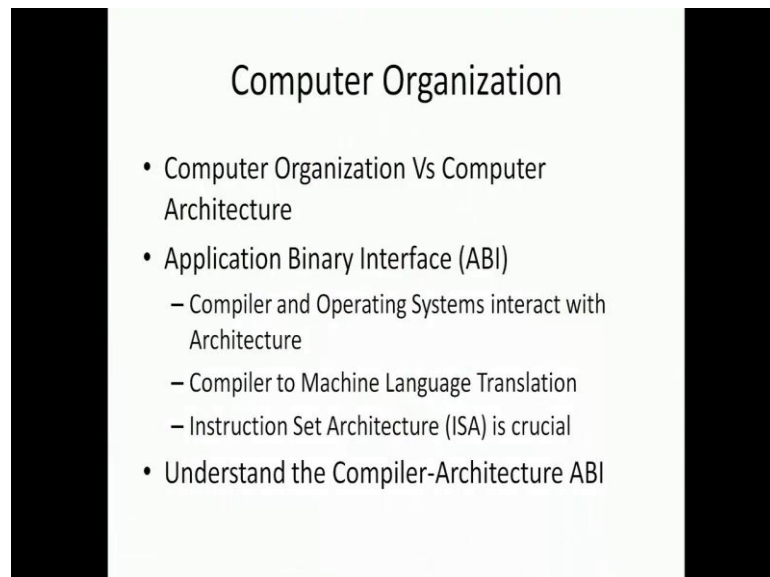


Information Security - II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 08
Architectural Aid to Secure Systems Engineering
Session - 7: Structured Computer Organization

So, we will start with what we call as Structured Computer Organization.

(Refer Slide Time: 00:28)



Computer Organization

- Computer Organization Vs Computer Architecture
- Application Binary Interface (ABI)
 - Compiler and Operating Systems interact with Architecture
 - Compiler to Machine Language Translation
 - Instruction Set Architecture (ISA) is crucial
- Understand the Compiler-Architecture ABI

Now, there is always a question between, what is you normally in your curriculum we have something called computer organization and then you have something called computer architecture, what is the difference between computer organization and computer architecture? That is a very interesting question, very important question. In computer organization, is already there in the slide. What you study is what we call as an application to binary interface, right? So, how does the compiler and operating system interact with the architecture? That interaction, study of that interaction is the subject matter of computer organization. We also talk about, how the compiler to machine

language translation happens? And, to understand this translation proper, your instruction set architecture is very crucial.

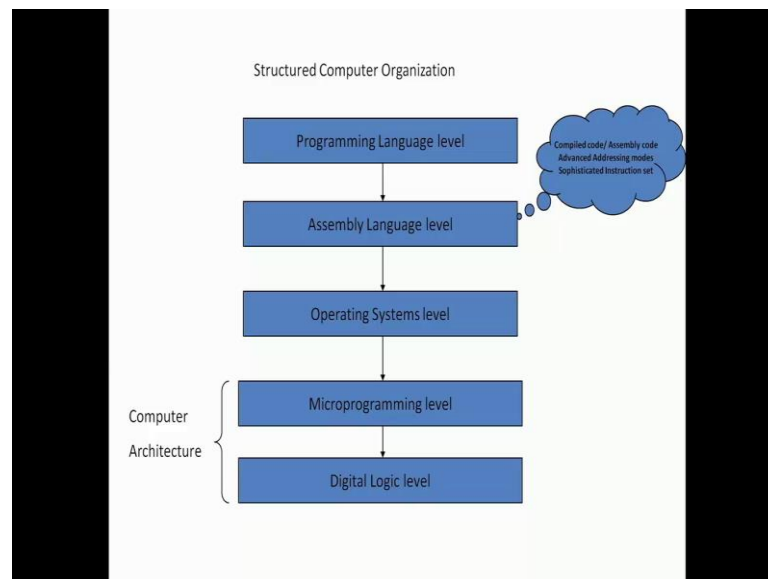
So, what we study in the computer organization course is basically the interaction between computer organization, but the architecture and the operating system and compiler. And, one very important thing that we study there is instruction in architecture. Then, we also study about operating system needs this feature how will I provide it? Compiler needs these feature how will a provide it? For example, compiler needs pointers; How did I provide it in the operating in the architecture? I had indirect addressing modes, correct?

