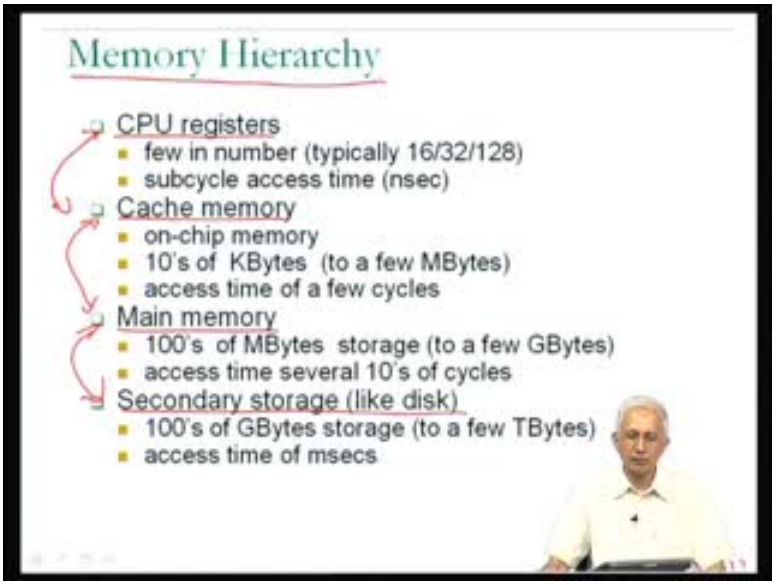
High Performance Computing
Prof. Matthew Jacob
Department of Computer Science & Automation
Indian Institute of Science, Bangalore

Module No. # 06
Lecture No. # 27

Welcome to lecture 27 of the course on high performance computing. We are looking at cache memories and in the previous lecture, I had introduced the term memory hierarchy, which we are trying to understand in little bit more detail.

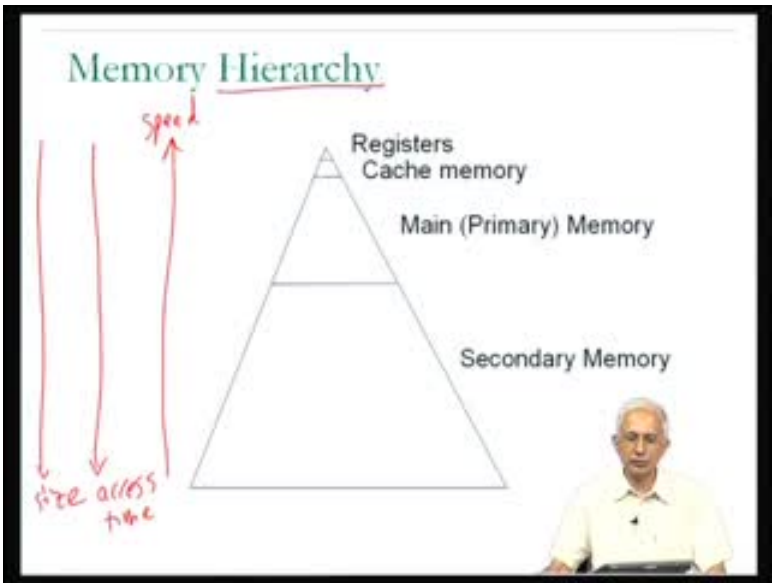
(Refer Slide Time: 00:30)



Based on our current look at computer organization, we recognize that there are many kinds of memory present in a computer system. Further, we understand that there is a lot of interplay between the different kinds of memory. For example, there is interplay between disk and main memory as a part of the implementation of virtual memory.

From our early discussion of cache memory, we understand that there is interplay between the main memory and the cache memory and we know that information is passed into the CPU registers explicitly, based on request in the form of load and store instructions between the CPU registers and the higher and the other kinds of memory that we see over here.

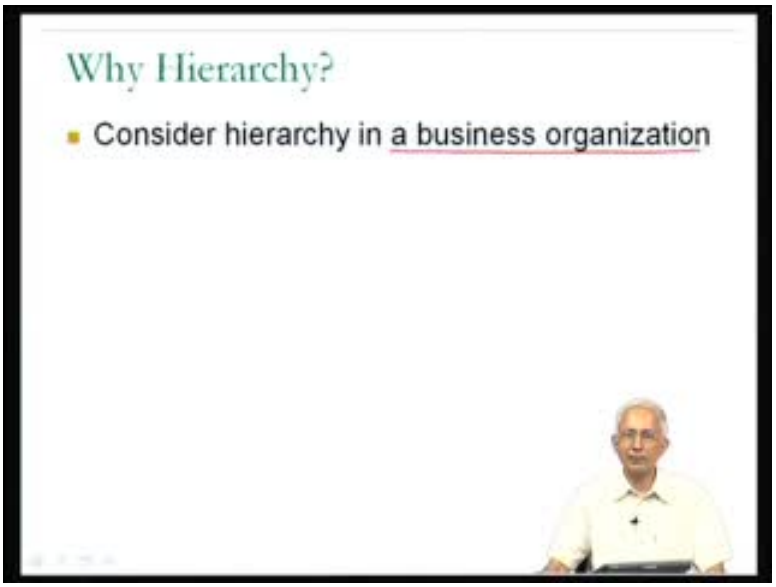
(Refer Slide Time: 01:20)



So, there seems to be something in this relationship between the different kinds of memory present that needs to be understood a little bit better, which we try to do partially in the form of a diagram which gives us a functional understanding of how the different forms of memory relate to each other. In this diagram, we were actually understood that as you went we are going to put different forms of memory into the diagram and as you go down the diagram, the size increases; in other words, the amount of information that can be stored in that kind of memory. Unfortunately, as you go down the diagram, the access time also increases. Another way viewing that is that as you go up the memory, the speed of the kind of memory increases. So, we have different ways of viewing the importance of dealing with different parts of the memory.

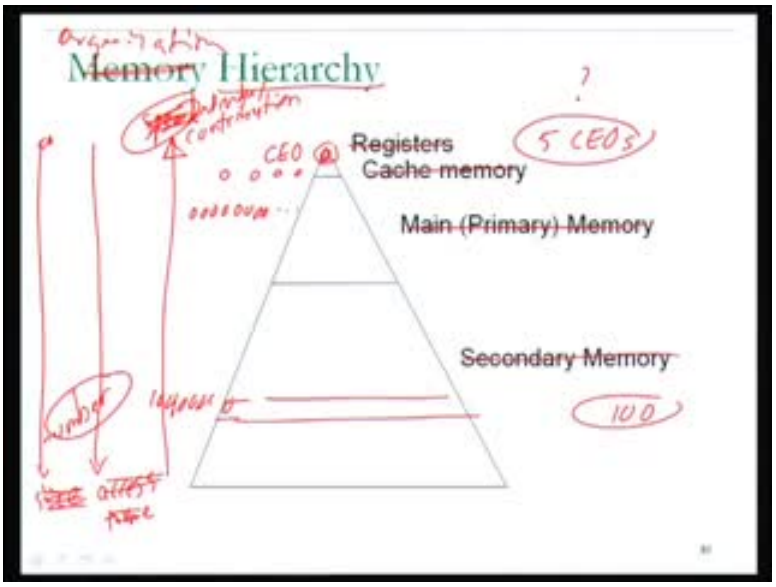
We put registers right at the top, very small in size, but very fast in speed. They have very low access time. That is why they are at the bottom of the access time arrow and the main memory somewhere in between, the secondary storage devices like disk are right at the bottom and the cache memory is intermediate.

(Refer Slide Time: 02:30)



Now, why is this called a hierarchy? That is the question which we need to get a better handle on today and in order to understand why the word hierarchy is appropriate in this context, we need to may be first understand whether the word hierarchy makes sense in a more general context and I am going to try to do this in the context of let us say, a business organization. It could be in the context of the administrative hierarchy in your college, if you are studying in a college or school.

(Refer Slide Time: 03:03)



But we will just think about this in the context of the hierarchy present in a business organization and so the question is, if I was to think about a business organization, could I think in terms of a diagram of this kind. Obviously, I would have to forget about the labels attached to each of these entities including the labels on these axes. Size, access time, speed, registers, cache memory, main memory etcetera do not make much sense in the context of the hierarchy in an organization, but what do we mean by the hierarchy in an organization.

