Computer Organization
Part – II
Memory
Prof .S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras
Lecture – 22
Segmentation

In the previous lecture we saw how the CPU's performance can be enhanced, that is, to reduce the miss penalty. We also saw how the performance can be enhanced by having different types of cache organizations and also we saw two examples with different cache sizes. It is needless to add that the sequence in which the addresses are generated will have a say but by and large you can say the cache size as well as the cache organization influence the cache performance, and in turn, affect or influence the CPU's performance. Now, let us go back and take a look at the page.

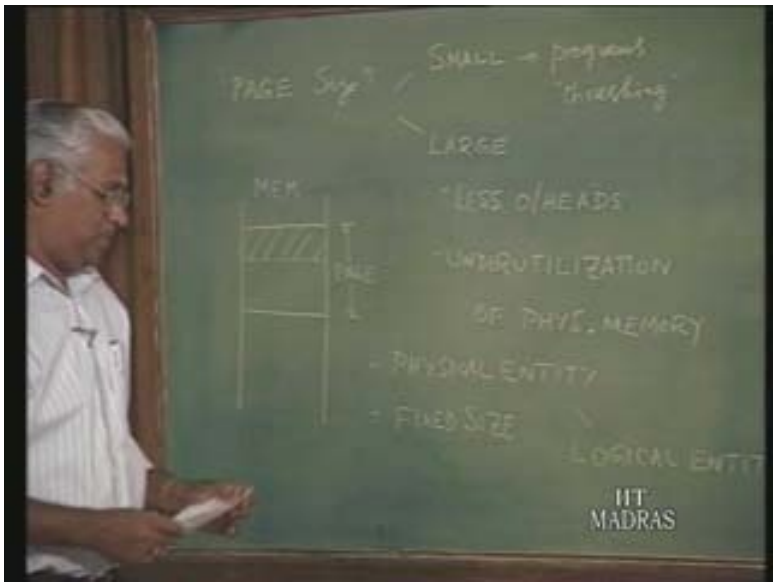
Page essentially page is a physical entity, like what we said about the cache size. Let us also take a look at what will be the influence of page size on the performance. Page is a physical entity, based on which, we have to fix the size of the page. We know that a page is swapped in or swapped out and we also noted that a program is said to be trashing. In case there are too many faults and too often you have to swap in and swap out the page, we made a note of that also. It will happen when the page size is small and especially if the program is fairly big enough. Then it will occupy more pages and the pages will be swapped in and out too often. Whenever a fault occurs, we had seen both in the case of cache and memory.

In the case of cache miss, at least the cache was faster, but in the case of memory when the page fault occurs, the penalty is very high because we are talking in terms of some milliseconds, because in the other one it could be in microseconds and maybe even nanoseconds. So now the page fault occurs when the page size is small and the whole program or meaningful parts of the program cannot be held in a page. When the page size is small then there is frequent page fault and we said this would lead to programs which are said to be trashing. This is the term we had used earlier. If you want to avoid programs which are trashing, that is, which cause page fault frequently, then obviously you would like to make page size large. This small and large are two fuzzy terms. How small and how large are things, which we have to decide.

What happens in the case of large page size, we can say that as far as the page fault is concerned, the occurrence will come down no doubt? So there will be fewer overheads; when you have large page size, the overheads will be less. What are these overheads? The overheads are associated with swapping in and swapping out the page, which really do not contribute to processing whereas in the case of small size page, the overheads will be increasing, but then, is having large page size a solution? Let us just see – we do not know about the size of programs.

Let us say you have a page of size 16 K and if there are programs which are not more than say 2 to 6 K, then what happens is just to accommodate a 2 to 6 K size program the whole page will have to be used.

