Computer Organization
Part – II
Memory
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture – 16
CPU – Memory Interaction

In the previous lecture, we saw how memory generally gets organized and also saw some details about the memory or different classes of memory in a very systematic way; we saw a few things which are more important for general understanding and from practical point of view. Now while talking about it, we said that essentially it is the speed match between the CPU and the memory module, which gives rise to this hierarchy of memory. That is, we talked about the type of memory, which can interact with the CPU at the CPU's speed itself – the one that is slower and the one that is further slowest. Now before we go into the details of the memory there are a few more points we have to take note of.

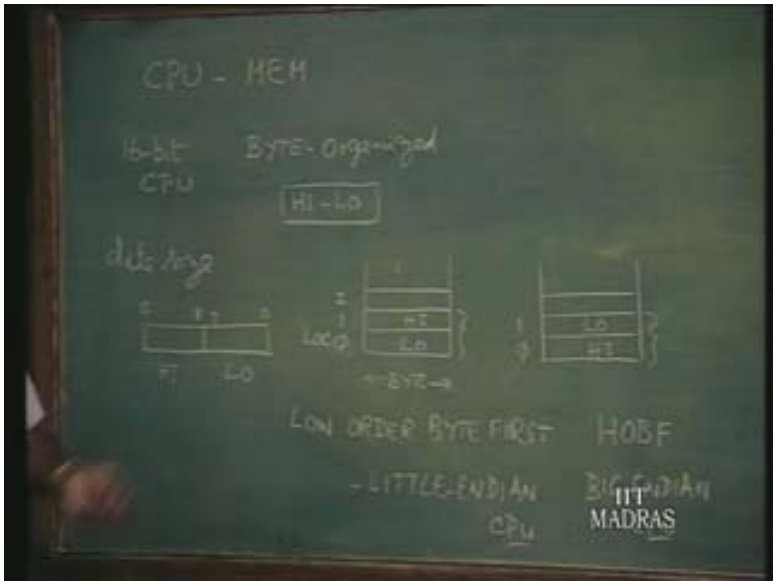
First, we always talk about something like an 8-bit CPU, 16-bit CPU and so on, or 32-bit CPU. Now how is the memory organized with reference to these different types of CPU: does it mean that if you have an 8-bit CPU, the memory must also always deal with 8-bit data, because essentially when we talk about an 8-bit CPU what exactly we mean is internally the CPU can process 8 bits at a time; that is its word size. Then, if it is a 16-bit CPU, essentially what we mean is internally the CPU is capable of 16-bit processing, that is, processing 16-bit data as one unit. That may be so. For instance, you may be having a 16-bit CPU, but the memory need not really have a word 16-bit wide. You can still have the unit; that is, we can loosely say that may be it still is an 8-bit word; that is a byte.

So generally, what we say is that the memory is usually byte organized – now there is a specific reason for that. We say the memory is byte organized, which deals with CPU that is classified as 16-bit CPU; then essentially a byte is one 8-bit unit, so 2 such bytes must be brought in here. We are not really talking about what is the bus; here it is meaningful if the bus is also 16 bits so that the CPU may fetch the entire unit of information in one go; you can process and then carry out the processing. But then the memory may be byte organized. By that what I mean is, whenever the CPU needs, you can always go and fetch less than 16-bit data also.

That is what the memory organization permits. In other words, we can talk about a low byte and a high byte, which together form one 16-bit word. Now anytime this low byte can be separately addressed and then brought; a high byte can be separately addressed and then brought if necessary. So we have what we make call the data size of the CPU and the way we organize the memory size, which generally we keep as byte organized. Because of this, we have different class of CPU itself. Suppose we will continue with these examples of 16-bit CPU as indicated here, essentially it consists of a 16-bit word and it consists of the low byte and high byte – this is what we have from the CPU end. At the memory end, suppose it is a byte organized memory, which means the width of the unit of information that is addressable is byte wide, and then we can talk about these as location 0, location 1, and location 2 and so on.

There are two types of process; for instance, suppose this location 0 holds the low byte and location 1 holds the high byte, together these 2 bytes are needed for the processor.