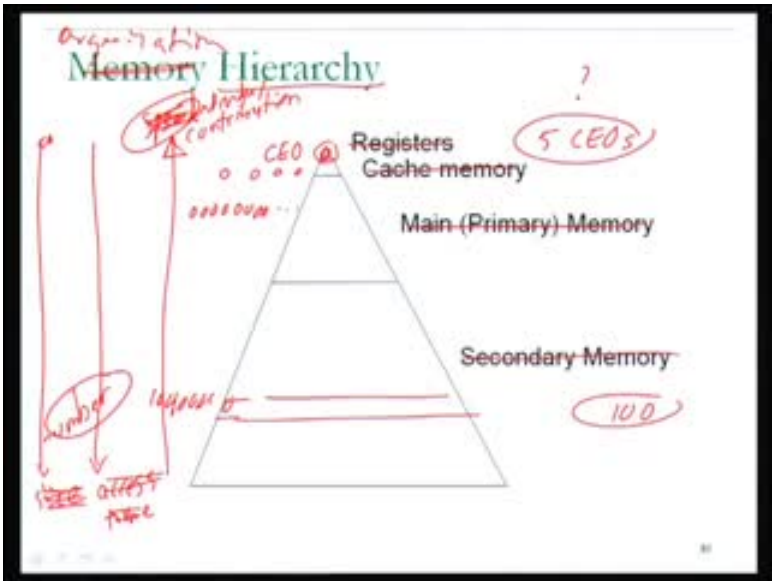
For example, can we identify your typical organization and you could at this point replace the word organization by any company or large company or organization, as I said even the administration in your college that you are familiar with and the question is can we identify let us say, individuals or an individual who will replace in the place of registers over here. In other words, are there individuals who are small in number and maybe, I will try to label this. Instead of talking about size, I could use the label number on this left most axis.

So, individuals who are few in number, but have a great impact on the efficiency of the organization. Rather than talking about speed, I could put something like individual contribution and again this is somewhat subjective labeling of that axis. But could I identify let us say, in an organization one person and that is the smallest possible number, who contributes tremendously towards the organization and typically, one expects that if there is such an individual, he might be the CEO; there may be only one chief executive officer, C star O, whatever the little initial may be. There may be only one of the individual, but he plays a very important part in the organization in terms of making critical decisions, in terms of coordinating those you need them, in terms of planning for the future etcetera, in terms of driving the organization. So, if there is one critical individual in the organization that is the person typically one would view as being the CEO. Now, lower down or just below that one could imagine that there are many divisions or divisions within the organization.

For example, there may be a division which handles finances and another division which handles human resource development and so on, each of which might be headed by an individual. So, there might be just two or three individuals at the next level, each of whom is in almost complete control of one of the key functionalities of the organization.

This is one way an organization could be structured. Not being an expert in organizational hierarchy, I could well be off the mark in terms of the way modern companies are run, but very clearly one could imagine that there are at the next level, small number of individuals; each of them plays a very important part in his own term of expertise and as one goes to the next level, one may find out that in each of the divisions that I just talked about, there are three or four vice presidents, each of whom coordinates some smaller area of activity within the organization.

(Refer Slide Time: 03:03)



If you are talking about a software organization and you are wondering what could come right at the bottom, it is conceivable that somewhere towards the bottom, there is this huge pool of programmers and there could be in a large company, there could be 1000's of programmers or 100's of 1000's of programmers, depending on which software developers as we shall call them, depending on which the scale of the company that you are talking about.

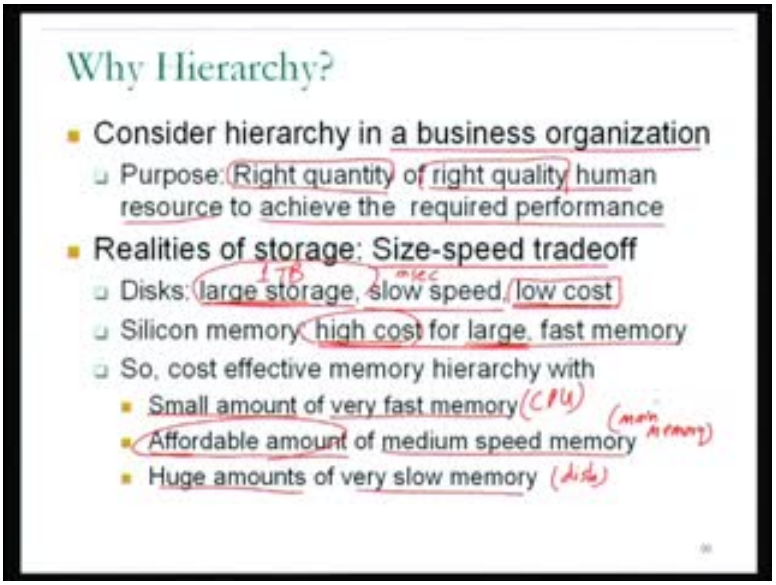
Once again they are large in number 100's of 1000's as opposed to the one CEO and each of them plays a very important part in terms of his job definition, but if one looks at the overall scope of activity within the organization, the activity undertaken by the individual would be small. All of these are key contributions and when added up they amount to a huge amount of benefit to the company, but the individual contribution will be much smaller than the individual contribution of somebody high up in the hierarchy.

Hence the labeling of number - now, there is a number of individuals of that kind there are in the hierarchy and the individual contribution gives us an idea of what it means to have an effective business organization.

We could readily understand that if I had a business organization in which there were 5 CEOs and that whenever a critical decision had to be made, all those 5 CEOs had to get together and discuss and arrive at a consensus. In other words, all five of them had to agree. There could well be situations where it was not possible and they would have to differ opinion. Consequently, they may end up in a situation, where rather than looking for consensus, they look for majority opinion.

So, until three of them could agree on something, the decision would not be made and this would clearly affect the efficiency with which the company or the organization is able to achieve its objectives. By the same token, if I had very few programmers, I just had 100 programmers and there were 2000 projects which the company had undertaken, then very clearly the organization is understaffed; it is not going to be able to achieve its objectives. So, too many at the top and too few at the bottom looks like the wrong way to view things from the perspective of the numbers.

(Refer Slide Time: 08:30)



In general, what this quick diversion into organizational hierarchy suggests is that in a business organization, the purpose of the hierarchy or you know that you have a good

hierarchy, if you have the right number of the right quality human resource to achieve the required performance at each level.

In other words, from the discussion that we had, we were talking about 1 CEO, 2 or 3 people at the divisional level and then 12 or 15 vice-presidents etcetera and a few hundred-thousand programmers and this is what allows the organization as a whole to achieve the required performance.

So, at each level, it was an important to have the right quantity as well as the right quality. The training, the background, the education that the CEO has is conceivably quite different from the training, the education and the experience that one of the software developers has, but it is important to have the right quality human resource in the right quantity at each of the levels of the hierarchy. So, that then seems to be the moral of the story from the perspective of successful hierarchy in a business organization and that was the business organization.

If we look back at our memory hierarchy perspective of the same diagram, we get rid of the CEO's and the programmers; we go back to speed, size and access time. The question is what can we understand from memory hierarchy and the answer is clearly we need to talk about hierarchy, in the sense of having enough or a fast enough resource in order to make the execution time of programs or the well-being of the computer system as a whole or the efficiency of the organization as high as possible.

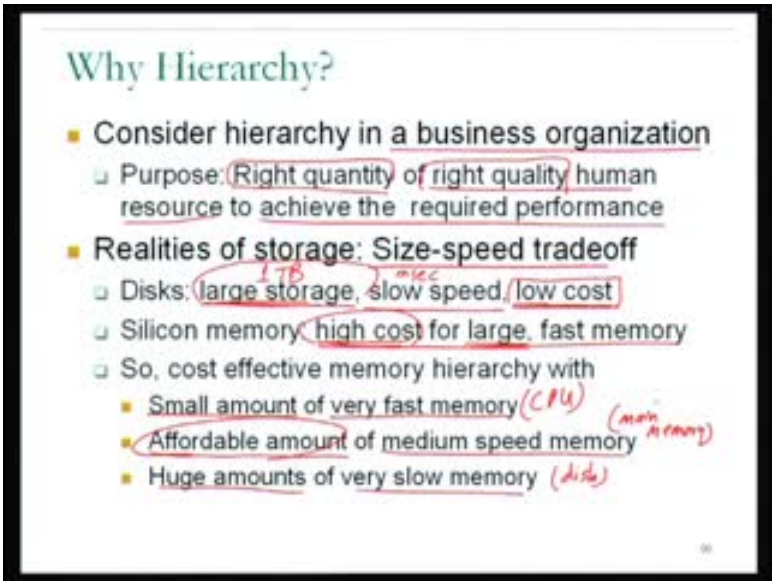
So, there is a mapping between the concepts for this perspective. So, if the purpose of the business hierarchy was to ensure that there were the right quantity of the right quality human resource to achieve the required performance, then bearing in mind the realities of memory, in terms of the size-speed tradeoff that I alluded to earlier and let me just elaborate on the size-speed tradeoff, a little bit.

