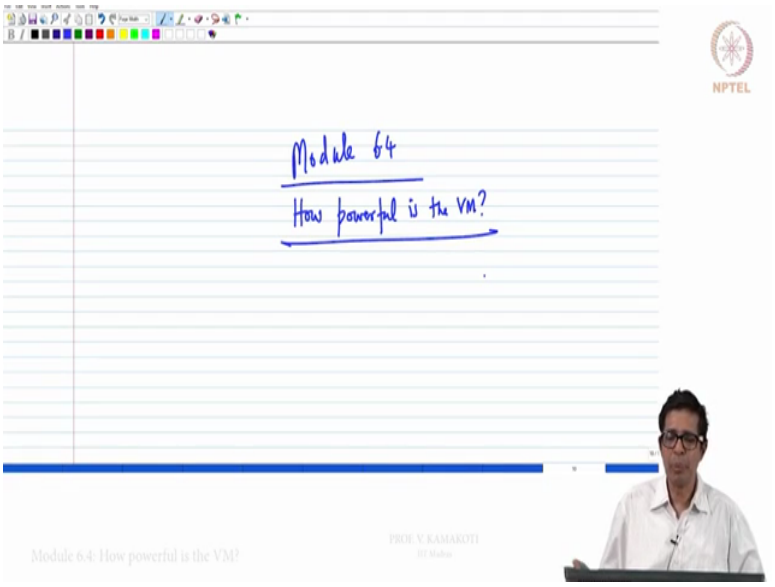
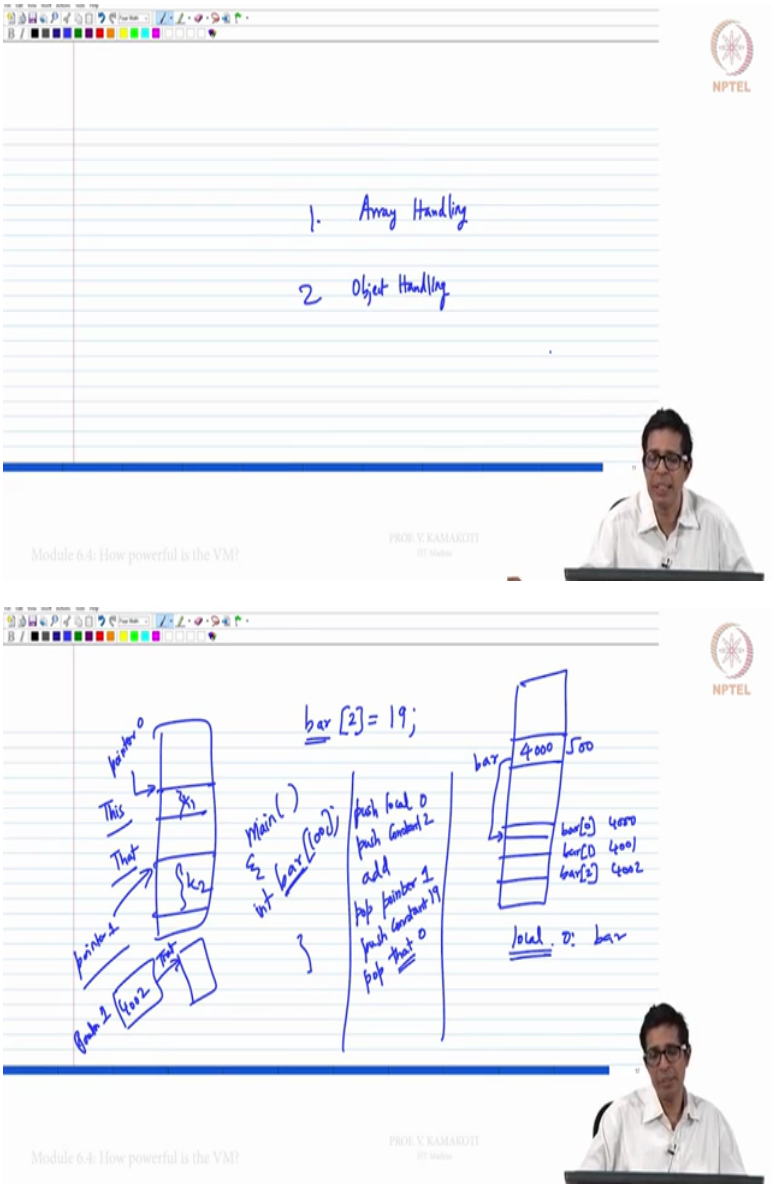
Foundations to Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 6.4
How powerful is the VM?