(Refer Slide Time: 10:17 min)



Location 0 has the low byte; location 1 has the high byte. As you can see, the low-order byte comes first in this location: this is the first location. So this particular thing comes in the first; and this arrangement is called a low order byte first and it is also known as Little-Endian arrangement and the processor is generally called a Little-Endian processor. In this, 2 bytes may come from the memory; whatever comes from the first location, that is, location 0, will be the low order; the other one will be the high order. Then such CPU will be called a Little-Endian CPU. It can be the other way also. Instead of this, we can also have an arrangement in which location 0 may hold the high byte and location 1 may hold the low byte.

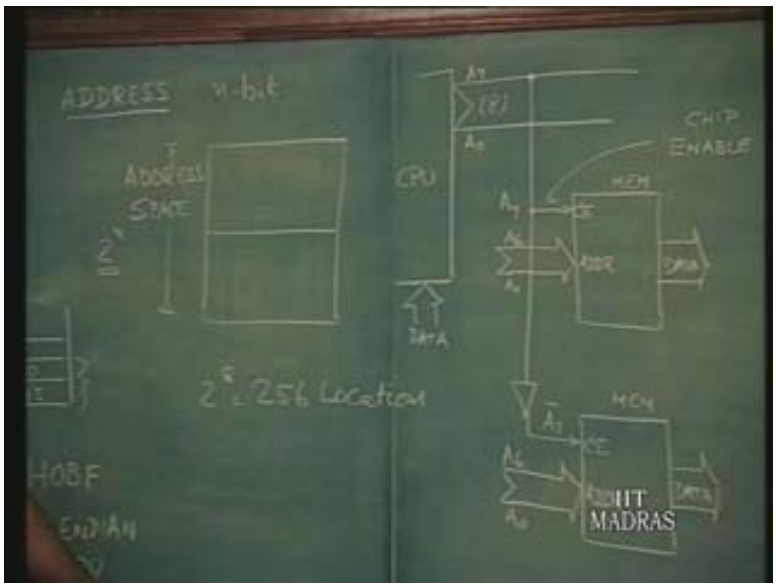
Together, still we have the 16-bit data. So what we are talking about here is the memory organization, mainly because it is a byte organized memory and the processor is a 16-bit processor; it is capable of dealing with 16-bit data. This particular one obviously will be called the high order of the byte as it comes in the first location. So that will be called high order byte first. I will just write the simplification, the abbreviation of it. Instead of low order byte first, what we have in this arrangement is a high order byte first, and as you can see this other one is called a Big-Endian CPU. This one is a Little-Endian CPU here and this one will be called a Big-Endian CPU. Both the types of CPUs are in use. For instance, I would like you to take a look at how Intel CPU organizes its data and how Motorol organizes its 16-bit data and you may see for yourself which Little-Endian is and which is Big-Endian.

From the memory point of view, as long as it is the byte organized memory, what we say is it is going to hold one unit of byte, which is addressable from memory point of view. Whether it is slow or high does not matter because it is the CPU that is going to fetch this data and then proceed. On the other hand, instead of a 16-bit CPU if you have an 8-bit CPU, here also the unit of information is going to be just 1 byte.

Another point is that as far as the data part of it is concerned – let us just take a look at the address part of the CPU also – the CPU places the address and how this address may be used in addressing the memory or choosing different parts of the memory and so on. We know that if with an n bit address, the total address space will be 2^n. That is, we talk about total address space; this is also sometimes called just the memory map – the map of the memory – the total address space. This whole thing would be of size 2^n because of the n bit address. Now let us draw a few sketches and then try to see how we may make use of this n bit address in addressing the memory. I think I will assume for the sake of simplicity that the address size is 8 bits; it is the same logic you can extend for 16 bit also. That is, we have address, let us say, from A_0 to A_7. That is, it is an 8-bit address I am assuming A_0 to A_7. With this, 2^8, the maximum address will be only 256.

Now let us – I can go to a maximum of 256 locations: that is what I can have on the memory side. Supposing I take this bit A_7 specifically, and then use it as one of the inputs to the memory part, this can be an address input to the memory. It can be one address input or it can also be one of the control inputs. I will just call this something like an enabled signal which, when on, activates the memory chip or system for some work. If it is at the chip level, then generally we call it as a chip enabled input.