We talked about how disks can be extremely large. For example, we could conceivably have one terabyte disks today, but they are very slow - several milliseconds to access them. One thing which I did not talk about was the fact that they could actually be extremely cheap. I had referred to this in passing when we talked about storage in one of the earlier lectures. But for those of you who have bought computer systems, you will realize that it did not cost a whole lot to upgrade your computer system from 80

gigabytes of hard disk to 500 gigabytes of hard disk. In terms of rupees, it may just cost you a few 1000 rupees to upgrade to from 80 gigabytes of hard disk to let us say, 300 gigabytes of hard disk.

Therefore, in general the disks are low in cost and I am using the word low, if I was to compare the cost of upgrading disk from 80 gigabytes to 500 gigabytes with the cost of upgrading main memory from half a gigabyte to 4 gigabytes. But the cost of upgrading main memory from half a gigabyte to 4 gigabytes would be substantially more than that of upgrading disk from 80 gigabyte to 500 gigabytes. So, in general, then we can talk about the silicon memory as having a higher cost, in order to be large and fast. Now, the disks in the packaging are fairly large. One can buy a single disk today which is a 1 terabyte disk, but in terms of memory, one could think about having a huge amount or large amount of main memory.

(Refer Slide Time: 08:30)



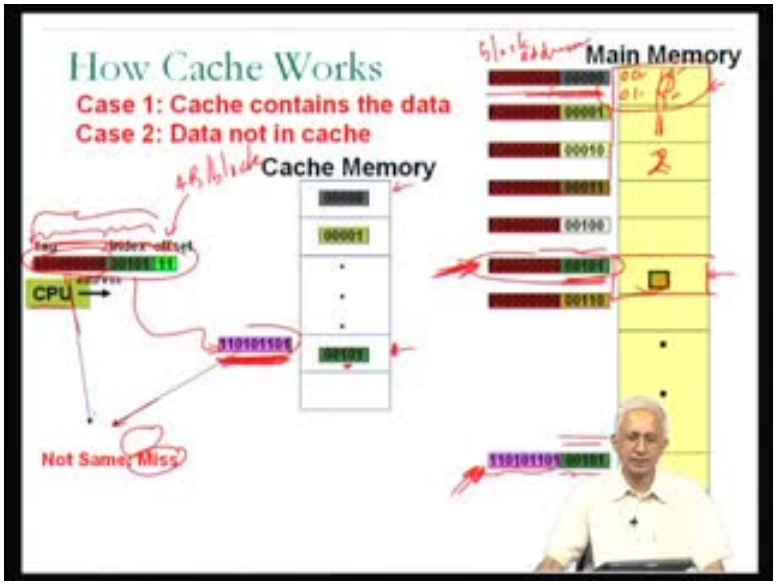
But unfortunately, that may not be possible because of the cost problem. It may not be possible to afford a large amount of main memory. Therefore, one talks about the cost. One must bring the cost into picture. Otherwise, the terms about large, relative size will not make sense.

We are here talking about on a given budget, how much of that kind of memory might be affordable. So, in general, one can talk in comparing disks with circuit memories such as

those used in registers the cache or main memory. One would use the silicon memory as being higher in cost, in order to have larger, faster memory and in that sense, one would say that a cost effective memory hierarchy should have certain small amounts of very fast memory because the processor requires that there should be some very fast memory. So, this is from the perspective of the CPU - very fast memory is possible, but from the perspective of budgetary constraints, it might only be able to have a small amount of such very fast memory.

At the other extreme, given the budgetary constraints, it should be possible to have huge amounts of very slow memory such as disk. One can talk about terabytes of disk and still be able to afford it and in the middle, whatever one can afford. In other words, affordable amounts of the medium speed memory, which is the main memory type of scenario that in our current hierarchy.

(Refer Slide Time: 13:45)



So, in this discussion about hierarchy, this is what we are talking about. We are arriving at some understanding of why cache memory is small, why main memory is only up to about 4 gigabytes and how come there can be such huge amounts of disk. Now, with this better understanding of what it means to be a hierarchy, let us look at how the cache works in a functional sense using a diagram. So, we will try to run through an example of cache in operation.

So, the setting in which we will do this is that the CPU is generating addresses as always and ultimately, the address is of a memory location. We will get into this a little bit more, but we are assuming that it is a virtual address.

The virtual address will get translated into a physical address and the physical address is a memory the address of a memory location either one memory location or four contiguous memory locations depending on the size of the entity being accessed. Now, let us suppose that for a particular address that the processor is just generated. This is the element in main memory, the brown blob is the element in main memory that the processor actually requires.

Now, if it is to access this particular entity directly out of main memory, it will take 100 nanoseconds or more and therefore, we have this cache memory intermediate. The purpose of the cache memory is that most of the time, if you are lucky, it will provide the piece of data or the instruction whatever this happens to be directly without the need for main memory being involved. So, we understood from our discussion in the previous lecture, that both the cache and the main memory can be viewed as being organized in terms of blocks, not in terms of being byte addressable.

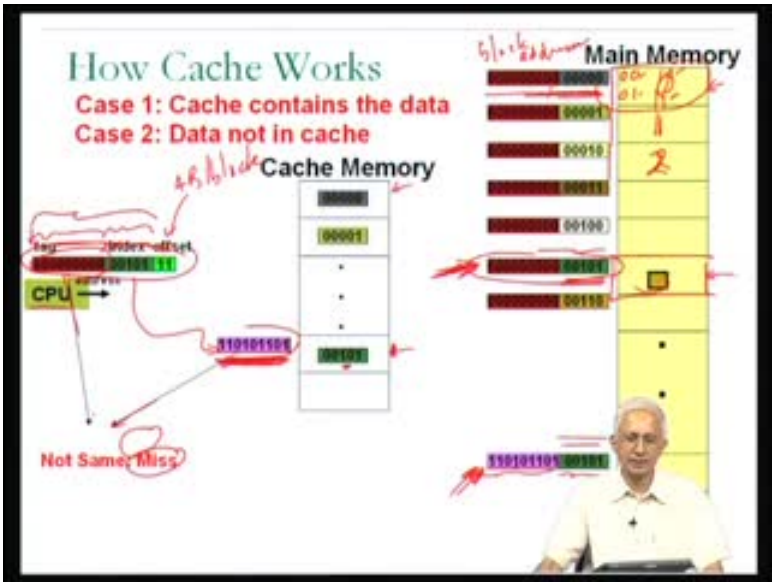
We view them as both being block addressable and the block is some kind of a design parameter of a cache memory. So, rather than talking about individual bytes, this brown entity over here could have been 1 byte of data and it might be present in this particular block and is one of the bytes within that block.

So, I show the block as being the larger entity. This is the block containing that particular byte. So, both cache and main memory are organized in terms of blocks and we look at the first possible situation in connection with the CPU generating an address. So, the CPU generated an address and sent it into the memory system and it essentially wants to access this brown piece of data. Let us assume that as a piece of data.

Suppose that this piece of data is currently present in the cache and it is present in the cache in this particular cache block. Remember, I am talking about both cache and main memory as being organized in terms of blocks I can therefore, refer to a cache block; any one of these boxes is a cache block and any of these boxes, I will refer to as a main memory block.

So, the current situation is what I call case 1. In case 1, the required piece of data is present in the cache. Now, to understand how the cache works as we figured out in the previous lecture, we have to look at the address not as an abstract entity, but as a bit pattern.

(Refer Slide Time: 13:45)



So, let us look at the address as a bit pattern and the bit pattern is as shown in that diagram above the address. The font may be a little bit small, but you will notice that as far as the address is concerned, I am showing you the address from the perspective of the cache. Remember that from the perspective of the cache, the cache hardware views an address as being made up of 3 parts. There are the intermediate bits, which are what we call the index bits, there are the least significant bits, which are what we call the block offset bits and there are the most significant bits, which are what we call the tag bits.

