CS 212 Part 1
Karroht & Seadz, Corp.
Core Familiarization Tasks
Spring 2008

Please be sure to read the material posted on the website prior to attempting this assignment. Also, make sure your submission follows all guidelines, as described in section guidelines.

Introduction

Welcome to Karroht & Seadz, Corp. This set of familiarization activities will get you up to speed with all of our backend systems. Here is a quick list of the backend systems that you need to know about:

- SaM - The Stack Machine Simulator is a runtime environment that emulates assembler code on your desktop. See http://www.cs.cornell.edu/courses/cs212/2008sp/Compiler/SaM/ for more information.

- SaM Code - This is a pseudo assembly like language that the SaM Simulator understands. For more information, and the instructions that constitute SaM Code, see http://www.cs.cornell.edu/courses/cs212/2008sp/Compiler/SaM/doc/SaMDesign-2.6.2.pdf. (While the current version of SaM is 2.6.3, the differences between v2.6.2 and v2.6.3 are minor, so the old documentation is sufficient for these tasks).

Tasks

The following tasks will introduce you to SaM and the SaM Simulator. Do not simplify any of the calculations in these tasks. For tasks 1-3 you are not allowed to use **PUSHOFF**, **PUSHABS**, **STOREOFF**, and **STOREABS** except as described for task 3, and you may not modify any of the built in registers. Also, do not use a return variable for tasks 1-3.

1. Write a Samcode program **task1.sam** that evaluates:

$$\frac{13}{2}$$

   Your code must actually evaluate 13 divided by 2 (ie: you must use a div instruction and the two operands it uses must be 13 and 2 respectively). Unfortunately due to union labor contracts, the first line of your code must be **PUSHIMM 13**, and you cannot use any instruction that takes any immediate operands. You must therefore use some of the less obvious commands found in the SaM documentation to complete this task. Please use only integer operations for the arithmetic part this task. As a reminder, to terminate your program you must use the “STOP” instruction.

2. Write a Samcode program **task2.sam** that evaluates the following expression:

$$\left[(\neg F)\vee\left(\left(\neg\left(\frac{\left(\frac{6-1}{3}\right)+2}{4}\wedge T\right)\right)\vee\left(\frac{6-1}{9}\right)\right)\right]$$

Remember that T is true and F is false. Standard order of operations applies for this task. If you are unfamiliar with logic symbols, please see an appropriate website[1]. Remember, in SaM the integer zero is treated as false and any non-zero is considered to represent true. Please use only integer operations for the arithmetic part this task. As a reminder, to terminate your program you must use the "STOP" instruction.

3. Write Samcode that calculates the value of $\vartheta(A,B,C)$ using only the **AND**, **PUSHOFF**, and **NOT** instructions, where the ϑ operator is defined by the following truth table:

A	B	C	$\vartheta(A,B,C)$
F	F	F	**F**
F	F	T	**F**
F	T	F	**F**
F	T	T	**T**
T	F	F	**T**
T	F	T	**F**
T	T	F	**F**
T	T	T	**F**

Remember that T is true and F is false. Also, according to De Morgan's laws:

$$\neg(P\vee Q)=(\neg P)\wedge(\neg Q)$$

Your code should fit into the following template, which prompts for three integers and returns the value of your computation. Remember, in SaM the integer zero is treated as false and any non-zero is considered to represent true (you can assume the user will enter an integer, which you can then treat as true/false according to this rule). Please note that your computation should not remove the three integers from the stack. This is handled by the code after your segment. Simply leave the result at the top of the stack. The SaM command **JUMPC** should *not* be used in this task.

[1] Such as http://en.wikipedia.org/wiki/Table_of_logic_symbols

```
PUSHIMM 0  // Return variable slot
READ       // Arg 1 (=A)
READ       // Arg 2 (=B)
READ       // Arg 3 (=C)
LINK
JSR CFT3   // Core Familiarization Task 3
UNLINK
ADDSP -3
STOP

CFT3:
// #########  YOUR CODE HERE  #############
// Use PUSHOFF -1 for Arg3
// Use PUSHOFF -2 for Arg2
// Use PUSHOFF -3 for Arg1
// Perform computations
// Place the return value in the return variable slot
//  (Use the STOREABS instruction)
// #########  END YOUR CODE  ##############
RST
```

You are allowed to find the solution using reference materials or the Internet. However, if you do not derive the expression on your own, you must credit your source in the comments at the top of your submission. Please submit the code you write, **including** the template we give you, in a file called task3.sam.

4. Write Samcode that corresponds to the following Java function using only the absolute addressing approach

```
public boolean function()
{
    int a, b;
    boolean c, d;
    a = 2 + 8;
    b = a - 6;
    d = !(c = true);
    return ( c || d ) && ( ( a + b ) != 0 );
}
```

Note that the && and || operators in this code do *not* short-circuit. You are not required to write the code that will call your function (this is something that will be introduced in later parts of the project).
You are required to use the following template:

```
PUSHIMM 0  // Return variable slot
LINK
JSR CFT4   // Core Familiarization Task 4
UNLINK
STOP

CFT4:
// #########  YOUR CODE HERE  #############
// Reserve space for “local” variables
// Evaluate and leave return value on top of stack
// Remove “local” variables (but not the return variable)
//  from the stack
// #########  END YOUR CODE  ##############
STOREOFF -1
RST
```

Please submit the code you write, **including** the template we give you, in a file called task4.sam. Make sure to allocate space for all four variables **and** the return variable. The variable slots you create will have addresses ≥ 3. After your code finishes running, the return value should be at address 3 (you will see that the code we provide uses addresses 0, 1, and 2). When your code terminates, it should leave the return value on the top of the stack.

5. Write Samcode that corresponds to the Java code in Task 4 using only the relative addressing approach.

   For this part of the assignment you should assume that the FBR register to which addresses are relative is set to SP - 2 when your code begins execution (immediately below the first free stack location). Therefore, the first free location is at offset 2, the second at offset 3, etc. . . . Other offsets are reserved and you should not write to it. You should again use the template provided for task 4 and submit the code you write, **including** the template we give you, in a file called task5.sam.

Submitting Your Work

Be sure to test your code before submitting it.

Partners

Although we allow groups of up to three people for future assignments, for part 1 you must work by yourself.

Submission Guidelines

For tasks 1-5, write each problem's solution in a separate text file, as described in each task description. For example, if we gave a task 0 its solution might be as follows:

```
// +---------------------------------------------+
// | Part 1, Task 0
// | Warrel Dane
// | wd40
// | 314159265
//+---------------------------------------------+
//Write a program that returns true
PUSHIMM 1 //push true
STOP //stop execution
```

For this assignment, you should comment every single line of Samcode so that we know that you know what is happening. For general submission instructions, follow the CMS submission specifications on the CS211 Spring 2008 website. Each file you create will be submitted individually on CMS. If we cannot read your work in a standard text editor, the work is unacceptable and will receive no credit.

Note that SaM only recognizes the single-line, double slash, comment style (//) and not the common multi-line comment convention of java and C (/* */).

Grading

We will grade your assignment based on style (how well you followed our instructions, how neatly you arranged your solutions, how neatly you wrote your code, and how clear you made your comments) and correctness (how well the programs work). The assignment will be graded out of 100 points.