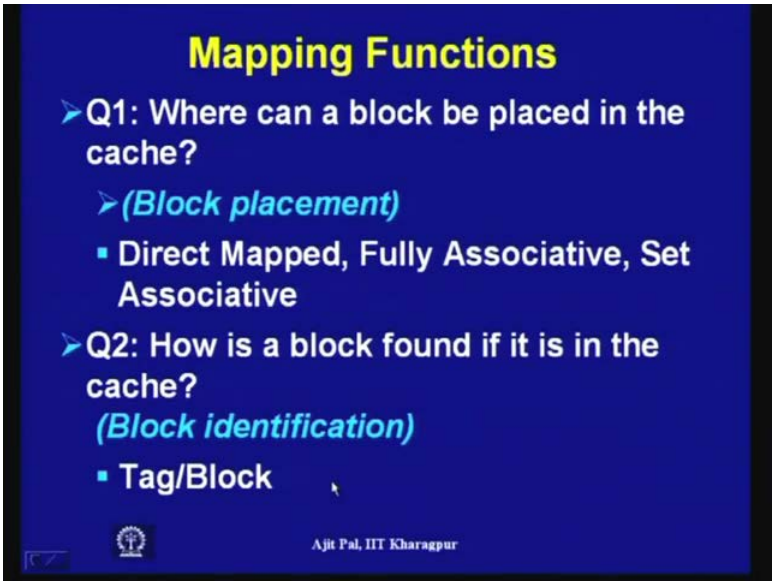
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 23
Hierarchical Memory Organization (Contd.)

Hello and welcome to today's lecture on Hierarchical Memory Organization, we have already discussed two topics on this and particularly, we shall be concentrating on mapping function today.

(Refer Slide Time: 01:05)

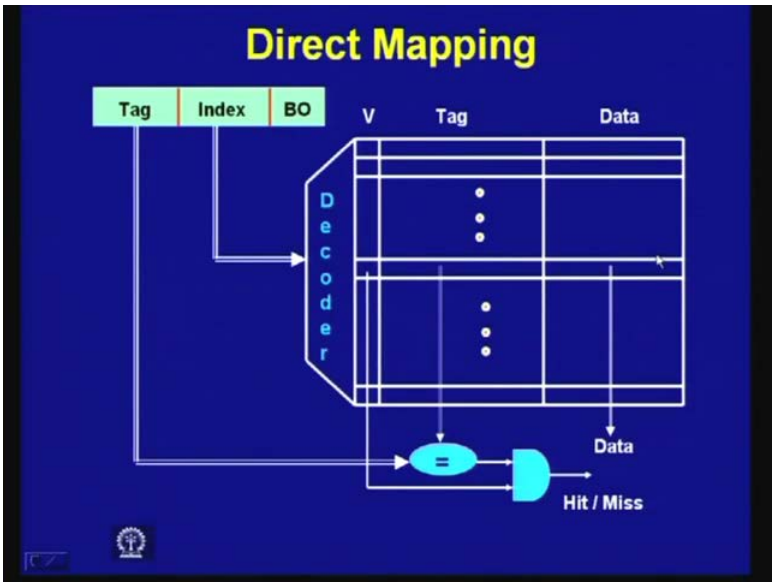


And as we have seen the mapping function can answer two important question related to hierarchal memory organization, number one question is where can be a block be placed in the cache memory, that is block placement. And we have already discussed about two approaches as I have mentioned, first one is direct mapping, second one is fully associative mapping.

And today, we shall continue our discussion on set associative mapping, and second question that this mapping function can answer is how is a block found, if it is in the cache. That means, whenever a particular block is present in the cache, how can it be found out from the cache memory, these two question can be answered. And in the particularly, the second question is answered with the help of for tag field and block, the

tag field and different locks present in a block different words present in the memory, we shall see how it can be found out.

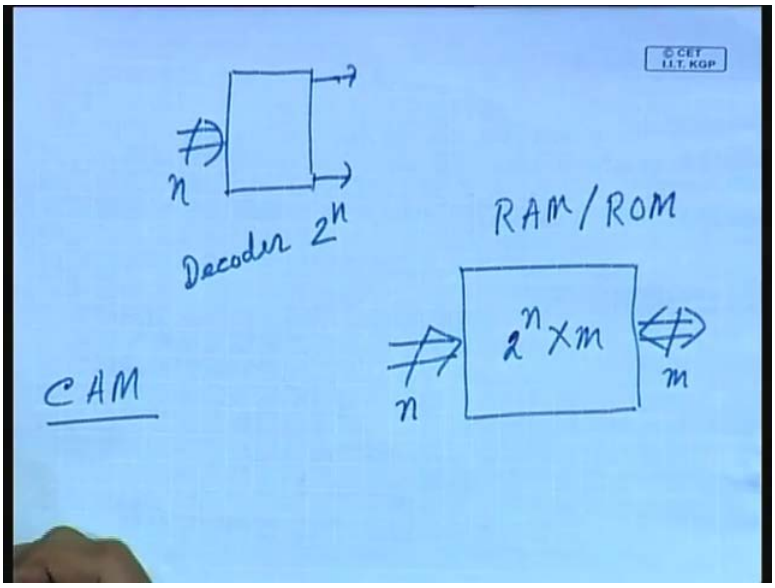
(Refer Slide Time: 02:15)



So, let us quickly recapitulate the direct mapping, we have seen in case of direct mapping, the address that is coming from the processor is divided into 3 fields, first is in the first field is known as block up set or bite up set also, you may see initially represent the number of bites present in a block. And the second field is index field and as we have seen in case of direct mapping index part is used from the purpose of identifying, which particular line I mean I points to a particular line, and checks whether the that line corresponds to the address that is being generated by the processor.

So, this is the address that you need by the processor, and as we can see that index field is used as a kind of I mean address to the cache locks or cache lines. And it goes to the decoder and decoder identifier a particular line that is present in a in cache memory, and as we know the function of a decoder is.

(Refer Slide Time: 03:28)



If it has got n inputs, it generate 2 to the power of n outputs, and one of them is active that is a function of a decoder, and here as you can see the index field is applied and; obviously, one of the lines that will be coming out from the decoder will be active. And in this particular case this particular line is active and; that means, this is the block or line we have to consider, that may correspond to the address. While, I am telling may the reason for that is it is many to 1 mapping, and by comparing the tag field of the address with the tag field that is present in the cache memory.

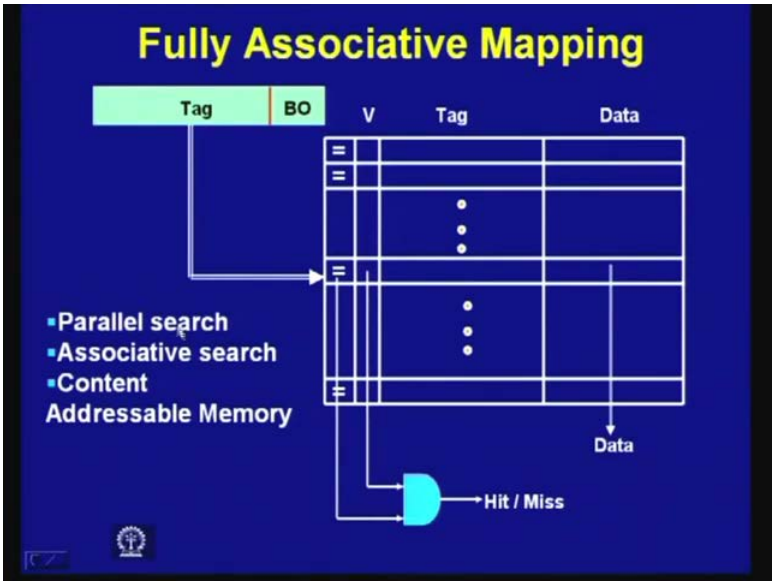
We have to see, whether that address really corresponds to this line of the cache memory, that means whenever there is a matching this compared with the help that this two tags are compared one coming out from the cache memory, another that is stored in the cache memory. These two are compared and if these two are same, then only you know that there is a matching of course, that valid v field has to be 1, but memory at some point of time has been transferred to, because the memory it indicates that it means this is one only, then output will be hit.

That mean if there is any matching here and if the valid it is 1, then it is a heat, otherwise it is a mess. So, this is the direct mapping and you can see, the memory that can be used in particular in direct mapping is conventional memory, as we know in conventional memory we have got we apply address at this is applied, and it produces data. And, if the

number of address lines in n number of data lines is m, then you know it that memory in 2 to the power n into m memory.

