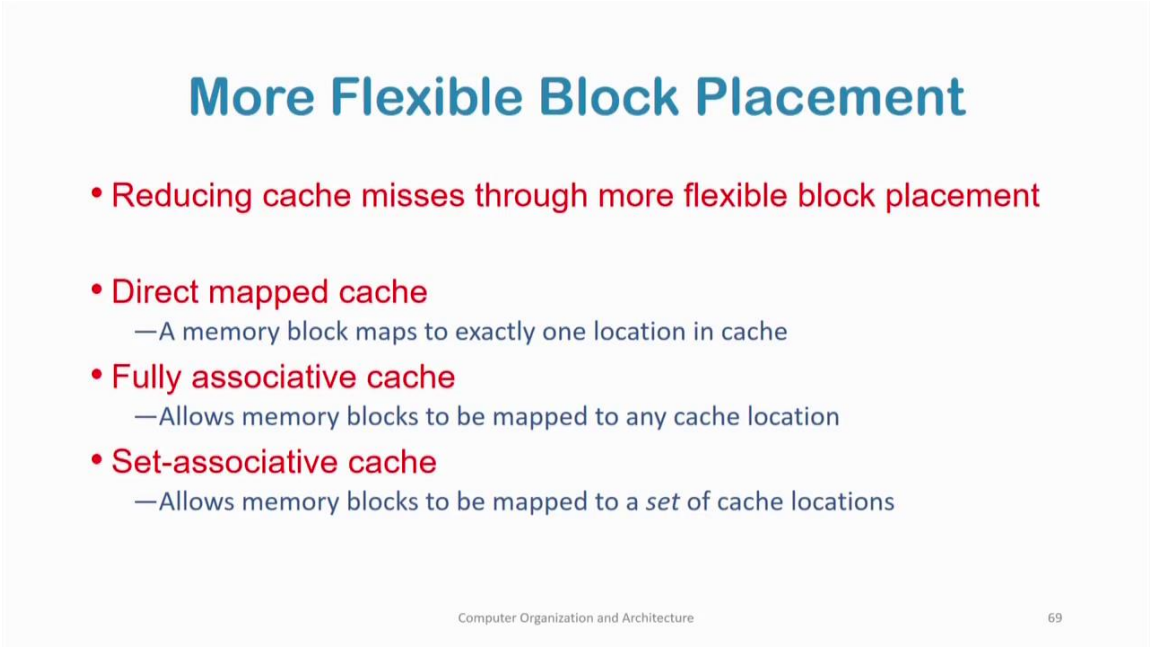
Computer Organization and Architecture: A Pedagogical Aspect

Prof. Jatindra Kr. Deka
Dr. Santosh Biswas
Dr. Arnab Sarkar
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 27
Associative and Multi-level Caches

In this lecture we continue our discussion with cache memories. We start our discussion by looking at how cache misses can be reduced through more flexible block placement strategies compared to direct mapped cache placement.

(Refer Slide Time: 00:36)

