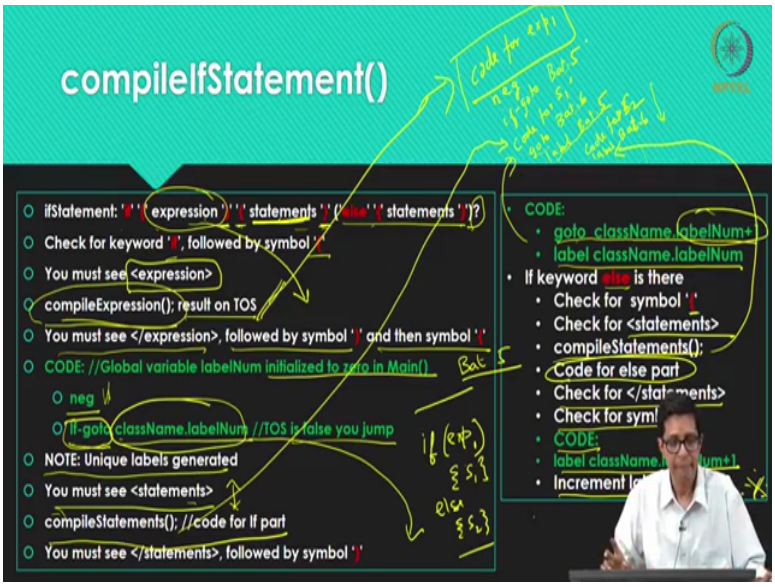
Foundations To Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 11.5
The Jack Compiler Backend: Code Generation - 3

Welcome to module 11 point 5 and this is code generation 3.

(Refer Slide Time: 00:20)



In this now we will see how compileIfStatement is coded, so ifStatement is of the form if parentheses there is an expression then there is a square brackets then there are statements ended by a square bracket and an optionally else please note that there is a question mark here if you see here, so there is an optional else if the else is there again there is a square bracket then statements again, sorry there is a braces then statements again ending with a braces, so this is how the if is constructed.

So how do you write the code for compileIfStatement? You check for the keyword if, you followed by the symbol parentheses left parentheses, now you must see expression as a token, ok then immediately you call compileExpression and as we have seem it will generate the code and the result of this expression will be on the top of the stack, then you must see slash expression next as your next rule followed by the symbol closing parentheses and then the starting symbol opening braces, ok.

Now you write the code, the code that is jumped us, now the compileExpression would have already jumped the code and the result of executing that code will be on the top of the stack.

Now you put neg means the top of the stack gets negated, right so the top of the stack gets negated, right and so now you see if-goto className dot labelNum, ok if-goto is as we know this is a VM statement, so if your top of the stack is false you would actually jump to this label called class (labe) className dot labelNum, right.

Now labelNum please note that you have several, so finally you will have several class jack files that you are going to compile and integrate, so every label that we are generating should be unique, so for this purpose what we do is that we create a global variable call labelNum in every when we are compiling a class we will create a global variable call labelNum this labelNum will be initialize to zero in the main routine and as an when you are utilizing this labelNum you are generating more and more labels you keep incrementing this labelNum and to make the labels unique you also append this className in front of that, so you will call className dot labelNum.

So suppose you are having class call you know Bat, Bat is a class that you are currently compiling, so it will be Bat dot 1, Bat dot 2, Bat dot 0, Bat dot 1, Bat dot 2 etcetera, so you may be compiling another class file which will be some ball then you will say ball dot 0, ball dot 1, ball dot 2 so that it will be different, ok. So by doing this exercise you will be generating unique labels and that is very important otherwise the compiler will get confused, right the VM file, VM interpreter will get confused, ok.

So what does happen here? So we have checked for if, we have checked for expression, we have return compile expression, the result of that evaluation of the expression is now currently available on top of stack you do a neg and then say if-goto, so if I do the neg and say if-goto, so this jump will happened if the top of the stack is false because we have negated here, if the top of the stack was originally false then this negate will make it true, so this jump will happened if your expression that you have put here is not evaluated to true.

So right, so this essentially this jump will happen if your expression is false, the negation of that expression becomes true and if-goto will actually make you jump there, ok, right. So then use then you just do this then you just you must see and you should also note that these labels generated are unique, right and now you must see statements after this because you have checked till this close parentheses a open braces.

Now you must see now statements, the moment you will see statements you will do compile statements and this is the code for if part, now you must see slash statements followed by this

close parentheses, right. So what you have done here, now you have generated a code for evaluating this expression here then you have that code will ensure that on top of the stack you have the result of that the evaluation, you negated and then you put these statement if-goto className dot name so what will happen is That the jump there will be a jump here and after this if-goto statement this compileStatements will actually generate the code for this if part of your statements that it will be here.

