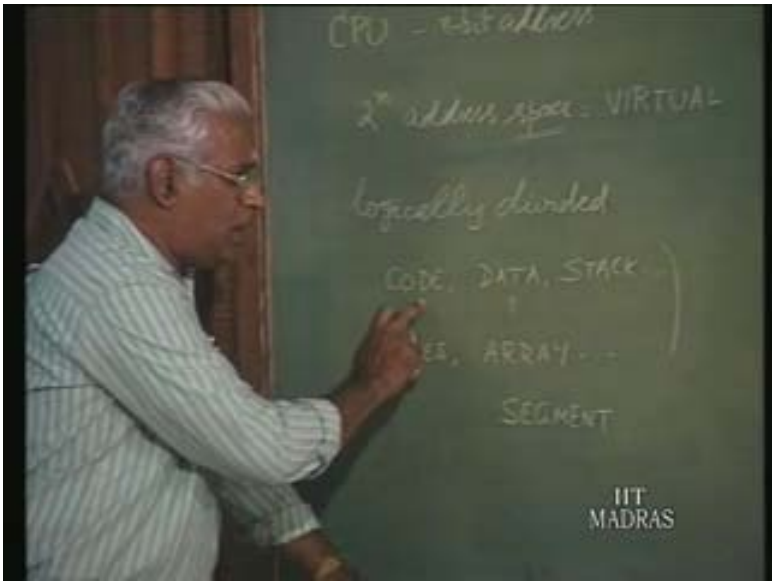
Computer Organization
Part – II
Memory
Prof .S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture – 23
Address Translation and Protection

In the previous lecture we saw about four different memory models, that is, basically they include segmentation and paging in different ways. Now let us quickly take a look at it as part of the program.

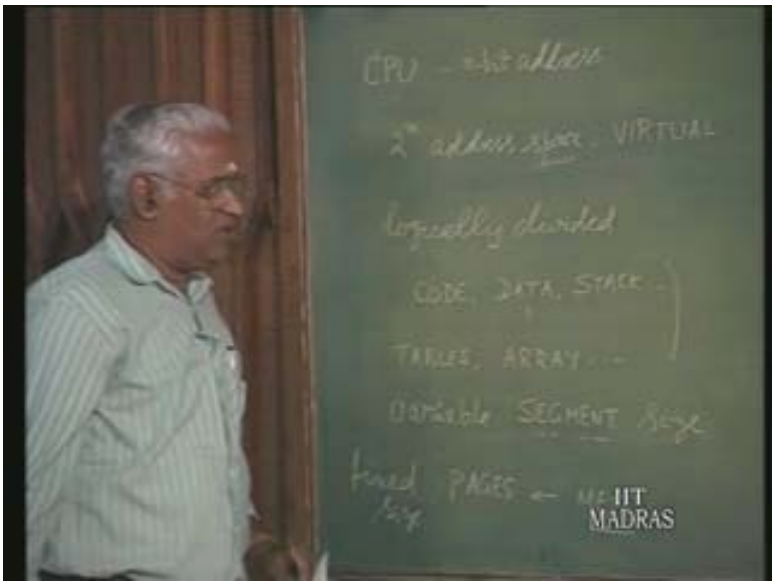
(Refer Slide Time: 04:29)



CPU generates an 8-bit address, so it is there as part of the program. When you have n-bit address, that is general, it does not mean that you are going to have those many memory locations. So in general again what you say is that the address space size is 2^n since we do not know that it may or may not really be available, that is, physically available. As far as the user is concerned, it is available and so we call this the virtual address space. What we mean is that if it is not physically available, this particular thing is going to be in the disk. We have seen this earlier; we had earlier seen different arrangements such as pages and segments. What is the rationale behind these things? Essentially this address space is logically divided into different portions in which functionally you will find one particular unit such as just a code or the program or the data in general. That particular portion may be used for some temporary storage that is stack and so on, or even if we take data structure specifically, for instance, we can have a logically cohesive unit, that is, a single unit for storing tables, some tables or array of data and so on and so forth.

