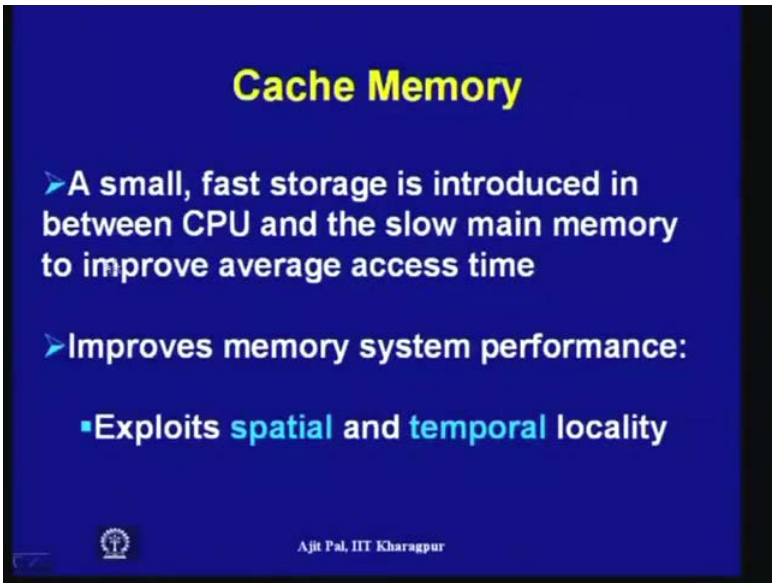
High Performance Computer Architecture
Prof. Ajit Pal
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture - 22
Hierarchical Memory Organization (Contd.)

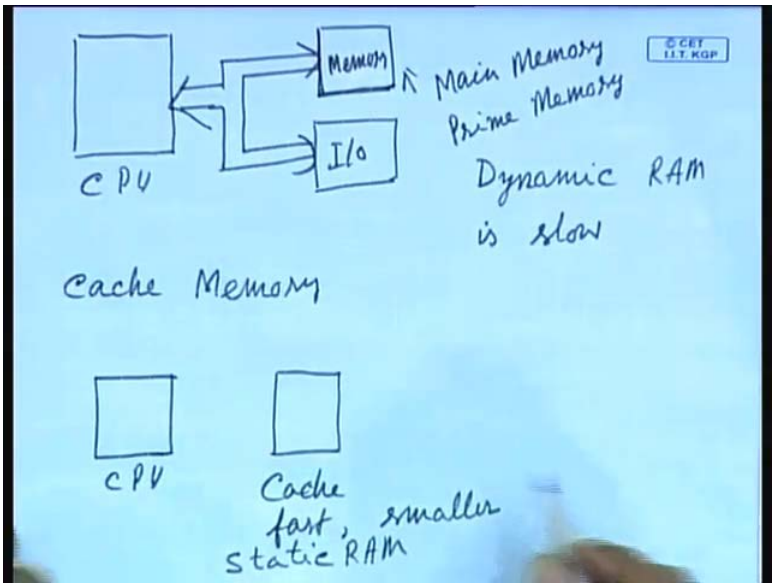
Hello and welcome to today's lecture on Hierarchical Memory Organization. In my last lecture, I had given enough background for hierarchical memory organization. We have discussed the basic characteristics of memory devices, which are used in a computer system. And we shall see how the performance of memory system can be improved, by using hierarchical memory organization, we shall start with the first level of memory hierarchy that is done with the help of cache memory.

(Refer Slide Time: 01:35)



So, what is a cache memory, cache memory is a small, fast storage device that is introduced in between CPU, and the slow main memory to improve average access time. So, this is the basic idea normally as you know, you have got your CPU - Central Processing Unit.

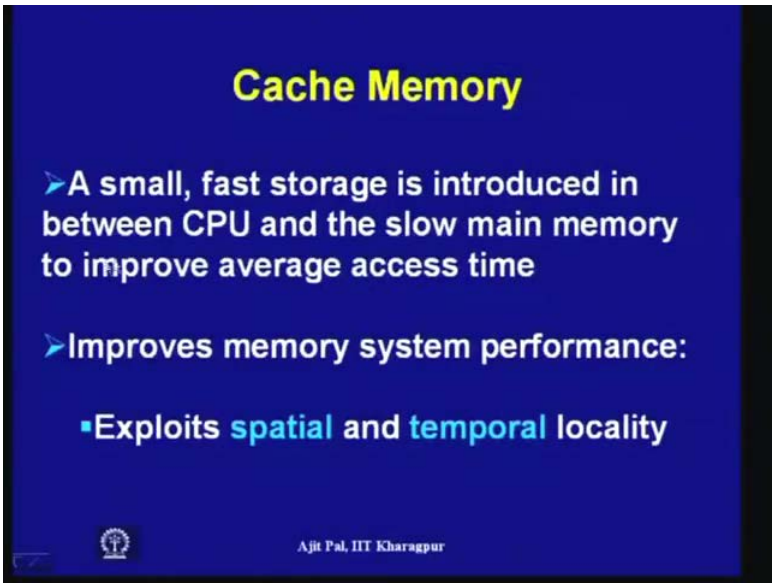
(Refer Slide Time: 02:02)



I have already drawn this diagram; CPU and it usually directly communicates; that means, before the cache memory was invented, the CPU used to directly communicate with the memory and I/O devices. And this memory is known as main memory or prime memory, the reason for calling it main memory or prime memory is that, to execute a program it is very much essential that the program is present in the main memory. That means, for the execution of a program to fetch data it is the memory which is accessed. So, that is why it is called main memory or prime memory, but as you have seen that main memory which is usually realized by using dynamic RAM is slow. And we have also seen the speed of processor is increasing at a very high rate, the speed of dynamic RAM is not increasing at that rate. So, the gap is widening, so we have to breeze the gap and that can be done by using cache memory, so what will be done that CPU will not communicate with main memory directly, first it will communicate with the cache memory which is fast much faster.

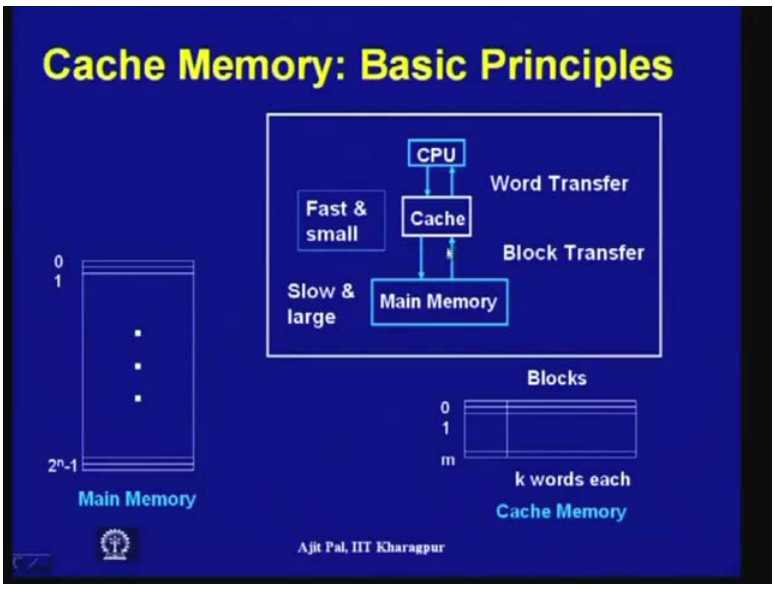
And usually it is realized by using static RAM, static RAM is used to realize this cache memory. So, it is much faster maybe one order of magnitude faster compared to main memory realized by using dynamic RAM, but one point you must understand, the cache memory size is smaller it may be faster, but it is much smaller in size.

