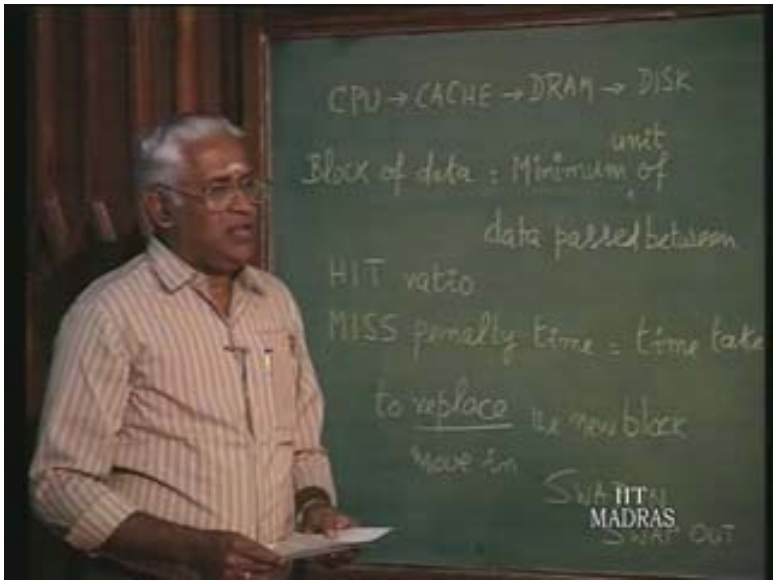
Computer Organization
Part – II
Memory
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture – 17
Cache Organization

We were talking about the different levels in the memory hierarchy, namely, cache DRAM or main and then storage or disk subsystem, for instance. So we said that the CPU generally addresses cache and tries to look for the required data there. If it is not available, there must be some mechanism by which the information is brought from the main DRAM memory. And we also said in the previous lectures that a block of data is the minimum unit of data that is passed between any two levels. Normally we can take it that we always talk about CPU and then two different levels such as CPU and cache and DRAM. We may not go very much in to the other one because there are other issues when you go beyond more than two levels.

Now if CPU addresses the cache and gets the data, we said the CPU is supposed to have a hit. In case it does not get the data, then we said it is just called a miss; that is, it has missed the data; it is not available. So obviously, to see that the processor utilization is at its highest, the hit must be maximum: something like 99% and so on. It means basically that for every 100 times that the CPU accesses the cache, 98 or 99 times, it is able to find the data that it wants. So the higher this particular figure is, generally we talk about a hit ratio. As you can easily imagine, the hit ratio obviously is just the ratio of the number of successful hits, meaning the number of hits that the CPU finds the data in, to the number of total attempts by the CPU. The CPU utilized 100 times to access; that is, CPU is making about 100 attempts and, let us say, if 98 times it is able to hit, then we talk about a hit ratio of 0.98; otherwise the same thing is also expressed as 98%.

Now the problem is, what in case there is a miss? In case there is a miss, then we know for certain that CPU cannot proceed because it does not find the data it wants there in the cache. So there is going to be some delay and this is going to cause a problem. So we say that associated with miss there is always a penalty that is associated; that is, we talk about a miss penalty time. That is exactly the time taken time taken to move in the new block of data that the CPU wants. So a block of data, whatever is passed on between two levels, the CPU does not find the required data in the cache and so that particular block in which the CPU wants that new block must be moved in.

(Refer Slide Time: 07:10 min)

