

Foundations To Computer Systems Design
Professor V.Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras
Module 10.6

The Jack Compiler-Handling Arrays

So welcome to module 10.6 in this module this is the last of the module related to the compiler theory part we will now see how do you handle arrays in the how do we translate array type of statements onto the VM code, so that is what we will be seeing as a part of this module.

(Refer Slide Time: 00:41)



Compilation challenges

- ✓ • Handling variables
- ✓ • Handling expressions
- ✓ • Handling flow of control
- ✓ • Handling objects
- **Handling arrays**
 - Array construction
 - Array manipulation





Module 10.6: The Jack Compiler - Handling Arrays PROF. V. KAMAKOTI
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

So there are two parts of this array one thing is we have to construct an array and then manipulate this array right so let us see how we are going to construct array and how we are going to manipulate array.

(Refer Slide Time: 00:59)

Array construction

```
var Array arr;  
...  
let arr = Array.new(5);  
...
```

Code generation

- var Array arr:
generates no code;
only effects the symbol table
- let arr = Array.new(n):
from the caller's perspective,
handled exactly like object construction

stack

host RAM

SP 0

LCL 1

ARG 2

THIS 3

THAT 4

...

8054 local 0 (arr)

...

8054

8055

8056

8057



8058

PROF. KAMAKOTI

...

heap

Module 10.6: The Jack Compiler - Handling Arrays
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright



For example we have `var array arr let arr = array.new(5)` so what essentially needs to happen is that `arr` is say let us say it is a local variable 0 so now `arr = array.new(5)` should initially the value of `arr` will be 0 but once you do this `array.new(5)` then this value of so the compiler should the machine should allocate 5 locations.

Let us say 8054 to 8058 and then update this value of `arr` by 8058 this is what it is supposed to do right so now we will understand from the callee perspective and the caller perspective what is going to happen in this case.

(Refer Slide Time: 01:53)

NPTEL

host RAM

SP 0
LCL 1
ARG 2
THIS 3 8058
THAT 4 8054
...
...
8054 local 0 (arr)
8058 local 1 (point)
...
...
8054 that
8055 0
8056 1
8056 2
...
8058 this
8059 0
...
8059 1
...

VM use: this that
used to represent the values of the current *object* used to represent the values of the current *array*

pointer (base address) THIS THAT

how to set: pop pointer 0 (sets THIS) pop pointer 1 (sets THAT)

stack
heap

Module 10.6: The Jack Compiler - Handling Arrays
Nand to Tetris / www.nand2tetris.org / Chapter 11

PRINCE V. KAMARAJOTTI
111 Madras
nd Shimon Schocken

So there are two virtual memory segments that can be aligned to different RAM first addresses first let us remind this, this and that right so this now points to 8058 that now points to say 8054 here right so let us say that has the locations 8054, 8055, 8056 so that 0 is that, 0 is 8054 that 1 is 8055 that 2 is 8056 similarly let us say this has 8058, 8059.

Now if I want to change this 8058 value I will say pop I put whatever value onto the stack and pop pointer 0 so suppose I say push 10 and I say pop pointer 0 then this 3 will become 10 and similarly if I say push 100 and say pop pointer 1 then this 4 will become 100 so I can change the address to which this and that are pointing namely I can change the content of 3 and content of 4 by popping to pointer 0 and pop pointer 1 respectively this we have seen it is just a recap what we have done.

(Refer Slide Time: 03:08)

Array access

```
// let arr[2] = 17
push arr // base address
push 2 // offset
add
pop pointer 1
push 17
pop that 0
```

Module 10.6: The Jack Compiler - Handling Arrays
Nand to Tetris / www.nand2tetrtris.org / Chapter 11 / Copyright

Suppose I have say array of 8056 is 17 say RAM of 8056 what should I do? I always use this that pointer to manipulate this because this will be always used for pointing to the current object so that is now will be now used so when I was describing this virtual machine I said this that do not bother now later I will explain why we are using this and that, now is the instance where I am going to explain you what is that, this we have already explained now that I am going to explain now. So when I say RAM of 8056 is equal to 17 what I do is push 8056 so 8056 goes into the stack. Now I say pop pointer 1 so 8056 will go to this location 4 it will adjust now I want push 17 so I am pushing 17 so what will happen?

