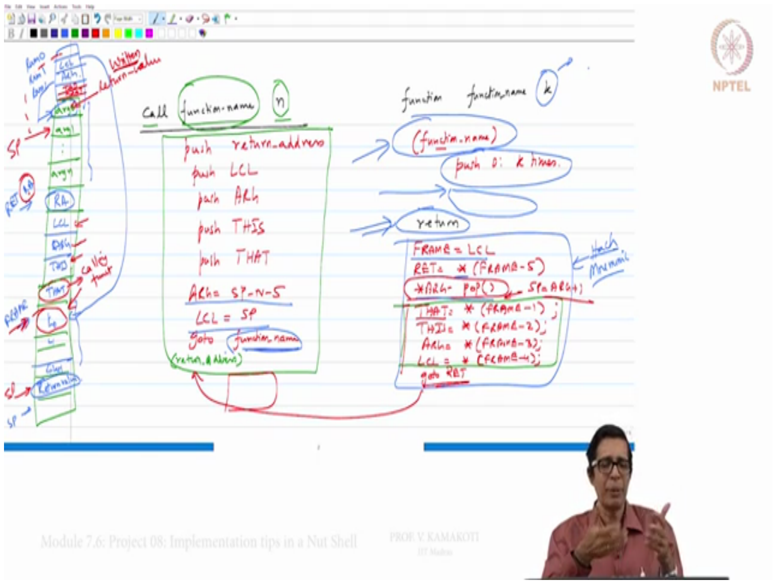
Foundations to Computer Systems Design

Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Module 7.6
Project 08: Implementation tips in a Nut Shell

Welcome to module 7.6, so this is the last part of module 7 and in this we will talk about certain implementation tips in a Nut Shell for specifically for function calls which is part of project 8.

(Refer Slide Time: 0:31)



So whenever you see in the VM file the statement call function name n, you should immediately replace it by the hack mnemonic for this the set of statements like push return address, push LCL, push ARG, push this, push that, ARG equal to SP minus n minus 5, LCL equal to SP go to this function name and then this will be your return address, so create a label here.

So this is what, so call function n will be executed by a calling function and what we assume n is the number of arguments, so what we assume here is that before this call function name n is done all the arguments for this function are already pushed into the stack. So let us say so this is the stack before the calling function is going to execute this call function name n the arguments will be pushed into this.

So argument 1 to argument n are already pushed into the stack and your SP will be pointing here, at this point you have call function name n or call some fact factorial, right call fact n whatever. Now this call is basically getting translated into the set of statements so what are the set of statements? Now so one is push return address so return address will get pushed here, so this is what happens at the time when the call is happening.

Then LCL of the calling function the local this thing LCL will be somewhere in the stack that address where LCL of the calling function stored will be pushed here, then ARG will be pushed here, this will be pushed inside, that will be pushed inside, okay and your SP will be now pointing to this at the stage. Now (the other three statements) the other two statements basically would be are ARG, where is ARG stored? ARG will be in some RAM 0, RAM 0, RAM 1, RAM 0 will be SP, RAM 1, RAM 2, right.

So your new ARG will be SP minus n minus 5, so your ARG which is stored in say RAM 2 will now this statement ARG is equal to SP minus n minus 5, SP is this, this is 5 minus n will go so this ARG will now point to this, okay ARG (())(3:33) and your local will be also pointing to SP. So all these things will happen in the calling function before I transfer control to the call the function, right so your local which is RAM 1 that will also keep pointing to this so that is what this will do.

So what before going to the calling function before going to the called function the calling function basically pushes all its local segment, ARG segment, this, that, into the stack it also pushes the argument for the calling function is already pushed into the stack, it also pushes a return address that is after the called function finishes where it has to come that is also pushed into the stack and then the argument for the called function and the local for the called function are initialized, the argument for the called function starts at this ARG 0, this is already initialized by this and the local for the called function is also initialized by here, right.

Now we go to the so when I say go to function name, so there will be some function with that function name k, so there a label is created so the moment I see function, function name k in the VM file what is the equivalent translation? Just put a label called function name and then a code which will push 0 k times or k is the number of local arguments. So when this finishes we go to go to function name when the call statement this statement finishes, now I go to function name, this go to function name will take us to this point here and where I am going to push 0 k times.

So when I push 0 k times so this will be so will say L 0, L 1 to L k, so L k minus 1, so your stack now will become this and your local variable will be pointing to this L 0 to L k minus 1. So when the function now after this your body of the function would be there, right. So when the body of your function starts executing, your argument for the function is set properly ARG 0 to ARG n as you see on the stack and your local variable L 0 to L k is also stored correctly on the stack.

