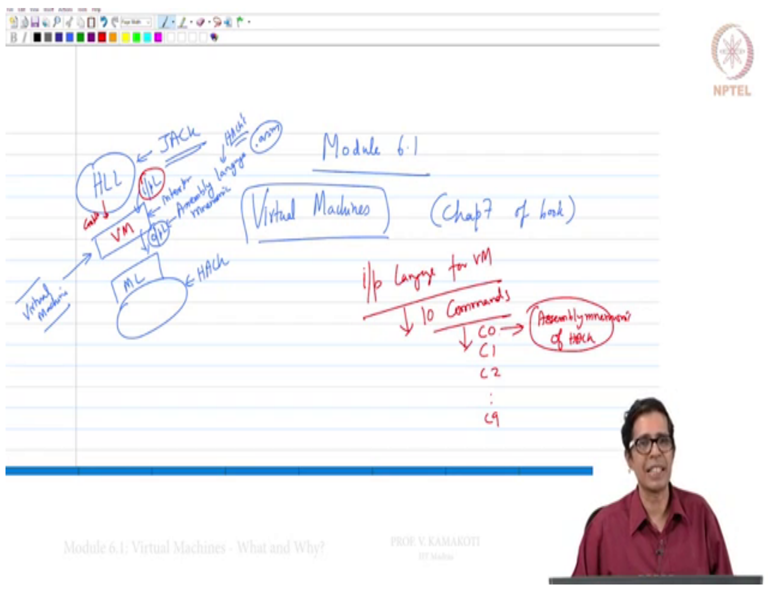
Foundations To Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 6.1
Virtual Machines-What and Why?

(Refer Slide Time: 00:17)



Welcome to module 6.1 and in this module will talk about virtual machines, virtual machine is chapter 7 of book, I hope you have purchased that book, there's any copy please do purchase that book its worthwhile and that will be a good reference not only you will enjoy that right. So now whatever I am going to cover is part of you know the chapter 7 of that book. Now virtual machine, why virtual machines? So I have an high level language in our case it is going to be called Jack which is slightly (())(00:53) form of Java.

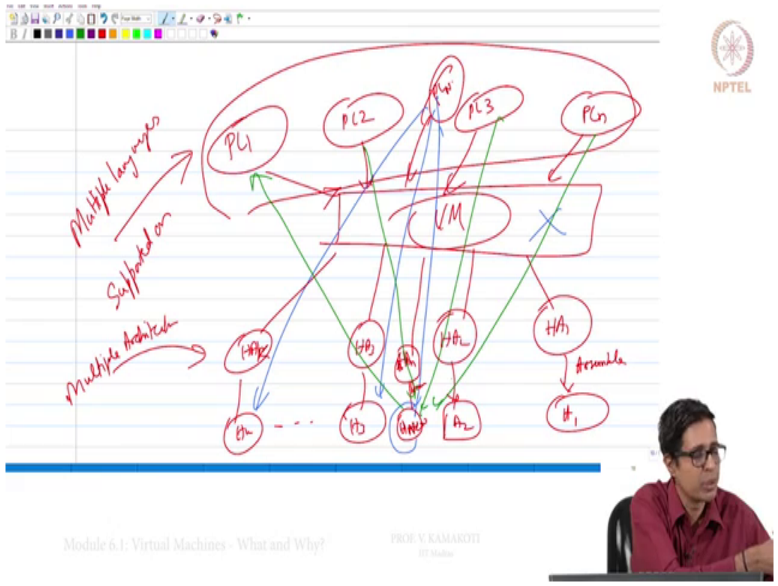
Now already I have my Hack which is my hardware and that had a machine language for which I have assembler also now, convey ping from HLL directly to this machine language is a nightmare right, it is going to be a nightmare so that is one of the reason we put one more level which is a virtual machine meaning I am not going to make a hardware for this machine but I am going to define one more level and this machine will have an input language and it will have an interpreter which will convey it into an output language and what is that output language?

This will be the assembly language or whatever that dot asm files that you see. Assembly language mnemonic. Once we get this assembly language mnemonic which assembly language hacks assembly language? Once you get this assembly language mnemonic I can use my assembler to make it into binary and execute right. So (there) now we understand more of this so Jack is a language that as a compiler that compiler will compile it into the input language of youe virtual machine. Now there is an interpreter which will take the input language of their virtual machine and convert it into an assembly language mnemonics and then the assembler will take it and make it into a machine binary and execute.

