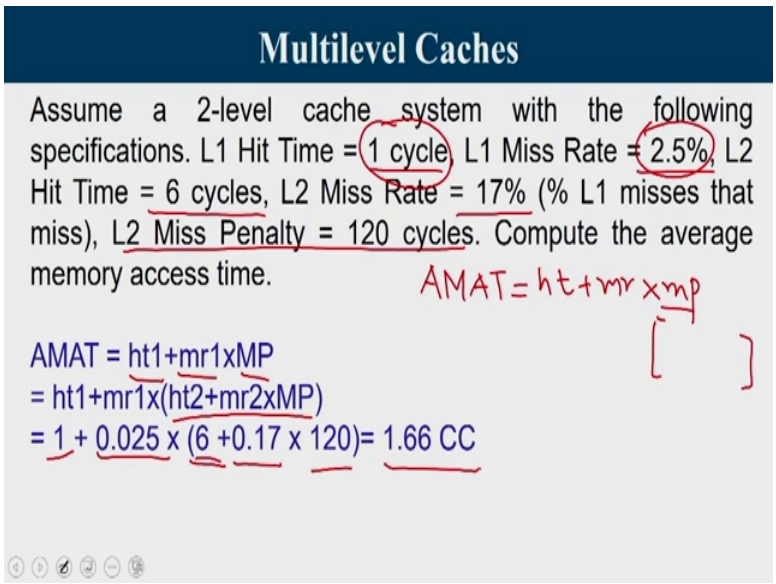
Advanced Computer Architecture
Dr. John Jose
Department of Computer Science & Engineering
Indian Institute of Technology Guwahati

Tutorial No. – 7
Optimization Techniques in Cache Memory

Welcome to tutorial number 7 on optimization on cache memory. In the lecture videos that you have seen in this week, our discussions were on how to optimize cache memory in order to reduce average memory access time. Different techniques were explored such that you can reduce hit time, miss rate, and miss penalty which will help us in improving average memory access time. We will work out a few problems related to cache optimization in today's tutorial.

(Refer Slide Time: 01:02)



So, assume a 2-level cache system with the following specifications. L1 hit time makes one cycle, L1 miss rate is 2.5%, the hit time of L2 cache is 6 cycles and with a 17% of miss rate. This 17% is defined as percentage L1 misses that missed in L2. Now, miss penalty of L2 is 120 cycles. Compute average memory access time. We know that average memory access time is defined as hit time plus miss rate into miss penalty.

AMAT= hit time + miss rate * miss penalty

Now, this miss penalty of L1 again should be defined as hit time of L2 plus miss rate of L2 into miss penalty of L2.

Miss penalty (L1)= hit time (L1) + miss rate (L2) * miss penalty(L2)

So, average memory access time is defined as hit time of L1 cache plus miss rate of L1 cache into miss penalty.

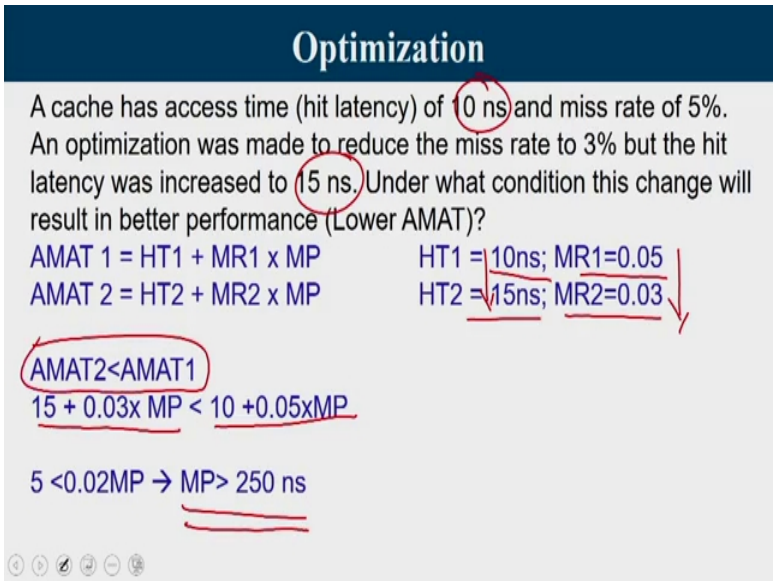
AMAT= hit time(L1) + miss rate(L1) * miss penalty

