Digital Computer Organization

Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

Lecture No. # 18

Cache Memory Architecture

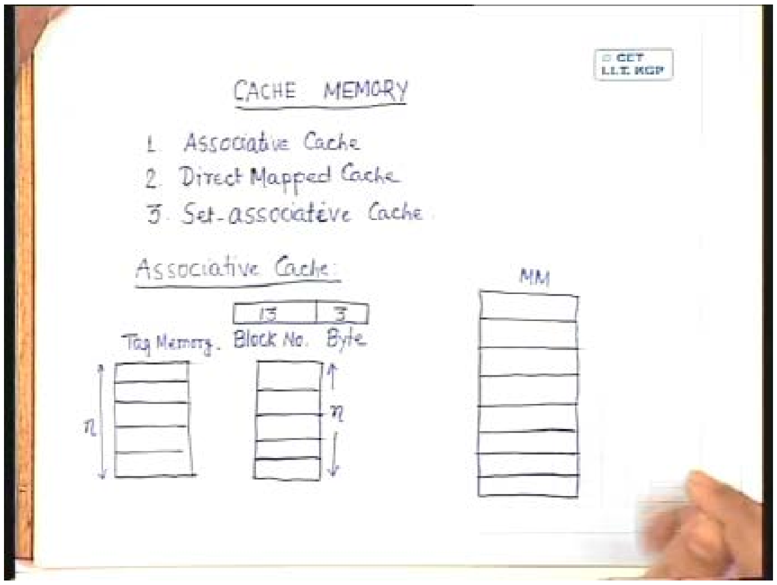
So, today we will discuss about the cache memory organization. Till the last class we have seen that main memory is organized either in the form of pages or in the form of segments or if we want to take advantage of both, the segmentation as well as paging in that case the main memory organization that is used is paged segmented memory organization. And we have seen that what are the advantages of different schemes in case of paged memory management or segmented memory management or in case of paged segmented memory management. However if you think in depth, you will find that whichever memory architecture you use, for accessing any particular memory location that means whenever the CPU generates the address of any memory location. Firstly, using that address you have to check either the segment table or the page table or the segment table and page table both incase of paged segmented memory organization before you actually reach the physical address in the main memory.

So, naturally the question comes that if we store the page table or the segment table or both in main memory itself in that case for every access to the main memory, we have to have one or more additional memory access before we get the actual physical address of the memory location, the data or the instruction from which is actually required by the CPU. So effectively the memory access in such cases will be very slow. So we have to think of an alternative where we have can have faster access to memory and that is what is given by the cache memory. In case of cache memory you get advantage in two ways. Firstly the cache memory is implemented using the static ram, unlike the main memory which is implemented with the help of dynamic ram.

The reason being, in any system we desire that the amount of main memory should be quite large compared to the cache memory then comes the cost involved. If we want to implement everything with the help of static memory, the static memory is very costly compared to the dynamic memory. So in a system if we want to have say 512 megabyte of main memory and everything in the form of static ram then the system cost will be very high. So instead of going for that what is done is when you implement the main memory, you use the dynamic ram to implement the main memory whereas you implement the cache memory which is much much smaller in size compared to the main memory and the cache memory can be implemented with the help of static ram. So access time in case of static ram is much less compared to that in case of dynamic ram but the static ram is costlier than the dynamic ram.

So firstly you get advantage because of the fact that cache memory is implemented with the help of static ram which is faster compared to dynamic ram and secondly you get speed advantage because of the architecture of the static ram or the architecture of the cache memory. So let us see what is this cache memory and what is its architecture? When we talk about cache memory, the cache memory can have three different kinds of architecture.

(Refer Slide Time: 00:04:35 min)



The first kind of architecture we can have is an associative cache memory, second kind of architecture that can be used is called direct mapped cache and the third kind of the cache memory that we can have is called a set associative cache. So firstly we will consider what is this associative cache. The concept of cache memory is similar to the paging concept that we use in case of the main memory. We have said that when we discussed about the demand paging or virtual memory management, that only the page which is currently active if that resides in the main memory then it is sufficient to execute the program. The other pages can be brought from the secondary storage to main memory on demand or as and when it is required. So the name came as demand paging or virtual memory management. When you talk about cache, the concept is something similar.

