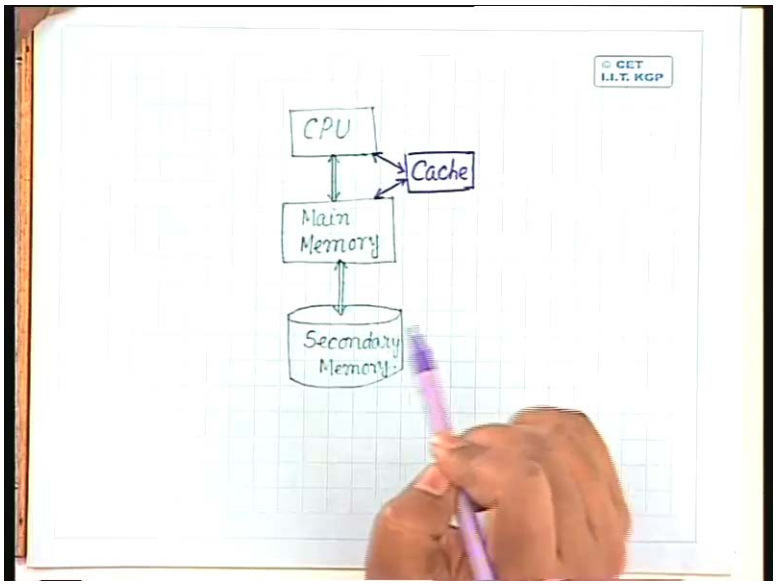
Digital Computer Organization
Prof. P. K. Biswas
Department of Electronic and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur
Lecture No. # 13
Memory Organization – I

Till now we have discussed about the CPU architecture and the CPU organization. Now in a computer system, CPU is just one of the components. You know that in a computer we have to have the CPU, the memory and the input output device and these three components taken together forms the entire computer. Now when you talk about the memory, the memory organization is something like this.

(Refer Slide Time: 00:01:29 min)

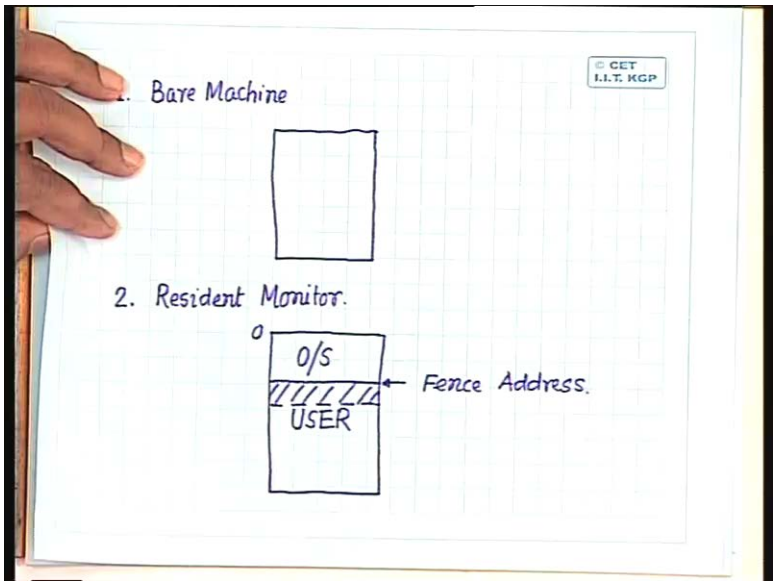


I can say that at the top most level, we have the CPU. Then this CPU has to interact with the memory for getting the instruction as well as data. So just below the CPU, we can put a part of the memory which we will call as main memory. So the CPU directly interacts with the main memory that means whenever an instruction is to be accessed for execution or a data is to be read or a data is to be written into all this access the CPU makes is with the main memory. The CPU does not have any direct access with the disc and at the next level in the hierarchy, we can put the disc or secondary storage. So this we will call as secondary memory.

Usually the size of the secondary memory is much larger than the size of the main memory and the secondary memory is very cheap compared to the main memory. So that is why cheap as well as slow. Now because secondary memory is very cheap, we can have huge amount of secondary memory which mostly satisfies all the practical requirements. The main memory being costly, it is not as large as the secondary memory. Of course in between the CPU and the main memory, we can have another level of memory which is called a cache. So it is the cache memory which comes in between the CPU and the main memory.

So we will talk about cache memory later. Right now let us consider what is the organization of the main memory that is usually used and we will, for the time being we will assume that CPU will have direct access over the main memory. Now when you come to main memory organization, there are various ways in which the memory can be organized.