This miss penalty is itself again going into one more iteration that is hit time of L2 plus miss rate of L2 into miss penalty. Since all the values are given; the hit time of L1 is given with one cycle, so that is 1; miss rate of L1, that is 2.5%. So, here the value is 0.025.

That is the miss rate of L1 cache plus the hit time is 6 cycles for L2 cache with a 17% miss rate, so 0.17 into 120 clock cycles, that is equal to 1.66 clock cycles is the average memory access time.

AMAT= 1 + 0.025 * (6 + 0.17 * 120) = 1.66 cc

(Refer Slide Time: 02:54)



Moving into the next question. It is an optimization problem. A cache has access time, that is, hit latency of 10 ns and a miss rate of 5%. An optimization was made to reduce the miss rate to 3%. So, from 5% you are trying to reduce the miss rate to 3% but the hit latency was increased. So, on this optimization we are able to reduce the miss rate, but the byproduct of which is that there is an increase in the hit time from 10 ns to 15 ns.

Under what condition this change will result in better performance or lower memory access time. So, in this particular problem, we are talking about a cache memory with some features of hit time and miss rate and trying to make some optimizations that we have seen in a few of the lecture videos like increasing block size, then increasing cache size, then associativity, so different ways by which we could reduce you miss rate.

But some of these optimizations you know that it will impact some other cache parameter like the hit time. This is the classical example wherein the hit time is increased. So, we want to see whether such a kind of an optimization is useful or not, and the term usefulness is subjective. Here, in this context, we are considering average memory access time as one of the parameter.

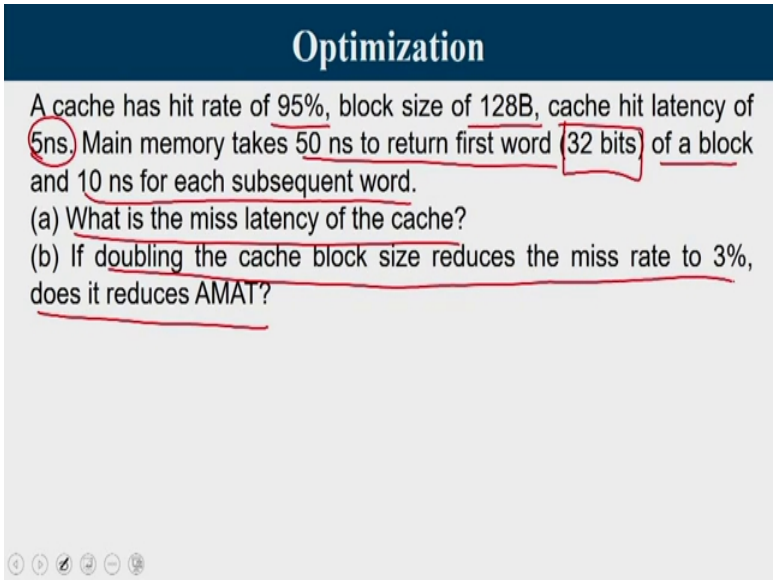
So, with this optimization can we reduce average memory access time? So, average memory access time of the first case is defined as hit time of the cache plus miss rate into miss penalty. So, here it is defined as hit time is 10 ns and miss rate is 0.05 and the average memory access time of the second one is defined as hit time plus miss rate into miss penalty where he time is increased from 10 ns all the way to 15 ns. So, that is the increase here.