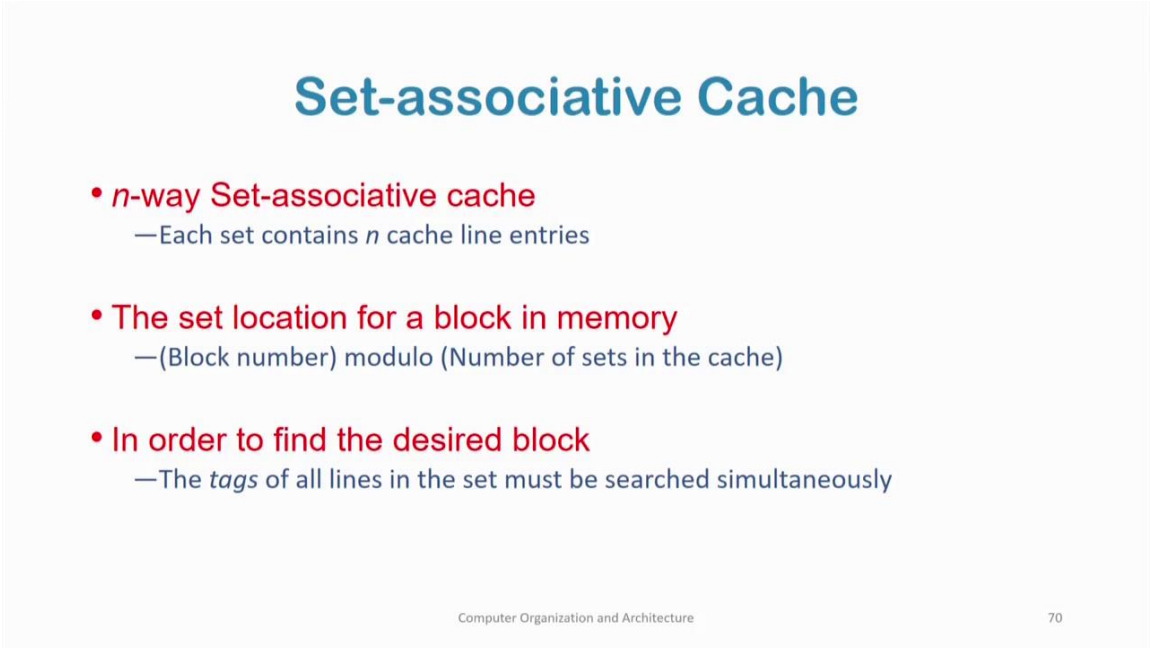


In a direct mapped cache placement, a memory block maps to exactly one location in cache. As compared to that, in a fully associative cache placement the fully associative cache placement allows memory blocks to be mapped to any cache location; that is, in a direct mapped cache I have only one line in cache corresponding to a memory block ok.

In a fully associative cache all lines in the cache can hold any memory block. In a set associative cache corresponding to a given block can be placed in a set of cache lines. So, in a set associative cache allows memory blocks to be mapped to a set of cache lines.

So, in a n way set associative cache I have n alternatives for placing a memory block ok. So, I have n alternative cache lines into which a memory block can be placed.

(Refer Slide Time: 01:49)



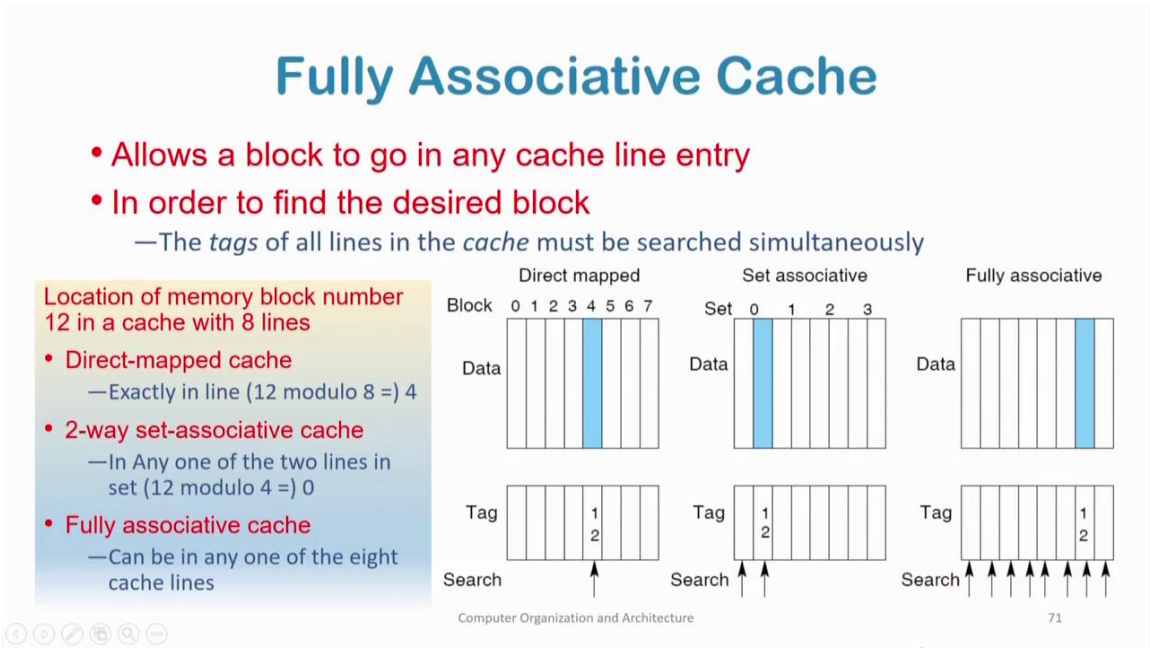
How do I get the set location corresponding to a block of memory? The set location is given by block number, modulo the number of sets in the cache. So, how do we get the block number? So, as if we remember memory address is divided into 2 parts, one is the block offset and the rest is the block number. So, block number is given by what by all those bits of the memory address; which is not the block offset. So, there will the least few bits are for the block offset, the remaining give the block number. So, block number modulo the number of sets in cache.

So, that is and how do I get the number of sets in cache if I have, let us say n lines in cache then, and if I have say 4 ways, if I have a 4 way set associative cache which consisting of n lines. Then n lines divided by the number of ways 4 so, n by 4 gives me the number of sets in cache. So, what will be the set location corresponding to a block in memory? Block number, block number in the memory, block number in the memory modulo the number of sets in the cache.

In order to find the desired block, all tags of all lines in the set must be searched simultaneously, why? Because any line in a given set is equally potential is equally can potentially hold my block in memory. So, therefore, for all because how do I get my

desired line? How do I get my desired block? So, what do I do? I compare the tag corresponding to my memory address with the tags of all the lines in the set. And this is the search is done simultaneously and whoever matches the data corresponding to that is a desired data.

