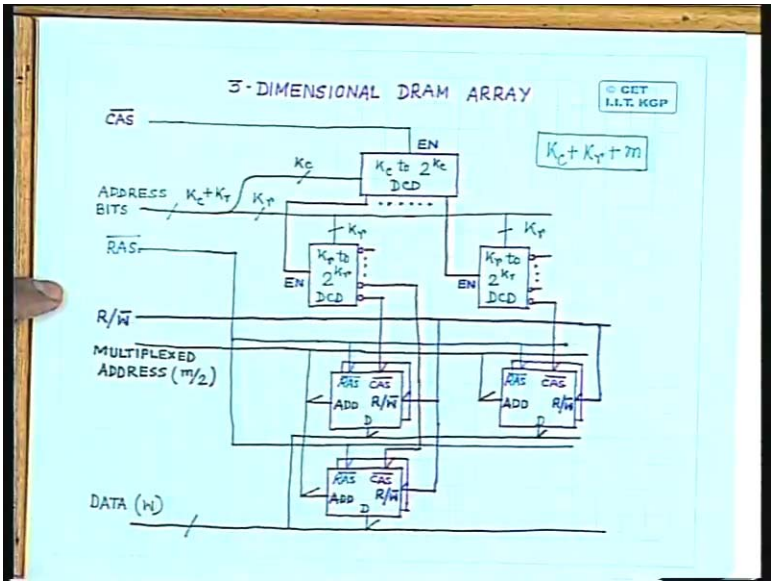
**Digital Computer Organization**
**Prof. Dr. P. K. Biswas**
**Department of Electronic and Electrical Communication Engineering**
**Indian Institute of Technology, Kharagpur**
**Lecture No. # 22**
**DRAM Architecture Buffer Cache**

So this is the complete three dimensional DRAM chip array that we started drawing in the last class but we did not complete.

(Refer Slide Time: 00:01:13 min)




Here you find that you have a set of address decoders and the address decoders, one of them is a column decoder. So this address decoder is a column decoder whereas the other address decoders, I can have a number of such address decoders which are actually called row address decoders. Now the number of address bits which will come to this unit is $K_c + K_r$. So the total number of address bits will be $K_c + K_r + m$, this is the total number of address bits that will come to this three dimensional memory chip array. Out of that $K_c$ and $K_r$ number of bits are used as row address identification and column address identification whereas this m bits, m number of bits is actually divided into two halves $m/2$ and $m/2$ which you have seen that when we discussed about the DRAM chip that first you have to give the row address along with $\overline{RAS}$ signal followed by the column address along with $\overline{CAS}$ signal. So this m number of bits, this actually addresses different locations in the memory chip. So this is divided into two halves $m/2$ and $m/2$ which is here represented as multiplexed address $m/2$ number of bits. So out of this $K_c$ and $K_r$ number of bits, $K_c$ number of bits goes to the column address decoder and depending upon the bit combination of this $K_c$ number of bits, one of the columns will be active. So if all the bits are 0 in that case zeroth column is active and that will activate this particular row decoder. Similarly,

if $K_c$ contains a 0 0 0 0 0 1, then the first decoder output will be active which will activate the next row address decoder.
So once you decide, you enable a particular row address decoder then output of the row address decoder will depend upon the $K_r$ bit combination. So you find that when $K_r$, all the bits in $K_r$ are 0 in that case the zeroth output of all these row address decoders are supposed to be active that means we are going to activate a particular row in this two dimensional chip array. So that is why these are row address decoders. And out of all these decoders, the decoder which will be active, which will be enabled that depends upon the corresponding output of this column address decoder. So you find that as we said that in case of dynamic RAM chip, we don't need any chip select because it is the $\overline{\text{CAS}}$ signal which gives the same function as chip select. So it is the $\overline{\text{CAS}}$ signal which enables the column address decoder. Then output of column address decoders activate or enables different row address decoders.