Now the function executes whatever the call the function executes and we believe that whatever we want to return back to the calling function that will be on the top of the stack, so this is the return value, after the function executes the called function executes the return value is on the top of the stack and after that it will execute a return statement as you see. So the moment you see the return statement in your VM file you have to replace it with this type of hack mnemonic equivalent of these statements, okay.

And what this will do, your frame is LCL first statement so your frame will be pointing to this, frame is LCL. Now return is frame minus 5, so this is frame, frame minus 5 is this so return will be having the value RA, frame will be having this address, return will be having the content if I put star, right star of frame minus 5 the content of this so return will have this.

Now star ARG equal to pop, right star ARG is equal to pop means, when I pop there is a return value this is the value the called function is going to give to the calling function, this I will store in star of ARG, what is ARG? ARG is pointing to ARG 0, so the return value is now return to this the return value is return the return value is return into this ARG 0 that is what this star ARG is equal to pop will do.

So when I say I pop your SP will be here, now the frame is here right, now what I am? Frame is pointing to this address so now I go and re initialize my that, so this that will be in RAM 0, RAM 1, I will now go and say my the new that is frame minus 1. So frame is pointing here as you see frame minus 1 is this that, this is the that of what? The that of the calling function, this is the this of this calling function ARG and LCL.

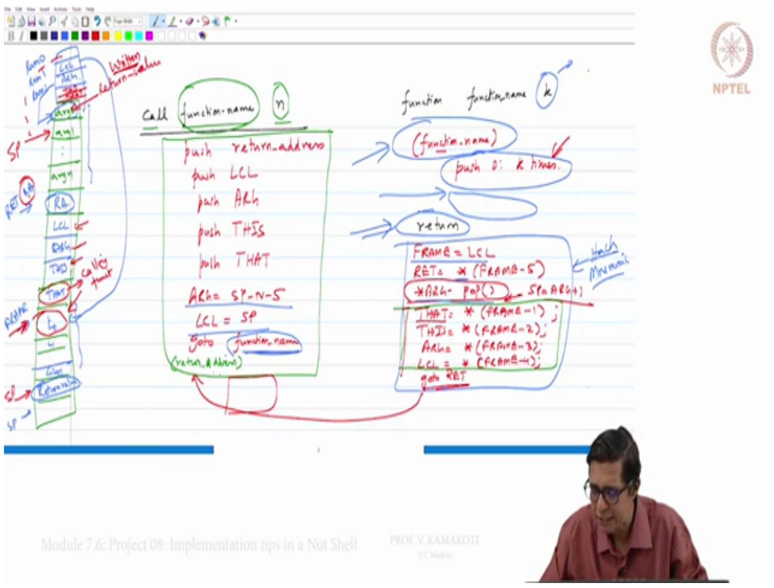
So these four statements as you see here will go and initialize my that, this local ARG that is stored in RAM 0, RAM 1, RAM 2, RAM 3 to that of the calling function because I am now going back to the calling function, right and then I say so SP is equal to I forgot one statement here, SP equal to ARG plus 1 so this should come here between this and this, my SP now which is RAM 0, SP will now point to ARG plus 1, so ARG plus 1 is this, so this will be my new SP that is with RAM 0, right.

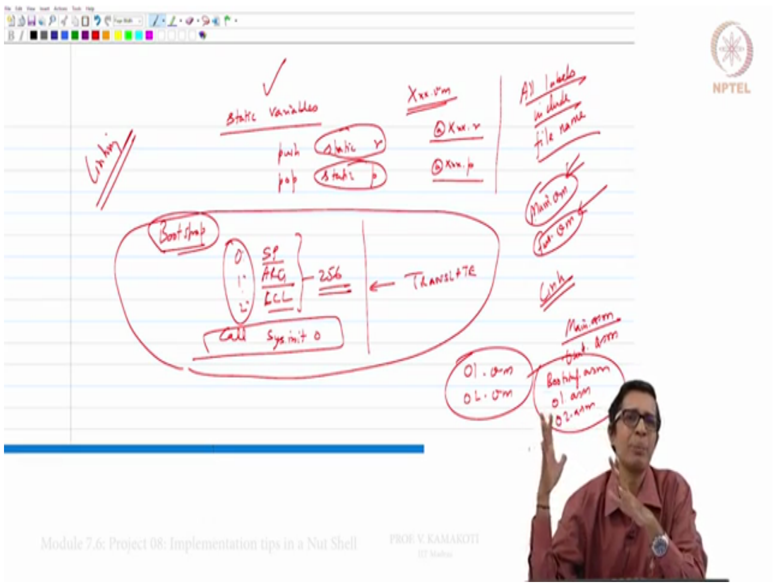
And then I say go to ret, what is ret? This is the return address, return address of the calling function. So this will take me back to this return address and I will start working from the remaining part of your called function, right. Now when I return back please note that the return value which was put in ARG 0, right ARG 0 the return value is in the top of the stack and the stack starts from here. So the return value is SP points here, so the return value is in the top of the stack.

