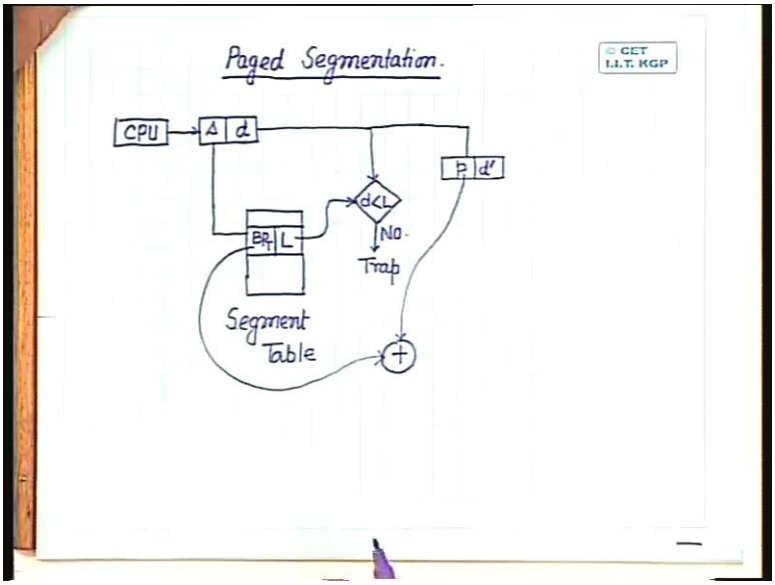
**Digital Computer Organization**
**Prof. P. K. Biswas**
**Department of Electronic & Electrical Communication Engineering**
**Indian Institute of Technology, Kharagpur**
**Lecture No. # 16**
**Memory Organization- IV**

Today we will discuss about paged segmented memory management. In the last class we have seen two types of memory organization. One is paged memory organization, other one is segmented memory organization. And we have seen that in case of paged memory organization, the main memory is divided into a number of partitions where every partition of equal size and at the same time the logical address space of the user programs they are also divided into number of partitions of the same size.

(Refer Slide Time: 00:01:05 min)




In case of paged memory management, we have said that it is simpler to implement but the difficulty is, it does not maintain the logical structure of the user address space. So to avoid that problem we have discussed about segmented memory management where, when you take the program into a number of partitions, every partitions corresponds to a module of the user program. That means along with having the advantage of partitioning, you also maintain the logical structure or the modular structure of the user address space but the problem in that case is the management of the main memory because the main memory has to be partitioned into different blocks.

So different partitions can be of different size and the sizes will be depending upon that which module you want to load in the main memory. Now to take advantage of both what is suggested is a paged segmented memory management or paged segmentation in which case your basic partitioning scheme is the segmented memory scheme. Then every segment can be broken into a number of pages.

So what we can assume is I can assume that every segment whatever is applied in case of an user program in paged memory organization, the same concept will be applied to every segment of the user address space or every module of the user address space. That is every module of the user program will now be divided into a number of pages where the module itself is a segment. So you are imposing paging technique over segmentation. So now again what the CPU has to generate is an address. The address has to be in the form of segment number and offset within the segment. So the CPU gives the address having two components; the first component is the segment number and the second component is d which is the offset within the segment. Using the segment number, as before you refer a particular segment table. So this is segment table.

Now in earlier case, in case of segmented memory organization we have said that this segment table, every entry in the segment table contains two fields. One field gives you what is the base address of that segment and other field tells you that what is the limit of the segment. So that this offset within the segment, you can compare with the limit value. So only when this offset is less than the limit value, then only the addressing is valid. If the offset becomes greater than the limit value then the addressing is not a valid address. Here also we will divide every entry in the segment table into two components. The first component instead of giving you the base address of the segment, now it will give you that what is the address of the corresponding page map table.