(Refer Slide Time: 04:12)



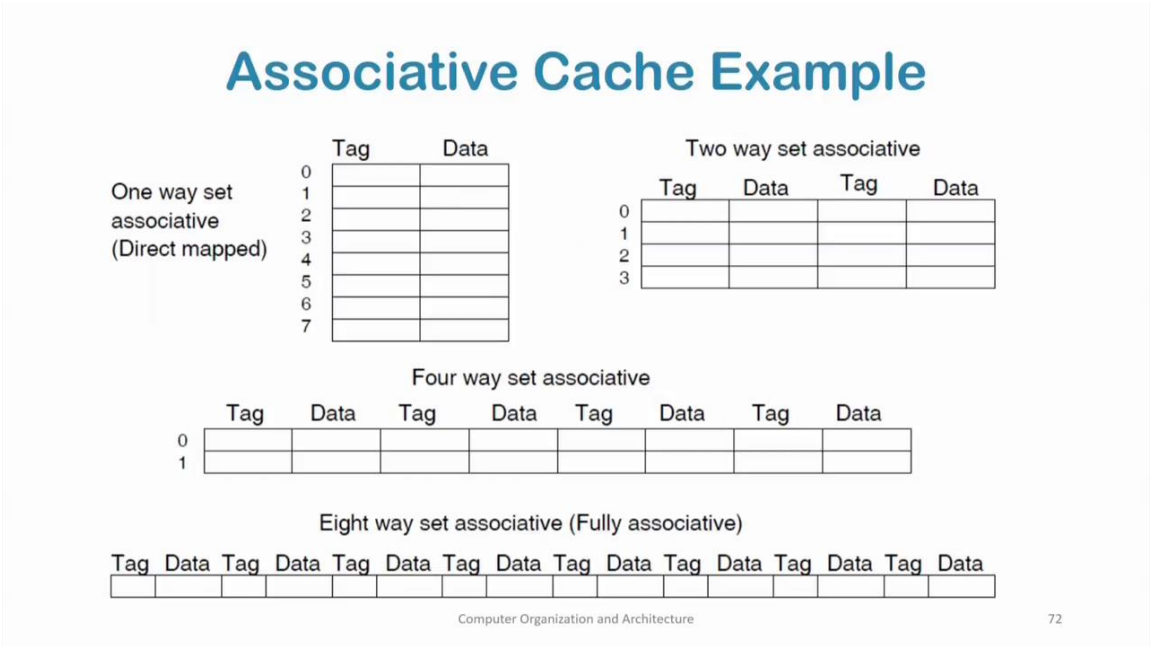
As compared to that, in a fully associative cache, the block may be placed in any cache line entry ok. In order to find the desired block, the tags of all lines in the cache must be searched simultaneously. So, in a set associative cache I only have n alternative lines in which a memory block can be placed. So, therefore, because I have just n alternative lines, I have to search the n tags simultaneously to find desired data. Now in a fully associative cache all lines in the cache can hold my data. Therefore, all tags of all lines must be searched simultaneously to find my desired data.

Now, let's take a toy example. So, we want to find the location of memory block number 12 in a cache with 8 lines. So, I have a small cache containing 8 lines, and I want to find my I want to find memory block number 12 in that cache. So, in a direct mapped cache, I have exactly one line that can hold. So, and which will be that line number 12 modulo 8. Because I have 8 lines I have exactly one place for memory block number 12 and that line is given by 12 modulo 8 = 4.

In a 2 way set associative cache; in any one of 2 lines in the set 12 modulo 4, why 12 modulo 4? Because in this 8-line cache because it's 2 way set associative number of sets is given by 8 / 2 = 4 sets. I have 4 sets in this 2 way set associative cache. And so, which set among these 4 sets will hold my memory block 12, 12 modulo 4 and 12 modulo 4 is 0. So, if we see in this diagram, in the direct mapped cache we see that this line number 4 is the home for memory block number 12. And we match the tag corresponding to corresponding to that I need to match just one tag that of line number 4 to find whether block number 12 is in cache or not. Corresponding to the set associative cache, we said that set 0 is the home for block number 12, and both the lines of set 0 can hold this block number 12.

And therefore, I need to search both these tags simultaneously together to see which one contains my desired memory block that is block number 12. And for this case we have we saw that line number, this line of the first set of the first set holds my data desired data. Now, we come to the fully associative cache. For the fully associative cache any one of the 8 lines can hold the data ok. So, therefore, I need to search all my 8 tags in simultaneously to find if my data is there in cache ok, my desired data in there in cache. Now after searching for the tag I have found that this line number 7 actually has my desired data.