(Refer Slide Time: 04:37)



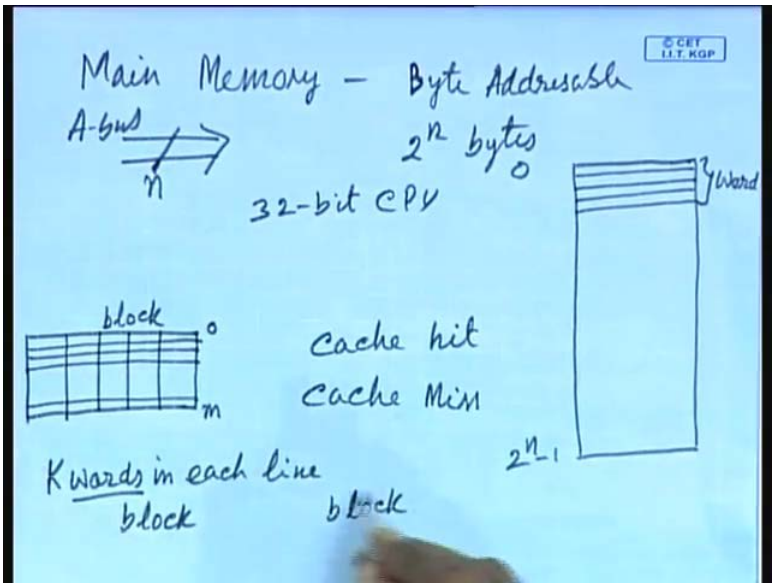
That means, a small portion of the main memory is present in a cache memory, then question will arise how do you improve the performance. Performance can be improved by exploiting spatial and temporal locality that is inherent in program execution, and this will help us to improve the performance, even with a small cache size and shall see how it is being achieved.

(Refer Slide Time: 05:07)



So, here you can here I have shown the main memory, which is at which is byte addressable.

(Refer Slide Time: 05:24)



And let us assume the total size of main memory is 2 to the power n bytes, where n is the number of bits that is being generated by the CPU as address. That means, the number of lines that comes out from the processor is n, this is the address bus and, so you can have a main memory size of 2 to the power n bytes. So, which is shown in this diagram, so main memory is organization is shown here, the starting with 0, 1, 2, so you have got 2 to the power n minus 1 bytes.

However, whenever you organize it in this way, you can see that a word will comprise of several bytes. So, maybe say suppose you are considering 32 bit CPU, in such a case you will be accessing 4 bytes, so this is the word 1, 2, 3, 4, so four bytes will be accessed simultaneously. So, this will form a word and which will be accessed simultaneously; that means, although the unit of reference is byte, but access is done by the CPU in terms of words.

So, that this thing must be clear to you, so for a 32 bit CPU, 4 bytes are accessed, for a 64 bit CPU, 8 bytes are accessed simultaneously and accordingly the memory has to be organized, now let us see what is being done in case of cache memory. So, side-by-side we are trying to implement a cache memory, but cache memory will have much smaller size. Let us assume, the total number of lines usually it is specified in terms of lines, total number of lines is m 0 to m.

And in a particular cache it is assumed that it has got K words in each line, which is called block; that means, you have got one block in one line. So, this is a block I shall discuss about the another bits, so this is a block which comprises several words, maybe let us assume four words. It is not mandatory that a block must have more than one word, only one word may be present in a block, but as we shall see to exploit the locality of references, more than one word is present in a block.

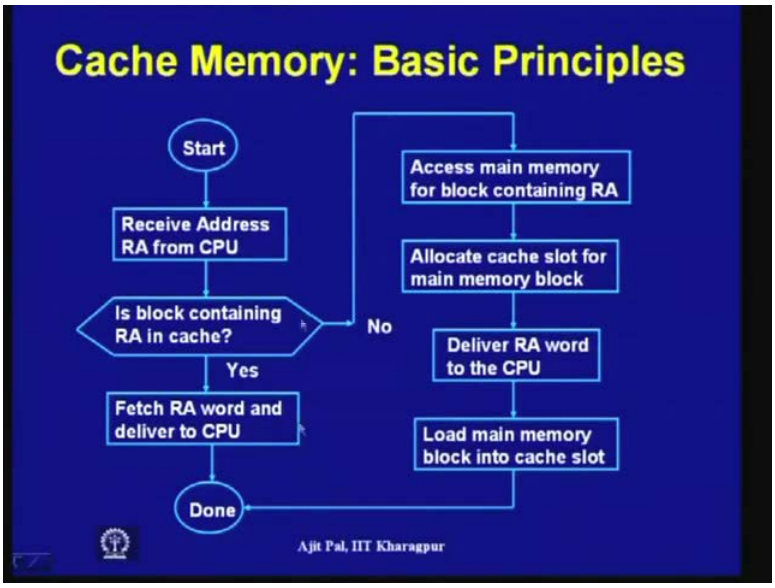
So, few words are present in a block that is what we shall assume, so we have assumed that there are K words in each line. Then we are and as I have already , the transfer between CPU and cache memory takes place in terms of word because, here CPU will want one word at a time. On the other hand, whenever the data which is not present in a cache; that means, it is a since a small portion of the main memory is present in a cache, you cannot expect that you will always find your instruction or data in a cache memory.

So, whenever the data or instruction is not present in the cache memory it is called cache miss. On the other hand if the corresponding block is present, in the cache it is called cache hit, so question arises how do you identify cache miss, and how do you identify cache hit that we shall discuss in detail. But, one point you have to remember that whenever the cache miss occurs, you have to get the word that is required by the CPU from the main memory.

So, from the main memory the blocks are transferred now you can see, the transfer is not in terms of word; that means, whenever there is a miss not just word is read from the main memory, but a block is transferred. So, here the transfer takes place in terms of words, I mean sorry in terms of block, a block if several words are present in a block all the words will be transferred, one after the other although CPU will require only one word at that moment.

That means, one once the block is transferred to the cache memory, only one word will be used by the processor at that moment, and the other words which is being transferred from main memory to the cache will be used subsequently.

(Refer Slide Time: 11:31)



So, the basic operation can be explained with the help of this flowchart, although everything happens in hardware, but for the purpose of explanation the way the cache memory works can be explained with the help of this flowchart. So, first thing that is being done, receive address from the CPU; that means, the CPU generates an address which has been stated as RA - receive address from and that address is applied to the memory system.

And then what has to be done, the memory has to find out whether a block is present in the, whether the block containing RA is in cache. That means, as I told you have to find out rather the memory system has to find out, whether a particular word which is been asked by the CPU is present in the cache or not. So, if the answer is yes then it is very simple, the corresponding instruction or data which is present in the cache memory is delivered to the CPU from the cache memory.