The miss rate comes from 5% to 3%. So, what we are trying to do here is average memory access time of the second one, that is the optimized one should be less than average memory access time of the first case, then only we can say that this optimization has resulted in a better performance. So, 15 plus 0.03 into miss penalty, that is the average memory access time of the second one should be less than that of the first, 10 plus 0.05 into miss penalty.

(AMAT2= 15 + 0.03 * miss penalty) < (AMAT1 = 10 + 0.05 * miss penalty)

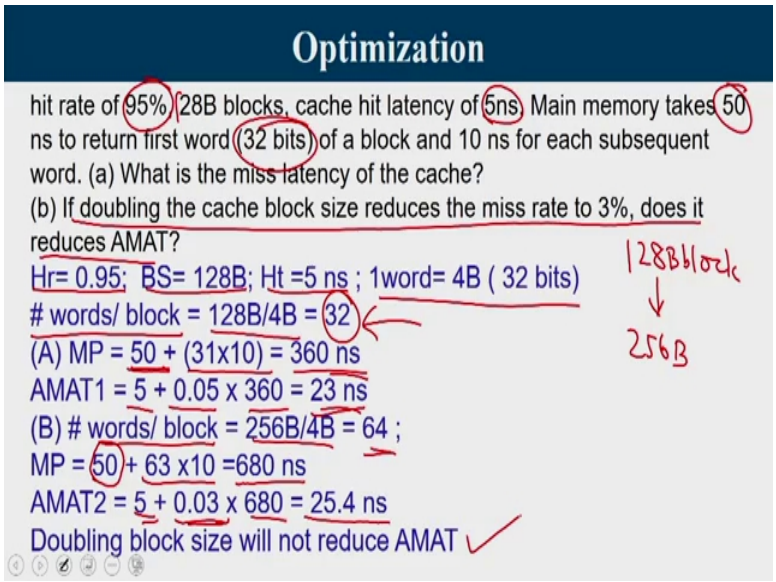
And upon solving you get that the miss penalty value should be larger than 250 nanoseconds if at all this equation has to hold. So, this optimization will work only in the context when the miss penalty is larger than 250 nanoseconds. So, when the miss penalty is smaller than 250 nanosecond, even though we are able to reduce the miss rate, the product of miss rate into miss penalty is not sufficient enough to overcome whatever increase that you have in the hit time.

(Refer Slide Time: 05:50)



We now move into the next question. A cache has a hit rate of 95%, a block size of 128 byte, cache hit latency of 5 nanoseconds. Main memory takes 15 nanosecond to return the first word. So, here one word is defined as 32 bits. So, it takes 15 nanoseconds to return the first word of a block and 10 nanoseconds for each subsequent word. What is the miss latency of the cache? That is the first question. If doubling the cache block size reduces the miss rate to 3%, does it reduce average memory access time? So, there are two different versions of the question.

(Refer Slide Time: 06:28)



So, the hit rate is defined as 95 and we have 128 byte blocks and cache hit latency of 5 nanoseconds. So, the values have been substituted now. Hit rate of 0.95, block size of 128 B, hit time of 5 ns. One word is defined as 4 B. So, it is 32 bits that has been given in the

question. So, 32 bits is one word. So, the number of words that you can accommodate in a block, words per block is 128 B divided by 4 B, so you get 32 words are there in one block.

No of words in a block = 128 B/ 4 B = 32 words

Now, miss penalty has been defined as, for the first word it takes a lot of time because you have to index into the memory, so it 50 ns to return the first word. So, first word will take 50. We know that there are a total of 32 words out of which the first word can be brought only in 50 ns. The remaining 31 words will take 10 ns each. So, that is going to take total of 360 ns. So, what is the average memory access time? You have hit time with you, that is 5.

