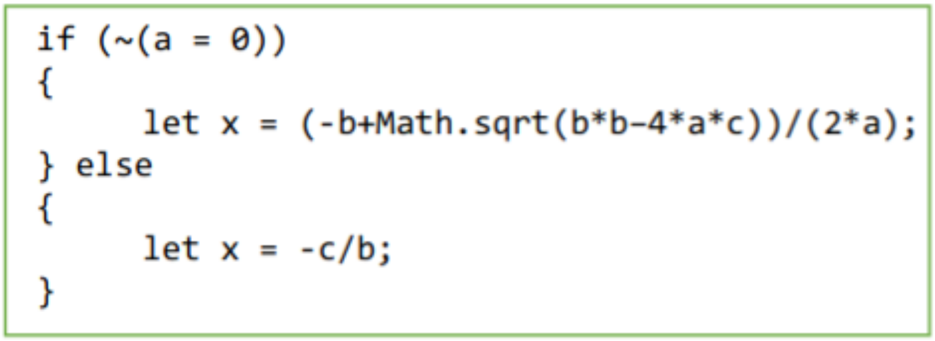
PASS - Computer Systems

Week 8

1. Flow control(branches and loops)
   1) What are the three main commands used in VM code which determine the flow of a program?

VM command	Description
label	serves as a point to goto
goto	unconditional jump
if-goto	if top most value of stack is true, jump

2. How would you write JACK source code in VM(pseudo) code?



```
if (~(a = 0))
{
      let x = (-b+Math.sqrt(b*b-4*a*c))/(2*a);
} else
{
      let x = -c/b;
}
```

```
push a
push constant 0
eq
if-goto ELSE
(TRUE)
  push b
  push b
  push 4
  push a
  push c
  call m.m 2
  call m.m 2
sub
call m.m 2
push constant 2
push a
call m.m 2
call m.d 2
neg
pop x
goto END
(ELSE)
  push c
  neg
push b
call m.d 2
pop x
(END)
goto -END
```

3. Flow control (functions)
    1) What is meant by a subroutine in VM language? What are the issues associated with implementing this?

Any kind of function. ?? - only 1 stack (only return 1 value)

    2) VM syntax

Define a function	function foo.bar nvars
Call a function	call foo.bar nargs
Function return	return

3) Implement VM for function sub (assume this function is in class XX)

function int sub(int a, int b) { return a - b; }

```
function XX.sub 0
  push arg 0
  push arg 1
  sub
  return
```

4) Implement a ***void*** function (assume this function is in class XX)

function void hello(String hi) { do Unix.print(hi); }

```
function XX.hello  0
   push argument 0
   call Unix.print 1
   pop temp 0 -> put unix.print value in temp
   push constant 0 -> dummy value
   return
```

4. VM implementation(VM ➔ Assembly. The black magic behind the scenes!)

Translate VM to assembly (assume static 3 is a static variable in class BOB)

pop static 3

```
@SP
AM=M-1
D=M
@BOB.3
M=D
```

sub

Line 1	@SP
Line 2	AM= M-1
Line 3	D = M

Line 4	A = A - 1
Line 5	M = M - D

5. Describe the processes for implementing the following and what happens behind the scenes

   1) Implement Calling a function:

1. arguments for function are pushed onto the stack
2. function is called
3. address of the next instruction that caller executes is pushed onto stack
4. Caller memory segments are pushed onto the stack - pointers to : LCL, ARG, THIS, THAT
5. Pointers to ARG and LCL for the callee are set.
6. Control is transferred to callee.