So, I show the tag bits in red, the index bits in dark green and the block offset bits in light green. So, the address that has just been generated by the processor is as shown in the diagram on the left. The tag bits are all zeros, the dark green bits are 00101 and the block offset bits are 11.

Now, when the cache does its look up, it will use the cache index bits to identify which of So, the terms are that we talk about the tag bits, the index bits and the offset bits and we can now think about these addresses in the light of main memory addresses. Now,

you will recall that main memory itself is accessed in terms of or is viewed as being organized in terms of blocks. Therefore, associated with any one of these main memory blocks, I can have a block address.

For example, the block address of that first block in main memory is all zeros because this is the main memory block 0. Similarly, the block address of second block in main memory is 1 and if you look at the address bits, you will see that they are all zeros with a 1 at the end; similarly, 2 and so on. So, the main memory blocks each has a block address. So, these addresses which I have put over here are block addresses. Just as we talked about page numbers When we were talking about virtual memory, we looked at each main memory physical page as a physical page number. In this context, we look at each main memory block has having a block address and the block addresses go from 0 up to some maximum possible block address.

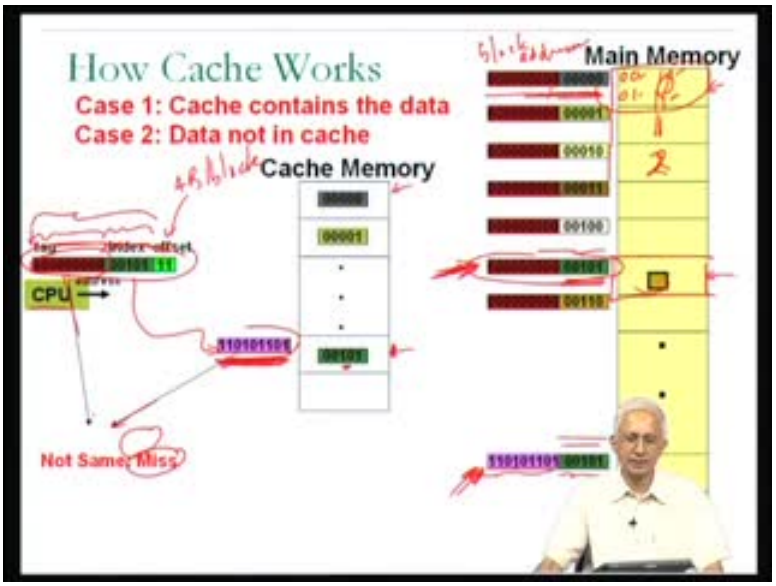
So, if you look at the block addresses now, you will notice that I have used a similar kind of a colour coding scheme to identify the different block addresses within main memory. I seem to be using a notation, where just as I coloured the most significant bits of the main of the address itself for use as a tag and the intermediate bits for use as index, I have similarly, split the main memory at block addresses into two fields, one of which is the same number of bits you notice here that there are some number of bits which are the same as a number of bits in the tag field and over here, there are some number of bits which are the same as a number of bits in the index field.

So, in numbering each of the main memory blocks, we can view that main memory block as having some most significant bits and some lesser significant bits. Within the main memory block, there are many bytes each of which will have some byte offset.

In this particular example, it looks like the size of each block is 4 bytes. That is why the size of the byte offset field is 2 bits. So, there are 4 bytes per block; that is what we can assume and therefore, the 4 bytes within the main memory block 0 would have block offset bits of 00, 01, 10 and 11. Now, from this perspective, if I look at the main memory block addresses, the first main memory block address is 0, the second is 1, third is 2 and so on and if I look at the sub breakup of the bits of the main memory block addresses as done per the colour coding scheme that you see here, all of the early main memory block addresses have 0 as most significant bits and some of the later main memory blocks such

as the one with a pink and the green has nonzero in its most significant part and some other bits in its lesser significant part.

(Refer Slide Time: 13:45)



Now, at any given point in time, some number of main memory blocks are present in the cache directory and let us assume that I am showing you three of the main memory blocks. For example, I am showing you the one if you consider the cache block, which contains the piece of data that we are interested in, then it is a cache block which has a main memory block address of all zeros, which is the red field, followed by 00101 which is remembered in the cache memory, in terms of 00101 as being the index value; in other words, which of the cache entries we were talking about as far as this main memory block and the tag bits. Essentially saying that among all the main memory blocks which have green least significant bits in their block address and there are at least two that we can see, the one with all red before the 00101 and the one with all pink before the 0010.1

At this particular point in time, the one which is inside the cache memory is the red one and that is indicated by having inside the cache directory, the red bits. In other words, 000 all zeros as the tag field. Now, when the time comes for the address when the address comes from the CPU to the cache memory, remember that the cache memory views the address a tag index and offset and it uses the index bits to identify which particular block in the cache could contain the data that is being requested.

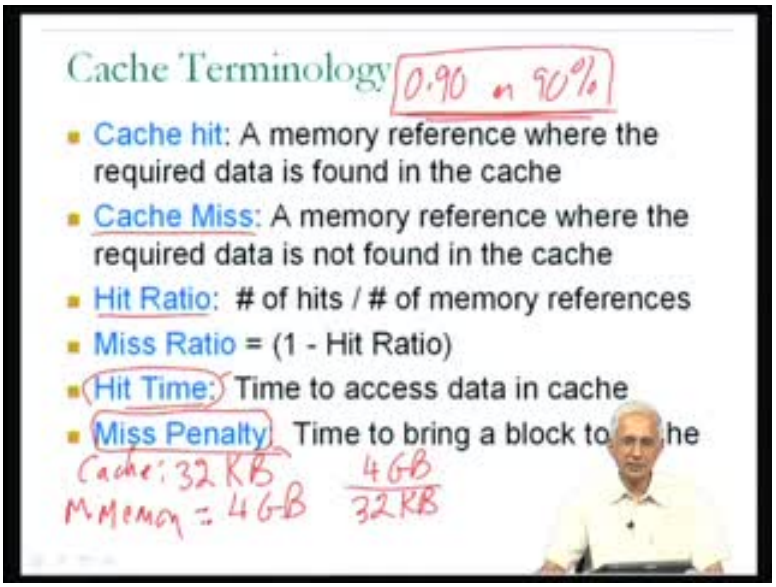
How does it determine whether the red or the pink main memory blocks is currently present in that particular cache entry? The answer it does this by comparing the tag bits from the address with the tag bits which are kept in the cache directory. So, it does this comparison and if they are the same then it is known that there has been a hit. In other words, the piece of data which the processor has requested is in fact present; it was present in the background; it is not covered up - the brown box and can therefore, be provided directly to the processor and that was case 1.

Now, the alternative is case 2. In case 2, the assumption is that the data is not in the cache. In other words, that brown entity is actually not present in this particular cache block suggesting that in that particular cache block, what we have is not the red main memory block, but possibly the pink main memory block - one of the other main memory blocks, which could have been present in that particular cache block.

Now, when the comparison is done of the tag bits from the address with the tag bits from the cache directory, it is found that they are not the same. In other words, this is not a situation where the piece of data that the processor is requesting is present in the cache directory. This is essentially how the cache hardware views the address and uses the bits of the address along with the information which is present in the cache directory to locate and check the validity of the data present inside the cache memory.

So, this is just a rough quick introduction. We now look at the details by looking at alternative design ideas which are used in practice in cache memories. Now, some of the terminology which I have used: I, in fact, used two pieces of terminology. I talked about a hit and a little later, I talked about a miss without telling you what these technical terms actually mean.

(Refer Slide Time: 24:45)



So, let me just run through some of the terminology now that we have this rough understanding of how the cache memory operates. Now, the first and most important concept in terms of successful operation of the cache is the idea of a cache hit and you will recall this is what I describe the case 1, from the example that we just went through. So, cache hit is the situation where the processor sends an address to the cache and it is determined that the required data is present in the cache.

In other words, this is a situation where the CPU requests something from memory and the required data or instruction is found in the cache. This is referred to as a cache hit. The alternative, which was case 2, for example, is what is known as a cache miss. It is a situation where the required data or instruction is not found in the cache and very clearly, it should be the objective of cache design to try to ensure that the number of cache hits is more than the number of cache misses.

