Tutorial 01

This tutorial assumes that you have already carried out the *Getting Started* setup steps and additionally prepared for Session 1 by reading the *Simple Stack Machine Overview* document (and know where to find the more detailed reference documents *The SSM Instruction Set* and *The SSM Assembler*).

In the `Tutorial01 Resources` folder in Moodle you will find several partially complete SSM assembly code files. Download these files and copy them into the `data` folder within your *Calculon* project folder. Some of the files include Java code snippets in comments and you may be asked to replicate the behaviour of these snippets by writing SSM assembly code. In order to test your solutions you will need to first assemble and then execute your code in a Terminal (just as you did at the very end of *Getting Started*, *Command-line Set-up*).

Open your *Calculon* project in IntelliJ and open a Terminal window from IntelliJ. To assemble your code, run the following command (we use Exercise 1 as an example):

```
java Assemble data/Exercise1.ssma
```

If your assembly code has syntax errors (quite likely) the assembler will report an error and you will need to edit the file and try again. Otherwise the assembler will generate a file called `a.out` containing executable SSM machine code. To execute this code, run the following command:

```
java Exec a.out
```

As a convenience, you may combine those two steps into the single command:

```
java Run data/Exercise1.ssma
```

(the Run command just executes Assemble and Exec in sequence).

Your code may contain logical errors (very likely) in which case it will either produce the wrong output or crash. There is a video resource in Moodle which provides some ideas for how to go about debugging your code. We will also look at this in the lectures.

Exercise 1

You are asked to write SSM assembly code to calculate (2+3)*7 and leave the result on the opstack (code which will then print the result is included in the file already). To get you started, here is the code for the first step (calculate 2+3):

```
push 2
push 3
add
```

This instruction sequence leaves the value 5 on top of the opstack.

Exercise 2

Similar to Exercise 1 but calculate 2+(3*7).

Exercise 3

The file contains four Java code snippets. SSM assembly code for the first two snippets has already been provided. Using this as your guide, write assembly code for the other two.

Exercise 4

The code in this file has the same effect as Java code which decrements an integer variable by 1 and then prints its new value. Note that `$x` is initialised in the .data section to contain the value 10,000, so when you assemble and execute this code it should print 9999.

You are asked to modify this code so that it repeats indefinitely, replicating the behaviour of a Java while-loop:

```
while (true) {
   x = x - 1;
   System.out.println(x);
}
```

You will need to add a new labelling at the start of the assembly code and an appropriate jump instruction at the end (just before the halt instruction). **Note**: since this is an infinite loop, you will have to force the SSM to halt by using Ctrl-C in the terminal window.

Exercise 5

Copy your code from Exercise 4. Modify it so that the loop terminates once `$x` contains 0:

```
while (x != 0) {
   x = x - 1;
   System.out.println(x);
}
```

Hint: add code which uses a conditional-jump instruction `jumpi_z` *before* the existing jump instruction, to “jump out” of the loop if `$x` contains 0 (you will also need to add a new labelling for the jump target).

Exercise 6

You are asked to replicate the behaviour of an if-statement. You will need to use appropriate labellings and one or more jumps (similar to the previous exercises). To test your code you should try different cases by editing the initial value for `$x` in the data section and re-assembling.