As you said that the paging concept which is used for every user program, in case of page mented, in case of paged memory management, the same concept will be used for every segment of the user address space. So as in case of simple paged memory management, where I have to have a page map table for every user program. Now I have to have a page map table for every segment of a user program. So now every entry in the segment table will have two fields. One field as before will give you that what is the limit of that segment because I always have to check that the offset that is specified within the segment must be less than the limit of the segment otherwise the addressing is invalid. The other entry will give you the base address of the page map table. So let me write it as BP T and this page map table whose base address is given in this entry is a page map table corresponding to this particular segment which has been specified by the CPU.

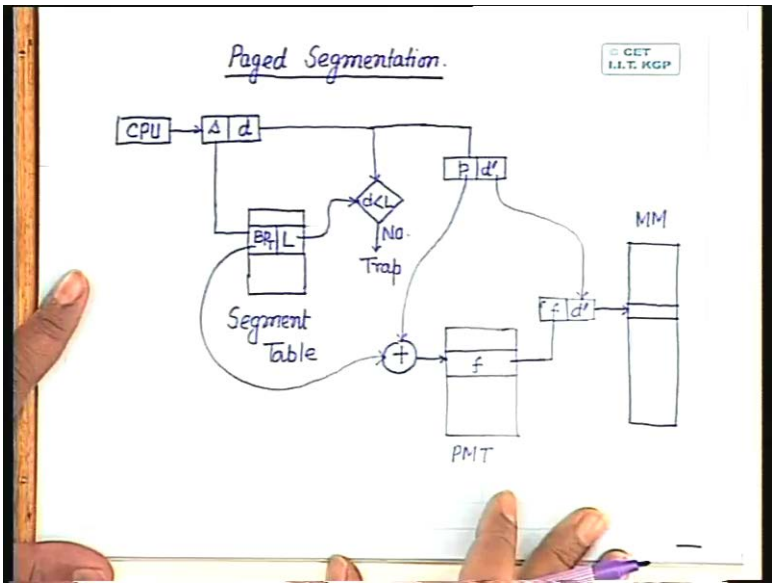
Now as before what we have to do is I have to check this limit with the offset that is specified within the address. So this d must be less than L, for this addressing to be a valid addressing. So I have to check whether d is less than L or not. If d is not less than L, in that case there will be a trap following which the program has to be terminated because the address that has been generated is an illegal address. In case d is less than L, then only this addressing is valid and what I have to do is as I said that every segment is broken into a number of pages. So the paging and the offset within that page for this particular d has to be found out for d itself. So if d is less than L, in that case what we do is this d is broken into two components. One is the page number and I put it as d prime which is offset within that page.

Again this is very simple because this d integer division with the page size that gives you the page number and d mod capital P give you the offset within the corresponding page.

Now from the segment table what we are getting, the segment table gives you the base address of the page map table, the page map table for this particular segment s. So every segment will have different page map table and accordingly every entry in the page map table will give you different page map table pointers. So once I have the base map table, base address I add to this the page number that is generated from the offset within the segment. And this gives you a particular entry in a page map table. So this is a page map table for this particular segment s and the content of this particular entry in this page map table is obviously the frame number f where this page of this segment is loaded in main memory.

So once I have this f, I can combine this f with d prime following the same formula that we have said earlier that f multiplied by page size plus d prime that gives you the physical address within the main memory. So this gives you the physical address within the main memory where the data or instruction is to be accessed. So this is the main memory. So now find that the concept is very simple that simple paged memory technique is extended to every segment of an user program. So effectively what I get is when it comes to the management of the main memory, the main memory is divided into a number of pages or number of frames where every frame is of same size.

(Refer Slide Time: 12:20)




Coming to the logical address space, the logical address space is initially broken by using segmentation that means every module of the user program becomes a particular segment. Then you applied paging technique on every segment. So because you are applying paging technique on every segment, so for every segment I have to have a page map table. So initially using the segment number that is generated by the CPU, come to the segment table. From the segment table I get two components, one is the base address of the corresponding page map table and other one is the link of the corresponding segment. So offset within the segment which is generated by the CPU that is compared

with this length of the segment value. So when I find that offset is less than length then only the addressing is valid.