So somewhere this 17 will get into the top of the stack ok then I say pop that 0 so 8056 17 will now get stored right so the way I execute RAM of 8056 is equal to 17 is I push 8056 pop pointer there so now that becomes 8056 now I push 17 onto the stack and say pop that 0 so 17 will be there. So this is the equivalent VM code for this jack level code which is RAM of 8056 is equal to 17 right. So now what has happened let me just again explain you once when I say push 8056 somewhere 8056 gets into the stack pop pointer 1 8056 goes and stays now in that. Now 8056 is this pointer to the start of the that segment that is what it means.

Now I say push 17, 17 goes onto the top of the stack and then say pop that 0 that means 8056 plus 0 which is 8056 gets 17 so essentially I have made RAM of 8056 as 17 ok there is small here this need not be the top of the stack so this arrow basically talks about var 8056. Right so if I say

let arr 2 is equal to 17 so how do I do this? First push arr ARR push 2 add when I say push arr what am I pushing? I am pushing the base address of arr, when I say add 2 I add 2 to the base address of arr so that will so that add will now be the address in which I want to store 17.

Suppose arr is stored at 8056 then 8058 is where I need to store 17 arr base address is 8056, 8058 is where I am going to do something so when I say push arr what will go into the stack? 80 sorry if 8054 let us say arr starts at 8054 if I push arr 8054 goes into that stack then I push 2 then I get 2 goes into the stack now I add then I get 8056 on top of the stack. I pop that to pointer 1 so 8056 goes to that. Now I push 17 and I say pop that 0 so 8056 basically gets 17 right. So this is how I use the that segment to handle manipulate arrays.

(Refer Slide Time: 07:06)

The slide is titled "Array access" and features the NPTEL logo in the top right corner. On the left, a code block contains the following assembly instructions:

```
// let arr[expression1] = expression2
push arr
push expression1
add
pop pointer 1
push expression2
pop that 0
```

To the right of the code is a memory diagram labeled "host RAM". It shows a vertical stack of memory cells. The stack grows downwards from higher addresses to lower addresses. The stack pointer (SP) is at address 0. Below it are local variables (LCL) at address 1, arguments (ARG) at address 2, this pointer (THIS) at address 3, and that pointer (THAT) at address 4. Below these are several empty cells, followed by a cell at address 8054 labeled "local 0 (arr)". Below the stack is the heap, which grows upwards from lower addresses to higher addresses. The heap starts at address 8054 and contains cells at addresses 8054, 8055, 8056, 8057, and 8058. Below the heap is the text "PROF. N. KAMAROTI".

At the bottom left, there is a cartoon character of a red devil with horns and a speech bubble that says "unfortunately, there's a problem". Below the character is the text "Module 10.6: The Jack Compiler - Handling Arrays" and "Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright".

In the bottom right corner, there is a small video inset showing a man with glasses and a white shirt, likely the instructor.

Now when we do this then there is an issue right let arr expression 1 is equal to expression 2, suppose I say push arr push expression 1 right what will push expression 1 do? It will basically expression 1 will evaluate something and that output of compiled expression suppose I do a compilation of the expression that code will evaluate that expression for example arr of a plus arr of 5 plus 6 so that will become arr of 11 right so 11 will be there when I add this let us say arr is 8054 starts at 8054 let expression 1 be say 4 so 50 58 will be there. Now when I pop pointer 1 what will happen? That will become 8058.

Now I push expression 2 right so I compile expression 2 so this will give an answer that will be on the top of the stack so when I compile and execute an expression what I assume is the

expressions output will be on the top of the stack after it completes. Now when I say pop that 0 ofcourse it will go to this 8058 but what will happen is when I am evaluating expression 2 I may go an change this that see what we have done is array of expression 1 let us say arr of 5 plus 6 so 5 plus 6 is 11 arr starts at 8054, 8065 is where I want to store the evaluated value of expression 2 that 8065 I have stored at that I should have stored it here in that, that is what happens at the end of this 4 things ok.