(Refer Slide Time: 00:16)



So welcome to module 6.4, in this module we will understand how powerful is the VM? So we will actually take two examples from common programming languages actually in module 6.3 we actually translated a C function into VM but now we will do something little more complex than that to show the power of the VM instruction set.

(Refer Slide Time: 0:42)



So we will take two examples one is namely array handling and another is object handling. Now let us say we have an array called bar, right and we are going to set that bar the second element of this bar 2 to 19, right. Now in practise what will happen is in C now bar itself will be mapped on to some location say 500 which will point to some 4000, in this 4000 bar 0 will be stored, 4001 bar 1, 4002 bar 2.

So essentially what we have to do is we need to go to the location given by bar, bar is mapped on 500 we need to go to that location that is 500, get that value 4000 at the index 2, so you get 4002, in 4002 we need to go and store 19, right. So this is how the whole thing has to work. Now let with the class of generality let us assume that bar is so in this function

whatever let it be some main, bar is your first variable shall I say int bar under, so what it stores, you know it stores the bar is the first variable, right.

So what we do is so in local segment, right bar is a local variable, right local segment 0th location bar is total. So what we need to do is first we have to get right 4002 and then store something there, so push local 0 that means the value of bar will be stored in the stack that is 4000 will be stored. Now I push constant 2, essentially 2 also will go there, now I add. So now I get 4002 because local 0 local variable 0th location is bar, bar's value is 4000, always the variable will store the starting address of the array.

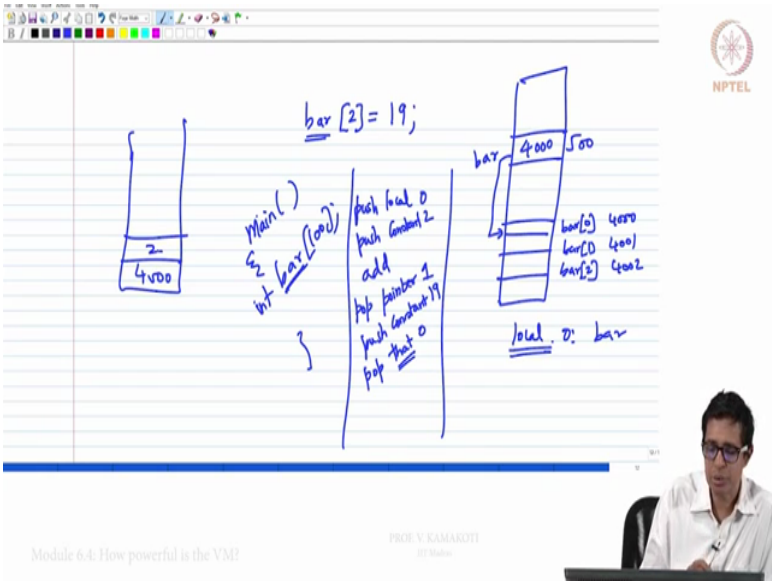
Now I have push constant 2, so 4000 plus 2, 4002 I added this so I got 4002. Now I have to store in the location 4000 to the value 19, now what I do is I pop, so what will be on top of the top of the stack now 4002 because I push local 0, 0 will be on the top, then I push 2 push local 0, so 4000 will be on the top, then push constant 2, 2 on the top when I said add 4002 will come, 4002 will be on the top of the stack.

