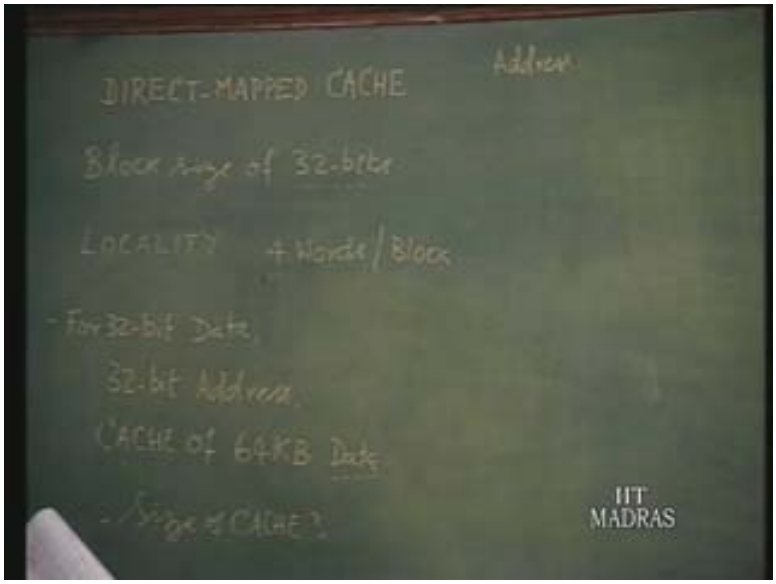
Computer Organization
Part – II
Memory
Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology Madras
Lecture – 18
Cache Organization (Contd….)

In the previous lecture we had been working out an example of a direct mapped cache also. Specifically, the example was talking about the case when the block size is 32 bits. Now what is this block essentially? It is one unit of information that pauses between two levels. Here we are talking about that between CPU and the DRAM or the main memory – these are the two levels. We took this block of size 32 bits, but then we also said earlier that generally the memory is byte oriented, that is, byte organized memory. Any time we may be interested in getting at a byte of data, when we have 2 bytes, generally it is called a word. Then when we have 32 bytes, that is, 2 bytes, it is 16 bits; when we have 32 bits, then it is called a long word; and then 64 bits is a very long word and so on and so forth. This usage keeps varying. Though we may have word, long word, very long word, generally we are also interested in accessing at the byte level. So what exactly is the data that will be subsequently used by the processor depends on the processor size.

Now we do not know just by saying that the cache has 32 bits, what exactly the CPU deals with. It may be 16 bits, it maybe 32 bits, it may even be 8 bit or 64 bit – whatever it is; nothing is known about it. Now when a program is organized and then subsequently executed instruction after instruction, then we have also seen that the subsequent instruction cycle involves instructions in adjacent locations. So as far as the CPU accessing the data is concerned, it is meaningful to assume that subsequent bytes of data of the memory will be accessed. This is what we were explaining earlier as the principle of locality, that is, something which is local. In this particular case, that is something which is immediate or is the next word, next byte, and so on and so forth, based on the principle of locality. So when we say that we are organizing the block size as 32 bits, we know that it has 4 bytes.

Let us assume that the processor is also a 32-bit CPU. Then it is meaningful that the processor always accesses the 32-bit data and it is also meaningful to assume that subsequently, 32-bit data will be accessed. The probability of accessing these neighboring things is very high. So why have a data that is only 32 bit? Why not has this subsequent 32-bit information also stored along? So we move on to the next one, in which we see what happens if the cache is organized such that it has four words.

