Computer Organization
Part – II
Memory

Prof. S. Raman
Department of Computer Science & Engineering
Indian Institute of Technology, Madras

Lecture – 19
Virtual Memory

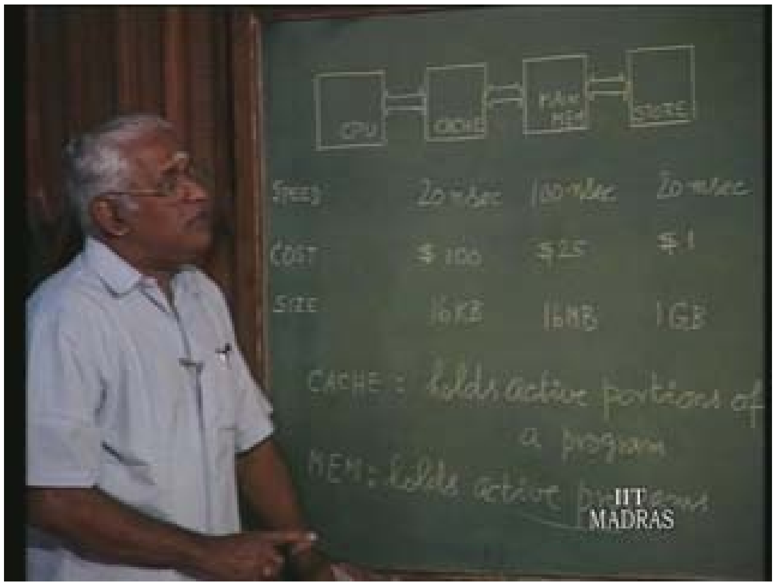
In the previous lectures, we have been considering the CPU–cache memory interaction. So as you can see here, cache is between CPU and memory. And we have not really completed our discussion on cache. We will continue, but then, at this point in time I will digress a little and take up the next issue also because these issues are similar and the techniques also will be similar. Why? Just like cache is between CPU and memory, and access is an intermediate thing, you can see that memory is intermediate between cache and store. And so, we can also consider memory as a cache between store and cache; of course, when we say memory as a cache, it is cache with an inverted comma. Just like this is in between, now we find this memory is in between this cache and store. So for the moment, we will also take a look at the issues in this. We have been considering this particular one, and it so happens that historically the cache development and then the other development related to memory storage had different routes actually. So what happens is that some of the terminologies are functionally the same, but then as far as the terms are concerned, they are slightly different – we will see that too.

Before that, I would like to go back a little and then take a look at the three factors. Do you remember I was always talking about the memory hierarchy, the speed, cost and size? So you must have this also in mind when we are discussing these issues. Now I will just give some typical figures. So it is only really typical and you should not think that this is a figure. We may say it is the order – you are familiar with that. If you just take cache for a particular unit, that is, you must relate all these things to some specific unit; for instance, if we take the speed about the cache as 20 nanoseconds, that is, basically, it is memory access time for a word, maybe a byte or whatever. The similar figure, in the case of the memory, will be of the order of say 100 nanoseconds.

Though we said 100, cache will be 200 or sometimes even less, whereas in the cache of storage, which is essentially magnetic, it takes a long time, say 10 to 20 milliseconds. So I will just put it as 20 milliseconds – now you can see here this is nanoseconds; this is also nanoseconds. Sometimes this can go to microseconds; that is very old technology. But then you can see this one is vividly different; it is in milliseconds and the cost of it again, as I said, for a specific unit it maybe a chip or whatever size you may consider. So let us say in terms of dollars, if you say we have to spend 100 dollars for a particular unit size, the main memory would just cost about 25 dollars – you see it is at least four times cheaper. Now in the case of magnetic store, it may be as low as 1 dollar. Now you can see why we had to introduce a hierarchy. If you want to go for a very cheap one, you may; but then you pay in terms of the speed – it is going to be very slow. Then typically, if the cache size is about 16 KB, that is, 16 kilobytes, assuming it can of course be different.

Now you can talk about an 88 to 16 megabytes system – a really good system will have about 8 to 16 megabytes and the storage size will be above 1 gigabyte; of course it can also be quite a few megabytes, hundreds of megabytes; we may say gigabytes. Now let us have this in mind: we will be referring to some of these figures because while discussing the issues here, we will see there are also issues that are related but then these factors will come into play; we will talk about it.