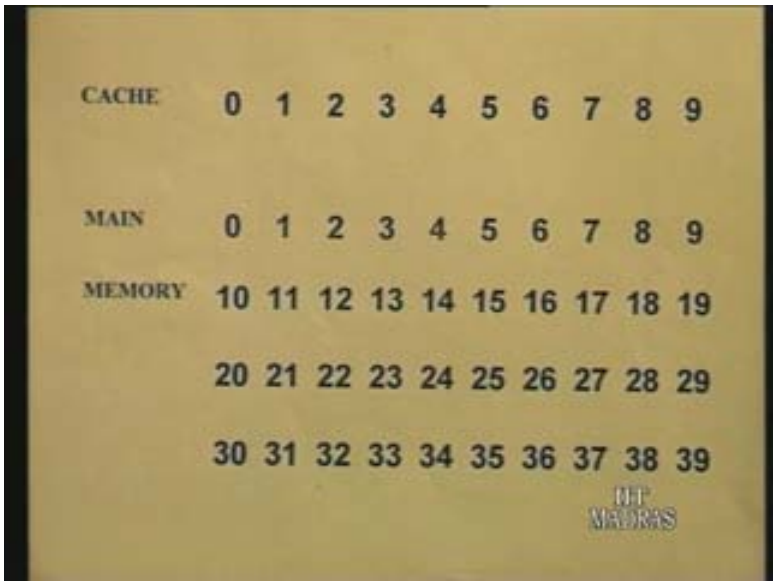


So this miss penalty time is actually the time taken to replace. Why do we say replace? In fact essentially what we mean is to move in; we may add that also. Actually this is what we are interested in, but then replacement automatically comes into the picture to move in the new block that the CPU wants, new block of data that the CPU wants, so that data which is available in this block can be accessed by the CPU. Now in this process, obviously, it may be that the cache has to lose some contents; that is, possibly, the cache will have to lose a block of data and then that will be replaced by the new block of data. The situation may very well be there because we may say that CPU had accessed some data and possibly the data is no more needed by the CPU for subsequent processing, in which case it is meaningful to move out. So normally what we say is we swap out – swap means exchange – the new data is swapped in; you swap in the new data and you swap out the old data. So we talk about swapping in and swapping out. Though these are the terms that are used, swap itself means exchange; you exchange two things that is swap in. So you have to swap out the existing data and create place for the new data, new block of data, to come in, which we say is swapping in the new data. Let us see how exactly this particular thing may be implemented from implementation point of view. Why?

There are many issues related to this particular process of miss penalty – what exactly do we want? We want to see that the miss penalty time is as low as possible, again from process utilization. Earlier we said the hit ratio must be as high as possible; that ensures the process utilization. Similarly, in case there is a miss, this miss penalty time must be as low as possible. So what exactly must be the ratio? That is, suppose we have 1 KB of cache, the DRAM may be, let us say, 4 KB or 16 KB and so on and so forth. So we will try to see the exact way in which this can be implemented. We take an example: we will arbitrarily assume that the cache consists of 10 locations; the cache address is, let us say, 0 through 9. We have just shown as 10 locations; for instance, you can even look at it as 100 locations or 1 K locations or 10 K locations or whatever it is.

Assuming for this particular discussion that cache consists of 10 locations, 0 through 9, and the main memory, that is, the DRAM part consists of 40 locations, that is number 0 through 39, there are 40 locations. Now if the cache arrangement is such that as soon in the figure at any time cache location 0 will consist of data from main memory locations 0, or location 10, or location 20, or location 30. That is a simple arrangement.

(Refer Slide Time: 00:09:51 min)



That is, in this particular location, only the data coming from this or this or this or this or this location alone can go in; that is, it is not arbitrary. Arbitrarily, the content of a memory location can go into anything else; similarly you just take cache location 2 into this main memory location, contents of the location being 2, 12, 22, and 32 alone can go. Now this arrangement in fact is called a direct mapped cache mainly because, as you can see, there is a specific relationship. Given a particular memory location address, immediately we can say what the cache location address is; so this arrangement is called a direct mapped cache, meaning a cache location can have only one of say n specific location contents. Let us just look at the chart again. Suppose the CPU places the address 22. Why do we say address 22? That is because as far as the CPU or the user is concerned, the cache is something in-between, in the sense that the user knows that there are only 40 locations. He is not really bothered about the cache arrangement; it is coming in-between; it has been arranged so that the CPU may interact with a fast memory, which is a cache memory.

Now from user's point of view: that is from programmer's point of view, he will imagine the system to be having 40 locations. He will not really bother about these 10 addresses. So from programming point of view, the user is permitted to use any of these addresses and they are also physically available, though in a slow memory. So as part of the program in which the CPU places instruction after instruction, the CPU will only have to deal with the main memory addresses. It is an internal arrangement that the main address memory will be mapped on to the cache as per some algorithm. Now in this direct map cache arrangement of the algorithm associated with that, there is a specific way and it is the easiest way; you can easily see.