(Refer Slide Time: 00:04:21 min)



The simplest form of organization of a main memory is called a bare machine. I think in your computer design laboratory, you had developed some microprocessor kits where you had to write the monitor program for execution of a program, for storage of elements, for accessing any function, for everything you had to write your own monitor program. You are not given any monitor program. That means the entire memory space was available to you. Now you had to decide that which part of the memory space will be occupied by the ROM and which part of the ROM will contain the monitor program. Isn't it? So a bare machine concept is just what you have done in that laboratory. That is the entire memory is given to the designer, there is no partition in the memory.

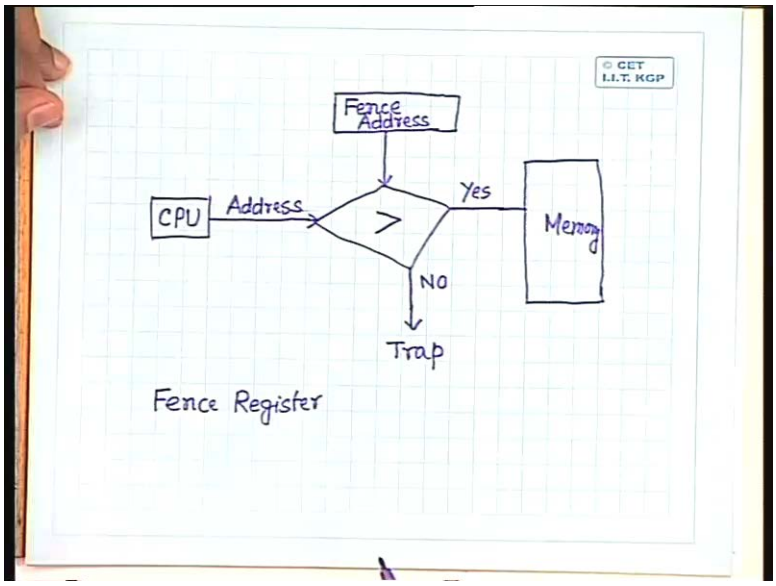
Now it is up to the designer to decide that which part of the memory will contain the monitor and which part of the memory will be used as scratch pad that is for storing of temporary data, for storage of raw data, for storage of processed data, everything. So everything has to be decided by the system designer. Usually this is the type of memory organization which is used for implementation of dedicated systems that is if I wanted to design a computer which is not to be used for general purpose computing. I can have a computer for controlling some power plant or for controlling some process. In that case the computer that I will use for that control operation, I will not want that computer to do any general purpose computation. So there, because the entire task is well defined, the designer can decide how to organize the memory so that the memory organization will be more efficient. So that is the bare machine concept and obviously in this case, this is a type of memory organization which is not very suitable for general purpose computing.

So for general purpose computing, the type of memory organization that can be used is what is called a resident monitor organization. What is this resident monitor organization? In case of resident monitor, again I can assume that I have a memory module. Now when I say memory module, this does not mean that it's a single memory chip. A memory module can consist of multiple number of memory chips but all those chips taken together that is the entire memory space that is available, I can consider that to be a single memory module. In case of resident monitor system, the memory is partitioned, divided into two partitions. One of the partition is occupied by the monitor program or the operating system and the second partition contains the user data or the user program. We have two clear partitions, one partition is occupied by the operating system and second partition is occupied by the user program. Now in this case also usually when I come for general purpose computing, in your design what you had done in the computer design laboratory for storing the monitor program you made use of some ROM. But in a general purpose computing it not the ROM. It is the same RAM which is also used for storing the user program and user data.

So usually what is done is the operating system usually decides on the secondary storage or hard disc. When you switch on the machine that is when the machine is booted, the operating system is read form the hard disc and put into this part of the memory. So this memory also has to be a random access memory. It cannot be a read only memory because we have to write the codes, the monitor codes into this part of the memory. Now this again being a RAM what can happen is the user program while execution, due to some error can try to access this monitor area or it can try to write something into the monitor area in which case the monitor itself will be corrupted. So you don't want the user program to be able to modify anything into the monitor area of the memory. So what we essentially need is protection of the monitor or protection of the operating system against any user program.

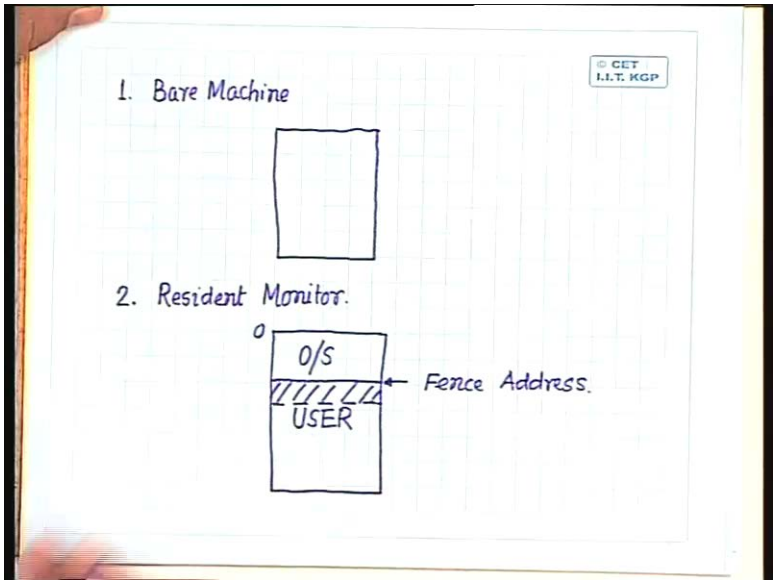
So how that can be done? Whenever the user program tries to access any of the memory location, the program will generate the address of the memory location which is to be accessed. Now if the part of the memory or the partition within the memory which is occupied by the operating system or the monitor is well defined. Then I can check on the address that is being accessed by the user program. If essentially what I have to have is what is called a fence address and now the strategy will be something like this.