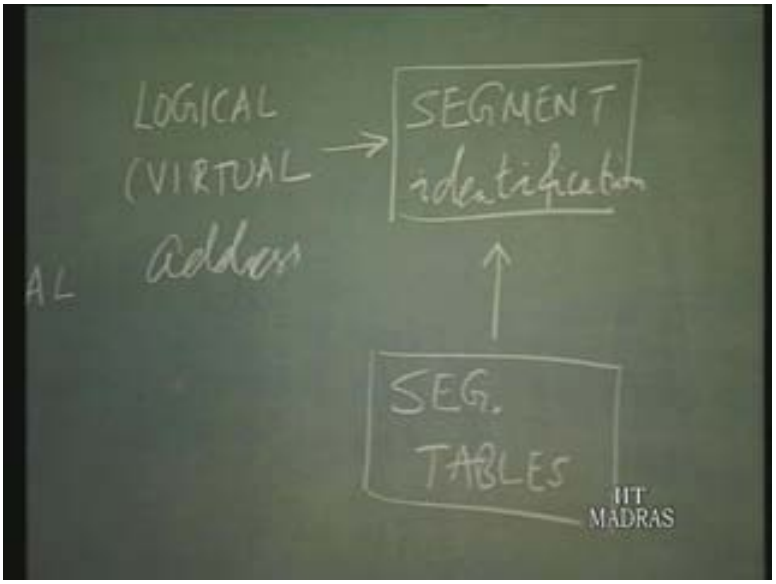
Now the programmer is really concerned with these because the logical part of it is more programmers. In contrast with this, what we are going to have is the physical part; that is, pages and physically existing memory and so on and so forth. Since we do not know about the size of each of these, that is going to vary from application to application and so generally we say that the particular space is logically divided into some units, and we call them segments. Within a segment, you are going to have one or the other type of information. So we talk about a code segment or a data segment or a stack segment and so on and so forth. But the size of it is going to vary from application to application and so we say that it is meaningful to have segment size as variable. We talked about a variable segment size. Now portions of the segments are brought into the memory so that the CPU may finally access them; well, unless CPU accesses, there is not going to be any program execution. So CPU accesses the memory and the portions of these will have to be brought into the memory. At that point in time, where we talk about pages, that is, in other words the portions of these segments are brought into the memory. We talk about pages, and so it is meaningful when we talk about pages to have fixed size, that is, in the physical memory, whereas this particular one could be essentially in the disk.

(Refer Slide Time: 05:50)



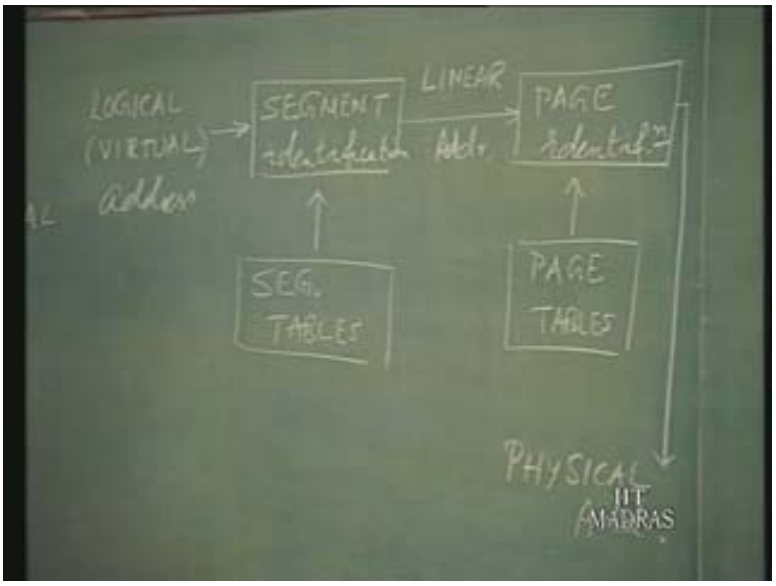
Now we have to do the address translation accordingly, because the starting point is what we may call a logical disk. This is the other term similar to virtual address; we start with that. So the CPU generates a logical or virtual address. This particular address goes and when the address goes the appropriate segment, to which this refers, must be got. In the first part, that segment must be identified – how to identify that or information segment identification. This information should have been made available earlier itself, say, in the form of some tables. So generally these are called segment tables. These tables have the information, and so, when a virtual address comes, the segment in which this particular thing is available will be picked up and so you can see that from the logical or the virtual address first there is one level of translation. Some people call this a linear address.

(Refer Slide Time: 07:06)



This is one level of translation and this will have to be looked into from the practical point of view, that is, in terms of the pages in the memory. The next one will be corresponding to this segment identification. What we have is page identification. Where is that information available? Again, it is in some page table, so actually these are all stored and made available to the system. So from the page identification now comes the final translation from the logical to physical address. In fact before we consider the segments, we started in fact as if this was the virtual address.