So using this $K_c$ and $K_r$ combination, you activate one of these memory chip combinations and here you find that I have given, every unit is given as an array of a number of memory chips because as we discussed that in case of dynamic RAM, if we have that every location contains one bit, say 1 m x 1 memory organization that we discussed, there if we want to have that every memory location should contain 8 bits in that case I can connect 8 such dynamic RAMs in parallel. That is why I have shown each of these units as an array of memory chips where within this all the memory chips are connected in parallel. That means they are $\overline{\text{CAS}}$ signals are connected together, $\overline{\text{RAS}}$ signals are connected together, address signals are connected together so that way all the chips in each of this module, each of this unit are parallel.

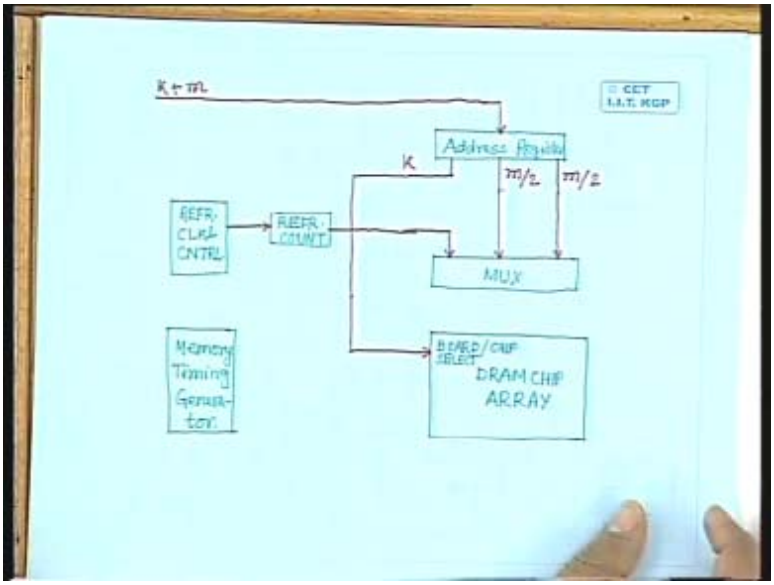
So once I select a particular module in that case, in that module I have to give the multiplexed address which is $m/2$ number of bits. So first we have to give $m/2$ number of row addresses along with the $\overline{\text{RAS}}$ signal. $\overline{\text{RAS}}$ signal for a particular module is also connected in parallel, so that is how it has been shown here. So for these memory chips, you have the same $\overline{\text{RAS}}$ signal. So when you give the row address on $m/2$ number of address bits along with that the $\overline{\text{RAS}}$ signal, $\overline{\text{RAS}}$ signal has to be enabled and once you give this $\overline{\text{RAS}}$ signal then with the help of the $\overline{\text{CAS}}$ signal, the $\overline{\text{CAS}}$ of the corresponding module will also be enabled, along with that $\overline{\text{CAS}}$ signal you also have to give the $m/2$ number of bits for column address. Then if it is a read operation then this $R/\overline{W}$ line has to be made high. When the corresponding memory module will be read and the output will be available on this data output lines and here you find that the data output lines of all these chips in a particular module are also connected in parallel.

So this is how we can generate a three and I am calling it three dimensional array because each of this modules, the memory modules are arranged in a two dimensional array. Within every module, we have another dimension where a number of chips are connected in parallel. So the total array, the total modules becomes a three dimensional DRAM array but still it is only the chip organization, it is not the DRAM board because as we said that what comes from the CPU? The CPU gives a number of address bits, the CPU gives either the read signal or the write signal and the CPU gives a number of data bits, data lines but the CPU does not give you $\overline{\text{CAS}}$ or the

CPU does not give you the $\overline{RAS}$ and we can also assume that the CPU, if the CPU gives different read line and write line then we have to combine those read lines and write lines into $R/\overline{W}$; R, $\overline{W}$ line into a single line.

So using such a type of three dimensional array, if we want to construct a memory module, a memory board in that case we have to have some additional control circuits which will provide all these signals. That means the control signals which are given by the CPU should be converted in such a form that the control signals become compatible with this three dimensional DRAM array. So the modules in case of a DRAM board will look like this.

(Refer Slide Time: 00:08:33 min)




First the CPU gives you a number of address lines. So initially let us have an address latch or address register which latches the address lines or the address bits given by the CPU. So I have to have an address register. Now all the address lines which are given by the CPU should come to this address register. So let me assume that the CPU gives let us say K + m number of address bits, out of which this K number of bits will be broken into 2 components $K_r$ and $K_c$ and they will be used for addressing a particular module within the three dimensional DRAM chip array, m number of bits will again be divided into two components $m/2$ and $m/2$ which will be used for addressing a row and then addressing a column in a particular chip.