(Refer Slide Time 10:04 min)



We talk about the whole page because that is one full unit, which will be swapped in or out. So if you take a look at the physical memory, you would keep seeing that. This is the physical memory; I am not specifically marking the locations. You will keep seeing large pages; let us say this is one particular page. Only a small portion of it may really be occupied by the program. Having a large page size will no doubt reduce overheads, for, what happens is that it also leads to under utilization of the memory space. This leads to under utilization of the memory space. Now you can imagine that, if, instead of having a 16 K page, if you had possibly say 4 K page, when a 2 K program is loaded only 2 K out of the 4 K will be under utilized or will not be utilized, whereas here, 14 K out of these 16 K will not be utilized.

So under utilization of memory space, which is really physical memory, is what we are talking about. Now if you have small size page, it will lead to programs which are trashing and more overheads. If you have a large one, the overheads will be reduced but the physical memory will not be efficiently utilized. So we really have to compromise between these two. Why is this particular thing happening really? What did I say about page? Page is essentially a physical entity and a page is of fixed size. That is in fact causing the problem – the page is of fixed size and there is a need to swap in and swap out one page always. There is no mechanism for filling in only half of this page. So with page being of fixed size, we are talking about this under utilization of physical memory. What will happen is that you will be having many pages and you will be having unutilized space. So you are not really making use of the available resource. So what is the alternate or what is the solution for this problem? How shall we go about? The best thing is always to see what we said about the page; we said it is physical entity.

Alternately, instead of some physical entity, why not go for a logical entity and take a look at it? That may provide a solution. We said page is of a fixed size and again essentially it is a physical entity. Now think of whether there is any logical entity. Given a program, users program, what are the logical things which we can think about? We can always think about the program part of it or the code part of a program, which essentially contains the code, otherwise called program or data part of it.

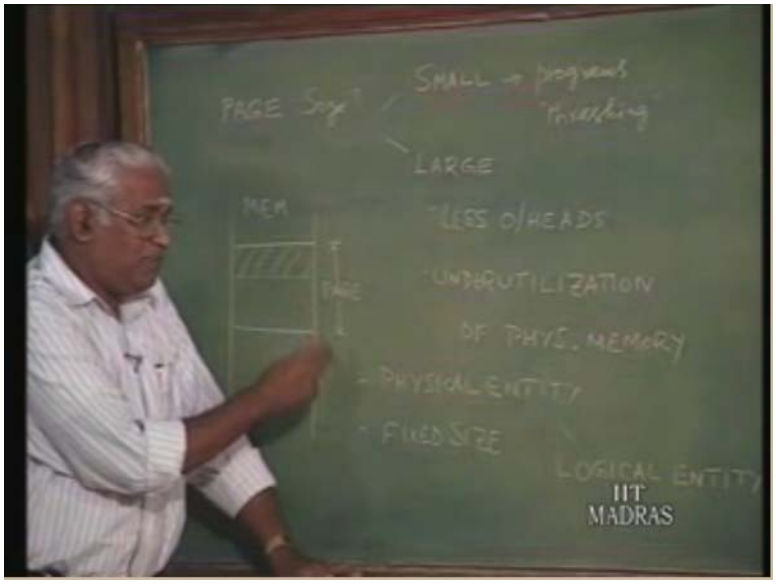
There is some space that may be needed for temporary storage is stack, wherein you always stack the information or the data that you need and remove from that stack; it is actually a temporary storage space, which is always needed. A code, data and stack are from the program point of view. From the data structure point view, for instance, we are talking about logical things. That is, we can say some portion of the memory is the code portion and some portion of the memory is the data portion and some portion of the memory is used for stack. This is a logical entity; we are not talking anything about a fixed thing like we are not talking about physical memory at all in this particular case. Some portion of the memory, whether it is in the memory or whether it is in the disc space is logical, that is, certainly the logical characteristics of the code part of the program will be different from those of the data.