So, here since the cache memory is faster whenever it goes through this path it is quite fast, and the operation is completed. Now, the what happens whenever the corresponding word which has been requested by the CPU is not present in the cache, how that can be identified we shall discuss little later, but let us assume it is not present, then we call that cache miss. So, cache miss has occurred, so what has to be done the memory system will automatically access the main memory or block containing the RA.

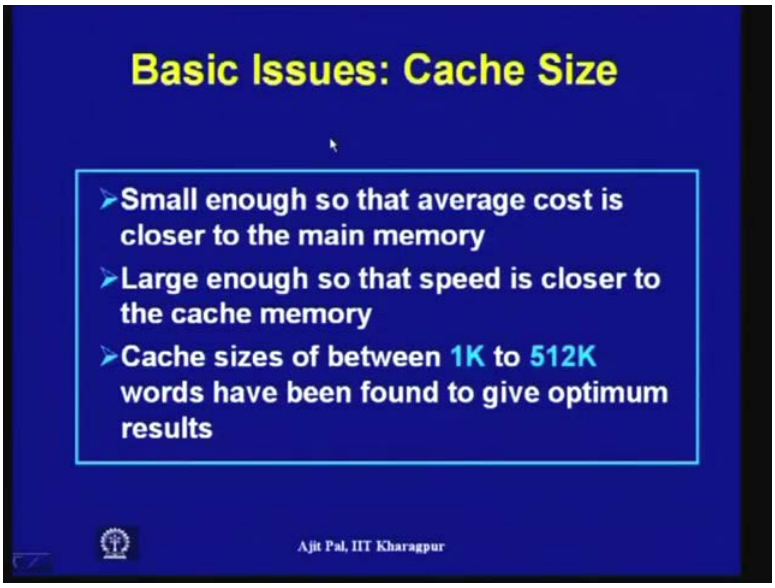
That means, the receive address, then one thing has to be done the block which has been read from the main memory, has to be stored in the cache memory. So, you have to allocate cache slot for main memory block, so once a block is read it has to be stored in one of the blocks one of the lines. So, you have got m lines, in one of the lines it is to be stored.

So, question arise; that means, that is called allocation of a particular line, which line it will be stored, we shall discuss later how that is being identified that say that is another important thing you should understand. You have got m lines, it has to be one of the lines has to be allocated for the word, for the block that is being transferred from the main memory to the cache memory. Then deliver RA word to the CPU, so once you have transferred the block to the cache memory, and stored it in the allocated memory cache slot, the if the requested word can be transferred to the CPU.

And load main memory block into the cache slot, so it will be done together, so whether you load it in the cache slot first or you deliver to the CPU that is immaterial. Because, once you have read it from the main memory, it can be immediately transferred to the CPU. And then of course, it can be loaded into the cache, why these two has been separated, the reason for that is you can see that your bus may be the by width of the bus can be same as the size of the word.

So, if you have to transfer multiple words it will require multiple memory cycles, so which can be deferred if necessary for later time. So, and then that is that is being done, so this is a not cell the way the cache memory works.

(Refer Slide Time: 15:42)



Now, let us consider several issues one important issue is the size of the cache with respect to the main memory, we have seen that small it should be small enough. So, that the average cost is closer to the main memory, so if the cache size is very large then cost will be high, and not only cost will be high whenever you make the size of the memory larger, it will be slower as well. Because, you know if you look into the details of the implementation of cache memory, which is realized by using static RAM there is a decoder portion.

Address decoder that address decoder will be more and more complex, as the size of the memory is larger and that makes it slower. So, it has to be small from two viewpoints, number one the cost should be smaller because, cache memory is costlier, second is it has to be fast. So, from that consideration the size of the cache memory should be small, but another is it should be large enough, so that the speed is closer to the cache memory.

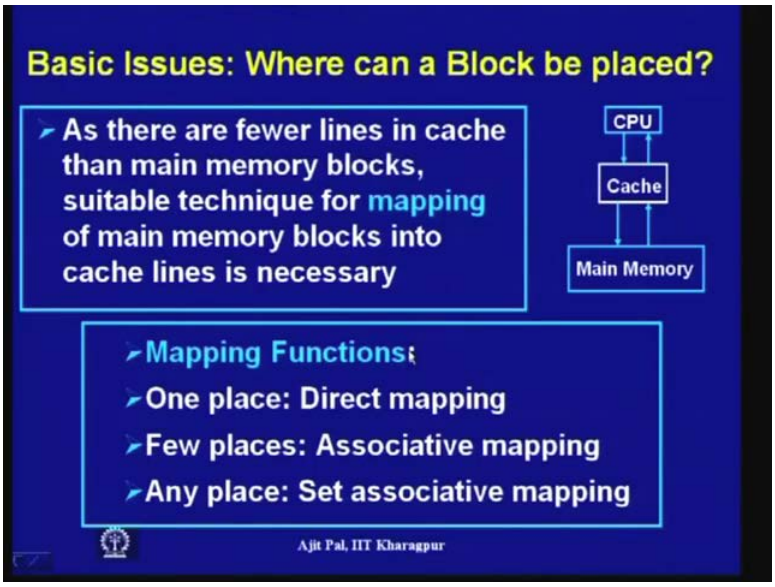
So, this is another requirement, so if we want that the cache memory should be smaller from one, angle another angle is cache memory should be larger why it should be larger. Than most of the time the processor will find the instruction or data in the cache memory, and, so the access time will reduce, the speed will be closer to the cache memory. So, these two are conflicting requirements, so you have to do optimization, so it should not be too large, it should not be too small.

So, the cache size is between 1 K to 512 K, so you may be asking why such a wide range, the reason for that is in the early years when memory was very costly, when cache was introduced first, size of the cache was pretty small. And as the with the advancement of fuel cell technology, the cost of cost as well as performance of cache memory has improved, cost has reduced performance has improved. And as a result, in as the with the advancement of time, the cache memory size as also increased.

So, as a result you will find in modern processors size of the cache memory is not 1 K 1 kilobyte here byte is not mentioned, so we will consider always in terms of byte. So, 64 K or 128 K or 256 K or 512 K these are the typical cache memory sizes, which are present in the modern contemporary processors. So, it has been found that this gives optimum result, so satisfying these two contradictory requirements you have arrived at some compromise, which gives you a optimal optimum result.

That means, it is not very small, not very large and it gives you speed closer to the cache memory, but cost closer to the cache sorry speed closer to the cache memory and speed also closer to the cache memory that is number one issue.

(Refer Slide Time: 19:22)

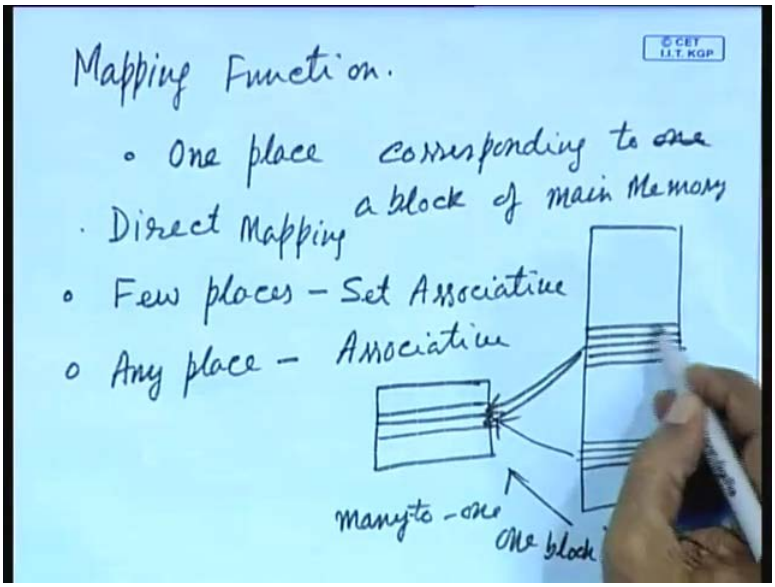


Second issue is where can a block be placed, so I have already mentioned about that question, you see you will be reading a block from the main memory, and you have to allocate one of the cache line to store that block. So, this is the question and that question is answered by what is known as mapping, so as there are fewer lines in cache than main

memory blocks, suitable technique for mapping of main memory blocks into the cache lines is necessary.