The operating system wants to implement Round-robin scheduling. What is the support I give from the architecture? I have timer interrupts, you got this. So, for everything I put, is not for stupid, just for the fun of it I keep building architecture. For everything that I put in the architecture what I need, I have a reason for putting out in the architects. I putting, of including something in the architecture. There will be a need, who will give that need? Either the operating system will give that need or it the compiler will give you the need,. There can be many things, there can be that needs to be you know understood from this perspective. So, that is what we call as an application to binary interface. Now, after understanding the interface, now the architecture has made certain promises that I will do this, I will do this, I will do this, I will do that; now how are those things actually implemented in practice, that is the subject matter of computer architecture, got it.

So, somebody ask you the question between computer organization and computer architecture, you should be in a position to give a very categorical clear answer. So, what we are going to do in computer organization, this particular advance computer organization which talks about security aids; aids for security from the. We are going to understand the compiler architecture interface, we are going to understand the operating system architecture interface, we are going to understand the application software architecture interface and depending on that the architecture changes. So, the architecture that I use in a network appliance is different from the architecture I use in a desktop, is different from an architecture that I use in a mobile phone, is different from an

architecture I use in a server, is difference from an architecture I use in an embedded system. What dictates those architecture? The way I am going to use that architecture, and this study of computer organization will tell you what sort of architecture you need to put for a given type of scenario, you getting this?

(Refer Slide Time: 04:22)



Now, as we have seen this code quite frequently, there is a Programming Language level, there is an Assembly Language level, there is an Operating System level, there is a Micro programming level and there is a Digital Logic level. So, there are 5 different levels in structured in computer organization. Now, the last 2 levels are called the computer architecture. How do I build the those levels? What is there inside these levels, the micro architecture and the digital logic level? That forms the subject matter of computer architecture. Now, there will be a question, so the compilers actually ask for features from the architecture to induce most sophistication in the programming language. I told you already a feature. If I want indirect, if I want pointer I am using indirect address, if I want function calls I needs stack call, like that compilers will ask for several features and what you will be addressing specifically in this course is more from a security point of view; what is the compiler going to ask and how does the architecture provide it.

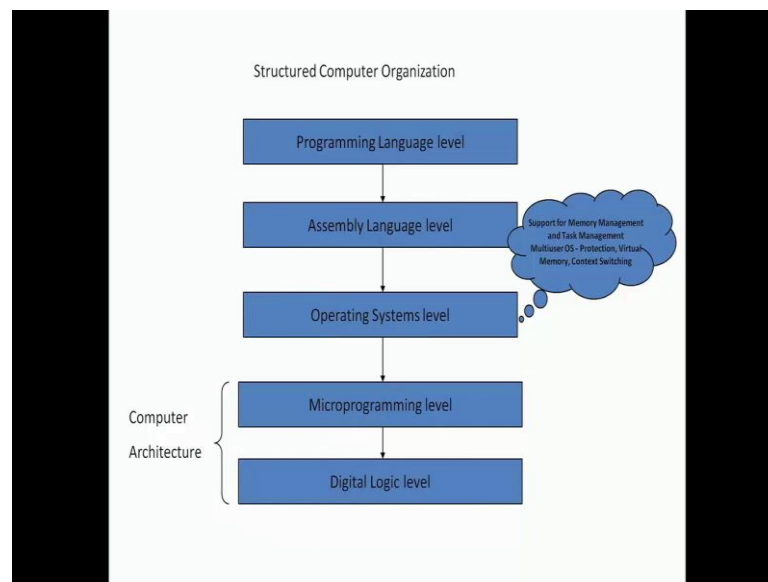
Similarly, the compiled codes as an assembly code, will need some sophistication. For example, advanced addressing modes; for example, I have a normal variable, I can address it using some normal addressing. Before, suppose if I have a structure, all of you know a structure in C, now I want to go and access a structure, I need a little bit complicated addressing mode; like I will have one register which has a base, base register, then I will have some offset from that, base plus displacement. Why I need this base plus displacement? The base will tell you where the structure is stored, the entire structure will be stored within that structure there will be, what? Other elements, and I want to access thus elements; it will be some bytes away from that thing. So, that extra bytes that I need to add, I will store it in the displacement. So, I will have this base plus displacement. So, there are many, many interesting arc way of expressing the memory that I need to go and access.