(Refer Slide Time: 08:07 min)

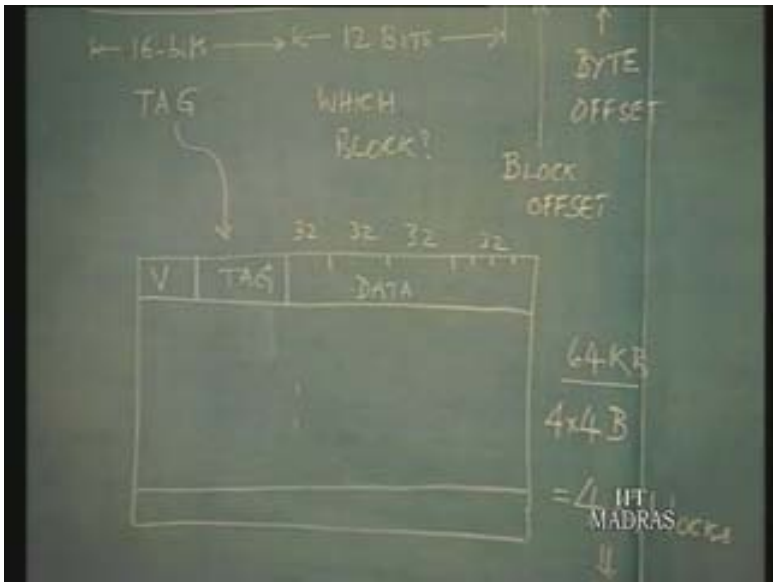


Now here what I am saying is the word is 32 bit – four words per block. In general, it is more than 1; that is what you have to see, it is multi-word. If you assume that the 32-bit CPU was dealing with this cache then it is possibly all the time it was accessing one word or 32-bit size from the CPU point of view. From the cache point of view, it is the block, one 32-bit word. Assuming in the same CPU, this has four words in a block, CPU will access only one word at a time. That is a different thing. But what come from the DRAM at any time into the cache will be four words. As now as I said we can take it as multi-words; if you think that the 32-bit word is a single word, here we are talking about multi-word.

Why has multi-word? As I said, it is meaningful to assume that the next or immediate data will always be accessed, in which case, let us work out the problem again. What is it we had assumed earlier? We assumed for 32-bit data; I am just rephrasing the problem which we are worked out earlier – and we also assumed 32-bit address. The problem was that in case we have a cache of 64 KB data size; this is only the data part of it, the data field of it. The data size is 64 KB; then the question was, what is the size of cache? It means essentially assuming valid bit is 1. What will be the size of the tag bit – that is what essentially it will boil down to? For this we had work out the problem. Now we had worked out the previous problem assuming we have only 32 bits in each data field. Since we are taking four words, we are talking now about four 32 bits of data available in one block. Now let us just work out the detail as before; the address is given as 32 bit. So let us see which one that is: bit 1 to bit 32 – now this is the address part. Let us see what possibly the cache consists of. We know that each cache location must accommodate the valid bit, just 1 bit; it is called Vbit. Then we have the tag field, and then we have the data. For this problem, we assume four words per block.

So each block consists of valid field, tag field and data field. Now this data field, for this problem, will have four words, each of 32 bits. This is 32-bit data; this is another 32-bit data; this is another 32-bit data; this is another 32-bit data – so we have four of these. Earlier we had only 32 bits in one field; so actually, 32 bits means essentially that consists of 4 bytes. As before, whatever we had worked out earlier holds good. Now for identifying which of these 4 bytes, we have to allow 2 bits in the address and that is what we called byte offset. The cache will consist of many blocks like this – what is the size? We have been given the cache of 64 KB data part; that is, the data part alone is 64 KB, which we worked out earlier. From that, we can see how many blocks are there, that is, we have 64 KB, that is, this part alone. And each block consists of 4 of 4 bytes each. So 4 of 4 bytes are 16 bytes; each block has 4 of 4 bytes. So we have this divided by that, which will be 4 K blocks. That is, 4 into 4 or 16 bytes; so we have 4 K blocks.

