

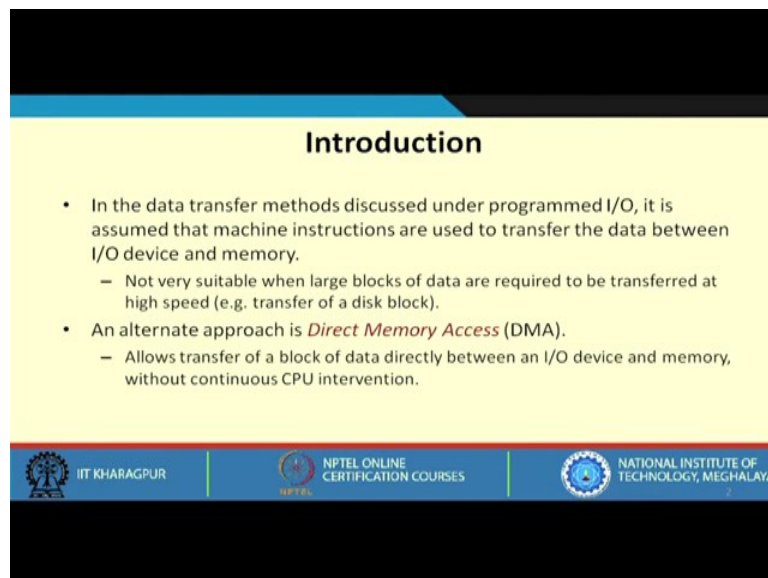
Computer Architecture and Organization
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 48
Direct Memory Access

In the last few lectures we have discussed the programmed IO technique, using which the processor of the CPU can transfer data to and from an external IO device. Now one thing to note in those methods was that during the actual transfer of the data the CPU was executing a program, some instructions were responsible for transferring the data may be from an from an input port to a CPU register and from register to the memory, or vice versa.

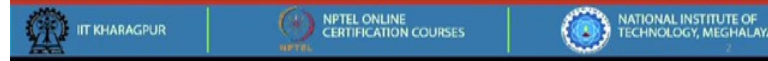
It was the responsibility of that program to transfer the data and this is true for synchronous, asynchronous or interrupt driven IO. Of course, in interrupt driven IO the overheads will be less because the CPU is not spending time to check the status of the device whether it is ready or not.

(Refer Slide Time: 01:38)



Introduction

- In the data transfer methods discussed under programmed I/O, it is assumed that machine instructions are used to transfer the data between I/O device and memory.
 - Not very suitable when large blocks of data are required to be transferred at high speed (e.g. transfer of a disk block).
- An alternate approach is *Direct Memory Access* (DMA).
 - Allows transfer of a block of data directly between an I/O device and memory, without continuous CPU intervention.



Today we shall be looking at an alternate way to transfer data between IO device and memory, this is called direct memory access. In short we call it DMA. Let us see what this method is all about. As I have just now mentioned under the programmed IO technique, whatever 3 methods we have discussed, there were machine instructions that

were actually executed and they were responsible for the transfer of the data between IO device and memory through some CPU registers. But one problem with these methods is that whenever you use a program to transfer data, the instruction execution time will come in as a bottleneck.

Suppose you need to execute a minimum of 8 or 10 instructions to transfer a data. The execution time of those 10 instructions will act as a bottleneck, you cannot have a data transfer speed which will be faster than that time. This is one drawback in these schemes. So, these methods will not be very suitable when we have a high speed device that is transferring data. Not only that, it is transferring data in large chunks which means there are large blocks of data which are transferred in one go and at a high speed. This is a characteristic of block IO devices like a disk, like a CD, or a DVD, this kind of devices will transfer a block of data at a given time.

An alternate approach that we are exploring now is called direct memory access. Here there is no execution of instructions for transferring of the data. Here by having some hardware mechanism we can allow the transfer of data, may be a block of data also, directly between IO and memory without CPU intervention during that time.

You recall in the programmed IO technique for every byte or every word of data that are transferred, CPU has to execute a program. But here, CPU has to initiate the DMA operation and after that the CPU can proceed and do something else, CPU need not have to interfere during the time the data transfer is taking place. And again at the end when the data transfer is over, CPU may have to do some book keeping operations; so only at the beginning and at the end; CPU does not require continuous intervention.