Miss rate is given as 0.05, and then miss penalty is 360 that we just computed. So, we will get 23 ns as the average memory access time. Now, the second portion of the question is if doubling the cache block size reduces the miss rate to 3%, does it reduce the average memory access time? Let us imagine that the hit time is unaffected since it is not given in the question.

So, in the second case, the block size previously we had 128 B block, now that is going to be increased to 256 B block. So, the number of words that you have is now 256 B divided by 4 B, so you have 64 words.

256 B / 4 B = 64 words

Now, miss penalty is defined as the first word will take 50 ns, thereafter you have 63 more words, each will take 10 ns each, so totally the miss penalty is 680 ns.

So, average memory access time in the second case, that is, by increasing the block size is going to be 25.4 ns, 5 that is the hit time plus miss rate is 0.03 only, and then 680 is the miss penalty, so it is 25.4 ns.

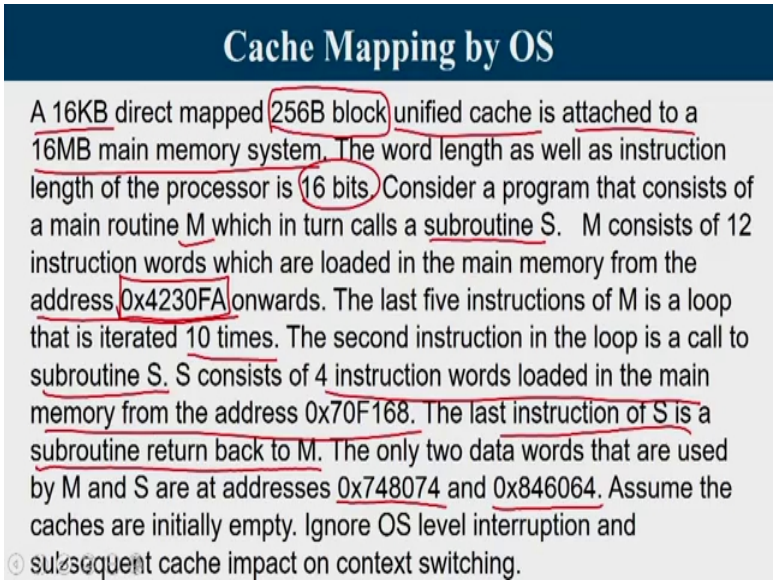
AMAT= 5 + 0.03 * 680 = 25.4 ns

So, doubling the block size will not reduce average memory access time, that is a takeaway that you have. So, this is a classical example of a case wherein the block size is increased.

When you increase the block size, then we are trying to reduce compulsory misses. So, that is reflected in this question, from 5% your miss rate has come down all the way to 3%. But when the block sizes in increased you have to bring multiple words from the next level of memory into cache. That has been reflected as we discussed in this question. The miss penalty value is increasing.

So, the increase in miss penalty is more dominant than the reduction in the miss rate thereby the average memory access time of the second version, that is the cache block doubled version is larger than that of the previous one thereby the optimization is not effective.

(Refer Slide Time: 09:53)



Now, we are moving into a question where cache memory mapping by the operating system is being discussed. A 16 KB direct mapped 256B block unified cache, unified cache means both I and D, both instruction and data are stored in one cache that is called unified cache, is attached to 16MB main memory system. The word length as well as the instruction length of the processor is 16 bits. So each of your instruction, each of your data word is 2 bits. Consider a program that consists of a main routine M which in turn calls a subroutine S.

M consists of 12-instruction words which are loaded in the main memory from the address 0x4230FA onwards. So, you have two subroutines M and S, M is a main protein and this is the subroutine. M is stored from this particular address. The last five instructions of M is a loop that is iterated 10 times. The second instruction in the loop is a call to the subroutine S. S consists of 4-instruction words loaded in the main memory from the address 0x70F168.