So, those sort of sophistication's will come here. And other thing is, suppose I want to do end of say 32 bits, I can do bit by bits separately; one end operation, next end operation, next end operation, but then I have memory elements which can store 32 bits together. Like for example, a register can store 32 bits. Can I do a bit wise end of all the register bits? Then it will make my life very very easy. I can do the bit wise end of all the 32 bits and that will make my life very easy. So, what will happen in the architecture, I will introduce one more instruction, we just end off 2 registers. Now, where will I use this? Suppose, I am simulating a circuit, a digital circuit which has un gates. There are 32 different un gates and I want to evaluate the output of that. If I want evaluate 32 different un gates, I may do 32 different end operations instead of that I can do it one shot; where the inputs of each of these registers I will put it on the corresponding bits in the 2 registers, the inputs of each of these gates I will put it in 2 input un gate, thus 2 inputs I will put in corresponding bits in the registers and I will just do the end of these register.

So, 32 un gates, I can do it in one shot that gives me parallelism. So, that is what resulted in an instruction called end; that may be one of the reason, correct? So, when I start compiling the code, when I start looking at the functionality and start compiling the code, what happens? I will get certain needs from the code like I have to do certain operations, in order to speed up those operations I will start introducing new instructions and

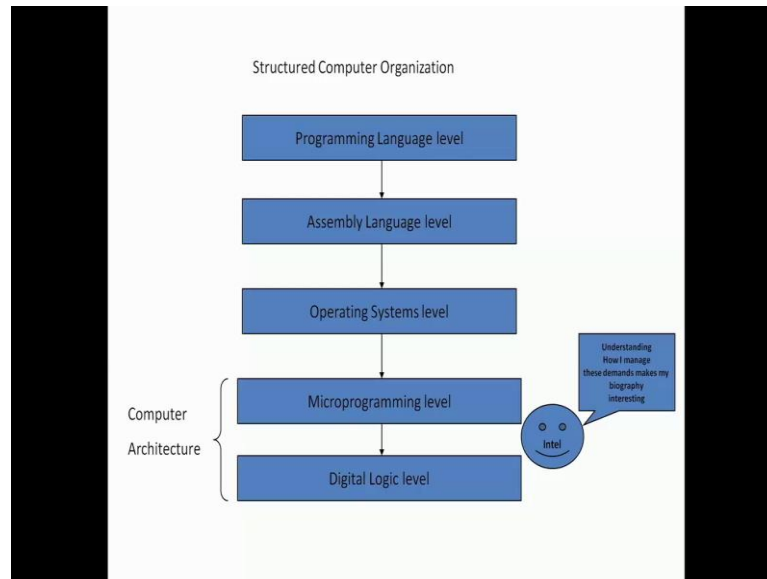
infrastructure. That is how my instructions set are. Today, it tells instruction sets has close to some 700 instructions or something like that; 700 plus instructions. What was the rationale behind adding these instructions? These are all the rations. So, as and when my different types of applications come in, I start using different types of thing. For old engineering programs, they use only floating point they may not need this multiple end, but when you start using computers for doing simulation of circuits you start using this some multiple end, correct. So, this is how things evolved.

(Refer Slide Time: 09:34)



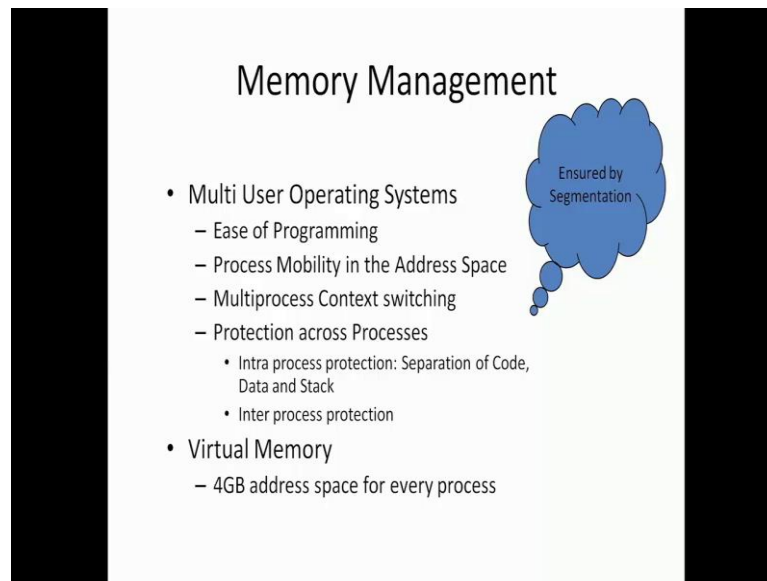
And then, operating system will also ask lot of things. Like, it will ask for support for memory management, a glass support for task management, specifically multi-user operating system were more than one fellow logs into the same system, it should ask for protection it should (Refer Time: 09:49) virtual memory, context switching so many things.