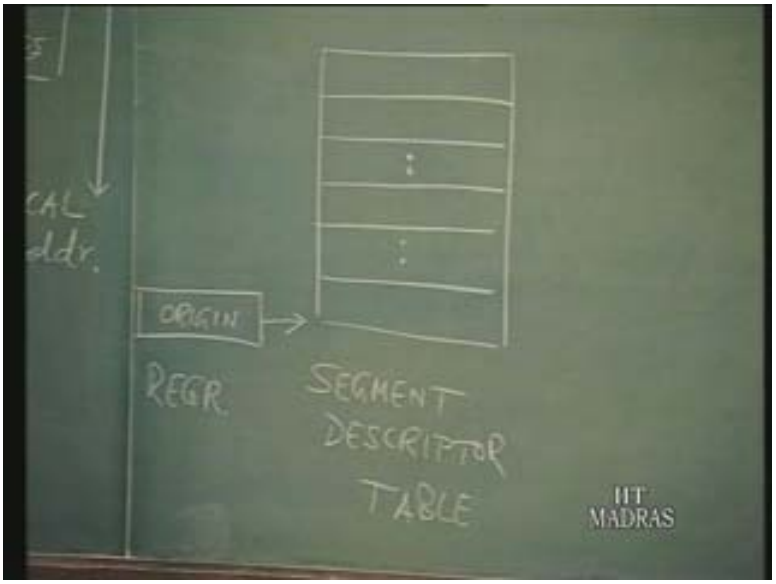
(Refer Slide Time: 08:29)



Remember the sequence in which we came to this? We were only talking about the translation from the logical or the virtual address to physical address and then we saw the reasons by having pages of fixed size there can be problem because it is meaningful from practical point of view to have a fixed page size, but then have variable segment size. So now, with the introduction of the segmentation at the segment, starting with the virtual address or the logical address to one translation for linear address, we come to physical address translation. Now let us just look into the details of this a little more. We start with the virtual address, which is the disk. Essentially that particular address will consist of two parts: we are calling that offset or virtual address, as we have already seen.

Virtual address is essentially going to identify a segment, and so, one part of it will be the segment offset and another part is going to identify the segment itself. So generally this one is called segment selector part, something which identifies the segment. Now the table is being made use of, so the table will essentially be in stored memory and there may be many entries in that. That table is generally called a segment descriptor table, that is, this one is going to talk about different types of segments, and for each segment there will be one entry. Some may call it a segment table; and since the segment table may be loaded anywhere, we also said there may be another register, which essentially gives where exactly this particular table starts. So this is a register; this is going to give the origin of the table, some address from where this originates. This is just a register; we will come back to this a little later.

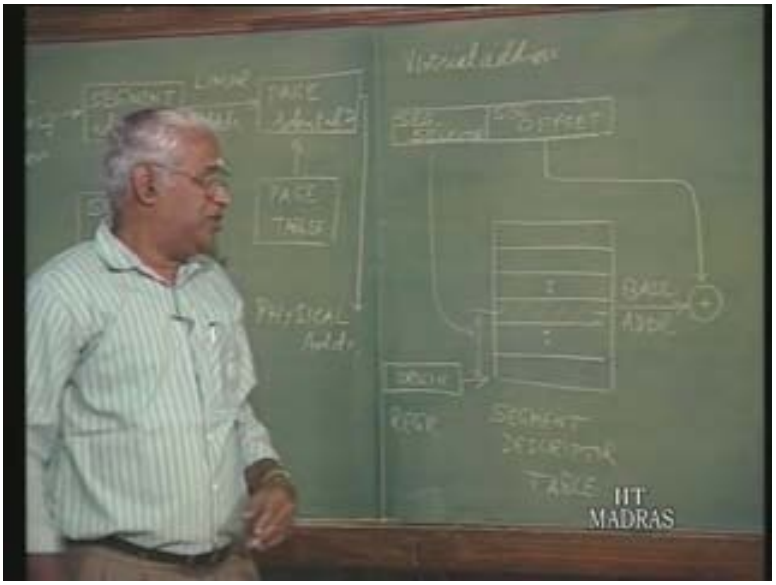
(Refer Slide Time: 11:44)



The register will tell where the table starts and the segment selector will identify it because it is going to be one entry; not necessarily 1 byte, may be a multi-byte entry, one entry for each segment. So the segment selector will identify the segment. Let us say if this is the segment that has been identified, and this is the entry which corresponds to the segment that has been identified by that, so essentially from the start address this information this what is going to be indexed by the selector part.

