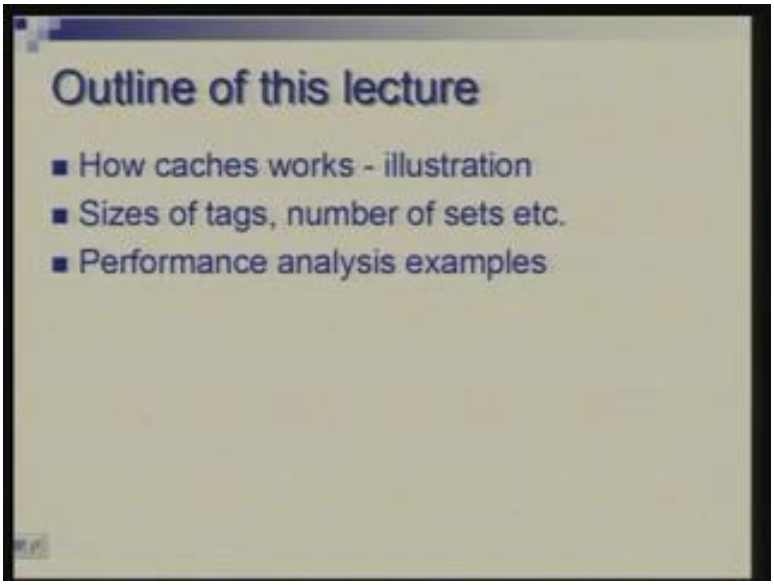
Computer Architecture
Prof. Anshul Kumar
Department of Computer Science and Engineering
Indian Institute of Technology, Delhi
Lecture - 30
Memory Hierarchy:
Cache Organization (contd...)

We have discussed various aspects of cache organization. In today's lecture I will give some illustration about how cache accesses are done and also we will look into some of numerical problems pertaining to cache performance. So this illustration would be essentially like simulation. We will take small example and try to see the effect of various changes in the cache organization for example, what happens when you change the block size, what happens when you change the associativity.

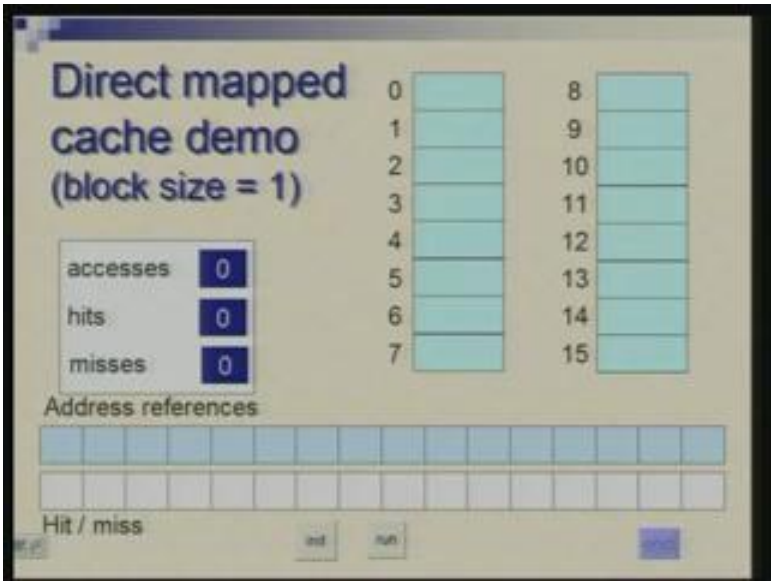
(Refer Slide Time: 00:01:08)



So, in a small example we will look at some sample references to memory addresses and see how you get a hit or a miss and at the individual access level you will be able to see when a miss gets converted to a hit or hit gets converted to a miss by making a change in the organization. This is one way of looking at cache. What is typically done is that you look at it grossly in a statistical sense which gives an overall behavior. But this detailed view detailed illustration is meant to see the exact mechanism which works behind cache accesses, misses and hits.

So We will spend little bit of time on trying to see what is the overhead in a cache organization in terms of the extra bits you require for holding the tags and what are the factors on which it depends that we will see; then couple of numerical examples about cache performance in different context we will see.

(Refer Slide Time: 00:02:39)



So here is a small cache which we will use for demo. This is a direct map cache with 16 words; we are not worrying about bytes and words let us assume that everything is words. So you will get address sequence from memory or from processor trying to access a memory and each access would be examined. We will see whether it is a hit or a miss depending upon what the status we see on the board.

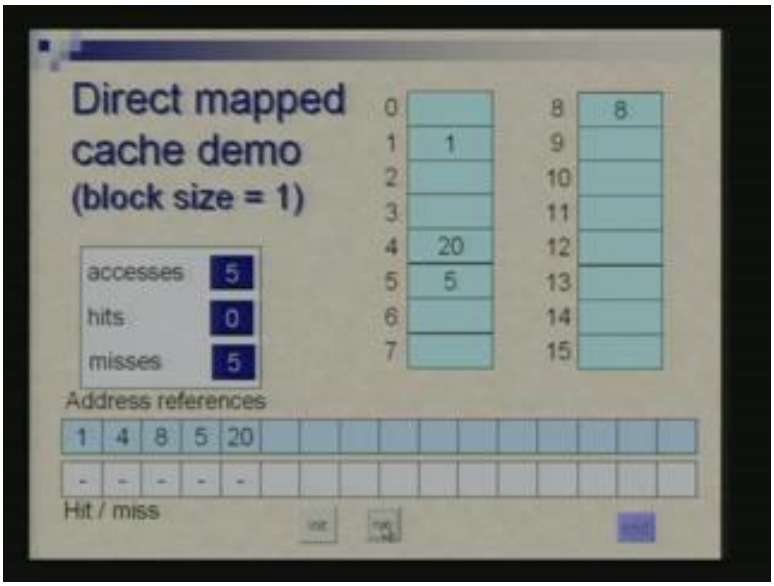
These are 16 words of the cache (Refer Slide Time: 3:22) numbered from 0 to 15 and the addresses which we encounter..... there is some sequence of address which is stored in this demonstration which will come out and we will record the hit and the miss. Also we will keep track of the total number of accesses made and total number of hits encountered and misses encountered. So, to begin with we assuming that it is a direct mapped cache. The feature of this kind of cache is that given an address of memory the location in the cache is fixed. So if cache size is 16 given the memory address you could say address modulo 16 and you will see where it is going to fit. We will not worry about the memory contents but what we will keep record is essentially which word of memory is sitting in which location.

You can see that the first address which came was address 1 and naturally it is supposed to sit here, the cache was empty initially and therefore we record this as a miss. (Refer Slide Time: 00:04:46) Here is the place where I am going to record the addresses and it will record the misses and hits here. So first one is a miss and obviously initially we are going to encounter lot of misses and it is only when you come to reuse of some address when addresses start repeating then there is some possibility of having hits so initially there will be mostly misses. Let us proceed further.