The moment offset becomes greater than L, the addressing is not valid. So there has to be a trap following which the program will be terminated. Only when the addressing is valid then what I do is I apply paging on this offset d to get a page number and the offset within the corresponding page. Now this page number and offset will be for a particular page in this offset d or for this segment s. Once I have this page number, I have the page map table base address from the segment table, at this page number with this base address to give you an entry, give you a pointer to a particular entry in the page map table and this page map table is the page map table of this segment. So for segment number 1, I will have one page map table. For segment number 2, I will have another page map table. So base addresses of all those page map tables will be entered in the corresponding entries in the segment table.

Now as before, this entry in the page map table contains a frame number. Now using this frame number and this offset within the page, I get the physical address of the main memory from where the data or instruction is to be accessed. So now we find that I can combine the advantage of both. I can maintain the logical structure, the modular structure of the user program and coming to the main memory, the management of the main memory is again simpler because the main memory is divided into number of partitions or number of frames where every frame is of same size. Excuse me sir, the size of each frame within size, what is the size? That size is to be decided while you install the system. The page size may be decided to be say 256 bytes, 512 bytes, 1 kilo bytes and so on. So once you decide that what is the size of the page, that page size will be applicable to all the segments. I cannot have that for a particular program I use one page size, for another program I use another page size because in that case again that lead to a complication of memory management. So this is what is paged segmentation.

Now find that in all these techniques that we have discussed whether it is paged memory organization or segmented memory organization or paged segmented memory organization, in paged organization of segment or the segmented organization of this paged segmented organization, we have assumed that if I have a job say 100 kilobytes in that case it is no more necessary that I have to have a single partition of 100 kilobytes to load the job into main memory and execute it. Rather if I have say 10 kilobytes of partitions free and there are 10 such partitions and my page size is also 10 kilobytes then what I can do is I can break my job of 100 kilobytes into 10 such pages. These pages can be loaded into 10 frames in the main memory which are free and the job can be executed. But still I need 10 such free frames because if my job size is 100 kilobytes, I had to have a number of frames. The total size of that must be 100 kilobytes or more.

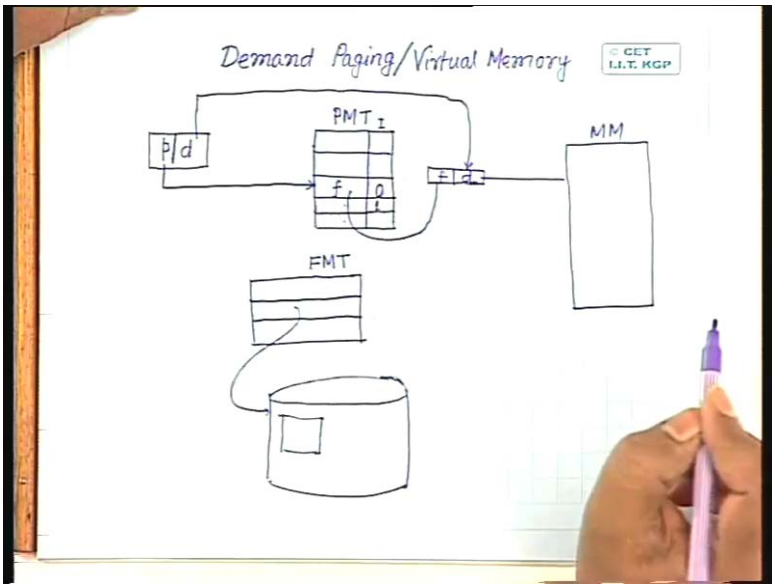
Again that is the limitation but these days you know that even if you have a machine which is having say 64 megabytes of memory and if you have a program whose size is say more than 64 megabytes, say 120 megabytes even then the program can be executed on the machine. So that clearly shows that it is not necessary that I have to load the entire program in the main memory for execution. But what is needed is as it should be clear by