Similarly in stack space, the way they are used is different. This is from the program point of view; in addition, from the data structuring point of view, you can talk about arrays of data, or data, which is put in some array form or maybe in tabular form or tables of data, suppose we talk about data structured in tree form. You can see each one these is a logical entity. Why not divide the total address space? When we say total address space, what is it we have in mind? We have the disc space because that is always the one which the user will talk about; he will not really bother about what is physically existing because that is essentially the computer system architecture that person is concerned with. So we can talk about these things as logical entities and we can say that a code is in this portion of the memory; the data is in this portion of the memory; and so on and so forth. In other words, what we are talking about here is some logical entity called segments. These are called segments of the program. Now we can see the code segment will be different from the data segment in the characteristics, in the individual characteristics. It has nothing to do with the physical part. But the only one thing here is that you do not know about the size of this.

It will vary from program to program whereas we can say very clearly a page physically exists and hence we talk about the size, which is fixed. The segment sizes will keep varying and, in fact, in that lies the solution. We are going away from the fixed size page to a variable size segment or segments. We will come back to the variability of the size. What will be, for instance, the minimum size of a segment? We said it is variable; there must be some minimum and some maximum. We know that the real maximum will be the virtual address space because that is the limit of the CPU's addressability; the user cannot have anything more. There is a minimum; size 1 will be the minimum size of the segment. Obviously that will be decided by the page size because ultimately the logical things that we have here will have to be physically stored in the memory so that the CPU may access.

So there is no point in talking about a segment, which is less in size than the page. We can certainly say the minimum size of segment is nothing but the page size. That is because page is the one thing that we were calling a block that is being swapped in and out.

(Refer Slide Time: 16:20)

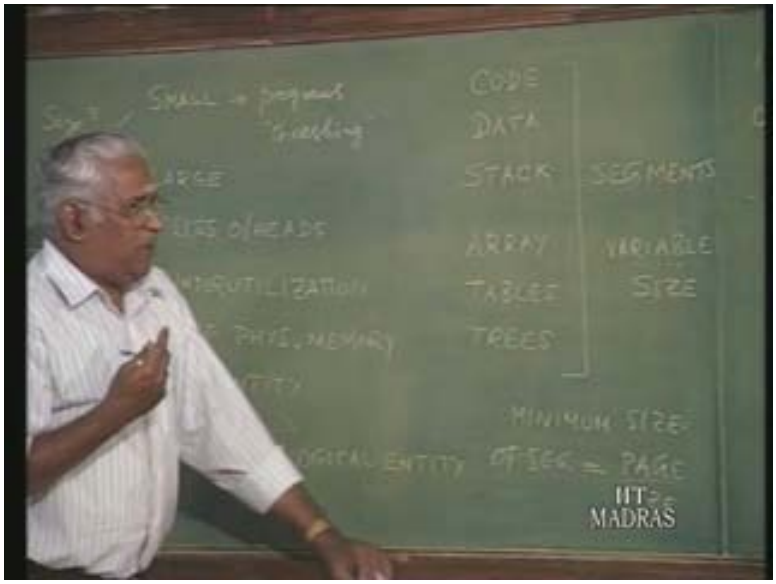


In case a data segment does not need one whole page but still it has got to be within that, we cannot afford that because that is the one which will be swapped in or out and it will be meaningful to look at that thing as one segment. Then what happens when you mix up is suppose in the same page you have both data and code, and then the characteristics of this will vary. So there will be a problem dealing with them at the logic level. Now what about the variability? There must be some criterion based on which you make something variable. It must be useful for that. Now like pages are swapped in and out, this segment also will be swapped in and out. In fact we always talk about something like an incoming segment and outgoing segment. Similarly we could have talked about an incoming page and an outgoing page. That is something which has been swapped in will be your incoming page; a page that is outgoing would be the swapped out page. Similarly in the case of segment also, all these things are from the point of view of increasing the CPU's utilization. So the user will of course feel very comfortable talking about the logical aspects of it but certainly not the physical thing. A segment that is incoming will consist of many pages: minimum will be one page. So for a segment to be brought in and then segment to be taken out, though we may talk about the logical thing, it will have to exist physically in the memory.

Finally we have to look at that; there must be some mechanism for handling this also. Like we are talking about page fault, segment fault will arise when a particular segment is not available. So there must be a mechanism for handling these things. Let us also remember that these contribute to the overheads and so these will have to be kept as low as possible.