So if your expression evaluates to false this neg will be make it true and this if-goto will make you jump or by pass this statement, right. So essentially so you have this structure now that means you are finished this if expression statements, so the if the expression is false these statements will not be executed you will be jumped of and if the expression is true then these statements will be executed that much as same.

Now after this you just put (got) you just put goto className labelNum plus 1, this was just labelNum, this is labelNum plus 1. So let us assume labelNum was 5 to start with, so this will be Bat dot 5 and now this will be Bat dot 6, so I will be goto className labelNum plus 1. Now this is actually className dot labelNum, now I will check for else, if else is that then again I will check for this statement this as opening parentheses symbol after that is should see slash statements there, so again I will call compileStatements this will be the code for the else part I will check for end after returning back from compileStatements I will check for slash statements, I will check for symbol close parentheses or close braces.

Now I will put this code label className dot label plus 1 and then importantly I will increment labelNum by 2, this is very important because this is very important for me to generate unique labels. So suppose I have if expression 1, s 1 statements 1 else statement 2, right let this class be Bat and your labelNum be 5 to start with, so what will happen is there will be code for expression 1 will be first generated by this compileExpression that you see here then you will have this neg, ok I am just writing it again here.

So code for expression 1 will be first generated by this routine then this code will come, so I will put neg then I will have if-goto Bat dot 5 then I will have the code for s 1 put here then I will have goto Bat dot 6 that is given by this code, the code for s 1 basically comes from this compileStatements this will give you code for s 1 then I go and put goto Bat dot 6 then I will have label Bat dot 5 then I see else of course then I will put code for s 2 here and that it will be generated by this person, this compileStatements then I am putting here label Bat dot 6 at the end of this, right.

So what is happened, if expression 1 is true your neg will make it false, so the shift goto will not happened, so the code for s 1 will be executed and then I will goto Bat dot 6 I will jump Bat dot 6, the code for s 2 will not be executed I will go here, if the expression evaluates to false then neg will make it true and if-goto will work I will goto Bat dot 5, so this is label Bat dot 5 I will execute code for s 2 and then come out, right so if the expression is true code for s 1 will be executed, if the expression on it is false code for s 2 will be executed, so this is how your compileIfStatement is works.

(Refer Slide Time: 10:44)

Now the compileWhileStatement, so while works like this. So while some expression e 1, expression 1 I have s 1, so how does it work? First I will check for while I will check for this symbol then I should see slash expression I should see expression then I call, so before I call

compileExpression, so the code I will see here is label say some Bat dot 6 I am saying label Bat dot 6 that is dumped by this.

Now I go and do compileExpression, this compileExpression will dump me code for x Exp 1 and the result of execution of the code will be on the top of the stack, now I will come back I will check for slash expression, I will check for the closing parentheses here these closing parentheses. Now I do a neg I do if-goto Bat dot 7 because label labelNum plus 1, right. Now I will check for the symbol opening braces then I will do compileStatements and then I will check for statements then now since I am seeing statements I will call compileStatements.

So when, so this will now this compile statement will dump me, the code for VM code for s 1 for all these statements here, to be collection of statements so all those code will come here and I will now I will come back I will check for slash statements then I will check for the closing parentheses this one. Now here I will go and say goto Bat dot 6 and I will put label Bat dot 7, ok.

Now what happens here? I start with label Bat dot 6, I have the code for expression 1 if it evaluates to true the negate will make it false, so this if goto will not work, so I will execute the code for s 1 again I will go to Bat dot 6 again I will (expre) evaluates expression 1 again if it evaluates to true this neg will make it false this if-goto will not work, we will not take the jump again I have let you code for s 1 again I will goto back to Bat dot 6, so the while loop is executing.

Now if the code for expression 1 becomes false then this neg will make it true then if-goto will execute the jump will be taken and I will goto Bat dot 7 which is label Bat dot 7. So I will jump I will now come out of this while loop if this expression is works, so this is how the whileStatement is taking place and at the end of this most importantly I have to increment labelNum by 2, so that it becomes the so the unique labels are generated for the different things.

So what we have seen in this module is 11 point 5 is that we have seen how and the if statement is basically compile, I have also seen how the whileStatement is basically compiled, right and whatever we see on the blue green is what you need to dump as a code rather things you need to do the check and call the corresponding routines. So as you see for each of this routine hardly we just put 2 or 4 or 5 lines of code that is the maximum, so that is the generated.

So if you see whatever we have put in green is what will go as the VM code, right and other things are basically you keep calling and checking and incrementing some labelNum etcetera, so the same thing for the while statement.