(Refer Slide Time: 09:57)



So, what will Intel do or Intel Arm MD? It will understand, understanding how the architecture basically manages these demands, forms the subject matter of computer organization. So, what does computer organization on this course will teach you, more from a security perspective? It will teach you how do I go and manage these demands, correct. That is made by the programming languages, made by the compiler, made by the operating system, correct? And, also we add a security thing to this.

(Refer Slide Time: 10:39)



The slide is titled "Memory Management" and is set against a light green background with black vertical bars on the left and right. It contains a bulleted list of topics. A blue thought bubble with the text "Ensured by Segmentation" is positioned to the right of the list, with a line connecting it to the "Protection across Processes" item.

Memory Management

- Multi User Operating Systems
 - Ease of Programming
 - Process Mobility in the Address Space
 - Multiprocess Context switching
 - Protection across Processes
 - Intra process protection: Separation of Code, Data and Stack
 - Inter process protection
- Virtual Memory
 - 4GB address space for every process

So, let us now start basically with memory management. Now, where this memory management essentially come up in a big sway? Memory management actually becomes quite big or becomes complex when we start using what we call as Multi User Operating Systems. What is a Multi User Operating System? More than one user can simultaneously log in to the system, and start using it. And the operating system will give equal amount of share of compute and storage resources to these uses. So, from a multi user operating system point of view, what is it that we need from the programmer? What we did that we need from the architecture? We need to have an environment, where it will become easy for me to program. What do you mean by easy for you to program? Easy for you to compile, what it means? We will go in the subsequent slides, I will explain you what we mean by ease of programming. But, I have to program with ease. I will define ease very shortly. You would have, you at some stage you will not even appreciate what is ease because you do not know what is happening beyond it.

Now, I have to tell you what is happening beyond it and then express you, if that does not exists, it would have become a hell for you. It is still now a heaven for you because that exist and I have to tell you what is that and then you will realize what is Ease, correct? So, at this stage, let us say, I need to have an environment where I need ease of

programming, then there should be ensure process mobility in the address phase. What I mean by process mobility? It should not be that, I should for executing this process. I should load it only in this memory location. I should be in a position to load it anywhere in the memory and it should execute, because it is a multi user operating system; at any point of times several users can be using. And, if I have a program which I say if only if you load it at you know location 1,000 it will start executing otherwise, it will not execute, it becomes location dependent. It is not possible in a multi user operating environment. Because, when I want to execute which ever memory is available I should be in a position to go and load myself into that memory and start executing, I cannot say I need this particular memory, I will wait for it then you have to eternally wait.

So, I should say any memory you load me, I will go and execute it. So, my compilation, the way I am built and at the way I am going to be executed should be independent of where I am loaded in the memory. So, that is what I mean by process mobility in the address phase. I could move the process from any location to any other location and still it should start working as if nothing has happened. And then, multi process context switching is also very, very important. What I mean by multi process context switching? I move from one process to another process. So, how do I move from one process to another process? And, when I move I should come back to the original process and start from where it stopped so, these types of issues are there.

And then, I should also there are many processes that are in execution together, I should now make protection between these processes. One process should not; a process will have stack code and data. This fellows stack should, one process A should not look into the data or this stack of process being, there should be some amount of protection and isolation across processes. So, there should be both, intra process protections, where the code cannot go and over write into the stack or over write into it is own itself. So, there is intra process protection which essentially talks about, separations of the code data and stacks, so that one will not corrupt another. And, inter process protection between 2 different process is one should not touch another and how it like. So, all these things are very important from a memory management prospective.

