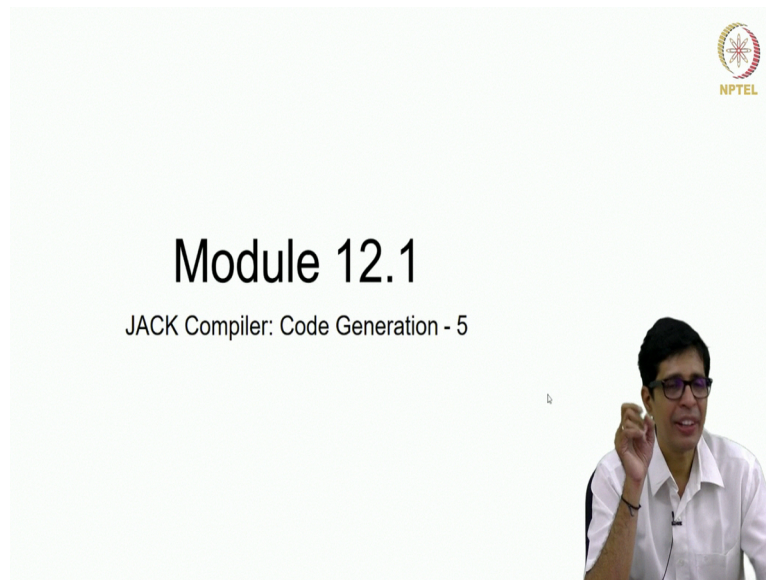


Foundations To Computer Systems Design
Professor V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 12.1
Jack Compiler: Code Generation - 5

(Refer Slide Time: 0:15)



So welcome to module 12.1, so we are on the doing the Jack compiler code generation, hope you finish the part of the project, which we decide at to do as a part of, towards the end of module 11 and hope we are able to generate the symbol tables and now will just proceed, so there were three routines that were left and there were a small changes that we need to do, it is left it as simple exercise, hope you would have realised those changes, but if you have not and these are some small bits and pieces that you may miss while you are writing a compiler. I just left it as a case study of how common mistakes could come while writing a compiler.

(Refer Slide Time: 0:58)

compileIfStatement() - 2 changes

- ifStatement: $\langle \text{expression} \rangle \langle \text{statements} \rangle \langle \text{else} \rangle \langle \text{statements} \rangle \langle \text{?} \rangle\text{?}$
- LabelNum = labelNum; Increment LabelNum by 2
- Check for keyword 'if', followed by symbol '{'
- You must see $\langle \text{expression} \rangle$
- compileExpression(); result on TOS
- You must see $\langle \text{expression} \rangle$, followed by symbol '}' and then symbol '{'
- CODE: //Global variable labelNum initialized to zero in Main()
 - `neg not`
 - `if-goto className.LabelNum //TOS is false you jump`
- NOTE: Unique labels generated
- You must see $\langle \text{statements} \rangle$
- `compileStatements(); //code for if part`
- You must see $\langle \text{statements} \rangle$, followed by symbol '}'

• CODE:
• `goto className.LabelNum+1`
• `label className.LabelNum`

• If keyword **else** is there

- Check for symbol '{'
- Check for $\langle \text{statements} \rangle$;
- `compileStatements();`
- Code for else part
- Check for $\langle \text{statements} \rangle$;
- Check for symbol '}'

• CODE:
• `label className.LabelNum+1`

So there is a small three changes that we, now in the compile if statement, this was dealt in great detail previous module, the only change is that instead of neg as you see I am just moving the browser here, so instead of neg we need to have not right, that is the only change, the second thing is that if I take a if statement what is this, so if statement is nothing but, if some expression followed by statements right.

So this is a recursive call, so again statements will call if statement, so there can be if inside an if or there can be if inside an else also right, so basically that is why we call as a recursive grammar right, so a statement, if statement can still invoke another if statement, now what we have done in the case of if statement is that we had one label, so only issue now is the label num, so the label, if you look at the previous, so we have one label num that we were using it continuously across this but none.

If I use a label num and then that label num, label num +1 etc are going to be call, for example, you are going to, when we are processing an if statement, we are going to call again compile statements here, so we have a label num and that label num we are going to use after the compile statements right, so the label num +1, so the compile statements can change the label num because in this compile statement can again call if statement and it can change the label num.