(Refer Slide Time: 00:10:32 min)



While execution of a user program, the CPU generates the address. The address of the memory location that is to be accessed while execution of the user program. I also have a fence address which is the boundary between the user area and the monitor area. So I have to compare this fence address with the address that is generated by the CPU. So I have to compare these two addresses.

(Refer Slide Time: 00:11:25 min)



Now coming to this figure, you will find that if I assume that this part of the memory address, here address starts form zero that is the operating system occupies lower part of the memory. So all the addresses of the user area will be greater than the fence address. So simple comparison

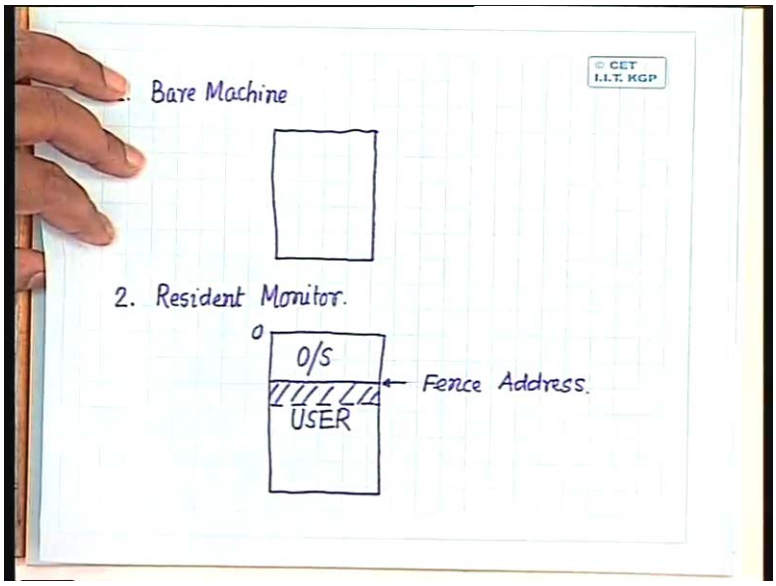
that I have to make is the address generated by the CPU, whether it is greater than fence address or not. If it is greater than fence address then only I know that the user program is trying to access user area only. In that case the address is a valid address and the user program will be permitted to access the memory. If it is less than or equal to the fence address that is the address is not greater than the fence address, then obviously I know that the user program is trying to access the monitor part of the memory which is not permitted. So if address is not greater than the fence address, in that case some interrupt is to be generated and let us call this interrupt to be trap and following this trap interrupt, the user program has to be terminated because it is trying to access a memory which is not legal. So you have to make use of this simple strategy to protect the monitor part of the operating system from the user program and it's a very simple strategy. You find that this can be implemented in one of the two ways. It can be implemented using software, it can also be implemented using hardware.

If you try to implement this using software then the advantage that you get is the fence address becomes flexible but the disadvantage is whenever a memory location is to be accessed by the CPU that means for every memory read or every memory write, I have to compare whether the address generated is greater than fence address or not by execution of some program. So effectively the memory access becomes very slow. So to avoid that what can be done is we can have a comparator, hardware comparator and using hardware this address generated can be compared with the fence address. It is the hardware comparator which will decide whether there will be a trap or the user program will be allowed to access the requested memory location.

Now if I use hardware and fence address also I implement in hardware, in that case fence address becomes fixed. Now what is the problem that we face if we make a fixed fence address in hardware? In that case this particular location is fixed so I cannot upgrade the operating system. because today if I load an operating system which may need say 40 kilobytes of memory and few days later I want to upgrade the operating system which will need 60 kilobytes of memory. So if this much is 40 kilobytes, a part from the user program will also be occupied by the upgraded operating system. So this part will be 20 kilobytes which is beyond the fence address that means this portion of the operating system is no more protected. So what we want is we want a fast comparison using hardware. At the same time we have to have a flexibility so that the fence address can be changed. So the suggestion can be that instead of implementing fence address in hardware, let us have a special purpose register called a fence register. So we will make use of a fence register. So whatever is the content of the fence register that is the fence address.