It can be it may this is standards organization of memory, it can be for RAM, as well as for ROM, so you can see here we need not use any special memory, for that whenever we are using direct mapping. Standard memory can be used only thing in the memory, that you stored different components like valid bit and the tag field, and the data, the data can be instruction or data depending on, it corresponds to be instruction case or data case that we see.

(Refer Slide Time: 06:10)



So, this is the direct mapping, then we have seen as fully associated mapping, and as we know in case of fully associated mapping, we have to perform parallel search in all the lines. So, your memory is not clearly conventional memory that memory has to here as you can see there is no index field, so since index field is not there, it does not point to any particular cache line.

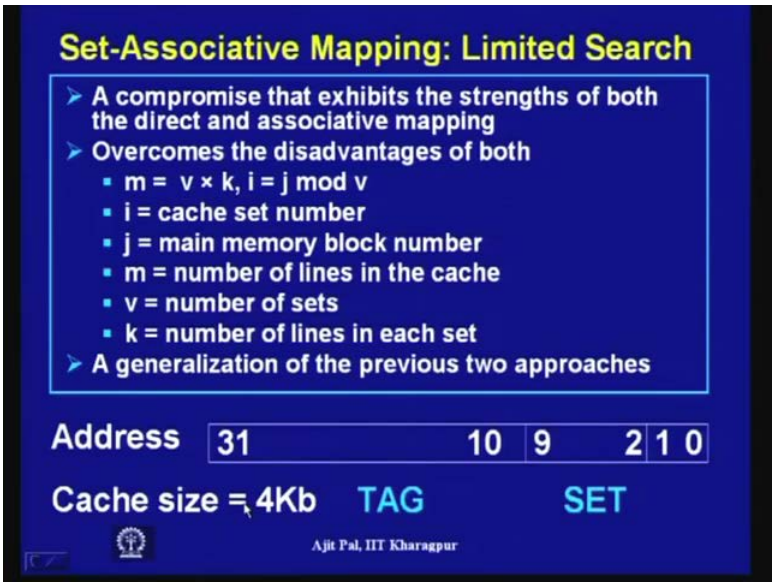
So, that tag field is compared with each and every tag field that is present, I mean focusing to each line kind of parallel comparison is done, and that is the reason why it is called cam, content addressable memory. So, here the RAM has to be a very special type; that means, RAM will compare with the part of the content, so here what is the content, content is the tag field.

So, that tag field content has to be compared with each and every tag field that is coming out from the address generated by the processor, and we have to do parallel search in all the lines that is present in the cache memory. And, it also called as associative search, and as I have already mentioned content addressable memory has to be used, and whenever there is a matching with one of the tag field, and also the valid it is 1, then it is a hit otherwise it is a miss.

So, we see that in this particular mapping, whenever it is used, the memory has to be a special type and it is quite complex, because not only you have to store some information in the form of a instruction or data, you have to also provide mechanism for parallel comparison with in each and every line that is present in the memory. So, it is quite complex and costly, so it is in practice, this is not used and also the direct mapping which I have discussed all that is simple, but it does not give you very good performance.

So, we have to go for a particular mechanism, which will give you best of both the rewards mentioned in my last lecture, which tries to combine the good features of both the mapping function, that is for direct mapping as well is fully associated mapping.

(Refer Slide Time: 08:45)



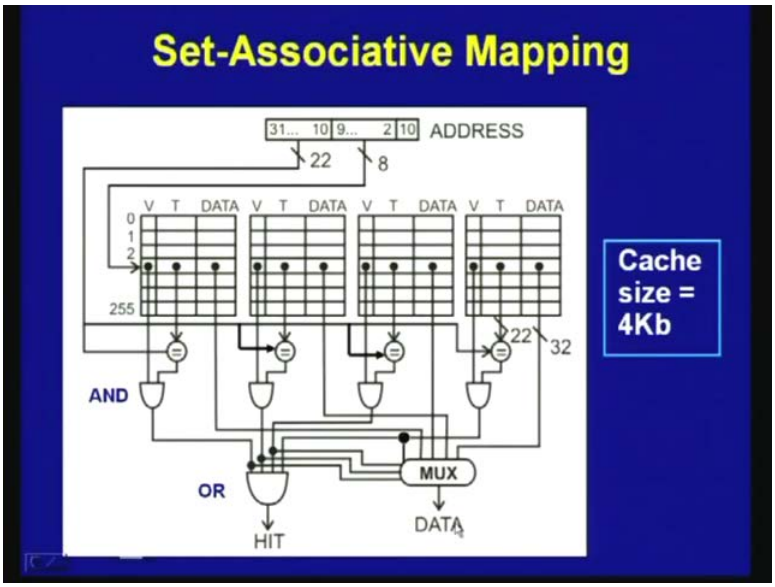
That is known as state associative mapping, so which performs limited search, so it is a compromise that exhibits the strengths of both the direct and associative mapping. And it also over comes the some of the disadvantages of both; that means, the complexity is

much lower than the fully associative mapping, and it gives you better performance, it is achieved in mapping is improved this techniques.

So, in this particular case m is the number of lines, that is equal to v into k, where v is the number of sets and k is the number of lines in each set, and j is the cache set number and i is the cache set number. So, that means, this i is equal to j mod v, because v is the number of sets that is present and j is the mean memory block number. So, you can find out i by taking a mod of for j with respect to v, j mod v and that will give you i that is the cache set number, so here as you can see again the address is divided into 3 fields.

The first part is byte of set that or block or you may say block of set it more than what is present, so it is essentially it presents the number of bytes, so at this time it has been assumed that for 4 bytes are present. So, 2 bits are required for that byte up set, and then you require several ways to represent the set, so here we have got 7 bits and assuming that cache size is only 4 kilo byte; that means, we require total of 9 bits to represent the size of cache memory, and the remaining bits are for address, so you can see here.

(Refer Slide Time: 10:55)



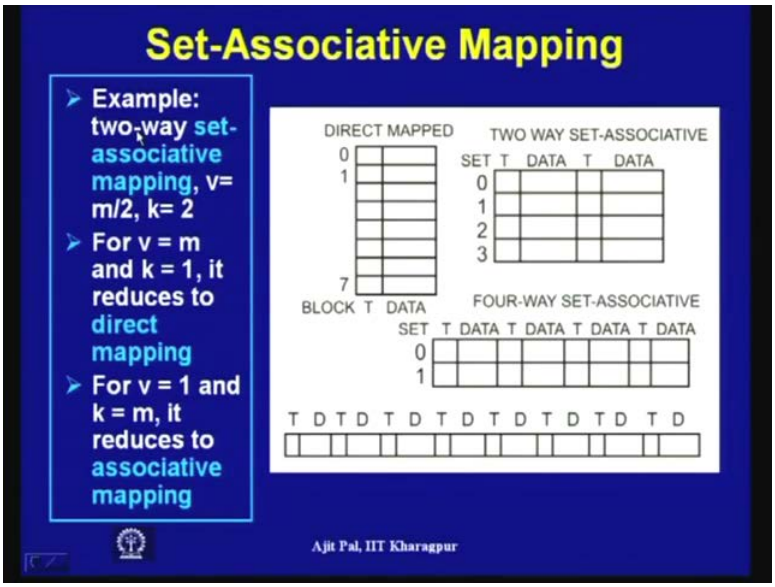
The example, that is being shown in this particular case, you require, you can see here it has been assumed that you have got 4 sets, I mean in a particular the you have got different in particular set, you a have got 4 lines. So, you have got 4 lines, and that is why it is called 4 way (Refer Time: 11:20), so 4 way (Refer Time: 11:22) has got 4 lines

in each sit, and you can see total number of sets is equal to I mean 256, because you require 8 bits.

And remaining bits that 10th bit to 31st bit, remaining 22 bits are to be stored in tag fields of different data sets, so you can see here the comparison has to be done, for each set I mean for a particular sake, you can see about four lines for each of the lines you have to perform parallel comparison. So, our can see the number of comparative that is required in this particular cases 4, because it is a 4 sata sit memory, so instead of that number of comparators that is equal to the number of sets.

Here, the number of sets is lesser, as we can see and only 4 and you require a multiplexer to find out, which one of the line is corresponding to be addressed that is being generated. So, that is found out by selecting one of the words, one of the lines and with the from the that will come from, that it will come from the block of set field, and that is being applied from the multiplexer, and with the identifier is one of the locks and that goes as a data.

