CPSC 359 – Spring 2015
Assignment 3

Due: June 26th @ 11:59PM

Objective: To work with Mic-1 MMV, extending the IJVM ISA

1. A JAS program (10 points total)

a. Adding a new ISA instruction (2 point):

In this part, we will walk you through modifying the microprogram (you will do more of this in part 2 below). We will add a new JAS instruction INEG that negates (in 2's complement) the value at the top of the stack. JVM has INEG as part of its instruction set.

Locate the file *ijvm.conf*, within the Mic-1 MMV files. Add to it the following line:

```
0x74  INEG  \\ write an appropriate comment
```

We are using the opcode 74 since it is not used by any other existing instruction (also this is the actual opcode for INEG in JVM).

Locate the file *mic1ijvm.mal*, which contains the Mic-1 microprogram. At the beginning of the file, there are few labels defined, one for each starting address of the mircoinstrucions that interpret a given JAS instruction. Add to these labels the following line:

```
.label  ineg1       0x74
```

(Make sure the value (0x74) is the same as the one you defined in *ijvm.conf*.)

At the end of the file add the following microinstructions:

```
ineg1 H = TOS
ineg2 MAR = SP
ineg3 TOS = MDR = -H; wr; goto Main1
```

Add appropriate comments to these microinstructions.

Run MMV from a folder that contains *ijvm.conf*. From MMV, open *mic1ijvm.mal* assemble it, and load it. If it contains errors, correct them. (The code provided here has been tested and is error-free).

We have modified the microporgam so that it supports the new ISA (or JAS) instruction INEG. You will need it for part b.

b. Write a JAS program that contains the method:

- power(int1, int2): returns int1 to the power int2. **(8 points)**

Since IJVM ISA does not have instructions for multiplication and division, your JAS methods must use addition and subtraction, instead. To calculate x^y, you multiply x by itself y times. Multiplication can be done by addition. The following algorithm (imul) shows you how to calculate (int1 * int2), using addition:

```
imul(int1, int2) {
m = 0
c = min(abs(int1), abs(int2))
o = max(abs(int1), abs(int2))
for (i = 0; i < c; i++)
        m = m + o
if (exactly one of int1 or int2 is negative)
        m = - m
return m
}
```

You need not worry about overflows.

Demonstrate the use of the *power()* methods by calling them from the main method **(1 point)**. Assemble and test your program using the Mic-1 MMV.

Note: In the file *add.jas* that is contained in the examples folder of Mic-1 MMV, there are I/O methods to read and print numbers. You may use these methods for testing; however, they do not work with negative numbers. Hence, when testing your code with negative numbers, you have to do it the hard way, monitoring the stack and registers, or write your own methods.

2. Extending the IJVM ISA (10 points total)

Add two new instructions to the IJVM ISA:

- IFEVEN *offset*: pops the top value on the stack and branches to a 16-bit offset if the popped value is even **(4.5 points)**.
- POWER: pops the top two values on the stack and pushes their the first (top) raised to the power of the second (next to top) **(4.5 points)**

For POWER, you can assume that both values are non-negative.

Modify the Mic-1 microprogram (in MAL) so that these two instructions can be interpreted by the IJVM hardware. Follow similar steps to the ones in part 1(a) above. Write a tester JAS program that uses these two new instructions. **(1 point)**

You need not worry about overflows since the IJVM ALU does not have an overflow flag. Up to **1.5 points** can be deducted from each implementation (IFEVEN and POWER) for inefficiency.

Thoroughly document your programs (both JAS and MAL); programs that are not properly documented can lose up to **3 points**.
Assemble and test your programs using the Mic-1 MMV.

Programs that do not compile cannot receive more than **5 points**. Programs that compile, but do not run at all can receive a maximum of **6 points**.

Submission: Follow your TA's instructions to submit via D2L or email.

Late submission and academic misconduct: Refer to the course syllabus for such policies.

Teams: You may work as a team of up to 3 members for the assignment.

Note: Here is one (important) observation that I learned the hard way when using MMV.

You may get a J**ava.lang.NullPointerException** or **Java.lang.Exception** when assembling some MAL code in MMV. Two things can help you recover from these, which I have learned the hard way:

1. Make sure there is an empty line at the end of you MAL file
2. If your code contains multi-way branching (if-else), add your code gradually to the MAL file as follows:
   (a) Add all microinstrcutions (MI) that precede the next MI that has if-else. (So this chunk you have just added does not have any if-else MIs). Compile this part alone and correct any errors before proceeding.
   (b) Add the next if-else MI, compile, and correct errors.
   GOTO (a) until all MIs have been added to the MAL file

Also when adding an if-else MI, such as **N=OPC; if (N) goto labelYes; else goto labelNo**, make sure labelYes and labelNo are new, never-used labels in your program. If you want the if-else to branch to an already existing MI, say power12, then have an unconditional branch as part of the labelYes or labelNo MI, such as **labelYes goto power12**.