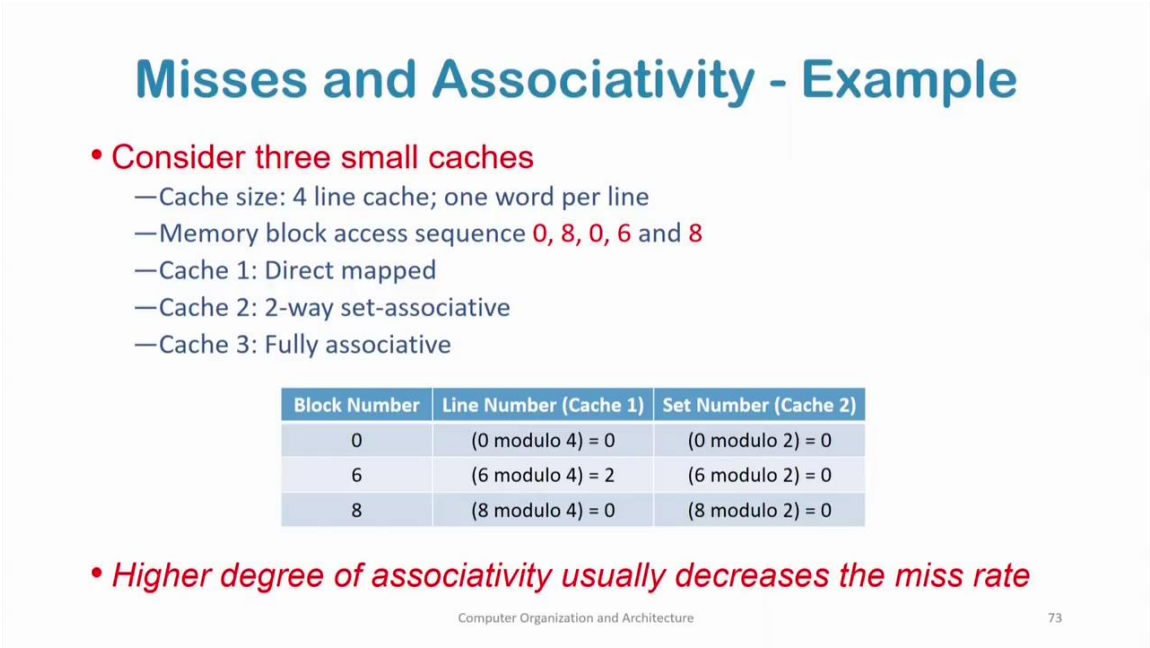
(Refer Slide Time: 07:55)



So, continuing with the associative cache example, I have 8 lines and we can see here that I have 8 lines. And these 8 lines can be compare can be considered to be a 1 way set associative cache. In a 1 way set associative cache, I have just one place for each memory block. In a 2 way set associative cache with 8 lines I have 4 sets with 2 alternative places for each memory block.

In a 4 way set associative cache, I have 2 sets and I have 4 alternative places for each block ok. And in a fully associative cache, I have just one, I have just one set with 8 different possibilities for the memory block.

(Refer Slide Time: 08:57)