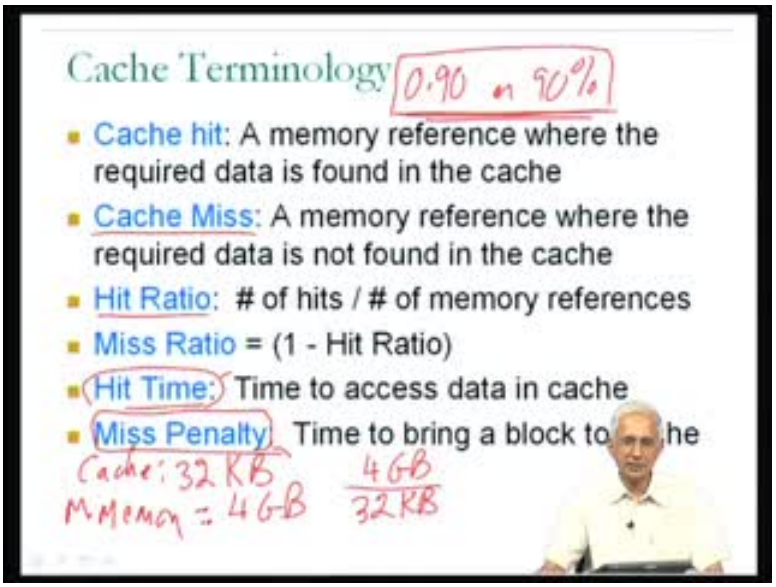
Now, we would therefore, try to talk about We could quantify how successful a cache is in satisfying the requirements of a program in terms of memory references by a measure called the hit ratio and the hit ratio is computed as the number of cache hits divided by the total number of memory references. So, for example, if a processor makes 100 memory references and 90 of them are actually hits inside the cache, then I would talk about this as being a situation with a hit ratio of 0.9, which we may also be described as 90 percent. We could alternatively talk about the miss ratio which will be 1 minus the hit

ratio. So, in our particular example I would have a miss ratio of 0.1. So, we would be happy, if our programs observe or benefit from a very high hit ratio and we would be unhappy, if the miss ratio is high.

Another property of a cache will be the hit time, the amount of time that it takes to access data in a cache and our hope is that this is less than the 1 nanosecond that we are talking about. Ideally, we would want the 1 nanosecond to be the amount of time for the cache look up as well as the actual time to access the cache RAM and all of these things put together is what we talk of as the hit time.

Finally, we could also talk about the miss penalty. Recall that the cache miss is the situation where the required piece that the piece of data, which has been requested by the processor is not found in the cache. Now, what does the cache hardware have to do, if there is a cache miss? Very clearly, the cache hardware must cause the data to be brought from main memory into the cache and subsequently, the data can be provided from the cache to the processor.

(Refer Slide Time: 24:45)



There is therefore, a time penalty associated with every miss and the time penalty is basically, the amount of time that it takes to bring a block into the cache and this may involve a memory access and therefore, for the kind of cache situation that we are talking

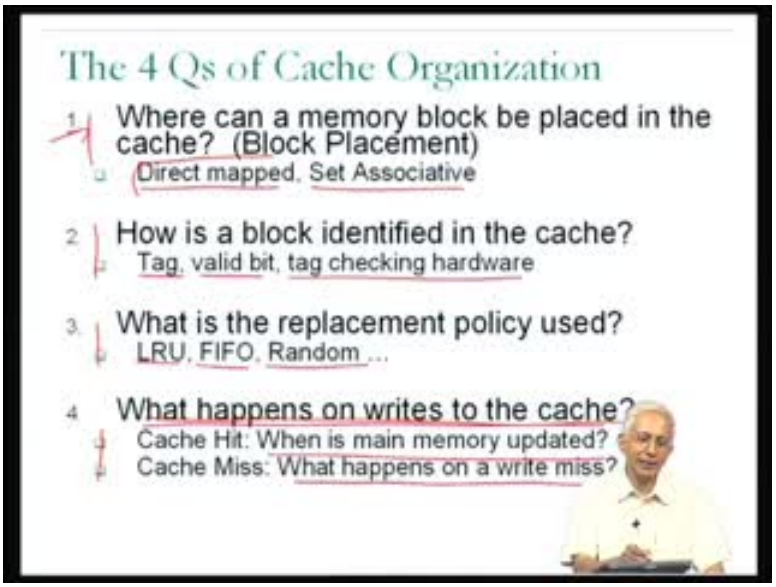
about right now, we suspect that the miss penalty could be as much as the memory access time which might be 100 nanoseconds or thereabouts.

So, these are some of the terms, which are used and let me just again remind you that we would like to have a high hit ratio for our programs and as far as the design of the processor is concerned, as far as the design of the computer system is concerned, we would hope that the miss penalty has been made as low as possible and the hit time has also been made as low as possible so that even every time there is a cache hit, the data comes in a very small amount of time and even in situations, where there is a cache miss, the amount of time penalty suffered should not be too high.

Now, in talking about the performance of programs, I have used numbers like hit ratio of 0.9, which I described as 90 percent as being apparently an achievable target. So, suggesting that for the kinds of cache designs that are common today, it is actually possible for programs to see cache hit ratios as high as 90 percent. In other words, 9 out of 10 or even higher, but this is not very typical; 9 out of 10 times, when the processor tries to access something out of memory the data or instruction is available out of the cache and therefore, only the remaining 10 percent of the accesses that have to be accessed at memory access speeds.

Now, this is an amazing number, given the size disparity between the cache and the main memory. Remember, the typical size of cache that I mentioned was something like 32 kilobytes and the typical size of main memory that I talked about was something like 4 gigabytes. Therefore, difference in size between the two is 4 gigabytes divided by 32 kilobytes and as you recall this is 10’s of 1000’s, 100’s of 1000’s; this is a very large number. Therefore, given that the cache is so much smaller than the main memory, how is it possible for the cache to contain 9 times out of 10, the piece of data or instruction that the processor is requesting? This is in fact, quite difficult to accept - just out of hand, but let us proceed to look at some of the cache design properties through which this is achieved.

(Refer Slide Time: 30:13)



Now, when people talk about the organization of caches, they typically stress four design parameters of caches and I will present or talk about those four design parameters of caches in terms of 4 questions that are addressed by settings of design parameters.

So I will talk about the 4 Qs - the 4 questions of cache organization and the 4 questions, which I will talk about, are: first of all, among all the different cache locations - cache blocks for a given main memory block, where could it be placed inside the cache? We have seen that the main memory is many times the size of cache – 1000's, 10's of 1000's 100's of 1000's times the size of cache.

Therefore, very clearly for each cache block, there are going to be 10's of 1000's of main memory blocks, which will be potentially entering the same cache block, but the underlying issue is for a given cache, how is the relationship between the main memory blocks and a given cache block specified and the problem is known as the block placement or the question is known as the block placement question.

The second question is how is the block identified in the cache? So, among all the main memory blocks, which could be present in the cache, at any given point in time, some small subset are present in the cache and how does the cache controller or how does the cache hardware keep track of which main memory blocks are currently present inside the cache.

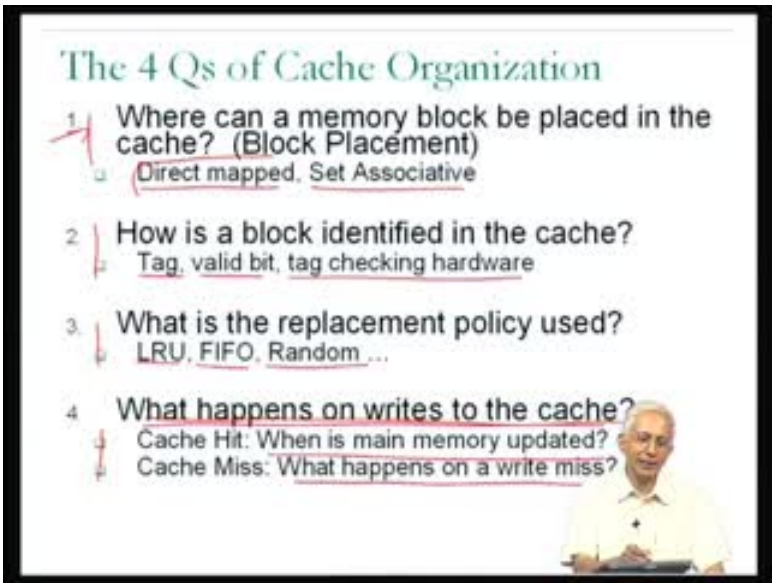
The third question, very clearly, an important question. What is the replacement policy used? Evidently, we know about replacement policies from our discussion of virtual memory. We realize that in the case of the relationship between cache and main memory, replacement is a much more important consideration because of the enormous size disparity between the cache and the main memory and we suspect that there may frequently be situations, where there is a cache miss and a block has to be replaced from the cache, in order to make place for the block, which has to be fetched from main memory into the cache, in order to satisfy the processor's address request.