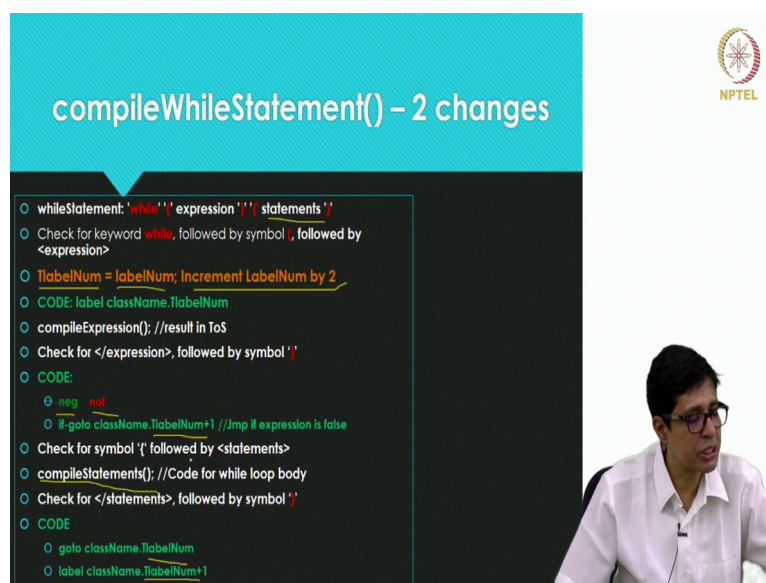
So the label num that you will be getting here will be different from the label num that we have taught here that should be there, so this label num can be changed by these compile statements, we have not assume that the change will happen right, so the label num is a global

variable, every time this if statement is called that the label num is getting changed and since this if statement can essentially call statements which can again call if statement.

So by the time and that will be part of your compile statements, by the time you reach this part your label num can be different from the label num that you have already seen here, so what we do is when we enter if statement we have a local variable called t label num and that you are signed to the label num and increment your label num by 2 and all your label nums you replace it by t label num, here also it should be replaced by t label num right, and here also very important right, so this label num and so this also should be replaced by t label num.

Right all your label num should be replaced by t label num right, so then every time even if the is the if statement again calls compile if statements since this t label num is a local variable, every avatar of this if statement, every time this is called your t label num will be a different local variable right, so this consistency will be met.

(Refer Slide Time: 4:44)



The slide is titled "compileWhileStatement() - 2 changes" and features the NPTEL logo in the top right corner. It contains a list of code changes for a while loop implementation:

- whileStatement: 'while' '{' expression '}' '{' statements '}'
- Check for keyword `while`, followed by symbol `{`, followed by `<expression>`
- `tLabelNum = labelNum; Increment LabelNum by 2`
- CODE: `label className.tLabelNum`
- `compileExpression(); //result in ToS`
- Check for `</expression>`, followed by symbol `}`
- CODE:
 - `neg not`
 - `! goto className.tLabelNum+1 //Jump if expression is false`
- Check for symbol `{` followed by `<statements>`
- `compileStatements(); //Code for while loop body`
- Check for `</statements>`, followed by symbol `}`
- CODE
 - `goto className.tLabelNum`
 - `label className.tLabelNum+1`

A video inset in the bottom right corner shows a man with glasses speaking.

Same thing with while, again in while we are use neg that should become not for obvious reason because we are just trying to nega, so will have true, it should become false or false it should become true, so that only not will do that and then again here while we enter this while define an local variable t label num and assign it to label num and increment your label by 2 immediately and everywhere the code use t label num okay, rather than this.


So this will take as while can actually call another while because while statement is a part of statements and when I called compile statements again while will be called and this label num and this label num can be changed because once this compile statements call another

while, if I use label num it can basically change, so I need the same label num across one call of that, while statement and that can recursively call while again and it can change right, so these 2 are the changes.

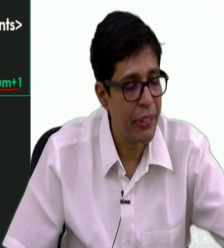
(Refer Slide Time: 5:50)

compileIfStatement() – 2 changes

- IfStatement: `'{' expression '}' statements '}'` statements '}'?
- `LabelNum = labelNum; Increment LabelNum by 2`
- Check for keyword `'{'`, followed by symbol `'{'`
- You must see `<expression>`
- `compileExpression()`; result on TOS
- You must see `</expression>`, followed by symbol `'{'` and then symbol `'{'`
- CODE: `//Global variable labelNum initialized to zero in Main()`
 - `neg not`
 - `if-goto className.LabelNum //TOS is false you jump`
- NOTE: Unique labels generated
- You must see `<statements>`
- `compileStatements()`; `//code for if part`
- You must see `</statements>`, followed by symbol `'{'`



- CODE:
 - `goto className.LabelNum+1`
 - `label className.LabelNum`
- If keyword `else` is there
 - Check for symbol `'{'`
 - Check for `<statements>`
 - `compileStatements()`;
 - Code for else part
 - Check for `</statements>`
 - Check for symbol `'{'`
- CODE:
 - `label className.LabelNum+1`




So just to some of the changes in your if statement we need to make is neg as not and then introduce a local variable called t label num assign it to label num at the start and increment label num by 2 at the start itself and everywhere in the routine make it t label num, wherever you have seen label num in the past make it t label num.