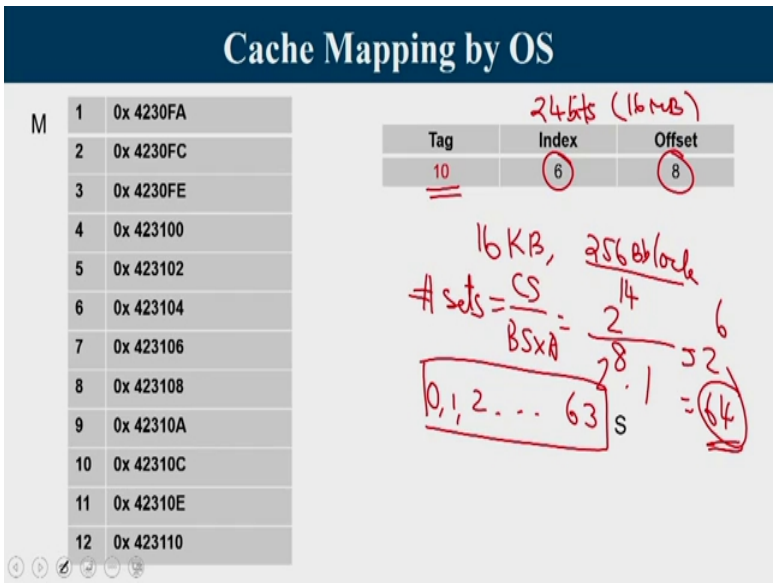
The last instruction of S is a subroutine return back to M. The only two data words that are used by M and S are at addresses 0x748074 and 0x846064. Assume that caches are initially empty and we hope to ignore all OS level interruption and subsequent cache impact on context switching. Let us try to understand what is being given in this question. You have

been defined that you are going to store a main program M and the subroutine S, they are going to interact.

Now the specification of M is being given from which address the M is stored in main memory, and we know that the last five instructions of M are going to be repeated in a loop and the second instruction is a call statement to S. So, M calls S and S is returning back to M. So, it is a main function which in turn call a subroutine and this subroutine call happens 10 times inside the loop.

So, during the execution of the program, since the cache is initially empty, upon demand contents are loaded to cache from the main memory. Both instruction and data are stored in the same cache, it is a unified cache. After the execution is done, we are supposed to tell what are the non-empty sets in the cache?

(Refer Slide Time: 12:28)



So, the first thing is we want to find out what is the specification of the cache. So, we know that it is a 16 KB direct mapped 256 B block. So, it is 16 KB direct mapped cache with 256 B blocks. So, the number of sets in the cache is defined as cache size by block size into associativity. Cache size is 16 KB, so it is 2 power 14 divided by 2 power 8, that is block size into associativity is 1. So, that gives you 2 power 6; and we have 64 sets that is there in the cache, and these sets are numbered from 0, 1, 2, 3, etc., up to 63.

$$2^{14} / (2^{8} * 1) = 2^{6} = 64 \text{ sets}$$

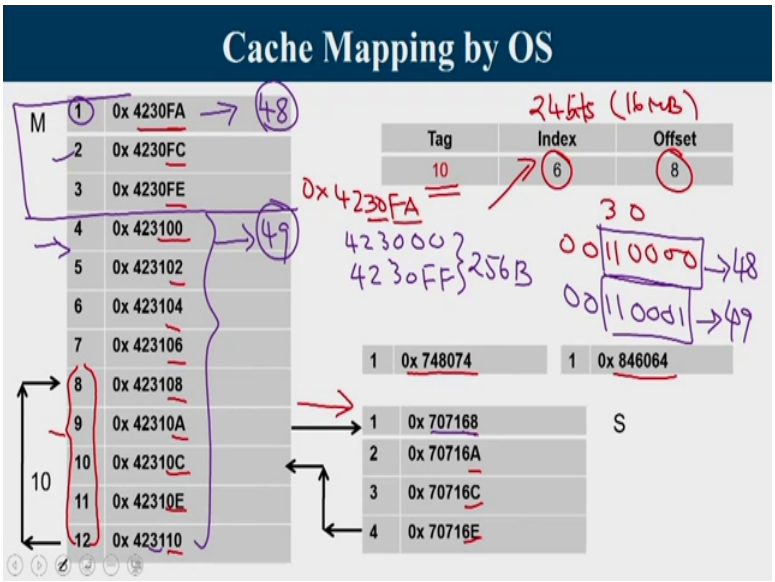
So, whenever I talk about my cache memory, then the sets are set 0, set 1, set 2, etc., up to 63. So, by virtue of it, you know that we are going to have a main memory which require 24 bits,