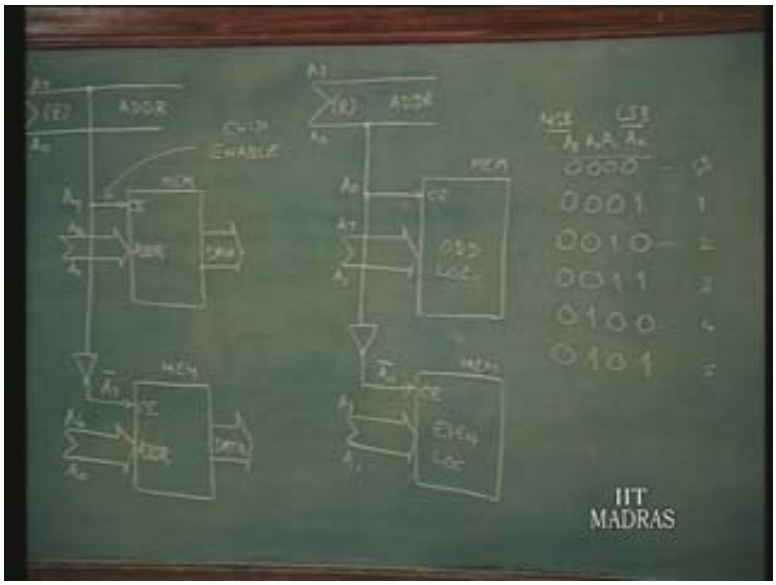
(Refer Slide Time: 18:03 min)



This particular one only enables this memory chip and so for effective address I get one bit less. It only enables, means this block is enabled. On the other hand, just see what I am doing – I will take the same thing, same bit input, and then let us say I invert that and generate another signal which we call A_7 bar and then have another memory chip, which I give to the other as its chip enabled input. That is, this is a chip enabled input. It means whenever A_7 is 0, this will not be enabled. So this part of the memory will not be in use. Whenever A_7 is 0, this is inverted and this will become 1. This is enabled and whenever A_7 is 1, this is enabled, whereas this is not enabled. So at any time, depending on whether A_7 is 0 or 1, either this or this appropriately is enabled. I can give the rest of them, A_0 to A_6, as an input.

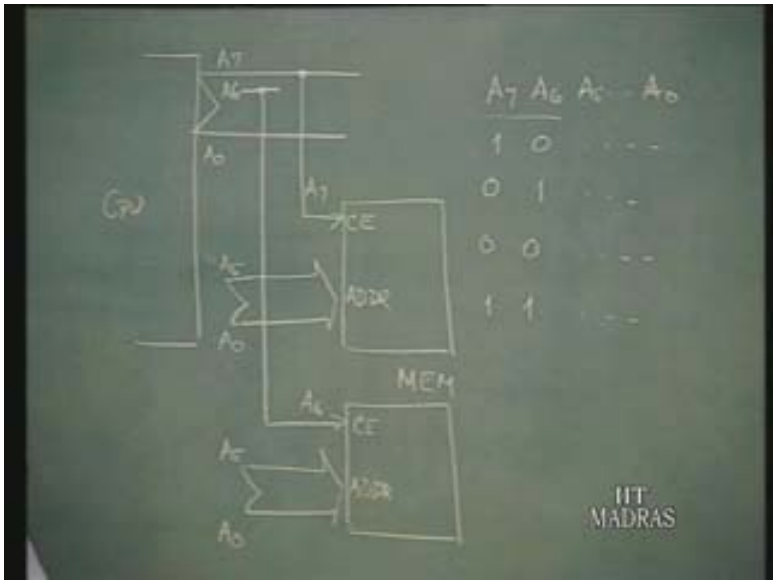
These rests of them, that is, A_0 to A_6, which of course are derived from here and though are not shown specifically, I may give the same A_0 to A_6 to this also. That, of course, is the address input. When the memory chip is addressed, the other signals like read, write also must be given. We are not showing them here when it is enabled and when the appropriate control signal like read comes, it will produce the data and that data will be going to the data part of the bus and it will go back to the CPU. I will just mark it; if necessary we will discuss or leave it. So the address input to the memory comes from the CPU and also the necessary control signals, read or write. Similarly the data output from the memory will be going to the CPU; maybe I will just show it here. This is the data part, which goes to the CPU. If it is bidirectional, it may come to this and also go there.