So, you have to develop you have to use appropriate mapping technique, and it has been found that the mapping functions can be divided into three basic categories.

(Refer Slide Time: 20:14)



So, here the first one mapping function first one is known as, only one place corresponding to one block of main memory to a block of main memory. That means, so if you read a word from a main memory, say maybe from here and it will always go to a fixed line that is fixed. So, there is a kind of one to I mean, it is not really one to one because, many blocks of main memory will actually map to one of the locations. So, it is many blocks will map to one block, so it is a many to one mapping.

Because, you have got many blocks and each of them although the line is fixed, but to where it will go I mean that is fixed; that means, several such for example, this block may also map to this location, to this particular line. So, but that is fixed that is not changed, so this is called one place is fixed and that is known as direct mapping, I shall elaborate this technique in detail little later.

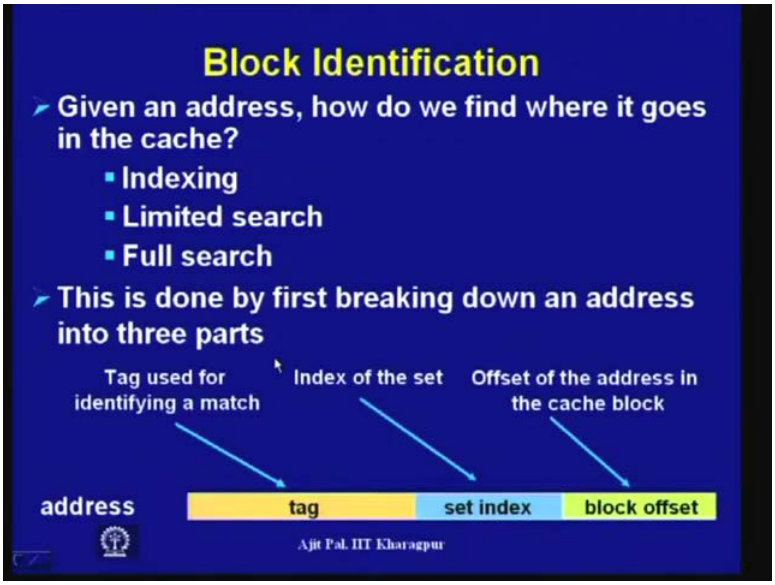
Then second possibility is that you can have few places; that means, there are few alternatives, you are introducing some flexibility. That flexibility is that instead of always storing in a particular line, you can have more than one alternative 2 or 4 or 8

limited number. So, you can store either in this line or maybe there is another alternative line you can store in this line, in one of the two lines, in one of the two places, so you can have few places.

So, this is little more flexibility that is being added, and later on we shall see how this improves the performance, and at what cost. Then the third alternative is anyplace, actually these two have changed places actually, any place will be associative mapping; that means, few places is known as set associative mapping. That means, one of the sets can be used for placing, and last alternative is anyplace that is your associative mapping fully associative mapping.

That means, a particular block from the main memory can be stored anywhere that is known as associative mapping and or full I mean fully associative mapping or associative mapping. So, these are the three alternatives which have been explored and we shall see their advantages and disadvantages.

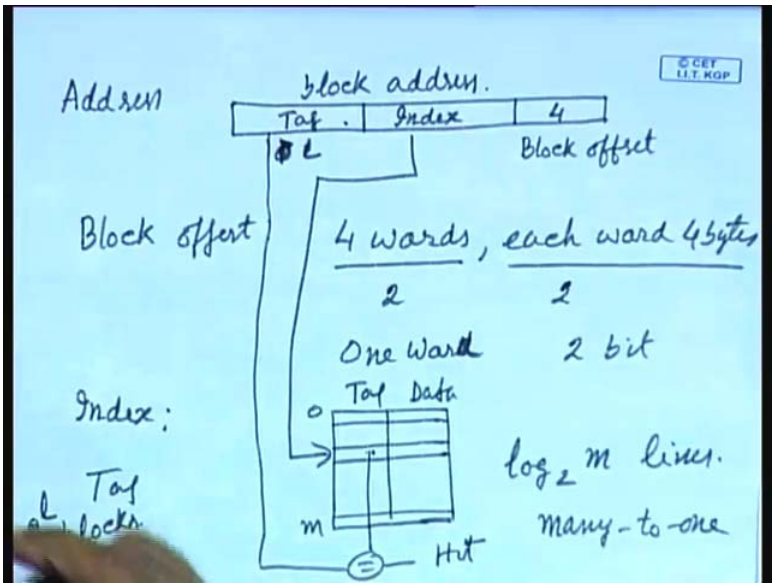
(Refer Slide Time: 24:28)



So, these three techniques that I have mentioned, has led to three possible ways by which you can identify where a block has to go. First technique is known as indexing; that means, there is a fixed place one place for a particular block in main memory, where it can be stored that is done by a technique called indexing. Second is done by limited search that is set associative; that means, some search will be performed within the cache to identify, where it can be placed.

And then full search; that means, to which line it has to be stored by using a full search, it will be a found out and there it can be stored in appropriate place by based on some policy, which I shall discuss little later. So, this is done by first breaking down an address into three parts, what are the three parts.

(Refer Slide Time: 25:43)



So, the address which is generated by the CPU is this, this is the address which is generated by the CPU. Now, this can be divided into three parts first part is block offset, what do you mean by block offset within a particular block, how many bytes are present this will be specified in this part. That means, suppose we assume that a particular block has got 4 words, and each word of 4 bytes let us assume, then how many bits do you require in block offset.

So, for 4 words you will require 2 bit to identify which one of these words, and 2 bit is required to identify which byte of the 4 bytes. So, you require 4 bits, so you will require 4 bits to as block offset, whenever you have got 4 words in the cache, and each word is of 4 bytes. If you have got only one word present, then say one word as I said that is also a possibility, so in such a case this particular bits will not be required, so you will require only 2 bit as part of the block offset.

So, essentially the block offset bits identifies different bytes that is present within a block, now remaining part is used for the purpose of identifying the block. So, this is known as block address, now this block address is present and this block address is again

divided into two parts. Second part is known as index, this is the index question arises what do you mean by index, and how many bits are to be present in the index, actually this index finds out say you have got m lines present in the cache 0 to m.

