this format. There is a reason I am just digressing and then explaining a few more. Incidentally there is some concept also you will learn that says you are moving, and this address starts physically with MSB 0, as it is in memory. So basically the effect of this is to move from memory, specifically, from address 010, because this is only indicating memory or I/O: this particular one to register. So this is going to cause a movement from memory to register. So I will just put it as there is some movement from memory to register. Now I will use the same instruction: MOV, I will use the destination same register R, I am just making a small change in the address. What is that? Instead of MSB 0, I am making it 1. What does this mean? We had said earlier that if MSB is 1, it refers to I/O address. Now what is the effect of this? This says move from an I/O device of address 010 to register; from I/O to register. Now what has happened? Earlier we were moving data from memory to register; now with this instruction we are moving from I/O to register. What is this? This in fact is the memory read operation, whereas this is a device input operation.

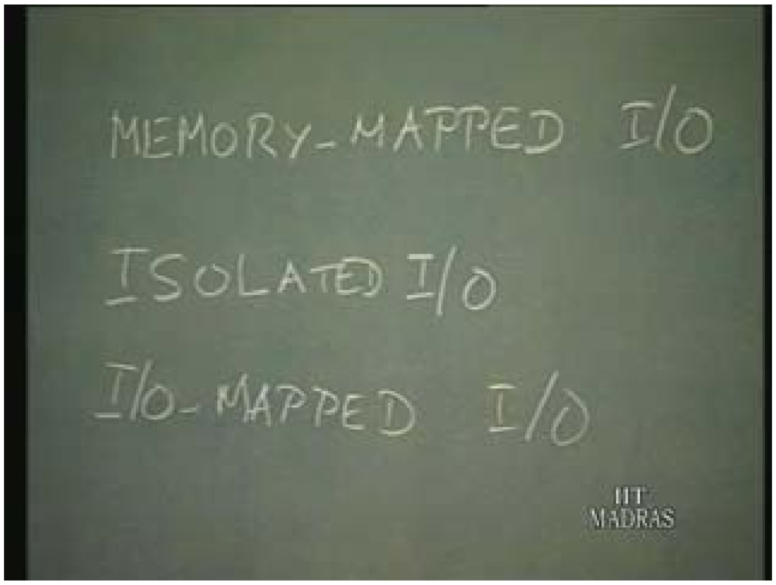
So I am using the same instruction; by using the same instruction I am either dealing with memory or with I/O mainly because I am dividing in such a way that 0 is corresponding to memory, which means essentially all those addresses should correspond to memory block and all the device addresses must correspond to I/O devices. In hard wiring we have to take care of that. Now that is the hardwire part; but as far as software part is concerned, what has happened? I am using the same instruction either for dealing with memory or with I/O. In other words when I say same instruction, we can say the same memory referencing instruction can be used either for memory or I/O transfer. Otherwise, what happens? If I use this entire address space for memory, then I need another set of address for I/O because you will be having the one address let us say 0010, which will point to memory and the same address 0010, which points to I/O. Then how do I distinguish?

(Refer Slide Time: 45:55)



The only way I can distinguish these is to have separate I/O instruction; I would use separate I/O instruction in case I use the entire address space for memory, whereas here I do not need a separate I/O instruction. So this particular scheme, in which the address space is divided and I/O space is mapped into the memory space, this scheme permits me to use a memory referencing instruction itself either for memory referencing or for I/O referencing; I do not have to change the instruction. This scheme is called a memory mapped I/O. Essentially it means the I/O space has been mapped into the total address space and there is a price I am paying for it. What is it? My memory size gets reduced.