(Refer Slide Time: 25:22 min)



The control signals are needed – let us not worry about that for our discussion now. What I have done as long as A_7 is 1, this chip will be enabled and as long as A_7 is 0, this chip will be enabled. In other words, from the total address space here, for n we have 8 bit; so we have 2^8 to 656 and A_7 is the most significant bit. So for the most significant bit, when it is 0, it means it is the lower part of the address space. When the most significant bit is 1, it is the higher part of the address space. That is, if this is the lower part, this is location 0, or the lowest part, and this one corresponds to the highest location, 2^{n-1}. In our case, for 8 it will be 2^8 to 256 and so this location number 2 is 55. Because we start from 0, we go up to 256. So from 0 to 1, location 1 to 7, that is, location 0 to 1 to 7 corresponds to A_7 being 0. So this will correspond to A_7 being 0, and this part will correspond to A_7 being 1.

(Refer Slide Time: 33:38 min)



We are making use of this total address space in this particular manner. You can extend this argument to any n bit. So you can see that apart from the address input to the memory chips, the same address output from the CPU can also be used not only as input to the memory but also for specifying some control signals. In this case, we are using it for enabling that part and so we can say that this location 0 to 127 is going to be accommodated in this memory chip and location 1 to 8, that is, the next one, location 1 to 8 to 255 is going to be accommodated here.

Alternatively, I just extend this logic a little. On the other hand, instead of using A_7 and A_0 as shown here, if I take two memory blocks or two memory chips as before, instead of A_7, if I use A_0 here for chip enable and invert, and use this A_0 and this A_0 bar and invert this for chip enable, and if I use A_1 to A_7, see for yourself the difference between these two. We saw that A_7 being 0 is the lower part; A_7 being 1 is the higher part of the total address space. Now in this one, this is the address bus just like what we have here – we have two different schemes here. In this, we can say the LSB is used. But of this address bits, the MSB or most signaled bit is used for enabling the least significant bit; A_0 is being used for enabling. What is the difference between these two? We can use the same argument – we said whenever A_7 is 1, this will be enable whenever A_7 is 0, this is enabled; the same thing holds good here also. Whenever A_0 is 1, this will be enabled; whenever A_0 is 0, this will be enabled.

Let us take a 3-bit address or a 4-bit address: let us just do for a 4-bit address. What do you find? This is your A_0; this is your A_1; this is your A_2; this is your A_3 – in other words this one is the most significant bit. This is the least significant bit. Because of these, A_7 will be MSB; that does not matter. Now just see what is happening – what is the next number? Next will be 1; the other number will be 10; then you have this: what is the pattern you see here? You keep seeing for every alternative code, that is, address code, this LSB keeps becoming alternatively 0 or 1. What is the equivalent value in decimal of this? This is 0 in decimal; this will be 1; this will be 2; this will be 3; this is 4; this is 5; and so on. So whenever we say alternate thing, you find that these addresses correspond to even addresses and these correspond to odd addresses.

In other words this particular one memory block or group will correspond to A_0 being 1, which means odd. So all these odd locations will be here and all even locations will be here. Is it useful in some way? Yes, in some places, it is useful. For instance here if you just see, all even bytes will be low bytes; all odd bytes will be high bytes. So we can possibly store odd bytes in these even bytes here and then make use of this scheme and then address these bytes appropriately. This is also useful. So in other words, here what you can see is location 0 comes from this block; location 1 comes from this block; the other way location 0 comes this block; location 1 comes from this block. The content of location 0 comes from here; the content of location 1 comes from here; content of location 2 and so on. In other words, it is like alternating interleave; this in fact is called an interleaved memory organization. Now all this comes just because of the way you are choosing the address and addressing the memory part. This is an interleaved memory organization. We are talking about blocks or contiguous addresses here; contiguous addresses location 0, 1,2,3,4, up to 127. So we talk about a block. This kind of starting from MSB actually leads to the design of memory by blocking them. So we generally call each of these blocks of memory; so you can call this as follows: A_7 must be 0 for enabling this, so this is the low block and this is the high block of memory.

Now instead of A_7, if you choose some other one, say, A_6, it will be another block. In the same way, instead of A_0 if you choose A_1, you can see A_1 is see here; you can see these two 0s, these two 1s, these two 0s and so on. So then you are interleaving memory, taking two locations. Similarly, you can go on doing. That is about this interleave memory part of it. One thing I would like to indicate here is in this series, A_0 to A_6, we have taken 7 bits and then this 7-bit code will have to be decoded. A_0 to A_6 is 7 bits, so it is 2^7 or 128. That will have to be decoded here, whereas here we take A_7 and directly give as 1.