All you have to do is drop the most significant digit in it and look at the cache address straight away. But there is a way to work it out. We will look in to it a little later. Now suppose the CPU places the main memory address, let us say, 23; so from 23, the cache address, namely, 3 must be derived. It is not much of a problem. So the CPU, while executing the program, will come across an address, which in fact, relates to the main memory address. But though the CPU may place a main memory address, the CPU will look for that information only in the corresponding cache location because that is the fastest way to do it in case it is there. In case it is not there, then we talk about the miss penalty. So CPU places the main memory address and from this particular address, the cache address will have to be derived – cache location address or cache memory address.

The cache location one will have to be derived. Now that depends on the specific arrangement that you have. Let us see how it will be derived – we have to have this algorithm. Generally we can say this cache address for the arrangement will be derived this way. We can say this algorithm is the easiest to derive in this particular case. Essentially, there is a way – I will give another example; then you will know why it is so. So this particular one will be given as that is the arithmetic related to this. That is the algorithm related; this will be modulo, then we have the number of cache locations. I will shortly explain what all this cache modulo is. You take the main address, that is, the main memory address. You have to take modulo of this and then the main address. In other words, take a look at the chart you will see that we have number of cache locations – there are 10 of them. So in our case, it is, say, mode 10 of the main address. In case the main address is 23, from this, 3 will have to be derived. As I said, in this arrangement, it is easy, straightforward – but then we need to have an algorithm and that is what we are working at.

What is this mod 10 23? This actually means it is also called remainder arithmetic or residue arithmetic. This is nothing but takes 23; divide by 10. Now the quotient is 2 and the remainder is 3. So mod 10 actually refers to that particular remainder, which are 3. This is how the cache address is derived; just taken a look at the chart – you can see it goes straightforward. If you draw this 2, that is, 3, actually you are getting it by this particular one. The same way, suppose we have 26, then 26 divided by 10 will be 2 and the remainder will be 6 and that particular 6 will be the cache address.

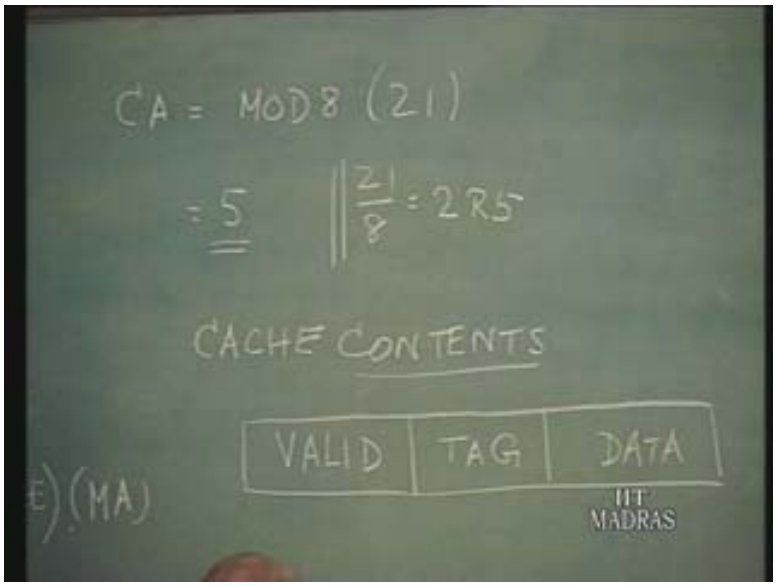
So as I said in this direct map cache, this is what you have. Now let us take a look at another example; I am varying it slightly – let us take a look at the chart. In this, you can see it is not directly derivable mainly because you have only eight cache locations, and of course, for the sake of simplicity, I am giving the decimal address and the subsequent addresses will all be serially numbered. Now you can see that up to seven it is alright; there is no problem. You must have some of these not by dropping any digit; you can get the cache address. The test you have done here, you follow the same formula, except that in this we said earlier cache address is modulo number of cache location. In this the cache location is reduced to eight so it is mod eight that we have to take.

Let us take some example. What is the cache address for the main address? That is, in this case, mod eight main addresses, let us say, is 21. So as per the previous one you would get essentially 21 by 8, which will have 2 and the remainder is 5. Just take a look at the chart: for 21 you get 5; so that is what it means. This particular one is derived like this; this is the cache address.