(Refer Slide Time: 15:22 min)



So there is actually 4 K blocks here, and to identify which of these blocks we would need, that is, 4 K means we could need 12 bits. Because 4 K is 2^{12} and so this means we need 12 bytes to identify these blocks. Once you have identified which of these blocks, within the block we have to identify which of the 32-bit data. Within that block, there are four such, so we need two more bits, specifically, bit 3 and bit 4. This is our block offset. So the first two bits in the address identify which of the bytes within the 32-bit data. The next two bits identified within a block are the block offset and then we would need 12 bits to identify which of the blocks because with that 4 K blocks can be identified. So starting from bit 5, if you said 12, it will go up to 16; so you will have 12 bits here. With these 12 bits you identify which of these blocks and for the direct mapped cache arrangement you would be using the remaining part, 17 to 32, which means another 16 bits. These 16 bits is our tag field. When an address comes – we already talked enough about valid bit – we know why we have seen that. Now when the CPU places the address, the corresponding tag field of it will be seen; it goes like this. Do you remember the direct mapped cache? At any time from one of n memory DRAM locations, which one will be the cache contents? This is what we have to know; we had seen that information earlier through many examples.

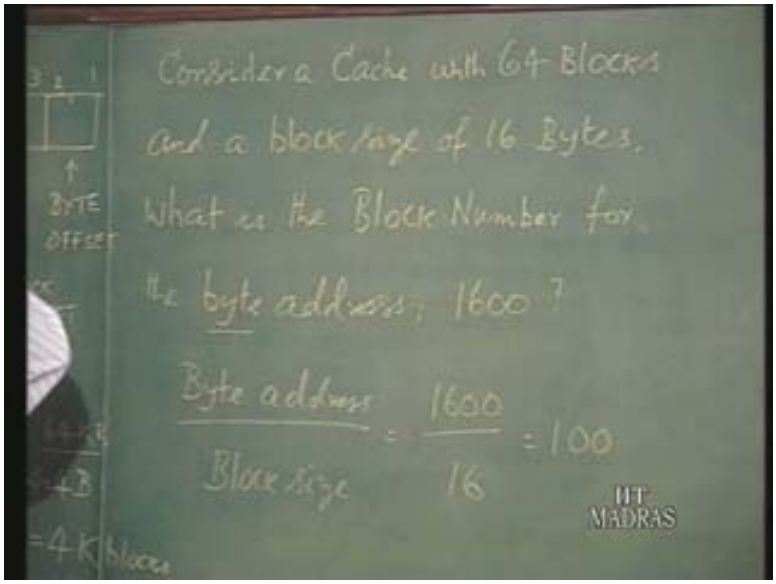
Now it is possible that in a given block, any one of these 4 K blocks, the required content may not be there. What exactly is required is indicated by the tag. In our examples we have seen earlier that in a cache location 4, the contents of four maybe 14 or 24 or 34. So in that tag we have seen in our previous example, it will be 0 or 1 or 2 or 3; that is the tag information. So in these 12 bits of the address, where the CPU will indicate the block, whether that particular block has the required information or not will be indicated by this tag.

So what is called a hit now? If the valid bit is 1 and if the tag field of the given address and the contents of the given block match, what we have here is the cache content. This is for the data, this is for the tag and we will work out the size of valid shortly. For a given block, if the tag of the incoming address and the tag content in that particular block match and the valid bit is 1, then this situation corresponds to CPU having found what it wanted in the cache. So we say the CPU has achieved a hit. Otherwise, there is going to be a miss. When will it be a miss? The valid bit may not be 1 if it is 0; now this is one situation when, to start with, the RAM contents maybe anything random and if valid bit is 1, the tag fields may not match the tag field of this. The contents of the tag field of the block and the contents of the tag field of the address may not match, in which case, the block contains some other location's data, in which case there will be a miss. This in fact summarizes the hit situation: if valid bit is 1 and the two tags match then it is a hit. Incidentally now that we have seen that tag field is going to be 16 and then this valid bit is just 1, so $1 + 16 + 4 \times 32$. So the cache is so many bits wide and you have 4 K of this. The cache contains 4 K blocks so 4 K multiplied by the sum of all these will be the cache capacity.