(Refer Slide Time: 13:11)



And let me illustrate it very simple example, where we shall have the same cache size, but we shall go from simple direct mapping to fully associative and 2 way and 4 way set associative. So, in this particular case, as you can see you have got only in case of direct mapping, we have got only 8 sets, so in case of two way set associative mapping number

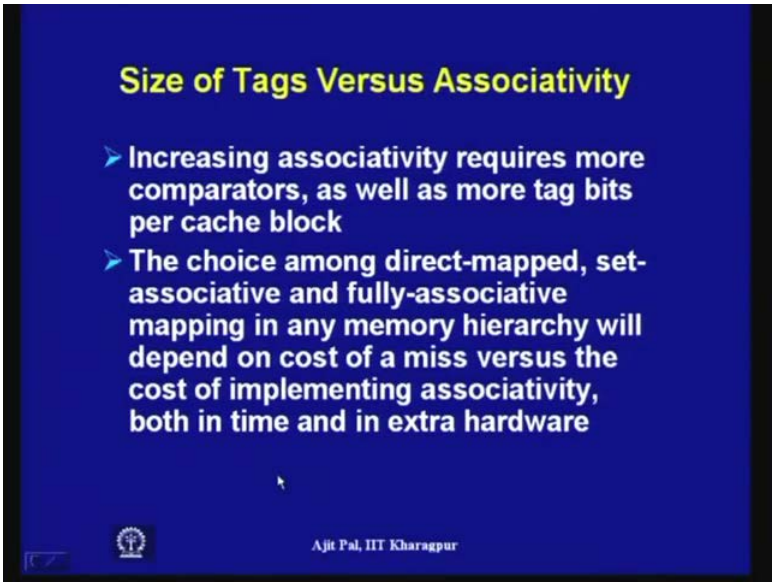
if sets will be equal to m by 2. So, here m is equal to 8, so 8 by 2 we have got 4 sets, and in each set will have to two lines that is two way set associative memory.

And similarly, as you go from and as you can see this is the direct mapping, from direct mapping v is equal to m, k is equal to 1, that is the number of lines that is present in the set is 1, that is why, it reduces to direct mapping. So, you can see that set associative mapping is a generalized situation, we can consider it as a general case, and direct mapping is a special case as set associative mapping whenever your k is equal to 1, that means the number of lines that is present in it is 1.

So, it is reduces to direct mapping, on the other hand when v is equal to 1; that means, we have got only one set, and in that set you have got m lines that is present. So, this corresponds to we have got only one set, as you can see, so in a single switch, we have got all the lines that is present in the memory. So, this is the to associative mapping, this is again the special case of two way set associative mapping, where v is equal to 1, that is number of sets is equal to 1.

And the number of lines is present is equal to the number of liens that is present in cache memory, and you can see this is direct mapped, this is 2 way set associative mapping, this is 4 way set associative mapping, and this is corresponds to fully associative mapping.

(Refer Slide Time: 15:49)

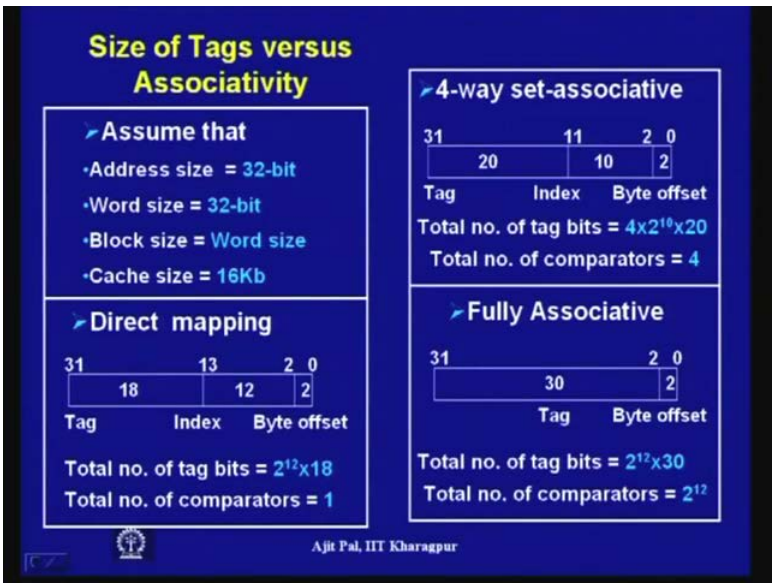


So, the mapping function how the mapping function varies for different situations with the help of the simple example, now as the associativity changes, your performance improves, it has been found that as we improve the associativity, the performance improvement improves, but it will not come free of cost. We have to pay price for it, what kind of price we have to pay, number one is increasing associatively requires more comparators, that we have seen as well as more tag bits per cache block.

So, as you go from down direct mapping to fully associative mapping, number of tag bits keep on increasing, number of comparators keep on increasing. In direct mapping, we have to see you require only one comparator, and the tag bits fields size will be minimum, as you go two way set associative mapping, you will see that size of the tag is increasing, number of tag bit is increasing number of tag bit will become double.

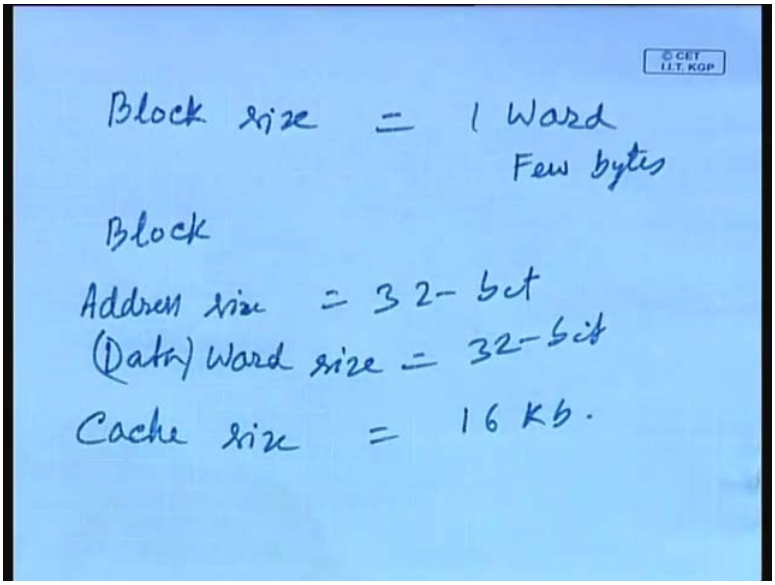
So, the choice among direct mapped set associative and fully associative mapping in any memory hierarchy will depend on the cost of a miss versus the cost of the implementing associativity, that means whenever this particular occurs, cost of miss means miss penalty versus the cost of implementation. So, you have to ultimately consider the we have to look at cost performance of the memory system; that means, if you can took to have larger cost, fully associative is very good. On the other hand, here we have to optimize the cost that you will go for either 2 way set associative or 4 way associative or sometimes 8 way set associative.

(Refer Slide Time: 17:44)



Now, let us consider a simple example, just to illustrate the how the cost increases, so let us assume that the address size is 32 bit, word size is 32 bit and block size is word size, so for the sake of simplicity we have assumed that here the a block.

(Refer Slide Time: 18:06)

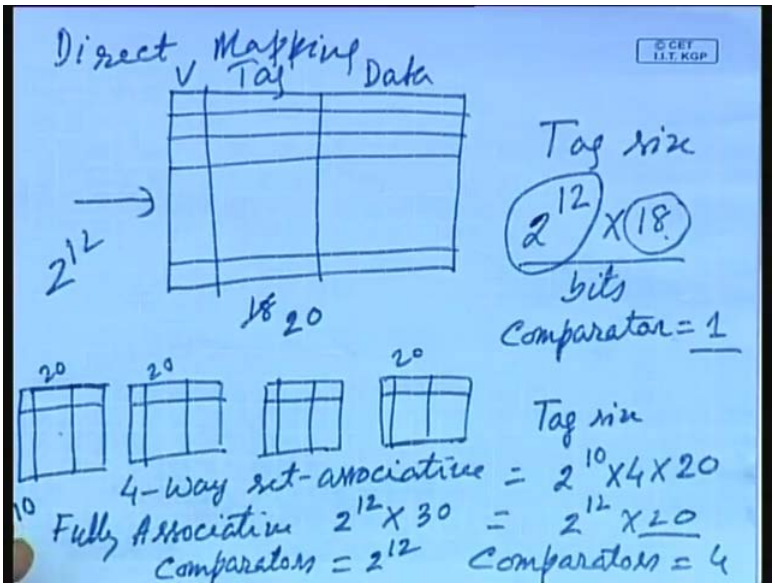


Size is equal to 1 word, 1 word may consist of few bites, normally a block comprises several words, but in this particular case we have assumed that there is only 1 word, in address size is 32 bit, data size is also data size or word size is also 32 bit. That has been assumed, in other words we are essentially considering a 32 bit processor, and 32 bit processor will have 32 bit word size, and 32 bit address size and cache size is consider is said to be 16 kilobyte, so cache memory size is equal to 16 kilobyte.