So now this comparator input will be this fence address will come from the fence register and this address will obviously come from the CPU. Whenever I upgrade the operating system, in that case what I have to do is the fence register content has to be changed. So there must be some privileged instruction using which we can change this fence register content. The fence register content cannot be changed by anybody because in that case there will be chaos. So there must be some privileged instruction using which you can change the fence register content and that gives you the flexibility of upgradation of the operating system. Now there is another way in which this protection can be obtained.

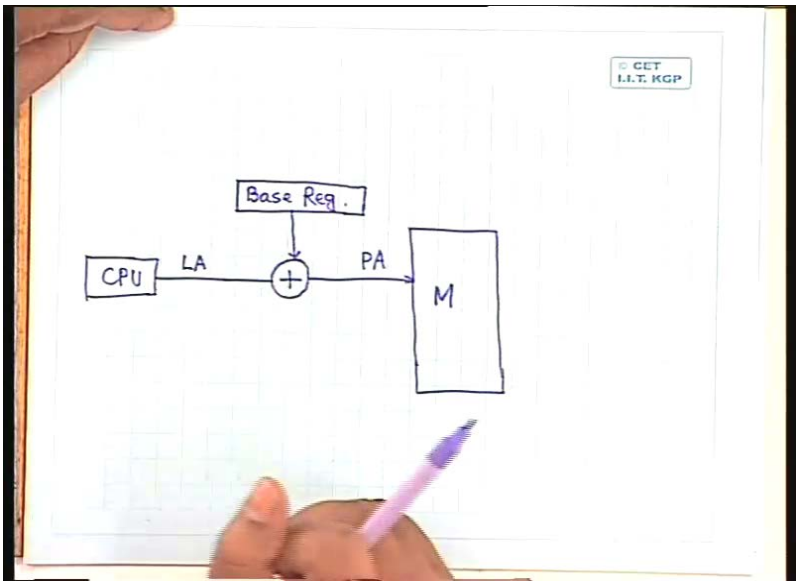
(Refer Slide Time: 00:16:46 min)



In this case we have said that we have a fence address. Instead of this, what I can have is I can have something called a base register. The CPU can generate the logical address and that is more logical because whenever you write a program, the program is compiled by the compiler to give you the executable code. Now if the program directly has to give you the physical address within the main memory where the data is to be loaded or where the code is to be found, in that case the compiler has to know that where from the program will be loaded in the main memory which the compiler does not know beforehand. So that is why the addresses which are generated by the complier they are all logical addresses and this logical address assumes that always the user program will be loaded from location zero logically.

Now with that logical address you have to add a base address. The base address will depend upon where from in the main memory, the program is loaded. So to that logical address, if you add the base address what you get is the physical address in the main memory. That means now the fence address can be replaced by the base address and the CPU will generate the logical addresses zero onwards. So this scheme will now be slightly different instead of fence address, we will have a base address and the base address will be stored in what is called a base register.

(Refer Slide Time: 00:18:20 min)



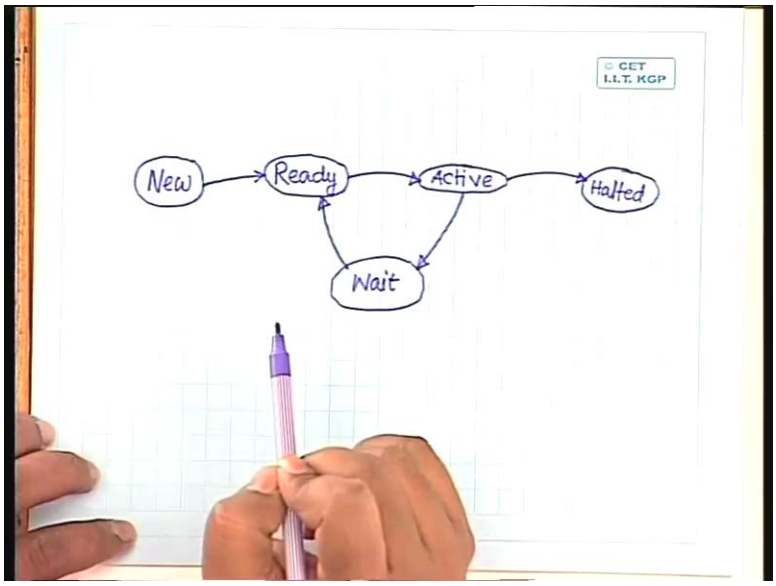
The CPU will generate logical address. Let me put it as LA and this logical address can never be less than zero, it has to be zero or more. So whenever the CPU generates the logical address, you add this logical address with the base register content and that gives you the physical address in the main memory where the instruction of that data will be found. So you will find that if I go for this scheme, I don't need any comparison because the base address content can be same as the fence address. The CPU is generating logical address which is zero or more that is added with the base address. So this ensures that whatever physical address that you generate, that address will always be greater than the base address. If I set the base address depending upon the monitor program size or the operating system size, this scheme ensures that user program will never access the monitor part of the memory.