Here the main memory is divided into a number of blocks may be a page or a segment will be divided into a number of blocks. So whenever we want to transfer any thing from the main memory to the cache memory, the entire block will be transferred to the cache memory, say block size may be say 8 bytes or 16 bytes or 32 bytes and so on. So naturally the block size is much less than that in case of a page or the size of a segment. So this block is a superimposed on the paging. So may be a particular page in the main memory is divided into a number of blocks. This block division is actually done to enhance the caching, to facilitate the caching option.

Now whenever the CPU generates an address, we can say that the address is actually divided into two fields. One field will be the block number and the other field will be identification of the byte within the block. So as I said that whenever I want to transfer anything form the main memory to the cache memory, one of the blocks or the entire block will be transferred to the cache memory. So accordingly the cache memory will also have a number of blocks where the block size is same as that in the main memory. So if we say that every block will contain 8 bytes then in the cache memory also we will have a number of blocks where every block will be of 8 byte. So that whenever I transfer any block from the main memory to the cache memory, the entire block can be transferred to one block in the cache memory. And along with this every block in the main memory has to be identified by this block number, the block number field.

So if we assume that we have a machine whose address line is 16 bits then I want that every block should be of 8 byte that means out of though 16 bits, 3 bits will be used for byte identification and the remaining 13 bits will be used for block identification. So using these 13 bits you identify one of the blocks within the main memory and using this 3 bit address, you identify a particular byte within that 8 byte block. Now whenever I transfer a particular block from the main memory into one of the blocks in the cache memory and I want to search for a particular byte in the cache memory whether the requested byte is present in the cache memory or not, in that case what I have to see is I have to check for this block number whether a block having this particular number which is given by this 13 bit address lines is present in any of the cache blocks or not which means that in the cache memory, I have to have another fields or some additional locations which will contain this block number.

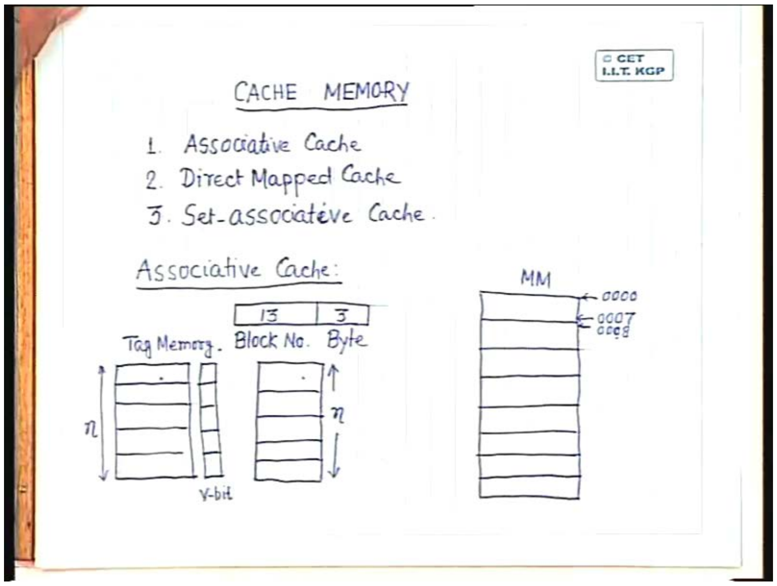
So if I have say n number of blocks in the cache memory, I have to have n number of additional entries where every entry in this particular case will be 13 bit. So in this case I will have n number of additional entries where every entry will have 13 bits and in this case, this 13 bit or block number is called the tag field and the part of the memory in the cache architecture which contains that tag field is called the tag memory. See what I have said is I am assuming that I have a machine which is 16 bit address which gives 16 bit address. So if I have a 16 address in that case with the help of the 16 bit address, I can address a main memory of maximum of 2 to the power 16 that is 64 kilobyte. This 64 kilobyte of main memory is divided into a number of blocks were every block of 8 byte that means I can have 8 k number of blocks where every block will contain 8 byte, so total gives you 64 kilobyte.