Next, we will do one more thing: we will take A_7 and A_6 just for example and then see what we will be getting out of that. Instead of taking only A_7 for chip enable, if you take A_7 and A_6, for instance, we will get like this. That is, A_7 is used for enabling this part of the memory, this block of the memory; and A_6 for the other block. Because both A_7 and A_6 are being used for chip enable, it is obvious that when A_7 seven is 1, this will be selected and when A_6 is 1, this will be selected. Of course, when both of them are 1, both will be selected; so let us not worry about that now. Because we are using A_7 and A_6, we are left with only A_0 to A_5 and that is what is being used to address these two specific blocks, thereby our addressing rates comes down; the addressing capacity comes down.

What I am interested in showing these is A_0 to A_5 are used like this: they can decode the entire A_0 not to A_5, that is, 16 bits, and get 1 of 2^6. In A_0 to A_5 there were 6 bits and so the addressing address space size will be 2^6 and at any time only one of these addresses will be selected. Now A_7 is used here, A_6 is used here individually, whereas A_0 to A_5 are collectively used. In other words, here this A_0 A_5 code will have to be decoded to get the address, whereas here directly A_7 and A_6 are given. You have to recall here once whatever we were talking about earlier. I said whenever you have a code, either you can do a linear selection, that is using 1 bit of these of the code at a time, or you can use the entire n bit code and decode and get. So there are two procedures we were talking about earlier, remember? That is, use a decode procedure and make use of an n bit code for each bit of the n bit code you use.

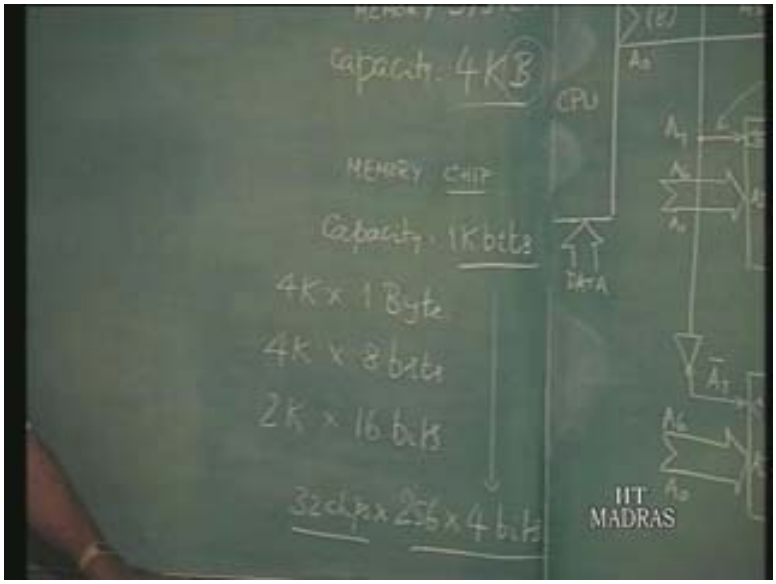
So that is what you have seen here. Linear selection is used – I was also mentioning 1 bit at a time; only 1 bit at a time – that is why if you have A_7 as well as A_6, you use both or one of them. Suppose you have an address, let us say, A_7 A_6 A_5 to A_0. Suppose you have an address like 10 and then another address, 01, A_0 to A_5 and say another address 00 – some address – it is going to be referred to some specific location and then another one. Now look at these four combinations: we specifically concentrate on A_7 and A_6. So when A_7 is 1; A_6 is 0, this is going to be selected. Depending on some code A_0 A_5, one specific location, this will be addressed. When A_7 is 0 and A_6 is 1, this will not be selected; this will be selected, and which specific location depends on that particular address A_0 A_5. These 0, 0 – none of them will be selected. So there is no memory location address. Now what happens is A_7 and A_6 both are selected. Sometimes it may be permitted; in some cases, it may not be permitted.

For instance, suppose this CPU requires an 8-bit data, and if these memory locations are going to say output is 8-bit data, both of them – here I am not showing that let us say that output also is an 8-bit data – then what happens? Both of them will be outputting 8-bit data and the CPU is going to receive. That means there is going to be a clash if both A_7 and A_6 are 1; that is, both are getting selected. Suppose this is a 16-bit CPU, these continue to give only 8-bit data; then possibly I can organize this to give the low byte and the other one high byte, in which case there is no problem. So in some cases this combination is permitted; in some other cases it not be permitted. Anyway, our point here is to note that the address lines can be used in both ways: that is, either you can have a bunch of address bits and then decode them and use them, or use each one of those address bits and select different parts of the memory in a linear select manner; so both the things are in use. That is as far as the addressing part is concerned.