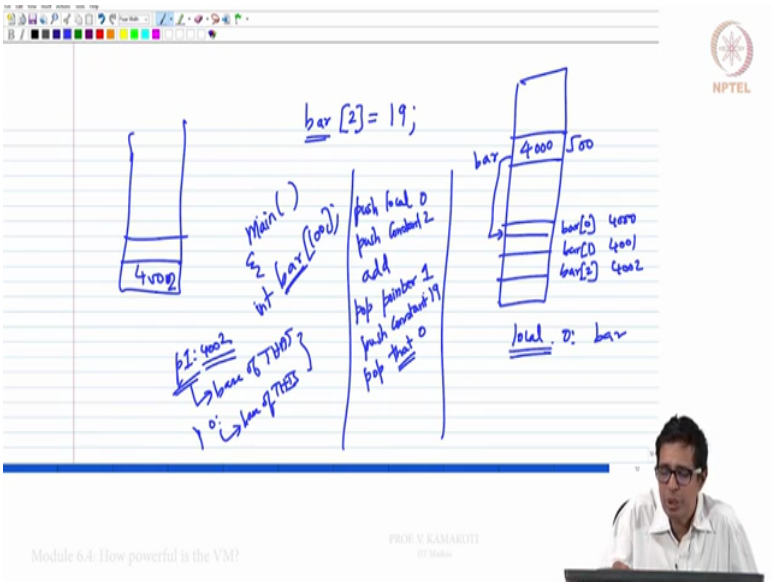
Now what I am doing is I am popping this value, popping this into what we call as pointer 1, right. Now for the use of the function we have two general purpose memory segments one is called this, another is called that. So in your entire RAM this this and that will occupy some k locations k 1, k 2 locations, right. So this is a segment, that is a segment, this and that are different segments this can be used for any general purpose.

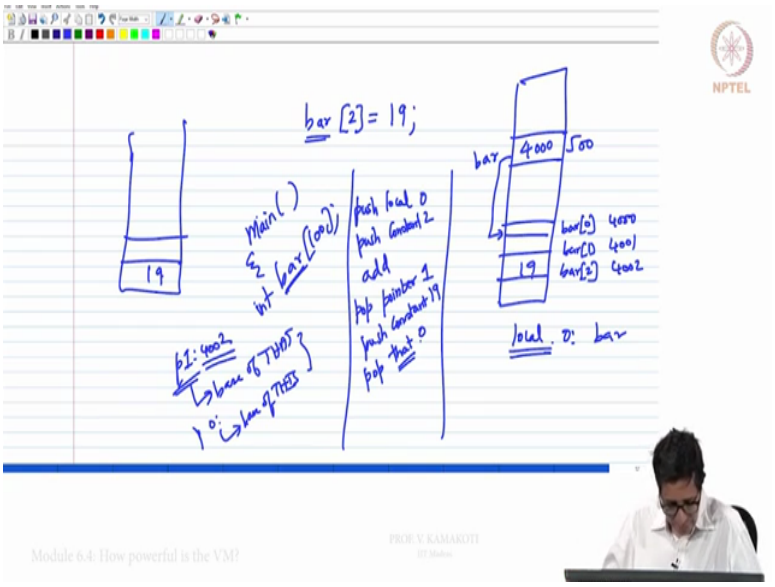
Now somebody has to point to the start of the segment and that is pointer 0 and pointer 1, pointer 0 and pointer 1 will actually point to the start of the segment namely this and that. So when I say pop pointer 1 that means 4002 is now pointer 1, essentially that means 4002 is the that segment now, so your that segment now starts at 4002, right. Now I am doing push constant 19, so 19 is on the top of the stack, now I say pop that 0, the moment I say that where is the start address of that? That is given by pointer 1 that is 4002, 4002 and 0 is the index of 4002 plus 0, 4002, in 4002 I need to go and store 19 so this is how the entire thing works.

So what I have done? This is the main routine I have int bar 100, so the bar is the first local variable 0th local variable which is stored in the local segment at 0th index.