Now suppose the main memory generates, suppose the CPU generates a physical address of say some physical address. The physical address is given by 16 bits of the main memory. Now first we will try to check whether that the data of that particular address is available in the cache memory or not. So if I want to see whether that data is available in the cache memory or not, how do I check? I know that the data will be available in the cache memory in the form of a block. A particular byte the block to which that byte belongs, the entire byte should be available in the cache memory, either the entire byte or none. So we try to check that whether the block containing that byte is available in the cache memory or not. So how do I check? I check the block number as I said that the main memory address which is generated by the CPU is divided into two fields, the block number and the byte within the block. I will check whether this block having the particular number as given by the address lines is present in the cache or not. That means along with this cache memory which contains the block, I have to have additional memory which will contain the block number.

So if I want to load in some cases a zeroth block from the main memory in the cache, the block number of the zeroth block will be zero that means all these 13 bits will be 0. So if that zeroth block is loaded in the first block in the cache memory then the first tag memory locations will contain value 0. So you see how it is coming. The first memory address here is 0 0 0 0, if I put it in the form of hexadecimal number. the last byte that is eighth byte within this block will have a physical address of 0 0 0 7 and block address is 0 0 0 because you find that out of this, this is a

16 bit address, out of this the more significant 13 bit gives you the block address. The least significant 3 bits gives you a particular byte within this block. So that is how these fields are been broken. When I come to the next block, the starting location of the next block will be 0 0 0 8. Is it okay? So this is the block address of the second block. Whenever this block is to be loaded into cache memory, the data contained in this block will be in the cache part and the block address will come to the tag memory. That means for every block in this cache memory, I have to have a corresponding tag memory. The number of bits in the tag memory will be same as the number of bits in the block number field of the address. Sometimes each of these blocks in the cache memory is also called a cache line. A cache line means a block of the main memory. So these 8 bytes taken together is also called a cache line, it's a block in the cache or it's a cache line.

(Refer Slide Time: 14:10)



CACHE MEMORY

1. Associative Cache
2. Direct Mapped Cache
3. Set-associative Cache.

Associative Cache:

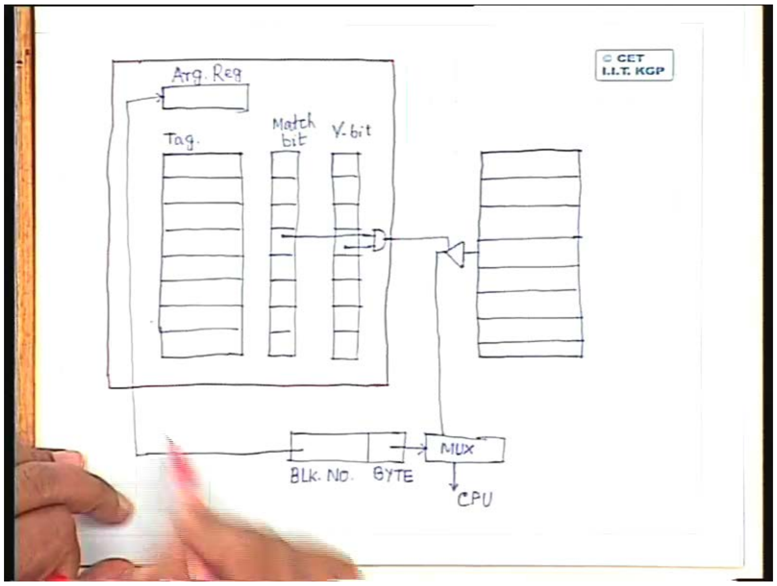
Now in addition to this, I also have to have an information that whether the content of a cache line is valid or not. That means I have to have what is called a valid bit. So whenever you place any block of the main memory into any of this cache blocks, then immediately what I have to do is I have to fill up the tag memory with the block address and at the same time, the valid bit of the corresponding cache entry has to made equal to 1. [Conversation between Student and Professor – Not audible (00:15:41 min)] it will be clear later. So initially you find that when there is nothing in the tag memory, all the valid bits will be equal to 0. Now why this valid bit is needed is because of the fact that tag memory can have any address value.