now when you discussed about the CPU, that for every program only one instruction is executed at a time. You don't execute more than one instruction. Even if we go for pipelined type of concept so in that case again if you have 6 stage pipeline, CPU maximum of 6 instructions can be in the pipeline simultaneously not more than that. So if I load only that part of the program in the main memory which is currently under execution, suppose I am executing say a hundredth instruction, so if the page containing hundredth instruction is loaded in the main memory that should be sufficient to execute my program.

The other instructions can be brought from the secondary storage to main memory only when they are needed. So only the page which is currently active that has to be in the main memory. Otherwise it cannot be executed but the pages which will be active later they can be brought to the main memory from the secondary storage on demand, only when they are needed. But your limitation is the total disc space must be sufficient to accommodate the entire program. Otherwise obviously I cannot store the program anywhere, so it cannot be executed.

So now your job size is limited by the disc size, it is no more limited by the main memory size. And because you are bringing the pages to main memory on demand, when they are needed so such a kind of memory management or memory organization is called demand paging and we assume that the basic structure is a paged memory structure.

(Refer Slide Time: 00:17:29 min)




Excuse me. Yes. sir what is the advantage of this technique over segmented memory management. Which one? What is the advantage of paged segmented memory management over a segmented memory management? In case of segmented memory management what we had? We have to break this main memory into a number of partitions where the partitions are not of same size. That means I have to employ something like M V T technique where I have to create partitions of variable sizes and

the numbers of partitions will also be variable that leads to a problem for accounting for the memory, main memory. But in this case in paged segmented memory management, though my basic scheme is segmented scheme, I am imposing paging technique on a top of segmentation. So what advantage I get? The main memory can be pre partitioned into a number of partitions where every partition will be of same size that is same as page size. That means number of frames that I have in the main memory that is fixed, size of every frame is also fixed. So I have a fixed setup, so the management becomes easy.

So in case of demand paging what is needed is whenever you start execution of a particular program, the page containing the first instruction should be in the main memory. I need not have immediately the second page in the main memory. Only when execution of the first page is complete then only the second page will be needed and that may be due to one of two reasons. One reason may be that in the first page some instruction is a branch instruction and the branch will take place to an address, to an instruction where the instruction is in the second page or it may be in the third page and so on.

So in that case you have to move from the first page to second page or first page to third page, even though the execution of the first page is not yet complete. That means all the instructions of the first page has not been executed. and the second situation may be that there is no branch instruction, every instruction in the first page will be executed sequentially. So when you complete execution of last instruction of the first page then you have to move from the first page to second page. So only when I need an instruction from the second page, then only I move the second page into the main memory before that putting second page in the main memory is not necessary. So I am bringing the pages from the secondary storage to main memory on demand, so that is why it is called demand page. This is also called virtual memory system.

The reason being though my main memory is limited, may be the size of the main memory is less than the size of the job and that is the physical memory size but to an user, it appears that as if the main memory size is virtually infinite because I am not using the main memory always. So only when I need, I will put a page in the main memory and execute. So it appears that virtually main memory size is infinite, so it is called a virtual memory system. Now in this case what are the things that are needed? Firstly as long as the program, some page of the user address space is in the main memory, I need a page map table for accessing a particular location in the main memory. Now if the page that is requested is not available in the main memory in that case what to do? So I need some modification in the page map table. In page map table I will put it this way that if an address space, user address space is having a 100 pages. In the page map table I will have 100 entries, one entry for every page. For the pages which are there in the main memory they will contain the corresponding frame numbers and along with the frame number in the page map table, I will have one more field which I will call as interrupt field.