Now still this is not the organization that we want because as it appears that when I said that main memory is divided into two partitions. One is given to the operating system and another partition is given to the user that means this is an organization which is suitable for single user system. In a single user partition, I can load only a single user program at a time. I cannot load more than one user programs at a time. But typically what we want is a computer should be able to execute more than one instructions at a time. So before I go into the memory organization which is needed for that, let us see what we actually mean by a multiprogramming system or a multiuser system.

When I work on a work station or on a server, you know that many of you can work simultaneously on a computer which was not possible in earlier computer systems like DOS based system, like PC XT or PC AT which were based on the DOS operating system, Ms DOS operating system. In those systems not more than one person could work simultaneously, Not only that, not more than one program can be executed simultaneously. Whereas in recent computer systems like Unix based systems or Windows based systems, whether it is Windows NT or Windows 95 or Windows 98 more than one user program can be executed simultaneously.

That means I must have something in the system which permits more than one program to execute at a time. Now what is that?

(Refer Slide Time: 00:21:52 min)



Whenever you compile a program and the program is ready for execution but you have not started executing the program. So that is a job which is called a new job. That means that the job is just ready, any time I can execute it. The moment I initiate execution of this job then it becomes a ready job. When the job becomes ready that means it is ready for execution, any time the CPU can start execution of this program. As we have said that a program or a data can be accessed by the CPU only when the program or data resides in the main memory whereas whenever you compile a program to make a new job, the executable code of that program resides on the hard disc.

In an Unix based system, usually the code which is generated is called as a dot out file. So if I want to execute that executable code, I give a command a dot out. When I give the command a dot out then the program becomes ready because a dot out on the disk cannot execute by itself. I have to give the command a dot out only when the program becomes ready and when it is ready it will wait for the CPU.

So what does this ready mean? Because the CPU cannot access anything from the disk, it has to access from the main memory that means whenever the program is ready, it must reside in the main memory. From the main memory I can have, in the main memory I can have more than one jobs and CPU will select the jobs one after another for execution. So even when many of you work on the computer system simultaneously, it is not that the CPU is working, is executing all your programs simultaneously. but the CPU is executing your programs in a time multiplexed fashion. But this multiplexing is done at such a fast rate that it appears to the user as if the CPU is executing your program only, it is not doing anything else but practically it is being executed in a time multiplexed fashion.

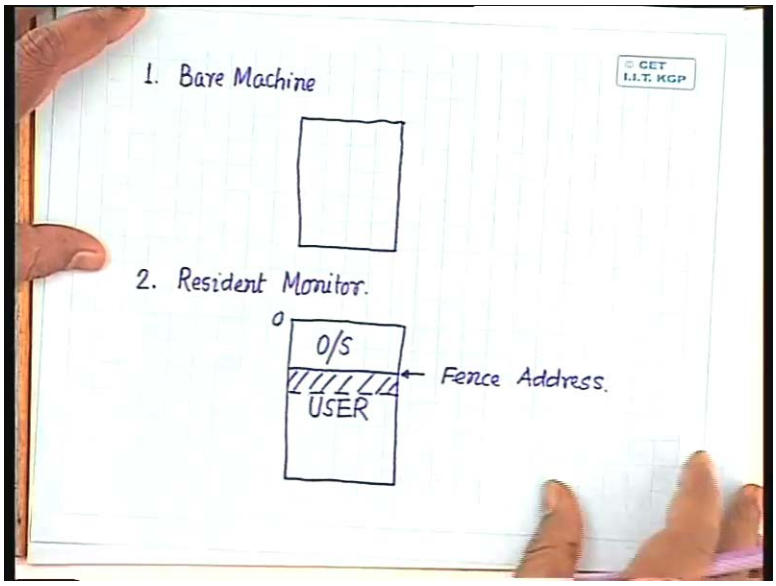
So whenever I give a dot out command the program becomes ready and so I say that the program has moved from the new state to the ready state. When the CPU actually starts execution of the program that means it takes a job from all the jobs which are ready, that program becomes an active program. So the job or the program moves from ready state to active state and finally when the program is complete and it comes out of the system, we say that the program is halted. Sometimes the program in execution is also called a process. So we say that the process is halted. So this is the entire path, entire states through which a process or a program has to pass.