The next one is Virtual Memory. Now, I am giving you 32 bit architecture. I say you have a 32 bit architecture, what does it mean 32 bit architecture, by the term 32 bit. I can address 2^{32} locations of memory. I can do 32 bit arithmetic, there are many manifestation, but one of the interesting manifestation is I can do one of the interesting definition is, if I am having a k bit architecture I could address 2^k bit of memory. So, I say you have an architecture which can address 4GB of memory. I should allow you to write programs which can use that 4 GB, correct? I cannot say though it is 32 bit architecture you can only use these much of amount, but in a multi user operating system there will be several fellows who are loaded in to the memory.

Now, I say you can use 4GB means the entire memory is not available to you. So, somehow I will tell you can use 4GB, you make that 4GB program and give it to me I will execute it somehow. As far as your concern that 4GB is available to you; as for as your concern the 4GB is available to you in full, you understand this. So, that means, I am giving you a virtual memory, I am giving you a virtual memory which you can use. That really, that memory will not exist, it will be virtually available. You think that 4GB is available you will run the program, somehow I will take your program and with very less memory much lesser than 4GB I will try and execute and give you back; this is a notion of virtual memory. How those concepts are implemented? what is the architecture support needed? Understanding that support, how it is implemented is architecture; understanding that support is organization. So, memory management is very crucial. In memory management we will look at these aspects of this course.

So, the first part of ease of programming, process mobility, multi process context switching, inter intra process protection all these things can be ensured by a concept called segmentation; memory segmentation. And, we will teach memory segmentation in full. But, please note that even some of the recent operating systems do not use the memory segmentation as it is done in inter, they do not use the complete facility of memory segmentation and that is one of the reason why they are still vulnerable.

(Refer Slide Time: 17:31)

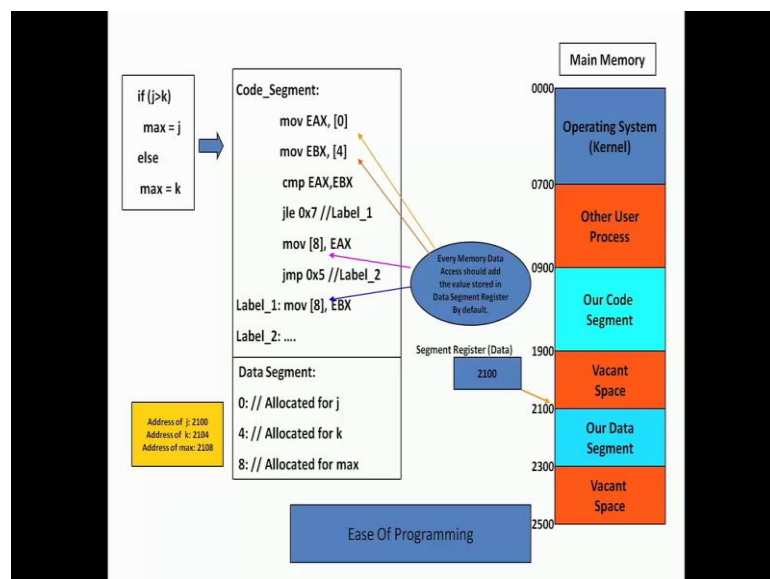
Memory Management

- Multi User Operating Systems
 - Ease of Programming
 - Process Mobility in the Address Space
 - Multiprocess Context switching
 - Protection across Processes
 - Intra process protection: Separation of Code and Data and Stack
 - Inter process protection
- Virtual Memory
 - 4GB address space for every process

Ensured by
Paging

And the virtual memory is ensured by a concept called paging, it is also prevalent in the both the x 8 6 and arm architectures.

(Refer Slide Time: 17:42)



Now, we will go and talk about what is this Ease of Programming. So, this is how the, there is a program let us take that c code; if j is greater than k then max is equal to j else max is equal to k. So, what does this code do? It compares 2 numbers and the maximum of it, it stores in a variable called max. Now, I write this, I translate it into a code. So, what you see on the middle is a machine language version of the c code, by this I am also introducing you to the x 8 6 assembly instructions. Now, there are 2 parts to this code; one is the actual instruction which is called the code segment and there is another aspect to this code which stores the data which is called the data segment.