So this entire address will be divided into three components, in one component I will have K number of bits then I will have $m/2$ number of bits, I will have another component of $m/2$ number of bits. And suppose the DRAM chip array is placed somewhere here, this is say DRAM chip array. Now this K number of bits will come directly to DRAM chip array and this will act for board and within the board chip select. Of course this has to work through all these decoder units and all those things. the remaining $m/2$ and $m/2$ number of bits will be used to address a row and then address a column and not only that whenever we feed some address to this DRAM

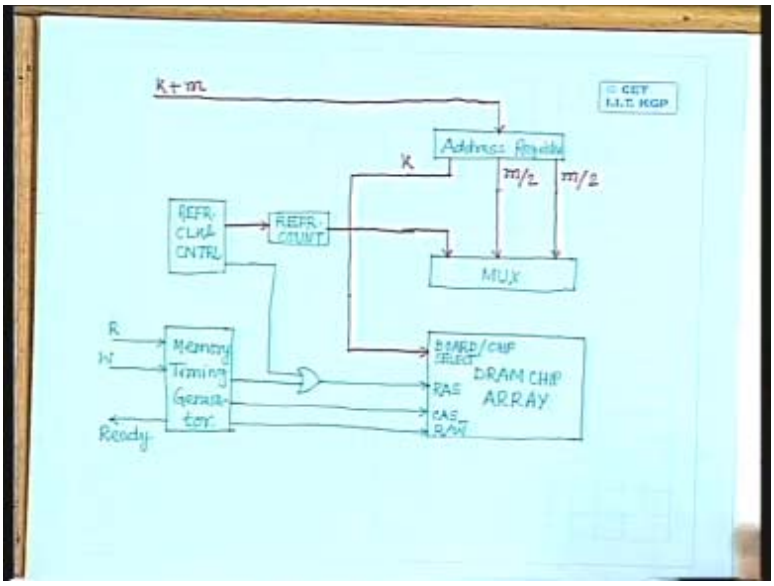
chip array, we have said that I have to have some refresh controller so that at regular intervals of time, the refresh controller can refresh different locations within every memory chip.

So for that what we will use a multiplexer, two inputs to this multiplexer will come from this $m/2$ lines so that I can select one of these $m/2$ lines to address this DRAM chip. The other possible address input to this has to come from the memory refresh controller which will give the row address whenever the memory is to be refreshed. So for that what I have to have is a refresh counter, this is say a refresh counter and it is the counter which generates the row addresses in case of memory refresh. I also have a refresh clock and control unit, so here I will put a refresh clock and control unit and it is this refresh clock and control unit which will activate the refresh counter whenever memory refreshing is needed.

In addition to this we have to have another unit, let us call that as memory timing generator. It is this memory timing generator which will accept the request of read from the CPU, which will also accept the request of write from the CPU and it is the responsibility of this memory timing generator to give out a signal called ready to the CPU because unlike in case of static RAM, where if you give the address then followed by read or write operation, the memory is ready for reading data of that particular location or writing the data from that particular location. But in case of dynamic RAM it is not so, the dynamic RAM has some sequential nature that is we have to give the row address along with the $\overline{RAS}$ signal followed by column address along with the $\overline{CAS}$ signal.

In addition to that there is some refresh operation that may also be taking place intermittently. So the CPU has to know that whatever operation the CPU wants to perform on the dynamic RAM that operation is complete. So it is the responsibility of this memory timing generator to give a ready signal to CPU and the CPU accepts the ready signal to know that the RAM is ready. So you find that even in case of 8085 CPU, you have an input called ready input and the CPU can continue with its work only when the ready input is high. If the ready input is low then the CPU will incorporate wait states instead of $t_0$ $t_3$ $t_4$ and all those things, after state $t_2$ 8085 will incorporate wait states if it finds that ready line is low. So that is the purpose of this ready signal which again has to be generated by this memory timing generator.