(Refer Slide Time: 04:50)

• Why programmed I/O is not suitable for high-speed data transfer?

a) Several program instructions have to be executed for each data word transferred between the I/O device and memory.

- Suppose 20 instructions are required for each word transfer.
- The CPI of the machine running at 1 GHz clock is 1.
- So, 20 nsec is required for each word transfer → maximum 50 M words/sec
- Data transfer rates of fast disks are higher than this figure.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

Again looking back into that issue, why programmed IO is not suitable for high speed data transfer? We cite two reasons; first one I have already mentioned just sometime back. Several program instructions may have to be executed for each word transferred between IO device and memory.

(Refer Slide Time: 05:30)

LD R1, 0(R5) → Address of an input port.

SD R1, 0(R10)

SUB --

BEQ --

ISR

Save

Restore

IIT KHARAGPUR

Typically what kind of instructions will be executed, let us see. If you look at the MIPS instructions, you may be loading some data into a register R1. The address is stored in some other register, let us say R5; 0(R5) may indicate the address of an input port. So for

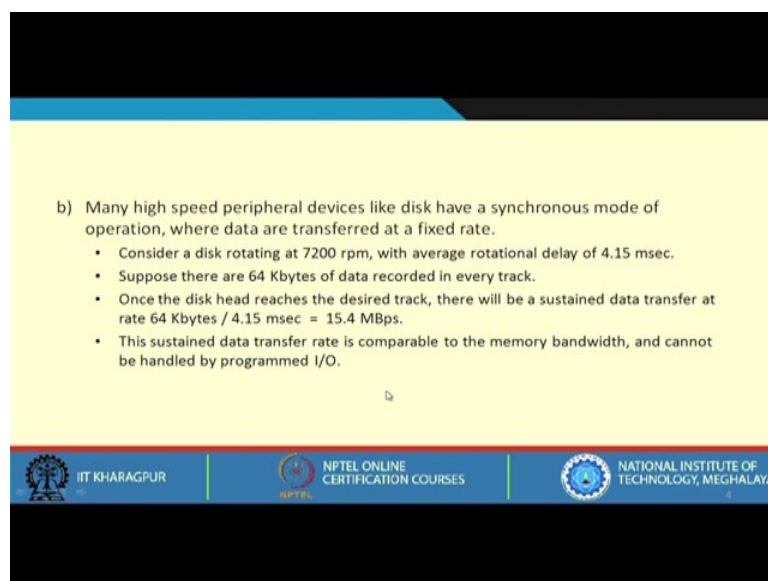
memory mapped IO you can load like this, and after loading you can store the data into memory wherever you want to store this data, let us say R10 is pointing to memory.

Basically you need two instructions, but there will be other instruction like you will be decrementing some pointer, you will be doing some subtract, then you will be using a branch if equal. So, there will more instructions involved in general. In addition to that if you are thinking of an interrupt driven data transfer, then within the ISR, you need to have some additional code like you are saving the status and at the end you will have to restore the status.

Let us take an example. Suppose for each word transfer we require 20 instructions to be executed which is quite realistic. Also let us assume in the ideal case that the CPU is running with a CPI of 1, and the clock is 1 GHz, which means every one nanosecond one instruction is getting executed. So, for the 20 instructions the total time required will be 20 nanoseconds.

So, for transferring every word we require 20 nanoseconds. If we just calculate $1 / 20$ nano, maximum data rate will be 50 million words per second. This is the maximum you can achieve, not more than this. But if you see in the fast disk unit that are available today, the sustained data transfer speeds can be more than 50 mega words per second, which means such programmed I O techniques will not be suitable for such devices.

(Refer Slide Time: 08:17)



b) Many high speed peripheral devices like disk have a synchronous mode of operation, where data are transferred at a fixed rate.

- Consider a disk rotating at 7200 rpm, with average rotational delay of 4.15 msec.
- Suppose there are 64 Kbytes of data recorded in every track.
- Once the disk head reaches the desired track, there will be a sustained data transfer at rate $64 \text{ Kbytes} / 4.15 \text{ msec} = 15.4 \text{ MBps}$.
- This sustained data transfer rate is comparable to the memory bandwidth, and cannot be handled by programmed I/O.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

The second reason is there are many high speed peripheral devices, again disk is a classic example, which have a synchronous mode of operation. And during this mode of operation data are transferred at a fixed rate.