Suppose while execution of a program, the CPU generates an address where the address corresponds to say page number 5 and the page number 5 is not available in the main memory. I have a corresponding entry in the page map table, there the frame number

field does not have any meaning because the page number 5 dose not exist in the main memory but it is the interrupt field which will be important in that case. If we assume that interrupt field is a one bit field, in that case for all the pages which are loaded in the main memory will set interrupt field to 0. The pages which are not there in the main memory will set interrupt bit to 1. So whenever a page number is generated, you go to the base map table, check the interrupt bit. If the interrupt bit is 1 then you generate an interrupt. This interrupt will be called a page fault interrupt that is page which is being looked for is not available. So that leads to a page fault and you generate a page fault interrupt.

Following page fault interrupt what we have to do is now I know that the page is not available in the main memory. So page is there on the disc. I have to bring the page from the disc and put it into some frame in the main memory whichever is free. So for that what I need? I need the address of the disc, a part of the disc which contains that page. So once I get the address of the disc block which contains that page, I can simply read that block and put that page into a free frame in the main memory. So now I need two tables, one table as before will be a page map table P M T, where every entry in the page map table will be divided into two fields. One field is let us put it as I field that is an interrupt field and the other field is for frame numbers. The CPU generates page number and the offset within that page.

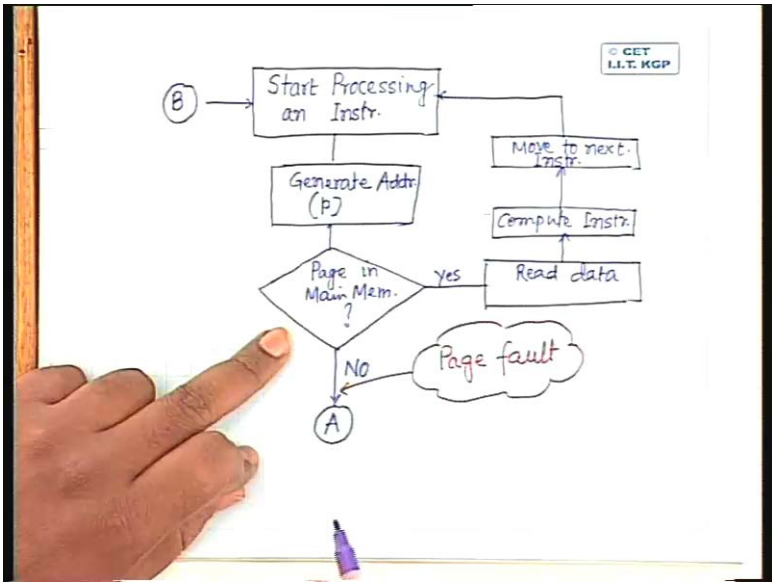
Now this is applicable both for segmented paged memory management or even for paged memory management because even for segmented paged memory management or paged segmented memory management what we are doing is we are breaking the segments into a number of pages. So in case of simple paged memory management whatever is applicable to different user programs in case of paged segmented memory management, the same concept is applicable for different segments. So we assume that we have been able to generate the page number and offset within the page from whatever address generated by the CPU.

Using this page number I go to a particular entry in the page map table. In the page map table what I do is first I check what is the status of interrupt bit. If the interrupt bit is 0 then I know this frame field contains a frame number where this page is loaded. So whenever interrupt bit is zero, this gives the frame number. Using this frame number and this offset within the page, so this is frame number and this is d which comes from here. I generate a physical address using this physical address, I go to the main memory and access the corresponding location.

Now in case you find that whatever page number that has been generated, using this page number, you come to page map table. The corresponding interrupt bit in the page map table is 1. So first checking we are doing on the interrupt bit whether the interrupt bit is 0 or 1. Now if the interrupt bit is 1 that means the page which has been referred is not available in the main memory. So I have to get that page from the secondary storage put into some frame in the main memory. So I have the secondary storage or the hard disc which contains that page for getting address of the page on this secondary storage. I need another table which is called a file map table or let us put it F M T or file map table. File map table actually contains the disc addresses of different pages. So using this file map

table I come to the disc, get the corresponding page from the disc and put this page into a frame which is available in the memory. So if I put this entire procedure in the form of an algorithm, it will look like this.