(Refer Slide Time: 18:48)




Now what are the control signals that this memory timing generator has to generate? One is the $\overline{CAS}$ signal, so this will give you CAS signal, it also has to generate $R, \overline{W}$ combined signal because from the CPU we get two different lines, one for the read operation and other for write operation. but in case of DRAM chip we have seen that it needs a single line, if high then it is read operation if low it is write operation so that has to be combined by this memory timing generator. The other signal that this memory timing generator has to generate is RAS signal but RAS signal is needed in two different cases. One is to perform a read operation or write operation on this DRAM chip array and the second way when this RAS signal is needed is for refresh operation. That means I have to have two generators for RAS signal, one from this memory timing generator which will generate the RAS signal in case of read or write operation and the other for refresh operation which will be generated by this refresh clock and control unit. So I will have two sources of this RAS signal and these two RAS signals will be, let us put it this way say logically OR together and this output becomes the RAS signal for the DRAM chip array. Now for refresh operation what has to be done is you find that if there is a write operation or there is a read operation, write operation means refresh is already done.

If it is read operation, then following every read operation there will be write operation or refresh operation. But this refresh clock and control unit that works in parallel. So if the refresh interval is say 4 ms may be at the interval of every 4 ms I have to refresh the DRAM chip. So what this refresh control clock and control circuit will do is it will keep count of 4 ms after every refresh is complete. At the end of that 4 ms interval, it will put a request to this memory timing generator that there is a time for refresh. So let us put it as a refresh request. On getting this refresh request from the refresh clock and control unit, the memory timing generator will decide whether it can allow the refresh operation to be performed now or the refresh controller will be asked to wait.

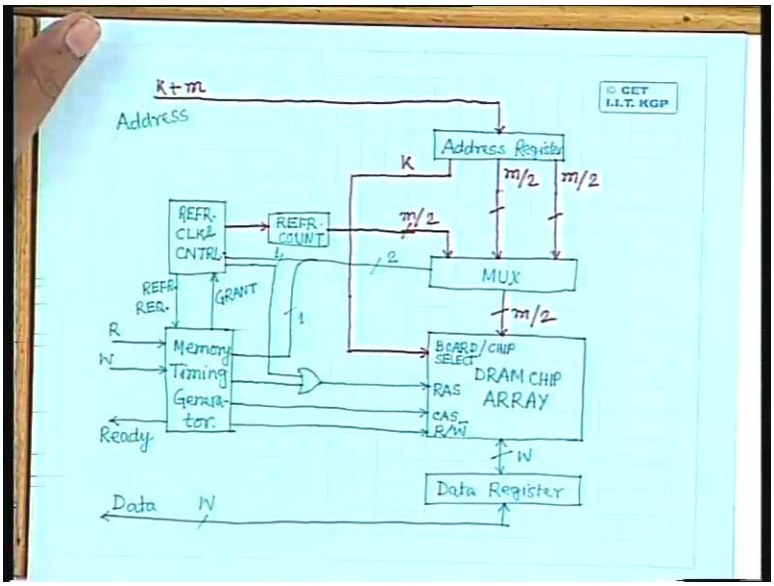
So accordingly this memory timing generator will give back a signal to this refresh clock and control unit, the signal which is called a refresh grant signal. So if the refresh grant is issued, on getting this refresh grant, the refresh clock and control unit will ask the refresh counter unit to generate the memory addresses which had to be refreshed and this is nothing but a sequential counter. Say if some read operation is going on or some write operation is going on, if at the same time this refresh control unit puts a request for a refresh. The read and write means a refresh will automatically be done. Isn't it? Write means it is refresh, I am writing a new data. At the same time whenever you read the content of a particular row, we have seen that in the DRAM chip architecture that before you come to the external data bus, the entire row is read at a time. It goes to the sense write amplifier, output of the sense write amplifier goes to a selector where a particular is column is selected by the column address.

So whenever you read a particular row, the entire row is read. Even if you want to read a particular bit, the entire row containing that bit is read up to the sense write amplifier. then the sense write amplifier itself, following that read operation writes back the entire data into the corresponding row but to the output to the external data bus what will be available is a particular bit because that passes through a selector or a multiplexer. So every read operation is followed by refresh operational automatically. So only when this refresh clock and control unit gets the grant signal from the memory timing generator then it can ask this refresh counter to generate the refresh addresses and not only that this multiplexer also has to be set properly because now the refresh counter has to provide the row address, the row which will be refreshed.