(Refer Slide Time: 6:10)

compileWhileStatement() – 2 changes

- whileStatement: `'while' '{' expression '}' statements '}'`
- Check for keyword `while`, followed by symbol `'{'`, followed by `<expression>`
- `LabelNum = labelNum; Increment LabelNum by 2`
- CODE: `label className.LabelNum`
- `compileExpression()`; `//result in TOS`
- Check for `</expression>`, followed by symbol `'{'`
- CODE:
 - `neg not`
 - `if-goto className.LabelNum+1 //Jump if expression is false`
- Check for symbol `'{'` followed by `<statements>`
- `compileStatements()`; `//Code for while loop body`
- Check for `</statements>`, followed by symbol `'{'`
- CODE
 - `goto className.LabelNum`
 - `label className.LabelNum+1`



- CODE:
 - `goto className.LabelNum`
 - `label className.LabelNum+1`



Similarly same thing you should do in while statements right, again in while statement this neg replace it by not and then t label num is equal to label num, increment label num by 2, again it is a local variable, so every time this while statement or if statement compile or compile if statements is called this t label num will have its fresh avatar, so it will, so again everywhere you see label num replace it by t label num, so these 2 are the simple changes that you need to do for your, whatever you have done in the previous week and now we proceed okay.

(Refer Slide Time: 6:44)

compileDoStatement()

Var Bala G; E-max; Math add

NPTEL

```
doStatement: 'do' subroutineCall ':';
subroutineCall: subroutineName '(' expressionList ')' | (className | varName) ':' subroutineName '(' expressionList ')'
```

- o First check for keyword **do**
- o Check for identifier and call it **id1**
- o Check for symbol ":"
- o if Yes do
- o if No do

- o Check for identifier and call it **id2**
- o Check if **id1** is in local or class symbol table
- o if Yes then **id1** is an object of a class, get its **kind** and **type**.
- o CODE: push kind type
- o if No then **id1** is a className - Array, String, Keyboard etc

- o **id1** is a subroutine of current class
- o CODE: push pointer 0 //Need 'this'

- o Check for symbol (; you should see <expressionList>
- o nP = compileExpressionList();
- o You should see <expressionList> followed by symbol) and ;

- o CODE: call classname.id1 nP+1 //this'
- o pop temp 0

- o if **id1** is in local or class symbol table
- o if Yes then
- o CODE: call Type(id1).id2 nP+1
- o pop temp 0
- o if No then
- o CODE: call id1.id2 nP
- o pop temp 0

compileDoStatement()

Var Bala G; E-max; Math add

NPTEL

```
doStatement: 'do' subroutineCall ':';
subroutineCall: subroutineName '(' expressionList ')' | (className | varName) ':' subroutineName '(' expressionList ')'
```

- o First check for keyword **do**
- o Check for identifier and call it **id1**
- o Check for symbol ":"
- o if Yes do
- o if No do

- o Check for identifier and call it **id2**
- o Check if **id1** is in subroutine or class symbol table
- o if Yes then **id1** is an object of a class, get its **kind** and **type**.
- o CODE: push kind type
- o if No then **id1** is a className - Array, String, Keyboard etc

- o **id1** is a subroutine of current class
- o CODE: push pointer 0 //Need 'this'

- o Check for symbol (; you should see <expressionList>
- o nP = compileExpressionList();
- o You should see <expressionList> followed by symbol) and ;

- o CODE: call classname.id1 nP+1 //this'
- o pop temp 0

- o if **id1** is in local or class symbol table
- o if Yes then
- o CODE: call Type(id1).id2 nP+1
- o pop temp 0
- o if No then
- o CODE: call id1.id2 nP
- o pop temp 0



Module 12.2

JACK Compiler: Code Generation - 6



And then next statement is compile do statement, so what is compile do statement do? This is compile do statement, so the do statement is do followed by a subroutine call and always this subroutine call will be a method okay, so how does the subroutine call local, it will be a subroutine name followed by an expression list, that means it is part of a same class that you are doing right, so I am in a class I can call the subroutine of the same class without anything or I could call a subroutine of a different class like array or match etc, keyboard etc or I could define my own class and that class I could initiate here and I can call that dot subroutine followed by an expression list.

So basically this is how a subroutine name is here okay, so I am just doing the laser pointer here, so the subroutine call will be just subroutine name with an expression list that is going to be part of your same thing, this subroutine name will be part of the same class, so when I am compiling a function within the same class or method or inside that I use the subroutine of the same class or it can belong to some other class, so that time it will be class name.subroutine name or var name.subroutine name.