There is another point which I would like to mention before I go to a discussion of something else. When we talk about memory, there are two things we may talk about – one is memory system or subsystem as a whole, and another one is the components in that, say, for instance, memory chip that constitutes the memory subsystem. So essentially for instance this may be a memory chip; this is another chip; and so on. Now, together with these chips which hold the memory data, that is the data in the memory, plus the controller, all these things will form the memory system. So the CPU looks at the memory system as a whole, which consists of different chips.

Now suppose we have a memory system of capacity, we talk about the capacity, say 4 kilobytes. There are many ways in which you may be able to organize this particular one. How? For instance you can have 4 K addresses, and each address is giving out 1-byte data, this is one possibility. The other thing is that it has 8 bits; in other words what I am talking about is 4 K byte; instead of byte, talk about 8 bit, or the other way. The total capacity of it is 4 K byte; we may have 4 K addresses, each of 1 byte. The other thing is you may have for instance 2 K addresses, each of 2 bytes or in other words 16 bits. You have many ways in which we can organize. Or, instead of 4 K by 1 byte, we can also have 8 K of 4 bits; instead of 4 K by 8 bit, we can have 8 K of 4 bits also. All these things finally would mean, from the CPU point of view, 4 KB memory system – that is all.

(Refer Slide Time: 41:22 min)



Generally whenever we talk about capacity of system, it will be indicated in terms of so many bytes, whereas when it comes to talking about the capacity of the chip at the chip level, generally it is a bit level. That is what you find in the data chips. So whenever you see capacity at the system, it will be done at the byte level. For instance, we can say suppose we have a chip of 1 kilobyte, and we want to use the chip, we want to construct a memory system of 4 kilobytes. From this we can see that we do not know how this 1 kilobyte is organized. Possibly it is organized like 1 K – there are 256 locations of 4 bits each; let us assume here also there can be many variations. For instance you can also have 1 K of 1 byte; that is also possible. If I have two, the capacity of this particular chip is 1 KB, which is actually 256 by 4 bits; basically it means this chip has 256 locations.

From each location I can get the data of size 4 bits. For using this chip, I want to construct 4 kilobytes memory system, so here I need a byte-organized memory. For byte organization I need two such chips. With two such chips I will get 8-bit output. So with two such chips I only have as many as 256 locations; and what do I want? I want 4 kilo locations. So, 256 into 4 will be 1 KB; that means with 8 chips I can get 1 KB. So with 32 chips of that, I can have this memory system of 4 KB. This is what we have to remember. Whenever we talk about the chips, it will usually be referred to in terms of bits, and whenever we talk about system it will be in terms of bytes and the memory system maybe organized in many ways. In the same way, chips also may be giving output of different sizes.

So, 32 chips are there in this particular memory system. I think that is about the CPU memory interaction. Now we will go into some detail about whatever we have talked about earlier. The CPU always addresses directly; I said earlier in the previous lecture that the CPU addresses directly the one part of memory, which will be the fastest one, because then only the processor utilization is the highest. So whatever we have been talking about is usually the one which is closest to the CPU; there will be other circuit which will be coming with other parts of the memory system.