that is 16 MB main memory, 2 power 24 out of which, since we have 64 sets, 6 bit is gone for index and our block size is 256 B. So, 8 bits are gone for the offset, that makes it a total of 14 and remaining 10 bit is your tag.

$$16 \text{ mb} = 2^{24}$$

So, this is the logic in which we are able to get the split up of the physical address, 24 bits of physical address that is 16 MB of main memory that is given in the question, 24 bits of physical address gets splited up into more significant 10 bits for tag and then 6 bits for index and the last 8 bits for the offset. Now, the next thing we want to know is where this main program M and the subroutine S have been organized.

(Refer Slide Time: 14:36)



So, as given in this question, it was told that the main function is stored in location 4230FA and each of the instruction will take 2 byte. Each instruction as well as data word is 2 byte. So, 4230FA and 4230FB, these are the two bytes in which the first instruction is located or first instruction is rather spread across 2 memory bytes, 0X4230FA and FB. So, at FC we have the next instruction.

So, the instruction is stored in continuous locations, so FA, FC, FE. Once FE is over, the very next one is 0x423100, then 02, 04, 06, 08, 0A, 0C, 0E, and then we have 10. So, these are the 12 instructions for the main program M. Similarly, your subroutine is being stored in 707168. So, 68, 6A, 6C, and 6E, each instruction will take 2 bytes. So, these are the bytes in which these instructions are starting.

So, the four instructions of subroutines S are kept like this. Now, we have to understand what is a program flow. It is given in the question that the last five lines of M is part of a loop. So, from M8, M9, M10, M11, and M12, that is being shown here, this is going to be iterated 10 times, and the second line of this loop that is M9 is a function call to subroutine S and the last line of subroutine is a function return back to M.
So, in a subroutine when M9 calls the subroutine S, the return statement from S will trigger M10, that is the balance that is happening. And the two data items that have been referred as 0x748074, 846064. So, now we got a complete picture of what are the addresses that are needed in order to execute this M and S along with their data. Now, if you take this address 0x4230FA, we know that the last 8 bits which is represented by FA, hexadecimal FA, that is the last eight bit they belong to offset.

Out of these 30, even though this 30 is 8 bits, the 6 bits will tell you the index. So, when I write 30, 3 corresponds to 0011 and 0 corresponds to 0000. So, these are the 8 bits that is there. You extract the 6 bits, this will tell you the value is 48. So, this particular location that is 0x30FA, that is address in the main memory, upon bringing to an empty cache it will be kept in set number 48.

