SimpleScalar Familiarization Exercise
CS 654 Exercise #1

Due Wednesday, Sept. 3, 9:30am

This exercise is chiefly meant to help you install and familiarize yourself with the SimpleScalar 3.0b Toolset. Three short written exercises related to Chapter 1 are also included. All these are meant to be quick and easy, but this doesn't mean you should leave them to the last minute!

Part I: Installing SimpleScalar 3.0b

0. Please use one of the Sun interactive machines: cobra, mamba, adder.

1. Copy `/af6/skadron/SimpleScalar/simplesim-3_0b_tar.gz` to your home directory

2. Unzip simplesim-3_0b_tar.gz with the gunzip utility:
   ```
   gunzip simplesim-3_0b_tar.gz
   ```

3. Create your own SimpleScalar directory by executing the following command:
   ```
   tar xvf simplesim-3_0b_tar
   ```
   This will create a directory named simplesim-3.0 wherever you execute the command.

4. Follow the directions for installation in the `README` file. The following are some additional changes to the README that will make the installation smoother.
   a) Using the editor of your choice, edit the Makefile.
   Uncomment line 156 of the makefile. The line reads:
   `#MLIBS = ``./sysprobe -libs`` -lm -lsocket -lnsl`
   To uncomment it, simply remove the # symbol.

   b) Replace the line which reads:

   ```
   OFLAGS = -O0 -g –Wall
   ```

   with the following lines:

   ```
   ifdef DEBUG
   OPT= -O0 -g
   else
   OPT= -O3 -finline-functions -funroll-loops
   endif
   OFLAGS = $(OPT) -g -Wall
   ```

   `OFLAGS` is the variable in the makefile which specifies the level of optimization used by gcc. The original line which you replaced always sets `OFLAGS to -O0 –g –Wall`, which is useful for debugging. The change you just made will

allow you to later specify whether or not you want optimizations turned on.
I.e., if debugging is to be turned on :

```
make DEBUG=1 <simulator name>
```

If no debugging information is needed, then the following is used:

```
make <simulator name>
```

c) In addition, comment out the line which reads:
`FFLAGS=-DDEBUG`
by adding # in front of the line. It should now read:
`#FFLAGS=-DDEBUG`

d) For the purposes of this exercise we will be using SimpleScalar's PISA instruction set architecture.
In step b) do:
`make config-pisa`

e) Edit `syscall.c`. Using the editor of your choice, uncomment line 133 in `syscall.c`. The line reads:
`/* #include <sys/file.h> */`

Remove the comment characters `/*` and `*/`. This file needs to be included in order to get the simulators to compile.

f) `make DEBUG=1`
This builds all the simulators in the SimpleScalar Toolset with debug information turned on.

5. Test the simulator installation. `make sim-tests`
As the README file says, this will take some time. If the installation has been done correctly, the output will show no differences.

Part II: Using a SimpleScalar simulator

1. Simulators in SimpleScalar are named with "sim" as a prefix. If you peruse your simplesim-3.0 directory, you will see there are several simulators packaged with SimpleScalar including sim-safe, sim-fast, sim-eio, sim-outorder to name a few. This exercise will use the simplest simulator, `sim-safe`.

2. From your simplesim3.0 directory execute `sim-safe` with no command-line arguments. Information about the usage, description of the simulator and default values for the command-line options will print to the screen. The name of the option and the type of values it takes appears as the first item on each line. The next items on the line are the option's default values, and then a description of the option. Take note of the following options:

Option Name	Arg Type	Default Value	Description
-redir:sim	<string>	<null>	redirect simulator output to file
-redir:prog	<string>	<null>	redirect simulated program output to file
-nice	<int>	0	simulator scheduling priority
-max:inst	<uint>	0	maximum number of inst's to execute

3. Make a results directory in your simplesim3.0 directory:
   `mkdir results`

4. Run sim-safe with commandline arguments for redirecting simulator output and program output set to be written into the results directory. In this case, the benchmark we will be using is a test program called test-math. It is located in `simplesim-3.0/tests/bin.big`. The commandline is as follows:

```
sim-safe –redir:sim results/sim1.out –redir:prog results/prog1.out
tests/bin.big/test-math
```

5. Change directories to the results directory and examine sim.out and prog.out. In sim.out, in the first part of the file you should see the same information as when you ran sim-safe without any arguments. The lower half of the file contains various statistics that sim-safe collects. In particular, note sim_num_insn (total number of instructions executed) and sim_num_refs (number of loads and stores). It should read as follows:

```
sim_num_insn              216084 # total number of instructions executed
sim_num_refs               56936 # total number of loads and stores executed
```

6. Rerun sim-safe on test-math, but this time, also set the –max:inst option to 50000 instructions. Redirect simulator output to results/sim2.out and program output to results/prog2.out. When you examine sim2.out, you should see the following:

```
sim_num_insn              50000 # total number of instructions executed
sim_num_refs              15626 # total number of loads and stores executed
```

7. Place copies of sim1.out and sim2.out in your submission directory.

Part III: Writing your own program to run through a SimpleScalar simulator

1. Write a c program that prints “Hello World!” For the purposes of this exercise, it is assumed that the file is called hello.c.

2. Use SimpleScalar’s version of gcc to compile this program for the PISA instruction set. A copy of this version of gcc, called `ssbig-na-sstrix-gcc` is located at:
   `/af6/skadron/SimpleScalar/sun/bin`

To compile your program to be run with sim-safe:
`/af6/skadron/SimpleScalar/sun/bin/ssbig-na-sstrix-gcc -o hello hello.c`

3. Now you have a compiled version of your hello world program which sim-safe can simulate.

4. Run sim-safe on hello, redirecting simulator output to results/hello.sim.out and program output to results/hello.prog.out:

```
sim-safe -redir:sim results/hello.sim.out -redir:prog
results/hello.prog.out hello
```

5. Examine hello.sim.out and hello.prog.out. Place a copy of `hello.c` and `hello.sim.out` to submit.

Part IV: Submitting your work

Make a directory called cs654/ in your home directory, and within cs654/, make a subdirectory called assign1/. Make sure that you have copies of sim1.out, sim2.out, hello.c, and hello.sim.out in the assign1 directory, and send e-mail to cs654@cs. Please be sure to follow the homework assignment submission guidelines in the Beginning of Course Memo.

Part V: Written Exercises

These should be submitted in hardcopy form as per the instructions in the Beginning of Course Memo.

1. A particular computer has a clock rate of 100 MHz. After measuring the execution of a program, it is observed that the program executes 1.2 billion instructions in 30 seconds. What is the average cycles per instruction (CPI) for this machine and this program?

2. Hennessy & Patterson #1.3

3. Hennessy & Patterson #1.7