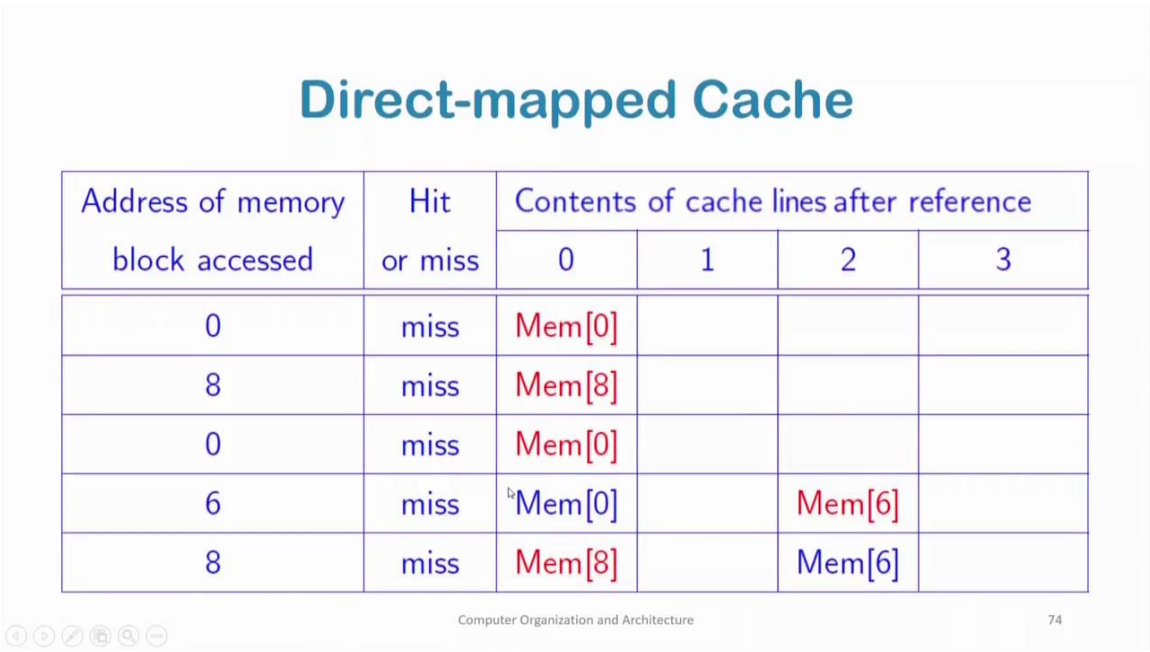


Now, we take another toy example to show how higher degrees of associativity are actually able to decrease cache miss rates. Consider 3 small caches okay, each cache is of size just 4 line and it has one word per line. And memory block accesses are in sequence 0 followed by 8 followed by 0 followed by 6 followed by 8 ok. Now I have cache 1 is direct mapped, cache 2 is 2 way set associative and cache 3 is fully associative. Now, the line number corresponding to memory block 0 in is given by for the cache one for the direct mapped cache is given by 0 modulo 4, 4 is the number of lines, 0 modulo 4 is 0, for 6 it is 6 modulo 4, that is line number 2.

So, the home for memory block number 6 is line number 2. The home for memory block number 0 is line number 0. And the home for memory block number 8 is line number 0

again. Corresponding to the 2 way set associative cache, I have 2 alternative two different sets in which the each block can be placed. And we see that block number 0 goes to block number 0 goes to 0 modulo 2 or set 0. So, I have two sets set 0 and set 1 it goes to set 0, block number 6 6 modulo 2 also goes to set 0, and block number 8 8 modulo 2 also goes to set 0.

(Refer Slide Time: 10:46)



So now what happens for the direct mapped cache? For the direct mapped cache, we see that firstly, when block when block 0 is accessed I have a cache miss the cache is initially empty so, I have a cache miss. And then block 8 is accessed. So, block 8 also goes to line 0, and I have a cache miss. Now when block 8 is brought into cache block 0 has been replaced. So, block 0 is not in cache when it is accessed here. So, this is also a cache miss. For line number 6, for sorry for block number 6-line number line number 2 is the home ok.

So, because 6 is accessed for the first time in line number 2 this also results in a cache miss and when 8 is accessed again, then the home for 8 is 0, previously 0 is currently the currently in the cache. So, again when 8 block number 8 is accessed this memory this block number 8 is not found in line number 0, and therefore, again there is a cache miss. So, all my accesses result in a miss in my direct mapped cache here.

(Refer Slide Time: 12:17)

2-way Set-associative Cache

Address of memory block accessed	Hit or miss	Contents of cache lines after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Mem[0]			
8	miss	Mem[0]	Mem[8]		
0	hit	Mem[0]	Mem[8]		
6	miss	Mem[0]	Mem[6]		
8	miss	Mem[8]	Mem[6]		

Computer Organization and Architecture 75

Corresponding to my 2 way set associative cache, what happens? Again when memory block 0 is accessed, it is accessed for the first time, it goes to set number 0. So, in if in my 2 way set associative cache, all my blocks map to set number 0 right. And I have 2 alternative lines, 2 possible places in which I can place my blocks in set number 0. So now, what happens?

