week 6 Pass

2. What data structure is used for the implementation of the VM model?

Stack

3. Our VM model features a single 16-bit data type that can be used as:

16 bit two's complement integer -
Boolean Value : -1 for true, 0 false
Pointer : this, that

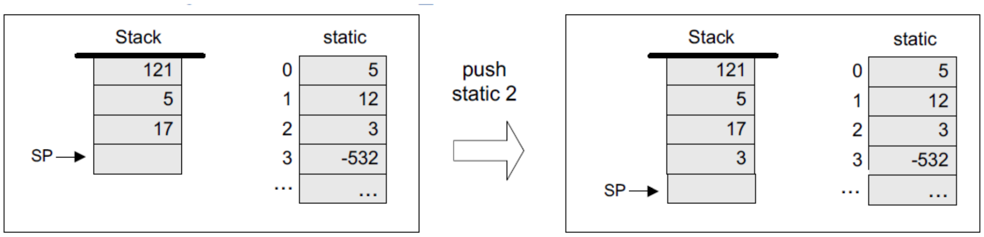
4. Memory segments:

constant local	(virtual)	pointer temp
argument this		
that static		

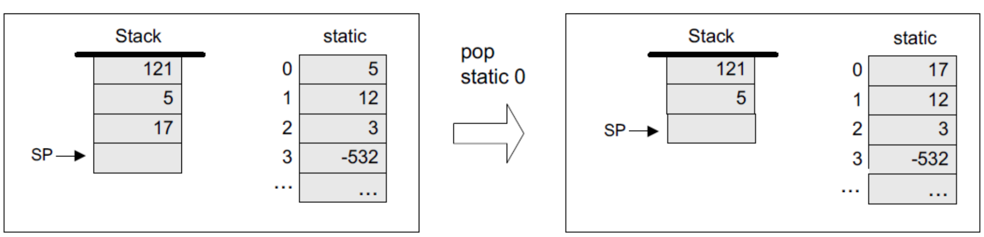
5. VM Operations

Arithmetic/boolean commands	add, sub, neg, lt, gt, eq, and, or, not
Memory access commands	push, pop
Program flow commands	goto, if-goto, label
Function calling commands	function, call, return

6. Example: draw the Stack and the Static Segments After each of the following operations:



(before) (after)



a. Following the example above, complete the stack after the following occur:
At the beginning the stack looks like this:

STACK
12
8
4
4
SP

After the operation add:

STACK
12
8
8
SP

After the operation eq:

STACK
12
-1
SP

After the operation or:

STACK
-1
SP

After the operation not:

STACK
0
SP

b. Complete the stack after the following occur:

At the beginning the stack looks like this:

STACK
28
123
4
890
SP

After the operation neg:

STACK
28
123
4
-890
SP

After the operation sub:

STACK
28
123
894
SP

After the operation gt:

STACK
28
0
SP

7. Implement the following in VM language:

   (x,y,z refer to static 0, 1, 2, respectively)

True and false	push constant 1 neg push constant 0 and
(x+y)*z	push static 0 push static 1 add push static 2 call math.multiply 2
(-y) or (x and z)	push static 1 neg push static 0 push static 2 and or
(4 + a) *(c-9) (Assume a is static 0 and c is static 1)	push constant 4 push static 0 add push static 1 push constant 9 sub call math.multiply 2

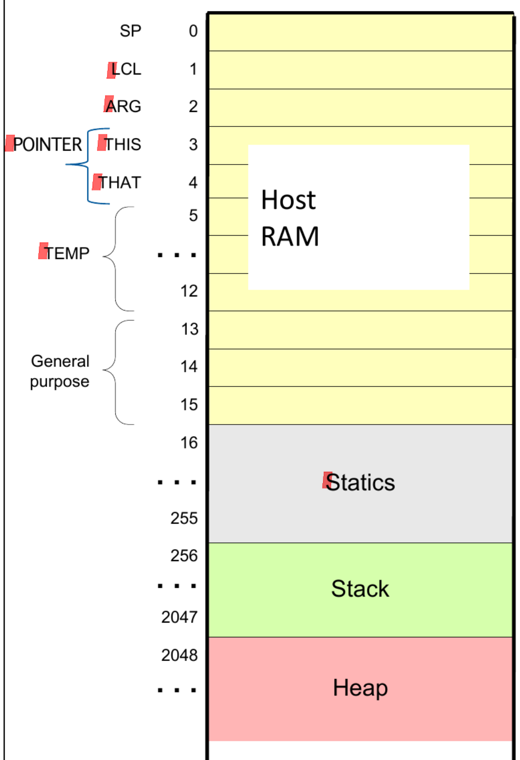
8. Convert the following Virtual Machine code to Assembly

Push constant 7	@7 D=A @SP AM = M+1 A = A-1 M = D
Push argument 3	@3 D = A @arg A = M+D D = M @SP AM = M+1 A = A-1 M = D
add	@SP A = M-1 D = M A = A-1 M = M+D @SP AM = M-1

Lecture slides for references

VM implementation on the Hack platform:

VM implementation on the Hack platform



Basic idea: the mapping of the stack and the global segments on the RAM is easy (fixed); the mapping of the function-level segments is dynamic, using pointers

The stack: mapped on RAM[256 ... 2047]; The stack pointer is kept in RAM address SP

static: mapped on RAM[16 ... 255]; each segment reference static i appearing in a VM file named f is compiled to the assembly language symbol f.i (recall that the assembler further maps such symbols to the RAM, from address 16 onward)

local,argument,this,that: these method-level segments are mapped somewhere from address 256 onward, on the "stack" or the "heap". The base addresses of these segments are kept in RAM addresses LCL, ARG, THIS, and THAT. Access to the i-th entry of any of these segments is implemented by accessing RAM[segmentBase + i]

constant: a truly virtual segment: access to constant i is implemented by supplying the constant i.

pointer: RAM[3..4] to change THIS and THAT.

Memory access VM commands:

- pop *memorySegment index* **pop: take the top item off the stack and write it to the memorySegment index.**
- push *memorySegment index*

Where *memorySegment* is static, this, local, argument, that, constant, pointer, or temp

And *index* is a non-negative integer