Now, let us see for different situation, how the number of comparator and number of tags changes, so this is this will corresponds to direct mapping, and 2 bits will be require for byte upset, because you know your word size is 32 bits. So, 2 bits with the help of that; that means, you can have up to 4 bytes, 4 bytes means 2 by 2 will be sufficient to address, and that is for byte option. And since, you have got 16 kilobytes of RAM, totally we require for 14 bit, but out if that 2 bit goes for byte offset remaining 12 bit will provide you the index.

So, this will with the help of this bits will be able to point a particular cache line, and then the tag bit will be 18, so you can see total number of tag bits, that you require we have seen your cache memory in this in case of direct mapping.

(Refer Slide Time: 20:26)



We have 3 fields as we know tag, valid bit and the data, now this tag field in this case is a 18 bit and number of lines that is present that will be decided by the number of index bits. Number of index bit is 12, the bytes actually number of I mean index number of index field is 12, so you will have 2 to the power 12 lines you can say, so in each line will be having 18 bit, so the total tag bit is particular is 2 to the power 12 into 18. This is the tag size, in case of direct mapping, and number of comparator that will be require is only 1, so this is the size of tag bits, and this is the size of comparator, now let us go for another case that is 4 way set associative mapping.

So since, this sizes of the cache is same 16 kilo bytes, now you will have 4 lines, so 4 lines in each set, so essentially you will be it will like this. So, you will be having 1 2 3 and 4 such in a particular set, you will be having 4 lines, 1 line, 2 line, 3 line, and 4 line, so this is 4 way set associative. So, in this particular set, since the size in this case will be 2 to the power of 12 and it is divided into 4, so number of bits in index field will be from now 10 bits, so the number of sets that will be present is 2 to the power 10.

And the size of the tag field, now will become equal to 20, because here this will now become 20, because the size of the index field is reducing. So, the tag size in this particular case is equal to 2 to the power 10 into 4, because you have lines into 20, because 20 is coming the size of the tag fits, in each of this field will be 20, earlier it was 10, now it will be 20. So, this is the size of the tag field, so this main can be represented

as 2 to the power 12 into 20, so this part is remaining same; however, the this is increasing from 18 to 20.

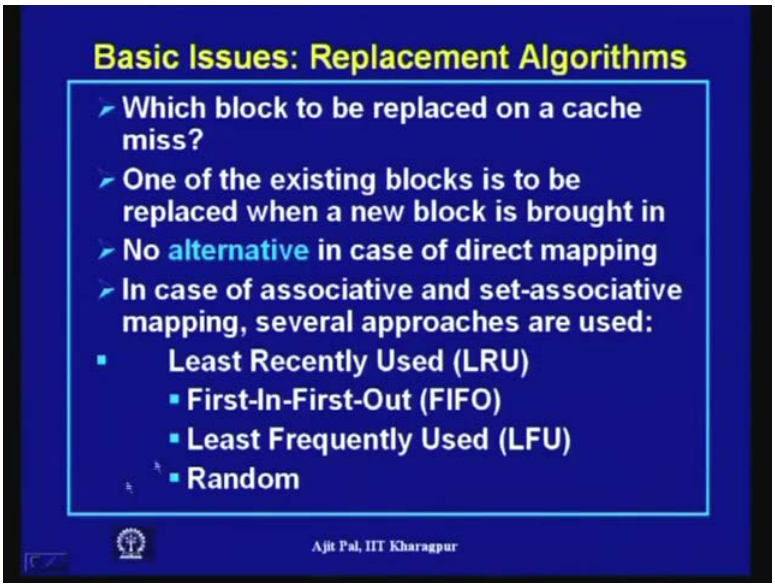
So, size of the tag field is becoming, you can see it is in increasing significantly, this is becoming 18, it is 18 into 2 to the power 12, and this is 20 into 2 to the power 12. So, it is significantly increasing and number of comparators, in this case it will be into 4, so number of comparator is increasing and the number of tags is increasing. Now, let us consider the fully associative memory, so in case of fully associative memory, we can see there is no index field, because index field is not required, you have got only one set because index field identifies particular set and since you have got only one set.

And so, you do not require the index field, and the entire remaining part of the address, that is your apart from your byte offset the remaining bit, 30 bit is the size of the tag field. In this particular case what will be the total number of tag bits, so this is your 4 way set associative, what about fully associative, in case of fully associative here that 2 to the power 12 will remain the same, because that will be the total amount of lines.

And that has to be multiplied with 30, that is 30 is the number of bits that is in the tag field, so this is the size of the tag field and number of comparator is equal to now 2 to the power of 12 not 1 or 2, but 2 to the power of 12. In other words you require a content at the special memory, a special type of memory in which the comparison has to take place in parallel with each and every line present.

So, you can see the how the size the tag bits is increasing size of the number of comparators in increasing, so this is the case for 4 different situation, direct mapping 4 way set associative and fully associative. Of course, you can have 2 way set associative, and 8 way associative and for them this figures will be different. You can easily find out the number of bits that is required for 8 way set associative, that is more than double of that 4 way associative and maximum number of tag bits is will require a fully associative, and total number of comparators will be 2 to the power of 12. So, by this source, how the cost is increasing for implementing as you increase the associative.

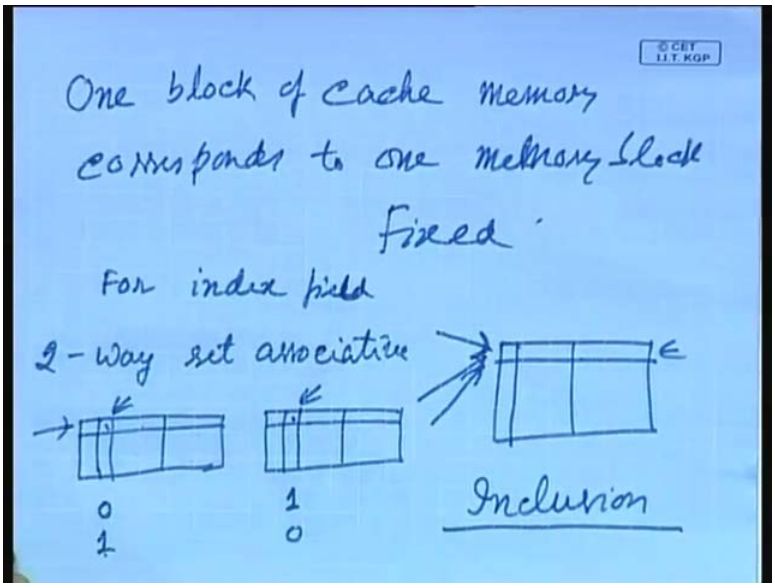
(Refer Slide Time: 27:23)



Now, comes the question of replacement, as we know only a small part of the main memory is present in the cache memory, it will be a many to on mapping. So, whenever there is a miss, you have to take new block from the main memory, you have to transfer new block from the main memory to the cache memory. So, you have to replace one of the block that is present, how will you do that, what value you will use for that.

So, that deals with the replacement algorithm, question the in this particular case, the question is which block to be replaced on a cache miss, so one of the; obviously, one of the existing blocks is to be replaced, when a new block is brought in, so this quite obvious.

(Refer Slide Time: 28:35)



Now, if you focus on direct mapping, we have seen in case of direct mapping there is a I mean one block of cache memory corresponds to one memory block. That means, it is fixed. We have seen that is this is the cache memory, and if the index field is same, I mean for same index field, it will map to the same line, you can say set of the cache memory, so it will map to the same. So, it will be in this case, it will be no alternative, that mean if the index field is same, then for different tags it is fixed. So, there is no choice you do not have any choice of replacement, it is fixed how ever in case of associative, an set associative mapping there are several approaches, because you have choice.