You may recall the way a disk system works. We mentioned that the disk access time consist of several components. First is a seek time. You will have to give some time for the read write head to move under the required track from where you want to read or write, second part is rotational delay you will have to wait till the desired sector where you want to access rotates under the read write head. And third once when you are there depending on the speed with which the risk is rotating and the density of the data with which it is recorded on the surface, data will get transferred at a fixed rate.

So, once you are on the track and the data transfer has started, after that it is something like synchronous data transfer; data will be available at fixed time intervals. That is what is mentioned here. Let us take an example again. Say we have a disk which is rotating at 7200 rpm which earlier we saw that the average rotational delay will be 4.15 milliseconds.

Let suppose that there are 64 kilobytes of data in every track. Once the disk head reaches the desired track and the starting sector is under it, then data can be transferred at a sustained rate because the disk is rotating with the average rotational delay of 4.15 milliseconds. In 4.15 millisecond time the disk will be rotating once and during this rotation 64 kilobytes will be transferred. So, the data transferred rate will be 64 divide by this, which comes to about 15.4 megabytes per second.

If you look at the typical memory bandwidths that are also in the order of tens of megabytes per second. So, it is comparable. If there was some mechanism with which you can directly transfer the data to memory from this device, then a speed match could have been obtained. This data rate obviously cannot be handled by programmed IO as we had seen earlier.

(Refer Slide Time: 11:57)

DMA Controller

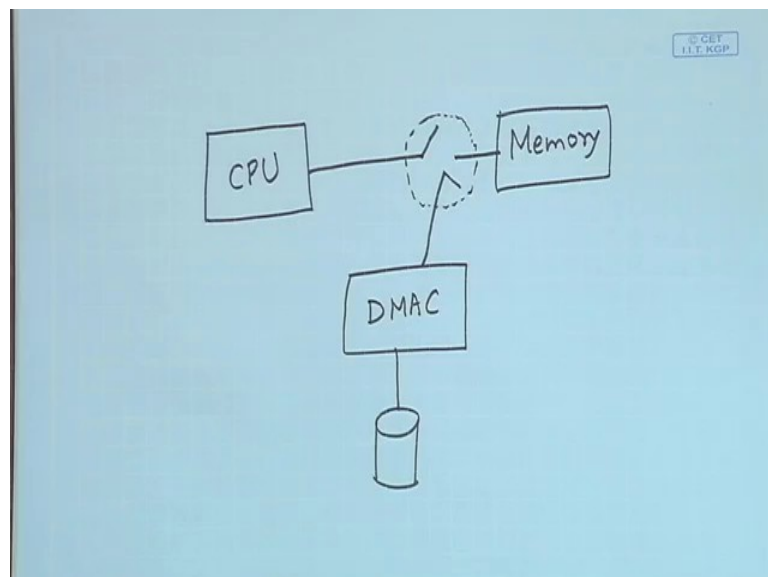
- A hardwired controller called the DMA controller can enable direct data transfer between I/O device (e.g. disk) and memory without CPU intervention.
 - No need to execute instructions to carry out data transfer.
 - Maximum data transfer speed will be determined by the rate with which memory read and write operations can be carried out.
 - Much faster than programmed I/O.

▢

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

DMA controller this is a hardware device we require to carry out DMA transfer. Now let us see what is a DMA controller and what are its functions. DMA controller basically enables direct data transfer between IO device and memory without CPU intervention.

(Refer Slide Time: 12:24)