The instruction in the code segment uses the data in the data segment to execute the program, right. Now, that j b is stored in location 0, k b is stored in location 4 and max b allocated the location number 8; 0 4 8 are the addresses and I need to make this assumption. If I do not know this 0 4 8, please note that I cannot go and get the code on top of it. So, let us for the time being assume that j is stored in 0, k is stored in 4 and max is stored in 8. Now, what do I do? I move 0 to EAX, means what? The j value is stored in EAX. EAX is a register, it is a 32 bit register; j is an integer and it is also a 32 bit. So, the value of j is stored in the register named EAX. So, move is a command to move data from one location to another location. EAX is a register to which the data comes in, so first is the destination and second is the source.

If I put a square bracket in Intel essentially, it means I am accessing memory. So, I am accessing memory location 0 where the value of j is stored, 32 bit value of j is stored and that I am moving it to EAX; register EAX, which is also a 32 bit register, got it. Now, why should k b stored in 4 and not in 1? The integer is actually 4 bytes, you got it. The integer is actually 4 bytes in size and that is why j is allocated 0, I allocate k at 4 because the 4 bytes are stored in location 0 1 2 3 and so, k will start b store at 4. So, in EBX k is loaded move EBX comma 4, the second instruction. So, the value of k is now in EBX, I compare EAX and EBX. That means, when I compare there are something called a flag register in the system which gets updated because of this compare instruction.

So, there are 2 variables here, the left hand variable and the right hand variable; EAX and EBX; that is j and k. If the left hand variable is less than right hand variable, there will be

a less than flag that is set. If the left hand variable is equal to the right hand variable, there will be a 0 flag that is set, why zero? Comparison is basically done by subtraction. And if the left hand variable is greater than right hand variable then the greater than flag would be set. So, there will be a less than flag, a 0 flag and a greater than flag. The left hand flag will be; the left hand variable is less than the right hand variable, equal to, if both are equal; greater than, if the left hand that is EAX is greater than. So, if j is less than k then, after doing this comparison then I can take something called a branch instruction. Branch is I go from one location to another and the branch can be, there are 2 types of branch; one is called the conditional branch, another is called the non conditional branch.

A conditional branch is one where, that branch is executed if some conditions are true. For example, JLE stands for jump if less than or equal to. So, if JLE; jump if less than or equal to 0x7; jump 7 bytes away from me, right? You got this? That 7 bytes would be that label-1.

So, if EAX is less than EBX or if j is less than k, what I do? This j is less than or equal to k, then this branch will be taken the JLE will be taken and 0x7; 0 x stands for hexadecimal, 7 is 7. It will jump to this label-1. Why 7 there? Because, if I compile this move 8 x and jump 0x5 the 2 instruction in between the total length of that instruction is 7 bytes, so I jumps 7 bytes away from me. So, essentially I come to label-1, you are getting this? Are you following what I am saying? So, why that 7, because that intermittent 2 instructions move 8 EAX and jump 0x5 that is following JLE is 7 bytes. So, now, what happens? Move EBX comma 8, EBX is what? k, right? EBX stores k, right. So, moved a value of k to 8, what is 8? max. You are getting this. So, if j is less than or equal to k, may k the maximum. This JLE is 0x7 will not execute when EAX is greater than EBX that is when j is greater than k; then if it is not, then what you do? You come to here move 8 EAX, what is 8, memory location 8 is max; so your EAX, what is EAX? j is stored in EAX. So, j will go to max and then jump 0x5. why 0x5? Because the next instruction is 5 bits, whatever I store in label-1 is 5 bytes in length. So, I will say jump 0x5 it comes to label-2, you followed.

So, it just jumps and goes and it starts executing the other code. So, what has happened here, in the location number 8 which is max; the maximum of j and k which are stored in 0 and 4 is computed and that maximum value is stored in max and that is what this code intends to do, you all got this. So, what are the things that have happened here? Now, let us take, this is the main memory now, this is how the main memory will look when you start executing a program. There will be some part of the main memory typically, the lower part of the main memory that is close to address 0, where the operating system and its kernel will be loaded; let us say 700 bytes of that is needed and it is loaded. After that it will all be user process, etcetera. You got this; there will be user process and other. What is a user process? Process like this;

For every user process there is a code segment, there will be a data segment and that will be loaded somewhere in the memory. In this case when this program is to be executed, it is loaded into the memory; there is something called a loader which will load it in to the memory. So, where is it loaded? So, there are 2 segments here; one is the code segment and there is a data segment. The code segment acts on the data segment to do this. So, where is your code segment loaded? Your code segment is loaded between 900 and 1900 in the right hand side as you see in the memory. Your data segment is loaded between 2100 to 2300, right? So, these 2 are loaded here, and then what happens? So, what is address of j now? 2100, because the data segment is loaded from 2100; the address of j will be 2100, the address of k will be 2104, the address of max will be 2108.