Therefore, the choice of replacement policy is clearly going to be important for the successful operation of the cache and finally, a new question, the kind of question we do not see in connection with virtual memory. What happens on writes to the cache?

Now, let me just give you some of the answers to these questions and in the case of the last question, there are two sub questions, but in the case of the block placement question we will look at two answers: one is known as direct mapping, the other is known as set associative mapping. These are technical terms, which you may find used in descriptions of the operation of caches that you were working with and therefore, they are important terms for us to understand.

In terms of how a block is identified in the cache, we will hear about some of the mechanisms that are used inside the cache directory, in order to check whether a particular block is in present and in fact, some of the fields of the cache directory entry.

(Refer Slide Time: 30:13)



In discussing the possible replacement policies used, we will some of our familiar replacement policies from our discussion of virtual memory. LRU, FIFO and random - all conceivable feasible policies for use inside caches as we will discuss and finally, when it comes to the issue of what happens on writes to the cache, this is a new question which we did look at one aspect of when we talked about virtual memory, but there are basically two sub questions which arise. I will refer to them as sub question a and sub question b.

The first sub question is what happens on the write to the memory, if the write is in fact a cache hit. In other words, the processor executes a store instruction, it wants to modify a memory location and it turns out that the block containing that memory location is present in the cache. Therefore the question arises what happens on that write and the underlying question is, is main memory updated along with the cache update or is main memory updated only once later on and we showed an allied issue when we talked about virtual memory in terms of what happens, when a write happens.

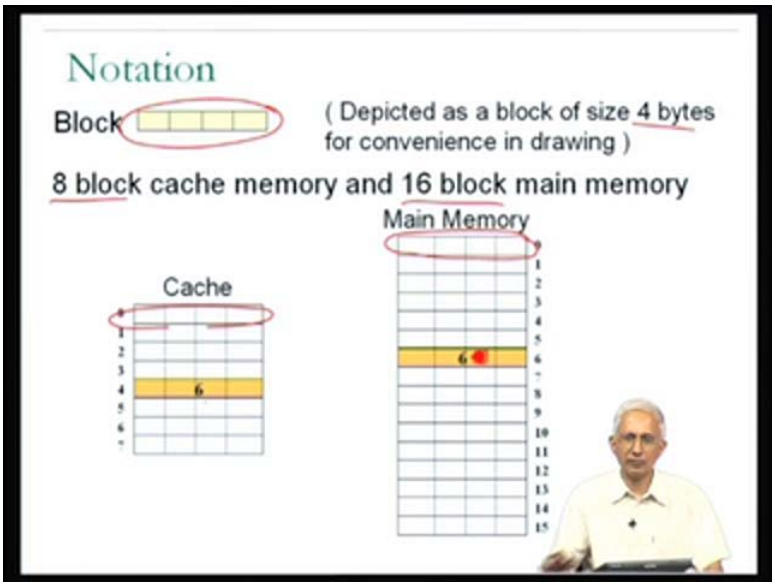
But there is a new question which could arise in the case of cache memories, in the event of a cache miss. Remember what I am talking about here is, writes on a cache miss. So, the idea is the processor executes a store instruction. In other words, it wants to modify a memory location, but it is determined by the cache hardware that particular memory

location is not present in the cache and therefore, the cache hardware could be built in a few different ways in terms of how to deal with that particular miss.

Now, in thinking about these four policies, it is not a bad idea to look back and try to understand how each of these four questions was answered in the connection of virtual memory because we now see that the relationship between main memory and cache is very similar to the relationship between secondary storage and main memory that was implemented through virtual memory. Therefore, these same issues must have a reason when we talked about virtual memory, but maybe, we did not look at in this light and therefore, when we look at any one of these issues in more detail, we may just look back and see how the same issue was handled in the case of virtual memory.

With this, we will proceed and start to look at the first question - the block placement question. Among all the different cache locations, how does the cache hardware decide where a particular main memory block could be present in the cache and we start by looking at the option called direct mapping.

(Refer Slide Time: 35:58)



Now, in discussing the caches, I am going to use diagrammatic examples and just to make sure that the diagrams are more useful to you, I will use the consistent notation. Now, in this notation, whenever I want to refer to a block, I will use a box that looks something like this.

Now, the reason that I use a box which looks something like this is to indicate that we are talking about some number of contiguous locations, some number of bytes and they could span more than a word, but we need to have some idea that there are some number of bytes associated with each block.

In this particular notation, the suggestion is that the size of a block is 4 bytes, but that is just for convenience in drawing. I am not suggesting that four is a typical block size in real caches. We will talk about block size a little bit later, but for the moment, I am just using this notation so that I can draw these diagrams a little bit quicker.

Now, we will use this kind of a box with 4 components in it to represent a block and that is just for ease of drawing. Now, in drawing these diagrams, I will have to show you both cache and main memory and therefore, I will have to show some number of blocks in cache and some number of blocks in main memory and we know that the typical size of a cache is something like 32 kilobytes and the typical size of a main memory is something like 4 gigabytes.

But I cannot draw diagrams that large. So, once again that I am just going to show very small caches which will be 8 blocks in size and in fact extremely small main memories which are only 16 blocks in size and you will notice that this is a ridiculously small size for main memory since that would amount to a 64 byte main memory, but for the purpose of understanding how these mechanisms work, this is adequate. I am showing you a cache which is smaller than the main memory size and which is the at least consistent with our understanding of the relative sizes though grossly disproportionate. We would expect the cache to be 1000's of times smaller than main memory.

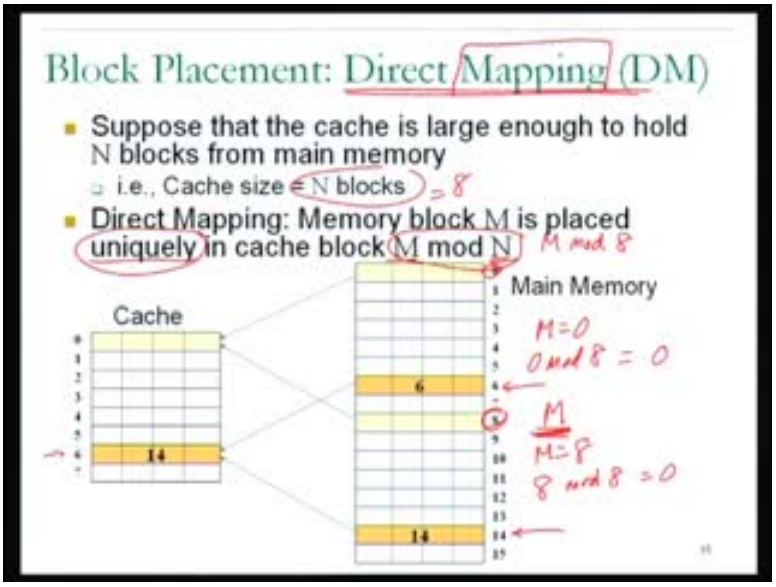
So, I will show a cache as something like this. You notice that this cache has several blocks. Each of the horizontal entities is 1 block and therefore, in this diagram I am showing a cache with 8 blocks. In the diagram on the right, I am showing you a main memory; once again, each of the horizontal entities is 1 block.

I am showing you a main memory which has 16 blocks. Now, since we are going to be looking at addresses generated by the processor and we have to look at the addresses of blocks, I will number each of the cache blocks and each of the main memory blocks with its block number.

We know that the main memory blocks start from 0 and go up to some maximum block number. So, the main memory blocks are 0 to 15. I will number the cache blocks from 0 to 7 in this example because there are 8 cache blocks and so, this is a typical setting - the typical kind of notation we will use in describing the different possible cache organizations.

For example, if at a particular point in time, I am interested in main memory block 6. This might be the main memory block that contains the brown piece of data that the processor was interested in. If I want to represent the fact that main memory block 6 is currently present in cache block 4, I might show it using this kind of a notation. Both of them are of the same color and I, in fact, labeled the cache block with the main memory block address, just for ease of understanding.

(Refer Slide Time: 39:45)



Now, let us start off by looking at one of the possible ways that a cache could be designed to answer question 1. Question 1 was where can a memory block be placed in the cache and this is called the block placement question. Now, we are going to use the example of cache with 8 blocks and main memory with 16 blocks, but we need to talk in more general terms. So, let us talk about more general terms, where the size of the cache is N blocks.