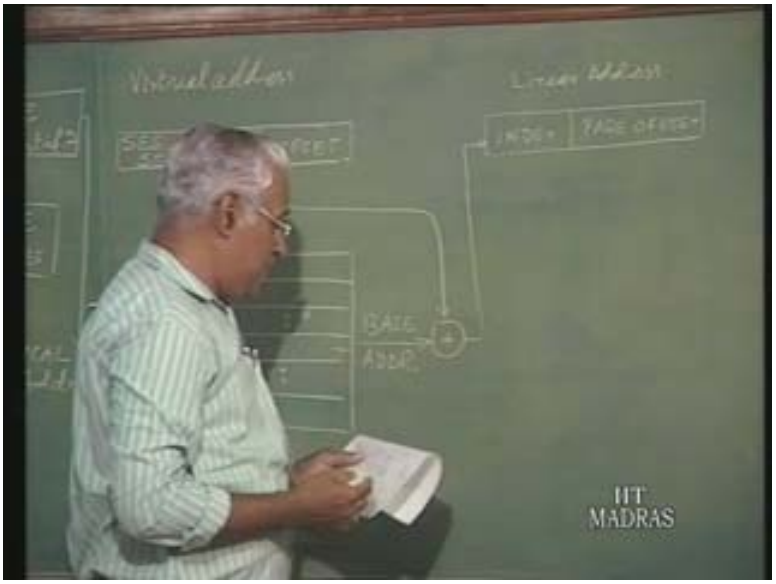
There is one segment identified; essentially this is going to give this. So instantly what is this origin address I said? This is the start address of this table indicated from the start address how many n entries a way this particular thing is identified. Now essentially there will be many entries in this; I said it is a multi-byte entry; there may be many entries. One that is useful from translation point of view is there is one address called the base address that is given. That is the base address that will be used. This base address will be added to the segment offset, which is basically the entry which describes a specific segment.

(Refer Slide Time: 13:31)



So this base address is nothing but the start of that segment this tells within that segment how many bytes a way this specific address is pointing to. So the output of these, that is, the sum of these is the linear address which we are talking about this. What we have done is we have gone into the detail of this. So the linear address is generated like this and what is that linear address? By using the linear address and then finally the physical address, the page is going to be identified. So essentially this one will consist of a page offset, that is, offset within the page and remember in our earlier calculation we had called this a virtual page? Now it is not a virtual page but some index will be there. So I will just call it some index; this is the one which will identify the page. Like here it was identifying the segment, this will identify the page – both are indexed inside and so we know this is it not page table.

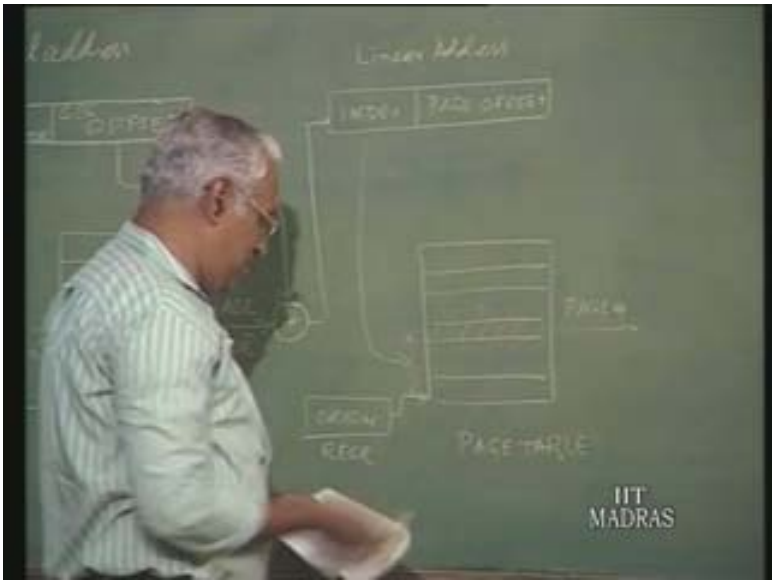
(Refer Slide Time: 15:04)



Here is the page table. Similar to the other arrangements where the start address of the segment table is indicated by a register, here also we can have another register, which similarly gives the origin of this page table. If that is so, then what is the page table? Page table contains an entry for each page. So the relevant entry for this address will be got; basically we should know this one. This will index into the page table and it is similar to what we had in the segment. This will basically identify the page and this particular page table, among other things for instance, may straightaway give the page physical page address. That was the thing that was available from here, or it may just in other words it would basically going to tell which page. So let us say each identifies the page, in which case, it is the page number. Can you see that there is no difference between that and this? Here we were talking about start address of the segment, here this is nothing but start address of that page; it is not different.