So for the calling function whenever it wants to return it wants, what is the return value? He has to just pop from the stack and that will be the return value and then it will start executing and before the control goes from the calling called function to the calling function the LCL, ARG, this and that of the called function calling function is actually restored by these four statements that you see here as a part of return and then you go back to start working.

So when I see called function name n replace it by these 4 plus 4, 8, 9 instructions including this label, when I see function function name k as a part of your VM file just replace it by this function name and the hack mnemonic equivalent of push 0 k times, whenever I see a return we need to replace it by these 1, 2, 3, 4, 8, 9 instructions and this is with that the hack mnemonic equivalent of these 9 instructions. So this is precociously what needs to be run and the very simple thing. So just go and replace when you see this in the VM file just go and replace with this.

(Refer Slide Time: 11:26)





Now some small things that we need to take care, another thing that we have talked about is the static variables I just want to tell you if I say push static are or pop static p we need to find out what is the address where this rth static variable is stored or the pth static variable is stored that you get it by at if suppose this file is Xxx dot VM you get it as Xxx dot r, Xxx dot p, right this will be the address where the static because the assembler will take care of assigning the address and then all the labels we have generated so many labels here, like go to return address, whenever I do a call I create a label called return address here, right.

So I create a label like that we have several even now we do push 0 k times this is a loop, there will be address that is generated so that we need to every time a label is there you should include the file name in that, right I would have several dot VM files and all there will be one main dot VM, there will be something called dot VM, all these things there will be some function dot VM, so all these VM's should work together.

So when I am interpreting VM, this VM and this 2 VM file separately then the same label will get repeated that we have already explained in the previous but just as a nut shell you need to take care that every label that you are generating should include the file name so that there is no collision in the label. In addition when we have more than one VM files this another thing that we have to take care is that we have to link them properly, link them means that first suppose main has to execute first and it is calling (())(13:12) put main dot VM first the main dot ASM first followed by (())(13:18) dot ASM, which ASM file has the first set of instruction that should come and that is how it will getting executed.

In general in two of the exercises we talked about bootstrap code, before that the script basically initialize the stack pointer ARG, LCL, but generally this is what we need to do, so just put bootstrap in the bootstrap code write hack mnemonic equivalent of making SP, ARG, LCL as 256, so this is RAM 0, this is RAM 1, RAM 2, or 1, 2 just confirm this and may call of them 256, right and then write the hack mnemonic code for call sys dot init 0, right sys dot init 0, right so this will call sys dot init, right and then from that that will call main and so on, okay.

So this is the bootstrap code, so there will be a sys dot init code in each of the exercises that you have, so that will call sys dot init and from there you will start executing. So this is basically the bootstrap code that we need to always (())(14:33). So if I have more than one VM file to be compiled, so order them so order 1, order 2, then before this thing you will have bootstrap dot ASM, you have a bootstrap dot VM that you converted into bootstrap dot ASM followed by o 1 dot ASM and o 2 dot ASM just (())(14:54) these three files together and start the execution it will start executing (())(14:58) so this is very very important so this is basically what we call as linking different libraries when you do the compiler you will learn much more and much elaborate things, this is a very very simple way of linking, right.

So with this the project 8 which is very important for you ends here, so please start working so whatever I explain, how much every I try to explain unless you work on that project you will not get the complete clue so work on the project, it is like you know I as I have been mentioning so far I cannot teach summing by showing demo slides so you have to go to the pool wet yourself and that is how you will learn swimming.

So doing this particular project assignment is like doing swimming, so just swim do not worry we will be the life board or the you know the rubber tube which will help you when you are drowning is the discussion forum, I hope it is very alive and kicking so just use that forum and we will be answering queries, we will be helping you to this assignment whatever possible from our end, try and finish it and it is a good challenge for you, right with this we close the module 7 and now we go into module 8 there we are going to talk more about high level programming, the jack which is the (())(16:19) version of Java and that will be the final phase of this course, right all the best.