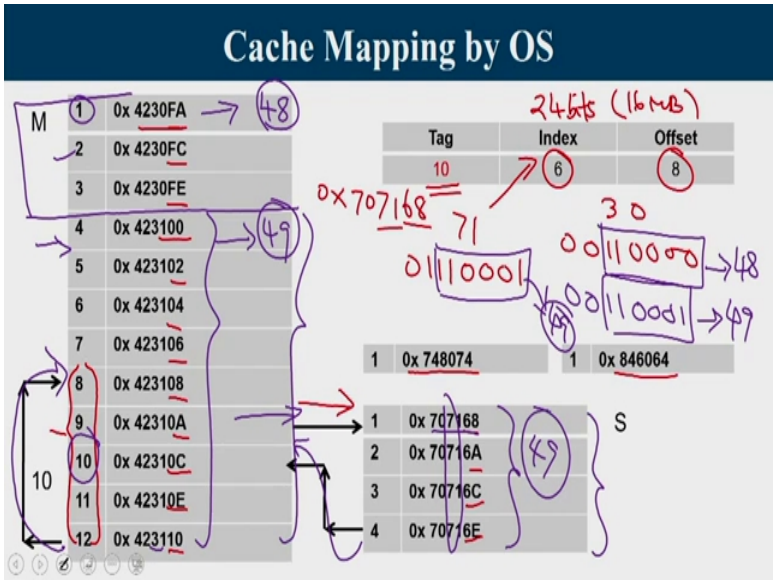
If you look at the next one also, so when you bring set number 48 everything with the offset value, the last value that is 423000 and 4230FF these constitute of 256 B, that much amount of data is being brought together to set number 48. So, in that process, M1, M2, and M3, these three will have already been brought. So, you encounter a miss upon fetching M1, but M2 and M3 are automatically brought to the cache during the miss of M1, because they are part of the same block.

But when you go to M4, it is actually 31. So, 31 means it will be 00110001, that indicates these 6 bits will point to set number 49. So, this is going to be set number 49 and all others, whereas you have this 31, this whole set will be part of set number 49. So, on M4 we have one more miss that will result in bringing of the contents of 423100 to 4231FF, 256 B are being brought. In the 256 B M4 to M12 is completely contained.

So, they will be brought to set number 49. But then, when you are fetching and executing M9, in the meantime, control is transferred to 707168. That is the address that is going to be

worked out at this point. Let us try to work it out and then see to which address or to which cache memory set this particular address is mapping.

(Refer Slide Time: 19:36)

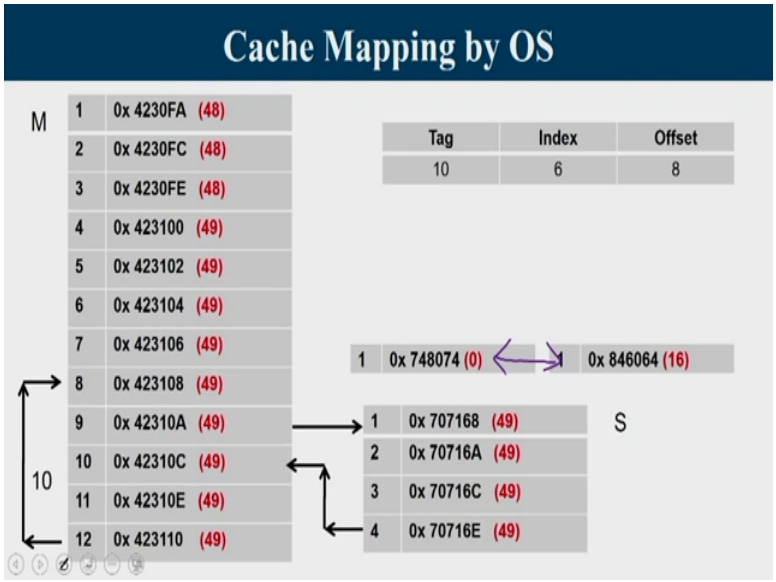


So, 0x707168, so this is the offset portion, so you expand 71. 7 stands for 0111 and 1 is 0001. So, if you extract the 6 bits off it, that is actually pointing to again set number 49. This is the interesting observation that you see from this question. When you are going to bring S they all are part of that 71, so they all map to set number 49. It is a direct mapped cache. So, M4 up to M12, whereas the complete S they both are mapped to set number 49.

So, M4 to M12 and S cannot coexist together. It is a direct mapped cache where part of the main routine as well as the subroutine they are mapped into the same set because of which when the main routine is existing in the cache then the subroutine is being evicted out. When you transfer the control back to the main routine, that is when you wanted to execute the M10, then I will evict out the subroutine.