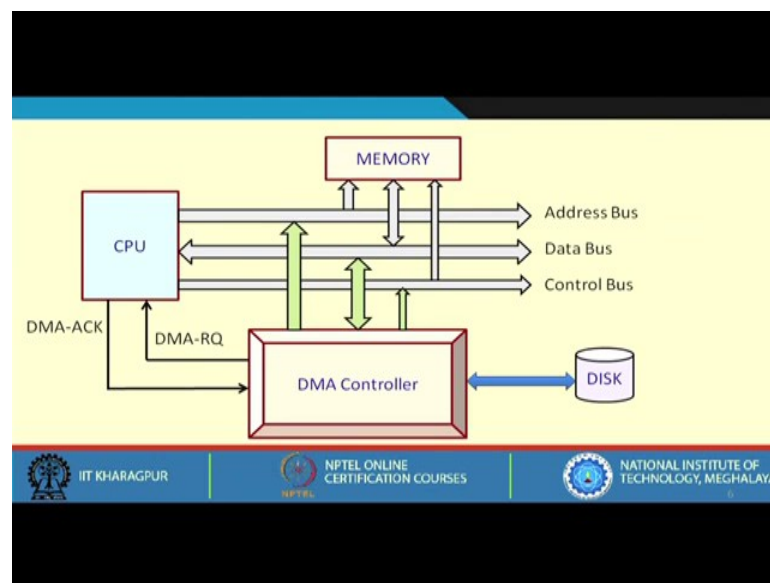


Pictorially speaking you look at this. We have CPU, we have memory, and now we have a third entity the DMA controller. Now CPU will be normally accessing memory for fetching instructions and also accessing data.

DMA controller will also be accessing memory because there will be some devices let us say a disk is connected here; this disk will be connected to the DMA controller and DMA controller will try to transfer data directly to memory. There will be something like a switch, this will be a 3-way switch. Either the CPU will be connected to the memory or the DMA controller will be connected to the memory. There has to be a mechanism with which this switch can be controlled in a proper way. Because under no circumstances both CPU and DMA controller should turn on the switches, both are accessing to memory because there will be a clash, there will be bus conflict.

So, if the DMA controller is able to carry out direct data transfer between the device and the memory, then obviously there is no need to execute instructions for the purpose. And the maximum data transfer speed will be determined not by instruction execution time, but by the memory read write times and also the IO data transfer times. And this method will be much faster than the programmed IO techniques.

(Refer Slide Time: 14:19)



Pictorially DMA controller is connected like this. You see when you have the processor or the CPU connected to the memory, there are 3 different buses: one is of course, the address bus, second is the data bus, and third bus will contain the different control signal like read write enable.

When CPU is interfaced with memory, all these buses are connecting to the memory. And CPU is in full control. CPU generates an address, issues a read signal, the data will

be read from the memory. Similarly when it wants to write it puts an address, puts a data, then puts a write signal, the data will be returning to memory.

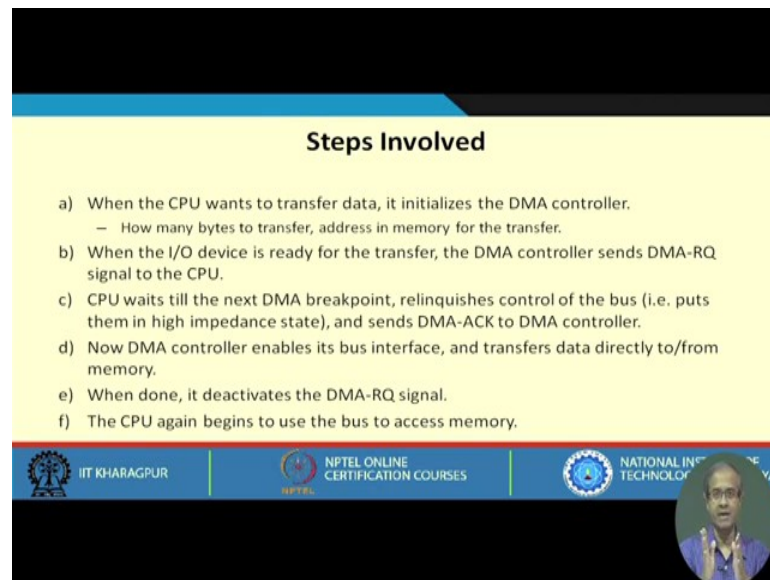
But now there is a third entity the DMA controller also sitting and wanting to use the bus. I am using wanting to because while CPU is using, DMA controller cannot use the bus. So it will set all the signals to the high impedance or tristate. Putting the signals in high impedance state means they are electrically disconnected. Normally CPU and memory are connected and they are working like that.

But now suppose the disk or the CPU wants to transfer some data, from the disk and memory. Now, the DMA controller will wake up. What the DMA controller will do? It will send a DMA request signal to the CPU; this signal will tell the CPU that the DMA controller or some device is wanting access of this bus.

CPU will be receiving this well; CPU might have been using the bus. After sometime CPU will relinquish control of the bus, means now CPU will make it is own bus tristate. CPU will be disconnected. And after doing that it will be sending back a DMA acknowledge signal DMA ack.