Now in between there is another state that is while the program is active, sometimes it may need some I/O operation. If it is an interactive program, the program may sometimes wait for getting, wait to get some input from the user. The user has to feed the input from the keyboard or maybe the program will read something from the disk. So it has to initiate a disk read operation or maybe the program will try to give you some print out in between while execution. So it has to send some data, some output to the printer or maybe any other device. So while execution the program may try to access some device in between. So whenever the program tries to access some device that means it wants to perform some I/O operation during that period it is not making use of the CPU, it is waiting for that I/O operation to be complete. What the CPU will do during that time? The CPU has to execute the other program, it cannot remain idle.

So the first program which initiates an I/O operation, we say that the program will go to an I/O wait state. So from active state, the process will move to I/O wait state, when the process is waiting for some I/O operation to be complete and by after sending these two I/O wait state, the CPU will take another job, another process from the ready queue make that active and start execution of that. Now for the first program which was there in the I/O wait state, when the I/O operation is complete then the process will again try to access the CPU.

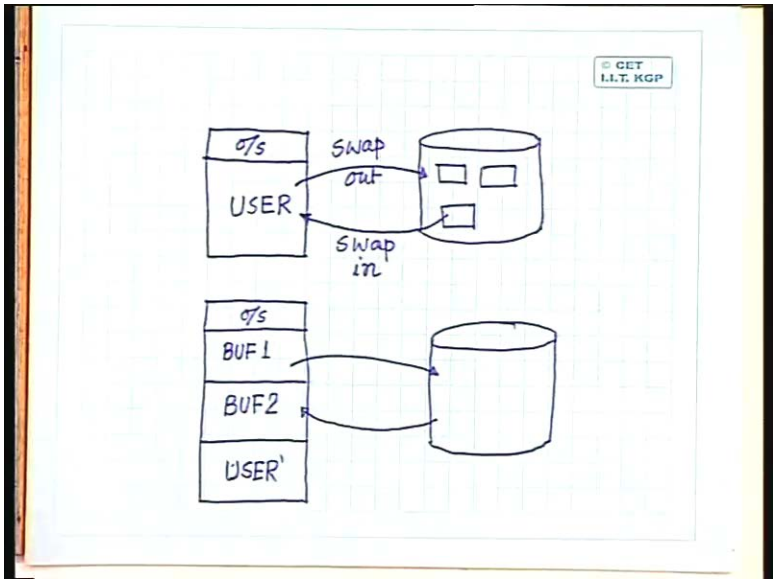
So what you can do is but the CPU is already executing some other process. So there are two ways either you can move from wait state to active state directly but which will not be very logical because CPU is working on some other process at that time and is expected that the process which is being executed by the CPU at that time will again go for some I/O operation later. So hoping that what we can do is you move the process from the wait state to ready queue or ready state. So in the ready state, it waits for the CPU to be free next time and the process which was being executed earlier that can move from active state to wait state. So these are the entire number of states through which a process has to pass before the process is complete or the process is halted.

(Refer Slide Time: 00:28:14 min)



So by this it is quite clear that I have to have a number of processes in the ready queue or a number of processes to be ready simultaneously for maximum benefit or maximum utilization of the CPU which is not supported by this resident monitor memory or the organization that we have said because here I have a single user partition and the single user partition can contain only ones user program. So one modification to this which can be done is something like this.

(Refer Slide Time: 00:28:33 min)



Let us have a single user partition that is a resident monitor system. A part of the memory is given to the operation system and this is the user partition and we want to use this same user partition to contain multiple number of jobs. That means whenever a job will be become active,

the job has to be there in the user partition. So what we can make use of is what is called swapping in and swapping out operations. So I have this secondary storage which contains the executable code of all the programs. So I will put it like this. So it contains the executable code of all the programs. There is one program in the main memory in the user area, the program which is active that is currently being executed by the CPU. At certain point of time, this user program wants to perform some I/O operations. So when it wants to perform some I/O operation, what we want is some other user program should be brought in from the secondary storage to the main memory which can become active.

So for doing that what I have to do is the user program which is already in the main memory that has to be swapped out to secondary storage. Now this user partition in the main memory becomes free. So once it becomes free, I can swap in a new process from the secondary storage into the main memory which can now be executed by the CPU. So we find that this simple configuration can be converted to a multi programming or multi user system by making use of the swap out and swap in operations. but you keep in mind that every swap out or swap in operation means disk access and a disk is a very slow device compared to main memory or compared to the CPU. So until and unless the swap out followed by swap in operation is complete, the CPU cannot start execution of the new program. That means during the time when swap out and swap in operations are continued, the CPU will remain idle. So though this solves the problem or converts a single user configuration to a multi user configuration but this is not an efficient solution.