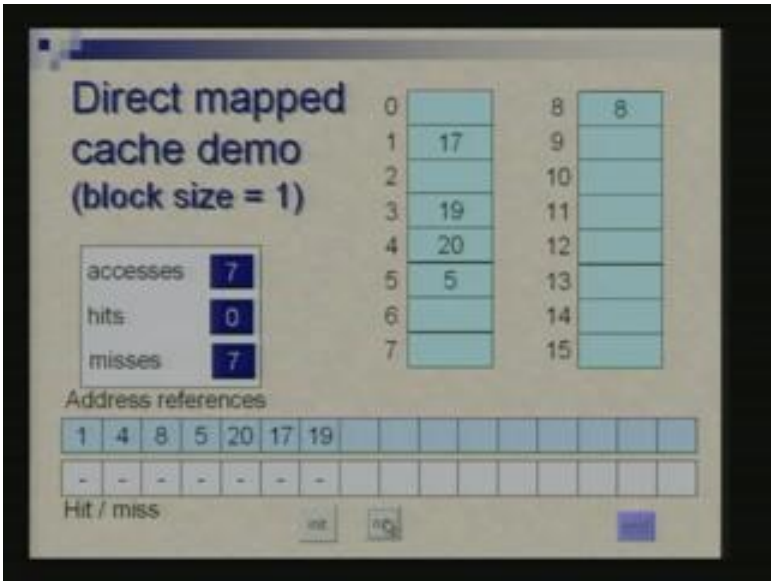
4 sits in this position, 8 sits in that position so just for space I have split the whole thing into two columns but assume that it is a single one dimensional array so you have addresses 0 to 15 and initially numbers are sitting in their natural places. Now you see

what happens. Address 20 which you take as modulo 16 is same as 4. So 4 has been replaced 4 has been thrown off and if there is occurrence of 4 again although we would have brought this to cache we will not be able to find it next time

(Refer Slide Time: 00:05:48)

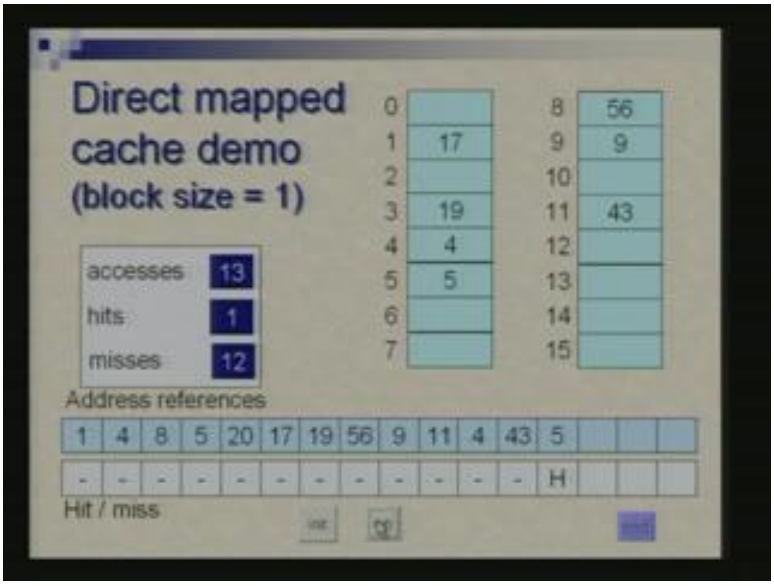


(Refer Slide Time: 00:06:26)



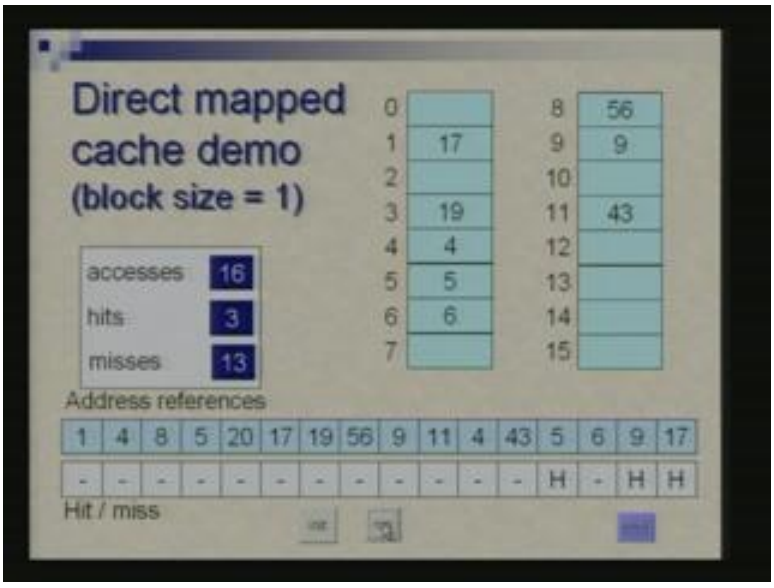
Now 17 comes and replaces 1 so these are the spots where you can see that perhaps different cache organization could have helped. 19 comes in position 3, 56 replaces 8 because 56 modulo, 16 is 8; 9, 11, 43 so 43 replaces 11.

(Refer Slide Time: 00:06:56)



So 4 came again and that time it was a miss;. So 20 replaced 4, I think 4 comes at this point so 4 throws 20 back again. Imagine if your program behavior was like this that 4 and 20 were being accessed repeatedly then they will keep on fighting with each other. So fortunately 5 which was loaded initially in the fourth cycle is still there and so you get a hit here; 6, 9.... 9 is already there so 9 is another hit and then 17 is again a hit because 17 is already there. So the overall tally is that there was 16 accesses which were made only 3 are hits and 13 are misses.

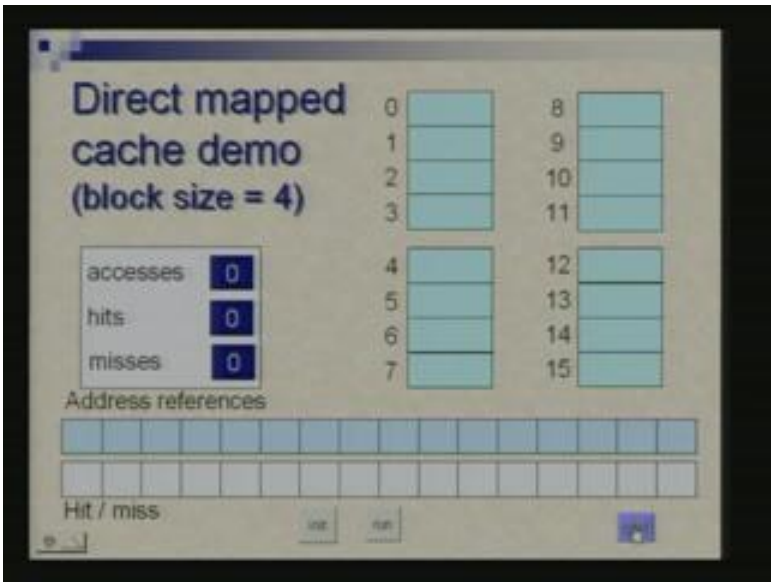
(Refer Slide Time: 00:08:17)



Now what do we expect if we make changes. suppose we change the block size we increase the block size what block size will try to help you with is spatial locality; it will capture spatial locality that if you are having references which are close to each other in address, in those areas you are likely to have additional hits. So the next one we look into is this but we are going to continue with direct mapping so it may not help perhaps in the areas where some data was brought in and it was thrown because of clash.

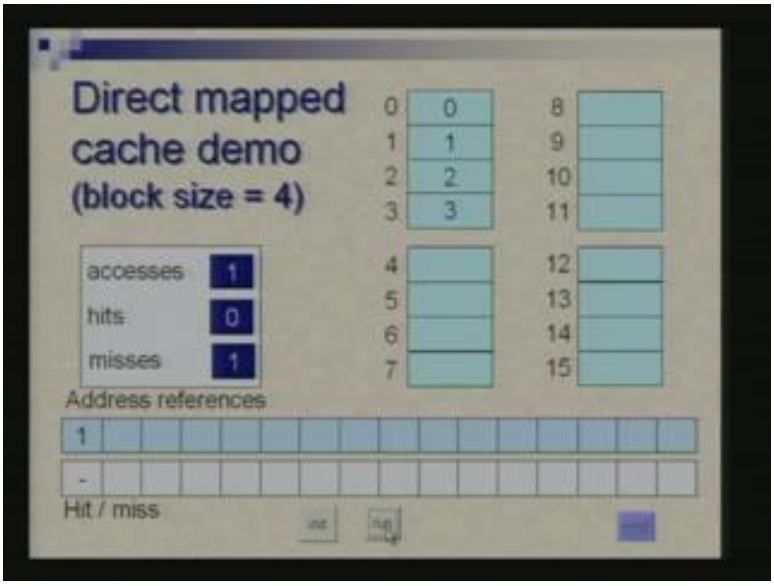
So, next scenario is same direct map cache with block size equal to 4. I have displayed the cache into four blocks: 0 1 2 3 these form one block, 4 5 6 7 form another block and so on.