But then I am evaluating expression 2 that expression 2 can now go and touch that and if it touch that 10 this value essentially will get replaced this whole thing will so expression 2 completes and you get some value when I say pop that 0 that has what? 8065, 8065 will get that value of that expression provided this expression 2 does not touch that, the expression 2 itself can be another array expression for example let a of 10 be equal to c of 15 right now that c of will go and touch that so that so what will happen is this particular code will not work correctly if this expression 2 touches that ok.

(Refer Slide Time: 10:03)

Array access

the problem, illustrated


```

// let a[i] = b[j]
push a
push i
add
pop pointer 1

// now handle the right hand side
push b
push j
add
pop pointer 1

```

- Our compilation strategy is generating code on the fly, left to right
- The term within the square brackets is treated as an expression, and compiled as such.




There are many other scenarios that will cause similar problems, e.g.

```
let a[a[i]+a[j+a[3]]] = a[j+2]
```

Module 10.6: The Jack Compiler - Handling Arrays
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTTI
IIT Madras



So for example as I told you let a i equal to b j push a push i add pop pointer 1 push b push j add pop pointer 1 that is all your pointer 1 is gone.

So if I have array on both sides a i equal to b j then this array this manipulation will not work right. Now we can give all this in as a of i plus a of j plus a of 3 equal to a j plus 2 you go mad

right so if I give something like so point is when we have this type of an expression for how do we do this right.

(Refer Slide Time: 10:46)

So this is what we do so push a push i add so this will give me the address in which I have to store b j right that will be on the top of the stack. So now let us look at this, at the end of this add what will be on top of the stack? The address in which I have to store b j similarly push b push j add this will tell you the address of b j pop pointer 1 and push that 0 that means it will get you the value of b j right.

So that will be on the value of 0, this value of b j I pop it to at temporary location pop temp 0 now what will be on top of the stack? This a i will be on the top of a plus i where I need to store that value at that I pop pointer 1 then I say push temp 0 then I say pop that 0 right. So this is some this is the thing that we need to do in general ok. So I will just try and explain this a bit for you so that you know we have little more clarity on this otherwise going to do so very simple a of 5 is equal to (a of) b of 8 let me say a is stored at 5000 b is stored at 7000 so essentially means content of 7008 should move to 5005.

So this is the stack and somewhere this is let us say this is that so let me say this is stack that is growing now I say push a when I say push a what will happen? 5000 goes into the stack push 5 I said push i right so 5 so 5 goes into the stack I say add when we say add this will become, this will become 5005 right now I say 5005 is push b so 7000 pushes push 8, 8 pushes add I get 7008

now I say pop pointer 1, when I say pop pointer 1 essentially that will this will get removed and basically 7008 will go to the that segment, that is what pop pointer 1.

Now in 7008 let us extend this memory suppose in 7008 we have 15 now I say push that 0 go to the that segment add 0 to it 7008 whatever value push it back to the stack so stack will now get 15, 15 will now move ok. Now pop temp 0, temp is a segment from 6 array to 15 so there is a temp segment so somewhere let us say there is a temp segment here so the temp segment this will now pop so 15 will now go and sit here temp 0, this is temp 0. Now this is the top of the stack now I say pop pointer 1 so essentially when I say pop pointer 1 again this is temp pop temp 0 then I say again pop pointer 1 ok so when I say pop pointer 1 what will happen?

Now this will go off and this will go to this so 5005 will get onto that ok 5005 will go onto this now push temp 0 so this 15 will now get pushed onto this push temp 0 now pop that 0, pop that 0 means now this 15 will go now and reside in 5005 so 15 now goes to 5005 so the content of 7008 which was 15 now gets copied right. So this is how we basically work on this particular array right.

(Refer Slide Time: 16:19)

Array access

General solution for generating array access code


```


// let arr[expression1] = expression2
push arr
VM code for computing and pushing the value of expression1
add // top stack value = RAM address of arr[expression1]
VM code for computing and pushing the value of expression2
pop temp 0 // temp 0 = the value of expression2
pop pointer 1
push temp 0
pop that 0
    
```

If needed, the evaluation of expression₂ can set and use pointer 1 and that 0 safely

What about handling, say, `let a[a[i]] = a[b[a[b[j]]]]` ?