There is some mapping that will have to be done between the main memory address and the cache memory address, and even this particular second example also is a direct map cache. That is mainly because what it says is we take any specific location that cache location can consist of data from these specific locations only; that is, of the main memory location addresses, 3 or 11 or 19 or 27 only can go in there. That is why we say it is direct mapped. Instead of deriving in this way, we can also have the same information in another form, may be in a tabular form or whatever it is. So now, when many memory locations can be mapped on to the same cache location, then the information must also be available. One thing is to derive the address, another thing is, suppose you go the reverse way, suppose you see cache location 3 – does it contain the contents of memory address 3 or 11 or 19 or 27? How is that to be got? We have to store the information. Anytime the contents are 3 or 11 or 19 or 27; now we proceed the other way. What does cache address 3 contain? There must be extra information, otherwise we would not know. We only know that one of these is mapped on to you that are all. So that information must be there and then to start with let us say right at the beginning before anything else is started the system is just switched on.

What does the cache contain? We do not know; the cache may contain some trash information, that is, some bits, which may not be related to what we want; it may not be related to the data that we want. Then we say that the cache, to begin with, contains invalid data. Only subsequently, when some blocks of information are moved from DRAM to cache, we talk about valid data being there, and in this example, we know that contents of this or this or this or this can be in this – any one of these is valid. Now which one of these is valid, we must see. To start with, that location 3 may not contain anything; it may not contain any of these. It may contain just like that because power is on; you say it is arbitrary. Is it not? So actually you have to worry about what are the things that must go into the cache memory. In other words, what are the cache contents? One thing that we know is that it will have the data, that is, the data that is required, which comes from the DRAM.

(Refer Slide Time: 25:35 min)



The other thing is that it must have some extra information, which indicates this particular data is from which part of the DRAM, which part of the main memory. Earlier, we saw it could be one of the four locations – which one? That information is given by what is called a tag. So the cache must include the actual data; it must include information about which part of memory has tag; and then another thing is that there must be at least one bit of information which says that it is a valid data or that it is not a valid data. So in other words, the format of the cache contents consists of three fields: the actual data, tag information and the valid field. Valid field it just only 0 or 1; just only 1 bit will do; the tag will depend on our mapping.

For instance, shall we take a look at the chart again? This location 3 has to say whether it is this or this or this or this. That is, in other words, whether the data from this row or this row or this row or this row is there. Basically we need to have four – so 1 means this; 2 means this; 3 means this; 4 means this – so if this tag information is there, then we know which one of the four is actually there – the actual contents of that location. So while trying to improve or increase the CPU utilization, see what all is happening: number 1 is about hierarchies of memory; and number 2 is this: once you say that cache will be of a reduced size but it will be very fast.

When it is reduced size, then we have to say which part of the main memory is in the cache and so on. So now you can see though essentially CPU is going to deal only with the data, it must also have this information because of the internal arrangement. Now the user is blissfully unaware of these things – this is important – the programmer need not really bother about this. This is all for the designer so to say; so this valid bit will be usually 0 to start with, and then, later on, this particular block is used – block basically means just one unit of information, it can be any number of bytes; we will talk about it. So for that particular block of data, whenever the relevant data comes from DRAM, then the valid bit will be set to 1; I am talking about the valid bit set to 1. So we may say it will be 1 when in use; when this thing is in use then it will be set to 1.

Now why do not we take an example and then work out? Suppose in cache location, let us say, 2, the data from main memory location 32 is already there. Then we would say the valid bit is 1 and whatever be the data in memory location 32, I just refer to it as the contents of memory location 32. That will be the data that would be there – cache 2, and what will be the tag? Thirty two will always get mapped to 2. Let us take a look at the chart and then see. Memory location 32 will always be in location 2 and the tag it will be referred to by some number, that is, tag 3, which will identify. In other words, in this simple arrangement, tag 3 and cache location 2 has no clue about the memory location being 32. So a tag will uniquely identify which of the two it is in. Otherwise, if the tag were 1, then the data contents will have been from location 12.