(Refer Slide Time: 00:09:21)



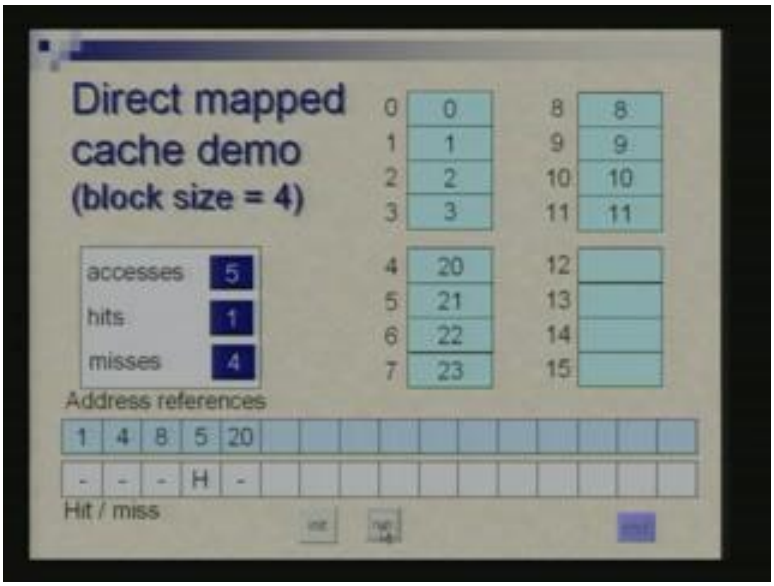
So now the meaning of this arrangement is that when you make an access you are looking at the block level. Tags are actually stored although I will show all the location filled up but tags are stored one per block and when you find a miss you getting the entire block from the main memory and filling up in the cache.

(Refer Slide Time: 00:10:23)



So first time when address 1 was encountered there was a miss but whole block got filled up 0 1 2 and 3 then address 4 was encountered and second block got filled. Its advantage would be seen if addresses within the block get referred again. So 8 fills another block and now since there is a reference to 5 there is a hit because of that so here you can see some advantage of a larger block size.

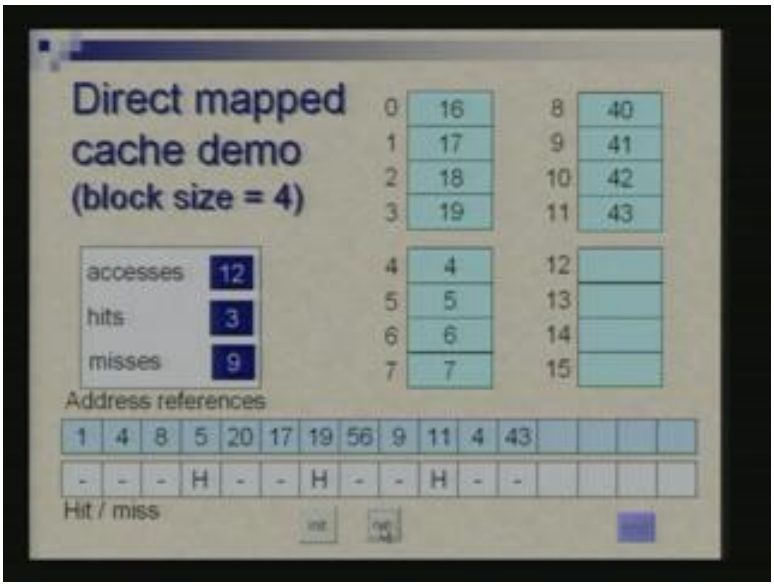
(Refer Slide Time: 00:10:57)



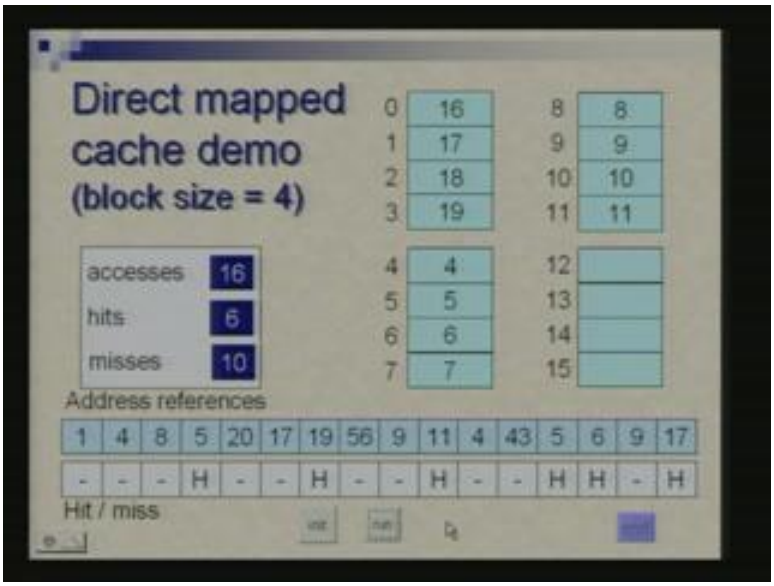
20 replaces this block so that problem continues; 4 was replaced by 20 now the entire block 5 is also replaced. 17 replaces 1 but 19 gives you a hit because 19 is in the same block as 17, 56 replaces 8; again since you are referring to 9 that block beginning from 56

is thrown out again so here is a miss again. 11 is a hit because we had referred to 9 and that block is there. So access to 4 is also a miss here, 43 the block beginning with 8 is again replaced, 5 is a hit because by virtue of 4 which was accessed, 6 is also hit, 9 is a miss and 17 is a hit. So on the whole there are three additional hits here and the total performance is improved. We can see which are the new hits which have come.

(Refer Slide Time: 12:01)



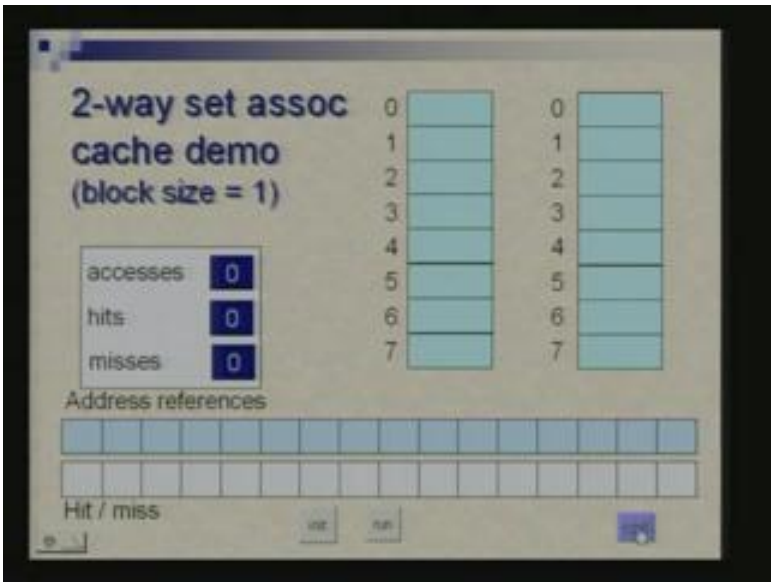
(Refer Slide Time: 00:12:45)



So earlier we had hit for the last 5, 9 and 17; 9 we have lost, 5 and 17 are still there, the additional ones have come here, here, here and here (Refer Slide Time: 13:16) at these four positions you have additional hits. So it is not that whatever is the hit remains a hit

because the total number of blocks is reduced so you can see some effect of increasing the block size too much so some of the hits maybe lost whereas other maybe added and one has to see it carefully. If you recall I had shown you a variation of miss rate versus a block size so initially the miss rate decreases so that is improvement in performance but if you increase the block size too much you start losing and miss rate may increase again.