In other words, the cache RAM is big enough to hold N blocks from main memory. In this particular diagram, N is equal to 8, but we will talk in the more general sense of N. Now, the block placement question is, where could a particular main memory block be present in the cache. Let us suppose that I am currently concerned about some main memory block M. M could be 3 M, could be 4 M, could be 15 etcetera.

I am concerned about main memory block number M. Now, under direct mapping main memory block M is placed uniquely in the cache block M mod N, where N is the number of cache blocks. So, we notice that this is a situation where a particular main memory block is going to be present. If it is present in the cache, it can be present only in one particular cache block and the particular cache block is uniquely specified by this expression, where M is the main memory block number and N is the size of the cache in terms of blocks. Now, in our particular example, we are using the size of the cache is 8 and therefore, when I talk about M mod N, I am talking about M mod 8. You will recall that what I mean by the mod operation is the reminder from the integer division of M by 8.

So, let us consider main memory block 0. So, M is equal to 0. If I calculate 0 mod 8, I get 0. So, I understand that if main memory block is to be present in the cache and I am referring to a direct mapped cache or a cache which uses direct mapped block placement, then main memory block 0 could be present only, hence the word uniquely, in cache block 0. Now, that is easy to understand, but we should also realize that main memory block 0 is not the only main memory block that could be present in cache block 0, even for this very small example.

For example, if I consider M equal to 8 - main memory block 8 then 8 mod 8 is equal to 0; the reminder when you divide 8 by 8 is 0 and therefore, main memory block 8 also is going to map uniquely in cache block 0, which means that if main memory block 8 is present in the cache, it could be present only in cache block 0. Hence, obviously, at any given point in time, either main memory block 0 or main memory block 8 could be present in the cache and not both, if I was talking about a direct map cache.

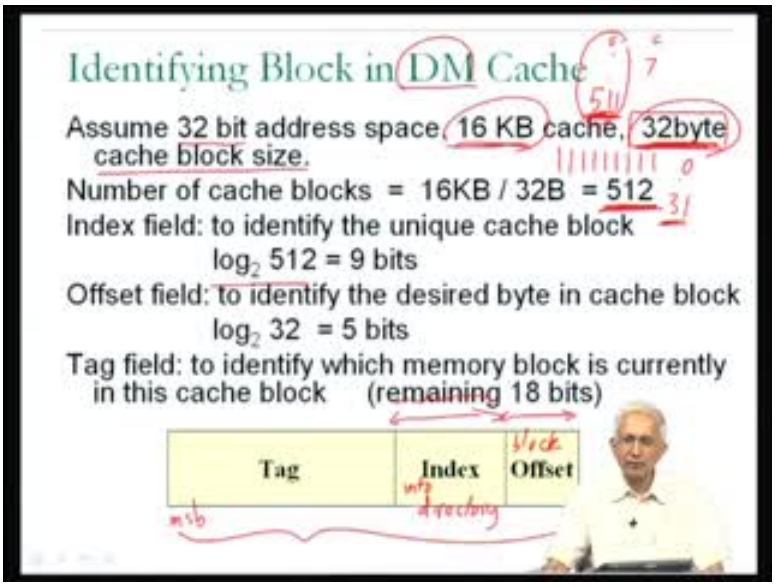
Now, just another simple example, it is quite easy to see that if I am concerned about what are the different cache, a main memory blocks which could map into cache block 6, then it will easy for you to see that both main memory block 6 and main memory block

14 have the property that M mod 8 is equal to 6 for them. Because 6 mod 6 is equal to 6 and 14 mod 6 is equal to 6. Therefore, we know that between main memory block 6 and 14, only one could be present in cache at a time.

So, it could be that right now, the main memory block 6 is present in the cache in which case, the cache will look like what we have on the screen, whereas after this, if main memory block 14 enters the cache, then the cache would look like this because both main memory block 6 and main memory block 14 mapped to cache block 6.

So, I will use the word map as a verb and we talk about direct mapping. The relationship between the main memory block address and the cache block address is given by main memory block address mod cache size. So, that was easy to understand and you will realize that this was implicitly what I was assuming in that quick, but detailed example that we ran through at the beginning of this lecture. As a consequence, you now understand that this is not the only possibility for block placement and that we are going to learn about other possibilities which were not taken into account by that quick example that we ran through at the beginning of the lecture.

(Refer Slide Time: 44:42)



Now, the second question which we have to understand little bit about is how does the cache hardware identify, whether a particular block is present in the cache or not. We understand now that given the main memory block address under direct mapping, I have

abbreviated direct mapping as DM, we understand how the cache hardware can identify which unique cache block could be the identity the cache block, where the main memory block is present. But what are the mechanics behind actually figuring out between 6 and 14 for example, which one is currently present in the cache. Now, to actually understand this, once again we will use some specific examples.

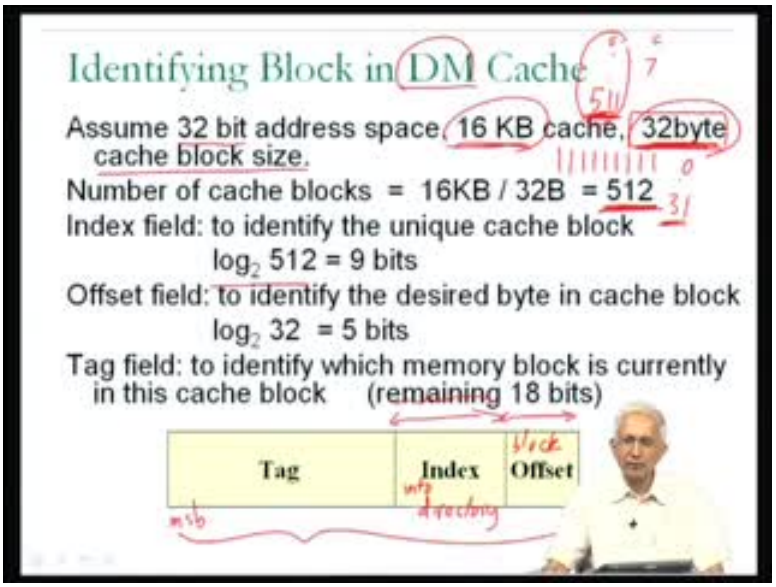
But to work with the specific examples, we will actually use realistic numbers for the first time in our discussion of caches. I am going to assume that the size of an address is 32 bits. I will assume that the size of the cache is 16 kilobytes. So, I am giving you the size of the cache not in terms of number of blocks, but in terms of number of bytes and finally, I will assume that the size of a block is 32 bytes and 32 bytes is a fairly typical block size.

In the diagram that we had used in our notation, we are assuming a block size of 4, which is actually much smaller than what is realistic and you will understand that the size of a block will have something to do with the typical spatial locality of reference that one should expect programs to show.

We now understand that it is more in the order of 32 bytes than the 4 bytes from our drawing - the pictorial notation. Now, that the first question that we have to understand is how do these numbers help us in understanding how the cache operates and we have to go back to our discussion about how the cache hardware views an address.

You will recall that the cache hardware views an address as being made up of three fields. If you go from the least significant bit to the most significant bit, it views least significant bits as being the block offset. It uses those intermediate bits as index into the cache directory and the other bits are called the tag bits and we saw quickly in our example earlier in the lecture, how they are used, but we can now think a little bit about how the cache hardware actually accesses these different bits now that we have some numbers to play around with.

(Refer Slide Time: 44:42)



Now, the first question we could tackle is for example, I mean we need to understand how many bits are used for the block offset and how many bits are used for the index and apparently, these can be computed using the facts that are the upper part of the screen. Now, the first thing that we can readily compute is the number of cache blocks; in other words, the size of the cache in terms of cache blocks. We know the size of the cache in terms of bytes, we know the size of a block in terms of bytes, we can therefore, calculate the number of cache blocks by dividing 16 kilobytes by 32 bytes and you will quickly be able to see that this is equal to 512. What does this mean? This means that there are 512 cache blocks.

This cache block 0 up to cache block 512 and the diagram that we had used, remember, we have talked about cache block 0 up to cache block 7 because we were talking about a cache of size 8 blocks. We now have a cache of size 512 blocks, which means that the cache blocks are numbered from 0 through 511. From this, we can immediately calculate the number of bits that are required to index into the cache directory. If the cache blocks are numbered from 0 through 511, then we know that they must be 512 entries in the cache directory and therefore, to index into the cache directory we need as many number of bits as necessary to distinguish between values that could be between 0 and 512 and that is going to be computable as the log base 2 of the number of blocks.

