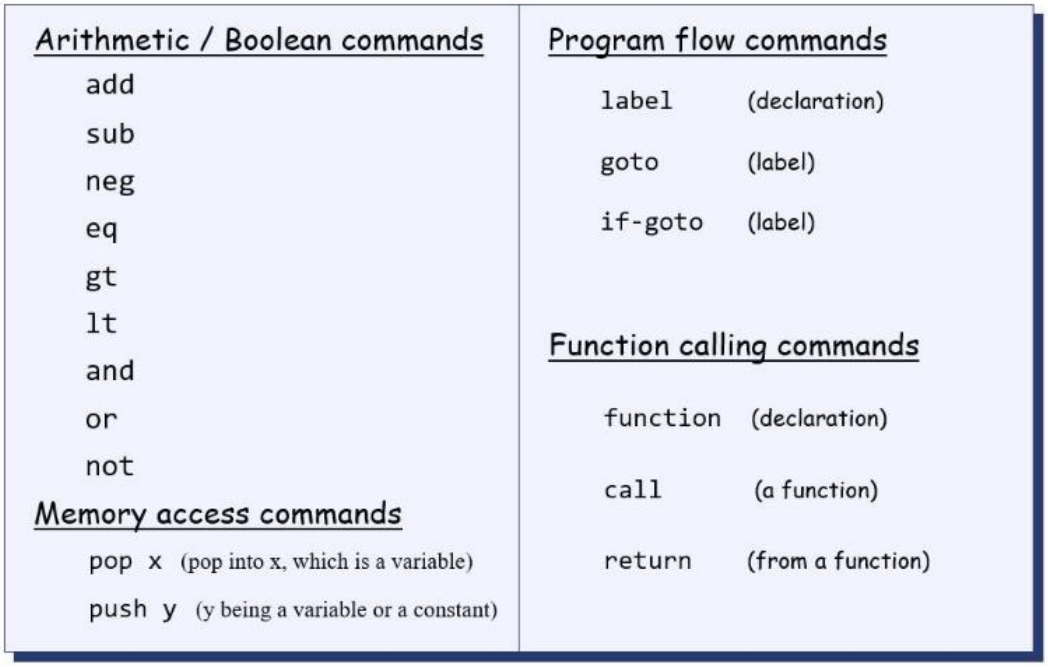
CSCE-312 QUIZ 7 [20 POINTS]

CSCE-312 | DUE: TUESDAY NOV 29, 2016 11:59PM ON E-CAMPUS

NAME: Matthew Stevens **UIN**: 924000693



Arithmetic / Boolean commands

add

sub

neg

eq

gt

lt

and

or

not

Memory access commands

pop x (pop into x, which is a variable)

push y (y being a variable or a constant)

Program flow commands

label (declaration)

goto (label)

if-goto (label)

Function calling commands

function (declaration)

call (a function)

return (from a function)

Question 1. [3 points] Write pseudo VM code for the expression z = x+y using stack arithmetic. You may assume x, y, z are stored in consecutive memory locations. Pseudo VM code follows VM syntax as shown above but does not list specific memory segments like static, temp, argument, etc.

push x

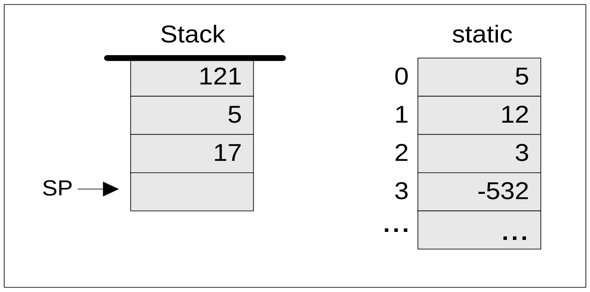
push y

add

pop z

Question 2. [4 points] For the picture below, draw the final picture of the stack and static segments after execution of the following command sequence:

```
push static 3
push static 0
add
pop static 1
```



Stack **Static**

121		**5**
5		**-527**
17		**3**
Stack pointer		**-532**

Question 3. [5 points] Write pseudo VM code (stack arithmetic, memory, control, and functions) for the following high-level code. Assume that divide rounds down to an integer (for e.g. 8/3 returns 2). In your VM code you will need to write divide and multiply functions and call them from the main program.

```
if (~ (a = 0))
        x = b/c
else
        x = b*c
```