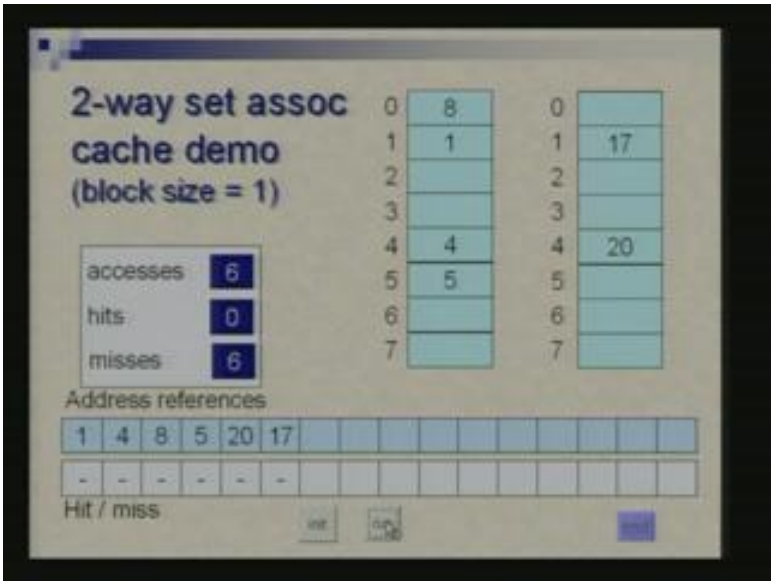
(Refer Slide Time: 00:14:05)



Now let us look at associativity which gives you flexibility of placing blocks at more than one location. And here one has to decide what is the replacement policy because each block can be placed at more than one location; when it comes to replacing an old block you need to see which one needs to be replaced. So in this example we will follow LRU policy. So the block which is least recently used will get replaced whenever there is a need. So what I have done is I have repeated those addresses from 0 to 7 here; effectively each row for example this 3 and this 3 they form offset (Refer Slide Time: 14:53) I will take the addresses now modulo 8; whatever address comes you take it modulo 8 and try to place it in a particular row.

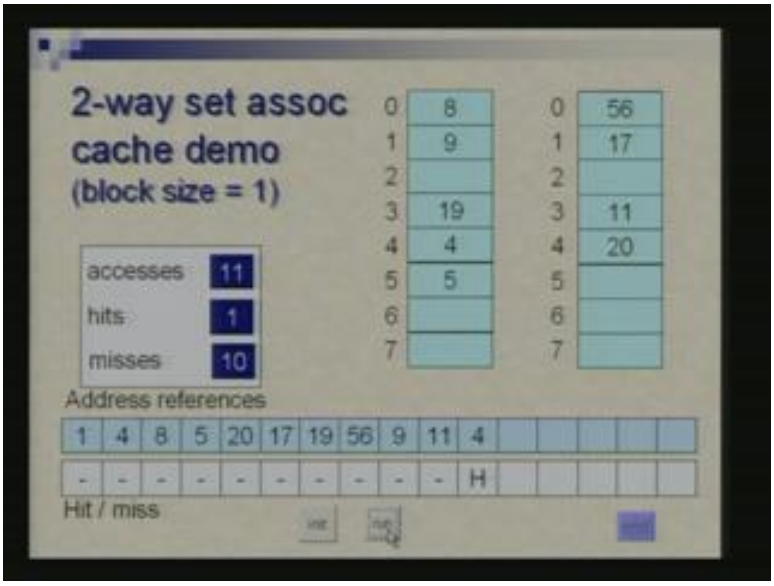
For example, if address modulo 8 is 3, I will see if this is free or that is free it can be replaced in one of these; if both are filled up both are occupied then one has to be replaced and that will be done by LRU policy. So there are eight sets and there is a degree of associativity which is 2. Block size is still 1 so initially everything is empty I place this one here, next is 4 it gets placed there, then 8 is placed at this location 0, the location for 8 was this earlier but now of course both are location 0, 5 comes here, 20 does not throw 4 away now so this is a change which you must observe 20 can be placed here 20 modulo 8 is four and there is a vacancy in this set so 20 can be placed here (Refer Slide Time: 16:17).

(Refer Slide Time: 00:16:21)



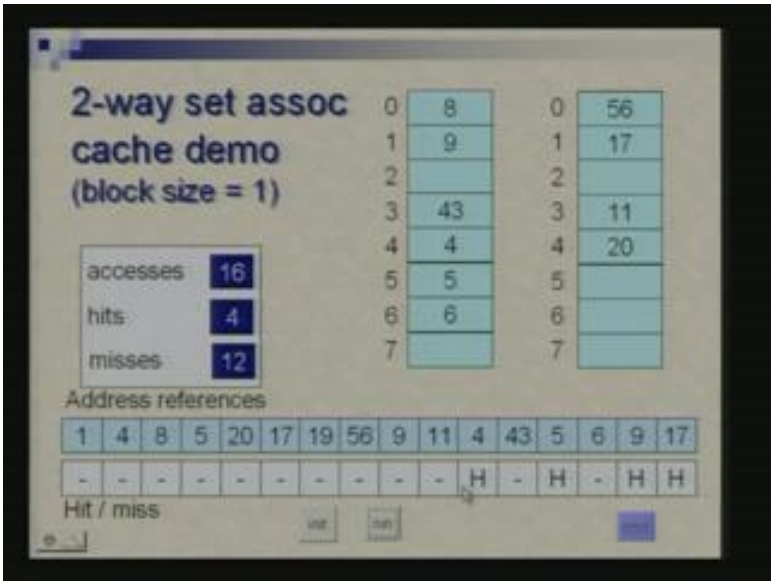
17 does not throw away 1 it has another place, 19 comes here, 56 again does not replace 8, 9 has replaced 1 because now we were falling short of space here and out of 1 and 17 which one should have been replaced that you can see so you either have to replace 17 or 1 but this is more recently used that is less recently used so 1 gets replaced, 11, 4 again so 4 was not thrown so here is a new hit we have got 43, 5 5 is again a hit as was the case earlier 6, 9 and 17.

(Refer Slide Time: 00:16:59)

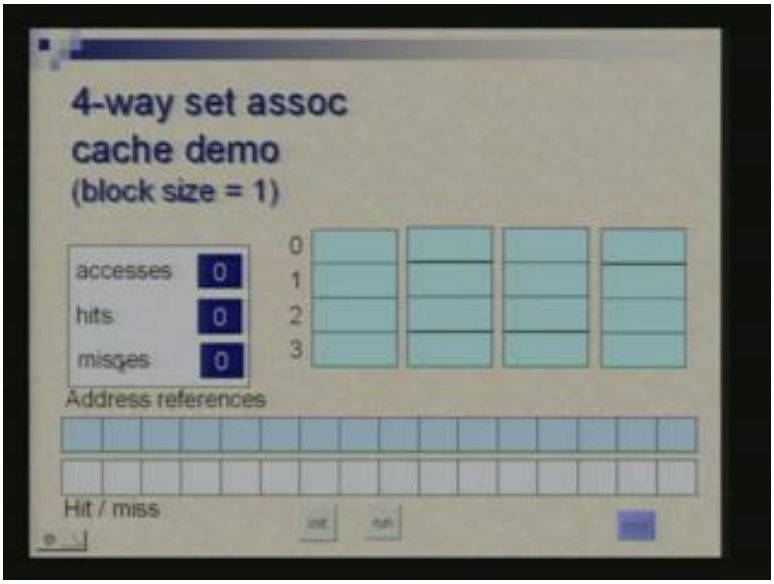


So, if you compare it with the first case we have marginal improvement and the new hit which you can see has come up here (Refer Slide Time: 17:29).

