

CSE 775: Computer Architecture
Autumn 2008
Laboratory Assignment #2

Instructor: D. K. Panda

Due: Monday, Nov 17, 2008

Purpose

The purpose of this assignment is to understand some of the internals of the simplescalar simulators, learn how to use the gcc compiler for PISA-big and finally gain experience on pipelined processing unit design and its related issues. Like Lab#1, please use a SUN Solaris machine to carry out this lab.

You will be using the program `matrix.c` located in `/usr/class/cse775` directory for these experiments. This assignment consists of three objectives.

Please take a look at the TIPS indicated on the last page of this Lab assignment.

1. **Objective 1: Modify `sim-outorder.c`:** Create a directory within your home directory space called `LAB2`. Within this space copy the files in the directory:

`/usr/class/cse775/newsim/simplesim - 2.0`

The modifications you will need to make to this file (`sim-outorder.c`) are so that you can add functionality which will allow you to specify the latency for a particular functional unit (e.g. fp adders). The current implementation of `sim-outorder` does not allow you to specify this off the command line interface. You can either choose to make the corresponding modification each time and recompile the simulator, or you can choose to make the modification in such a way that you can pass the latency arguments to the command-line interface. Which way you do it is immaterial. Note that the first thing you will need to do is to identify where the latency parameters for the functional units are specified. Please identify any changes to the source you make and describe them for full points. You will need to read through Objective 3 in order to state all the changes you made to the source code. [15 points]

2. **Objective 2: Compile `matrix.c`:** This objective is an easy one. Basically you will have to copy `matrix.c` to the above working directory and then compile it using the appropriate compiler located at: `/usr/class/cse775/newsim/bin`. Part of this objective is to figure out which of the several programs available in that location is the compiler you will need. An easy way to figure out if you have got the correct compiler is to run the simulator on the file that is produced to see if it works. [5 points]
3. **Objective 3: Understand various pipeline-related design tradeoffs** The overall objective of this lab is to understand the tradeoffs between different optimizations for this simple benchmark (`matrix.c`). Note that for all experiments below you will need to ensure that the simulator is in the `issue:inorder = true` and `issue:wrongpath = false` mode.

- **Experiment 1:**

You are the newest whiz-kid architect hired by your company. Your first job is to evaluate the following hardware configuration on the given program. The configuration is four fully pipelined FP adder/ subtractor with 3 clock cycles latency, one fully pipelined FP multiplier with 4 clock cycles latency, and one unpipelined divider with 22 clock cycles latency. Use the default latencies for the integer units but configure it so that you have only one of each integer unit. Report the average stall cycles per instruction for this environment. [5 points]

- **Experiment 2:**

You have told your boss that with your logic wizardry you have added enough space on chip for another floating point unit (FPU: one of adder/subtractor, multiplier, divider). Note: to ensure that you have two multipliers and one divider, you will have to make changes to the simulator source. You are commissioned to comparatively evaluate these options and evaluate their impact on overall performance. Which would you choose and why? [10 points]

- **Experiment 3:** While conducting the previous evaluation you find out that you can reduce the latency of the adder to 2 clock cycles. You also find that you can further **either** reduce the latency of the multiplier to three clocks cycles *or* the divider latency to eighteen clock cycles. You are now asked to evaluate which of these six options (remember you still can choose which FPU to duplicate) is best and why in a manner similar to the previous set of experiments? Use the results from the previous experiment as a basis for comparison. [20 points]

- **Experiment 4:**

Going back in time when you conducted experiment 1, you have discovered that you can partially pipeline the divide unit into two roughly equal stages. The latency remains constant (w.r.t the above is at 22) but the initiation interval is reduced. Compare the results you obtained against the results obtained in experiment 1 and identify the impact of this optimization. [5 points]

- **Experiment 5:** Change the the value assigned to the *size* variable in *matrix.c* to 32 and recompile and execute. Report results for the above four experiments as above. Are any of the results unexpected? [40 points]

You will need to submit a report answering all the questions. Use tables where appropriate and when comparing amongst different strategies.

TIPS collected from earlier offerings of the course

1. *How to calculate average stall cycles per instruction:* Unfortunately, the simulator does not give you this information outright—you have to compute it yourself. Basically you assume that, if there were no stalls, there would be one cycle per instruction. The simulator gives you the cycles per instruction (sim_CPI) for the program in its output – this includes cycles that are stalls. From this it should be trivial to compute the stall cycles per instruction.
2. *Separating out FP multiplier and divider functionalities:* The simulator has the FPU multiply and divide functionality combined onto 1 single unit; the students need to split this unit into two in order to do the experiments. However, there is no need to split the integer mult/div unit at all since the lab doesn't pertain to the integer unit at all.
3. *Adding more FP units are not giving any benefits:* You may see that adding more FP units may not lead to any benefits in some cases. It is OK. If you take a look at the assembly code (.s file) being generated and try to find out the part of the code reflecting the innermost floating-point computation of the matrix.c program, you will be able to find the answer yourself and explain the behavior.