(Refer Slide Time: 12:15 min)


CPU
CACHE
MAIN MEM
STORE
SPEED 20 nsec 100 nsec 20 msec
COST $ 100 $25 $1
SIZE 16KB 16MB 1GB
CACHE : holds active portions of a program
MEM: holds active portions
IIT MADRAS

So as I already said, we can look at memory as a cache in-between store. What is the difference between the cache and the memory? We were calling it DRAM; generally we may call it main memory. Normally, it is a DRAM subsystem. Now what does the cache hold and what does the main memory hold? We knew the cache consists of blocks of data; a block can be of variable size – it may be a single word or a multi-word – we saw both the cases. What does cache hold? Cache always holds something, which CPU wants as frequently as possible and necessary and as quickly as possible – these are the two things. So while a program is executed, we find that the probability of accessing the adjacent instructions is going to be very high.

So we would find most of them in the cache or the data; it depends on the organization of the data. We can always say that cache holds the active portions of a program in the sense that the cache holds those portions of the program, which will be used or acted upon by the CPU. So remember portions of a program; that is, a program is executed, which consists of instructions and data. The adjacent data may be location, remember the principle of locality is something which is locally available. So there is always very high chance that all the locally positioned instructions on data are going to be required by the CPU as quickly as possible, and of course, to see that the CPU utilization,   processor utilization, is the highest, we tend to put whatever is going to be required as frequently as possible for access by CPU as quickly as possible. That is for these units or chips, which will respond very fast.

So the blocks actually contain portions of the program; they are going to be used by the CPU. Now what will be the memory hold? What will be the memory holding? If you see, a cache holds active portions of a program; we can say that the main memory holds active programs – in what sense? Assuming the storage has multiple programs of different users, we can say that the main memory holds active programs – what do you mean by that? They can be possibly active programs. Of course, when we talk about active programs what we have in mind is the multi-programs or the so-called multi-programming situation. So historically the issues here came up when multi-programming was needed.

Now we say that because today we are all familiar with desktop PCs and possibly you would be the only user and you may be running only one program. But historically it was not so. We had essentially main frame computers in computer centers. Even today, you may still have them in big establishments and it is possible that there are many users. So there are multiple programs; that is, the multi-programming feature is still a necessity. It is possible that the CPU, which is a very special kind of CPU, is running very fast and possibly something like what you are thinking of is a highly parallel programming system, in which case we do not talk about a single processor. When you invest so heavily in systems then you would like to see that it is used as well as possible, that is, the utilization factor must go up, in which case you accommodate multiple users. So the situation is still there, nevertheless this is true only in the case of multi-programs. So you can see that historically there was a necessity for allowing multiple programs and also again in those days this memory was costly.

In other words, all these things, like memory, would be looked at as a scarce resource mainly because it was costly – I am still talking about historical condition. So you can refer to them as scarce resource and we can see that multiples programs will have to be accommodated and it is possible that when we talk about multiple programs, let us just assume multiple users. So it is possible that when one user program is taken up, the program cannot be completely executed. There may be some problem, in which case, that program execution will have to be stopped or what we call aborted, and then another program must be taken. This is one situation. Another thing is that there can be some programs, which may consist of many routines and one routine will call another routine. For instance, we may be having a lot of CPU bound process of program and that may lead to or may call for I by O routine, and I by O routines need not really be taken care of by the main processor. So there may be a separate channel processor for that, in which case temporarily this main processor is relieved from that particular activity. While waiting for that I/O process to be completed, the CPU can take another program and proceed – this is one situation.

Another thing is that it is possible that the application is such that the program is really huge in size and we may be able to store the entire program in the storage unit, but we are not able to bring the entire portion of the program into the main memory. So this is the situation that is possible; it all depends on the application, in which case only some portion of the program itself may be brought, very much like cache but there is a different situation. That is, we are talking about a huge, a large program, consisting of multiple routines or generally referred to as multiple tasks.