This I have been explain it again I am explaining it for sake of clarity. So what we going to do in this particular module? We define what is the input language for the VM? And now you define so this input language say will have something like 10 instructions or 10 now we can't say I don't want to say instructions then commands and each of this command each one of this command I should now translate it into a assembly language mnemonic of hack. So for each of this command so let me say C1, C2 or C0, C1, C2 to C9 I will get a routine which will be in assembly of assembly mnemonic of Hack.

So for each of this command I will get an assembly language (())(03:24) so C0, C1 to C9 right and this is what we are going to look here. Now what is the biggest advantage of having this virtual machine?

(Refer Slide Time: 03:40)



Is that I could have several programming languages PL1, PL2, PL3 till PL n now I have one virtual machine, from this virtual machine I could have several interpreters which can give it for hardware architecture 1, hardware architecture 2, hardware architecture 3 (K) ok and then I have an assembler which will take it to the actual hardware ok.

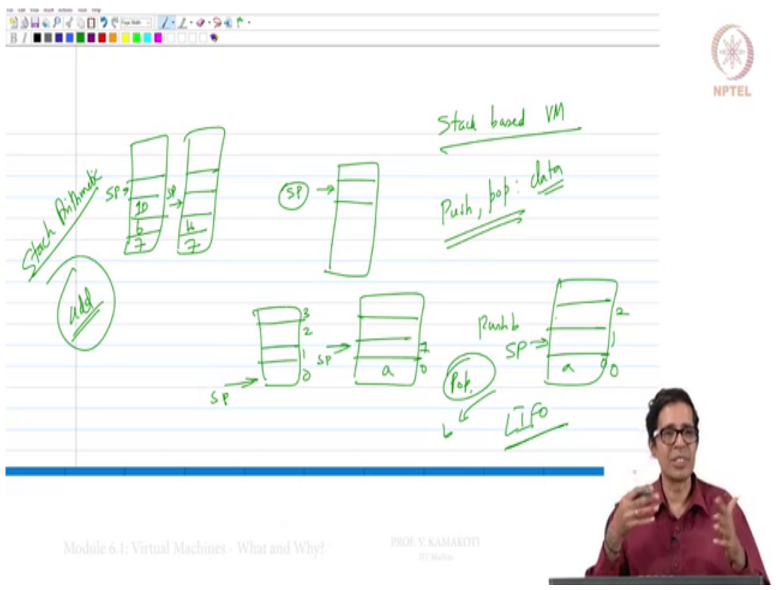
So multiple compilers running on multiple architectures multiple languages supported on multiple architecture this is what the virtual machine basically ensues I could have some language, so what it means to invent a new programming languages? This is some PL new and as planned as our complier which will make it into this virtual machine then that can be program of PL n can execute on all this architecture you get it.

Similarly what is means to now introduce a new architecture so this is one (())(05:18) of the story. Suppose I am putting a new architecture H of n right and if suppose I am H of new suppose I am getting assembler for this and interpreter for this which will convert so I have an assembler and a interpreter which will make from VM it will you know interpreted into assembly right, so the moment I have that interpreter that means all this languages PL1 to PL n can basically execute new architecture.

So this means this basically his approach of having virtual machine will basically allow you to start inventing new-new architectures and use all the existence of tour on top of it with somehow lot amount of is right and similarly keep inventing new programming languages and keep and make it executable and all the existing architectures very quickly. We suppose this virtual machine did not exists and I make a programming language and then basically I have to do for every architecture another compiler which will make it totally useless right it is impossible right.

Similarly when I put suppose this virtual machine (())(06:38) I create a new what you say a new hardware then for every programming (new hardware) for every programming language I need to have a separate software stack for it. Now all this things are avoided by the virtual machines and that is the power of virtual machine. This concept was it is a old idea in 1970's it started but it came back in full spirit in the 1990's when (())(07:08) started using this as a concept right so now what we will do in this particular thing is that we basically understand this virtual machine in it proper perspective.