Now following some address generated by the CPU, if I want to check whether there is any tag memory location which matches with this block number or not because of the garbage value is contained in that cache tag memory, may be I will get false match. But whether this is actual match or a false match to identify that I need this valid bit. So initially all the valid bits will be equal to 0 indicating that the cache memory does not contain anything that means even if I get a match in the tag memory field that is invalid. So only when I place something in the cache memory and set the tag memory field then only the corresponding valid bit will be made equal to 1.

So, now what is the checking concept that we have to do? Whenever the CPU generates an address, I want to see whether the corresponding byte or the block containing the corresponding byte, the required byte is available in the cache memory or not. For that what I have to do? I have to check whether there is any bit in this valid invalid bit array is equal to 1 and the corresponding tag field matches with the block number or not. And here any block in case of associative cache, any block from the main memory can be placed in any free block in the cache memory or any free cache line in the cache memory.

So if I want to check then there are two concepts, either you can go for checking sequentially that is you take the block number, try to see whether the first valid invalid bit equal to 1. If the first valid invalid bit is equal to 1 then you check whether the tag memory location matches with the block number. If not, if they don’t match come to the second location. Again if the second field, second bit in this valid bit array is equal to 1, you check this tag memory content whether it matches with the block number or not that means I can go for a sequential checking. And if I go for a sequential checking, in that case obviously if there are n number of entries of n number of blocks in the cache memory, in the worst case I have to perform n number of comparisons. And if I go for that, obviously my main purpose why I am going for this cache memory is to enhance the memory access time, to reduce the memory access time that itself will be lost. So I cannot go for this sequential changing. So what I have to do is I have to go for what is called parallel checking. So instead of going for software checking, the entire checking has to be done using hardware. So how do you do that? That is what is cache memory architecture.

(Refer Slide Time: 00:18:55 min)



So I will put the cache memory like this. On this side let me have a number of cache lines, so I will call this as cache line or it is a block in the cache memory, so these are different cache lines. For every such cache line I have to have a tag memory field. For every tag memory field or for every cache line, I have a corresponding valid invalid bit. So this is valid bit, this is tag field. Suppose the CPU generates an address and as I said that address will be broken into two components, block number and byte identification within the block. Now in this architecture we have a resistor which is called an argument resistant. The block number which is obtained from the CPU generated address is placed in the argument register. Then from the argument register, the argument register is hardware wise connected to each of this tag locations where they are compared parallelly. So the content of the argument register will be compared with the first tag location, it will be compared with the second tag location, it will be compared with the third tag location and so on and this comparison will be done in parallel.

Accordingly depending upon the comparison result, I will have a bit, I will generate a bit which is called a match bit. So corresponding to every tag entry or corresponding to every cache line, I have to have a match bit so this is what is match bit. So when the content of the argument register is compared with different tag locations any particular location, any particular tag array which matches with this content of the argument register, the corresponding match bit will be equal to 1. Now if the match bit is 1 and the corresponding valid bit is also 1, so valid bit 1 indicates that the tag field contains a valid block address. Valid bit 1 indicates that the tag field contains a valid block address and the match bit will indicate that this tag address matches with the block number as specified in the memory address generated by the CPU. [Conversation between Student and Professor – Not audible ((00:22:45 min))] argument register contains the value form the block number. It is just this block number which is placed in the argument register.