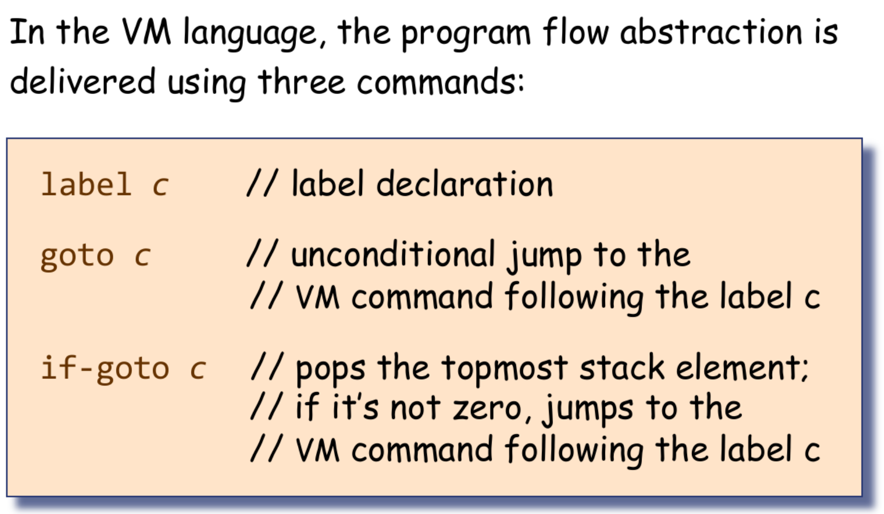
   2) Implement running a function:

1. push nVars onto the stack and initialise to 0
2. execute function commands.

3) Implement Return:

1. replace first arg of callee with result.
   * ARG 0 = pop()
2. set SP to be SP+1 (this essentially recycles memory used by callee)
3. restore memory segments of caller.
4. transfer control back to caller by jumping to return address

Lecture slides for reference

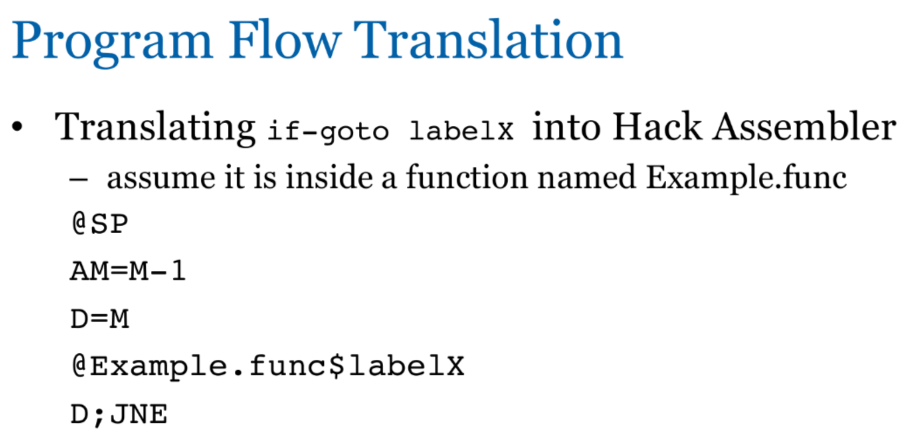


In the VM language, the program flow abstraction is delivered using three commands:

```
label c       // label declaration

goto c        // unconditional jump to the
              // VM command following the label c

if-goto c     // pops the topmost stack element;
              // if it's not zero, jumps to the
              // VM command following the label c
```



Program Flow Translation

- Translating `if-goto labelX` into Hack Assembler
  - assume it is inside a function named Example.func

```
@SP
AM=M-1
D=M
@Example.func$labelX
D;JNE
```

Function commands in the VM language

```
function g nVars   // here starts a function called g,
                   // which has nVars local variables

call g nArgs       // invoke function g for its effect;
                   // nArgs arguments have already been pushed onto the stack

return             // terminate execution and return control to the caller
```

VM Programming Examples in Class XX

```
function int add(int x,int y) { return x + y ; }

   function XX.add 0   no local variable
   push argument 0
   push argument 1
   add
   return

function void hello(String hi) { do Unix.print(hi) ; }

   function XX.hello 0
   push argument 0
   call Unix.print 1   number of arguments
                                            pop temp 0
   push constant 0
   return
```