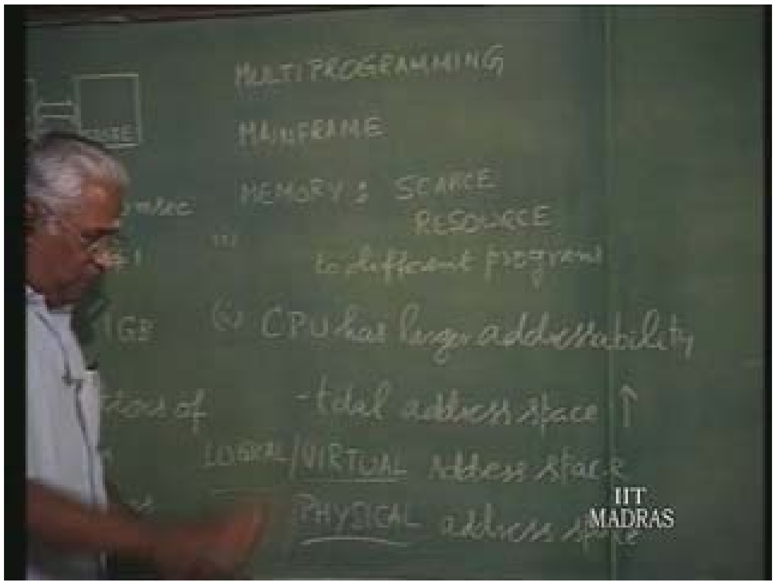
So then what we would have is we may not have the entire thing running, we may be having only a part of the program in terms of so many tasks, in which case, in case the memory is costly, then the only solution is to have as much of storage as possible and whenever required bring the portion of the memory to the main memory, the size of which you keep low because it is costly. You can see here that whereas here we were talking about the blocks of data coming from the main memory, here we will also talk about chunks of programs coming in here. You can see that historically multi-programming was the issue mainly because of the main frames and so we had a problem about allocating the scarce resources to different programs. And this, of course, was specifically for main frame situations. Today the situation is somewhat different. It is like this: the CPU has become very powerful; the CPU word size has increased from something like say 8 bit, 16 bit, 32 bit, 64 bit to 128 bit. The address size also keeps increasing; 1 bit extra in address is going to double the size of the memory. If with 16-bit memory, memory address, we have 2^{16}, that is, 64 KB, if it is just 1 bit extra, 17 is going to be 128 KB – that is double.

So one bit in the address size is going to increase a lot. When we talk about the CPU's addressability increasing and one small bit in this is going to double the memory size, what happens is that it is possible that we do have the situation where CPU's addressability is very high. But then we are not able to have that much of memory, physical memory, mainly because if you want to have you can just add 1 bit to this, it is not going to cost more, but then 1 bit means double the size of the memory – total address space is going to be double. We are not able to have that, in which case again this storage being cheaper, it may not be a problem having that. So what we may have is the CPU's addressability has increased and the user may be told 20 bits – 2^{20} is 1 GB; that is available, let us say. But then, it is possible that you do not have 1 GB memory, but you have only 16 MB.

Then again, we have the same situation – this of course is currently possible; that is, the CPU's addressability has gone up and so the program size can go; that is, from the user point of view. But from the system implementation point of view, we cannot populate the total address space, in which case, the user may be told the CPU has larger addressability and so the address space has increased from programming point of view, but physically speaking, the memory size is less – the user need not be told. The user may just be given the impression that now he can have a very large program and then it is up to the designer to see that the large program is brought in chunks and then executed here.

So, in other words, what we are doing is, the CPU has larger addressability. That comes about because of the number of address bits, addressability; that is, the total address space from the user point of view has, in fact, increased. This is from the user point of view. But physically, the memory is not going to have that much; in other words the user is told, you have a large address space, and he is given an impression that the address space is high. But then, physically speaking, the address space is low. Who is bothered about this physical address space? It is the one who was to run the program, not the one who has to write the program. So these are the two things: the user writes the program and he will be told you have a large address space. Let us say 1 GB or more is available; even 0.5 GB is available, but then, when it comes to the physical, it is less.