(Refer Slide Time: 7:00)





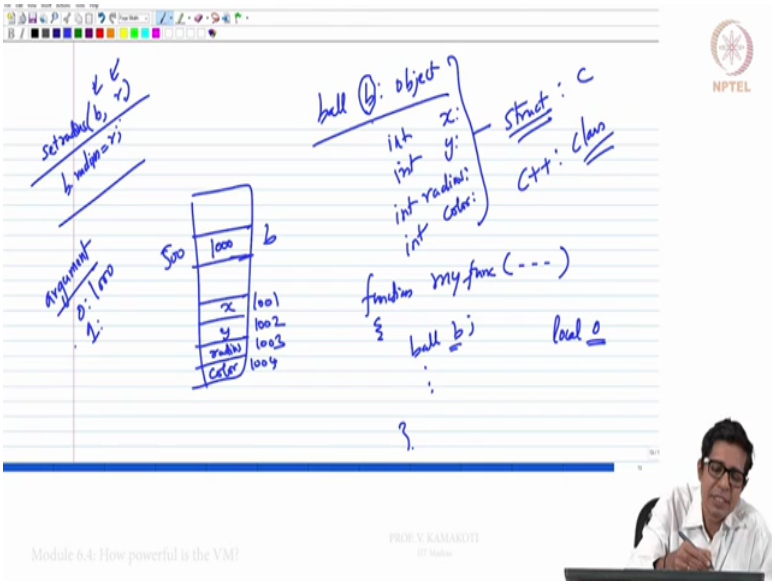


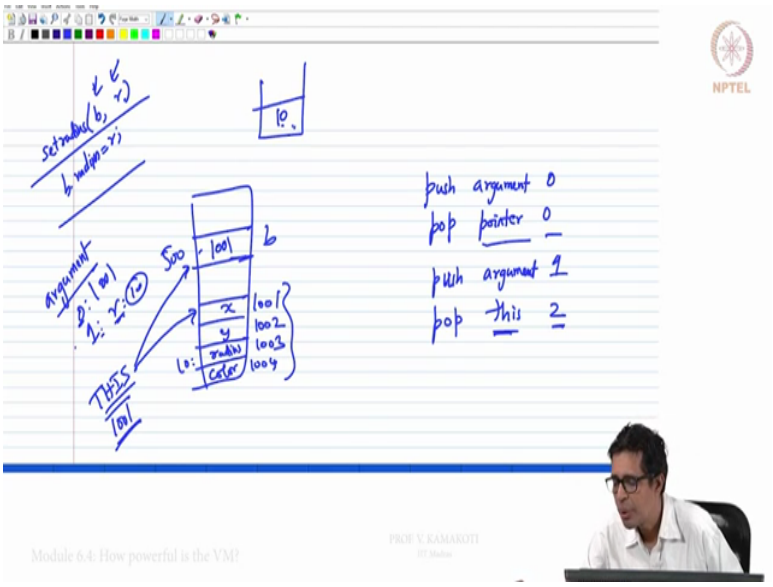
Now I want to make bar 2 as 19, so when I say local 0 basically has the address of the start of bar in this case it is 4000. Now I push that address into the stack, so my stack so let me just show you the stack how it is going to go here so that we have very good understanding of this, so now in the stack I will have 4000 because of this push local 0, now I am pushing constant which is 2, now I am adding it basically now this becomes 4002, now I am popping this 4002 into pointer 1, so pointer 1 will get 4002, pointer 1 always points to the base address of the that segment and this and that are basically temporary segments. So (P1) pointer 1 will always point to the base of that, pointer of 0 will point to the base of this, correct.

Now I pop pointer 1 that means your pointer 1 got 4002, so this is gone because I popped it out, now I am pushing constant 19, now I pop 19 to that segment 0th index in that segment so where does the that segment start, it starts at 4002 because pointer 1 gives you the start of the that segment, so in 4002 plus 0 this is the index, 4002 plus 0 that is 4002. In 4002 I will now pop this 19 and store it in 4002, so that is how I get bar 2 equal to 19, right.