Now you can see that though essentially the data is only in this portion, you need all these extra things mainly because we want to see that the CPU can run what it wants to run at the highest speed that is possible. So let us work out another example, in which we will see how exactly to calculate the block number, given a byte address. Suppose we consider a cache. Let us consider a cache with 64 blocks, and what is the block size? Suppose the block size is say 16 bytes. It is not different from what we have here, is it not each 4 byte and there are 4 words; we are talking about the same thing. Now, given this, what is the block number for a byte address? We have to know that. So you are being asked to find out the block number for a given byte address; what is the byte address? For the byte address what is the address? Let us say 1600 – so that is the question. That is, given a byte address we have to know in which block it will come. In the previous example, we had as many as 4 K blocks – 4000 and odd; here we have only 64 blocks; that is the difference. How do you go about it? The way to do it is, take the byte address, which is 1600 in our case and divide that by the block size.

Then you will know which block it is, that is, what you are talking about is take the byte address that is 1600 hundred, divide that by the block size, which is 64 in this particular case – 64 is not block size the cache size is 64; the cache consists of 64 blocks; the block size is only 16 bytes. Now 1600 into 16 of course is a number you get it as 100. Where is the 100? The cache consists of only 64 blocks; how do you do it? Basically it means in a cache the 64 blocks are filled and then, for 100, you actually roll over and fill it. In other words, mathematically, we have come across this situation earlier. So if you want to calculate the actual block number because essentially what we have is only 64 blocks, the actual block number must be calculated as 100. We have seen earlier mod 64 is the modular arithmetic; 100 mod 64 means basically 100/64, and then, you have a quotient and a remainder.

(Refer Slide Time: 22:43 min)



That is essentially the way to compute: it is 100/64; what you get is quotient, which is 1 and the remainder is 36. In other words, the actual block number will be 36. So it is the 36th block that is the one which will hold this particular byte address. As far as this cache dealing is concerned, what is it we have studied? We have only been reading from the cache so far. Now, what about writing into cache – because the CPU would not only read but also write into the cache. For this, you must know that we have the CPU. The CPU accesses the cache and we had said earlier that if cache does not contain what the CPU wants, it will be accessed from the DRAM memory. Now you see the problem: CPU wants to read something from the cache; if the cache does not contain it, it will be got from the DRAM. One CPU has it in the cache, it can read – that is not a problem. The contents that the CPU reads from the cache and the contents in DRAM will be the same.

Suppose CPU writes into that location in cache, then the cache contents and the DRAM contents will be different. When CPU writes something into the cache, after all from CPU point of view, it is just a memory; it can read or write. In this particular case, if CPU writes into the cache, what happens? There will be a different number in the cache, different data in the cache, different from what was in the original memory. The situation is not bad; what happens at some point in time is that the cache content will be removed and then, in that particular block, another data may come. At that instant, the modified cache contents can be written into the DRAM – this is one way. Another way is whenever CPU writes into the cache, you should also write into DRAM. What is the problem you are facing in this? We know that CPU can write into cache very fast because it is fast memory. But writing in the same time into the DRAM we know that it's a slow memory. The very reason cache has come into being is to take care of the speed matching between the CPU and the memory. So, whereas CPU can write into the cache very fast, it cannot write so fast – CPU or some other controller – the same content cannot be written into the DRAM very fast and so we are again going into the problem of speed. That is, the mismatching speed – one thing is as it writes into the cache it writes into the DRAM without minding about the speed.

Then it is called a write through cache. As a CPU writes into the cache, the DRAM contents also will be modified or updated; in which case, we talk about having a write through cache. So you have seen that whenever DRAM contents are also written when cache is written, the other one, which I was suggesting is whenever the cache contents are modified – the right word is whenever the cache contents are swapped out – new contents come into the DRAM.

(Refer Slide Time: 32:36 min)

At that time, the data that will be swapped out of cache will be recorded in the DRAM; thereby you update the DRAM contents. So the second one is called a copy back; that is, whenever we can specifically copy back or swaps out; that is, whenever the data from the cache contents are swapped out, you may copy back. Now there is small problem in this. What is that? Suppose there is an I/O part of the program and if we find a specific block that has been transferred to cache and it is active, meaning it is a valid block and it is being used by the CPU, at that instant if an I/O is required, the I/O would be, that is, the input/output would be with reference to the memory. Because this is going to be a huge size memory, the cache is essentially for CPU's reading. For the I/O part, things will be taken from the DRAM. At that instant, what will happen is that since cache contents and DRAM contents may not be the same, the updated data will be in cache but DRAM may have outdated data, which has not been updated. So if you are waiting for copy to copy from cache into DRAM and update only on swap out, then before swap out there is a possibility that some I/O program may need reading from the memory, which is not the updated data. So there is a possibility of that particular thing.