Now, there is a register called a segment register. That segment register will store the base address of that segment. In this case, it is 2100. Now, every memory data access that is happening will use the address; now if I just execute `move EAX, 0`, what is stored in 0? Some operating system kernel is stored in 0. You understood. Some operating system data is stored in 0. `move EBX, 4` some, what is there in 4 in the main memory? So, what should I do now? Instead of `move EAX, 0`, it should be `move EAX, 2100`, `move EBX, 2104`, `move 2108, EAX`, you got this. If I just execute this code segment as it is, then what will happen? I will be reading some operating system data here; I will not be reading j and k as envisaged. You got this. So, how does the architecture support this? Are you getting the problem what I am

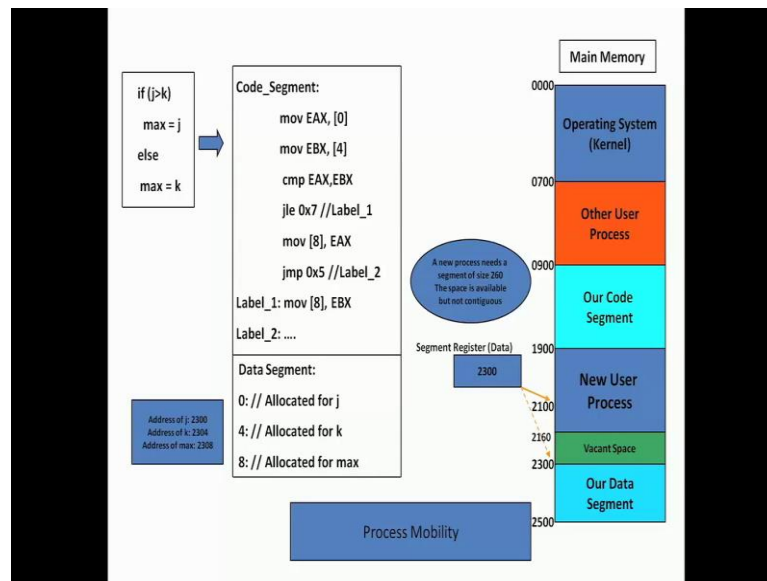
saying? So, how does the architecture support this? The architecture says, whenever you are accessing any data you add whatever is there in the segment register to it. So, what happens now? Move EAX comma 0; 0 plus 2100 (Refer Time: 28:57). So, move EAX comma 2100; j is loaded there. Move EBX comma 2104, k is loaded there. Move 2108 comma EAX, correctly EAX goes to max, correct? You are understanding?

So, what does the architecture do here? Architecture will take the address generated by the compiler and add it to the segment register and then that is what it will be using for memory access. So, that is the support of architecture to segmentation, as 2 memory segmentation. You are getting this? Are you understanding this? Without that this intervention, what will happen? This particular code that I have written on the top will access something in the operating memory, operating system memory and it will not be accessing something in the actual memory, you are getting this? Yes or no?

Now, let us go to the next steps. So now, what will happen? Why cannot I put 2100 itself move EAX comma 2100, move EBX comma 2104? I cannot do it because at the time of compilation I do not know where the code segment will be loaded, I do not know where the data segment will be loaded. So, I need to do what? I need to do position independent compilation. I need to compile assuming that my code will start at 0; my data also will start with 0 and that how I compile. And after compilation, the segmentation should ensure that I load wherever I load my program, the correct data should be accessed, correct? And, it ensures it like this.

So, this is the support of architecture for implementing segmentation, do you understand this? So, ease of programming essentially from the compiler point of view, it assumes that it is 0 and leaves everything else to the architecture to manage. So, as you see at all these places, the addresses 0, 4, 8 will translate to 2100, 2104 in 2108 by the procedure that we have stated here.