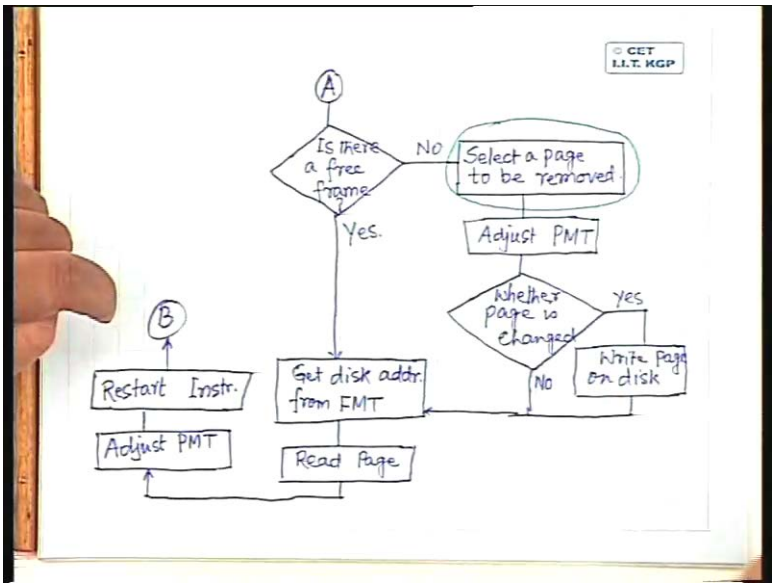
(Refer Slide Time: 00:27:11 min)




First you start processing an instruction. So while processing the instruction, you will generate the address for the data. So generate address, this address will give you the page number. Then you have to check whether this page is available in main memory or not. So I have to check page in main memory. If the page is in main memory then I have to simply read data. After reading the data, you have to compute the instruction and then you have to move to the next instruction and start processing the next instruction. So this is what has to be done if the page which is required while execution of an instruction is available in the main memory.

Now if the page is not available in the main memory, in that case what you get is what is called a page fault or page fault interrupt. So here it leads to a page fault interrupt. Now you have to do when you get a page fault, when I get a page fault then the first thing that I have to do is I have to check whether in the main memory, there is any free block or not or free frame or not where this new page can be loaded. If there is a free frame then there is no problem, I can simply get the page address on the disc from the file map table, read the page, load it into free frame. The other situation can be there is no frame in the main memory which is free, every frame contains some page. So in that case I have to forcibly evacuate one of the frames where this new page can be loaded. So in case of page fault what I have to do is so let me mark this as A where this page fault interrupts will start.

(Refer Slide Time: 00:30:33 min)




So in case of page fault, the first thing I have to check is there a free block or free frame. If there is no free block then what I have to do is I have to select a page from the main memory which is to be removed. So select a page to be removed. So once I select this page which has to be removed, next what I have to do? Because as long as this page was there in the main memory, the corresponding page map table had an entry for this page where the interrupt bit was set to 0. Now because I am removing this page from the main memory, so the corresponding page map table also has to be modified. So what I have to do is I have to adjust the P M T, the page map table where I have an entry corresponding to this page which is now being removed.

Now after this what I have to do is because I am going to remove this page, I have to check whether this page was modified while this was in the main memory because it may so happen that whenever I am loading a page from the secondary storage into main memory then I am working on that page. While working on that page, it may be the case that the page was modified. So some new data was written into the page. Copy of which is not available on the disc but if there is no change in that page, I already have a copy of that on the secondary disc. So I need not write this page back into the secondary storage but if the page is modified then before overwriting that frame with a new page, I have to store this page on to the second page storage. So what I have to check is whether page was changed. If the page was changed then I have to write the page on disc but if the page was not changed then I don't have to do anything. I can simply overwrite this frame with the new page.