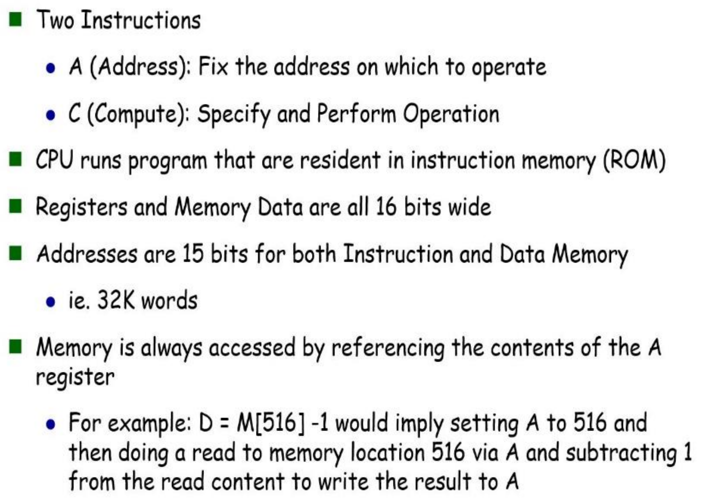
```
//MULT function

function mult 2

        push constant 0

        pop local 0

        push argument 1

        pop local 1
```

```
  label loop
          push local 1
          push constant 0
          eq
          if-goto end
          push local 0
          push argument 0
          add
          pop local 0
          push local 1
          push constant 1
          sub
          pop local 1
          goto loop
  label end
          push local 0
          return

//DIVIDE FUNCTION
function div 2
          push constant 0
          pop local 0
          push argument 1
          pop local 1
  label loop
          push local 1
          push constant 0
          eq
          if-goto end
          push local 0
          push argument 0
          sub
          pop local 0
          push local 1
          push constant 1
          sub
          pop local 1
          goto loop
   label end
          push local 0
          return
```

```
push a
push 0
eq
not
if-goto divide
push b
push c
mult
goto end
(divide)
```

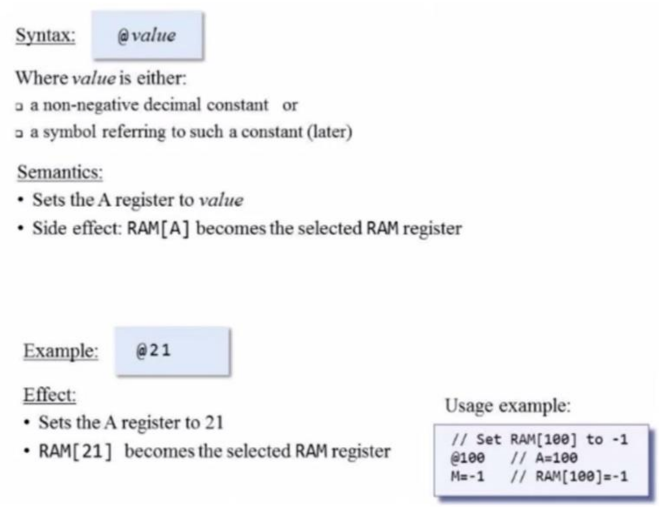
```
push b

push c

div

(end)

pop x
```

Here is a reference for HACK assembly language syntax that we practiced in this course. All details are given below for references and then the questions follow.



- Two Instructions
  - A (Address): Fix the address on which to operate
  - C (Compute): Specify and Perform Operation
- CPU runs program that are resident in instruction memory (ROM)
- Registers and Memory Data are all 16 bits wide
- Addresses are 15 bits for both Instruction and Data Memory
  - ie. 32K words
- Memory is always accessed by referencing the contents of the A register
  - For example: D = M[516] -1 would imply setting A to 516 and then doing a read to memory location 516 via A and subtracting 1 from the read content to write the result to A

A-Instruction



Syntax: *@value*

Where *value* is either:

- a non-negative decimal constant or
- a symbol referring to such a constant (later)

Semantics:

- Sets the A register to *value*
- Side effect: RAM[A] becomes the selected RAM register

Example: @21

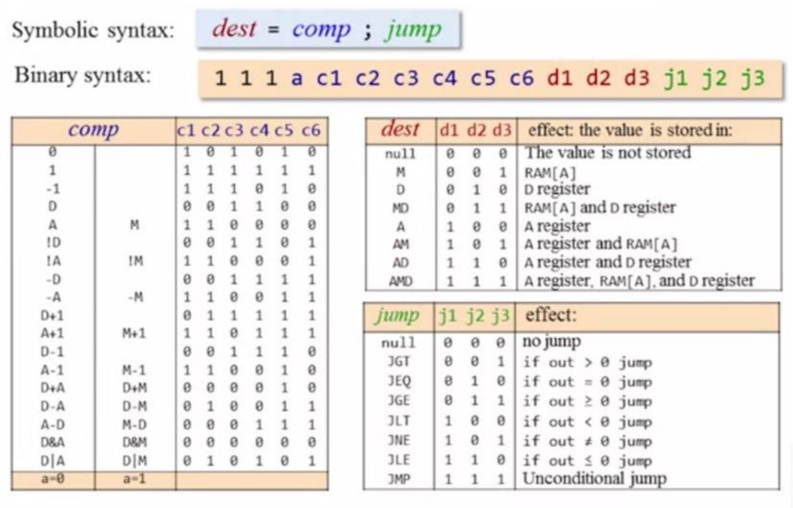
Effect:

- Sets the A register to 21
- RAM[21] becomes the selected RAM register

Usage example:

```
// Set RAM[100] to -1
@100   // A=100
M=-1   // RAM[100]=-1
```

C-Instruction in Entirety



Symbolic syntax: *dest* = *comp* ; *jump*

Binary syntax: `1 1 1 a c1 c2 c3 c4 c5 c6 d1 d2 d3 j1 j2 j3`

comp		c1	c2	c3	c4	c5	c6
0		1	0	1	0	1	0
1		1	1	1	1	1	1
-1		1	1	1	0	1	0
D		0	0	1	1	0	0
A	M	1	1	0	0	0	0
!D		0	0	1	1	0	1
!A	!M	1	1	0	0	0	1
-D		0	0	1	1	1	1
-A	-M	1	1	0	0	1	1
D+1		0	1	1	1	1	1
A+1	M+1	1	1	0	1	1	1
D-1		0	0	1	1	1	0
A-1	M-1	1	1	0	0	1	0
D+A	D+M	0	0	0	0	1	0
D-A	D-M	0	1	0	0	1	1
A-D	M-D	0	0	0	1	1	1
D&A	D&M	0	0	0	0	0	0
D|A	D|M	0	1	0	1	0	1
a=0	a=1						

dest	d1	d2	d3	effect: the value is stored in:
null	0	0	0	The value is not stored
M	0	0	1	RAM[A]
D	0	1	0	D register
MD	0	1	1	RAM[A] and D register
A	1	0	0	A register
AM	1	0	1	A register and RAM[A]
AD	1	1	0	A register and D register
AMD	1	1	1	A register, RAM[A], and D register

jump	j1	j2	j3	effect:
null	0	0	0	no jump
JGT	0	0	1	if out > 0 jump
JEQ	0	1	0	if out = 0 jump
JGE	0	1	1	if out ≥ 0 jump
JLT	1	0	0	if out < 0 jump
JNE	1	0	1	if out ≠ 0 jump
JLE	1	1	0	if out ≤ 0 jump
JMP	1	1	1	Unconditional jump

Question 4. [8 points] Write HACK assembly code for the following VM commands:

- □ **push constant 5**
- □ **sub**
- □ **pop local 2**
- □ **if-goto label (assume label is at ROM location 58)**

```
//push constant 5:
@5
//A = 5;
D = A;
@sp
A = M;
M = D;
@sp
M = M + 1;
//sub
@sp
AM = M-1;
D = M;
A = A-1;
D = M – D;

//pop local 2
@sp
```

```
A = M - 1;

M = D;

A = A – 1;

D = A;

M = M – 1;

@sp

//if-goto label

@SP

AM = M – 1

D = M

@58

D;JNE
```