For example math, for example if I say math.abs than this is a class name followed by a subroutine name, suppose I have said like var say some ball t right and I say and the ball has no t.mov whatever then these T is a var name. Okay, of some class ball, so T his object of type ball, so T is a var name, it points to an object and that .mov okay, so that is this is the thing. Okay, so this is the way we basically look at this subroutine call okay.

So what we do first we need, once we do compile do statement, first we should check for keyword do can check for some identifier and call it ID 1, this ID 1 can be a subroutine name

or a class name or var name right, then you go and check for the symbol dot, if dot is there than your ID name has to adhere to the second route that is class name or var name.something, dot is not there than it is just a subroutine name.

So if the next after you check for the ID 1, if your next is simple is dot if yes then you need to do some action, if no, then do this 3 steps what, so ID 1 is a subroutine of current class itself, so when you are executing, so first I have to do push pointer 0 because I need this, because this will use same class, so I need this, so I say push pointer 0, then I check for the symbol, this thing, then I should see opening parenthesis, I should see expression list, then I do nP equal to compile expression list okay.

So the expression list will tell me how many number of expressions are there right, how many expressions are there in this list, so many arguments are there, we will just see what is compile expression list, so how many arguments out there, so this will give you the number of arguments, then at the end you should see/expression list and then followed by the symbol, this thing and followed by the symbol; right, then you dumb this code class call class name, this is where you use the current class name because it belongs to a current class.

So you have a global variable call class name, which is the current class, so that you use it here class name.ID 1, ID 1 is the first thing that you have seen here, nP +1, nP is the number of arguments in the call right, +1 because you are already pushed this is, so this is basically the number of arguments which will be nP +1 and after you finish this, since this is going to be do statement, this is a subroutine call, but every call as per our VM has to push something on to the stack, so something will be pushed by the subroutine call, the call the subroutine, so I just say pop temp 0, essentially it gets okay, so this is for a subroutine which is belonging to the same class right, so these are the point that you need to.

Now let us go and look at if there is a dot to the second part that is I say I am either getting math.abs or say some t.mov whatever okay, now I will check for the identifier and call it ID 2 right, so I will check whether there is sideway ID 2, so ID 1 will be a class name, now I am seeing the class name or var name, ID 2 will be the subroutine name now right, because I see a dot here, so I am in the second part of this route.

So now check if everyone is in local or class symbol table, in the local, in the subroutine symbol table or in the class symbol table, if yes then ID 1 is an object of a class, for example if you have var ball t or field ball t, then it will be, then surely it is a object of a class okay,

that means it is a var name right, so we will get its kind and type, so we put a code here: push kind type. Okay, right, if no, then ID 1 is a class name. Okay, so because you have to push kind type, this is because I am pushing the this of this because when it is var name, it is a var name, then I for that objects you know local variable should be accessible to the mov right that objects will variable should be accessible to this mov.

So I just say push kind type, this is equivalent to pushing the, so when the call is subroutine the first argument would be that of the objects, so I should, when I say T.mov again when mov is executing it will access the field variables of T correct, we have already explained that and that subroutine should have an access to the T, to the segment, storing the variables of T and that is what we are pushing here, because push kind type will push the address of the starting location of T right.

And if no, then ID 1 is a class name, it will be like array, string, keyboard math.abs, then you come here, then you see, you should see a symbol start parenthesis, left parenthesis you should see expression list, then you come do compile expression list, you call compile expression list and it will return an integer which will be a number of expressions in the expression list and you should see/expression list after that and then this closing symbol and then; this is the last bit of this.

And then you come back, if ID 1 is in local or class symbol table, that is, if ID 1 is a var name than call the type of ID 1.ID 2 nP+1 okay, so this is what you need to call, so I need to call, call ball.mov right because ball.mov and nP +1 because you are having that this of that okay, so if I have var ball T and have T.mov, the code that in the that we put in the VM is call ball.mov with nP+ 1, nP+1 is number of parameters +1 for that this, then again, pop temp 0 if no that means it is some math or array or keyboard then you say call ID 1.ID 2 right.

So math.call, math.abs and nP because here in this case there is no need for pushing the kind type and then pop temp 0, after it returns then you to just to pop temp 0, so this is how we do this subroutine call right, so this is how we compile the do statement right, so carefully go through this once and then if you have any doubts do put up in the discussion group and we will clarify anymore doubts in this direction right, so a small change again here. Okay, this should be reading subroutine or class symbol table. Okay, so this is about compile do statement, now will move to the next module. Thank you.