(Refer Slide Time: 48:18)

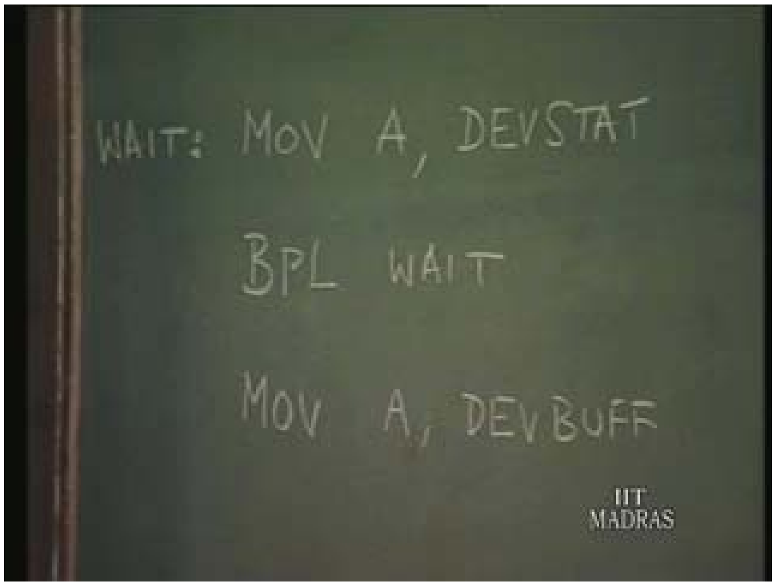

MEMORY-MAPPED I/O

ISOLATED I/O

I/O-MAPPED I/O

But the advantage is that I do not need a separate I/O instruction. Now both the schemes are there; it all depends on how you connect. If you have a separate memory space and I/O space then I can have whole 2^n address space for memory and may be another 2^n, which you generally do not need for I/O space, because we talk about 16 MB for memory, but you are not going to have 16 mega devices, may be just 16 devices itself may be too much. So in the other in case I use the total address space for memory and I have a separate I/O space; it is another n bit instruction for n bit address for I/O. Then I have to use separate I/O instruction; in that case it is called an isolated I/O scheme, and sometimes it is also called mapped I/O from the motivation from memory map. So both are the same; isolated I/O is understandable; that is, I/O space is separate from memory space, and it is isolated and because this particular one is called memory mapped I/O, they also call this is mapped I/O. Now I will use this particular memory map because it is easy to write the program. I have to use this for generating a program. Now quickly let us see one or two lines of the programmed I/O. First is checking the device status; so what I would do is say the first point is MOV to accumulator.

(Refer Slide Time: 50:57)



```
WAIT: MOV A, DEVSTAT
      BPL WAIT
      MOV A, DEVBUFF
```

I am following the same format MOV–destination–source. So MOV accumulate to the accumulator; whatever you have in the device status. That is the address; that is just nothing but moving the status word to the accumulator, which is the CPU. And what did we say? I say positive because the most significant bit is 0 and then corresponds to the sign; generally 0 is taken for positive and 1 for negative. So I can just say the next branch plus, meaning, in case the device status happens to be a positive word, plus word, then wait. That is the label that says keeps waiting; that is, keep checking the status of the device. Next the process is going to take the instruction and now it can come to this instruction only if it is not plus, meaning the device, when it is not plus, it means it is negative, which means it is ready. The device is ready and the data must be available in the data buffer. Now just see here. What is it I am doing? Like status register I assumed the data buffer for input because from the data buffer the data has been moved to the CPU accumulator, which is input. And so I assumed actually an input device. So I used these two for checking the status and seeing whether the device is ready. If the device is not ready as per this program, it is just going to loop here, just keeps waiting. So this instruction will be taken up only if the incoming word is negative and that will be negative if the device is ready.

The device indicates that it is ready when its buffer has the data. So at this point when it comes, it means that the data buffer or the device data buffer is having the data, which is input. Previous to this, we said there is some initialization at 5 bytes or 10 bytes and subsequently whatever you want to do. This is essentially the crux of the matter in programmed I/O technique. This is for checking the status; this is for effecting the data transfer. Now you just see I am doing I/O with one instruction; and I can also use the same instruction for memory referencing. That is essentially because I am following memory mapped I/O. If we have isolated I/O the program will alter slightly; you will have to use in instruction on because you are dealing with the in and out instructions, depending on what the device is if you are having a separate I/O space.