There will be a problem in the second cache in the copy back. What was the problem in the first one – write through cache? We have already seen memory speed mismatching. The problem with copy back is that there is a possibility that in case of I by O, the correct or updated data may not be output, and so that is the problem with this particular one.

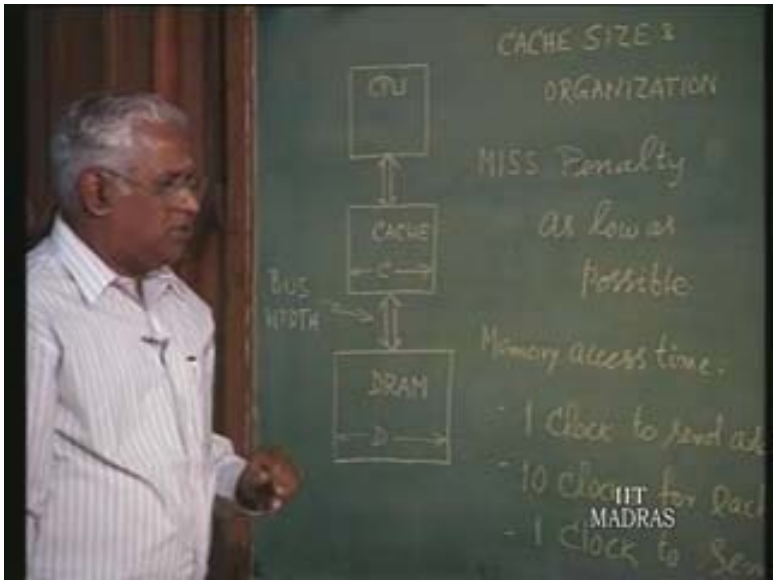
So what is done is, in addition to the various things that you have in CPU block, like valid bit, tag field, data – of course that is the main thing – in addition to this, we also introduced one bit called a dirty bit; what is this dirty bit? We introduce dirty bit like valid bit: this is 1 bit, which indicates whether the cache contents have been changed or not; actually it means whether cache contents had been dirtied or not. In case there is an I by O process which is going on, the first thing will be to check whether the particular block is active in cache. If it is active, whether the dirty bit has been set; if the dirty bit has been set, it means the CPU has altered the cache contents and indicates that cache contents and DRAM contents are not the same. So if the dirty bit is 0, then cache content has not been altered and possibly the CPU has used it only for reading, and it does not dirtied it. So we use that dirty bit and add the dirty bit in the block.

Now we can see the advantages of having a single word and the multi-word block. Earlier, we were talking that it is very likely that adjacent words will be used immediately by the CPU and so it is convenient; but now, it is possible to alter only one of these words and it will be indicated as dirty, using the dirty bit. While copying back, all the four words will have to be written into the memory because at any instance a block basically means the minimum unit that transfers between two levels, cache and DRAM. So suppose four words are going into the cache as a block, even if one word is altered all the four words will have to be updated in the DRAM. In this way, you have advantages and disadvantages of different organizations.

Let us look at some more details of this cache organization with reference to these. Let us study this cache size and organization. So this is the arrangement: essentially we have CPU dealing with cache and then there is some communication between cache and DRAM parts. What was our essential objective? We have to see that CPU finds, as far as possible, all it wants in the cache. In other words, we want to make sure that the miss penalty is as low as possible, that is, the time taken in case miss penalty is time really, but we will not specify that – miss penalty as low as possible – that's our aim. Now you actually calculate the penalty time, but that depends on the way we organize the cache and DRAM. That is what we are going to study now. Essentially, these being related to the time, we have to have some data, basic data, on which we can calculate the times involved.