So for the variability of the size of the segment we will go by the logical part of it. It all depends on the application; there are situations when you may have very few data, during code processing, in which case the code size will be much bigger than the data size. There are applications the other way; that is, the program is small; there is not much processing to be done, but then the data it has to handle will be huge and in running a specific program, we come across the stack space because essentially it is for storing the temporary variables. It is a temporary storage space; so in some cases as and when the data keeps coming you may continuously process it but in some other cases you may have to stack and keep it for future use.

(Refer Slide Time 22:46 min)



So depending on the applications, the sizes will vary. But given a specific application, we know what will be the size of the code, what will be the size of the data, or what will be the size of the stack – at least by and large we would know. That is why we say the criterion for fixing the variability will be based on the logical aspect of it, that is, a logical entity. Certainly this is different from the physical entity because the page is physically in a memory whereas these segments will be logical segments in the virtual address space. So we talk about segments in virtual address space and pages in the physical address space, but ultimately for CPU to proceed the segments will have to be brought in. When the segments are brought in to the physical memory, they will be brought in chunks of pages and the minimum size will be one page size. When a segment is not available and for parts of a segment you see a segment can consist of many pages, it is not necessary that all the pages of a segment will be available. In the case of a very huge program which is huge in size, certainly the 10000th instruction can wait.

The first 1000 instructions will be taken to start with; the 10000th instruction is going to come much later. So you can see that even though we may talk about a code segment of a very large program, you will see a part of this code segment only need be loaded and they will be in pages.

You will have to keep swapping in and swapping out the pages depending on the need. In fact here we are talking about the page fault occurring and so on. Similarly when a segment fault occurs, it is the same thing; the situation is not different. We talk about incoming page and outgoing page. A page should go out because we have a page which is no more in use and we need to bring in a page which is needed. There is a term for that also, let us notes it. So this thing is generally called demand paging; demand paging essentially means whenever there is a need a page is brought in.

Similarly here too, whenever a segment or in other words part of a segment is needed, it will be brought in or taken out. Just like in the case of page replacement, in the case of segment also we talk about segment, which is no more needed; it is going to be thrown out. So there must be an algorithm by which we replace a segment. Like in the case of page, in a segment also we talk about a segment replacement algorithm, based on what criteria or consideration we bring in a new page and replace, that is, place a new page in place of an old page, which can be swapped out.

Whatever we saw in the case of page holds good here; remember what we are talking about in the case of page. We were talking about an LRU and the FIFO algorithm, that is, first in first out algorithm and least recently used algorithm; that is, a page that is least recently used will be swapped out. As per the other algorithm, FIFO algorithm, the first page that was brought in will be swapped out; it will be removed and that will be replaced by a new page. Similarly, we have this here based on the similar consideration; just the terminologies are different.

(Refer Slide Time 25:46 min)

For instance, you talk about the best fit algorithm and the first fit algorithm – first fit algorithm and best fit algorithm. What is that essentially? Now we are talking about a segment, which will have to be brought into the memory. We have to take a look into the memory arrangement.