Let us now work out the relation between these. Suppose the CPU generates the sequence of addresses like this, that is, the CPU is generating the main memory addresses. Assuming it first generates 22, it is a little arbitrary, and then generates 23, then 70, then again it generates 22, and then again 7 and then 17 and then 22 and so on. Now what exactly is happening? The CPU is generating these main memory addresses, and what will cache contain for each of those instances? Let us work it out. So first we start with the sequence of memory addresses; that is, the main memory. We will write that down and then see what the corresponding cache address is.

And then, we will see whether this particular process has resulted in a hit or whether there was a miss. And then we also note down the corresponding valid bit, then tag bits, the field, whatever it contains, and of course the data part – we know roughly what it must be. Actually these are the contents of the cache. Now we will start: the first one is that CPU generates the address, let us say. This is, to begin to with, the very first address that the CPU generates: 22. Now the cache address is 2 as per our arrangement, and to start with, cache location is 2. What will be the valid bit? The valid bit must be 0 because it just starts with the cache; does not contain any valid data. So initially it would have been 0, and by looking at it, at the cache contents of 0, the system knows that the cache does not contain the required data. And so the contents from location 22 must be brought in to this part of the cache. So whatever be the data, we just recall maybe 22 were containing some number.

What about the tag? Remember the algorithm we were calculating? Let us take a look at the chart again and then see. That is we are talking about location 22 and it is in cache location 2; so the tag must be 2 to indicate that it is from 22 memory location. So that is the tag. Now having got this, what about the valid bit? The cache contains a valid data. So this valid bit; 0 must be removed and the current content must be 1. Now before going in to it, the CPU really misses the data. So this is a case of miss, and having missed data, it brings in these contents from location 22 and then sets this valid bit as 1. Now just for argument sake, suppose the next CPU address is also 22, then by looking at it – valid bit and tag 2 in cache location 2 from tag 2 to cache location 2 – we will give the addresses 22 and then valid bit is 1, which means the data is already available from location 22.

Now let us get back to the problem; after 22 the CPU generates the address 23. Let us put it down, that is, the next address 23. So 23 will be in cache location 3. Now recall the cache initially contains nothing; right now all we know is that in location 2, it has a valid data; that is all. So for cache location 3, as before, it is a case of miss; and valid bit would have been 0 because nothing was brought in earlier, and now the tag will be 2 as before and because there is a miss, a miss penalty cycle must be gone through and the data from location 23 must be brought in – that is the next case.

After 23, there are 17. Now you must be quite familiar; so for memory address 17, we have to look into the cache location 7, and again it is a question of miss. We do not have this data from location 17. And so that is indicated by valid bit 0. After this previous one was brought, this valid bit must have been set to 1 – I forgot to mention that. So the data from location 17 is brought in; the tag for this in our previous case would have been 1; this is the tag bit and, having brought this valid information, this is set as 1; the valid bit is set as 1. See here: the CPU generates the memory address 22 and the corresponding cache location will be 2. For the corresponding cache location 2, we already have valid information and what is to be checked is the tag because cache location 2 can contain memory location contents 2 or 12 or 22 or 32. So it is the tag which is going to indicate that it is 22 and not the other things. Now we find that this is a valid bit and the tag is 2, which means the data field contains content location 22. For the first time, I think, we are coming across the fact that the required data, that is, the data required by the CPU, is available and so it is a question of hit. There is no change; it is already available. That is, valid bit is 1 as before, tag is 2, and for the first time, CPU fetches the data which is already available in the cache.

And what is the data available in the cache? It is from memory location 22. Next we just took 7. CPU generates the address 7. Now in cache location 7, we have a valid bit, but look at the tag bit – it says 1. What does it mean for cache location 7, if the tag bit is 1? It means actually it is the contents of location 17, but what is required is 7. The corresponding cache is also 7. So in location 7, we have the contents of 17 but what is required is from 7. So this is also a question of miss, even though you find that the valid bit is 1 mainly because the tag shows that it is not from location 7, but from location 17, which is what we had worked out. So this is also a question of miss; now into this cache location 7, the data from memory location 7 is brought.