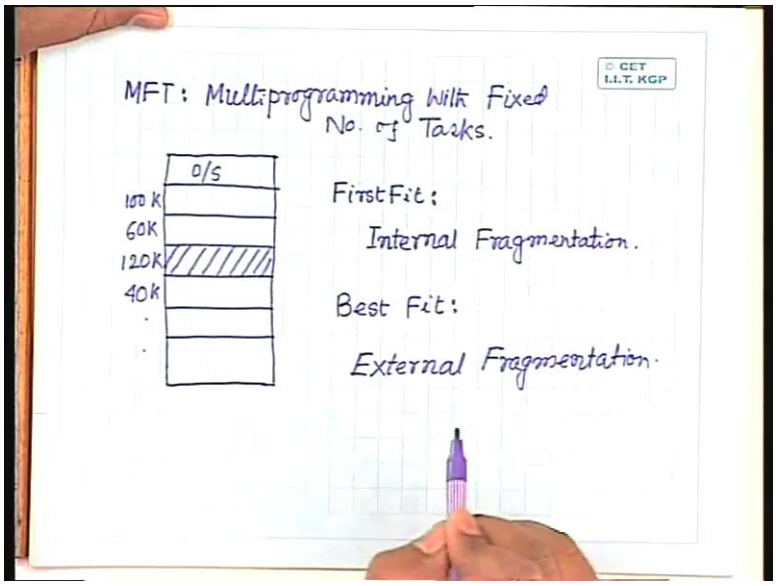
So the next hierarchy in the next higher level what can be done is instead of having a single user partition, let us have multiple number of partitions so that we can overlap CPU execution with swap in and swap out operations. So the simplest configuration that can be thought of, that was thought of was something like this. Let us now have 4 partitions. One partition as before will go to the operating system. The remaining space is divided into three partitions. One of the partitions we will call as buffer space one. The second one will be called as buffer space two and the third one is actually the user ready where a program is to be executed. As before we have the secondary storage containing the executable code of all the programs as well as all the data. So our assumption is the secondary storage is large enough so that it can contain all the executable codes which are to be executed on the machine. It will contain all the data which will be required by any process which will be executed on the CPU. So we are not putting any limit on this hard disk.

Now what can be done is an user program which is being executed by the CPU can be contained in this user area. The buffer area will contain, buffer one area will contain a process which is to be swapped out and buffer two will contain a process which is being swapped in from the secondary storage. So I will have a situation like this. From buffer one I will swap out a process on to the secondary storage, I will swap in a new process from the secondary storage, put it into buffer area two and simultaneously the process which is there in the user area that can be executed by the CPU. So I can have overlapped swap in swap out and CPU execution which improves the efficiency of the CPU and an improvement over this. But still this is not a very good solution because all these operations whether it is swap out operation or swap in operation or execution of a process by the CPU all of them will access the main memory. Now whenever the swap out operation is accessing the main memory, no one else can access the main memory.

Similarly when the CPU is accessing the main memory for execution of a program, none of these two can access the main memory. So though it appears that it is an improvement over this but the improvement is not very significant. What you have is a single memory module. In a single memory module I have single address bus, I have single data bus. So because it is single address bus and a single data bus, so even if I have multiple number of processes, one for swapping out one for swapping in, one for execution of the program but all of them cannot provide the address to the address bus simultaneously. only one of them has to give the valid address.

Similarly all of them cannot write the data, cannot send the data on to the data bus simultaneously or all of them cannot get the data from the data bus simultaneously. Only one of them can send a valid data onto the data bus. So there even the access of the address bus or the access of the data bus has to be multiplexed that means accessing of the entire memory has to be multiplexed. So these operations of swapping out or swapping in or execution of the user program they are not really independent they become dependent, one dependent upon the operation of the other. So even this configuration a multiple partitions does not give the advantage as we expect over this. So that means we have to go for further modification. So the next memory configuration that was thought which was an improvement over this is what is called MFT.

(Refer Slide Time: 00:35:58 min)



MFT means multiprogramming with fixed number of tasks. So this task, job, process or a program we will use interchangeably. So this task means a process. So MFT is multiprogramming with fixed number of tasks and this one that we have said, it is nothing but a special case of MFT where we have only 3 partitions. So in case of MFT, the organization that is used is something like this. This memory module is again divided into a number of partitions but in this case number of partitions are very high. So I have a large number of partitions in the main memory. The partitions may be of same size or different partitions can be of different size. one of the partitions as before will be given to the operating system and let us assume that this

partition may be of size say 100 k. this partition may be of size say 60k. This one may be of size say 120 k, this one may be 40 k and so on. So I can have different partitions of different size.

Now obviously because I have so many partitions in the main memory and every partition can contain an user process. so if there are n number of partitions, n number of different programs can reside in the main memory simultaneously and because there are n number of programs residing in the main memory, so the degree of multi programming of the system is n that means I can keep n number of processes in the ready state simultaneously and any of those processes can be executed by the CPU anytime. So all n processes are ready for execution. so you say that the degree of multi programming of this system is n. I have different partitions and once these partitions are met, they are fixed that means this partition size is 100 kilobytes and it is fixed. This partition size is 60 kilobytes and that is fixed.