So after that what I have to do is, so this completes one part and here what I have to do is get disc address from F M T file map table. Here this part will be identical. So once you get this disc address from the file map table then what I have to do is I have to read the page, page into the frame that has just been made free.

After reading this page, the next operation that I have to do is I have to adjust the corresponding page map table. Now once this page map table is adjusted, the page is already there in the frame that has just been freed or the frame which was already free. After adjusting this page map table, now what I have to do is I have to restart the interrupted instruction and with this I go to level B where B is this one. So this is the entire procedure that has to be followed in the demand paged memory management, that you start processing an instruction, while processing an instruction you have to generate the data address that is the page number. We have to check whether this page is available in the main memory.

If the page is available in the main memory then you don't have any problem. You simply read the data, compute instruction, move to instruction and start processing this next instruction. But in case the page is not available in the main memory then you have a page fault then the operations that you have to perform during page fault are this. First because there is a page fault, so this page has to be brought from the secondary storage into the main memory. So first you check whether there is a frame in the main memory which is free. If there is a free frame in the main memory then I don't have any problem. I have to simply get the disc address of the page that has been referred from the file map table.

Once I get this disc address, I can read the page and fill, put it into the free frame which is available. So once I put this page into the free frame, I have to adjust the corresponding page map table because in earlier case, earlier the interrupt bit in this entry was set to 1. I have to make it 0 because the page, I have put into main memory. not only that, the frame field should get the value of the frame, the address of the frame which is used for loading this page and now you are ready with execution of the instruction. So I have to restart the instruction which was interrupted and then I go back to the initial stage that you start processing this instruction which is now ready for execution. The other case is I face a problem when there is no free frame in the main memory. if there is no free frame in the main memory then what I have to do is I have to select a page from the main memory which is to be removed and this frame which will freed will be used to load this page.

So I select a page to be removed. Once I select this page which is to be removed, the corresponding page map table has to be modified because now the interrupt bit of this particular page has to be made equal to 1 because next time if any process requests for this page, that process will know that the page does not exist in the main memory. So, I have to adjust the corresponding page map table then what I have to check is, I have to check whether the page which is being removed was modified or not while it was in the main memory. If it was not modified then I don't have any problem because a copy of this page is already available onto the disc. I can simply overwrite this frame by this new page. But if it was modified in that case, the copy that I have on the disc is an unmodified copy, unchanged copy. So this modified copy I have to store back onto the disc before overwriting this frame by the new page. Otherwise all this modified data will be lost.

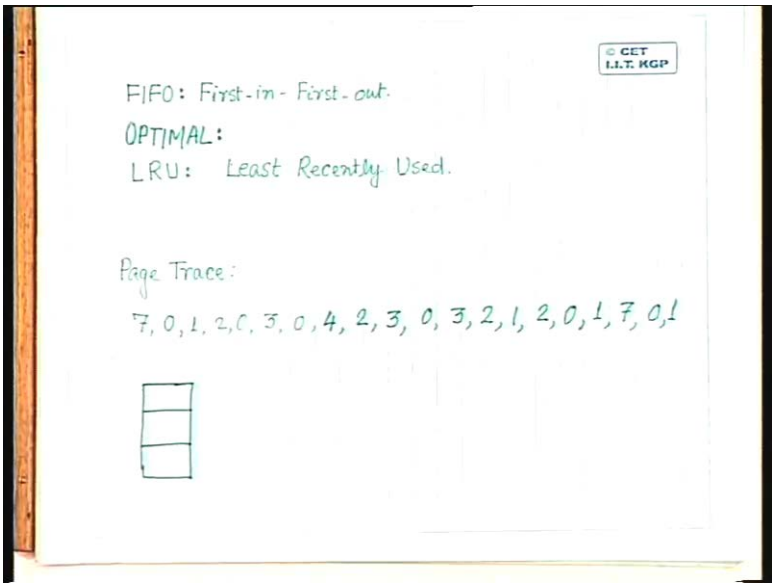
So if the frame was changed then you write the page onto the disc after that I follow this procedure that is you get the disc address of the new page that is to be brought from the