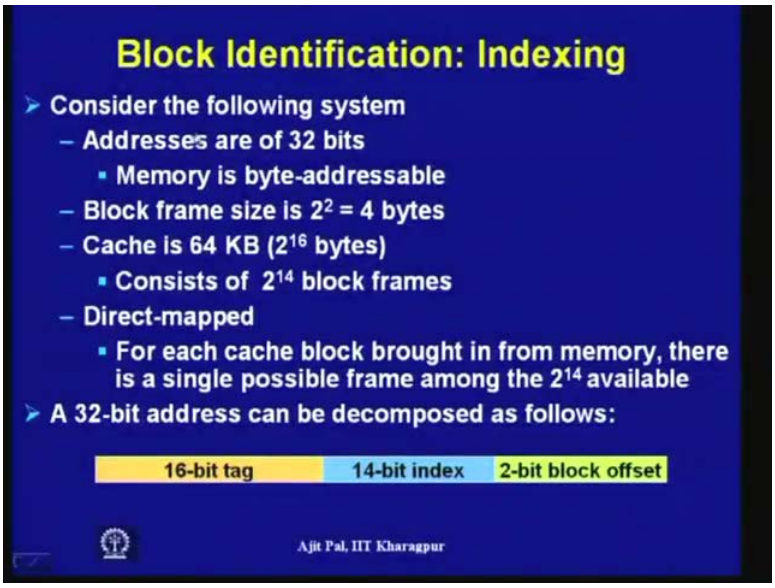
So, how many bits do you require to identify one of the lines, you will require log 2 m lines and that is actually specified by the index. So, index specifies the address part which identifies, which of this line is present in a cache I mean one of the lines, it points to the line where there is a possibility of storing that particular block. So, from the index field it points to the cache memory, now why do you need the third part.

So, you have already identified a particular place in the cache, where a block can be stored, but as I said it is many to one mapping. And there are with the number of bits present here, depending on the number of bits present here, there will be, so many alternatives say if the number of bits present here is say n not n is usually specified the address size. And this can be say k say let it be some number L, so you can have 2 to the power l different blocks, which can be mapped to the cache memory line having the same index.

So, one of the 2 to the power l blocks has to be will be present here, question naturally arises which block of main memory it is. So, that has to be found out by a technique known as, by putting the tag as part of the cache memory, so here you are storing the tag, this value this is known as tag is stored along with data not only data, but tag fields that is the higher order bits are stored in the cache memory. So, what will be done this will be compared with this tag, and if they match then you know that particular block which is reference by the cache memory.

I mean which is reference by the processor is present; that means, cache it will occur if this matches with this tag, on the other hand if this does not match with this tag field then it is miss; that means, this is hit if it is not then it is miss. So, you can see the roll of the three different fields I have discussed in detail, and later on when I shall discuss about the different types of mapping how they are used we shall discuss.

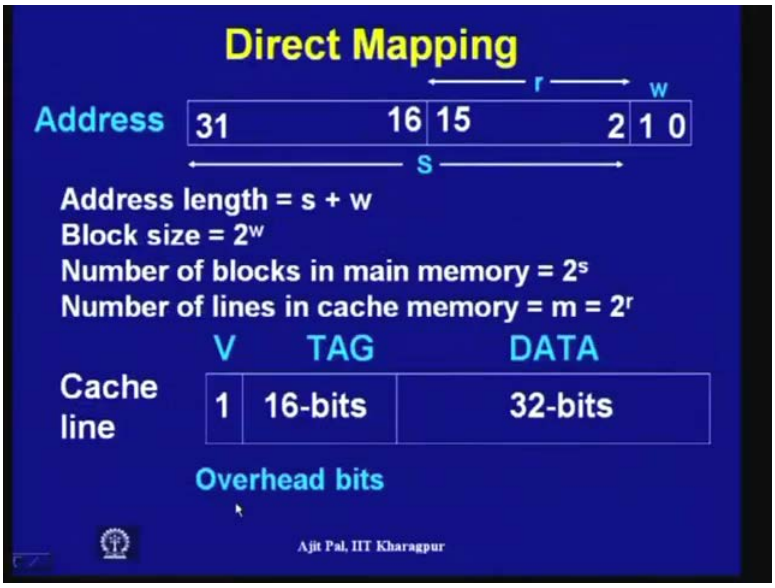
(Refer Slide Time: 32:44)



So, you find let us consider a following system, addresses are of 32 bits which is generated by the CPU, the block frame size is 2 to the power 4 bytes. That means, this is the size of the block, which has been assumed to be of one word let us assume 32 bit processor. So, the cache is 64 kilobyte; that means, you have got 2 to the power 16 bytes that is present in the cache, and since 4 bytes are present in each block, and what will be the total number of lines, which is also called block frames that is 2 to the power 14.

So, for each cache block brought in from memory there is a single possible frame among the 2 to the power 14 available. That means, 2 to the power 14 available this is the size, this is the number of frames or lines which is present, and the we have to compare the tag to identify, whether a particular address that has been generated by the CPU is present at that moment or not.

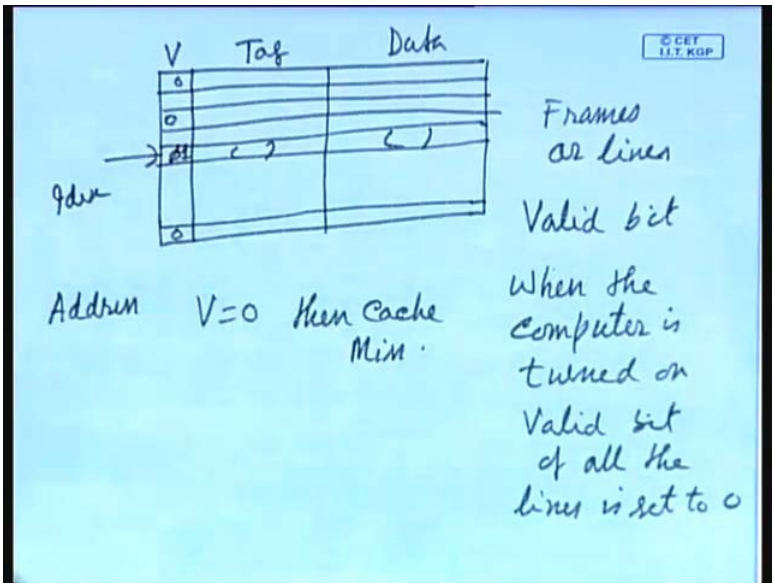
(Refer Slide Time: 34:03)



So, this is being elaborated here as you can see your address is 32 bit, and these two bits 0, 1 is used as byte offset, I mean block offset then r is the number of bits that is used for the purpose of indexing. So, since you have got you have already seen that you require 14 bits for the purpose of indexing, so 2 to 15, 14 bits are used for purpose of indexing, and remaining 16 bits are used as tag. So, the number of blocks in main memory is 2 to the power s that is the number of blocks.

So, in this particular case it is 16 plus 14, 2 to the power of 30 is the number of blocks in main memory, and the number of lines in cache memory is m which is equal to 2 to the power r. And, so this is how and address length is s plus w, s is the block address plus the block offset, so together forms the address length, so you can see we have this is what will be stored in the cache memory, your data will be present, which is 32 bit as you mean that 32 bit processor. Then 16 bits you will require as tag field, oh you see there is another bit is present v what do you really mean by V.