So this multiplexer input will get two kinds of select inputs, so you can put it this way that one select input will come from this refresh clock and control unit and it should also get the select input from memory timing generator. So what can be done is here this can generate one bit, this can also generate one bit, these two are combined together to give you two bit select lines because there, here we have 3 input lines, so for three input sources I need two bit select lines. So those two bit select lines can be generated in this way. Then output of the multiplexer that actually gives you the address lines for the DRAM array. So here I will have $m/2$ number of addresses. So in case it is memory read or memory write operation, this $m/2$ addresses will come either from this $m/2$ lines or from this $m/2$ lines depending upon whether it is the row address or column address.

In case of refresh, $m/2$ addresses will come from the refresh counter and that has to be selected through this select lines. Then finally this DRAM data lines will be connected to a data register. Here we have a data register which will either accept data from that DRAM chip array or feed the data to the DRAM chip array and we can assume that suppose the width of this data lines is w. If I say width data lines is w that indicates that if every location in a DRAM chip contains one bit, so if it is a bit organized DRAM chip in that case for every such module in this diagram there has to be w number of chips connected in parallel because my data bus width is w bits whereas for every RAM, for every DRAM a location contains only one bit and then finally from the data register this w by 2 number of data lines goes to the CPU. So these are the data lines of width w, these are the address lines of width K + m.

(Refer Slide Time: 24:45)



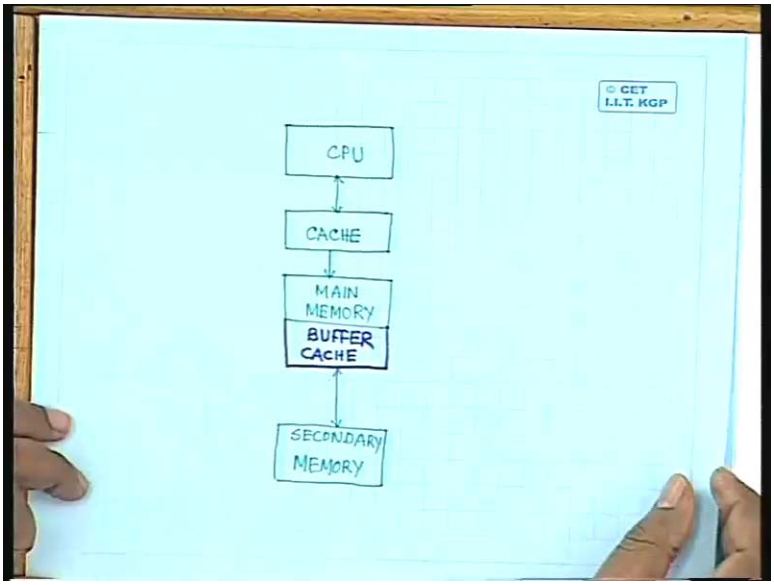

So you find that how the whole thing will work whenever the CPU wants to read a particular location from the DRAM or once you write something into the DRAM, what the CPU will do? CPU will give the address on this address lines and either read signal or write signal. Now on getting this address and read signal and write signal, this memory timing generator will generate the corresponding control signals which are needed for this DRAM chip array.
These address lines out of that K number of address bits will be used for selecting a board and within a board, a particular chip, so when I say a particular chip that means a chip module like this. Then $\overline{\text{RAS}}$, it can be generated either from this memory timing generator or from refresh clock and control unit. So these two are logically OR together and that gives the RAS signal to the DRAM chip array. CAS signal is generated by the memory timing generator, so when I say this generated by the memory timing generator, actually internally this signal will be combined with this because if you look at this architecture, you find that CAS actually comes from this unit. So this CAS signal which is generated by the memory timing generator is the CAS signal which is given here but to the actual chip, the CAS signal comes from the column and row address decoders.