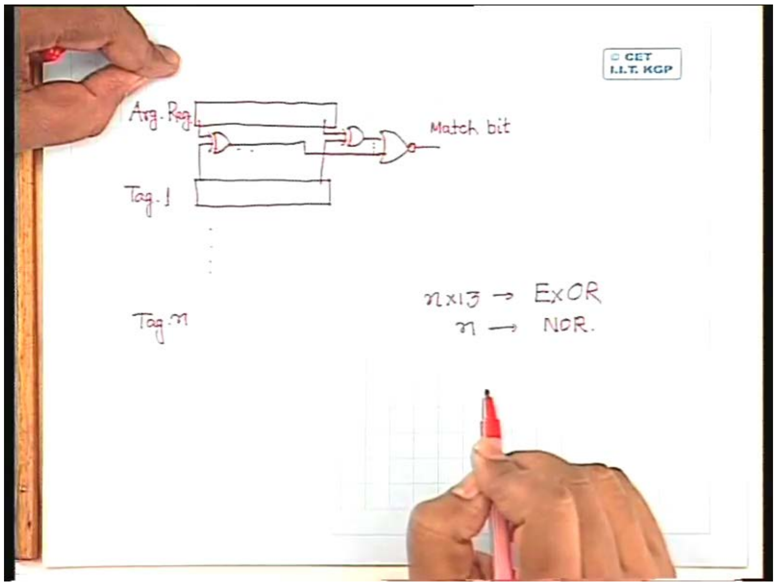
So assuming that this particular match bit has become equal to 1 that means the content of this tag memory location has matched with this block number and the corresponding valid bit is also equal to 1. So what I do is I simply AND these two together. So whenever this match bit has become equal to 1 that indicates that the corresponding cache line, say this one it contains the block containing the requested byte. So what we do is you simply pass this through a buffer and the buffer is to be enabled by this output of the AND gate. So this entire cache block or the entire cache line will be available at the output of this buffer only when output of this AND gate is 1. If it is 0 then this will not be available at the output of the buffer.

So now you find that here comes the significance of this valid bit because the tag field may contain some arbitrary value or some garbage value because of which the tag field may give a false match with this block number. That means the corresponding match bit will be equal to 1 but if the cache line does not contain a valid block or this tag field does not contain a valid block number, in that case valid bit will be equal to 0. So even if the match bit indicates a false match that will be filtered out by this valid bit. So following this, what we do is you read the entire cache line in parallel, pass it through a selector. The selector is nothing but a multiplexer where the multiplex select inputs will confirm the byte identification in the memory address and output of this multiplexer, this is the desired data which can go to the CPU.

So now we find that as we said, so I can simply put this part in one block and this is another block. So this is a set of cache lines and this is the part which contains tag, match bit and valid bit. So we find that this is being done in parallel that means for each of this match bits, I have a corresponding handling operation with the corresponding valid bits. So I have just shown only one of them. The corresponding AND gate output goes to the enable input of the buffer where the buffer input comes form the cache line and all the buffer outputs are connected together. This is possible if you have this buffer as a tristate buffer. So only the buffer for which the enable input is active, enable input is one, only that buffer output will be valid. The outputs of all other buffers for which the input is low will not be valid. The outputs will be tristated then only I can do directly where all the outputs of all the buffers and they are simultaneously coming to the multiplexer input.

So only the buffer which is active, the corresponding cache line will be available to the multiplexer input and using this multiplexer, I filter out the required byte as requested by this address. See the match bit tells you that whether the content of the argument register that is the block number because it is the block number which we are placing in the argument register. This is nothing but the block number. This block number is compared with all the tag fields, with all the tag locations parallelly. [Conversation between Student and Professor - Not audible (00:27:33 min)]. We have to have n number of comparators. If there are n locations, n entries in the tag field we have to an n number of comparators and in this case the comparators can be EXOR gates. Say I can have an architecture something like this.