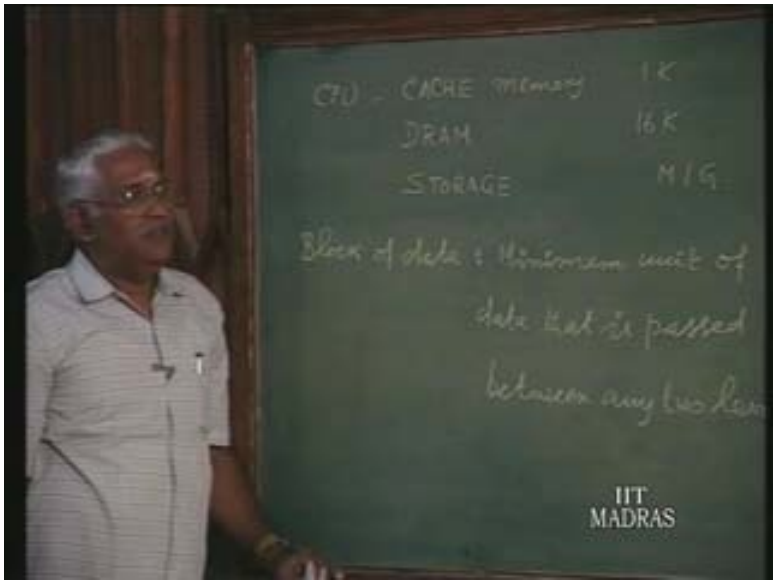
Now CPU always places an address and that address will be decoded and then the contents from that particular address will be sought first from the cache memory. We were talking about it earlier; about three levels of memory: cache memory and the second one will be the main memory because the main memory usually consists of dynamic RAM. We also just refer to it as DRAM memory, and the third level of memory is not really memory but it is storage. So usually it is disk or MAC tape so I will just call it as storage system. This is very fast; because it is very fast, the cache memory will always consist of chips which are very fast. Because it is very fast, it is always costly and because it is always costly the size of that also will be the least. So let us say we have 1 K of cache memory.

Generally when we say 1 K we mean 1 K byte; generally because it is byte organized. The DRAM let us say is 4 K or 16 K or something like that; here it can be more also. Just arbitrarily I am talking about here it may be some mega or giga and so on and so forth. The CPU addresses the memory; when we talk about the program address that would correspond to the address of the CPU that the CPU can handle. Let us assume the CPU has a 20-bit address; the 20-bit address will correspond to 2^{20}, that is, 1 megabyte address. If you assume a byte addressability, it is 1 megabyte for the addressability of that. Where is 1 mega byte and where is 1 K or 16 K? So the CPU can address as much as 1 megabyte but in DRAM or in the cache we have far less space, which means assuming there is a program which possibly need not be as much as 1 megabyte, but when the programmer writes he knows that is the 20-bit address this CPU can handle. So he will be generating a 20-bit address but he does not know the internal arrangements. So a 20-bit address may be generated but it has to be translated into some lower size of the memory, which means assuming a program of size 1 megabyte.

Then when we have a cache memory when we say CPU addresses or accesses only cache, normally it accesses only 1 K block. So 1 K of that 1 mega is possible; the CPU places an address and it does not find that particular data, which it wants here, because any time this is going to have only 1 K of total 1 mega. So the next thing is that if the data is not available in the cache, it will have to look for it in the next level. If it is not available there, it will have to come from here. Remember that the CPU will deal only with this. The CPU will not start looking for in the DRAM or storage or anything then what will happen? If that happens, the processor utilization will come down. So there must be some extra circuitry. The CPU will look for these; if not some other circuit will take care of moving the required information in the block of data from here to here.

So we always talk about something like a block of data, which can be moved from one level to the other level. Now this block of data, for instance, here we are talking about 1 K block there, a block of data which will be moved. This is the minimum unit of data; this entire 1 K need not be a block; there is some minimum unit of data that is passed between two levels whenever something is not available. When we say that, between any two levels normally we will be having a different set of issues. When we have to move these blocks of data from DRAM to cache memory or from storage to DRAM, there are different issues here.

(Refer Slide Time: 48:54 min)



First we will take up this particular one: CPU looks for some data in the cache; it is not available. Then the cache must get that particular block of data from the DRAM. This cache also will not be organized as 1 K: full block. We will see that when we discuss the details. Suddenly a decision must be taken to see which of the existing contents in the cache can be moved away because when we say we have to move something from the DRAM and then something must be moved out of cache, the same set of issues exist here also, because essentially we have 1 K and let us say the 1 K is full. But then CPU does not find what it wants in that 1 K, so something must be removed and then whatever it requires must be moved in. This is one specific issue; the same thing holds good from here to here also. So we have to see how this particular thing is organized. Now as long as the CPU finds what it wants in cache, then we say the CPU is a hit, that is, CPU addresses and gets the data then we say the situation is called a hit.

Whenever the CPU looks for the data but does not find it, then we call it as miss; that is, the CPU misses the data or CPU gets the data and proceeds with this processing further and if CPU hits and gets it, there is no problem. But when CPU misses, there is a specific miss penalty time that we have to work out. This penalty associated with missing must be kept as low as possible because, only then, the processor can be utilized well. We will go into these details in the next lecture.