Now we know that the physical memory is divided into fixed pages size. Suppose this is our starting page number 0 – it is fixed – so that is page number 1 and so on and so forth. So a specific page is from page number n; that is what this particular thing is going to identify. Let us say this is page number n; it is going to identify this as fixed size. This is our physical memory; we are just putting it as memory, nothing but physical memory or DRAM as we were calling it. So page number is identified; and this page offset, which is just an offset within this one specific address.

(Refer Slide Time: 17:20)

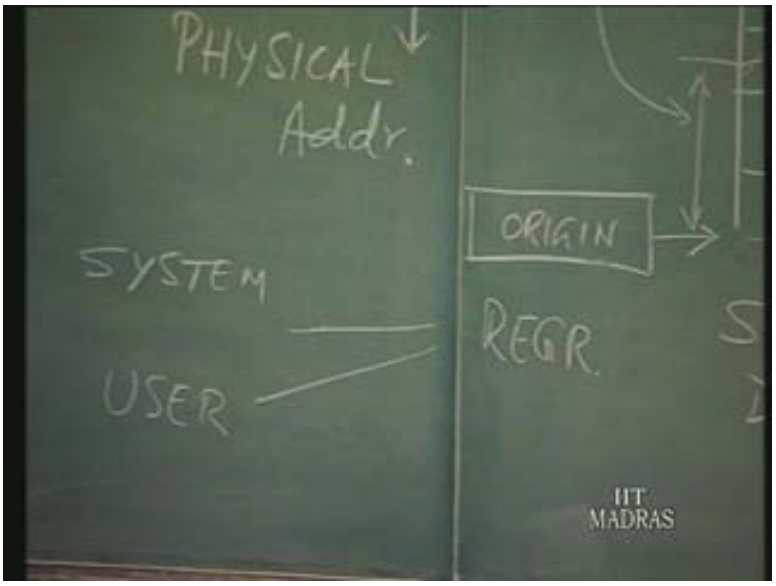


This is one specific location address and this particular location may contain an instruction or a data. Ultimately for identifying the contents of a location which will be an instruction or a data, all these things are done. Now the page table will be having information on whether a specific page is available; if so where it is available and so on. Because we said earlier any page, now this page table will have entries for all; originally we had called it virtual pages all the pages may or may not be physically present in the memory. This will give information about whether a particular page is present or not. If it is present then it will also tell is the location and so on and so forth. So now you can see the translation from one to the other. Finally, where is our physical address? This syntax, whatever is indicated, this syntax is the physical address or the location; this syntax is the address. That is what we have seen from the logical address translation to linear address, from linear address to physical address. So throughout you can see that essentially an address vector will consist of two parts: one thing we will which will identify the segment and page, and another part which will identify specifically the location within that segment or page. So it is like having an address x y; x indicating the area and y indicating the street, something like that or x may be the street, y may be the specific door number or something like that.

We had earlier talked about protection. Let us talk about that in this context because earlier we were mentioning while discussing the page table that the page table entry will have the protection but after segment was introduced we said it may be better to have that protection information here rather than there. There is no harm; you can have in both places too if necessary. This segment descriptor table is going to give description of all the segments that are there. What are the segments? Well, apart from the user point of view such as the code data stack, etc., we know that there is a system and the system may be used by different users and the basic characteristics of system should not be altered at any stage. It is possible that the different users make use of the same system, programs that are available, in which case any user should not corrupt a system program also. So now you can see here we can talk about system programs and user programs. I will call these application programs, user programs or application programs.

We have system programs and application programs; that means both are software both are programs so we can have, for instance, one segment descriptor table for system and one for the application or the user, that is, you can have two different things so that we can always define where exactly all the segments related to system software are available. You can have one table there; you can have another table for the applications, that is, from the user point of view.