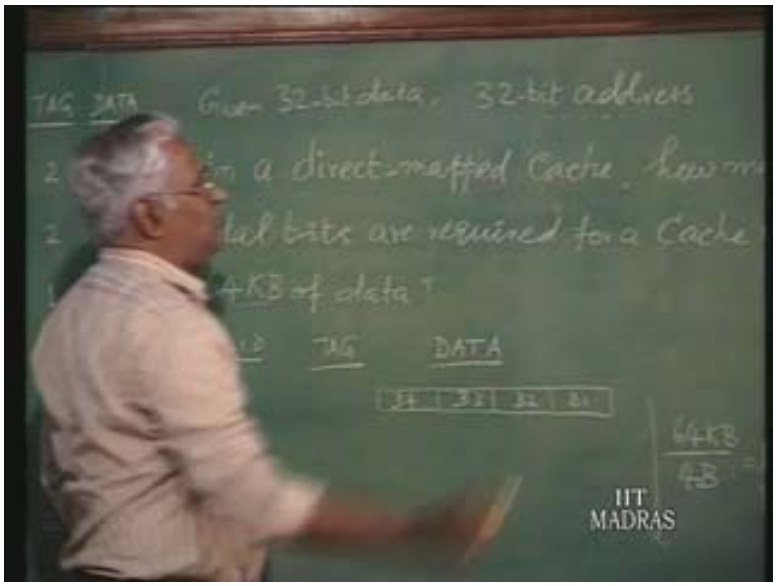
Earlier it was containing data from memory location 17; 7 is brought and the tag corresponding to this is 0, and valid bit is 1. For moments just take a look at this: memory address 17; cache location 7; memory address 7; cache location 7. So, these two are distinguished only by the tag. The contents are 17 in this situation; that is indicated by tag 1. The contents of memory location 7 are indicated by tag 0. So this is a question of miss again. After 7, we have 17. Again the cache location is 7. Now the valid bit corresponding to 7 is 1; valid bit is 1.

(Refer Slide Time: 43:31 min)

But what is the tag? The tag shows 0, which means memory location 7, not from 17 or 27 or 37 is available and so this is also a question of a case of miss. And from memory location 17, the data must be swapped in and the tag must be set to 1 to indicate that it is 17 and valid bit is set as 1. In our example, we have taken 22 after 17 as the next address generated by the CPU. For 22, the cache address is 2 and valid bit is 1 and the tag is also 2, which means currently the cache location 2 contains the contents of memory address 22, which is already there. So this is the question of hit and there is no change – for 22, the tag will be set as 2; the valid bit is set as 1. Now when we proceed in this way, we will find that we are not done for all the cache locations as we are seeing for 2, 7, 3.These are only three locations; we are seeing that because of the example we had taken. So for these three locations, we indicate the valid bit, tag bit and data bits – actually the three fields.

Now you can see that after you proceed, sometimes, the valid bits will all be 1; that means the respective cache locations contain some valid data – precisely which location is decided by the tag. That is the sequence in which this cache is participating. Now one thing is the cache contains something other than the data bits, it also contains the data information and also the valid bit information. As I said, locations 2, 7 and 3, will be containing this information appropriately. These are the ones, because the previous one would have been changed.

(Refer Slide Time: 48:55 min)