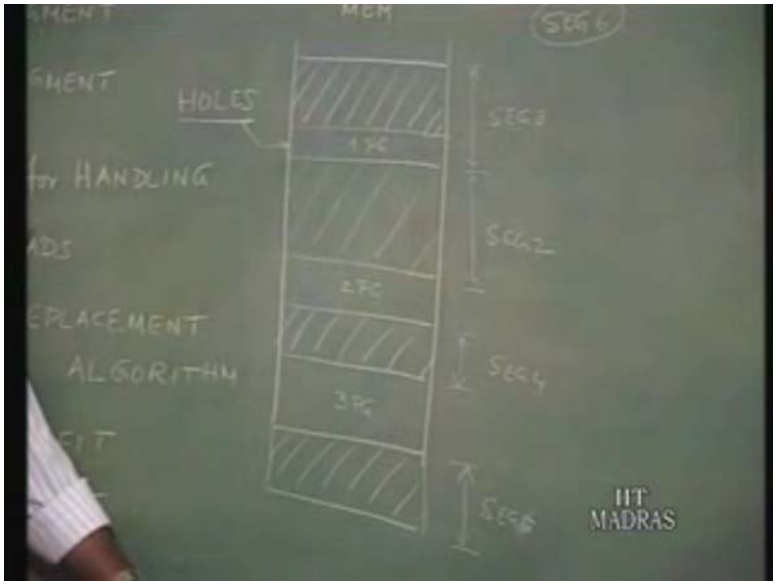
Finally the segments will have to be brought into the memory; so let us just assume that there are many pages. To start with, memory will be empty and into any portion of the memory you can load any segment anywhere that you want, and after sometime when programs are running we have to take a look at that situation. So possibly there was a segment earlier, old segment, which was spanning a few pages, and then let us says I am not bothered what exactly that segment is. And let us say there was some segment that was spanning some four pages and that needs to be replaced by an incoming segment. Now the new incoming segment is only three pages let us say; so what will happen? The new incoming page occupies only three-fourth of the whole space.

That means in-between we have another segment that is being used; they are not of fixed size. Let us say earlier this portion as well as this portion was in use, this for say segment 1 and this is for segment 2, and segment 1 has been swapped out a new segment, let us say segment 3, has been brought in. Now segment 3 is less in size and so what happens is an empty space or unutilized space is created. Such things are called holes, meaning portions of memory space which are not utilized. Similarly, we can see that after program execution there will be holes in the different places.

Let us say these are the things that have been used. Generally segment size will be multiples of pages, minimum is one. That is why I said earlier four pages are used; in subsequent incoming segment we need only three pages. So essentially this is one page and that page is fixed size; we cannot change the definition; only segment size is variable. You can see here that you have got different segments: this is one segment and this is another segment. Why do not we call these segments 4 and segment 5? Segment 1 has been swapped out. Now suppose there is a need for bringing a new segment, let us call it segment 5.

Suppose a new segment, segment 6, is to be brought in. Let us say we have one page hole here; another hole of two pages size; another hole of three page size. Now suppose segment 6 comes and I will just assume that it is two page sizes, so it is an incoming new segment. Now we have some space, that is, we have holes that can be filled. As per the first best fit algorithm, let us say first one will take that best fit algorithm; an incoming segment is of two page size. So let us look at the first hole; it cannot be filled in. That is, it is better always to have the segment continuously; so that it can not be filled in.

(Refer Slide Time: 30:22)



So now let us take a look at the second one – this is two pages; we will go in. It will go into this; that is, two pages fixed into two pages. Suppose we have this particular thing as three pages, and suppose this is four page holes. We can see the difference between the algorithms; otherwise with the previous example I took, you would not be able to shift. This two page can go either into this or into this. Again I think I am running into a problem. Now look at this; it cannot be filled in here. So the next hole can be filled in and that is of course the best fit in the sense it will leave only a hole of one page size and also that happens to be the first fit. Unfortunately I have created a situation where I cannot work. I will just change the size of the segment, only then I can work it out differently.