(Refer Slide Time: 17:33)

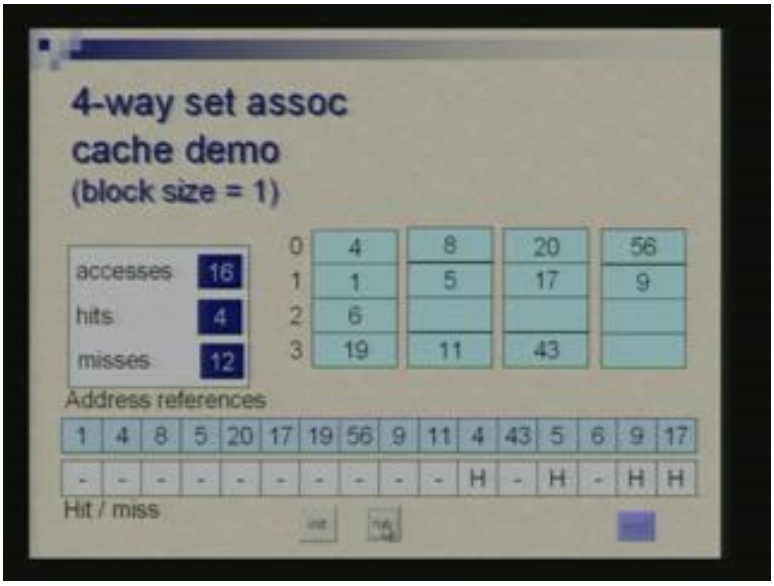


(Refer Slide Time: 00:17:45)



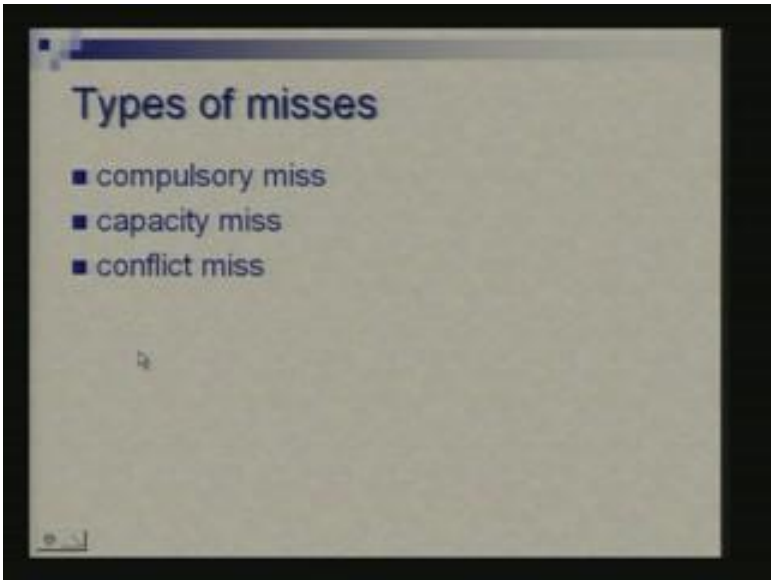
Now let us extend the degree of associativity further. Let us make it 4-way set associativity. The 16 word cache is divided into four sets each with having four locations. So we will do it in similar manner. Notice that the sequence of addresses you are following is same so that we have a fair comparison and in this if I remember correctly nothing will be thrown away there is enough room for everything. So, 4 is a hit, 5 is a hit, 6 is a miss, 9 is a hit and 17 is a hit.

(Refer Slide Time: 00:19:11)



We still capture the same number of hits and what one could say is that this is a kind of situation point as far as this scenario is concerned; by increasing the degree further we are not really going to get anything better. So now after having gone through this exercise we would like to look at different types of misses.

(Refer Slide Time: 00:19:50)



There are three types of misses: one is compulsory miss. Compulsory misses are those which you encounter initially; initially when you are starting with empty cache there are compulsorily some misses because nothing is there in the cache. So first time anything has to be referred to has to be brought.

Then there are capacity misses. Capacity misses are because the cache has limited capacity, it cannot hold everything which is there in the main memory so something has to be out and any misses which are resulting from that limitation are categorized as capacity misses.

Thirdly, we have conflict misses. Conflict miss come because of specific mapping approach which you have and more the degree of associativity you have, less are the conflict misses. So conflicts are because many main memory location contain for same position in the cache and one word comes and throws something else.

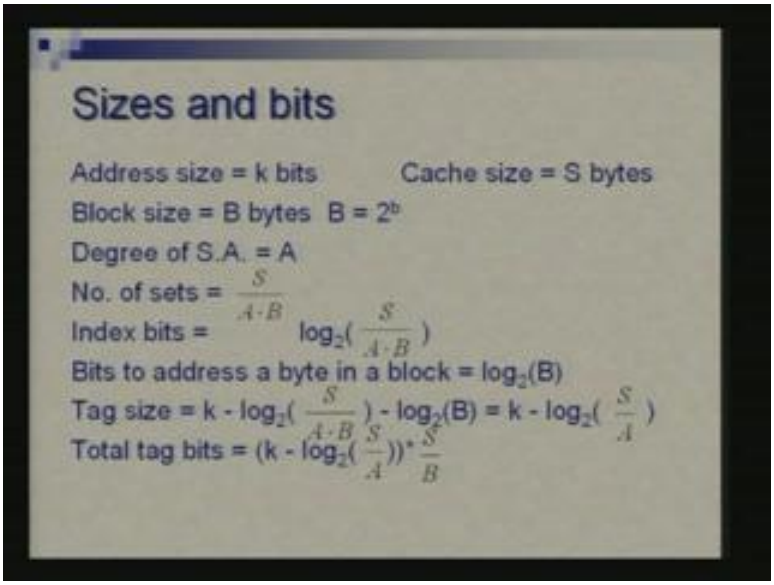
So now if you look at previous examples (Refer Slide Time: 21:16) let us start with this one which of the misses are compulsory misses and which are not.

[Conversation between student and Professor: (21:27)] Yeah, there are lots of compulsory misses, let us find out which misses are not compulsory misses. Wherever we are making second reference that is either due to conflict or limited capacity. So, for example, second reference to this miss is not a compulsory miss because it was something which could have been saved and I suppose everything else is fine, everything else is a compulsory miss.

In the next one you notice that compulsory misses have been reduced here and what has reduced is the larger block size which has reduced compulsory misses here. 5, for example, which is a compulsory miss is no longer a miss at all but we have lost 9 so what kind of miss is that, this is not a compulsory miss.

Now here is a little bit of problem whether you call it capacity miss or conflict miss. Between capacity miss and conflict miss it is hard to pinpoint for individual miss whether it is conflict or capacity. This notion is actually at a statistical level. You look at the overall miss rate let us say 1 percent misses are there 99 percent hits are there so that 1 could be broken up into let us say 0.4 could be a compulsory miss, and 0.3 could be a capacity miss and 0.3 could be conflict misses. The way you precisely define, capacity miss is that you take fully associative cache which means there are no conflict misses. So whatever you are getting over and above compulsory misses is capacity misses. So fully associative cache has only compulsory misses and capacity misses. In a cache which is not fully associative the additional misses you get in terms of average in terms of the total gross behavior they are the conflict misses because they can be attributed to conflict. Otherwise individually it is a bit of problem to categorize a miss as a capacity miss or a conflict miss.

(Refer Slide Time: 00:24:35)



Now let us move to the issue of sizes and bits, you have cache of certain size certain organization how many bits you have to incur as overheads for storing the tags, it depends upon various parameters. For example, suppose the memory address space is addressed by k bit addresses so in MIPS for example this will be 32 and suppose cache size is S bytes now we are assuming that cache is or the memory is byte addressable and let S be the size of the cache in bytes, let B be the size of block so it is B bytes, B would typically be some power of 2 and let A denote the degree of associativity. So A equal to 1 means it is direct mapping cache and when A gets its maximum value then it is fully associative.

