CprE 381 - Computer Architecture and Assembly Level Programming
Spring 2017

Lab-2

INTRODUCTION:

This introductory lab is aimed at introducing you to the *Simplescalar* simulator, while letting you explore some of the topics introduced in section 1.3 of your textbook. It is assumed that you are familiar with C programming and the basics of UNIX command line (see optional readings for materials to brush up your C and command line skills).

SimpleScalar:

SimpleScalar is a suite of processor simulators and supporting tools. It enables the user to model a virtual computer system with CPU and Memory. The user can then run programs on these models using execution driven simulation and evaluate performance parameters. SimpleScalar consists of a collection of microarchitecture simulators which emulate the processor at different levels of detail. You only need to know a few of them for the course, however you are encouraged to play around with them to see the different performance metrics each simulator evaluates.

SimpleScalar is installed on the ECpE Remote Server(linux-1). The installation directory contains the simulators (*sim-safe, sim-fast, sim-profile, sim-outorder etc.)* along with some sample benchmark programs located in *tests-alpha* (for Alpha ISA) and *tests-pisa* (for P-ISA) directories. These benchmark files have been precompiled for you. You only need to pass them as arguments to the simulator and analyze how the processor performs.

1. Getting Started:
Let's get started! Run **Putty** from the Start menu and connect to any of linux-1, 2, 3 or 4.

linux-1.ece.iastate.edu (type this into the host name field, click Open. Click **YES** if a pop-up window appears asking for key authentication).

Enter your login credentials. You should now get a command-line interface to the server. Navigate to SimpleScalar installation directory

```
cd /usr/local/ss/simplesim-3.0
```

(Use the ***ls*** command if you want to view all files/directories in your present working directory.)

Next, simulate the program *test-math* located inside *tests-pisa/bin.little* on the *sim-fast* simulation tool. To do this, run:

```
./sim-fast tests-pisa/bin.little/test-math
```

The simulator will dump the program output and the performance metrics on your console screen. Locate the line ***sim: ** starting *fast* functional simulation **.*** The program output can be found after this. Towards the end, you will find ***sim: ** simulation statistics ***** after which the performance metrics are listed.

Try simulating *test-fmath, test-printf, test-lswlr* and *test-llong.* Note down the endian-ness (big or little) and total number of instructions executed for each program using the table below.

Test Program	# Instructions	Endian-ness
test-fmath	53459	0 little
test-printf	1813891	0 little
test-lswlr	8877	0 little
test-llong	29642	0 little

2. Compilers Optimization:

The compiler plays an important role in your program's performance. It processes your high level code and turns it into machine code that the processor can understand. In doing so, it performs several tasks such as semantic analysis, syntactic analysis, optimization etc. For this lab, the optimization part is of interest to us. Code optimization is done, in the general sense, to make your code run efficiently or use fewer resources. We will use the GCC compiler in all of the Simplescalar experiments. GNU Compiler Collection (GCC) was developed by the GNU project, initially for C alone, but now supports other high level languages as well. It can perform different levels of optimization, depending on the requirement. Without optimization (level 0), gcc attempts to make the compilation process faster. Optimization levels 1, 2 and 3 attempt to improve performance of the generated code but at the cost of compilation time, where level 3 implements the most rigorous optimization methods.

Let us try observing performance differences between a compiler optimized code and the non-optimized version of the same. Go back to Windows and download the files (in lab2.zip) **math_unopt** and **math_opt** and place it inside your **U:** drive in <home directory>/cpre381/lab2. Now go back to Putty.

Next, navigate back to **simplesim-3.0** directory and run the above two binaries using *sim-profile* like you did for *sim-fast* (but replace "*tests-pisa/bin.little/test-math*" with */home/<username>/cpre381/lab2/***math_unopt**. Do the same for **math_opt**. **math_unopt** is precompiled with optimization level 0 (no optimization) using gcc. **math_opt** is precompiled with optimization level 3 (max optimization) using gcc. (i)Compare #instructions using the table below. Also find the % reduction in #instructions.

Program	Optimization level 0 number of instructions	Optimization level 3 number of instructions
math	2813320	1013303

63.97% reduction

(ii) Write a selection sort algorithm (or any sorting algorithm you are familiar with) in C that sorts 50 random integers and save it in your lab2 folder. Now compile it with 0 optimization

```
cd /usr/local/ss
./bin/sslittle-na-sstrix-gcc –O0 –o /home/<username>/cpre381/lab2/<filename>
/home/<username>/cpre381/lab2/<filename>.c
```

Next compile using level 3 optimization

```
./bin/sslittle-na-sstrix-gcc –O3 –o /home/<username>/cpre381/lab2/<filename>
/home/<username>/cpre381/lab2/<filename>.c
```

Compare #instructions using the table below. Also find the % reduction in #instructions. Submit your code as well.

Program	Optimization level 0 number of instructions	Optimization level 3 number of instructions

sel_sort	39663	21573

45.61% reduction

3. Hardware Configuration:

In this section, we are going to find the best simulator configuration for a given program. Download the file **hardware** and **hardware.c** into the Windows directory that you have been using so far. In Putty, navigate to /usr/local/ss (use `pwd` to verify) and run the following (all one line):

```
. /simplesim-3.0/sim-outorder –dumpconfig
/home/<username>/cpre381/lab2/config_out.txt
/home/<username>/cpre381/lab2/hardware
```

This creates a configuration file for you that you can edit.
(i) Open up the config_out.txt file. Locate the line ***run pipeline with in-order issue*** and change it value to true (out of order issue is beyond the scope of this class). Towards the end of the file, you will find the number of integer and floating point ALUs/multipliers/dividers defined. Make the values for the ALUs, multipliers, dividers equal to 1, save the file and run the following:

```
. /simplesim-3.0/sim-outorder –config
/home/<username>/cpre381/lab2/config_out.txt
/home/<username>/cpre381/lab2/hardware
```

Report the CPI and # instructions that you get.

commited (420627)
# instructions : ~~420627~~ ~~[illegible]~~ 420802 excuted
CPI : ~~0.8368~~ 1.9015

(ii) From your understanding of the code in hardware.c, make appropriate changes to the values(do not go beyond 10) and report the best CPI (which is an improvement over (i), if any).

~~0.5566~~ new compiled changed file: hardware_0

To simulate with the modified config_out.txt file, use the following:

1.2958

```
./simplesim-3.0/sim-outorder –config
/home/<username>/cpre381/lab2/config_out.txt
/home/<username>/cpre381/lab2/hardware
```

This lab should have given you some basic idea about what machine code looks like and the Simplescalar simulator. Feel free to play around with the simulator. Good luck!

Submission:

All submissions are through Blackboard. You may edit this document and add your answers at the end of each section, or submit a separate report file. Submit your modified file(s) (config_out.txt) in a zip with your report. If you are working in a group don't forget to mention the names of both group members in your report/submission.

Optional Readings:

1.	C programming basics:	http://www.tutorialspoint.com/cprogramming/index.htm
2.	Linux terminal basics:	http://linuxcommand.org/learning_the_shell.php
3.	The simulator:	www.**simplescalar**.com/