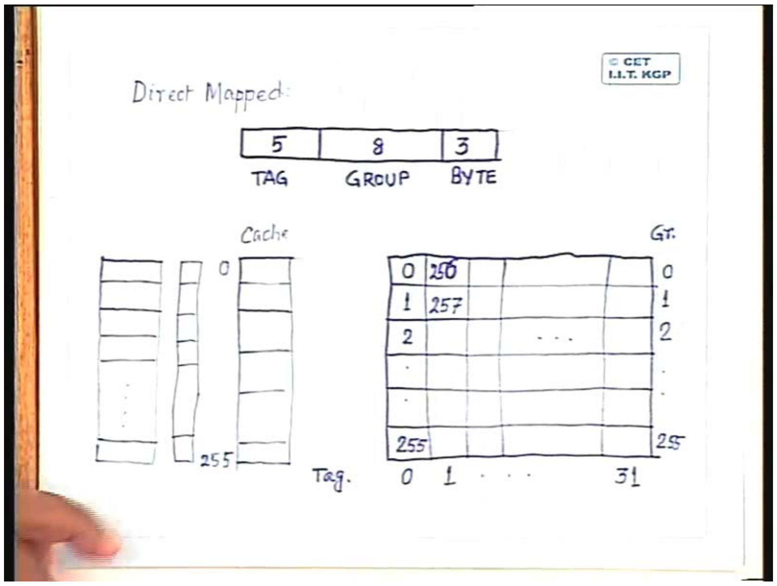
(Refer Slide Time: 00:27:59 min)



Say this is my argument register then suppose this is a particular tag location, this is the argument resistor and this is say tag 1. I want to find out whether this argument register matches with the tag 1 or not. Then what I will do is I will perform bit by bit EXOR operation, so like this it will continue. So I can have a number of such EXOR gates. So if they match bit by bit, in that case outputs of all this EXOR gates will be equal to 0 and after that what we will do is you just simply place a NOR gate. So if all these EXOR gate outputs are equal to zero, the NOR gate output will be equal to 1. So this is what is the match bit. So I have to have similar arrangement for all the tag locations that means for every tag location if there are 13 bits, I need 13 EXOR gates and one NOR gate.

So if there are n number of such tag locations then I need n into 13 EXOR gates and n number of NOR gates. [Conversation between Student and Professor – Not audible (0:30:07 min)] 13 input NOR gate that's true, we have to have a 13 input NOR gate. So this is what is needed. So obviously the circuit complexity is quite high. So in case of associate memory, you have a flexibility that any block of the main memory can be placed… main is on the other side. Any block of the main memory can be placed in any of these cache lines. So I have flexibility but at the cost of this hardware complexity. So to reduce hardware complexity, what is used is another kind of cache architecture which is called direct mapped cache. [Conversation between Student and Professor – Not audible (00:31:02 min)]. So if the match bits, all the match bits are 0 that indicates that there is no tag field which is matching with that block number that is the cache miss. So the hardware complexity will be reduced, if we use another kind of cache which is called a direct mapped cache.

(Refer Slide Time: 00:31:34 min)



So the difference between the associative cache and the direct mapped cache is that in case of associative cache, we have said that any block of the main memory can be placed in any of the cache lines. In case of direct mapped cache, a given block of the main memory in the main memory can be placed only in one of the cache lines. It cannot be placed anywhere. A given block in the main memory can be placed in only one cache memory. In case of associative cache what we have said is suppose I want to bring in a new block from the main memory where it is quite possible that when I want to bring in the new block, as we have seen in case page replacement in demand paging, the similar concept is also applicable in case of cache memory. Because I have limited number of cache memories and number of blocks in the main memory is much larger than that of the number of cache lines that we have, cache lines or cache blocks.

So whenever I want to bring in a new block from the main memory into one of this cache lines, it is quite possible that many of the cache lines will be occupied by other blocks. So I have to search for a block in the cache memory which is free, where I can put in this new block. In case of associative memory that free block can be anywhere within this cache. I can place the new block here, I can place the new block here, I can place the new block here also depending upon whichever is available. But when I go for direct mapped cache, given a particular block number of main memory where it should be placed in the cache line that is fixed. So I lose the flexibility but the advantage is the hardware complexity is much less. So let us see what is the scheme? So in case of direct mapped cache as I said that the main memory address which is generated by the CPU is divided into two fields in case of associative cache that is block number and byte within the block.