(Refer Slide Time: 23:37)



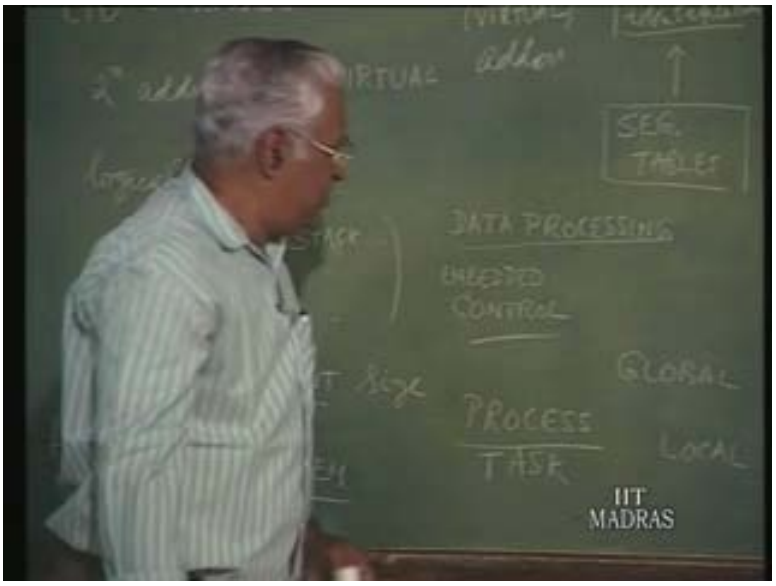
You can have two different registers itself. In fact specifically the system will be used by any one any user. We say that the system may be accessed in a global manner whereas all the things that are related to the user are more specific or you may say local to that particular user. It is local to that particular user. So we can have a global table, we can have local table, and these registers are in the CPU. In other words what we are talking about is the CPU architecture must include the provision of these registers; that CPU must support this. In other words, what we are talking about is the CPU having some extra features for offering protection at the software level. Now we really talk about a CPU having these features and possibly another CPU not having these features. You can just see we started our series of lectures talking about a general purpose CPU and a special purpose CPU, that is, in the sense we remember we were talking about it while talking about applications. We said it can essentially consist of spectrum of applications, spanning over a wide range such as data processing application on one hand and then control oriented applications on the other hand. Now when we talk about software essentially what we have is the data processing applications. A CPU which has this kind of features will support and protect system software from the user software and so on.

Essentially they are for the data processing application, whereas from control application it may not be that critical because once designed, the CPU and processor will go into the system itself. It will not be used by different types of users. For a specific application something is designed and so we even talk about an embedded controller, embedded controller, and the CPU that is used for such applications is going to be very special, very specific. It need not have these kinds of features.

Some simple way of protecting these things will do, because once the program is developed and loaded, it is not going to be altered. Now let us look into this. Suppose given a register such as this at different times if different addresses are loaded, what does it mean? It essentially means at different times different start address of this table is given, which means different tables can be created. Now it is meaningful possibly if we have ten users, each user can create his own ten different tables in addition to the table that may be used for the system or by the system. This way the entire address space can be split, that is, our entire virtual address space can be split among the multiple users and I said ten different users creating ten different tables. We have not really said what will be the size of the table; that could depend on the application. May be some people do not have many segments and some users may need more segments, and segment itself is of a variable size. So depending on the application they could have any.

