Name:____________________
Date: ____________________

The A-instruction: symbolic and binary syntax

Semantics: Set the A register to *value*

Symbolic syntax:

@*value*

Where *value* is either:

- a non-negative decimal constant ≤ 32767 ($=2^{15}-1$) or
- a symbol referring to such a constant (later)

Example:

@21

Effect: sets the A register to 21

Binary syntax: 0vvvvvvvvvvvvvvv
For example: @21 (symbolic syntax – assembly code)
0000000000010101 (binary syntax – machine code)

The C-instruction: symbolic and binary syntax

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

Name:______________________
Date: ______________________

Lab #6 – Computer Architecture

Recall the two Assembly Instructions, A and C:

The 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

The C-instruction

dest = *comp* ; *jump* (both *dest* and *jump* are optional)

where:

comp = `0, 1, -1, D, A, !D, !A, -D, -A, D+1, A+1, D-1, A-1, D+A, D-A, A-D, D&A, D|A`
`M, !M, -M, M+1, M-1, D+M, D-M, M-D, D&M, D|M`

dest = `null, M, D, MD, A, AM, AD, AMD` M refers to RAM[A]

jump = `null, JGT, JEQ, JGE, JLT, JNE, JLE, JMP` if (*comp jump* 0) jump to execute the instruction in ROM[A]

Semantics:
- Compute the value of *comp*
- Stores the result in *dest*;
- If the Boolean expression (*comp jump* 0) is true, jumps to execute the instruction stored in ROM[A].

1) Write an assembly program to perform the multiplication of two integers (stored at R0 and R1) and store the result in R2. You can perform multiplication through successive addition. For example:

5 * 4 = (5 + 5 + 5 + 5) = 20

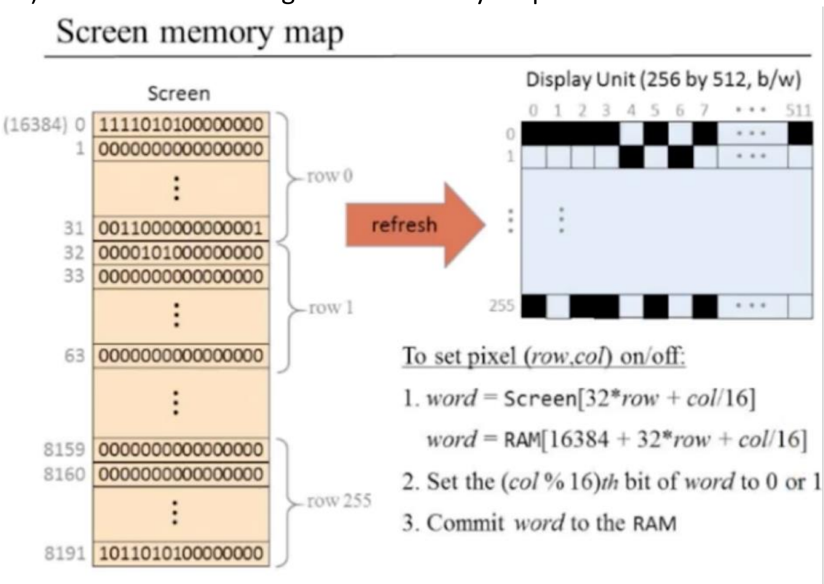
a. First write Java code using a for loop to perform the successive addition. Assume the following:
int R0 = 5;
int R1 = 4;
int R2; // Stores the result of R0 * R1 (using successive addition in a for loop)

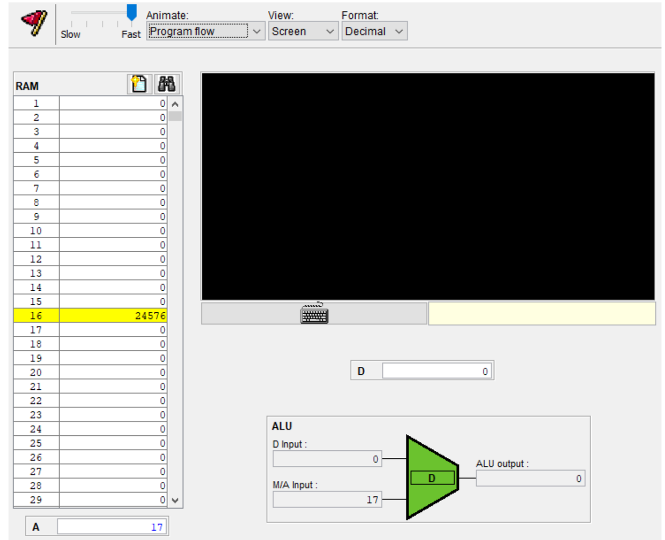
```
Public Class main {
    Public Static void main(String[] args) {
        int r0=5, R1=4, R2=0;
        while (r0 >= 0){
            r0=r0-R1;
            if (r0<0){
                r0=r0+R1;}
            R2=R2+1
} } }
```

b. Next, convert the high-level Java code into assembly code:

```
(Start)
  @R2
  M=0
(Loop)
  @R1
  D=M
  @END
  D;JLE
  @R0
  D=M
  @R2
  M=M+D
  @R1
  M=M-1
  @Loop
  0;JMP
(END)
  @End
  0;JMP
```

2) Given the following screen memory map:





Write the assembly code that will loop through all the screen locations (in RAM) and darken them to produce the screen shot above:

```
@Screen
D=A
@address
M=D
@8192 //total pixels
D=A
@Pixels
M=D
(Loop)
@Pixels
D=M
D;JEQ
D=M
D;JNE
@Address
A=M
M=-1
@Address
M=M+1
@pixels
M=M-1
@Loop
0;JMP
```

Directions: Complete the translation from Assembly to Machine code for the following instructions.