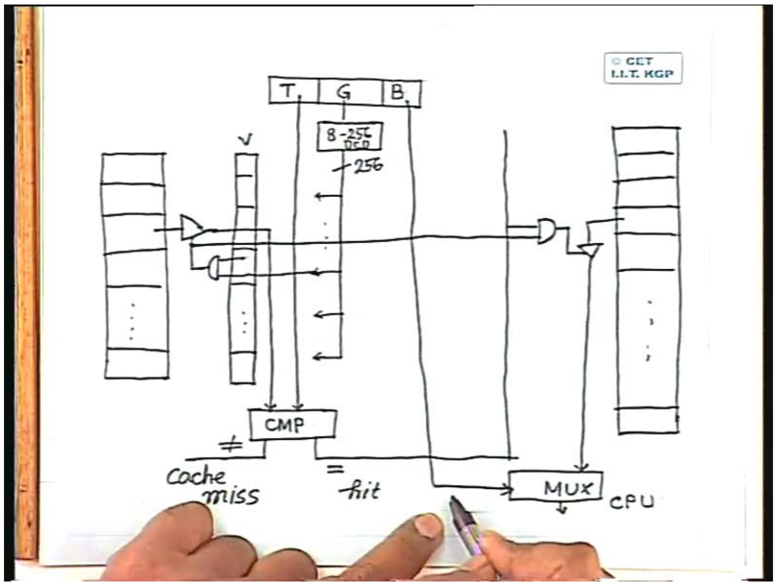
In case of direct mapped cache, the main memory address is divided into 3 fields. Let us take the same architecture that is 16 bit address lines. I will have 3 bits, the least significant 3 bits which will identify the byte within a given block. So for this I use 3 bits, the remaining 13 bits were used as block identification number in case of associative memory. In case of direct mapped cache, what we do is this remaining 13 bits is again divided into two fields. One field is used as group identification and the other field is used as tag identification. So let us assume that out of these 13 bits, say 8 bits are used as group identification and say 5 bits are used as tag identification. So unlike in the previous case where every block was identified by a 13 bit number which was the block number. Now every block is having a two component address, its tag number and group number.

So naturally I can place these blocks in the form of a two dimensional array because every block is identified by a two dimensional address. So I will place it them like this, say this is block number 0, this is block number 1, this is block number 2. So likewise this is the block number 255. So on this side I have the group identification numbers group 0, group 1, group 2 so like this it will continue up to group 255. After block number 255, the next block that is block number 256 that will again belong to group zero sorry this should be 256, 257 will belong to group number 1. Now all the blocks contained in a particular column in this two dimensional array will belong to a particular tag, so on this side horizontally I put the tag identification. So this is tag number 0, this is tag number 1 so like this it will continue, this is tag number 31. So whenever I identify, the address generated by the CPU gives tag number 0, group number also 0 that means the block which is requested is block number 0.

If it is tag number 0, group number 2 that means the byte is contained in block number 2, so like this it will continue. Accordingly I have to have cache blocks or cache lines, so these are the cache lines. If I have n number of groups I have to have n number of cache lines. So total number of cache lines that I have to have is 256 numbered from 0 to 255. So any block which belongs to group number 0 whenever it is to be brought in the cache memory that must be placed in cache line 0. Any block belonging to group number 1, when it is placed in cache memory must be placed in cache line 1. Any block belonging to group 255 must be placed in cache line 255 in the cache memory.

