Virtual Machine II

Tom Kelliher, CS 220

Chapter 8, Textbook

- Section 8.1: What background "bookkeeping" must take place when a subroutine is called and when a subroutine returns?

  Why is this bookkeeping necessary? Be specific; cite examples.

  Consider a subroutine, `foo()`, that's called from two other subroutines, `bar()` and `fubar()`. How can `foo()` determine if it should return to `bar()` or to `fubar()` after it completes its execution?

  Conceptually, is there any limit on the number of invocations of a recursive function?

  Consider a subroutine, `foo()`, that is recursive, with several parameters and local variables. How can `foo()` manage this separate data for all of its active invocations?

  What is a frame?

  Why is a stack a good data structure for managing subroutine invocations?

  Does the Hack virtual machine use separate stacks for subroutine frames and for virtual machine instructions such as `push`, `pop`, `add`, and so on? If a single stack is used, how are subroutine frames and virtual machine instruction data distinguished?

  How are virtual machine instructions such as `add` and subroutines with parameters and that return a value similar? For example, consider the subroutine `pow(a, b)` that returns `a` raised to the `b` power.

  Here's an example program that computes an element from the Fibonacci sequence.

  `Sys.vm`:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/08/FunctionCalls/FibonacciElement/Sys.vm

// Pushes n onto the stack and calls the Main.fibonacii function,
// which computes the n'th element of the Fibonacci series.
// The Sys.init function is called "automatically" by the
// bootstrap code.

function Sys.init 0
push constant 4
```

```
call Main.fibonacci 1   // Compute the 4'th fibonacci element
label WHILE
goto WHILE              // Loop infinitely
```

Main.vm:

```
// This file is part of www.nand2tetris.org
// and the book "The Elements of Computing Systems"
// by Nisan and Schocken, MIT Press.
// File name: projects/08/FunctionCalls/FibonacciElement/Main.vm

// Computes the n'th element of the Fibonacci series, recursively.
// n is given in argument[0].  Called by the Sys.init function
// (part of the Sys.vm file), which also pushes the argument[0]
// parameter before this code starts running.

function Main.fibonacci 0
push argument 0
push constant 2
lt                     // check if n < 2
if-goto IF_TRUE
goto IF_FALSE
label IF_TRUE          // if n<2, return n
push argument 0
return
label IF_FALSE         // if n>=2, return fib(n-2)+fib(n-1)
push argument 0
push constant 2
sub
call Main.fibonacci 1  // compute fib(n-2)
push argument 0
push constant 1
sub
call Main.fibonacci 1  // compute fib(n-1)
add                    // return fib(n-1) + fib(n-2)
return
```

Chapter 8, Slides

- Slide 4: Assume that the three virtual machine instructions occur in the subroutine `foo`. Translate each of them to Hack assembly.
- Slide 7: Why are `nVars` and `nArgs` needed here? (See slides 12 and 13.)
- Slide 8: What is the contract between caller and callee?
- Slide 10: What are the virtual machine's responsibilities during subroutine call and return?

- Slide 11: What are the components of a subroutine's frame?
- Slide 12: Implement `call`'s translation on paper. How do we generate `returnAddress`, given that we need many such return addresses?
- Slide 13: Implement `function`'s translation on paper.
- Slide 14: Implement `return`'s translation on paper.
- Slide 15: Implement the bootstrap on paper. How many arguments has `Sys.init` pushed onto the stack?
- Slide 17: Note that `writeInit` isn't needed for all of the project test cases. Refer to the project handout for details.