1) @1

parts	0	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
values	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

2) @SCREEN //@_________

parts	0	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
values	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

3) @KBD//@_________

parts	0	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
values	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

4) @R5//@_________

parts	0	V	V	V	V	V	V	V	V	V	V	V	V	V	V	V
values	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

5) A=-1

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	1	1	1	0	1	0	1	0	0	0	0	0

6) A=D

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	1	0	0	0	0	0

7) A=M

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	1	1	0	0	0	1	1	0	0	0	0	0

8) A=D&M

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	0	0	0	0	0	0	1	0	0	0	0	0

9) D=A

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3	
values	1	1	1	0	1	1	0	0	0	0	0	1	0	0	0	0	

10) D=M

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0

11) D=D|A

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	1	0	1	0	1	0	1	0	0	0	0

12) D=!D

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	1	0	1	0	0	0	0

13) M=A

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	1	1	0	0	0	0	0	0	1	0	0	0

14) M=D

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	0	0	1	0	0	0

15) M=D-1

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	1	0	0	0	1	0	0	0

16) M=D+A

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0

17) MD=M-D

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	0	0	0	1	1	1	0	1	1	0	0	0

18) AD=M+1

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	1	1	0	1	1	1	1	1	0	0	0	0

19) AMD=!M

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	1	1	1	0	0	0	1	1	1	1	0	0	0

20) 0;JMP //0 is ________

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	1	0	1	0	1	0	0	0	0	1	1	1

21) D;JGT

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	0	0	0	0	0	1

22) D;JEQ

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	0	0	0	0	1	0

23) D;JLE

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	0	0	0	1	1	0

24) D;JNE

parts	1	1	1	a	c1	c2	c3	c4	c5	c6	d1	d2	d3	j1	j2	j3
values	1	1	1	0	0	0	1	1	0	0	0	0	0	1	0	1

Lab #6 Computer Architecture - Memory

Name:______________________

Section/Time:______________________

Date: ______________________

❑ Write in the decimal values for each address below, and convert to binary:

NO CALCULATOR! Use back as scratch paper

RAM16K minimum address (first address) = (0)$_{10}$														
[14]	**[13]**	**[12]**	**[11]**	**[10]**	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3]**	**[2]**	**[1]**	**[0]**
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

RAM16K maximum address (last address) = (16383)$_{10}$														
[14]	**[13]**	**[12]**	**[11]**	**[10]**	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3]**	**[2]**	**[1]**	**[0]**
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1

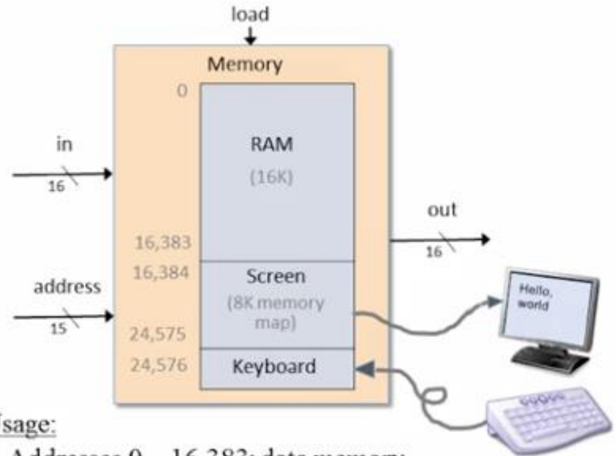
SCREEN Pointer address (start of screen memory map) = (16384)$_{10}$														
[14]	**[13]**	**[12]**	**[11]**	**[10]**	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3]**	**[2]**	**[1]**	**[0]**
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Last possible address for screen memory map = (24575)$_{10}$														
[14]	**[13]**	**[12]**	**[11]**	**[10]**	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3]**	**[2]**	**[1]**	**[0]**
1	0	1	1	1	1	1	1	1	1	1	1	1	1	1

KBD Pointer address = (24576)$_{10}$														
[14]	**[13]**	**[12]**	**[11]**	**[10]**	**[9]**	**[8]**	**[7]**	**[6]**	**[5]**	**[4]**	**[3]**	**[2]**	**[1]**	**[0]**
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0

Lab #6 Computer Architecture - Memory

Memory implementation



Usage:
- Addresses 0 – 16,383: data memory
- Addresses 16,384 – 24,575: screen memory map
- Address 24,576: keyboard memory map

- ❑ When interacting with the Memory chip, how can it differentiate between an address for the screen memory map, the keyboard register, and RAM16?

RAM 0-16,383
Screen 16,384-24575
KBD 24576

- ❑ What are the bus sizes for each chip in Memory.hdl?

DMux
RAM16k
Keyboard
Screen
Mux4way16

- ❑ What does the **`load`** pin accomplish?

write enable bit

- ❑ How is it used for the Memory chip?
    - ❑ Play through scenarios for the value of load and what SHOULD happen to each part of memory. How can the chip differentiate where load should go?

write enable bit