I also to have to have a number of tag locations, one tag for each of the cache lines. So you find that from here, if block number 256 is placed in the cache memory, the tag field should contain value 1 because that is the tag identification number. If say 255 is placed in cache line 255, the corresponding tag field should contain value 0 and simultaneously I also have to have number of valid bits, a valid bit for each of the cache lines or each of the tag memories. So I have to have 256 entries in the cache tag memory, I have to have 256 entries in the valid bit array. So given this scheme, the architecture to find out whether a given requested byte is available in the cache memory or not. What is the scheme that we should follow? The scheme will be something like this.

(Refer Slide Time: 00:40:27 min)



The main address as I said is divided into 3 fields byte field, group identification and tag field. As per our example, the group field contains, it consists of 8 bits. So first what you do is this group entry, you feed to a decoder and because here you have 8 bit so the decoder should be 8 to 256 decoder. Depending upon the content of this group field, one out of 256 decoder output lines will be active, so let us put it like this. There are actually 256 lines one of which will be active and others will be inactive depending upon what is the content of the group field. Now on this side, we have cache tag memory. So this will also have 256 entries. What we do is output of this cache tag entries goes to a buffer, tested buffer.

Now let me put it here. These are say valid invalid bits. The valid invalid bits will also be 0 or 1 depending upon whether the content of the tag field or the content of the corresponding cache line is valid or not. so what I do is depending upon which of the groups is active, I know that a particular group of blocks can belong to only a particular cache line for which the tag field can be present in only one of this tag fields, tag entries. So what I do is suppose this output of this decoder is active, I simply do ANDING of the corresponding valid invalid bits with this decoder output and this AND gate output goes to the corresponding enable input of the corresponding buffer. So once I get this then what I have is content of this tag location is available at the output of this buffer.

Now this buffer output is to be compared with that tag field because I know that this tag memory locations contain the tag identification number. So I have one comparator which will compare this buffer output with the tag field as specified in the memory address. So now you find that I have similar arrangement for each of the entries in the cache line and all these buffer outputs are wired OR they are connected together because I know that only one of the buffers will be active, all other buffers will remain inactive. This comparator will give you two outputs either not equal or equal as we have said before also. This not equal output indicates a cache miss, equal output indicates a cache hit. So whenever there is a cache hit, suppose let me put it this way, so this is cache hit line.

Whenever there is a cache hit what I have to do is I have to… and the output of this AND gate with the cache hit line. On this side I have cache lines. Again I have a set of buffers. There are number of cache lines, outputs of each of these cache lines are coming to a buffer input. This AND gate output gives the enabled input of the corresponding buffer. Then again I have a selector or a multiplexer, the buffer output comes to the input of the multiplexer, select input to this multiplexer comes from the byte identification number and output of this multiplexer goes to the CPU. [Conversation between Student and Professor – Not audible (00:46:18 min)] but that can come from anywhere, I have to identify a particular cache line. [Conversation between Student and Professor – Not audible (00:46:32 min)] all the buffers, I have a set of buffers here all the buffer outputs are wired together.

At the most one buffer output will the high but I have to identify which one. How do I know that? [Conversation between Student and Professor – Not audible (00:46:47 min)] I am selecting only that. See here I have to identify all group identification number, tag field. The field is compared with this. The group identification number is available from the output of this. So only when and the corresponding valid bit is also 1. So only when all these conditions are true that is valid bit is 1, I have to get the output from the decoder, I also have to have a condition of cache hit then only the corresponding cache line will be read. So this is needed because out of so many cache lines, I am interested in only one but which one? That is given by this line. All the conditions must be true. So which one I am interested in that is given by this AND gate output. That is ANDED with the cache hit line to give me a particular cache line in the cache memory. So that cache line is going to the multiplexer input and the select input to this multiplexer is coming from the byte identification number and the corresponding byte goes to the CPU. Data is available in the cache memory. [Conversation between Student and Professor - Not audible (00:48:18 min)] which cache line, which one? [Conversation between Student and Professor - Not audible (00:48:29 min)] but which cache memory, which cache line? I have similar elements for all of them; all the outputs are wired together. I have to identify a particular cache line; I am not interested in anyone. So with this we stop today. Thank you.