Now whenever a new job is to be made ready that means, I have to transfer the job from secondary storage to main memory in that case that job has to be put in one of these partitions. So I have to look for, out of all these partitions in the main memory which partition is free that means which partition does not contain some other process already. So this new job I can put in a free partition, provided the size of the partition is more than or equal to the size required by the job. So in order to enable that I have to maintain a number of information's. The information is that for every partition what is the starting location of the partition, what is the size of the partition, I also have to maintain information about what is the status of the partition that means whether the partition is free or the partition is already occupied.

If it is occupied which process is occupying that partition, that information is also needed. So all those information's can be maintained in the form of a table. Now whenever a new job has to be moved from the secondary storage into the main memory, it will be the responsibility of the loader. I hope you know what is loader. So it will be the responsibility of the loader to see that which partition is free and what is the size of that partition. So if the loader finds that there is at least one partition which is free and the size is also more than what is required in that case the loader can bring the job from the secondary storage and put into the appropriate partition and after putting the job into this partition, that information table is also to be modified because earlier status of that particular partition was free. Now it has to be made occupied.

Now when you put a new job into this partition, there are two algorithms which can be followed. One is called first fit algorithm. What is this first fit algorithm? I can have a number of partitions which are free and the information of all of them are kept in the table. What the loader can do is loader can just scan the table starting from the first entry. Whenever it finds a partition whose size is more than the required size and the partition is free that partition can be given to that particular job. But what is the disadvantage? Wastage of memory. Suppose this 100 kilobyte partition is free, 40 kilobyte partition is also free and I have a job whose size requirement may be say 39 kilobytes. So because I am scanning that entry in the table starting from the beginning, I find that this is 100 kilobytes which is more than the size required and it is also free. So this 100 kilobyte partition will be given to the job but what will happen to the remaining 61 kilobytes? Because I have fixed partitions, that 61 kilobytes cannot be used by any other process. So the 61 kilobytes memory becomes wasted and that is what is known as fragmentation. In this case it

will be called as internal fragmentation because it is an wasted memory within a partition. So it is called internal fragmentation.

So though the first fit algorithm is very simple because I just scan the table entries from the beginning. Whenever I find a free partition of sufficient size, I allocate that partition to that job. So this allocation strategy is very simple but the problem is it can lead to more of internal fragmentation. So the improvement over this allocation strategy can be what is known as best fit strategy. In case of best fit strategy what you do is wherever I find a partition which is free and of sufficient size, I don't allocate it immediately. Rather what I do is I scan through all the entries in the table, check for all the partitions and I find out a partition, free partition whose size is nearest to the size that is required and more. So if I follow that strategy, you will find that though this 100 kilobyte partition and 40 kilobyte partition both of them are free, following this first fit I have allocated 100 kilobytes but following the best fit I will not allocate this but I will allocate this one, in which case out of this 40 kilobyte of partition, 39 kilobyte will be used by the process and 1 kilobyte will become internal fragmentation whereas in the earlier case 61 kilobyte was internal fragmentation.

So now amount of internal fragmentation by this best fit algorithm is always minimum but only problem is the complexity of this algorithm is very high compared to the complexity of this. So this leads to internal fragmentation. An internal fragmentation we have defined that which is internal to a particular partition. There is another kind of fragmentation which is called an external fragmentation. An external fragmentation is both the fragmentations or memory wastage. in case of internal fragmentation it is the memory wasted within a partition and external fragmentation in case of MFT technique will occur when I find that I have some jobs to be executed but the job cannot be fitted in any of the partitions because the size is less. So coming to this if I have say this 120 kilobyte is already occupied by some process and I have another job which needs a 105 kilobytes. I can have a situation that all these partitions are free but none of them is greater than 105 kilobytes. So though I have a job to be executed, I have a number of partitions which are free but this job cannot be fitted into any of these partitions because size of all these partitions is less than the amount of memory that is needed.

So again now this is a fragmentation because I consider this as a memory wastage. The total amount of memory is more than the size required but I cannot fit this memory because size of every partition is less than the size required. So again I consider this as an wastage of memory and the fragmentation arising out of this, we will call this as external fragmentation. So this external fragmentation is not part of a partition whereas the internal fragmentation is the fragmentation which is part of a partition because the entire partition is not used by the process which has been allocated partition.

Now if we modify this MFT technique, in that case it is possible that the internal fragmentation will be avoided altogether. That is instead of having partitions of fixed size, we can make partitions of variable size and partitions will be created as and when required. So if there is a job of say 120 kilobytes, I will make a partition of size 120 kilobytes not more than that. So by modifying this MFT to what is called MVT technique, I can eliminate or minimize; why minimize, I will come later on. I can eliminate this internal fragmentation. that we will do in the next lecture.