

CSE 775: Computer Architecture
Autumn 2008

Lab Assignment #1 (Performance and Instruction Set Architecture Issues)

Instructor: D. K. Panda

Due: Monday, Nov3rd

This exercise carries 100 points.

You need to do this lab on a SUN Solaris system, the default system in the CSE environment. Each one of you must have an account on the CSE system. If you are not aware of this, please check with the SOC lab.

The purpose of this exercise is to get yourself familiarized with simplescalar simulator environment. There are seven individual simulators *sim-bpred*, *sim-cache*, *sim-cheetah*, *sim-fast*, *sim-outorder*, *sim-profile* and *sim-safe*. In this lab we will be focusing on issues related to instruction statistics and its corresponding bearing on execution time. Your first task is to select the appropriate simulator required to solve this task.

The simulators are available at `/class/cse775/simplesimbin`. There are other files available there as well. An easier way to invoke these programs from your home directory is to append the above path to your 'set path' command in the `.login` file. Alternatively, you can include these paths in your environment by appropriately modifying your `.cshrc` file. Yet another alternative is to set symbolic links to the particular executable desired using the `ln -s` command.

The handout titled *SimpleScalar Simulator Tool Description* (a copy is available on the class web page) provides you an idea about the simulator as well as the PISA-big (MIPS-based) instruction set.

The last two pages provide some tips (prepared by an earlier grader). Please follow these tips so that you can get used to the simulator environment quickly.

1. Given that the goal of this assignment is to focus on issues related to instruction statistics, instruction class behavior, and its corresponding bearing on execution time (for the PISA-big instruction set). Which of the seven simulators will you pick to achieve your objectives? [5 points]
2. For the four SPEC benchmarks located in `/class/cse775/benchmarks` and for the inputs prescribed in `/class/cse775/benchmarks/README` your objective is to construct a dynamic instruction mix table (similar to figure B.27/B.28 in the textbook). There should be one column per benchmark. In addition there should be two additional average columns with the following mixtures (MIX1-2:3:3:1 and MIX2-4:2:2:1) for the benchmark programs *cc1*, *anagram*, *compress95* and *go* respectively. [25 points]
3. Construct a similar table for instruction classes and addressing modes. Note that the addressing mode option in the simulator only provides information relating to the addressing modes for loads and stores. However, your table will need to include information relating to register direct addressing (assume that all non-memory references are register direct) as well. [20 points]

4. Assuming we are working with a chip in which all instructions take 3 clock cycles to execute and that there is no pipelining. Due to a dramatic technological breakthrough a research unit has determined that the time to execute floating point and integer ALU instructions can be halved. However, the cost to this new approach requires a 70% increase in the time to execute memory operations. Assess the impact of this breakthrough on the individual benchmarks and on the two mixtures. [15 points]
5. Another breakthrough, independent of the above one, is reported for all add operations (floating and integer) where the speed of just those operations has been increased by a certain factor. Unfortunately due to errant internal mail delivery problem the CEO of the company does not have the exact factor of improvement. Since the CEO requires an estimate of the maximal benefit of such an operation he asks his technical wizard, you, to outline a best-case-worst-case scenario based on the two mixtures. In the worst-case assume no improvement (but assume that it does not do any worse than the existing system). [15 points]
6. It is determined that three of the PISA-big addressing modes, offset global pointer, offset stack pointer and offset frame pointer can be replaced. Instructions using special purpose registers containing the frame pointer, stack pointer and global data pointer can be replaced by other instructions using a more intelligent compiler. In 85% of the cases the replacement can be with a single displacement addressing mode instruction (basically the frame, stack, and global data pointers are in a general purpose register). In 15% of the cases one needs two instructions (one to load the corresponding pointer, the second to access the memory reference via displacement mode). The additional space gained by doing this speeds up the average time to execute an instruction from 1 cycle per instruction to 0.90 cycles per instruction.
Being the company tech-wiz it is your job to determine which configuration is faster for each of the two benchmark mixtures. [20 points]

The following materials need to be turned in hardcopy.

1. Listing of the simulator runs (one for each benchmark program).
2. Detailed work as part of the solutions for above questions.

Some Tips, prepared by one of the earlier graders for this course.

A few people have been asking about how to (conveniently) get the output from the simulator. From my experience with the simulator, the simulated program's output is written to stdout while the simulator output (i.e. sim-safe, sim-cache, sim-fast, sim-profile, etc.) is written to stderr. Since what we're interested in is the stderr stuff (it contains all the statistics, etc), I've developed the following technique (I'm only a partial Unix geek, so if you know a better way to do this, let me know):

1. Write a script to run the simulator:

```
#!/usr/local/bin/tcsh
{simulator-name options benchmark1} > bm1.output
{simulator-name options benchmark2} > bm2.output
{simulator-name options benchmark3} > bm3.output
{simulator-name options benchmark4} > bm4.output
```

The stuff in the curly braces should be replaced by the simulator you chose in problem 1 and the options you need to get the correct info (problem 2 & 3), and the stuff from the README file.

2. Make the script executable: `chmod +x MyScript`
3. Run the script: `your username~] ./MyScript >& MyResults`
4. At this point, the simulator results (i.e. the stderr stuff) should be in the file `MyResults`. This is the data you need to do the lab.

Also, to make things easier, you should copy all of the benchmark files to your local directory and make sure you have write permissions (i.e. `chmod u+w *`). The reason is that some of the simulated programs require write permissions to some other files in that directory. The README file doesn't make that clear. If you don't have write permissions, the simulated program will exit early and your results will be messed up.

Reading the description on SimpleScalar is extremely helpful. Section 4 of the paper describes what each of the simulators does, so this should make it clear what the answer to #1 is.

You use the same simulator for both questions 2 and 3. For question 2, you use the option that will list all of the instructions and their

counts. For question 3, you need two options: one to give the instruction class, and one to give the address modes used.

For a given simulator, you can type "sim-name -h" to get a list of what options can be used with that simulator (you can also find these in the paper)

Some of the simulations take a few minutes to run, but none should take more than 10 minutes or so. If the program won't terminate after that (e.g it's been running for an hour and nothing has happened), make sure you've got the command line options correct. Some of the programs get data from stdin, so you have to redirect a file to that program.

This is true for compress95:

```
sim-name -myoption compress95.pisa-big < compress95.in > OUT
```

and for Anagram:

```
sim-name -myoption anagram.pisa-big words < anagram.in > OUT
```

where compress95.in and anagram.in are the re-directed files. Long story short: keep the same format as in the README file in the benchmarks directory (but change the name of the simulator), and everything should work fine.

For Question 2, you need to create a dynamic instruction mix table like in figure B.27/B.28. To do this, you need to get a count of how many times each instruction was executed during the run of the simulated program.

The simulator will spit out a list with each instruction and the # of times it was executed, but you need to group the instructions together. %so that the resulting table is similar to the one in figure B.32.

For

example, you need to group all the cond. branch instructions (BEQ, BNE, etc) into 1 row. Here again the paper mentioned above is useful -- Appendix B gives a description of what each instruction does. There may be multiple ways to classify certain instructions, so along with the table you turn in, you should also list which instructions you put in which category.

For Question 3 (and 6) if it's not clear from the output:

```
fp = frame pointer  
sp = stack pointer  
gp = global data pointer
```

For Question 3, you should have 2 tables: one for instruction class, and one for address mode. Again, there should be a 2:3:3:1 mix and a 4:2:2:1 mix