When the signal reaches DMA controller, it knows that CPU has disconnected itself on the bus. So, now DMA controller will enable its own bus and will directly access memory. This is roughly how it works. The DMA controller and the CPU has a handshaking mechanism via the DMA request and DMA acknowledge signals, by virtue of which exactly one of the 2 devices are having their bus signals enabled, the other one is having them in the high impedance state. In normal situation CPU is accessing, and while transfer is going on DMA controller is accessing.

(Refer Slide Time: 17:54)



Steps Involved

- When the CPU wants to transfer data, it initializes the DMA controller.
 - How many bytes to transfer, address in memory for the transfer.
- When the I/O device is ready for the transfer, the DMA controller sends DMA-RQ signal to the CPU.
- CPU waits till the next DMA breakpoint, relinquishes control of the bus (i.e. puts them in high impedance state), and sends DMA-ACK to DMA controller.
- Now DMA controller enables its bus interface, and transfers data directly to/from memory.
- When done, it deactivates the DMA-RQ signal.
- The CPU again begins to use the bus to access memory.

The slide footer contains logos for IIT KHARAGPUR, NPTEL ONLINE CERTIFICATION COURSES, and NATIONAL INSTITUTE OF TECHNOLOGY KHARAGPUR, along with a small video inset of a speaker.

The steps involved as I had said, when the CPU wants to transfer data, the first is that it initializes the DMA controller. Because it has to tell something to the DMA controller. Like how many bytes to transfer what is the size of the block. If it is read or write, where from in the memory we have to transfer the data, etc.

Second when the IO device is ready for the transfer, the DMA controller will send the DMA request signal to the CPU. CPU will wait for some time, wait for the next DMA breakpoint, I will explain this a little later. DMA break point means the time step where CPU can release the bus. CPU will wait for that and once the next DMA breakpoint comes it relinquishes control of the bus; that means, puts them in the high impedance state. And after doing this it will be sending DMA acknowledge signal to the controller.

DMA controller is now having control. It will now enable its own bus interface and will transfer data directly from the memory and the device. When the DMA controller has finished transferring the data, it will deactivate the DMA request signal, but before that it will be disable its own bus lines, so that now the CPU will know the DMA controller has finished and I can again gain control of the bus by enabling them. So, CPU again begins to use the bus. Now here one thing I have not mentioned; there is also an interrupt signal which is coming in to the picture. After the DMA controller has finished it can send and interrupt the CPU telling when the transfer is complete fine.

(Refer Slide Time: 20:17)

The slide features a yellow background with a blue header and footer. The main content includes a bulleted list, a diagram of an instruction cycle, and a question. The footer contains logos for IIT Kharagpur, NPTEL, and National Institute of Technology.

- The DMA breakpoints:
 - DMA request can be acknowledged at the end of any machine cycle.

DMA breakpoints

IF ID EX MEM WB

Instruction Cycle

Why cannot we have interrupt breakpoints at the end of any machine cycle?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY

Talking about the DMA breakpoints, when we discussed interrupts we mentioned that the instruction execution cycle can be divided into 5 machine cycles IF, ID, EX, MEM and WB. You recall for interrupts we were acknowledging the interrupt only at the end, but for DMA we are saying that we can have DMA acknowledged at the end of a machine cycle also. Now why this difference let us try to understand.

In interrupt processing what was happening, whenever there is an interrupt you jump to an interrupt service routine, you save the status registers and other information, execute the ISR, restore the status, and then come back to the point from where you were interrupted. This is how interrupt works.