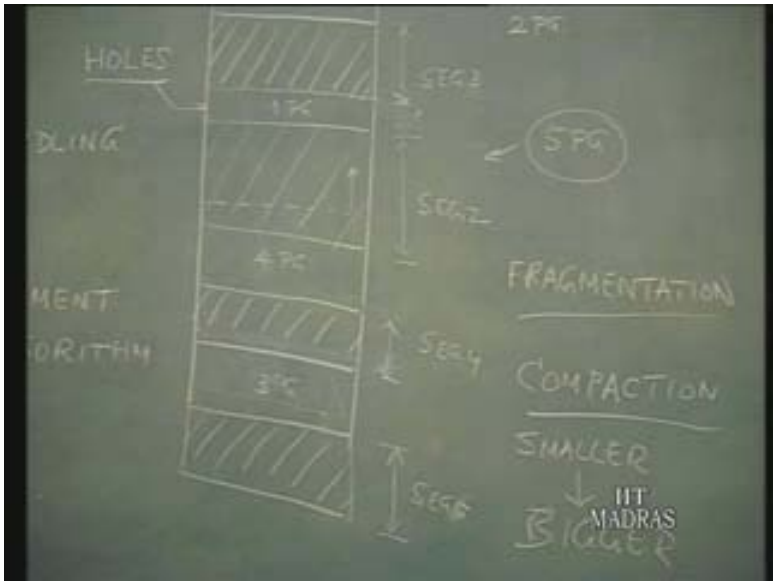
Suppose I have this as four pages and this as three pages; now here is the segment. We can see that as per best fit algorithm, this two page segment cannot go into four pages; it can also go into three pages. As per best fit algorithm, the two page segment will be brought into this and not into this, mainly because the best fit means as far as possible of course if there were a hole with two page, that is the best. But between these two, the better of the two is chosen. As per best fit algorithm, these incoming segments will go into this; as per first fit algorithm it will go into this because whatever hole that is seen first will be filled in.

So these are the two methods by which the holes maybe filled in. Just take a look at it: is best fit always the best arrangement? What happens in the case of best fit? It is not always the best. As far as fitting in is concerned, it is the best but you can see that if the incoming statement comes here, that is as per the best fit algorithm, the best fit algorithm is going to create holes which are small in size. Now it so happened in this particular example that one page is a minimum thing, but then, suppose this were six pages and this were four pages then you would see that best fit algorithm will lead to creation of holes of smaller size and these holes may be distributed in the entire physical memory space.

So what may happen is we will be having one page here, a hole, a four page hole here, you may be having another one page hole here. Now suppose this will be one page and for this suppose I have a five page segment, which must be brought in, now there is a hole of one page and there is a hole of four page as physically five pages of memory is available but the segment cannot be brought in mainly because I do not have contiguous memory location. In other words what happens is that by filling in the physical memory, as per whatever be the algorithm that you may be following, you are actually fragmenting the memory and so even though the memory space is available the memory space is available in fragments in different places.

So fragmentation is one problem; now you can see that best fit algorithm will create more and smaller number of holes and it will lead to more fragmentation problem actually. Because first fit algorithm does not really know which holes are available, it will fill in and we do not know what the remaining portion of it is. So the fragmentation problem has to be solved: one way is of course through overheads. Whenever a hole is created, for instance, a hole was created here earlier.

(Refer Slide Time 38:48 min)

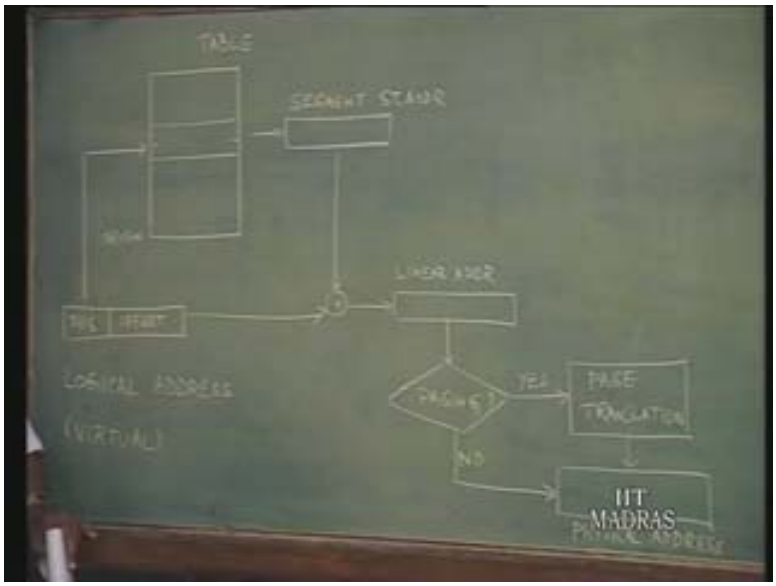


The whole thing was used and subsequently a new incoming segment created this hole. So what can be done is whenever a hole is created immediately the adjacent segments can be moved up by these, that is, the whole segment will be moved up. You can create a hole with the adjacent hole and thereby you combine smaller holes into bigger holes. It is an overhead problem.