(Refer Slide Time: 35:48)



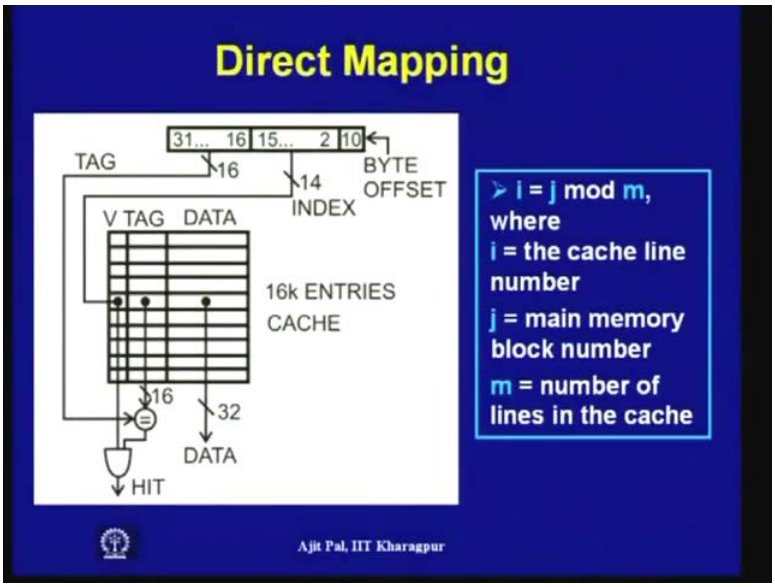
There is a now, your cache comprises three fields first part is data, where useful information that will be required by the CPU is stored. Then as we have seen, we will require to store the tag corresponding to different frames or lines, one tag is required for each line. Now, another bit has been added V that is called the valid bit, why do you need this, why do you need another additional bit, need from that arises need for the valid bit arises because, whenever you turn the computer on your cache is blank, there is nothing no information, no useful data is present in the cache.

So, it is full of garbage, so you have to take into account that situation as well, so; that means, when the computer is turned on the valid bit of all the lines is set to 0. That means, what is done in the beginning, all the valid bits of the line all are turned 0 all are 0. Now, suppose the CPU generates an address and whenever the address is generated by the CPU using the index field it will point to a particular line let us assume it points to this line.

And if the valid bit is 0, which will immediately indicate that the corresponding data or instruction is not present in a cache memory, so; that means, if V is equal to 0 then cache miss. So, what will be done then at that time you will be fetching it from main memory as you have seen, and after you fetch it in the main memory and you store your data here, and store the tag field here then it will set to 1. So, there will be a tag field present here and data present here, once that is stored then the valid bit is 1 is set to 1.

Now, next time whenever if the index field is same, then that time not only this valid bit will be checked. But, this tag will be compared with the higher order bits of the address to check; that means, the tag field has to match, which I shall elaborate in with the help of this diagram.

(Refer Slide Time: 39:24)



As you can see, here you have got 32 bit address these two are block offset or byte offset, which is identical at this moment because, one block comprises only one word. So, that is why byte offset and block offset is same in this case, and this index field is used to point to a particular frame in the cache, so this has been pointed, so what it does it first checks whether the valid bit is 0 or not. If it is 0 then this hit will be 0 it is an and gate, if it is 0 output will be 0; that means, it is a miss, only when it is one then it is hit.

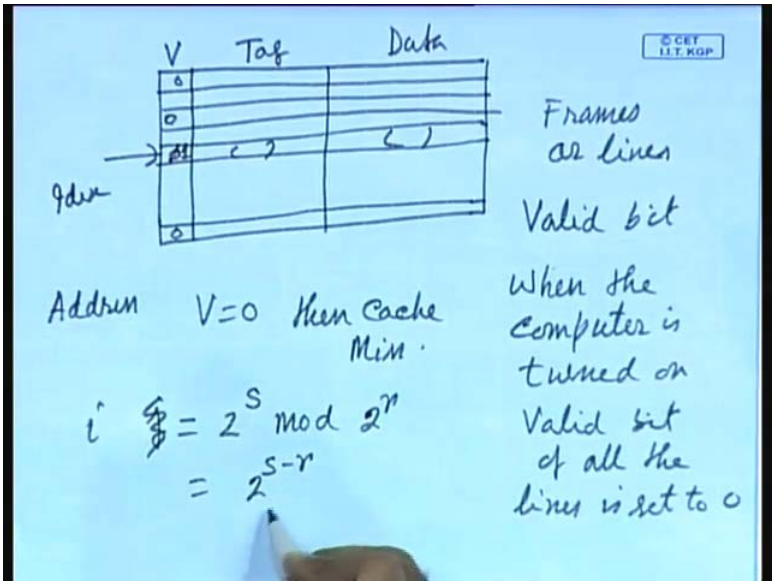
Now, in addition to that it will compare the higher order 16 bit with this tag field, which is the tag field that is being stored at the time of bringing in the block to the cache memory. So, this tag field will be compared with this higher order address lines of the which is known as tag field of the address, and if they are same then this will be one, so the tag has to match, and the valid bit also has to be 1 to get a hit. That means, if the tag does not match, even when the valid bit is 1 it will be a miss.

Because, it is a many to one mapping, so there is a possibility that particular cache line does not corresponding to this address. So, in such a case in spite of the fact it is the valid bit is 1, you will not get hit because, the tag field is not matching, now question arises

you can see the relationship between the number of lines, number of bits which is given here. Where i is the cache line number, cache line number means that the cache line number in the cache memory, how do you find it out i is equal to j mod m, where m is the number of lines that is present in the cache.

Now, it is very easy to do it whenever I mean do it by breaking it up in this manner; that means, you have to take j mod m.

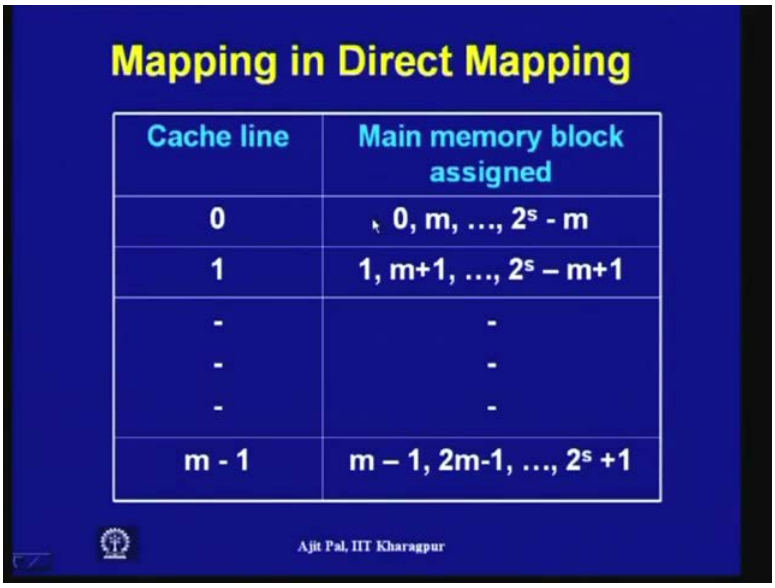
(Refer Slide Time: 42:22)



So, we have seen in our case the j is the number of blocks that is present and we have found out that, that is equal to 2 to the power s, j is equal to 2 to the power s. And on the other hand your that m that mod m is equal to 2 to the power r, so how do you how it can be easily found out. So, this will be equal to 2 to the power s minus r, so that is actually the value of j main memory I mean sorry that is how you can find out this is i, i is equal to 2 to the power s mod 2 to the power s.