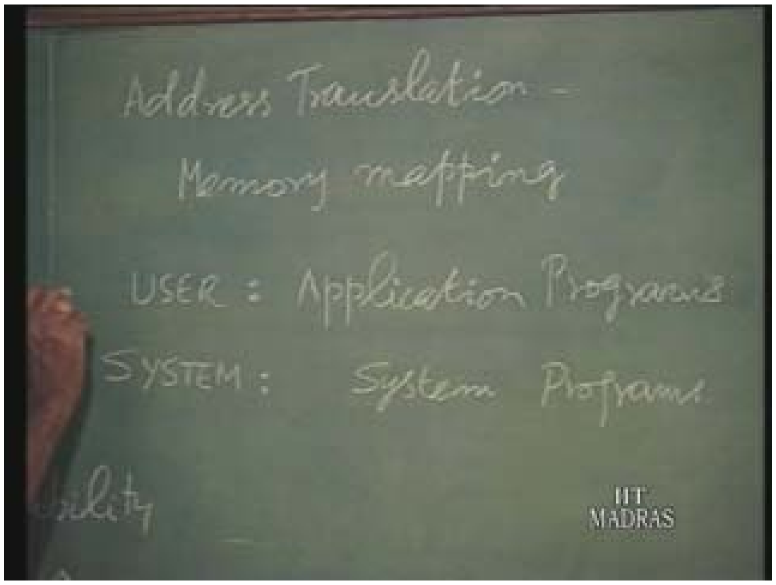
(Refer Slide Time: 24:03 min)



So this is the situation and that has brought in two kinds of address space: that is the total address space being high is indicated to the user and this particular thing is called the virtual address space. But in the system, what we have is what is physically available. So the physical address space is restricted; the physical address space is less. But the user is not really concerned with this; the user is only concerned with this particular one. So this virtual address space, that is, from the user point of view, he has large address space, but it is not physically existing. From programming point of view, this exists – this particular virtual address space is also called logical address space, mainly because from the user or programming point of view, this space is available in the storage. But in reality, for the CPU to access and get it, it is not available because that is going to be restricted by the main memory size, what is possible given the cost and other considerations.

Now what will happen is that in case this is made physically available to the CPU, this CPU, which can really run at 20 nanoseconds, will have to run at 20 milliseconds. The CPU utilization comes down because of this particular thing – we are talking about hierarchy also. Incidentally, we are talking about the hierarchy mainly because of this; now let us proceed with this, that is, though historically multi-programming had brought in certain issues, the issues are still valid though from different point of view.

(Refer Slide Time: 26:06 min)



Address Translation –

Memory mapping

USER : Application Programs

SYSTEM : System Programs

ility
IIT
MADRAS

This means the user writes his program assuming large address space and while running the program this virtual address must be translated into physical address space. So we talk about address translation from the virtual to the physical address – this is also called memory mapping because you have to map the virtual address space to the physical address space. Of course, memory mapping is a term used in some other context also; we will talk about it later. What about these multiple programs we are talking about here? A situation exists now mainly because software complexity has increased.

Today we talk about application programs from the user point of view because user is essentially concerned with writing a program for a specific application, and the software complexity has increased so much. From the system point of view, we talk about system programs or system software – that is, we have system software and we have the application software. So we have both user programs and system programs. Essentially the user is running this program and this is going to be run under the care and supervision of this program. So we still have the two things; that is, you have both user programs: this is one type and then we have the system program – this is another type. Still, the situation of multi-programming exists, but it is from a different point of view because under the care of these, this is going to be an error. So at each system, generally, when we talk about an operating system, this is one thing and the other is the user or the application.

These two types of programs are there, and there should not be any confusion between one and the other. In other words, there will be some portion of the memory which will be used by the system and some portion of the memory used by the user or the application. The respective programs must be stored in separate spaces so that there will not be any confusion. Otherwise what will happen is that it is something like doctor and patient – now what happens if the patients become the doctor? Of course, if the doctor becomes the patient, you can cure him, whereas if the patient becomes doctor, what will happen? So that confusion should not be there; the same thing holds good here too.

Essentially we still have to make sure that in the case of multiple programs, the respective types of programs are compiled and put in separate spaces. So there is the need to compile, that is, generally the software is written at a very high level. It must be translated or compiled to lower level programs, lower level codes, which can be run by the hardware to compile each program; now here we are talking about the system program and the user program or system software and user software. We have to compile each program in its own address space; that is, they must be isolated. Otherwise, there may be some problem. That is what we are talking about when we say in its own address space.