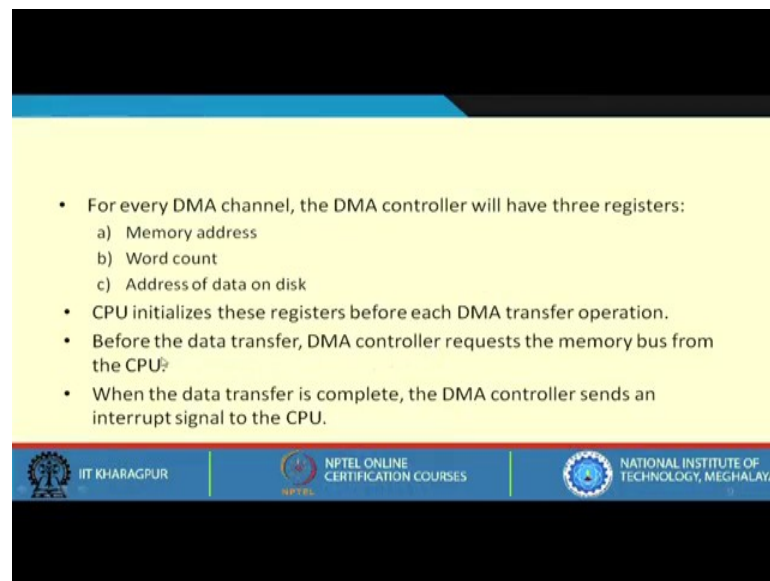
Now, you see if we allow interrupts to be acknowledged in between the machine cycle, the only trouble is that you will have to store or save much more information than just the registers. Suppose you are stopping the CPU at the end of the ID cycle and instruction had completed ID instruction decode that time you are stopping interrupting and you are jumping to the ISR. Here you will also have to store all the intermediate registers, the instruction that was fetched in IR, after decoding that sigh extended value. All those things also will have to be saved. So, unnecessarily you are complicating the saving and restoring process.

But in DMA there is nothing to save and restore because you are not interrupting the CPU. You are just putting it in the pause mode. You tell the CPU just wait for a while let

me do it after that you resume. So, it is not like interrupt that the CPU is interrupted the CPU jumps to another program, ISR executes it and then comes back. In DMA it does not happen that way. CPU is just going into the pause mode.

So, let us answer this question. Why cannot we have interrupt breakpoints at the end of any machine cycle? Because we will have to save and restore lot of status information.

(Refer Slide Time: 23:05)



The slide contains a list of bullet points describing DMA controller registers and operations. At the bottom, there are logos for IIT Kharagpur, NPTEL Online Certification Courses, and the National Institute of Technology, Meghalaya.

- For every DMA channel, the DMA controller will have three registers:
 - a) Memory address
 - b) Word count
 - c) Address of data on disk
- CPU initializes these registers before each DMA transfer operation.
- Before the data transfer, DMA controller requests the memory bus from the CPU.
- When the data transfer is complete, the DMA controller sends an interrupt signal to the CPU.

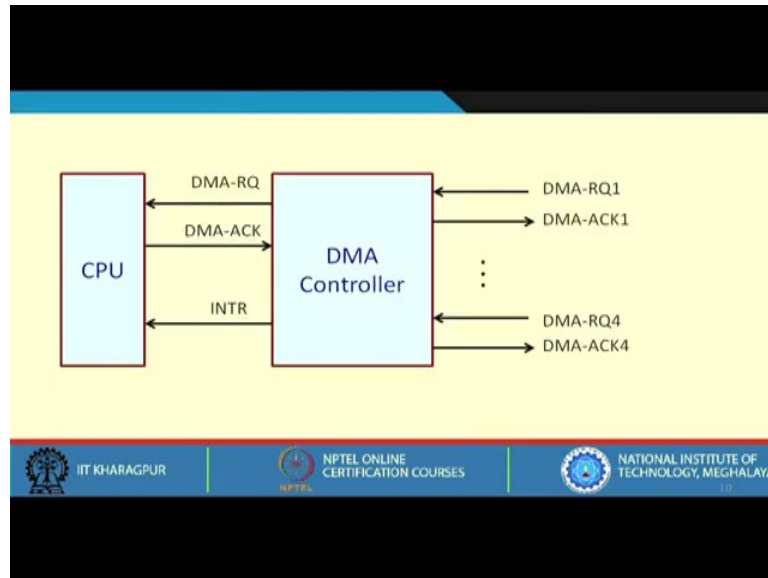
For the DMA controller for every IO device that is connected to it, we call it a DMA channel. There are 3 registers which are there inside the DMA controller. Memory address register, word count register and another register, which stores the address of data on the disk.

Before a DMA operation starts CPU has to initialize all these 3 registers. Because CPU has to tell, for a write operation this is the memory address where the data is stored. It will also tell how many words to write, that is the data count. And lastly it will also have to tell in disk where to write, which sector number, which track number, etc. So, these 3 things CPU has to write into the registers and only then DMA controller can take over.