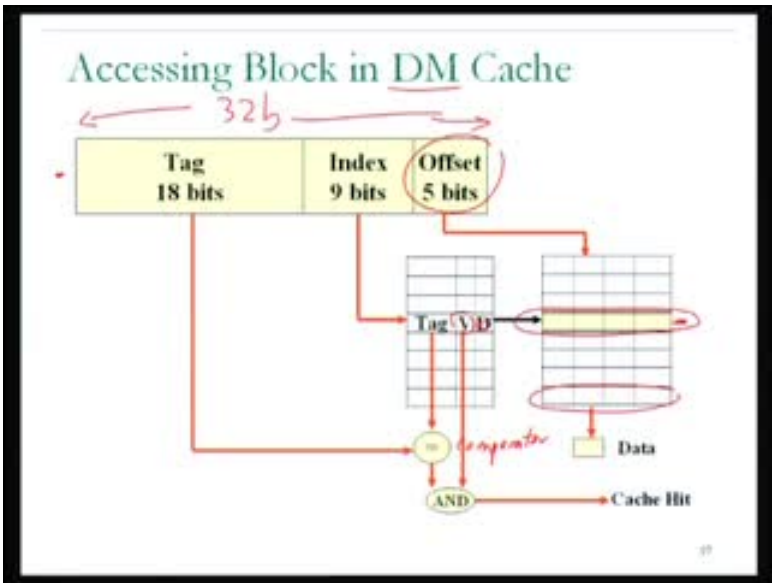
So, log base 2 of 512 is 9 bits, which tells us that the number of bits that are needed for the index are 9 bits. Just to make sure we understand this. If I think about the fact that there are 512 blocks and the blocks are numbered between 0 and 511 therefore, the blocks are numbered between 0 and 511. Just ask yourself the question in binary, how many bits does it take to represent 511 and if you run through the binary representation of 511, you will realize that it is 256 plus 128 plus 64 plus 32 plus 16 plus 8 plus 4 plus 2 plus 1.

You add up those numbers and you see, that is equal to 511 and if you count the number of bits, you will realize that there are 9 bits. This is a just quick way of doing the same calculation. So, we realize now that the number of bits, which will be needed for indexing into the cache, is 9 bits and this could be calculated from the facts that we had been provided with. The next question, how many bits are going to be used for the block offset and again, we are trying to understand how the address is used by the cache. So, we do need to know how many bits are used as a block offset.

Now, once again, we know that if the size of the cache block is 32 bytes that means that the different bytes within the block would be numbered from 0 through 31. There is byte 0 Within any particular block, there would be byte 0 through byte 31 and therefore, to identify any particular byte within the block, I need to use as many bits as are needed to represent 31 in this sense and you will quickly be able to figure out that the number of bits is going to be given by log base 2 31, which is 5 bits.

So, from just these two pieces of information and the fact that the cache is direct mapped, we are able to view the address in terms of a 9 bit offset and a 9 bit index and a 5 bit offset, which leaves, given that the size of the address is 32 bits, 18 bits for the tag. So, we end up with the remaining 18 bits are being used for the tag. Now, the next question which will immediately arise is how is this used by the cache hardware and so, we will pictorially understand how this works.

(Refer Slide Time: 51:20)



So, here for the first time, I am adding some notation. I have not included part of this. To the right, you will recognize our cache. This is a cache, which has 8 blocks. Each of the horizontal things is one cache block. So, that is very clearly a cache RAM, but in addition to the cache RAM, remember that the cache contains a cache directory and what I am showing you to the left is the cache directory. So, the cache directory contains one entry corresponding to each block in the cache and therefore, the cache directory in this case contains 8 entries.

So, each entry in the cache directory has a corresponding entry in the cache RAM. Now, within each cache directory entry, I am showing you several fields. For the moment, we will try to understand We understand that there may be a need to have a valid field form our discussion of virtual memory and this may largely arise out of the need to have a meaningful interpretation of the state of a cache. For example, when a machine starts executing, before any data has been fetched into the cache, the cache hardware may have all zeros and therefore, it is important to have an explicit bit which can be set to 1, when meaningful data is brought into the cache. Hence, there is a need for a valid bit which as in the case of the page table, I will label as V. We will come to the dirty bit later.

But you already know what that means from your exposure to virtual memory. However, many of the fields inside each cache table entry or labeled as the tag field. Now, the way that the accessing of the block will happen in the direct map caches, you will remember

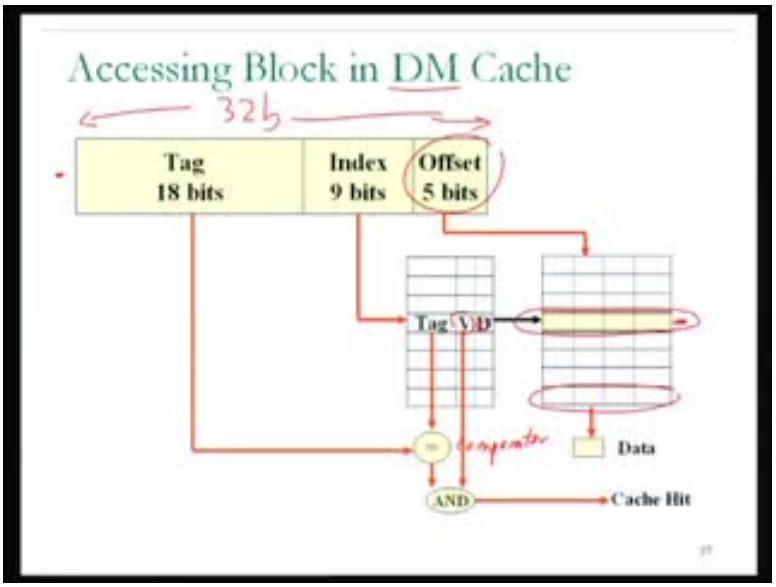
that in the previous slide, we have worked out that the 32 bit address is viewed by the cache hardware as the least significant 5 bits are the block offset, intermediate 9 bits are to be used to index into the cache directory and the most significant 18 bits are what are called a tag.

So, what the cache hardware will do is it will use the intermediate 9 bits to index into the cache directory. So, if the value of those 9 bits is three, then it indexes into the corresponding element of the cache directory. In other words, since it is a direct mapped cache, it has determined that among all the main memory blocks, which could be present in this particular cache block, this is one of them; that is what this information reveals. Now, in order to determine whether the particular main memory block which is currently present in this cache block is the same as that represented by this address, the tag from the address is compared with a tag, which is present in the directory entry.

So, a piece of hardware called a comparator compares It is capable of determining by the two things are identical to each other or not - 2 bit patterns. It compares these two fields the 18 bit tag field from the cache directory with the 18 bit tag field from the address which the processor has sent to the cache and if they are the same then this is an indication and in addition if the valid bit is set, then this is an indication that this is a cache hit; that the data which is required by the processor is present in the cache.

In other words, the data that is present in the processor is present somewhere in that particular part of the cache RAM. Which particular piece of data is required by the processor? That is determined by the offset bits. Remember that the offsets bits identify one particular byte or one particular word within the block. So, the cache hardware then uses the offset bits to pull the relevant piece of information out of the cache block and that is what is provided to the processor.

(Refer Slide Time: 51:20)



So, this in a sense gives us a complete picture of what happens from the perspective of the processor sending an address to the direct mapped cache and I will stop here for today, reminding you that we are looking in detail that how cache memory operates. We have understood that cache plays in important part in the overall memory structure of a computer system. In some sense, it plays a part in what is called the memory hierarchy of a computer system. It forms an important part of the memory hierarchy of a computer system which also includes other components such as main memory and disk. We have seen that the operation of the cache can be understood in the light of 4 questions, which I call the 4 Q's of cache organization.

We are currently looking at the first of those question, which is called the block placement question, which basically gives you an idea of how for a particular cache how the cache has been designed to determine for each main memory block where in the cache that main memory block could conceivably be present and we have looked at the first example of a block placement strategy which is called the direct mapped strategy.

We will continue with the discussion of the 4 Q's of cache organization. We will continue from the block placement question and proceed to the other 3 Q's - the remaining 3 Q's of cache organization in the lectures to come. Thank you.