Now, let us consider the simplest case, simplest case is 2 way set associative, in case of 2 way set associative you have got for the same index, you can have two lines. That means, you have got two choices either it is present in this particular line or it is present in this particular line, depending on the index field. So, it is either here or here, so you have to replace one of them, so you have got two choice says one of them has to be replaced, and which one do you replace, and for that there are several algorithms our approaches the simplest one is known as least recently used.

That is the most common word what do you mean by least recently used; that means, you replace that one which has not been used in several times, how do you keep track of that what you can do, you can add another bit. So, whenever another bit and that particular bit

is used only for the purpose of replacement, what you can do, you can if this is being accessed the particular in stand, you can set it to that particular bit can set to one, and the other bit can be set to 0. So, that means, so far as the present instant is constant, this is the most recent line compared to the this one.

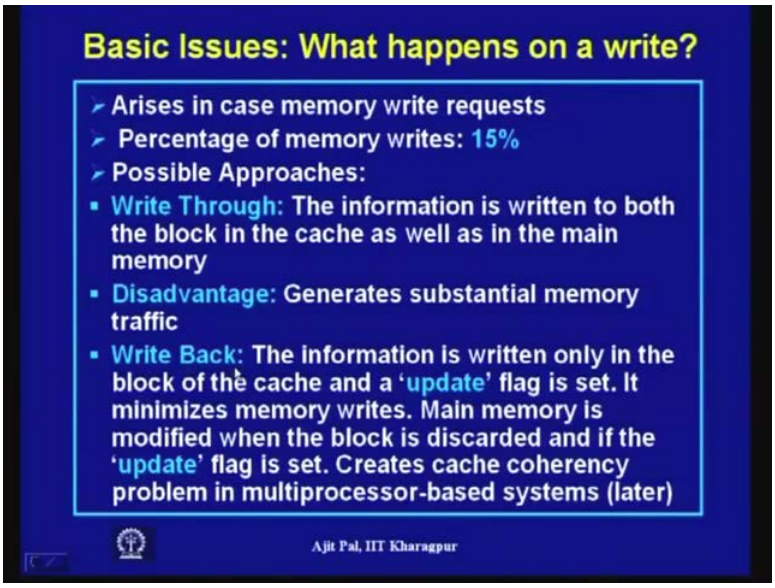
So, if this is accessible at next instant this will be again set to one, this will be set to 0; that means, what you can do by changing these with the help of one additional bit, you can find out. And whenever you want to make the replacement what you will do you will check which one is one, I mean replace that one for which it is 0, that is how the replacement can be done, but matter get complicated.

As, you go for 4 way sort associative 8 way set associative or fully associative in that case you cannot keep track of this least recently used with the help of a single bit, you have used multiple bit and you have to modify each of these lines. So, it becomes quite complicated there are other approaches for example, first in first out, so you can maintain a first in first out first in first out I mean kind of different register, so in this case the it is organized in the form of (Refer Time: 33:10) register.

So, the first in which was brought first will be replaced, now, so this approach can be used another is least frequently used. So, in this we have to keep track of the number of access of different lines in a particular set, and that again essentially you have to maintain an account starting from I mean when you put the power on or you may regular interval you can synchronize it by setting all the bit 0. Then increasing in physical account counter, so least frequently used can be implemented read with the help of having a counter, for each and every line that is present.

Of course, the simplest one is random, you simply replace any one of them there is no extra over head you have to maintain, any I mean any housekeeping information no extra weight is required, but random may not require be very good results. So, this can be used as I mentioned least frequently used most common one, but there are other options that have discussed in sake of complement.

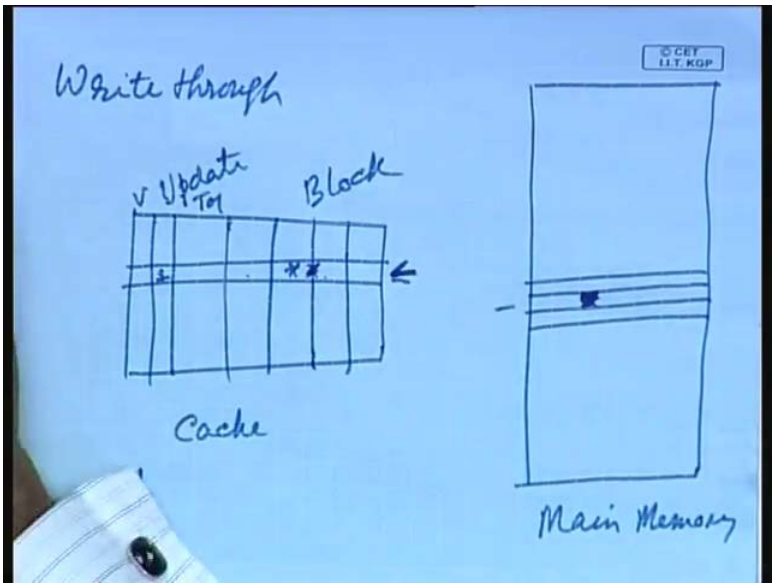
(Refer Slide Time: 34:30)



Now, another important issue is what happens on a write, that means arises in case of memory write request. So, as we know you have to perform write, you know there are instructions like load and I mean load and store, or you perform write in a particular memory. As, you write why do you write, you brought a particular block to the cache memory and; obviously, this CPU will modify in the cache memory, but in that case what will happen the information that is present in the mean memory, will be different in form that is present in the cache memory.

And as we have discussed in our first lecture on this topic, there we have seen there are a propertic or inclusion property demands that whatever is present in main memory must be present in the cache memory it cannot be different. So, what will happen in this case and it has been found that number of memory writes is about 15 percent of the total number of instruction, and there are two different approaches used, which is used for the purpose of writing. One technique is write through, in case of write through the information is written to both the block in the cache as well as in the mean memory.

(Refer Slide Time: 36:33)



So, what you do in case of write through, so you have cache memory here, you have your mean memory here, normally you are accessing cache, but you have to perform write. So, as you write in a particular you modify this data, then if this corresponds to a particular main memory location that can be easily found out, you have to modify this as well. So, you have to modify cache memory as well as mean memory; however, the time required to access cache memory is much lower than the time required to access main memory.

So, in that case what will happen, whenever there is a write operation can perform, then you have to access both the cache memory. And the main memory and as a result the time required to perform that write will be much longer, if you had written only in the cache memory, that write operation could have been finished much earlier, but since you are writing in the mean memory it will take longer time.

And more over it generates substantial memory traffic; that means, normally this CPU is accessing the cache memory, so traffic on the mean memory is much less, but whenever you are using write through not only it will take longer time, but it will generate more traffic on the on the mean memory. So, you may be asking, so what you may be telling, so what, let there may be more traffic from the mean memory.

For a single processor system, this is it does not change the situation much; however, whenever we go for multiprocessor having a shared memory, then the traffic on the mean

memory is very important, because different processors will be accessing the mean memory. So, in such a case it is important for each and every processor the traffic I mean on the mean memory between that processor then the memory is less, so that is why this is the this particularly important in the context of multi processor system having a shared memory.

Now, you overcome the disadvantage of write through another technique can be used that is known as write back, now in this case you are writing only in the cache memory, so the information written in only in the block of the cache and an update flag is set. So, what you are doing in this particular case, you are adding additional bits, so in this case you are adding an update flag bit.

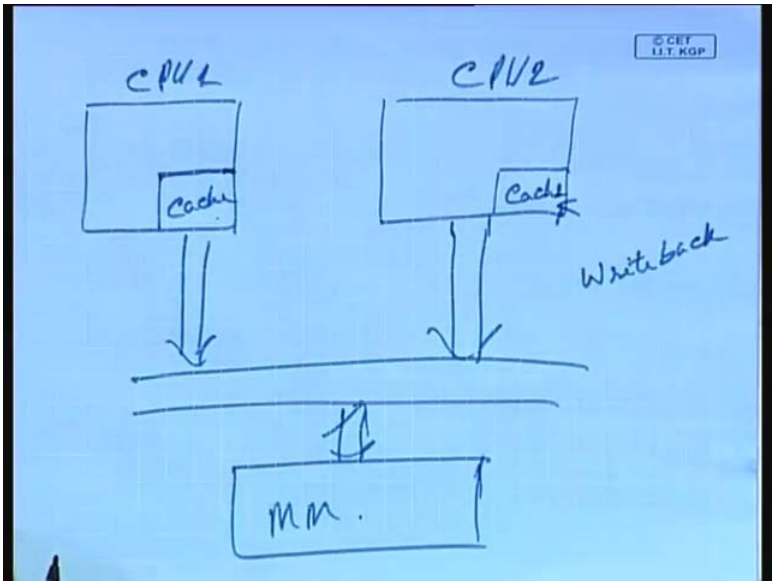
So, update flag bit it was initially 0, now you are setting it to one because this particular data has been modified, so you have not yet modified, this that the main memory has not yet been modified the reason for that is you may have multiple word that is present in the in the in a particular block. So, a particular word is modified that does not mean that I mean, but other words that is present in a block may not be modified, so they may be remaining same, so tag field we will be there and you are using value bit is there and update bit is you are adding.