This is one thing I did not mention in the previous diagram, before the data transfer DMA controller requests the memory bus from the CPU by sending the DMA request signal. But when the transfer is complete typically there is also an interrupt signal which is activated. Because when the transfer is complete it is not just that CPU can be brought

out of the pause mode, you also tell the CPU that well you had initiated an IO operation, now the IO operation is complete; it is in memory if it is an input operation. Now you can do whatever processing you want to do on that data.

(Refer Slide Time: 25:00)



Pictorially speaking, this will be the interface. DMA controller will be interfacing with CPU with DMA request and acknowledge, and in addition the interrupt request. And on the other side for every device or group of devices there will be a separate request and acknowledgement.

Whenever any of the devices sends a request, DMA controller in turn will inform the CPU by activating DMA request. CPU will be relinquishing the bus sending back acknowledgement, and then DMA controller will be using the register values corresponding to this device to carry out the transfer. For all these devices there will be different sets of registers.

(Refer Slide Time: 25:53)

DMA Transfer Modes

- DMA transfer can take place in two modes:
 - a) DMA cycle stealing
 - The DMA controller requests for the for a few cycles 1 or 2.
 - Preferably when the CPU is not using memory.
 - DMA controller is said to steal cycles from the CPU without the CPU knowing it.
 - b) DMA block transfer
 - The DMA controller transfers the whole block of data without interruption.
 - Results in maximum possible data transfer rate.
 - CPU will lie idle during this period as it cannot fetch any instructions from memory.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

DMA transfer broadly speaking can take place in 2 different modes. The first mode is called DMA cycle stealing mode. Cycle stealing mode conceptually is like this, you see CPU is normally working, it is accessing memory because for every instruction execution it has to fetch.

Now, what the DMA controller is doing? DMA controller will be grabbing the bus from the CPU by activating DMA request, but only for a very short time, only for the transfer of 1 or 2 words of data. It will not continuously transfer the whole block. It will transfer 1 or 2 words and then it will again give bus back to the CPU.

Well this serves 2 purpose, first the CPU need not have to be paused for very long. Because if it is a long block to be transferred if you transfer it in one go, then the CPU will have to wait for the whole time. And secondly, CPU does not necessarily access memory in all the machine cycles. For the MIPS memory is accessed only during IF and MEM.

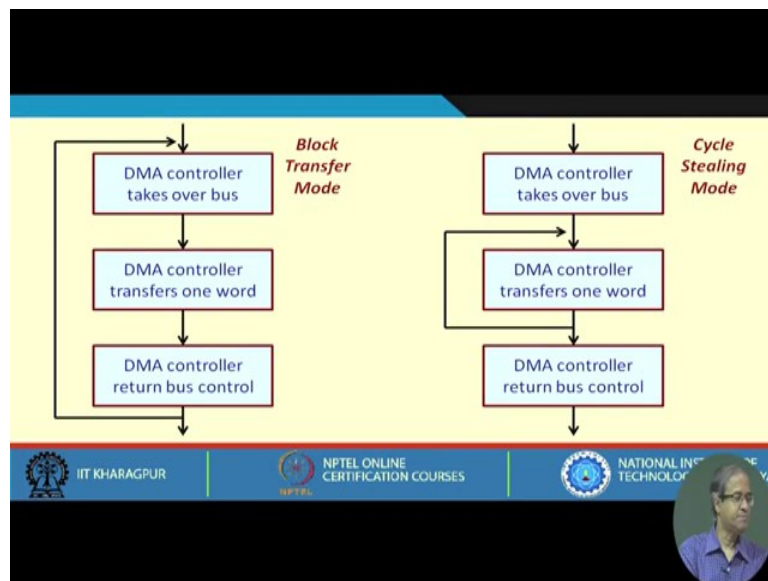
Suppose when DMA request came CPU was in the ID cycle. So, internally CPU execution can proceed because ID does not require memory, ID can proceed; EX also does not require memory, EX can also proceed. So, some of the cycles which CPU is not using, the DMA controller can to steal from the CPU. CPU is also not getting hampered due to that.

So, the DMA controller requests for a few cycles, just 1 or 2 cycles; preferably when the CPU not using memory. Here we say that the controller is stealing cycle from the CPU. So, while the CPU may be doing something internally, some machine cycles have been executed which does not require memory.