So let us assume that the memory access time is as given here because what is it that we are talking about? We want to see that the miss penalty is as low as possible, which means the time taken for getting the data is as low as possible. In case it is not, the CPU does not find it in the cache. Always for one address, some timing is needed. Instead of really worrying about time in terms of seconds, we will see in terms of the clock. Suppose one clock time is needed to send the address, we will make use of this data for all our calculation. One clock is needed to send the address. After the DRAM receives the address, let us assume that it needs as many as ten clocks for accessing from the contents and putting it. That is, it is called memory access time – ten clocks, whatever be the clock. If it is 1 MHz clock, that timing is going to be 1 microsecond. So let us say ten clocks for each DRAM access, that is, to get from the memory taking ten clocks.

(Refer Slide Time: 40:09 min)



Let us say one clock is needed to send that word to the cache; ten just for accessing and then one more clock is needed to send the data, that is, to send memory word to cache. This must go to cache; only when it reaches the cache, the CPU can access that. So assuming these timings, we will work out. What else do we need to know? You remember what we have been talking about? We are talking about one word block; four word block; multi-word block in general.

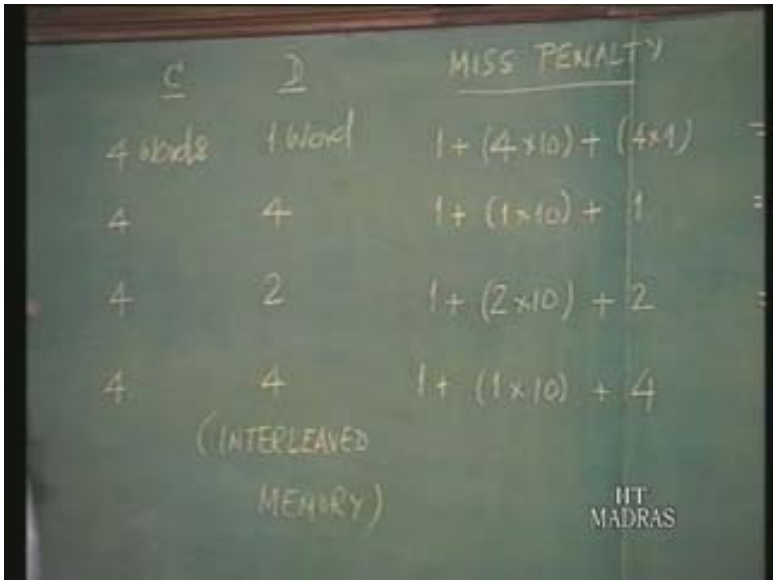
Now we discuss the same thing: how many bytes can be accessed at a given time from the DRAM? Some times we may access only one word from DRAM, some times more than one word. Whatever is accessed from memory, we will assume this particular bus width is such that, because the width of this bus is known, we should know all these things. Now whatever we access from the memory, let us assume the bus width is such that whatever is accessed is passed on to the cache. So where C is the width of the cache, as far as the data is concerned, we are not bothered about the tag and valid – all those things are needed for organization. Suppose the cache width is C as far as the data is concerned, and the DRAM width is D. We are talking about it because the RAM may contain 64 bits the cache may contain 128 bits or more; so these are independent.

By having different values of C and a different value of D, we are talking about different organizations. But we must be careful about this bus width; so we will assume that whatever it is; this access can be passed on. Let us work out the detail. Suppose we are going to assume some different values of C and D, and then, based on the access times, we will work out the miss penalty. That is our job right away. Suppose C is four words in general, and D is one word. So what I am assuming is that, let us say one word is a 16-bit word, in which case, what we have here is the 64-bit cache. So it is a multi-word block. Suppose D is one word and C is four words, it is meaningful to assume in that case that the bus width is one word, because one word is going to be accessed at a time and it is being passed on. So in four steps, it will be put in the cache. What is the miss penalty there? What we mean is that the CPU is trying to access a word that is not available in the cache.