Now, in this case you are keeping this block, because it has been found that until this block is replaced there is no problem the main memory may be different from the present in the cache memory, because as long as you access from the cache memory it is updated, it is most recent. So, although the main memory content is old, invalid it does not affect you, but whenever a particular block has to be replaced that time, you have to update your main memory, because, so that next time whenever you bring in anything that block from the main memory to the cache memory you get the updated block.

That means, you have, but how do you do that, that is the reason why you are adding an additional bit to update bit. So, at the time of replacement normally, you keep on using because that particular block maybe access many times, before it is replaced, so whenever you replaced it that time, you check the value of the update bit if it is 1, than you update the main memory. So, that is being, that is what is being same mention main memory is modified, when the block is discarded if the update flag is set, otherwise there is no need to change.

So, this creates cache coherency problem in multiprocessor based systems, so this will of course, create another problem, because we have modified in a particular cache memory, but what can happen if you are having multiple processors.

(Refer Slide Time: 42:28)

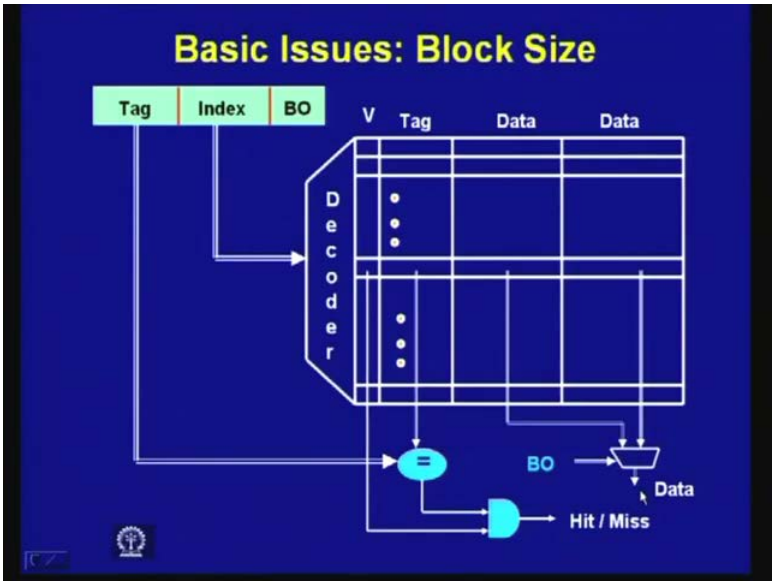


This is one processor, this is another processor and each of these processor can be having its own private cache memory, main memory may initiate through a bus, this is your main memorial, which is being initiated by the processor this is CPU 2 and this is a CPU 1. So, this is your cache, so whenever you are using this is your main memory, so in this particular case what is happening, whenever using write back policy then you have modified let us assume in this block, but you have not modified in the main memory and also you have not modified in this cache memory.

So, what is happening the cache memory contains of the two of the same block corresponding to I mean corresponding to the same block of the main memory, this now becoming different this is a special situation in the context of multiprocessor system. So, in the context of multiprocessor system two cache memory blocks, aligns may will call correspondence to same line of a main memory, but what is happening these two are becoming different. So, that will be that problem will be arise in the multiprocessor system, and we have to deal with this problem in a multiprocessor system.

And this particular program is known as cache coherency problem, and we shall see there are some special algorithm, which is to be used to overcome this cache coherency problem in a multiprocessor system.

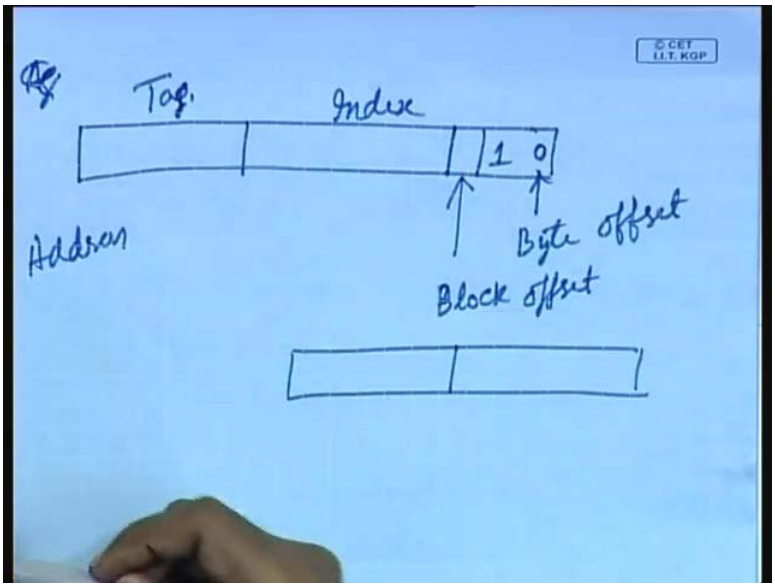
(Refer Slide Time: 44:25)



Another important issue is the block size, as I mentioned the size of a block, so far we have assumed only one word is present, but whenever multiple words are stored, then how it has been done. And particular it has been observed that as the size of the block is increasing, the performance improves why it improves because of the locality of the reference.

And since, you know you can see here, I have got a cache memory, where 2 words are present, the tag bit is common value bit is common index field same index field is used to access ,the data of this block and which is having two words. So, what you have to do, you have the use some bits of that I mean there is a block offset field, so that block offset field has to be used to select one of the two words that is present.

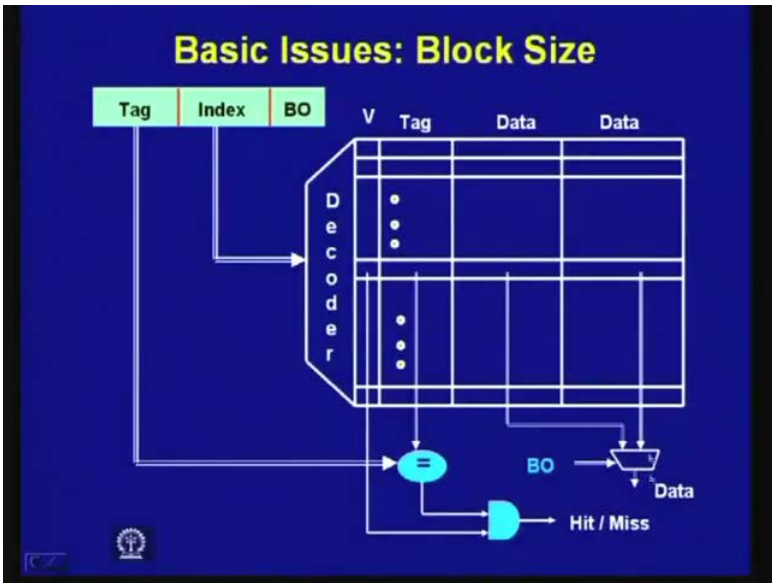
(Refer Slide Time: 45:49)



So, what is happening now you may say earlier you will having 3 fields, this is your byte offset, so considering it has 32 bit, you required 2 bits, and then we had the index field and the tags that is present in tag field this is the address united by the processor, and we are dividing it in this way. Now, what you have to do in the index field, whenever you have got multiple word in the cache memory, that index field is a again have been two parts one is your block offset, so here in a single block you have multiple words.

