CMSC 301 Simulator Project - Fall 2003

This project will be completed in three phases that model the development process, except that I am providing the basic class definition and the structure of the code in the main method. The end result will be a simulator for the STACK computer. This simulator will accept load modules and execute them.

Getting the Files

Create a folder in your account called simulator and perform a copy after making it the active folder.

```
cd cs301
mkdir simulator
cd simulator
cp ~lbarnett/share/simulator/* .
```

This should get you the following files:

- `makefile` // directives for the make utility
- `phase1.cpp` // skeleton for test program for four utility methods
- `STK12.cpp` // skeleton for simulator class
- `STK12.h` // declaration for simulator class
- `STK12sim.cpp` // main program for simulator

Phase 1 - Creating and Testing Some Utility Methods (8 pts)

Due Wednesday, October 1st before midnight.

**What to do.** You will complete the file `phase1.cpp` which implements four utility methods and tests them. Descriptions of the methods appear in the comments in `phase1.cpp`. These methods must work before you can do the next two phases. A solution will be available next Thursday.

You will need to compile this by hand or add an entry in the makefile. (I would suggest the latter; make is a very powerful and convenient tool and it would be well to learn to use it as soon as possible. Further instructions can be found on-line on the “Tools for Computer Scientists” web page: [http://www.mathcs.richmond.edu/CS/Tools.html](http://www.mathcs.richmond.edu/CS/Tools.html) The command is

```
g++ -o phase1 phase1.cpp
```

To turn in your program, use the script

```
~lbarnett/share/cs301_sim
```

Phase 2 - Implementing Most of the Simulator (26 pts)

Due by 5 p.m. Friday, October 10th. (Shoot for the day before!)

**What to do.** You are to complete the file `STK12.cpp` which implements the STK12 class. Of course, you will want to copy the completed utility methods of phase 1 from `phase1.cpp` to this file and indicate they are part of the STK12 implementation. You will also be filling in the `load_module()` and `execute()` methods. You are only required to have your simulator accept the following instructions at this point:
NOP, HALT, ADD, SUB, NEG, MUL, DIV, TEST, RESET, IN, OUT

You can write some simple programs with just this subset, using IN to add values to the CPU stack and OUT to see the results. Each of these must be implemented according to the description of each instruction in the handout on the STACK computer. I will only use test programs that contain these instructions. I have put a copy of the executable for my assembler, called ASM, in ~lbarnett/share/sim_exe. You can use this to assemble simple test programs you’ve written into load modules to test with. You may also wish to see what output my simulator gives by using my executable SIM, available in the same directory. You may want to keep them separate from your code as the makefile will create SIM and overwrite my version in the same folder.

The key to success is careful planning and the development of a number of simple cases for testing. Start with the trivial ones (NOP, HALT, IN, OUT, RESET) first, then move on to the others. Whenever the cpu stack is changed be sure to check for overflow or underflow and set the flags if such an error occurs. Also, the arithmetic instructions (except for NEG) must have two operands on the cpu stack.

The STK12sim.cpp main program is set up to either let you put the name of the load module to simulate on the command line or enter it interactively if no command line arguments are specified. It is also set up to accept the “-d” flag to turn on interactive debugging. Trace the code to see how this information is passed into the STK12 class. If debugging is turned on, your execute( ) method should stop after each instruction is executed, display the contents of the registers, and ask if the user would like to see the contents of memory. (Note that methods have already been provided to display the registers and memory.)

The turn-in script you used for phase 1 will also work for this phase of the project.

**Phase 3 - The Completed Simulator (16 points)**

Due Sunday, October 19th by midnight.

Follow the instructions for turning in the previous phase to turn in this one.

**What to do.** Complete the remaining instructions for the simulator:

PUSHI, PUSH, POP, JMP, JMPN, JMPZ, RET, ARG, CALL, INC-SP, DEC-SP, PUSH-REL-SP, POP-REL-SP

and correct errors in the returned code for phase 2.

**WARNING**

You have three weeks to complete all three phases of this project, but our first test and Fall break occur during this time frame, and this may be the largest program you have ever been asked to write. Use your time wisely.