What we are not talking about is the protection that must be offered between the two types of programs. One should not corrupt the other, certainly the user program should not corrupt the system programs. So you can see here that there are two things: one is that we need virtual to physical translation; the other thing is that we need to protect the appropriate segments of software that is user or system appropriate. So it is that although these things would have started historically, these issues are still very much relevant even in the present situation. Now we will be having the logical or virtual addresses here and we will be having the physical addresses here.

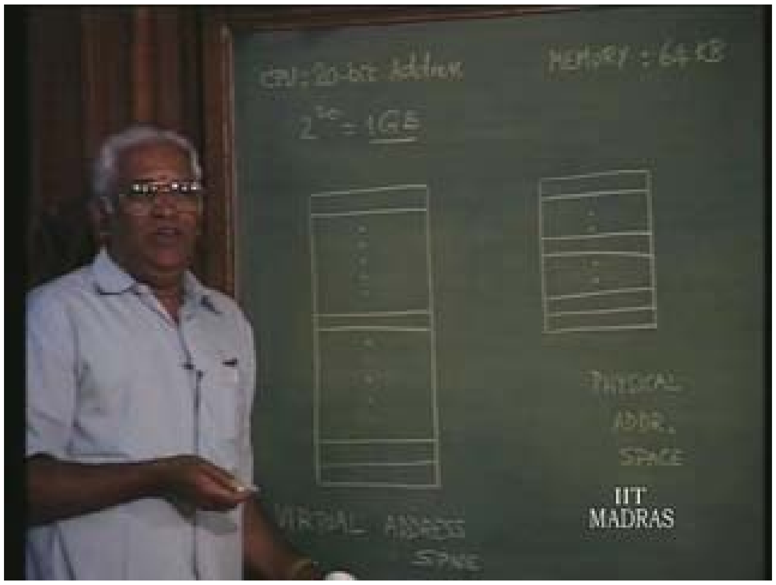
It is possible that we have different programs at least user program and system program minimum one user there can be multiple users when we talk about multi-programs. A core of system program must always be available and the appropriate portions of the user programs may be brought in, depending on the storage memory size consideration. At any time the cache will have the part of the program, which CPU can access as frequently and as quickly as possible so that the utilization of the CPU maybe high. So this holds the active program's main memory; all the programs are available but unnecessary in storage. The main memory holds the active programs; cache holds the active portions of a given program; and CPU usually holds a portion of one instruction of a program, nothing more.

Specifically in one state of the CPU, it is a simple register transfer level activity. Are you able to see the relation between these? The CPU goes through instruction after instruction; that instruction cycle ultimately consists of states and cache a portion of a program memory. The system must be having those programs, which are to be run, and storage. The CPU looks for something in the cache; when it is available, we say the CPU has hit. If it is not available, we talk about a miss. Now whatever is not available there, it has to fetch from the main memory. It fetches a block, because cache is defined in terms of blocks of data, which the CPU may access. So whatever be the size of the block, it may have been between main memory and storage. The storage is going to have a very large address space so there will be lot of locations; we think in terms of some location, address location, whereas the main memory is what is going to accommodate whatever is physically possible. This, in fact, is the subset of this; this is less in size, this is larger in size; this is a super set.

Similarly, if whatever the CPU needs is not available in main memory, it has to be fetched from here. Here again we can talk about cache hit and miss; that is, something the CPU requests is available or not available. The problem is this: look at the timing. If something is not available in cache at a penalty of five times the time duration given, you can get it. If something is not available in the main memory, which the CPU could have accessed in 100 nanoseconds, now it is going to access in 20 milliseconds – this is 10^{-9}; this is 10^{-3}. So, it is a factor of 10^6 or a million.