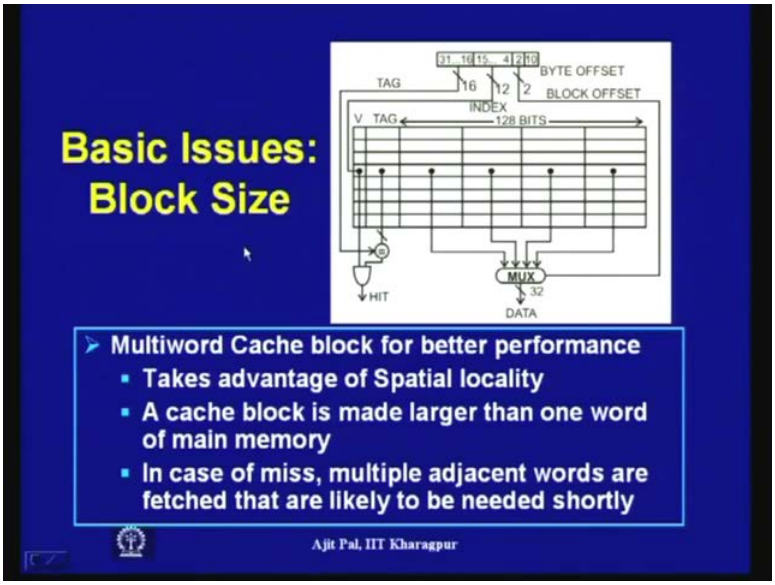
So, these bytes will be used to select the word that is being addressed by this processor, so this size of the index, the number of index bit will reduces, you increase the size of the block. And these bytes have to be used to select the particular word, that is required by the processor and this is what is been shown in this particular diagram.

(Refer Slide Time: 47:21)



That means, a part of the index field I mean that is byte of set, which is shown here is applied to the to the multiplexer, and multiplexer will selects, one of the two words that is present in that block. And that is provided to the processor, because processor will require only one word at a time, the transfer between the processor and the cache has to be in terms of words. And so, only one word will go, and of course the heat means mechanism is identical that the comparison of the tag field, than handling with the value bit those things will be identical.

(Refer Slide Time: 48:10)



So, this is the situation, where you have got you know 4 words that is present in a cache memory, so this shows that there are two bit byte of sets and two bytes for block offset here. So, you can save these true bytes are used to select this particular word from this particular line of this cache memory, and that data is provided. And of course, tag bit, if common and value bit is common as I have already mentioned, so multi-word cache block is used for better performance, and take use for special locality.

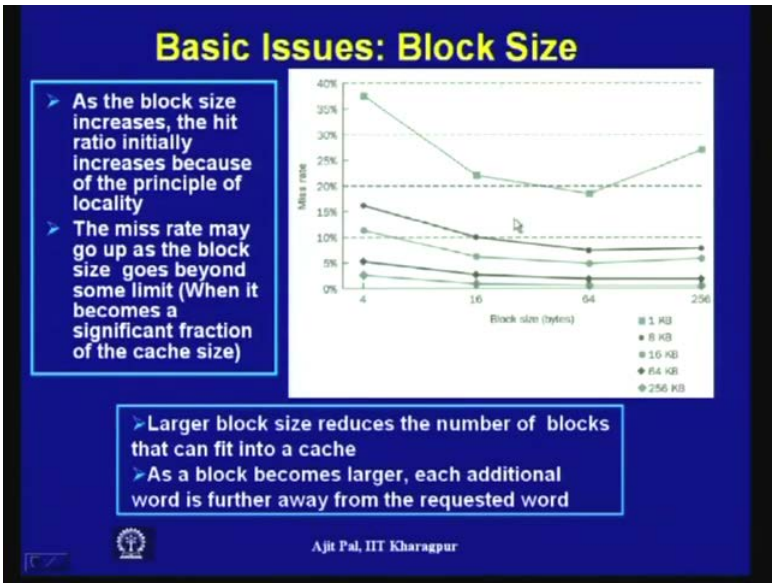
What is special locality, special locality says that, if you use a particular I mean address, I mean you are accessing a memory for a particular memory location, then adjacent memory location will be used in your future, that is the special locality, may be used. And since you are having for different blocks and; obviously, they are in adjacent memory locations, and they are likely to be used in future.

And, so this will takes advantage of special locality, and as a result this improves the performance, so cache block is made larger than one word of main memory. And in case of miss multiple adjacent words are fetched that are likely to be needed shortly, so here there is a the advantages that it will improves the performance; that means, the number of that heat rate will increase, miss rate will decrease, but unfortunately another problem will arise.

What is the problem, in this case multiple adjacent words are fetched and they are likely to be needed shortly; that means, you have to transfer multiple words, whenever there is a miss. So, whenever you have the transfer multiple words from the main memory to the cache memory it will take longer time; that means, we missed time, so whenever there is a miss that time will increase.

Earlier, if there is only one word if it is present in the block, so one word has to be transferred from the main memory to the cache memory and the processor will resume execution of the next instruction, but this will not happen whenever you got multiple words presents in a cache line. You have to transfer all the words from the main memory to the cache memory, so this will increase the miss time.

(Refer Slide Time: 51:11)



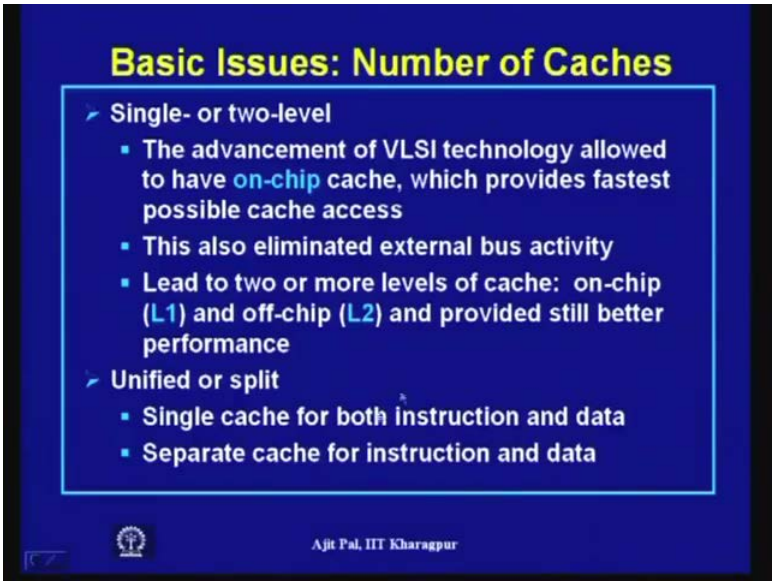
As, the block size increases the heat ratio initially increases as I have explained, because of principal of locality, the miss rate may go up, as the block size goes beyond some limit, when it becomes particularly significant fraction of cache size. So, this particular diagram shows, how they miss rate changes as you increase the block size, initially as you see miss rate decreases, because it increases the because of the principle of locality.

And however, as the block size keeps on increasing, you can see a point reaches when the miss rate keeps on increasing, that happens because it goes up of the block size goes beyond some limit when it becomes significant fraction of the cache size;. That means, when the block size becomes significant portion of the block size the total cache size, then this problem will arise; however, as you increase the size of the cache memory then this problem reduces.

As, you can see this correspondence to only 1 kilo byte of cache memory, whenever you go for 2 kilo byte of cache memory you can see although you are not getting much benefit, but definitely this miss rate is not increasing. So, miss rate is reducing as you increasing the size of the cache memory that is quite natural, but you can see they miss rate is not increasing, as you increasing the block size, because in this case block size is insignificant portion of the total cache memory.

So, this is the larger block size reduces the number of blocks that can fit in to a cache, and as a blocks becomes larger each additional world is further away from the requested word, so because of these two situation, you are getting this type of (Refer Time: 53:19).

(Refer Slide Time: 53:19)



An important issue use number of caches, how many cache memory you will have, single or two level, the advancement of VLSI technology allowed on chip cache which provides fastest possible cache access. Earlier, when the cache memory was introduced, it was not on chip, it was outside the CPU it was off chip cache memory, but as with the advancement of technology, you are able to have more and more transistor on a single chip.