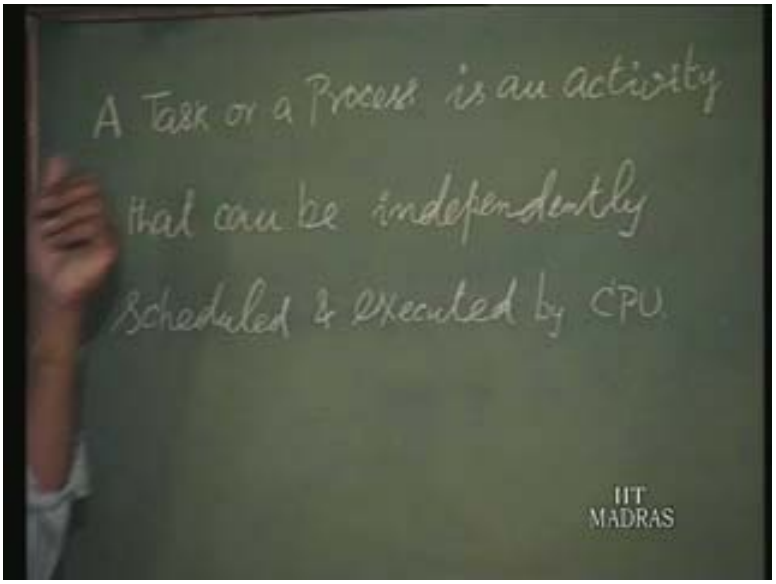
Suppose when a user loads his program his segment descriptor table will have to be referred to and so the corresponding start address will be loaded by the system. The system software will only ask for the user id and then the moment he is allowed to make use of the system, the system software will load that particular address and the user will be taken specifically to his allocated portion in the memory. In the segment descriptor you can have all the access rights, protections, privileges, and so on and so forth. Even among the users we want to talk about different privileges being given. For instance suppose a system is being used by a professor and ten students, whereas the professor will have access to all the ten different students' program area, any student is allowed only in his program area; so you restrict the rights. In other words the professor very much accesses the system but he also may be controlled by the inherent system software. So now what we are talking about is having different processes, that is, the other term that is used is also a task.

(Refer Slide Time: 30:17)



We will presently see what a process is and what a task is. So by having different tables and appropriate registers we can split your overall program; first we saw in general as a system user and more specifically as a task or a process. In other words even a single application program may consist of multiple tasks.

(Refer Slide Time: 30:42)



That is why we have the term like multi-tasking or CPU multi-tasking system. Now what is a task? Another name for that is a process; it also must be familiar with the multi-processor. That is slightly different because there is more than one CPU. That is multiple processors: more then one CPU. Here what we are talking about is a program that can be split; for instance, say two tasks, which will accept two different types of data and then something like integer numbers, one type, another one floating point number and then it may consist of say two different I by Os. So we talk about integer, processing task, floating point, processing task, then we talk about an I by O task one, I by O task two, you may be able to carry out some of them in parallel. What is a task or a process? A task or a process is essentially an activity of the CPU, which can be independently scheduled and executed by CPU and anything across the task will be done through some specific communication procedure, that is, in other words, a task or process is a program, an independent program by itself.

The overall application program may consist of multiple tasks but then communication between the tasks or the processes will be following some specific standard. This can be modularized, so that today a particular I/O task may be meaningful, tomorrow, depending on technology development, we can remove that and then put something else. So we can modularize this particular one. Now suppose such things come by having a set of tables, which are independent. First of all we said a user can have his own table, thereby think in terms of different segments. A single user can also have more then one task; that is also possible. You can keep referring to the different tables by loading the different addresses into this register. That is supported if the CPU has these features.

For instance, let us take this particular page table. Suppose instead of a single register such as this as shown, which we had discussed earlier, contains nothing but start address of the page table. Suppose this register is replaced by another table let us say instead of single register suppose we have another table, then just try to extend and that table may be pointed to by this a similar register like this. Suppose there is another table; instead of the single register like that, each entry in that particular table will act like this register, which means each entry can point to different page table. All these facilities will help very large software development because when different members in the software group are working and developing, they will need to have all these facilities so that they can create their own things and then some of these tables may be made accessible by another software developer, and some of them will be kept only for themselves, thereby the communication between these software processes can increase.

So these are all the features. Now register a table such as this and also some times special instructions meant only for this kind of register, special instructions which may not be used by the user – all these things must be supported by a CPU, which are meant for data processing applications. That is why you find that with CPUs which can support all these kinds of system features, which are needed for operating system support, they will all come under the data processing at the data processing end and somewhat complementary to this, you have the control application oriented processors.

Let us not elaborate any more on this. I hope now you understand a single processor, a single CPU – we are not talking about multiple processors here you are talking about a single processor dealing with different tasks or different processes. Multi-processing is not the same as the multi-processor; in the case of multi-processor, we are talking about multiple processors. In the case of a multi-tasking or multi-processing we are still thinking only in terms of single processor, which is capable of dealing with different tasks or processes, that is, at the software level only, it is multiple; at the hardware level there is only one processor. What is the restriction of one processor? One processor can handle only one instruction at a time; it can be in only one step. Bear this in mind all the time.