Similarly, for refresh operation whenever refresh is to be done, the address is generated by the refresh counter and through the multiplexer it goes to the DRAM chip and then finally from the data register, the CPU can read the data or for writing a data into the DRAM chip, the CPU will write the data to DRAM chip array through these data registers. So we have done three kinds of memory organizations till now, the cache memory organization, static RAM organization and dynamic RAM organization. And I think we have said that regarding the memory hierarchy in a computer system, usually we have a two level hierarchy that is between CPU, from CPU I have the main memory, between main memory and then from the main memory we have the secondary memory usually that is the hard disk.

Now if you put the cache memory in between the CPU and the main memory that becomes a three lever hierarchy. so whenever the CPU wants to read something, any data or wants to write any data, firstly it checks that the block which will be read or the block which is to be written into whether that is available in the cache memory or not. If it is not available in cache memory, then only the CPU will try to find out that block in the main memory. If it is not even available in the main memory that leads to what is called page fault interrupt.

Page fault interrupt that is page which is being looked into is not available in the main memory. Whenever you have a page fault interrupt then the required data or the required page containing the data has to be loaded from the secondary storage into the main memory.
Now between the secondary storage and the main memory, we have another layer which is logical not physical layer. Say for example between main memory and CPU, we have another layer additional layer which is cache memory and that is a physical layer. Cache memory must be physically present but between the CPU and the main memory another level of memory which is maintained which is a part of the main memory but logically it is managed in some other way which is called a buffer cache.

(Refer Slide Time: 00:29:58 min)




So the entire hierarchy will be something like this. At the top most level you have the CPU, CPU directly accesses the cache memory. Of course you might be knowing that we have two different kinds of cache memories which are called $L_1$ cache or $L_2$ cache. $L_1$ cache is the cache memory which is in built within the CPU whereas $L_2$ cache is the cache memory which is external to the CPU but the architecture is same as the cache memory architecture. So here we can have cache memory which again is optional, earlier systems didn't have this cache memory. From the cache memory we have main memory then finally we have the secondary memory usually the hard disk.

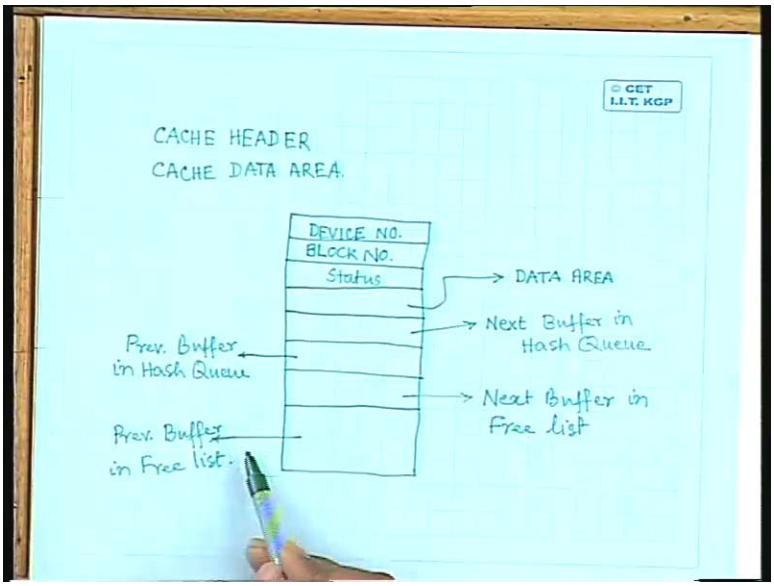
Now instead of directly connecting the main memory to secondary memory, a partition is maintained in the main memory which is called a buffer cache. So I will make it attached with

the memory. So this is a unit which is called a buffer cache which is a part of the main memory and directly controlled by the operating system. this is not the part of any of the user spaces that we discussed earlier, while you talked about the memory organization, main memory organization we have said that the main memory is divided into a number of partitions, some of the patricians are given to the operating system whereas rest of the partitions are given to different users.

This buffer cache is actually the operating system area; it is not the user area. So whenever anything is to be read, is to be accessed from the secondary storage, a block containing that required data is put into the buffer cache. From the secondary storage I cannot read a single byte or I cannot read a single character, I have to read the entire block containing that character. The block will go to the buffer cache; from the buffer cache it will go to the user area. So the entire memory hierarchy will be something like this. So effectively we have 3 level hierarchy cache memory, main memory and secondary memory between main memory and secondary memory you have the buffer cache. So in case of page fault when the required data is not available in the main memory, when I say not available in the main memory means user space in the main memory.