So that particular thing is called compaction, that is, you compact the holes thereby. All the smaller holes will be brought together and then you create bigger hole. A bigger hole is always better to have so that if one page and four page hole would have been together, a five page hole would have been available and a new coming segment would have been easily accommodated.

But then, every time whenever a segment fault arises, this compaction must be done so as to avoid the fragmentation but then as we said this is in fact is a clear increase in the overheads. Now always what happens is you try to solve a problem. We are trying to solve the problem associated with under utilization of physical memory and so from page which is essentially a physical entity we moved on to a logical entity and create the segments. Once you know that segments are there very much like in the case of page we also need a mechanism for handling the segment because we talk about incoming and outgoing segments.

(Refer Slide Time 42:19 min)



It is very much like incoming and outgoing pages and the mechanism for handling the segments will automatically be based on some replacement algorithm and now we are finding that the physical memory can be fragmented, filled with holes and there may be a problem when holes are distributed all over the place and we will have enough physical resource physical memory available but not in contiguous form, so that an incoming segment cannot be accommodated.

That would mean compacting them and making smaller holes into bigger ones; thereby we can accommodate an incoming segment which again leads to overheads. So now, what happens? With segmentation and paging we have come here that is first address will include a segment address; from that we have to look into how many pages are there and where they can be loaded. After you load the specific page then you proceed with the rest of it, whatever you had seen earlier. This address, that is, the address translation, starts with first resolving the segment and then the paging and then the rest of whatever we know that is going to the cache and so on and so forth. So let us take a look at the address translation involved in the whole process because now we have introduced segmentation. We now have to take a look at paging and segmentation, which is what we will do maybe with a specific example. In the case of paging we saw how to translate from virtual address to physical address – virtual address is also called a logical address.

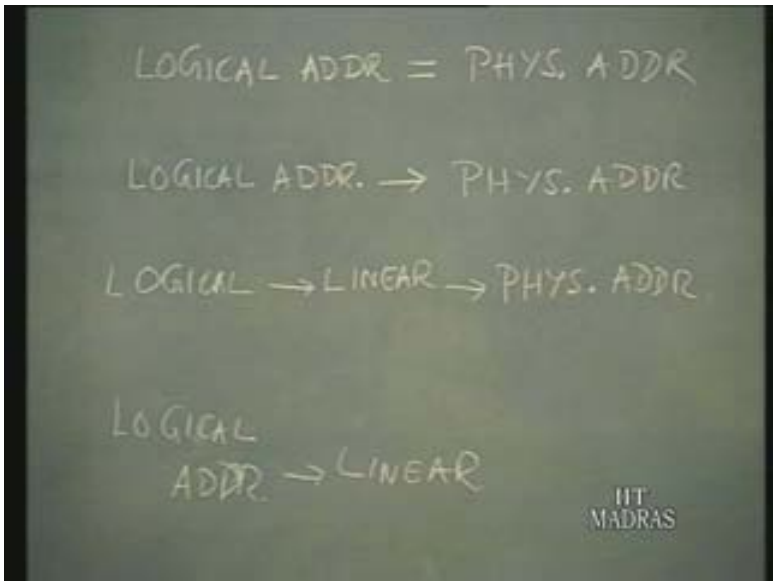
Now we have introduced segmentation in addition to paging so there is one other level of translation. How do you start from a logical address and go on to translate ultimately the physical address? We will take a look at this. The mechanism is similar to what we have seen earlier; so nothing much is new. Given a virtual address, the CPU produces this logical address or the virtual address because this is the disc address this is what is there in the disc address space. That is, the total address space which CPU can generate will consist of essentially something which points to a segment because now segment has been included and then a specific location within the segment, which is the offset. Just like we had page table, now here also we need a table and especially we need one with lot more features mainly because the page was of fixed size but now segments are of variable sizes. In addition to other things, the segment size also must be indicated, is it not? So there will be a table; generally this table is called a segment descriptor table; of course some people may call it differently I just take segment table or something a segment descriptor table.