If something is not available here, getting it from storage is going to cost hundreds of thousands of clocks as far as the CPU is concerned, whereas here it may be tens or possibly a few hundred clocks. So the penalty in case something is not available here is going to be very heavy, and so the situation must be avoided. That is why we say the memory must hold active programs; that is, whatever is needed for executing a program must be made available. The only situation when it cannot be is when the program is very huge in size. Now talking about this cache and the main memory, let us also look at the terms and how they have come about. You had seen the similarity, but then the terminologies are different again for historical reasons. In the case of cache we are talking about a block of data – maybe I will put it like this; the cache and then the main memory. In the case of cache we are talking about a block of data. Similar to block of data, in the case of memory, we have what is known as page. Here also, it is similar to block of data called the page. In the case of cache, if it is not available, there is a miss; in the case of memory, we call it a fault. In this, we will be talking about a cache miss; here we will be talking about the page fault; otherwise they are similar.

(Refer Slide Time: 39:16 min)



So really we have not looked into some details of cache because in this memory to storage also the issues are similar. After we study something about this memory to storage interaction, that is, this part of memory to storage, we will come back and discuss and continue the discussion on the cache. What is this cache and memory and other things? We really have to work out and go in to the details of how this virtual to physical address space this translation or relationship holds good. So we have to draw a few more diagrams. We are talking about the virtual address space essentially created by the CPU’s addressability and the physical address space; essentially what is physically implemented is the memory part. Now we may say that this virtual address space is a storage space. Essentially, it is also disk address space; you can have anything that you want in the disk; the only thing is that what we can have in the memory gets restricted. So we will start with a few figures assuming we have a system with memory of size 64 KB, and so that is the total size of the physical address space.

Suppose we have the CPU of 20-bit address, which generates a 20-bit address – what does it mean? That is, 2^{20} fetch is 1 GB. This in fact is the total address space, which we may call virtual address space. When the CPU supports 20-bit address, the user is going to assume that the address is of 20 bits. So he is given the impression that 1 GB space is available; so the program is going to consist of 20-bit address – that is going to be the case, whereas physically the system designer has included only memory of 64 KB. Now whatever address is generated by the program will have to be translated. Again we have to just recall what happened historically. In those days the job of translating this virtual to physical address was left to the users; in other words the total program will be sliced into some chunks, which can be loaded at any one time into the memory. Those chunks of program were actually called overlays, memory overlays. In other words, I hope this conveys the impression that the user looks into the program and if the program is of a very large size, then he splits into portions of the program, which will be exclusive or which can be run independently. So these things were left to the user. In other words, the user was burdened with the additional responsibility of looking into the programs, splitting and organizing, and all those things. These days, the operating system will look into those things. In other words, there must be the appropriate hardware or software support, which will talk about the translation and protection. So historically it was left to them.

What is the meaning of software? Software essentially is one which softens the whole system to the user. The user was to create overlays and then load them appropriately, not only creating overlays which are mutually exclusive portions of program but also loading them into appropriate time – when exactly to remove and when exactly to load were left to the user. Now the system software takes care of it and remembers this has absolutely nothing to do with the application part. The user is essentially concerned with the application program, not this kind of thing the system has got to do. We said memory is 64 KB; that would mean essentially we are going to have a 16-bit physical address because 64 KB is 2^{16}. So we are talking about a 16-bit physical address and a 20-bit virtual address. We were talking about pages, is it not? What is a page? We said in the case of cache we are talking about the block, a portion, which is always available. Similarly, in the case of this physical memory, we talk about a page consisting of 1 KB or 2 KB, whatever be the size.