Like that, when the control again is passing on to S, that is when the loop is repeated from M12 to M8, and in M9 we have one more control, at that time S is not present so it is again a miss. So, that is going to evict out M and then it is going to bring back S. So, in this way, there is a mutual eviction because of the address peculiarity.

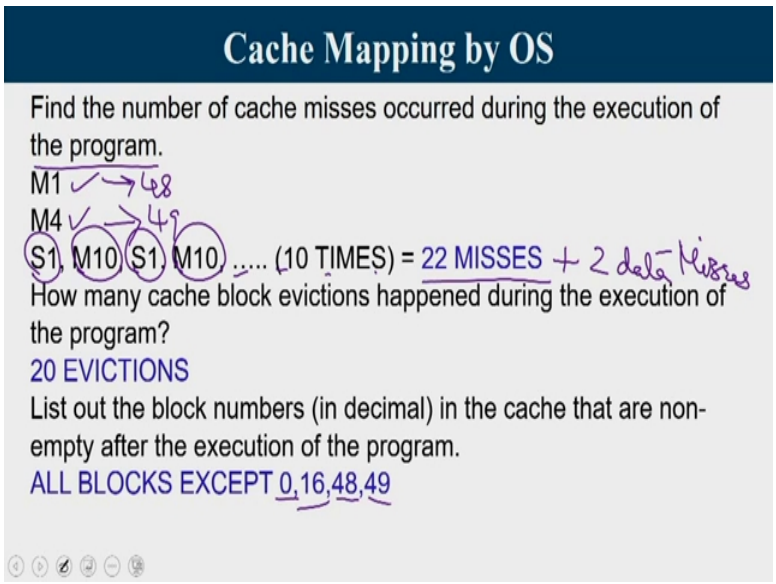
(Refer Slide Time: 21:11)



So, based upon that, we can find that wherever you see that red color they are telling which is the set number to which these addresses are mapped. So, you have 48 and 49 that is part of M, whereas S is mapped to set number 49 only and two data elements, the first one is mapped to set 0 and second one is mapped to set 16. So, both the data elements there is no conflict.

There both will be residing in this cache from the beginning. So, accept the first two compulsory misses, like when you encounter this data items for the very first time there is a compulsory miss and here after that nobody is going to pull out.

(Refer Slide Time: 21:49)



So, find the number of cache miss occurred during the execution of the program. So, if you look at the execution of the program, you have a miss in M1 that is going to be fetched to set number 48, and then you have a miss in M4, that means the very first access happened to 49.

And then there is S1, whenever you have S1, then that is again a miss even though it is a compulsory miss, but it has to evict out whatever was kept by M4, and then when it comes back to the main routine, again M10 is another miss.

So, every time you are going to run, so this is going to be repeated 10 times. So, there are 22 misses plus there are two data misses that are also happening. There are two data items they are also missing once. How many cache block evictions happen during the execution of the program. So, basically already there is a data and I going to evict out. So, whenever I miss in S1 I evict the contents that has been brought from M4 like that. For this loop you have eviction. So, you have total of 20 evictions that is there.

List out the block numbers in decimal in the cache that are non-empty after the execution of this program. So, block number 0 and block number 16, these are the set numbers. Set 10 block is same in the case of direct mapped cache. So, set number 0 and set number 16 will have the data and set number 48 and 49 will carry your main routine M and the subroutine S. So, this gives us a clear idea about what will happen whenever you run multiple programs.

So, with that, we come to the end of this tutorial on cache optimizations and OS role in mapping. So, just to summarize, especially in the last two problems, once you work out problems like this where multiple functions are being involved, one function calling other and there can be mapping issues, when two functions are mapped on to the same cache block, then there will be evictions. So, that will give you a concrete idea about how cache memory mapping happens. Thank you.