So, this i is found out that index is found out by this, so this will be equal to this by this, so that is how you can get this value, and this is the relationship and i is equal to j mod m that is used for the purpose of direct mapping.

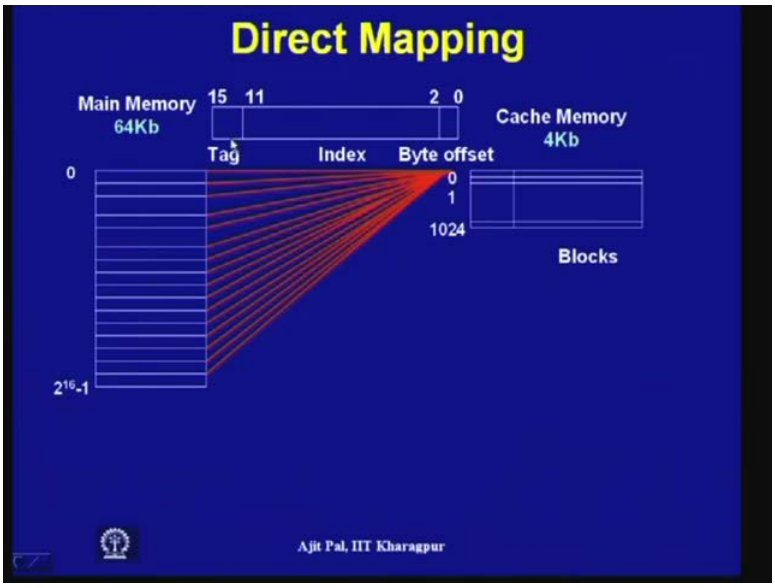
(Refer Slide Time: 43:42)



Now, you can see here how the mapping in the mapping takes place in direct mapping, as I have told that it is a many to one mapping. So, you can see a single cache line is being mapped by, so many main memory blocks, so these main memory blocks can be assigned to this cache line 0'th, m'th then 2 m'th, 3 m'th in this way 2 to the power of s minus m.

Similarly, the first cache line can be mapped by 1 plus, m plus 1, plus 2 m plus 1 in this way it will go up to 2 to the power of s minus m plus 1. And the last line m minus 1 line that will be mapped by that m minus 1'th block of the main memory or 2 m minus 1'th block of the main memory, and in this way it will go up to 2 to the power s plus 1. So, you can see this is how the it takes place in case of direct mapping.

(Refer Slide Time: 44:46)

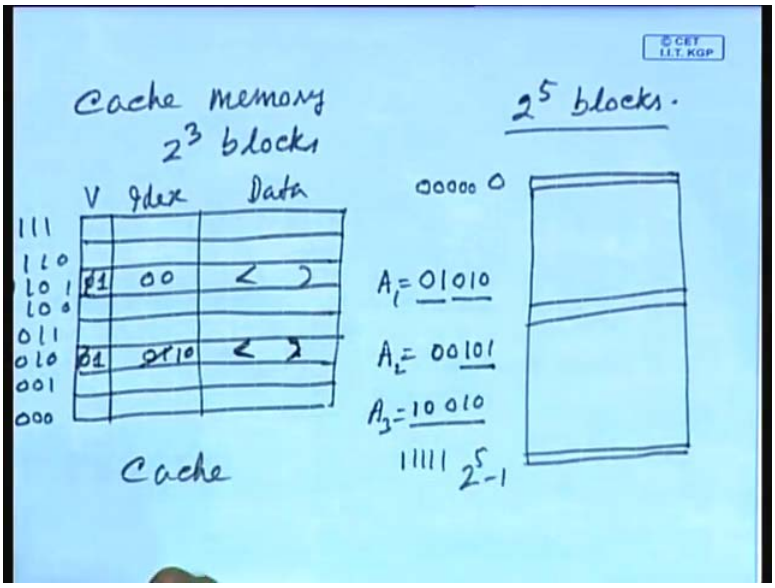


Now, let me elaborate this particular thing has been shown in this example you can see, this is the different main memory blocks which are mapping to a single cache line. So, in this particular case, you have got 16 kilobyte of main memory and 4 kilobyte of cache memory. So, you have got 4 kilobyte of cache memory and 16 kilobyte of cache main memory, so you can see accordingly the byte offset is 2 bits, and 14 bit is required to I mean that is the total number of bytes in the cache memory because, it is 4 kilobytes.

And out of which 12 bit is required for the purpose of 10 sorry you require 0, 1 10 bits because, it is 4 kilobyte; that means, you will have 1 K lines. So, 1 K lines for addressing purpose you will require 10 bits, so 10 plus 2 to 12 and you have got only 4 bits for the tag field. So, what does it mean that 4 bits in a tag field means 2 to the power 4 different blocks of the main memory, can map to a single cache line.

So, you can see this is 16 different blocks of the main memory can map to this, and that can be found out from the tag field. Tag field identifier which one of these block is present.

(Refer Slide Time: 46:37)



Let me explain this with a much simpler example suppose your main memory has got 2 to the power 5 blocks let us assume, you have got 2 to the power 5 blocks, and your cache memory has got only 2 to the power 3 blocks. So; that means, you have got 8 lines, so 8 lines are present here, so this is your cache and your main memory has got 0'th and 2 to the power 5 minus 1, this is the I have considered block I have not stated in terms of words or bytes, so it is in terms of blocks only. So, 1 of the 2 to the power 5 blocks will be here, now it is divided into 3 fields V index and then data.

Now, suppose the first an address let us assume, the address here the index fields will be start with 0 0 0 and it is 1 1 1, and here it is 0 0 0 0 0 and it is 1 1 1 1 1 five lines. Now, suppose the block at this that is generated by the processor is 0 1 0 1 0, this is the address which is generated by the processor. Now, where it will search, it will search where it is 0 1; that means, this part it will be done, so this part will be used this will be used these 3 bits will be used for the purpose of indexing. So, this is 0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 then 1 0 1 1 1 0 and 1 1 1.

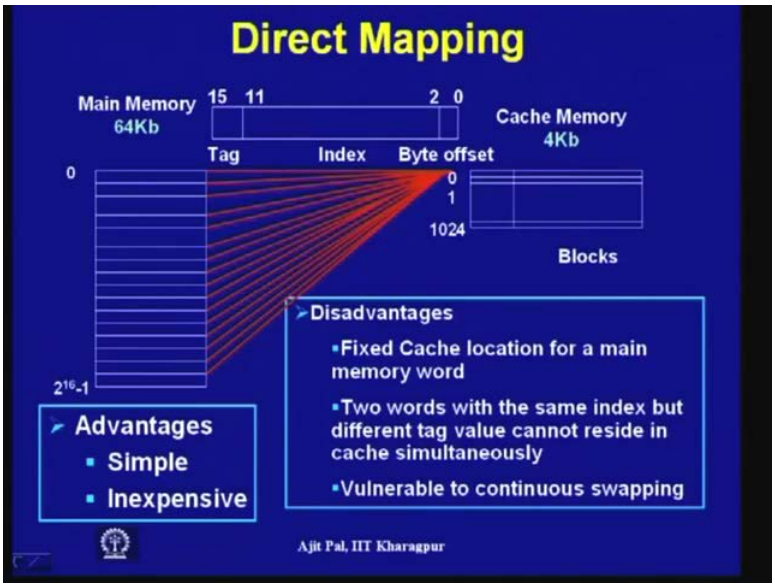
So, 0 1 0 it will point to this indexing will be done here, so once this address is generated it will search this line indexing is done by this field. So, this bit will be initially it was 0, now it will be set to 1 and you will write 0 1 this is the tag field, in this and the corresponding data that is present in this location 0 1 0 1 0 will be stored here. Now, suppose subsequently the CPU generates another address that is A is equal to 0 0 1 0 1.