This table essentially will have information about all the segments and so the portion of the logical address, which generally is called a segment selector and the offset – selector and offset these are two things – the segment selector will indicate within the table where exactly information about this particular segment is, and the segment descriptor table itself, like page table, will be in the memory. So there will be another register which points to the origin of the table like we had earlier in the case of page descriptor table. We had page table or page descriptor table; similarly this is something which points to the origin. In that particular table there will be a specific entry that is pointed to by one portion of the CPU generated address. That will tell where exactly the segments starts: a part of that information in that table entry will give information on where exactly that particular segment starts in the memory.

(Refer Slide Time 47:22 min)

So with reference to that and the offset coming from here, there may be a need to generate an address called a linear address. We will see what exactly is there in the linear address; after you get the linear address, depending on whether paging has been enabled or not, you may or may not have paging – that possibility is there. You may or may not have segmentation and you may or may not have paging.

(Refer Slide Time 48:50 min)



In case paging is enabled, then take this address and then do the translation the details of which you have seen earlier, and then arrive at the physical address. Otherwise, suppose there is no paging at all, directly this it is the physical address. So the result we have here is that we have to consider different memory models. Some in which segmentation will be enabled and segmentation will be there and paging will be enabled or not. So let us see the different models; for instance, suppose there is no segmentation and there is no paging also. As you can see, this is called a flat model, in which case, the logical address that is generated is the physical address; it is the same as physical address.

There is no translation, nothing need be done. On the other hand, suppose you have only paging, you are on segmentation, this one will be called a linear address. This is what we had seen earlier; there was only paging translation there was no segment, which we had considered earlier, where we said from the logical address it is translated to physical address. So from logical address a translation was done, making use of entries in the page table. A translation is done to physical address. Then, suppose there is segmentation as well as paging, this in fact is the model in which segmentation is there with paging. Now here, this is precisely what we have indicated, that is, from the logical address we have to get the linear address making use of segment descriptor table, and then from the linear address we have to get the physical address making use of the page translation mechanism. It is very much like that with reference to page table and so on and so forth. So here, from logical to linear address, there will be a translation. You make use of the table as indicated, from which you get the physical address.

The last one is that where you have segmentation but without paging, there is no paging involved. We have segmentation but no paging but this is somewhat curious situation – paging is more meaningful than segmentation. But suppose you have, then what happens is the logical address is translated as before to linear address, which itself is used as the physical address.

So you can just see how the overheads have been increasing, that is, in the address translation. We started with this earlier, then we said paging. In case there is paging from logical address there is a virtual address. In the virtual address we are getting the physical address. Now you see with both, we get some address, which again must be translated into physical address. But the mechanisms are similar; there is no difference. So if you have page there must be a page table, which gives information about the page and if you have segment there must be a table which gives information about the segment.

Recall that while talking about the page table in the entries in that, we are talking about lot of bits you know like protection bits and then access right bits – there are many things we were talking about, and then protection from between system and user segment; then access rights – read or write or read and write or write read only, etc. In case of page whenever you have segmentation it is meaningful to have all this information in the segment because segment is a logical entity. Once you define data segment, you say this data is read only you do not bother anymore about the page.

So whatever we have been talking about earlier in case of page, in case there is segmentation all that information will be included in the page descriptor table and so you can see that an entry in the page descriptor table will not only give a segment starting address, it shows where exactly the segment start, but also it includes all this information and a lot more too. So these tables are essentially useful for having this extra information and then, for instance, there may be one bit in this, which says this segment is for system and this segment is for user. In this way, you can specify what you call as privilege levels, so that the system is always set at a higher privilege level than the user and so on and so forth.