So this example is a very very simple example but what we need to understand is about different segments, etc so we have learnt what is that, what is this, what is pointer 0, what is pointer 1, what is constant, what is local, some of the things like argument, etc we saw in the previous example in the previous module, now we are now seeing this in some details, okay.

(Refer Slide Time: 9:14)





Now the next thing is we will also discuss the power of this VM by understanding what object had link. Suppose I have a object called (())(9:21) ball which has you know radius and a colour colour can be and it also has the you know so we are say we are now try to make a game so this ball is moving on the screen so it has x, y coordinate let us say x, y are also integers.

So a ball b so we are now making a it is an object, we are now trying to do a game with all these balls floating around here and there and so this ball b is an object and it has some x coordinate, it has some y coordinate and then it has radius and it also has colour there can be multi-colour, right. So normally let us say these are all int int just to make it comfortable, so in C we write it using as struct structure this is in C, in C plus plus you can actually use a class and instantiate an object, right.

Nevertheless, in both cases if I just say b, b will give me the address where it is stored. For example, so let us say I have a function let us say my func, in this I have ball b, right. Now essentially what will happen is b is a address, so b is mapped on to say some so with the class of generality 500, so this is b. Inside b I will store some 1000, so in 1001 I will have x, 1002 I will have y, 1003 I will have radius, 1004 I will have colour.

So this is how this particular object is mapped on to the memory, now b is a local variable so let us without (())(11:54) say that in the local segment 0th location this b is stored, so local segment 0th location I will get 1000 there, from there I have to go and calculate. Suppose I want to set the radius to r so suppose I have a function called set radius b, comma r, b is a ball

and r. Now I have to set b dot radius is equal to r, suppose I want to do this, so the address b is coming in and I am getting an int r so I have to set b dot radius is equal to r.

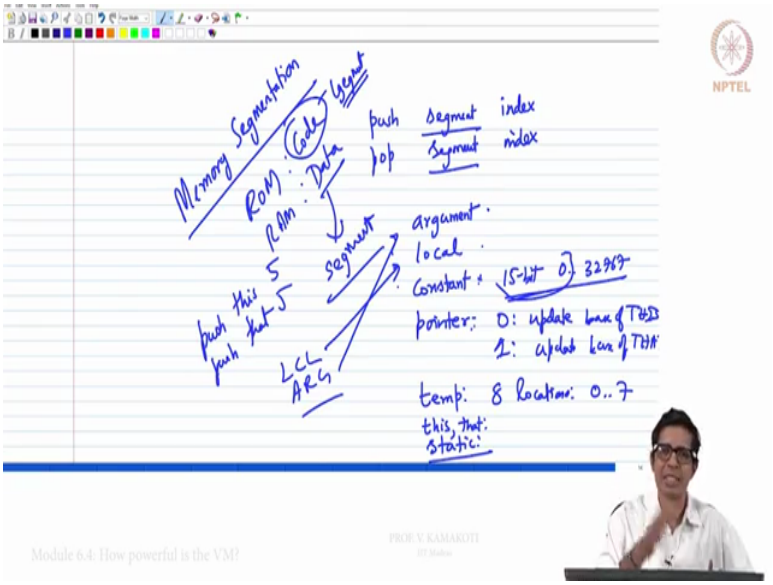
Now what is b and r? b and r are basically arguments, right. So in the argument section, in the 0th location b's base address should be stored b will be stored that means 1000 will be stored, in the 1 the new radius r will be stored, push argument 0 I push to 1001 inside because argument 0 is b sorry this should be 1001, I push 1001 inside and I popped it to pointer 0, so that means the base of (())(13:27) this segment is now pointing to 1001.

Now I push argument 1, argument 1 is r so I push 10 into this, now I pop that 10 into this segment with index 2, this segment is pointing to 1001, 1001 plus 2 is 1003, so this 10 gets assigned to radius. So this is how we make b dot radius is equal to r, so essentially I am having (two local) two arguments b and r, b is nothing but the base address of the structure which stores the details about the ball or is an integer which is the radius, so the base address of the structure which stores the details of the ball in this case 1001 is stored in argument segment 0 and argument segment 1 which is the first argument is stored as 10.