So from the memory address that is passed on by the CPU to the cache, the particular address will have to be sent to the DRAM because it is not available in the cache; it is sent to the DRAM. So one clock will be to send the address and from that particular address, the data will have to be got. What actually goes between cache and DRAM – one block of information – that block, in this particular case, corresponds to four words. So even if only one word is required by the CPU, the whole block will have to be replaced. Four will have to be read because the whole block will have to be replaced. So there are ten clocks for each DRAM access; after it gets it, it takes one clock to send that word here; that is required for this transfer, in which case, the miss penalty, which is one clock, is needed to send the address. Because the cache is four words, four words will have to be read from the DRAM, each of which will be ten clocks. So $4 \times 10 = 40$ clocks are needed. Those four words will have to be sent to be cache, which, for each word, needs one clock.

So again, here it is 4 into 1. So I will repeat one clock to send the address and because a block will have to be replaced, four words will have to be accessed from the memory. So we need 4 into 10 is equal to 40 clocks, and again, four, 1 clock each, to send word to cache. So you need 4 into 1, which means that this particular thing needs about 45 clocks. That is the miss penalty. Now if C were four and D also were four, meaning we have a four word block and DRAM also is four words, it can be accessed at a time – that is what it means. In that case, you have to assume bus width as four. In each case, bus width will be different; so we are not talking about it. If this is four, what happens? We need one to send the address, and now four words can be read. So it is enough if we read it only once.

(Refer Slide Time: 49:21 min)



It says, in this case, one DRAM access is going to access four words; so only ten can be accessed at once. Once a reading is to be done, four words will be received, and just in one step, the four will go into the block. So now we have only 12 clocks.

Suppose we have four words, and DRAM is two. If C is four and D is two, then what happens? We have one to send the address and two words from DRAM and a four-word block in the cache. So twice it has to be read from two locations, so we need 2 into 10 clocks, because for each access, ten clocks are needed. This will be sent in two steps and so again two clocks are needed to send the word. That would be 23 clocks.

Now let us just go in for a slightly different organization here: the DRAM. I will slightly alter the cache and DRAM part alone; that is, suppose we have a cache and the DRAM consists of four different memory banks. It is not a single one as we had assumed earlier; so four different memory banks are there that will be accessed. So DRAM is consisting of four memory banks; this particular one is in fact called an interleaved memory. Now let us assume for that, that is, we have four-word cache, that is cache block is size four, and then we have four here, but each one is one, but together we have four. Now, as I said, this is called interleaved memory – this is different from this case just these two cases are different so here we have the interleaved memory. Remember we talked about this odd memory and even memory? Now here we have two more things: location 0; location 1; location 2; location 3; and so on. Then again we have 4; 5; 6; 7. Now what is the idea? Physically, this has been separated; suppose we had only two, we can talk about odd bank and even bank. But we have four here. Now when you physically separate them, once you put in a request for access, all the four can be read in parallel.

Basically, it is a form of parallelism; so once you address, four words will be read in parallel. But what is being transferred will be only one at a time: that is the situation here. So, in this particular one, you have one to send the address, and that address will go to all the four banks. All of them will get addressed and then in parallel, they are going to access the memory in parallel. So in unit time, that is taking ten clocks, all the four will be accessed, but the accessed data will be passed on only one at a time. So clocks will be needed to pass on the word that has been accessed to be passed on to the cache. What is happening with this? We have only 15 clocks; we have four different organizations here. That is, four words; one word; four words; and four words – they are two categories. This is the way we organize, and now, you can see how, by changing the organization of cache in DRAM, we can go on ensuring that the miss penalty is reduced.

That is, it depends on the cache size, the DRAM size, and also the memory organization. The main aim is to reduce the miss penalty, for which you go to any length, any type of organization that you want. Once you have this, then we talk about the bandwidth, which is nothing but once you have the clocks, then how many bytes that can be transferred per clock? If you can express how many bytes are transferred per clock, then we talk about the bandwidth. It looks like it is based on the various studies – the optimal block size of a cache is about 64 bytes. It looks like various things that have been studied. So it is only optimal; it can vary; so optimal block size of the cache is 64 bytes. This has come about in practical studies.