If it is not available in the user space in the main memory then there would be a page fault, following page fault I should go to secondary memory to get the data. now before going to the secondary memory what the operating system does is, it tries to find out whether that data is available in the buffer cache or not. If it is available in the buffer cache, then I don't have to read it from the secondary storage. From the buffer cache it can be written to main memory, subsequently to cache memory and subsequently to the CPU. In case the required data is neither available in the buffer cache then only we have to physically read it from the secondary storage. Now this buffer cache is maintained as a linked list of different memory areas.

(Refer Slide Time: 00:34:07 min)



When I talk about the buffer cache, the buffer cache actually has two portions. It has called, it has got a cache header. Now whenever I say cache, now it will mean buffer cache not the cache memory. So it will consist of a cache header, it will also consist of a cache data area.
Cache header will consist of a number of fields. Initially the first few fields will contain the device number because in a computer system, I can have more than one hard disk or even if I have a single hard disk that can be logically divided into more than one hard disk. I can have c drive, I can have d drive, I can have e drive and so on. Each of them will have a different device number. So one field will contain the device number and the other field will contain the block number. Now what are these blocks in the secondary storage, I will come to that later. So for the timing being, you assume that every device is divided into a number of blocks. So whenever I have to read any data from the secondary storage, of course I can have two types of devices, some devices are block devices, some devices are character devices. For example, a printer is a character device. Whenever I want to take some printout, I can send character by character to the printer, I can get character by character print out.

Similarly, keyboard is a character device. I can enter a particular character; a character can be read by the CPU through the keyboard unit. But a secondary storage it's a block device, from the secondary storage, I cannot write a single character to a secondary storage, I cannot read a single character from the secondary storage. So logically I can think that a secondary storage is divided into a number of blocks where every block will consist of a number of bytes. the length of a block can be say 512 bytes, may be 256 bytes, may be 1 kilo byte, may be 4 kilo bytes and so on that depends upon how the installation has been made. And whenever I have to write anything to a particular block, suppose I want to write a single character in the particular block, I cannot write a single character to a block physically. What I have to do is I have to read the entire block to the buffer cache, from the buffer cache it has to be brought to user space in the main memory.

In the main memory I can modify that particular character. May be I don't want to modify the entire block; I want to modify a particular character in that block so that has to be done in the main memory. Then once you modify that particular character in the main memory then you have the entire modified block and if it is to be reflected on the secondary storage then this entire block has to be written on the secondary storage. so these are block devices, so I have to have the device number and block number, the block which is contained in that particular buffer. Then I have to have another field called status field. This status field says what is the status of the buffer? whether this buffer is being currently used by some process or locked by some process, whether the buffer is free, whether the buffer contains some data and marked as something called delayed write?

What is this delayed write? Suppose I have a situation in which case a particular buffer contains some data but at some point of time, I decide that these buffer should contain some other data. Now when the buffer has to be written by the data from some other block from the device, at that time I have to check whether the data is there in the buffer should be saved on to the disk before overwrite or it is not to be saved. If the data contained within the buffer is different from the disk content of the same block, in that case before you overwrite the buffer, the buffer should saved on to the disk because otherwise the disk will not get the updated data. whereas if the content of the buffer and the content of this disk block is same, in that case I need not save the buffer to the disk because copy of that is already existing on to the disk. So I can simply overwrite the data by

a new block. So all those information are to be contained in the status field. Then it has a number of pointer fields. As we said that I have two areas in a buffer, I have a cache header and I have a cache data area, this is actually the header. From the header I have to be have a pointer which points to the data area. There are a number of other pointers, one pointer points to next buffer in the hash queue, I will come to hash queue later on.