Now what I do is I push the argument 0 that is 1001 on to the stack and then I pop it to pointer 0 that means 1001 comes to point to this so that this segment's base (is pointer) is now 1001 that means this segment essentially points to the ball b, right. Now I push argument 1, argument 1 is r for me so I am pushing 10 into the stack and now I say pop this to so now is pointing to the ball that is 1001, now when I pop this segment plus 2 that is 1001 plus 2, 1003 so I pop the value 10 onto this radius which is 1003, right so this is how we do this.

So essentially now we can handle objects, we can handle arrays, we have given two interesting examples which are not very trivial in terms of realization and but we have realized this.

(Refer Slide Time: 15:31)



So to sum up this particular module let me just go ahead and tell you what are all so we have two memory access functions that we have seen one is push segment index and pop segment index, so what are these segments? The segment can be argument, the segment can be local, the segment can be constant wherein this is a 15 bit positive constant in the range 0 to 32767 so this is the range.

It can be a constant, right, so the index of this constant should be just a 15 bit number, so argument local we have already seen. Then I could have pointer, this will go an update so pointer 0 will update if I say pointer with index 0, this is the index, pointer with index 0 then it will update base of this segment if I say pointer 1 it will update base of that segment and this and that can be used for all temporary computations as we have seen in the previous example.

Then I could have temp which is just maximum 8 locations that is index 0 to 7, this I can use for some temporary computations if needed, then I have I can have this, comma that so if I say push this comma 5 it will go to pointer 0 that will give you the base address that will add 5 and what so let us that base address be 1000 so 1005 I will get that 1005 in that 1005 there is some say 512 as a content that 512 will be now pushed onto the stack that is what it means, right push this comma 5, similarly I can say push that comma 5, similarly pop this comma 5, pop that comma 5, so it is comma that.

Then I could also have static because as a function I would have some static variables, static variables will be stored separately so I could have, so essentially you are seeing now 8 such

segments 8 such things that can go as segment, each is some separate as far as the function is concerned each is the separate part of the memory and each is a separate segment and each has its own base address and we need to access from that base address plus an index.

So this gives us a concept of what you learn in your computer organization course as memory segmentation, already when we de the architecture we have two segments two memories itself one ROM and another RAM I hope you remember, this ROM is basically storing code, RAM is actually storing data. So this is code segment so we have already segmented it, while RAM is data segment and inside the RAM is what you could have several such segment for every function, right.

Now just to just keep in mind that when we constructed the symbol table we had thigs like LCL, ARG, etc now some bell should ring this is argument local so for every function that we are compiling the base address for your argument, the base address for your local segment, the base address for your temp segment, etc will be stored in these variables like LCL, ARG you will see how it is going to be done, right.

So what you have done so far, so we have understood the 17 instructions of the virtual machine, 9 of them are basically the arithmetic logic instructions we had 2 memory access now and 3 each of your program flow and the program and the function calling instructions and so hopefully this gives us this gives you an understanding of the VM works and also the power of your stack based virtual machine.

So what we will do in the subsequent two modules is that we will see how these instructions are basically each one of these instructions will take add, we will take all the ALU instructions, all the function calls, all the memory calls, memory access, all the 17 instructions each of these 17 instructions how I can translate into the corresponding mnemonic and that is what we will start seeing in the next two modules, module 6.5 and module 6.6, hope you understood this, I have tried my best to explain so there can be still doubts so do not give up please do put your queries on the discussion block and we will basically answer these queries and also do not repeat the queries please check if your query is already answered in the discussion block, if it is answered then you can take it, if it is not answered and it is a new query you are always welcome to put that query.

So this will help us to you know optimize answering queries and so that we could do more justice to each query rather than answering 1000 similar queries if I answer once but I do

better answer for that, I think it will be much more useful. So all the best with your understanding of this VM, I hope you completed your assembler project that is very important for us to now start understanding the remaining two modules. So please do start your module 6.5, right after you complete your assembler project, right thank you.