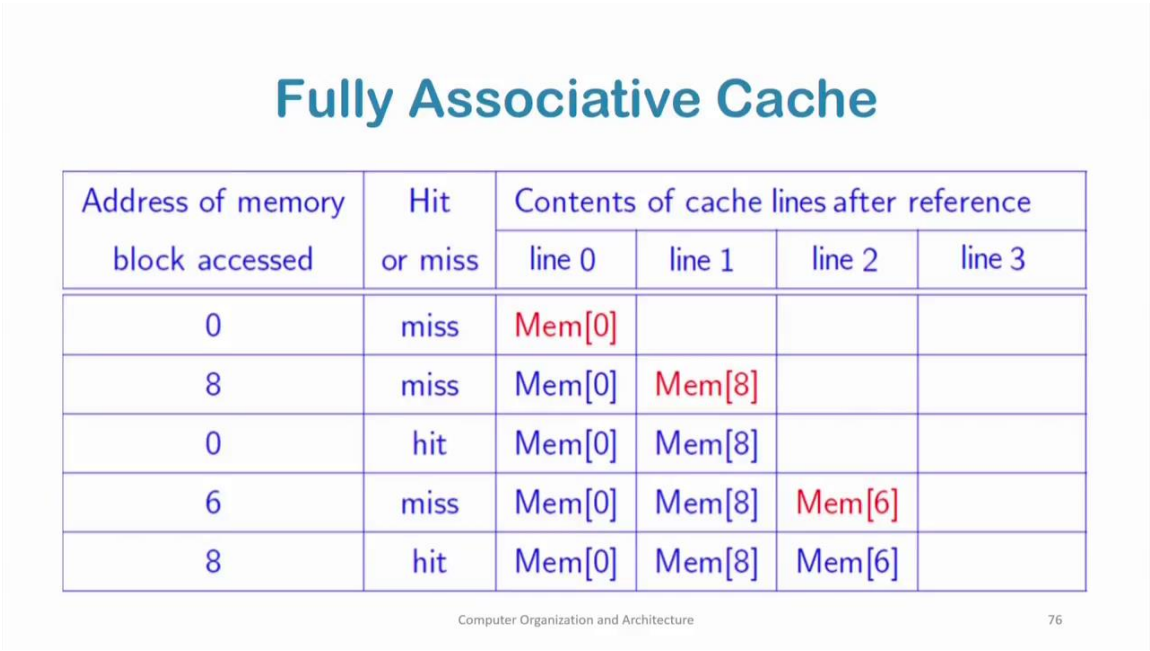
Firstly, when 0 is accessed, I put it to the first place in set 0, and I bring it into the cache there is a cache miss. Because initially the cache is empty, then 8 is accessed when 8 is accessed. I put it in the, to into the second alternative second place in set 0 say. So, this one memory block number 0 remains in the first place in the first line of set 0, and in the second line of set 0 I bring in I bring in memory block 8.

Then when 0 is accessed again, my first line of set 0 still contains the still contains block number 0. So now, I have a hit, I have a hit here. And then what happens. When block number 6 is accessed, when block number 6 is at accessed I need to replace somebody. So, before block number 6 is at accessed lines line the first line of set 0 contains memory block 0, and the second line of set 0 contains memory block 8. So now, when 6 is accessed, I have to replace one of these among 0 and 8.

We will see later how we do this replacement? But for the time being let us assume that I replace I randomly I arbitrarily replace, memory block memory block number 8 and put in memory block number 6. This obviously results in a miss, because this was the first

time memory block 6 was accessed. Now after this 8 is accessed again. And now I have just replaced memory block number 8, and with memory block number 6 in the second line of set 0, and therefore, memory block number 8 is currently not in cache. So, when memory block 8 is accessed here it again results in a miss. And now I replace memory block number 0 with memory block number 8, and bring memory block number 8 into the cache. So, here I have just one hit. So, whereas, in the direct mapped cache all my accesses resulted in a miss, I have one hit in this 2 way set associative cache.

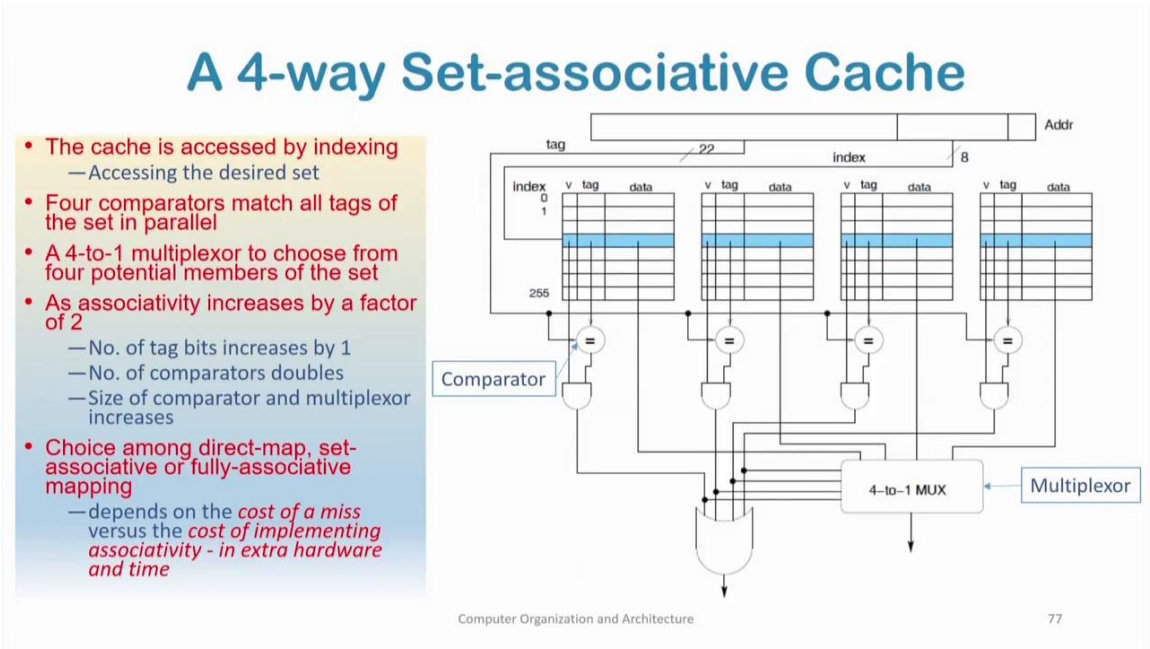
(Refer Slide Time: 15:20)



Now, when I have a fully associative cache all my lines can hold data okay. Can the data the memory block can be placed in any of the lines. So firstly, 0 is accessed, I put in line 0 ok. There it results in a miss being the first accessed. Then 8 is accessed I put in this line number 1, it again results in a miss because 8 is accessed for the first time; when 0 is accessed for the second time 0 is currently there in cache and therefore, this results in a hit.