So, in such a case it will in index bits will sorry this index bit will point to 1 0 1 1 0 1 is this 1.

And then this 0 bit will become 1 and it will write 0 0 and corresponding data will be stored here. Now, suppose the address that is being generated next time, third address that is being generated is 1 0 0 1 0, so 1 0 0 1 0 address has been generated 0 1 0 is matching with this. So, it will point to this now you can see, this is this bit is 1; however, this is 0 1 and this is 1 0, so at this moment there will be a cache miss, in spite of the fact some data is present here, but it is not matching with this address, the third address being that is generated.

And, so this will be a miss, and this will be replaced by 1 0 at this moment, and correspondingly the data which is being stored here will be replaced. So, this is how it will work, so I have illustrated with a very small main memory and cache memory. So, in real life processors; obviously, the number of bits that is present for I mean that is used for addressing, indexing and number of bits in the tags will be much larger.

(Refer Slide Time: 51:28)

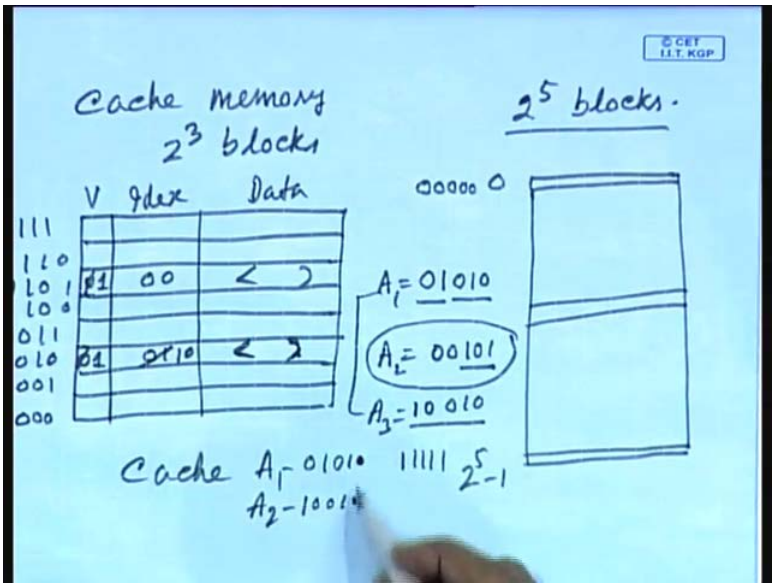


Now, this is how it is done in direct mapping and as you can see it is quite simple, directly from the address bits you can find out you can separate out the index field, which will be applied to the cache memory. Then composition can be done, I mean with the tag field and by checking the valid bit field, it will identify whether the hit is there or

it is miss. But, this direct mapping has a number of disadvantages, the various disadvantages are number one is fixed cache location for a main memory word.

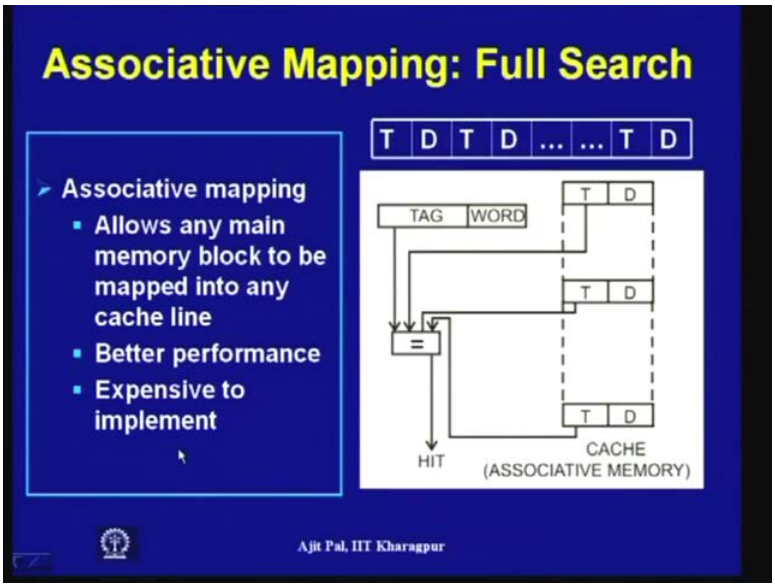
So, as I have already mentioned that direct mapping gives you a fixed location for a given main memory block. So, for a given main memory block this a particular line is fixed, so two words with the same index bit index, but different tag value cannot reside in cache simultaneously. So, this is a serious restriction because, only one can be stored. So, two words with the same index, but different tag value cannot reside in the cache simultaneously, what does it mean; that means, for example, these two words A 1 0 1 0 1 0 and 1 0 0 1 0 these two are generated one after the other.

(Refer Slide Time: 53:09)



If it is a direct mapping each time say suppose alternately this is being generated, each time there will be a miss; that means, since two locations two main memory having the same index field, cannot reside simultaneously this will always lead to a miss. So, this particular restriction can be overcome by using other type of mapping as we shall see. So, as I have explained with the help of this example, this is vulnerable to continuous swapping, so continuous swapping will take place in such a case.

(Refer Slide Time: 53:57)



And this can be overcome by using other type of mapping, like say associative mapping where it will do full search. So, this is another extreme in the earlier case our restriction was that only one address can be used, one frame or line can be used to store. Now, in this case the extreme is you can store, any block in any line that is your associative mapping. But, in such a case what you have to do, you have to compare the tags of all the lines simultaneously, with the tag that is being read that is being that is coming with the processor coming from the CPU.

So, you have to do what is known as parallel search; that means, you have two search in parallel in all the cases. So, you can see although I have shown only one comparator you will require, the number of comparators will be equal to the number of lines or number of frames that is present in your cache memory. So, this associative mapping allows any main memory block to be mapped into any cache line, so this is the flexibility your achieved.

But, this makes this cache very costly and this type of cache is known as Content Addressable Memory or CAM. Because, you are using a part of that address to be stored in the memory, and used for a checking and; obviously, this will give better performance and, but it is extremely expensive to implement. So, we find that the we got two extremes, direct mapping and fully associative mapping, so both of them have their own advantages and limitations.

Now, in my next lecture I shall discuss about another technique, which is known as set associative mapping, which tries to achieve best of both the words. That means, good features of direct mapping, and good features of fully associative mapping, it will try to incorporate in that particular technique that is known as set associative mapping, which I shall discuss in my next lecture.

Thank you.