(Refer Slide Time: 31:39)



Now, let us talk about what is; this is ease of programming, right. Otherwise, the compiler will have, cannot do anything about and specifically in a multi user operating system. It has to be independent of the location. Now, let us go and see process mobility. This is the original memory organization, address of j is 2100, k is 2104 and max is 2108. Now, a new process wants to come and it needs 260 bytes of segment. Now, is that 260 bytes large segment here? The vacant space, look at the vacant space there is 200 bytes in the first vacant space on the top and 200 bytes in the next vacant space. So, there is 400 bytes available and that fellow is asking only for 260, but I cannot allocate it; why I cannot allocate it? Because, I do not have 216 contiguous locations as you see in the slides, OK. I do not have 260 in the contiguous location as we see in the slides.

Though I have 400 bytes of vacant memory, I cannot allocate it because you need 260 contiguous junk of memory. So, what I do now? I moved my data segment down. So, this was how my data segment looked. Please note that, the blue data segment towards at 2100, I moved it down, I just copied the content from 2100 to 2300 to 2301 to 2500; I just copied it. Once I copied, that is space become vacant and collectively, now how much bytes I have as vacant? 400 bytes, now I can load my 260 byte data. The moment I moved this data segment down, then what happens? Where will the new j b is stored? So,

the new the j will be stored at 2300, the k will be stored at 2304 and max will be stored at 2308. Now, when I moved this data segment to 2300, please note that I have also changed the value of that register also to 2300. Now, when the program is executing, now what will be the new value of j? It will be 2300, k will be 2304, max will be 2308. So, I did not change my program, automatically the values of j, k and max got adjusted, immediately. And, this is possible because of segmentation. Because, every time I do a memory access I add the base value in the segment register to that. Now, your segment registers value has changed to 2300 and it takes care, immediately takes care of all the accesses that we are talking of it, got it.

So, this is what I mean by process mobility. By process mobility, I moved a data segment and I did not change any program, it still executed without any change. And, why I need process mobility? I needed because, what we needed because, I need to do occasionally or sometimes even frequently I need to do something called garbage collection. What happened here? The 2 vacant space or bit of garbage, right, they do not have any thing, but this garbage is spread in 2 different space in your address space, each of 200 bytes and 200 bytes. If they were together then I can load some other program, now any program which is larger than 200 bytes cannot be loaded here.

Now, I have a program which is 260 bytes. So, now, I do what you call as a garbage collection and the moment I do that garbage collection please note, that I get enough space for loading the new program and now, the program is actually loaded and now may new vacant space becomes 2160 to 2300 as you see here; because new processor of 260 bytes is already loaded here. Now, automatically, your addresses of the variables j, k and max also changes. So, in this session, so far we have seen about segmentation and paging and what are the roles and responsibilities of segmentation and paging and we also talked about some advantages of segmentation in terms of process mobility and you know ease of programming. So, any doubts in this?

Hardware does not get addresses with the actual addresses, hardware actually gets what you call as logical address, we will deal when we are doing with segmentation and before it goes to memory, it is getting added with this segment register to actually find the actual

registers. So, the hardware is in different forms, there is a register, there is a segment register, there is a normal register and there is memory. So, what goes as an address to the memory is not just whatever is there in the register value. It is a register plus the segment registers value base, which goes to the memory for accessing the data. For example, in the session 6 that last part that we saw sorry, session 7 on the last part we saw. So, 0 is an address, correct? Which is available with your instruction, but it is not the actual address; 2100 would be the address or 2104 would be the address, which goes to the memory. 0 does not go to the memory at all, 0 is added with the segment base and that is only given to the memory.

That is completely the responsibility of the architecture. It cannot be the responsibility of the operating system because it does not know; it does not have direct access to the segment registers. So, it is completely the responsibility of the architecture to take your 0 and add it with 2100. So, what is role and responsibility of you see here? The role and responsibility of the compiler is to compile this code, assuming 0 and 4. The responsibility of the operating system is to give location, give enough space for the program to store its code segment and data segment. So, the operating system is responsible for loading that code segment between 900 to 1900 and the data segment between 2100-2300; that is a responsibility of the operating system. And then, once the operating system has loaded it is the responsibility of the hardware, is the responsibility of the hardware to see that every memory access the segment register base also gets added; it is a responsibility of the hardware. So, there is an operating system responsibility, there is a compiler responsibility and there is a hardware responsibility, as you see in this slide.