The other alternative is you transfer the whole block in one go. The DMA controller transfers the whole block of data without interruption. This of course, will result in very high data transfer rate, but CPU will have to lie idle during the period of transfer.

If you see that there is a continuous stream of data coming at a very high speed, then you may have to go for block transfer block mode, but if you see intermittently data are coming where this speed is not that high, you can go for cycle stealing mode.

(Refer Slide Time: 29:25)



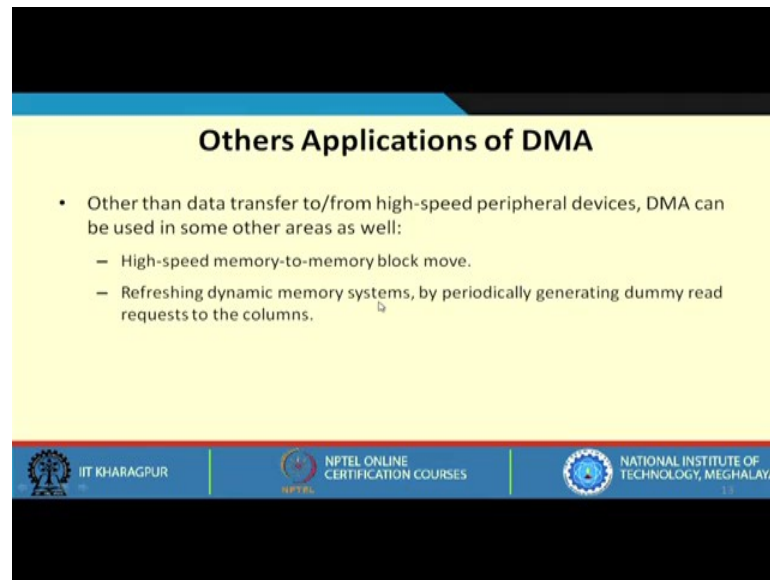
When you are initializing the DMA controller for a particular device, you also specify which mode of DMA transfer you want.

This can be pictorially represented as follows. In the block transfer mode the DMA controller will first take over the bus, then the transfer of the word will take place. Then the DMA controller will return the bus control. Now this is done in a loop.

For every cycle for every data transfer DMA controller will take over the bus, transfer the data, and return the bus. Again it will go back for the next word, it will again take over the bus, again transfer the word, again release the bus. So, actually this is the cycle

stealing mode and this is the block transfer mode. In the block transfer mode the DMA controller will take over the bus once and after that words are being transferred continuously till the whole block gets transferred. After this is done the DMA controller will return control of the bus.

(Refer Slide Time: 30:52)



Others Applications of DMA

- Other than data transfer to/from high-speed peripheral devices, DMA can be used in some other areas as well:
 - High-speed memory-to-memory block move.
 - Refreshing dynamic memory systems, by periodically generating dummy read requests to the columns.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | NATIONAL INSTITUTE OF TECHNOLOGY, MEGHALAYA

DMA is used not only for transfer of data in IO devices and memory. There are other interesting applications also. Like, you think of a high speed memory to memory block move. Suppose you have an application where you want to transfer, let us say, large block of data from one part of the memory to another.

What is the normal way to write program? We will be reading the data one word at a time, read and store in the other place. Again this will involve instruction execution and the data transfer of the block will be slower. But you can use the DMA controller for this purpose also. Here DMA controller instead of interacting with the device it is interacting with memory and memory in both sides. So, it will read from memory, it will write into memory. And these 2 things are done by the hardware, no need of any instruction execution. So, the block transfer can happen very fast.

The other application where it is used is in case of dynamic memory systems. You see dynamic memory system requires periodic refreshing. For this you have to issue some dummy read signals to the rows or columns depending on how the memory is organized.

DMA controller can be used to periodically generate such dummy read requests, so that the dynamic memory systems are getting refreshed. Of course, nowadays the DRAMs that are available are intelligent enough; this refreshing circuitry is also built into those boards or the cards or the chips. So, you need not have to do it separately.

With this we come to the end of this lecture. Over the last few lectures we had looked at various data transfer techniques between the CPU and the external IO devices, which are also called peripheral devices. In the next lecture we shall be just looking at two simple devices and look at some issues regarding the interfacing or how the data transfer can take place.

Thank you.