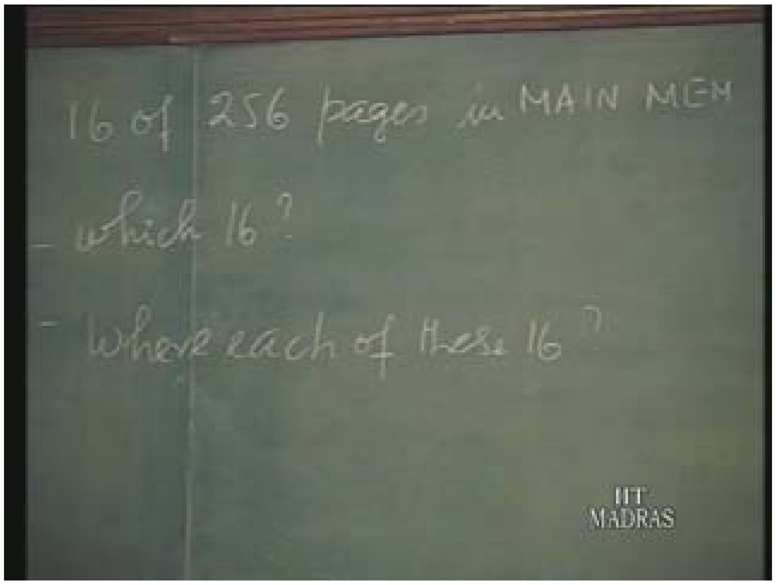
So we will just assume that the memory in this is the total address space. Suppose this is split; we just arbitrarily take that suppose we have 16 pages. Basically what we mean is that we have 64 KB; 16 pages each of 4 KB is our page size. This is in disk and this is in main memory; the transfer between the main memory and storage takes place in such a way that the whole page of data, information in general, is transferred between these. Just like we are talking about a block here, we are talking about a page. So we just assume arbitrarily that the page size is 4 KB; meaning any time 4 KB, whether for that particular program 4 KB is required or not and is enough or not, that is not the situation. Maybe 4 KB is not enough, the program size is 16 KB, in which case, we talk about a program that consists of four pages. Maybe the program size itself is only 1 KB, in which case, 4 KB is still transferred. So what is transferred between the storage, that is the disk space, and the memory is the DRAM space that is of size 4 KB.

Always remember that this transfer is going to take a lot of time compared with DRAM or main memory to cache. So from that point of view, it is always better to have the page size as large as possible, because everything can be transferred in one go, but of course, there are other things. Now we talk about 4 KB pages, which means here we essentially have 16 pages because we have 4 KB and 64 KB. So let us say if this is a page, normally we start with page number 0, but we will say 1 because we can easily follow that way. This is the first page; this is the second page and so on; then we have 16 pages here. It is of course meaningful; this is also a page. It is also meaningful to split the disk space into the same size of page, because the same chunk is going to be transferred.

What about here? Here we can work it out – we can talk about each one; for instance, this is page number 1 and then, appropriately, the page. That is, basically, 1 GB divided by 4 KB will give the number of pages: we will work it out. When we take the address, automatically this will come. I would refer to few more terms before I proceed because in literature, different terms are used. For instance in some books, you may find this being referred to as virtual page; that is, this has so many virtual pages and whatever you have here will be referred to as physical pages – this is one thing, this is one term. In some other literature, a page is always only a virtual page and what you have in the memory will be called a page frame. I hope you can understand why it is. That is, essentially it says here you have so many, say, 16 frames into which this data or these pages will be fitted in.

So I think we will follow this term; that is, we will keep calling this a physical page and keep calling this a virtual page. Because there are some variations, you may find page and page frame – this is also another term or virtual page and physical page – we will follow this physical page. Now let us see one more thing. Maybe I would specify that this particular thing will have 256 pages overall; we will work it out. This means essentially you can take it for granted that there will be only 16 physical pages and we have 256 virtual pages for this initial assumption. This means at any time we can have physically only 16 out of the total 256 pages. It is not a very good figure actually. What happens is that it is possible that quite often the page that is needed is not at all available there because between 16 and 256, you can see it is not very good. We started with some figure; we will continue. At any time, only 16 of the 256 pages will be available in the physical memory; now there must be information on which 16. This is number 1; only 16 of 256 pages are available in the physical memory. I will call that main memory.

(Refer Slide Time: 50:37 min)


16 of 256 pages in MAIN MEM

- which 16?

- Where each of these 16?

Now there must be information of which 16 – this information must be available somewhere, and another thing is, is there any necessity that page 1 must go only into page 1? No, it is not meaningful. To make it universal, what we would say is that any virtual space can go in to any physical page – that way the efficiency can go up. If you bring any restriction, there will be a problem. Of course there may be problem even otherwise, we will not worry about that. So if you say that any page can go into any page, we need to have that information also. So number 1 is which 16 of 256 pages, then, among each of these 16 pages, this information also must be available somewhere. Now you can see that when we are trying to solve the issues of timing and then giving the user this kind of a virtual addressability, then we have to worry about maintaining some extra accounting information. In other words, these are all overheads; they are not really contributing to the actual processing. We will continue this particular thing in the next lecture; I mean starting with a 20-bit address and then going in for a 16-bit address and then remembering the need for maintaining this information.