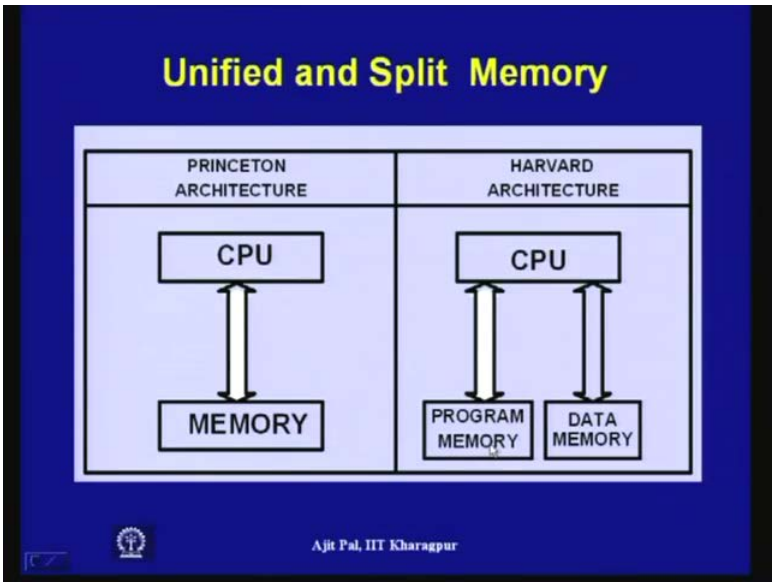
So, in additional to the CPU, you are adding cache as part of the CPU on the same chip and this is known as on chip cache, so this will definitely be faster than the of chip cache. And all present processor are having on chip cache, now the question arises whenever you are having on chip cache will you have another cache, which is of chip or another second level cache and that is why it is called two level, this also eliminated external.

So, it is leads to two or more level of caches, on chip cache L 1, off chip cache L 2, and it provided still better performance, so normally you will have 2 levels of cache memory. And other important issue is unified or split cache, as you know you have to fetch instruction from the memory, as you keep on executing the instruction, and not only that you have to fetch data area and other things from the memory. So, you have the single

unified cache both for instruction and data or you will have separate cache one for instruction and one for data.

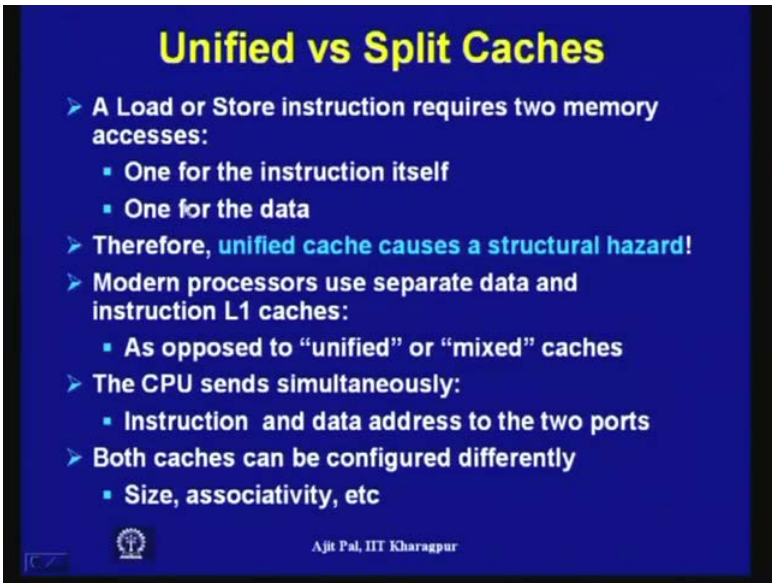
Actually, to support pipelining to overcome structural hazard two separate memories, particularly two separate cache memories are preferable and it is widely used, so separate cache for instruction and data.

(Refer Slide Time: 55:47)



As, it is shown in the diagram, so when the Princeton architecture was introduced as I mentioned earlier, they proposed a single memory both of instruction and data. And Harvard architecture they proposed two separate memories, one for data another for program, but unfortunately when their proposals were evaluated in those days the cost of memories was very high, so they could afford a single memory. So, the Harvard architecture was discarded, and Princeton architecture was accepted, but that is a history now, now all modern processors are having Harvard architecture.

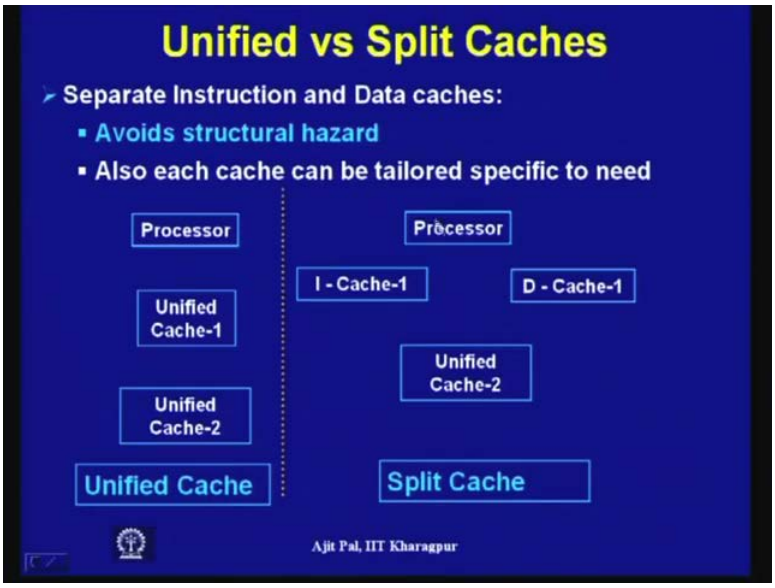
(Refer Slide Time: 56:39)



So, a load or store instruction requires to memory accesses one for instruction and one for data, therefore unified cache causes a structural hazard as I have already discussed. And modern processor used to separate data and instruction caches as opposed to unified or mixed caches, so CPU can simultaneously access instruction and data address to the to the 2 ports, so both caches can be configured differently.

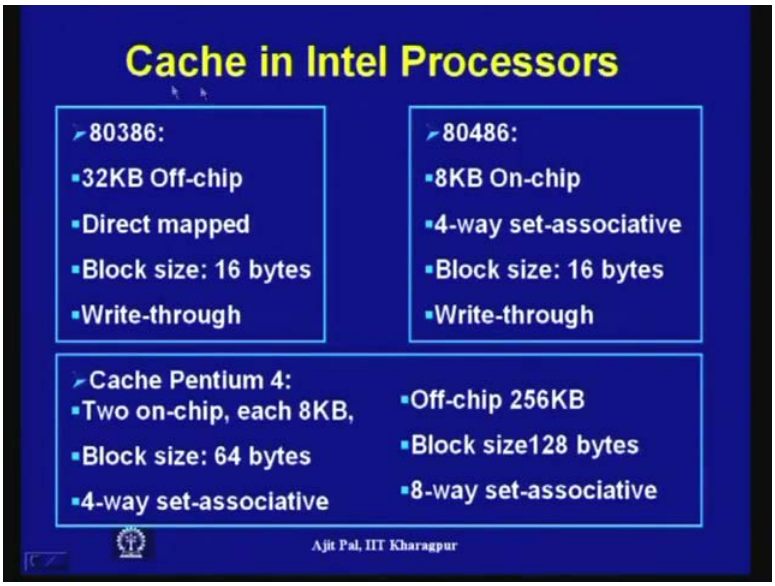
Another aspect is see you have got to separate caches, one is instruction cache, one is data cache, and they can be organized differently, different in the sense say instruction cache can be can used direct mapping, data cache can use say translating mapping. So, that flexibilities provided in modern processors, and you will see that man to separate cache memories are used, you can have two different types of I mean organization, like the size can be different for two different caches associatively can be different as I mentioned like that.

(Refer Slide Time: 57:53)



So, this shows the unified versus split caches in this case you have got a processor, and the unified cache 1, the unified cache 2, so I mean both instruction and data are accessed, but this shows a split cache examples. The first level cache has got instruction cache and data cache separate, but second level cache is unified essentially, for cost consideration. So, you can see it is a combination of unified and split cache, first level cache is separate split and second level cache is unified. So, separate instruction and data cache avoid structural hazard, also each cache can be tailored to specific need as I have already mentioned, they can be organized differently.

(Refer Slide Time: 58:45)



So, let us quickly considered the cache in Intel processors, Intel it has started being used I mean cache memory is being used starting from 80386, it has got 32 kilo bits of chip cache, direct mapped block size 16 bytes. And it is write through and 80486 kilo bits on chip cache, so here it is used both on chip and off chip cache, so first one you can see for 80386, it was off chip cache only.

There was no on chip cache, and here it is on chip cache and it uses 4 way set associative, and here it was direct mapped and block size was 16 bytes and it was write through. Then in case of Pentium 4 2 on chip cache, I mean you have got split cache data and instruction age of 8 kilo byte, block size is 64 bytes and 4 way set associative and off chip cache is 256 kilo byte, and block size is 128 bytes and it uses 8 way set associative map. So, with this let us come to the end of today's lecture in the next class, we shall start our discussion on how you can improve the performance of this hierarchy memory of organizations.

Thank you.