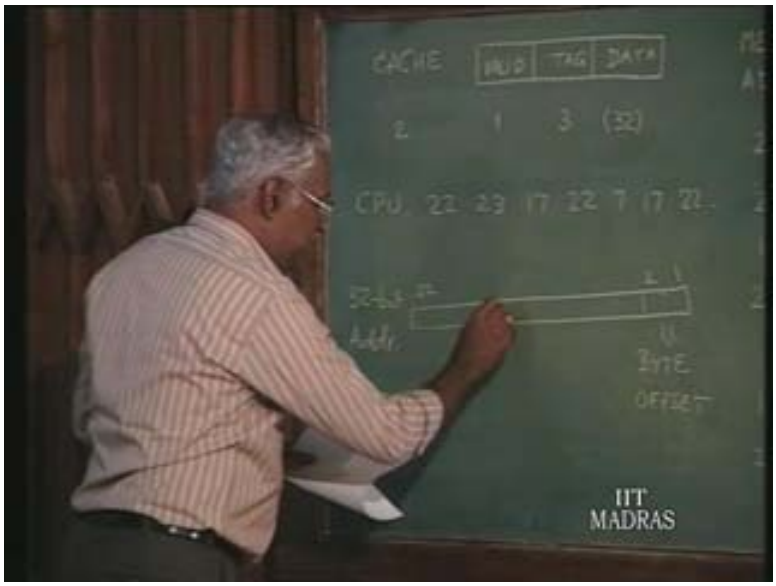


Now let us work out one more examples. Suppose we have been given that we deal with 32-bit data and the CPU also generates a 32-bit address. We need that information also, 32-bit address; we will assume that the memory is still byte addressable and so, that data, if and whenever the CPU wants, can also refer to any 8 bits of this data – that is the idea. Now the problem is, we will assume the same direct mapped cache. The question is, in a direct mapped cache, that is, the same arrangement; we are not changing the direct mapped cache. If you have this information, then how many total bits are required in the cache? The cache contains different fields; specifically we know it contains data field, tag field and valid field. As far as the data field is concerned, we know that it is 32 bits. But, what about the tag? What about the valid? We more or less know valid is going to be 1 bit, but, what about the size of the tag field?

Let us work that out. You will see that will be worked out from the 32-bit address. In this cache, how many total bits are required? We will assume one more thing, that is, the cache size. For a cache of, let us say, 64 kilobytes of data alone, apart from this data field, we also have these fields. Essentially we are going to work out the size 64 KB. So now we know this much, which we have to accommodate for the valid bit. Then we have to accommodate for the tag bits and the data part. What I will do is I will assume that the block of data is still 32 bit. A block of data is the minimum unit, which passes between two levels; that unit can be any size. So I will assume there is 32-bit data in each cache location, which means it is actually 4 bytes. So the data field is 4 bytes and that accommodates for 32 bits. I will just call these byte 1; byte 2; byte 3; and byte 4. So I am assuming the block of data as 4 bytes.

Now since it is given that the cache is 64 KB and you have 4 bytes in each block, now I know how many blocks are there. That must be 64 by 4, which means there must be 64 KB, by 4 bytes each, and so, we will be having 16 kilo blocks.

(Refer Slide Time: 51:25 min)



So there are 16 K blocks are there we are talking about 16,000 blocks. Now let us take the address and work out the other things. We do not have to discuss the valid bit much because if the data is valid, it is going to be 1; otherwise it is going to be 0. So the valid field size is only 1 bit. Now the tag is what we have to work out, which we will have to work out from the address generated by the 32-bit address size.

So let me say, suppose I have this 32-bit address starting from bit 1 to bit 32. What did I assume? Each block has 4 bytes and then I have to accommodate 16 K blocks incidentally. How do you compute 16 K? It is actually 2^{14} – just work out if you have any doubt; 16 K will be 2^{14}. So I need to have a 14-bit vector to indicate that there are 16 K blocks. And then, once I know which block, since I said it is byte addressable, I should also know which of the bytes, if necessary. If it is only 32-bit data, we do not have a problem. It is enough if I identify the block number; but if I also want to know which of the bytes; I need to have at least 2 bits to identify these: one of these four. So in the address, I have to allow for 2 bits; let me say I allow these 2 bits. These particular 2 bits are for indicating which of the bytes. So these 2 bits are called byte offset; that is what are the combinations possible – 00, 01, 10 or 11.

One of these four will identify whether it is this byte or this byte or this byte or this byte. Now once I identify which byte, then the next thing is I have to identify which block for which I need 14 bits. The block offset part of it, that is, the next one, 14 bytes, is starting from 3 till it will be 16 – so these 14 bits will identify 14 bits. These 14 bits will identify which of the 2^{14} blocks, that is, one of 2^{14} blocks. This one 14-bit thing is called a block offset.

So we had allowed 2 bits for byte offset and 14 bits for block offset. The remaining ones are: 2 gone here; 14 gone here; of 32, the remaining is 16. So this is what is used for tag field. The tag size will be 16 bits: so 1 bit for valid; 16 bit for tag field; and for data we already assumed 4 bytes, so 32. So 32 + 16 + 1, that is, essentially you have got 49 bytes; just add these 49 bytes. This, in fact, is the size of the cache, meaning 49 bytes by 16 K. So because there are 16 K blocks each of 49 bits, one important thing you have to notice is that essentially, 49 bits include the 32-bit data. So we need this extra thing basically to keep track of which part of memory address is in any cache location. This, of course, is specifically for the direct mapped cache organization. You will have different ways of working out for different organizations.