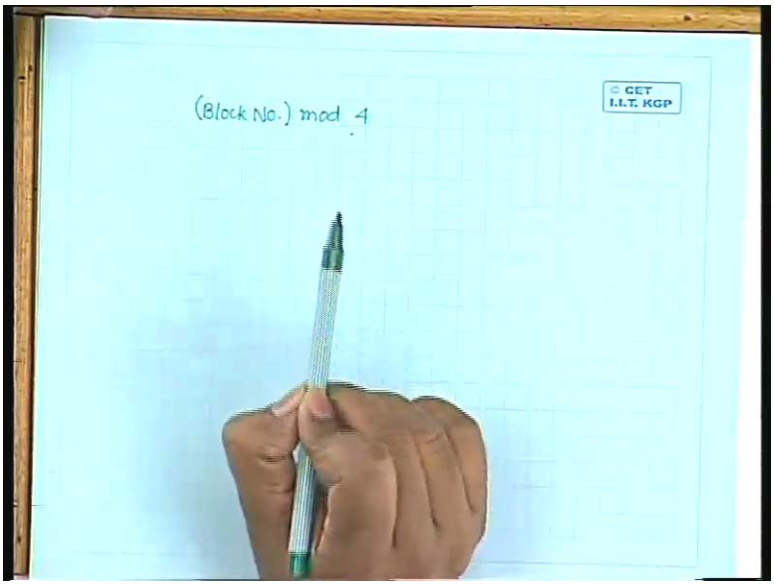
As I said that all the buffers are maintained in the form of a linked list, so I have to have a number of pointers. So one pointer is pointing to the next buffer in the hash queue; another pointer pointing to previous buffer in the hash queue. One pointer will point to next buffer in free list and I have to have one more pointer which points to previous buffer in the free list. Now let us see what are this hash queue and what is this free list? Hash queue is suppose, I decide that the system should contain 1000 buffers. So for 1000 buffers I will have 1000 such headers, I will have 1000 such data areas. So when I have 1000 buffers then there can be 1000 different data blocks, disk blocks present in the buffer simultaneously.

Now at a particular time suppose the CPU encounters a page fault and following the page fault, the CPU finds, the CPU determines that it is a block number 5 which has to be read from the secondary storage. and as I said that before going to the secondary storage, before actually going to the secondary storage, the CPU will try to find out or the OS will try to find out whether block number 7 is present in the buffer cache or not because if it is present in the buffer cache then time to access that particular block will be very small because this buffer cache is maintained in the main memory. I don't have to read it from the secondary storage.

So first, the OS will try find out whether block number 5 is present in the buffer cache or not. How does it do it? For every buffer I have a header, the header contains block number. so the OS can go to the head of this linked list, the first node in the linked list checks whether block number 5 is present in this buffer cache or not. If block number 5, it gets block number 5 a match then that particular data area of the buffer can be copied to the user space and the process gets the required data. In case it finds that block number 5 does not match with the first entry, the first node in the linked list it has to go to the second node, if it does not match there it has to go to the third node and so on.

So if I have 1000 number of such nodes then in the worst case, I have to have 1000 search operations to find out whether the block exists in the buffer or not. Now in the worst case if the block actually does not exist in the buffer, I have to have 1000 search operations because until and unless I search for every buffer, I cannot say that the block does not exist. So on an average the number of search operations that has to be performed, to find out a buffer is quite large and it is linear with the number of buffers that we will have. So to reduce that what is done is this buffers are maintained in two lists, one is the hash list and the other kind of list is free list. Now why do you go for hash list? To reduce the search time. So if we decide that in a system, I will have say 4 hash queues then a simple hash function that can be used is say (Block No.) mod 4 .

(Refer Slide Time: 00:44:06 min)




So if I perform this block number 4, block number mod 4 this can give me 4 possible values, one of 4 possible values 0, 1, 2 and 3. So accordingly I can have 4 hash queues, every hash queue will have a header which will identify that whether it is hash queue 0 or hash queue 1 or hash queue 2 or hash queue 3. Whenever you search for a particular buffer, you perform the same hash function, find out what is the hash value. If the hash value becomes 2 say for example I want to find out whether block number 6 is present in the hash or not, so I will perform 6 mod 4 which gives me a value 2. So if block number 6 exists in the buffer, it has to exist in hash queue number 2. It cannot exist in any other hash queue. So I will only search those buffers which are present in hash queue number 2, I will not search in any other hash queue for block number 6, so that can reduce the search time to a great extent. We will have more discussion on this in the next class. Thank you.