Secondary storage into main memory from the file map table. Once you get the disc address, you can read in the page, put in the frame which was either already free or has just been made free. So once this page is put into that particular frame, I have to adjust the page map table for this page because earlier the interrupt bit was 1, now I have to make it equal to 0 and the frame number while this page is been loaded has to be put into the frame field of the page map table. And with this now we are ready for execution of this instructions, so we restart the instruction that was interrupted and restart means I come back to this and start processing the instruction. So this is the entire operation that has to be done in case of demand paging technique.

Now here we find that a particular operation which decides the performance of the demand paged technique or the performance of the virtual memory technique is this particular block. that is you select the page to be removed. Now how do you decide that which page has to be removed? It may so happen that the page that I select for removal, may be the next time the same page is going to be referred. So if another process refers the same page immediately next, in that case again that will lead to a page fault. So again I have to bring this page from the secondary storage into main memory then only that process can start execution. and if this happens quite frequently, in that case you find that I have, what you said is frequent swapping in and swapping out operation and disc being slower in speed compared to your main memory or CPU, if you have frequent swap in or swap out operation that is going to heavily affect the performance of the computer.

So I should have a selection criteria, so that I can reduce the number of page faults or reduce the number of swap in, swap out operations that is to be continued, that will take place. So depending upon different criterias and this is also called a page removal technique or the criteria that will be used at different page removal criteria or page removal algorithms, I can have different kinds of algorithms. The simplest kind of page removal or page replacement algorithm is called the FIFO or first in first out page removal algorithm. So there are in general three kinds of algorithms which are talked about.

(Refer Slide Time: 00:41:47 min)




One is FIFO that is the simplest kind of page removal algorithm which is called first in first out that means a page which has been brought into the main memory first that is page which is to be removed first. So in that case what you do is whenever I find a situation that some page has to be removed from the main memory, to make a frame free to bring in a new page then out of all the pages which are there in the main memory, I find out which is the oldest one. Then what I simply do is I simply remove the oldest page from the main memory and make that frame for loading the new page. The other kind of page removal technique is an optimal page removal technique and you will see that this optimal page removal technique is the best one but though practically this cannot be implemented, we will see that later.

So an approximation to optimal page replacement is called an L R U or Least Recently Used page removal technique. This is an approximation to optimal page removal technique and this is something which can be implemented. So we will see the relative performance of all these page removal techniques with the help of a sequence of page differences and that is also called a page trace. So what you do is you take a random sequence of page numbers and see if the pages are referred following that random sequence then with the help of these different page removal techniques, how many times you get page faults. So whenever a page is referred by an user program and the page is found in the main memory that is called a page hit. You are hitting the page. If the page is not available in the main memory that is called a page miss.

So accordingly we can define two terms, one is called hit ratio. Those are basically performance measures. One is called hit ratio and the other one is called miss ratio. Hit ratio is the ratio of the numbers of the number of hits you have to the total number of page access that you have made. Similarly miss ratio is the number of misses divided by the total number of page access.

So if we will consider a page trace something like this and with respect to this page trace, we will try to find out that what is the relative performance of these different techniques. So the page trace that we will consider is something like this, so 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1. So we will take this particular page trace and apply these different techniques on this page trace and we will also assume that initially we have say 3 frames in the main memory which can be used for loading these pages. So I have to load all these pages in this sequence, only in these 3 frames in the main memory. So, that we will do in the next part of the lecture.