Name:
Date:

Source: VM language

Arithmetic / Logical commands

```
add
sub
neg
eq
gt
lt
and
or
not
```

Branching commands

```
label label
goto label
if-goto label
```

Function commands

```
function functionName nVars
call functionName nArgs
return
```

Memory access commands

```
pop segment i
push segment i
```

1. Translating High-Level to VM Code: Translate the following high-level Jack/Java statements into its virtual machine equivalent. Do this in two steps: 1) Write the pseudocode with symbol names, then 2) Write the VM code (no symbols, use memory segments: e.g. local, argument, etc)

```
int greatest;
int x = 3;
int y = 5;

if (x > y)
    greatest = x;
else
    greatest = y;
```

Pseudo VM Code	VM Code

// pseudo
 	
push 0 onto the stack
pop the value from stack into greatest
push 3 onto stack
pop value from stack to x
push 5 onto stack
pop value from stack onto y
push value of x onto stack
push value of y onto stack
compare x and y if its less than
if true than jump to if_true
push value of y onto stack
pop value from stack into greatest
jump to end
label true block
push value of x
pop value from stack into greatest
end of if else
	// VM
	
	// Initialization
	push constant 0
	pop local 0
	push constant 3
	pop local 1
	push constant 5
	pop local 2
	push local 1
	push local 2
	lt
	if-goto TRUE
	// Else Block
	push local 2
	pop local 0
	goto END
	label TRUE
	push local 1
	pop local 0
	label END

2.

```
int sum = 0;
int n = 5;
for (int i = 1; i <= n; i++)
    sum += i;
```

Pseudo VM Code	VM Code

// Pseudo
int sum = 0; // Declare and initialize 'sum' with 0
int n = 5; // Declare and initialize 'n' with 5
 	
for (int i = 1; i <= n; i++) {
sum += 1; // Increment 'sum' by 1 in each iteration
}
	// VM
	
	push 0
	pop local 0
	push 5
	pop local 1
	push constant 1
	pop local 2
	LOOP_START
	push local 2
	push local 1
	gt
	not
	if-goto END_LOOP
	push local 0
	push constant 1
	add
	pop local 0
	sum
	push local 2
	push constant 1
	add
	pop local 2
	goto LOOP_START
	END_LOOP

3.

```
// Multiplies x * y
// (by summing x, y times)
int x = 2;
int y = 5;
int sum = 0;
int n = 1;
while (n <= y)
{
    sum += x;
    n++;
}
```

Pseudo VM Code	VM Code

// pseudo
int x = 2;
int y = 5;
int sum = 0;
int n = 1;
while (n <= y) {
sum += x;
n++;
}
	// VM
	
	push constant 2
	pop local 0
	push constant 5
	pop local 1
	push constant 0
	pop local 2
	push constant 1
	pop local 3
	push local 3 // push n onto the stack
	push local 1 // push y onto the stack
	lt // check if n < y (less than)
	not // negate the result to get n <= y
	if-goto WHILE_END
	push local 2 // push sum onto the stack
	push local 0 // push x onto the stack
	add // add x to sum
	pop local 2 // store the result back in sum
	// n++;
	push local 3 // push n onto the stack
	push constant 1
	add // increment n by 1
	pop local 3 // store the result back in n
	goto WHILE_EXP
	label WHILE_END

4. Now it's time to translate functions!

```
int greatestOfTwo(int x, int y)
{
    int greatest;
    if (x > y)
        greatest = x;
    else
        greatest = y;
    return greatest;
}
```

Pseudo VM Code	VM Code

//pseudocode
// Multiplies x * y (by summing x, y times)
 	
int x = 2;
int y = 5;
int sum = 0;
int n = 1;
while (n <= y) {
sum += x; n++;
}
	// VM
	
	push constant 2
	pop local 0
	push constant 5
	pop local 1
	push constant 0
	pop local 2
	push constant 1
	pop local 3
	label WHILE_EXP
	push local 3
	push local 1
	lt
	if-goto WHILE_END
	push local 2
	push local 0
	add
	pop local 2
	push local 3
	push constant 1 add
	pop local 3
	goto WHILE_EXP
	label WHILE_END

5. A tad more complex.

```
int mult(int x, int y)
{
    int sum = 0;
    int n = 1;
    while (n <= y)
    {
        sum += x;
        n++;
    }
    return sum;
}
```

Pseudo VM Code	VM Code

`// pseudo`
 	
`function mult(x, y):`
`  sum=0`
`  n=1`
`  while n <= y:`
`  sum=sum +x`
`  return sum`
	`// VM`
	
	`function mult 2`
	`push constant 0`
	`pop local 0`
	`push constant 1`
	`pop local 1`
	`label START_LOOP`
	`push local 1`
	`push argument 1`
	`lt`
	`if-gt END_LOOP`
	`push local 0`
	`push argument 0`
	`add`
	`pop local 0`
	`push local 1`
	`push constant 1`
	`add`
	`pop local 1`
	`goto START_LOOP`
	`label END_LOOP`
	`push local 0`
	`return`

6. The Greatest!

```
int greatestOfThree(int x, int y, int z)
{
    int greatest;
    if (x > y)
    {
        if (x > z)
            greatest = x;
        else
            greatest = z;
    }
    else if (y > z)
        greatest = y;
    else
        greatest = z;
    return greatest;
}
```

Pseudo VM Code	VM Code

// pseudo
 	
function greatestOfThree(x,y,z):
  greatest = 0
  if x > y:
  if x > z:
  greatest = x
  else:
  greatest = z
  else:
  if y > z:
  greatest = y
  else:
  greatest = z
  return greatest
	// VM
	
	function greatestOfThree 3
	push constant 0
	pop local 0
	push argument 0
	push argument 1
	lt
	if-gt IF_X_GREATER
	push argument 1
	push argument 2
	lt
	if-gt IF_Y_GREATER
	push argument 2
	pop local 9
	goto END
	label IF_X_GREATER
	push argument 0
	push argument 2
	lt
	if-gt IF_X_GREATER_OR_Z
	push argument 0
	pop local 0
	goto END
	label IF_X_GREATER_OR_Z
	push argument 2
	pop local 0
	goto END
	label IF_Y_GREATER
	push argument 1
	push argument 2
	lt
	if-gt IF_Y_GREATER_OR_Z
	push argument 2
	pop local 0
	label END
	push local 0
	return

Summative Questions:

7. The execution of which of the following VM command(s) changes the state of the stack?

   a) goto

   b) if-goto

   c) label

   d) None of these commands changes the state of the stack.

8. Consider the following pseudo VM code:

```
1  push 8
2  push 7
3  push 10
4  push 5
5  sub
6  call Math.multiply
```

   Suppose that we begin with an empty stack. What value will be at the top of the stack following the execution of the above code?

```
top stack value = 25
```

9. In the VM language, functions are **declared** using the VM command "function functionName *n*". The integer *n* stands for the number of this function's:

   a) local variables

   b) arguments

   c) local and static variables

   d) static variables

10. In the VM language, functions are **declared** using the VM command "function functionName *n*". The integer *n* stands for the number of this function's:

    a) local variables

    b) arguments

    c) local and static variables

    d) static variables