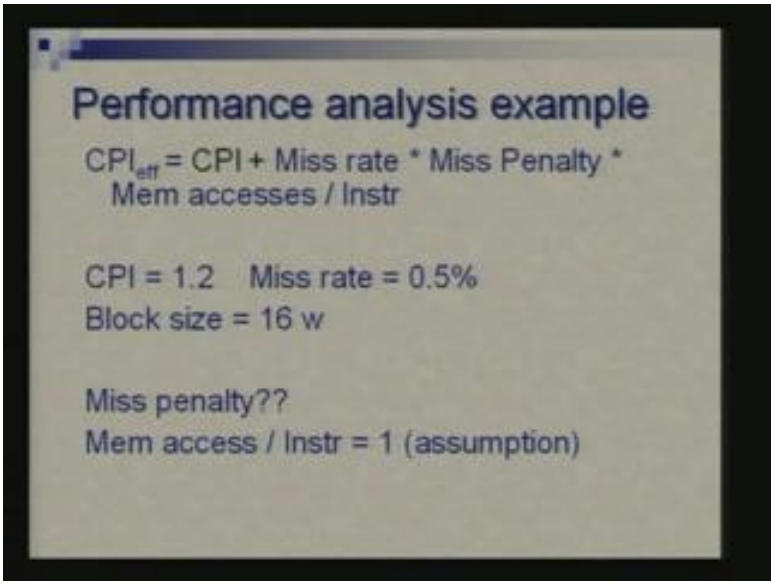
So how many sets you have?
S is the total cache size and B is the block size so S upon B actually gives you the number of blocks, you divide that by A you get the number of sets. So, in a 2-way set associative the total number of sets is half the number of blocks and so on.

How many bits you require to index; because the address is divided into tag part, index part and then there is an offset within the block to access a byte or a word within a block. So, number of bits required to access one set is naturally log of this (Refer Slide Time: 26:46) and number of bits required to address a byte within a block is log of B. So the address which is of k bits if you remove if you subtract the number of index bits and number of bits required to address a byte the remaining are the tag bits. So k minus log of S by A B minus log of B which you can write as: k minus log of S upon A because this B will cancel with that B. So these many bits are required to define each tag and you can see that as A increases the tag size will increase; the total number of tag bits equal to the tag size multiplied by the number of tags and number of tags equal to the number of blocks which we said was S upon B. So this way you can actually calculate the overhead of cache in terms of the tag bits.

So actually all these tags together are what is called the cache directory, so cache has two parts: cache directory and the cache data.

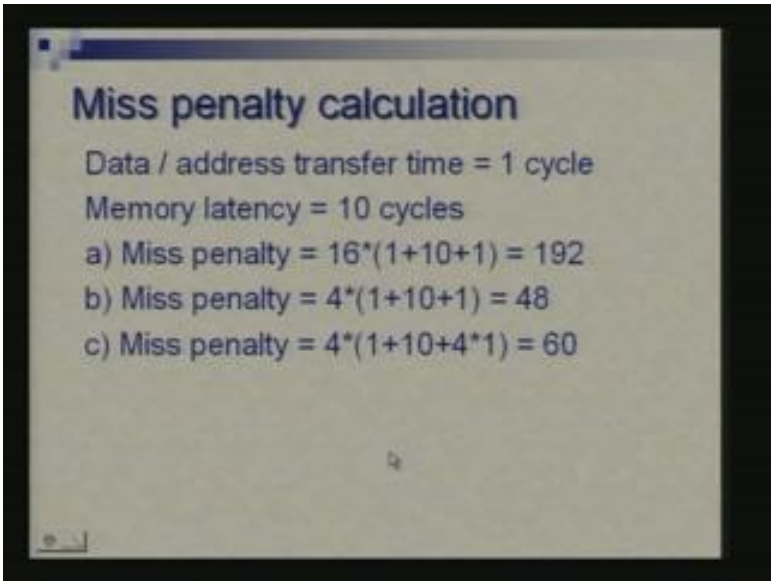
Now let us take some numerical example and try to relate various performance parameters to each other. We have seen that effective CPI in presence of a cache is the CPI assuming that there were no cache misses it was an ideal situation plus the additional cycles which are coming because of misses so these are stalled cycles they depend upon the miss rate, miss penalty that means additional cycles encountered, additional stall cycles are encountered per miss and the third factor is number of memory access per instruction. So the product of this gives you the total number of stalled cycles per instruction.

(Refer Slide Time: 00:29:05)



Suppose in a given case we have the basic CPI as 1.2, miss rate is 0.5 percent and block size is 16 word, how do we find the miss penalty?
We will need to find miss penalty to get this, so how do we find miss penalty, that we will come to, in a moment. For this example we assume that number of memory access per instruction is 1.

(Refer Slide Time: 00:29:48)

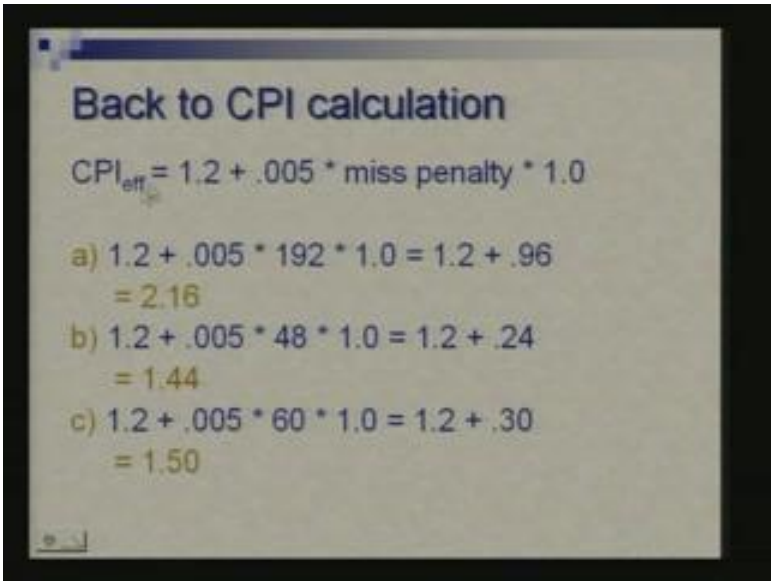


The miss penalty will depend upon what kind of memory cache interface you have and we have talked of three interfaces if you recall; in one case we have memory which is 1 word wide and you access one word at a time transfer one word at a time. Second case was that you have memory which is multiple worldwide and multiple words can flow on the bus simultaneously, lastly we have interleaved memory or a page DRAM. So, suppose it takes 1 cycle to transfer either data or address over the bus, so effectively assuming that bus cycle is same as the CPU cycle assume that memory latency is ten cycles that means once address is given memory has a delay of 10 cycles and after 10 cycles it will respond with the data. So, given this we can find out miss penalty for case A where one word is accessed at any time so for each word you spend one cycle sending the address, 10 cycles taken by the memory, 1 cycle again by the bus to send the data back so total 12 cycles and for a 16 word block this process will be repeated sixteen times and a total of miss penalty is 192.

In case B, let us assume that the memory is 4 word wide and the bus is also 4 word wide. So that means this cycle of 1, 10 and 1 this needs to be repeated only four times and therefore we encounter 48 cycles.

In the last case, suppose we assume that there is 4-way interleaving but the bus is 1 word wide that means in one transaction we will send address take 1 cycle then memory takes 10 cycles; actually they are four modules, each one will be ready with its data and 4 cycles will be required send it back to the CPU. So total of 15 cycles are required to get 4 words and that is repeated four times to get a block of 16 words so that gives you a total miss penalty of 60. So we will get three different answers for CPI for these three cases.