And then when 6 is accessed, 6 is accessed for the first time it this results in a miss and I put it in line number 2. And then when 8 is accessed for the second time, it again results in a hit because it is there in line number 1. So, we have 2 hits here. So, we see that a fully associative cache is able to further reduce the number of misses. So now, we will look at how set associative caches are organized.

(Refer Slide Time: 16:27)



Now, we will consider the case for a 4 way set associative cache. So, let us consider the memory address here. This memory address has a block offset; it has 8 bits for indexing ok. And 22 bits for the tag bits now these 8 index bits, these 8 index bits are used to find the set in the cache. So, if I have 8 bits for indexing, the total number of possible sets in the cache will be 2^8 ok. Or I will have 256 different possible 256, 0 to 255 256 different possible sets in the cache. And each set will have 4 lines.

So, for example, let us assume that this the this index the value of this index has resulted in this location, this set location. And therefore, for this set I have 4 alternative lines. Here is one line, this is the second line, this third line; this is the 4th line ok. Now any of these 4 lines can hold my data. So, how do I find is my desired is my desired data there in cache? And if it is there in cache which one of these lines hold my data? So, this will be found by matching with the tags, this 22-bit tagline. So now, what happens ah? These 22 bit tags are matched using comparators. So, these there are 4 comparators corresponding to this the tag the tag corresponding to the line comes over here.

And it is it is matched with the tag that comes to this. So, this tag is matched with each of the tags in my lines. And if it matches and if the data is found to be valid then I get my data okay. Suppose whoever matches will hold my data now so, how do I get my actual output? I get it through a multiplexer ok. So, this will tell me that whether this will tell

me whether I have a cache hit or a cache miss, because this is this is going to tell me that if any one of these lines are 1 or not. So, when will any one of these lines be 1, if I have valid data which matches so, I have a tag match. So, if I have a tag match, I will I will get a 1 on these lines.

Now if I have a 1 on these lines, I will use a 4 × 1 multiplexer. So, 4 × 1 multiplexer, and these will be my select lines to choose one of these data inputs. And so, whoever is 1 over here whoever is 1 over here, corresponding to that the particular in data input will be chosen, and the correct data will flow from this line to the output of the multiplexer and we will be able to fetch the data, that the desired data from my cache. So, we see that the cache is accessed through indexing. The 4 comparators match all tags of the set in parallel.

A 4 × 1 multiplier is used to choose from 4 potential members of the set. As associativity increases by a factor of 2; that means, I go from let us say from 2 way set associative to 4 way set associative, or from 4 way set associative to 8 way set, associative what happens is that the number of tag bits increases by one. Why does this happen? Because when my associativity increases the number of sets reduces by a factor of 2. So, when associativity increases so, I instead of having 4 alternative places to in the in a given set, if I had a 4 way set associative. Now if I make it a 8 way set associative cache I will have 8 alternative lines corresponding to any given set.

So, for if the cache size remains same the number of sets in cache reduces by 2; so, therefore, the number of index bits reduces by 1 and the number of tag bits increases by 1. So, fine the next point is that, the number of comparators double why because, the number of lines in a set double. The number of lines in a set doubles. So, the number of comparators will also double. The size of each comparator and multiplexer will also double, because my tag bits increase the now tag bits increase. So, the size of the comparator also increases. Now if it goes from 4 way set associative to 8 way set associative, the size of the multiplexer increases from 4 × 1 to 8 × 1. So, the size of the multiplexer also increases.