No problem...





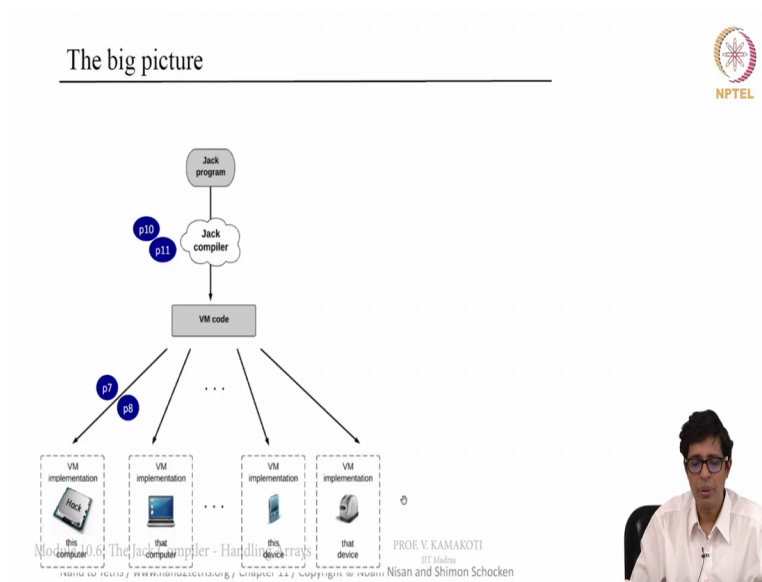
Module 10.6: The Jack Compiler - Handling Arrays PROF. V. KAMAKOTI
 Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

So in general solution for generating array access code for example let array expression 1 is equal to expression 2 so what we do is that we push array then we put the VM code for expression 1 then we add so the VM code for expression 1 will give the value of that expression 1 there on the top of the stack will add so that is the at the end of this what you see at the end of

this address is the which address I need to store the evaluated value of expression 2 that is will be on the top of the stack.

Now I do VM code for computing expression 2 then that answer I store it in temp 0 now I pop pointer 1 which will be this address and push the value of temp 0 again back to the stack and pop it onto that 0 so essentially this gets, so this is the general way of handling let expression is equal to expression 2 so please note that a of i a of i will work very nicely so if you just code this, this recursively works very nicely ok.

(Refer Slide Time: 17:32)



So the big picture to conclude is that there is a jack program now we have done project 10 the current project 11 we have to do that will give you the VM code and that VM code using project 7 and project 8 you will write it into a VM implementation one of this but the same thing we can write for different other devices different VM code and this will basically taken.

(Refer Slide Time: 18:02)

The big picture

The diagram illustrates the compilation process. At the top, three boxes labeled 'Java program', 'Jack program', and 'C# program' are connected by arrows to three boxes labeled 'Java compiler', 'Jack compiler', and 'C# compiler'. These three compiler boxes all point to a central box labeled 'VM code'. To the right of the 'VM code' box, there is a box labeled 'classes, functions, constructors, methods, arrays, objects, ...'. An arrow labeled 'map on:' points from this box to a stack structure. The stack is represented as a vertical column of boxes, with the top box labeled '0' and the bottom box labeled '3'. An arrow labeled 'push' points from the stack to the right, and an arrow labeled 'pop' points from the right to the stack. To the right of the stack is a series of boxes representing 'memory segments', with the top box labeled '0' and the bottom box labeled '3'. An arrow labeled 'map on:' points from the stack to these memory segments. The NPTEL logo is in the top right corner.

Standard Mapping on the VM platform:
Specifies how to map the constructs of a given high-level language on the constructs of the virtual machine

Module 10.6: The Jack Compiler - Handling Arrays
Nand to Tetris / www.nand2tetris.org / Chapter 11 / Copyright © Noam Nisan and Shimon Schocken

PROF. V. KAMAKOTI
IIIT Madras

So this is how the whole thing works and this is the big picture here ok and so we will now go to module 11 where in the next we will start looking at the project 11 in greater detail.