Multiple processor and multi-processor have a different story. In all our discussion, we have said memory or let us say storage holds the data. Memory or storage holds data. Should it always be so? Is it true for all types of applications? In general we say it holds data. Data is the term we are using not in any specific manner; it may be instruction or a data. And then for better utilization of the CPU, for the CPU's performance utilization may improve, we had talked about memory hierarchy; that is, we said memory holds something or storage holds. For instance, the pages is here, or let us say, a physical page is here, a virtual page is here, but CPU always deals with the fastest memory.

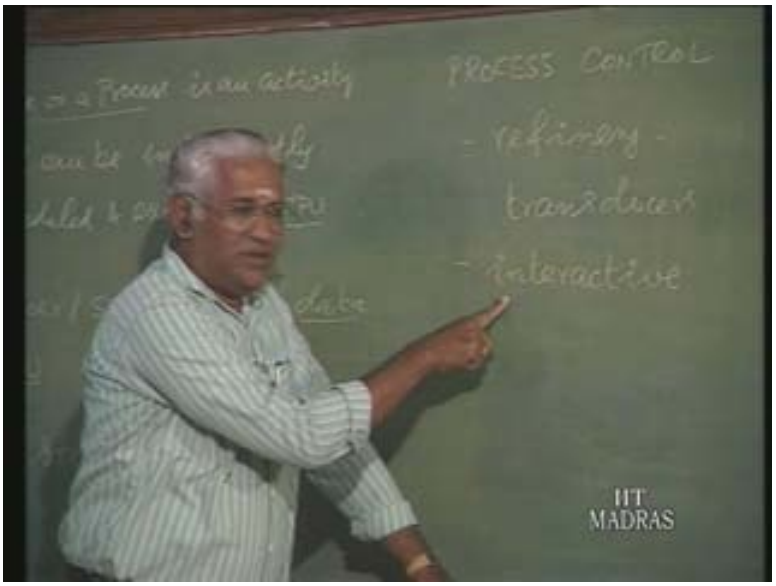
(Refer Slide Time: 38:12)

That is, from performance utilization point of view, we were talking about the memory hierarchy. But essentially memory holds the data; should it always be so? If it holds the data what type of data does it hold? It can hold only data which have been already prepared. By this, we may say that stored or frozen data is what memory holds. In contrast with this, what about raw, fresh data or let us say live data? We do have applications – raw or live data, which are being currently generated; it is something like fast food kind of a thing.

(Refer Slide Time: 39:22)

For instance we go to a petrol refinery application; that is, in a general process control application the data will be flowing to the computer, which is carrying out this controlling of the process, from something like thousands of sensors distributed in the field. They also carry data. So these data will have to come raw, fresh; they will keep coming. For this any process control applications can be used; I said refinery as one example. Essentially a lot of transducers placed in different places will pick up the data and then feed the system. There is another type of application, in which the user is interacting with the system; that is, he keeps looking at the result and then he is manually feeding in the information.

(Refer Slide Time: 40:46)



That is also a raw or a live data. It is not stored. In one case the transducers may keep checking the condition; that is, they are in the factory or the field and then automatically generate data. In the other one the data is generated manually. Now these data also will have to be processed; these cannot be stored in the memory. They can be stored provided the processing can wait but in many situations you will find that it cannot be stored; it will have to be processed immediately; that situation is very much there. So to support the flow of this data to the system, we need to have the appropriate input/output mechanisms in general or devices.

In other words, the data that is going fresh into the system will be handled by the input/output devices and what is the difference as far as the CPU is concerned? The CPU may take the data from a disk which is this system, in this part of storage, or it may take the data from the memory unit or it may take the live data from these I by O devices. As far as memory is concerned, it is the same but the important thing is most of these I by O devices are slow. I would say majority of them are slow. Even in the memory system we had seen storage is much slower than memory and even in the memory subsystem we say cache is the fastest, cache can match with the CPU but not the main memory. So when you go out from cache to main memory to storage to I by O devices, the CPU has to keep dealing with slower and slower data.

(Refer Slide Time: 42:08)

That is the difference that is very much there; nevertheless as far as the CPU is concerned it has to deal with the data in a similar way. So if you introduce this input by output devices into whatever scenario which we have discussed involving CPU and memory, what are the various issues that are there and how can these things be tackled? These will form the focus of our lectures in part three. In part one, we were essentially talking about the processor and processor related issues. From that we moved to part two of this particular series. In part two we first looked at different types of memory and then subsequently the CPU–memory interactions. Now in the series of lectures to follow, which form part three, we will take a look at the input/output devices or, in brief, I by O.