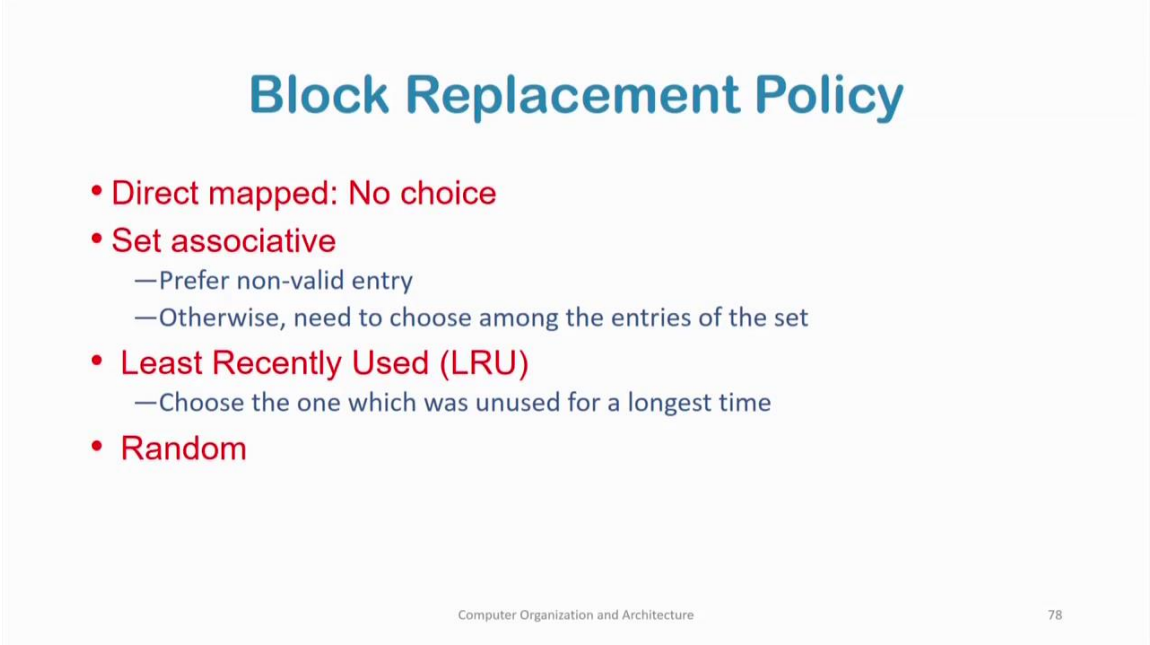
Therefore, the choice among direct mapped cache set associative cache and fully associative cache depends on the cost of a miss. So, what did we see that, we saw that when my associativity increases, I have a decrease in the number of misses my hits

increase; however, there is a cost in implementing this associativity, we saw that as associativity increases number of comparators, size of comparators, size of multiplexers double. Size of all of them increase, number of comparators double, now the comparator sizes increase, the come the size of the multiplexer doubles. Now so, the area cost corresponding to this implementation becomes very high. Along with that there can also be certain increase in the in the in the hit time. Why? Because just assume that we had a direct mapped cache.

In a direct mapped cache, it will require just one comparator. There is one line in which my block can remain and therefore, and therefore, what will happen is that, I will just compare with that and get my data. Now, I have to in parallel go through many comparators, the size of these comparators are bigger, because the tag bits are bigger and then it has to go through a second level of multiplexer to get my desired data, right.

So, this all of them that multiplexer will also have a delay. So, the delay corresponding to the comparator, the delay corresponding to the max may increase along with that I have to go through 2 levels of logic. So, one more level of logic at least. So, therefore, time of access may also increase. So, I have a tradeoff between the reduction in miss with respect to the cost of implementing associativity in terms of both area hardware area, hardware cost as well as time.

(Refer Slide Time: 24:25)



So, block replacement policy. As we did in the example as we saw in the example that when for example, in a 2 way set associative cache, I will just go back to this example once. In the 2 way set associative cache consider this 6 here. Before this 6 was at accessed set 0, the 2 lines of set 0 contain memory block number 0 and block number 8. Now when 6 was brought in I replaced, this memory block number 8 in line one of set number 0 and put 6 in it. So, we at that time I said that I replaced this memory block number 8 arbitrarily. But what is the algorithm that I used? For replacing memory block number 6, memory block number 8 with memory block number 6.

So, this is: what is the question we are looking at, block replacement policy. In a direct mapped cache, I have no policy. I need there is no choice shown hence no policy. In a set associative cache, we first prefer non valid entries. So, if I have a non-valid entry, I will if I have a non-valid entry I will directly use that and replace that because, any way that cache location that cache line contains non valid data, so I will replace it. Otherwise I will choose among the entries of the set. Otherwise, I have this case here that memory block number 0 and memory block number 8 is in line number 0 and line number 1 of set number 0 and whom to replace.

The policy that is typically used is least recently used. So, what is least recently used policy? Choose the line choose the line which was unused for the longest time. Now, in the above case if we if we now go back we will see that in the above case 8 was accessed in time step one in means in time step 1 and 0 was accessed in sorry in times if this is time step 0, 1, 2, 3, 4 in time step 1 8 was accessed in time step 2 0 was accessed, now in time step 3 when 6 is accessed, 8 is the least recently used; because it has been used that the longest time backwards. So, we choose the one which was unused for the longest time. The other option is to choose randomly. Least recently used is typically mostly used random has some advantages we will look at later.

(Refer Slide Time: 27:24)