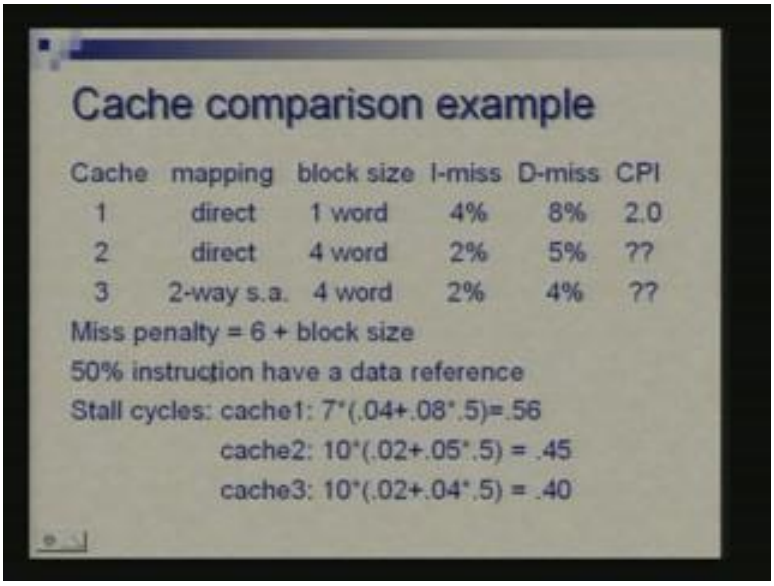
(Refer Slide Time: 00:32:21)



So, coming back to CPI calculation we had effective CPI as 1.2 into 0.5 percent was the miss rate so we put that as a fraction here, miss penalty we have just calculated we will substitute the values and number of memory accesses per instruction is 1. So, putting the value of 192 as miss penalty we get CPI effective as 2.16.

In the second case, miss penalty is 48, CPI comes out as 1.44; third case, miss penalty is 60, CPI comes as 1.5. You would notice that if you look at miss penalty there is so much variation but the factor by which performance varies on the whole is not the same. It is a variation from, let us say 1.44 to 2.16 as you change the miss penalty from 48 to 192. So ultimately, in terms of CPI this is what matters.

(Refer Slide Time: 00:33:30)



Now take another example where we try to do comparison of different architectures. Suppose we have three different caches with different organization, first two are direct mapped cache and then the third one is set associative cache with 2-way associative access, the block size is 1 word in the first case and 4 word in the second and third case. Here we have given instruction misses and data misses separately. That means out of hundred instructions we try to access, four times we get misses, in the first case. Similarly, we make hundred accesses to the data and there are 8 misses.

Therefore, we are given CPI for the first case; we need to find CPI for the remaining cases. Miss penalty, as we know, depends upon the block size so without going into details, let us say that miss penalty is given by 6 plus block size this is in terms of number of cycles. Also we need information about how many instructions make reference to data. So each instruction makes one reference for fetching the instruction itself and here 50 percent of the instruction make data reference so they are either load or store. So now we can calculate the number of stalled cycles; this is the miss penalty in the first case since the word since the block size is 1-word this is 7 (Refer Slide Time: 35:22) multiplied by the miss rate. So, for instruction it is 0.04, for data it is 0.08 but only half the instruction makes data access so multiply by half here.

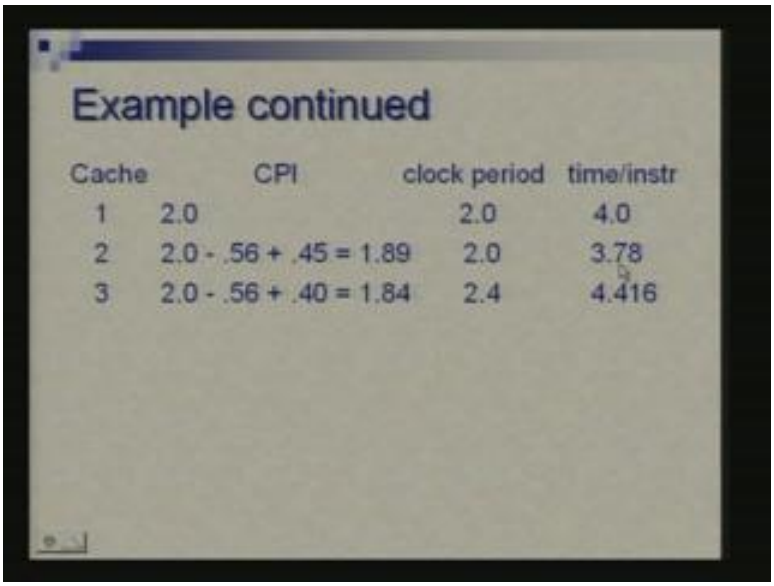
What this says is that in cache 1 on the average there are 0.56 stalled cycles per instruction.

Going to cache 2 the miss penalty is 10 because block size is 4. I miss 0.02, D miss 0.05 as given and because half the instructions refer to data we have factor of 1/2 here and this comes out to be 0.45.

In the third case again similar but this factor is reduced. So these are the extra values which have to be added over basic CPI to get the overall CPI. What we have given here is

effective CPI 2.0. So basic CPI can be obtained by looking at this. Out of 2.0 0.56 is because of stalls, cache stalls and the rest is the basic CPI which can be assumed to be same for these cases.

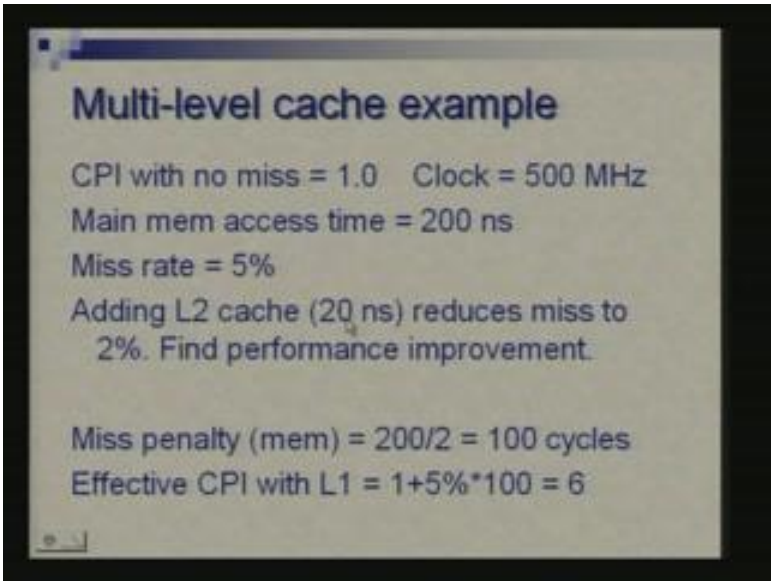
(Refer Slide Time: 00:36:46)



So CPI for three cases it is 2.0 here, for the second case 2.0 minus 0.56 plus 0.45 and similarly for the third one. So the figure we are getting is 1.89 and 1.84. Now this is all in terms of number of cycles. But as you know, we need to take into account the clock cycle, the clock period also. But typically more the complex structure you have like you have associative cache the cycle time is likely to be larger because the hardware has to allow multiple comparisons at the same time so it may be slower and it may pull down the clock period. So the figure may be little bit exaggerated here but assume that for direct mapped caches, clock period is 2 nanoseconds and for the set associative case it is 2.4 nanoseconds.

So, if you multiply the CPI figure with this figure you will get the time spent per instruction and this is the overall indicator of the performance taking into account all the aspects. So, looking at that it is the second one which seems to be the best. Of course if the increase here (Refer Slide Time: 38:15) this increase is not so much, possibly this could have done better. So we have to see that, by making it associative by what amount is this miss getting reduced. If the advantage you are gaining here is not substantial it may be lost by the negative effect you will have on the clock period so that has to be kept in mind.

(Refer Slide Time: 00:38:21)



Now this example looks at multi-level cache. Suppose you have two level caches what we will do is first we will analyze situation with one cache and see what improvement the second cache brings to the overall performance. So, suppose we have processor with CPI of 1 when there is no miss and it is running at clock of 500 megahertz, the main memory access time is 200 nanoseconds and the miss rate is 5 percent. So if suppose there was only a single cache we are saying that the miss rate is 5 percent. It is observed that by adding L2 cache second level cache there is a reduction in the miss rate.