(Refer Slide Time: 07:29)



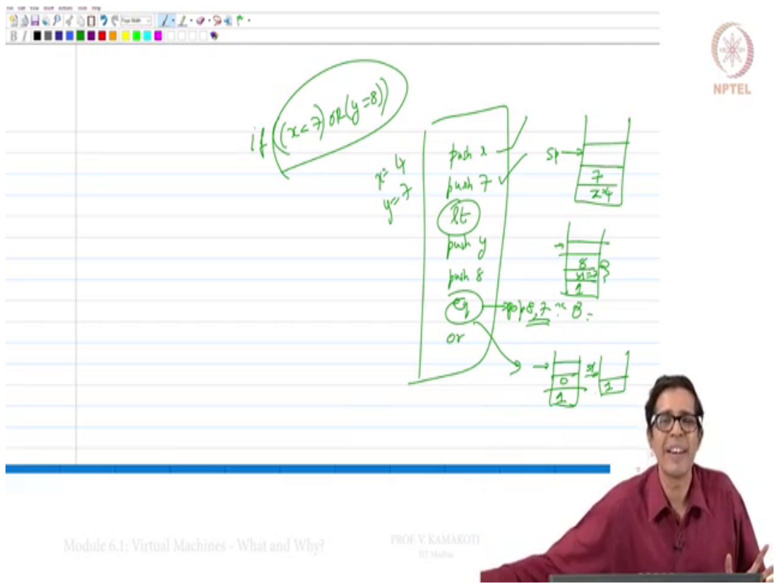
The virtual machine that we are going to use is a stack based virtual machine. Stack is a data structure where it supports two command PUSH and POP in addition on this this is basically data instructions into the stack I can push an element I can pop an element and there is always something called the stack pointer which will point to the top of the stack and I so let us do simple operation so there is the stack there is a stack pointer initially it will be pointing to zero, let me say this are all the memory locations.

Stack is implemented on your data memory right so 0, 1, 2, 3 is there suppose I say push A so A will come here and here sorry suppose I say push A so A will come here and here stack pointer will now point to this. Now I say push B now A will come here B will come here now a stack pointer will point to this. Now I say pop then what will happen? This B will be coming out so it will return a value B will come here and your stack pointer will now start what? You may erase this or not normally we erase it for security reasons ok.

So now when I say pop then the top most comes out, so B went last in and it came out first so this stack follows what you call as the LIFO policy that is last in first out policy. Well in addition I this stack I can do some operations so like for example I have pushed 7 I have pushed 6 and the stack pointer is here I can say add, what will add do? It will pop the first two top two operands or let us say pushed 1 also here 10 here and 9 stack pointer is pointing here when I say add this will pop 10 and 6 it will add it and again push it so what will happen is at the end of this this will be 7, 10 and 6 are popped and again it is added 16, so 16 goes here and then I have SP.

So this is how I do what we call as stack arithmetic. Your entire virtual machine is based on the stack so will have stack data moment and will have what you call as stack arithmetic right in then we can also do some simple operations like you know compare.

(Refer Slide Time: 10:50)



So I can say if X is less than 7 or Y equal to 8 right so I will say push X push 7 ok less than push Y push 8 equal to or this is statement complete so what will happen is initially stack is empty now I have pushed X then I push 7 so this is done then I say less than the stack pointer will be pointing here ok.

Now when I say less than the second bottom is compared with the first bottom so X is less than 7 and what will be this less than operation do, it will pop the first two operands first top two in compare the bottom with the top like X is less than 7 right and if it is really less than 7 it will push 1 otherwise it will push 0, so let me say X is 4, X is 4 and Y is 8 ok so now this will be 4 now the moment less than executes 7 will be compared with 4 will be compared with 7 for less than this 4 is less than 7 so what will happen is this two will be popped out and the answer will be written 1, 1 means true and the stack pointer will be there.

So what less than will do it will popped down the two entries compare them in that order bottom in the left hand side and top on right hand side because less than is not a (())(12:45) A less than B is not B less than A right so we have to be careful of which is on the left hand side and which is on the right hand side and then the answer is if it is true 1 will be pushed if it is false 0 will be pushed and say if just for to make life little (())(13:02) now I push Y now when I push Y what will happen? Stack pointer will go up now Y is equal to 7 then I push 8 right that is a constant so now the stack pointer is now here.

Now I say equal to so 8 and 7 are popped so this will pop 8 and 7 and it will compare them, so 7 is not equal to 8 so it will put 0 so the answer is 0 and that will be pushed so at the end of this you will have 1,8 and 7 are ofcourse popped and now 0 is pushed and the stack pointer so this will be the status of the stack after this. When I say OR, OR again will take 0 and 1 and I will do an OR and the answer will be here 1. So this essential expression basically gets evaluated.

So this is how I can meaningfully (())(14:01) to do certain logical comparisons also right so this is all this so our virtual machine is going to be a stack based machine and now we will see in the next module what is the input language to this virtual machine and for every instruction in that language will have something like 10 instructions we will see what is the equivalent assembly mnemonics of hack ofcourse we have an assembler so we can afford to write it in assembly mnemonic and take it further right. So we will now meet again in module 6.2, thank you.