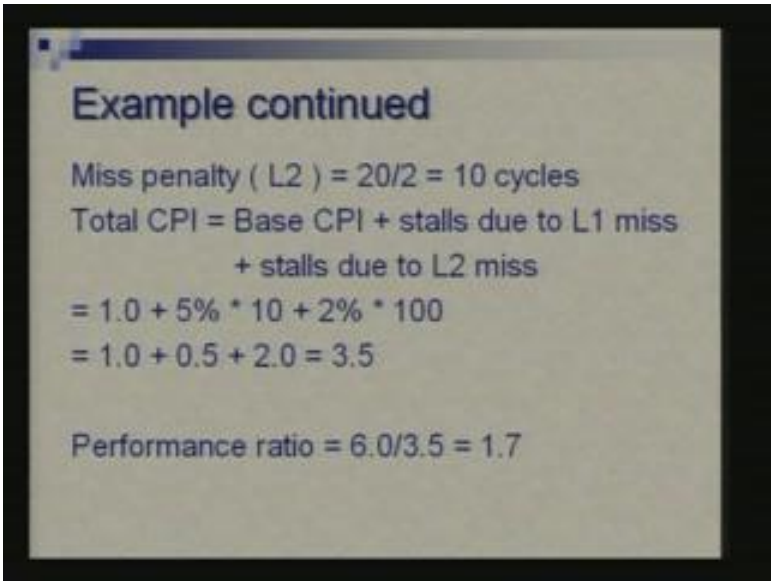
Now, suppose the miss rate becomes 2 percent only it drops from 5 to 2 what is the performance improvement what is the net effect on the CPI. So we need access time data about the L2 cache suppose that is 20 nanoseconds, main memory is 200 nanoseconds and L2 is 20 nanoseconds and the access time of L1 cache is subsumed in this, (Refer Slide Time: 40:12) this is when there are no misses.

So, miss penalty when one as to access memory is 200 nanoseconds, to convert this into cycles we divided by time period of this which is 2 nanoseconds. So 200 by 2 or 100 cycles is a miss penalty for making access to main memory.

We are not going into details of how we get this because ultimately it depends upon the bus architecture, block size and so on. But whatever all that is the net result is let us say 200 nanoseconds.

So, if you have just L1 cache then what is your effective CPI? it is 1 which is this figure (Refer Slide Time: 40:57) plus 5 percent that is a miss rate multiplied by 100 cycles which is the penalty. So this comes out to be 6. Once again we have made an assumption here that number of memory access per instruction is 1. There are no data accesses or everything is just 1.

(Refer Slide Time: 00:41:26)



Suppose we introduce L2 the miss penalty of L2 itself is 20 nanoseconds divided by 2 nanoseconds which means 10 cycles. So now the total CPI, taking into account misses at various levels will be the base CPI plus stalls due to misses at L1 level. So when there is miss at L1 you make access to L2 so there are some stalled cycles encountered, if you have miss at L2 level also there are some additional cycles you incur. So this will have to be 10 multiplied by the miss rate here this is 100 multiplied by the miss rate here. So 1.0 the base CPI, 5 percent miss at L1 level multiplied by 10 which is the miss penalty and there it is also multiplied by 1, we assume that one memory reference per instruction otherwise that factor will come here plus 2 percent is a miss at L2 level multiplied by 100. So now this first term gives us 0.5, second term gives us 2.0 and the total is 3.5.

So what is the ratio; by what factor has performance improved?
You can simply divide CPI which we had earlier 6.0 and the CPI you got now 3.5 to 1.7. So now let me clarify certain terminology here. we are saying that when you put L2 there is a 2 percent miss on the whole so this is what is called, in a multi-level cache terminology this 2 percent will be considered as global miss, that is all caches put together, that has a system it is taking care of 98 percent of the accesses and it is leaving out only two.

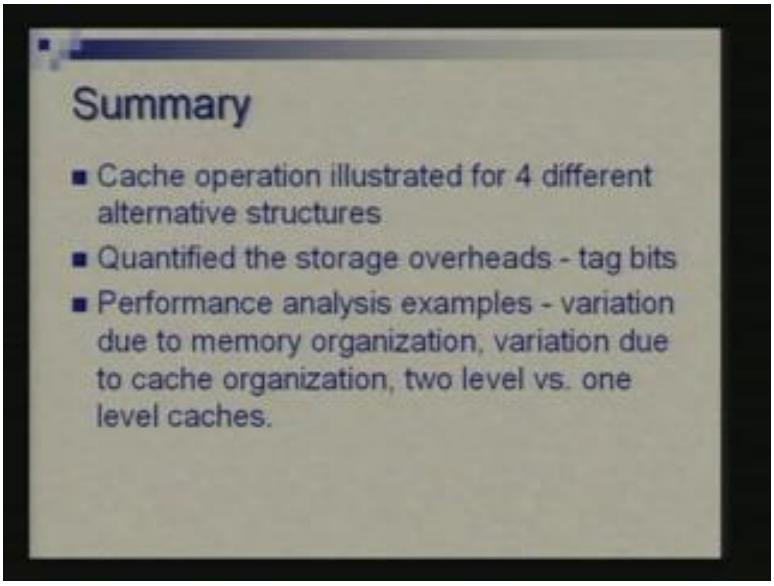
But if you were to focus your attention on L2 cache only, L2 is receiving some accesses, it is receiving some requests, some of them it is able to serve some it is losing so there is something called local miss for L2 that is not shown up explicitly here. There is also a term called solo miss. solo miss is a term..... suppose now in this two level cache system you remove L1 and L2 is the only one left then the misses which you will see at that time are called the solo miss of L2 solo means when L2 is alone. So is there some relationship between these? In a specific case there is a very nice relationship between these. When the data in L2 is all inclusive of data in L1. So, if you have a situation that anything which is there in L1 is there in L2 but L2 has something more. Because L2 will be

typically of much larger size than L1. This property is called inclusion property; it may hold, it may not hold.

We will come to the point why it may not hold. But suppose it holds then the solo miss of L2 will turn out to be same as the global miss which we are seeing here. What it means is that if L1 and L2 together are showing you a 2 percent miss having put L2 alone would also have shown you 2 percent miss then what purpose L1 is serving?
L1 is reducing the time because you know L1 is taking care of some request and reducing the penalty you are incurring. And also, you can relate now the global miss also to the local miss. You can look upon it like this that suppose hundred requests are made by the processor, 95 are taken care by L1, only 5 are going L2, out of those 5 effectively L2 is taking care of 3 and losing out on 2 so what is the local miss of L2? Local miss is 40 percent, local miss is 40 percent so it looks a very large figure but it is because only selected requests are coming to L2 and it is able to serve some of those and even by doing that it is reducing the overall miss rate.

Now, coming back to the inclusion property the inclusion property may not necessarily hold. Why? because it may happen that L1 may retain some data which L2 throws off because of its own structure and replacement policy so L1 and L2 could be structured totally independently, the degree of associativity could be different, the block size would typically be same but degree of associativity is typically different and other policies like write through, write back they could also be different. So there is a possibility that something which was brought to L1 maybe at that time it was there in L2 but eventually by the time you have second reference to that L2 has meanwhile thrown it so those situations could arise and inclusion property may not necessarily hold.

(Refer Slide Time: 00:47:28)



So let us conclude by summarizing what we have seen today. We tried to simulate cache for a few simple situations and idea was to focus attention on individual hits and misses

to see which factor is affecting what. Then we moved on to quantify the storage overheads in a cache organization, counted the tag bits and the number of tags and so on. Finally we took some numerical example to get a better insight into what factors influence the performance in a statistical sense; in a gross sense what are the miss rates, how do they affect the stalled cycles and how ultimately affect the CPI. We have seen that what are the variations in these parameters due to change in the cache organization and as you go from one level, two level how things change. I will stop at that, thank you.