LSU EE 7700-2 Homework 2 Due: 27 February 2008

To complete this assignment the PSE (pronounced see) visualization program will be used to analyze the execution of a SPARC v8plus program (our lighting demo). Use your class Linux accounts to complete this assignment and to avoid sluggishness, log in directly rather than remotely. (PSE is sluggish but still usable from a remote connection via Cox cable modem service, at least at my node on College Drive.)

For instructions on using PSE visit the EE 4720 procedures page, `http://www.ece.lsu.edu/ee4720/proc.html`*, and look for the PSE instructions.*

For documentation on the SPARC instructions see `http://www.ece.lsu.edu/ee4720/doc/JPS1-R1.0.4-Common-pub.pdf`*.*

Datasets, source code, and assembly code for the timing analysis versions of the demo-4-lighting.cc have been placed in `/home/classes/ee4720/com/gp/08/hw2` on the ECE Linux/Sun filesystem.

To view a dataset you could issue a command like.

```
pse /home/classes/ee4720/com/gp/08/hw2/demo-4-gcc43-4way.ds &
```

To save typing you might create a symbolic link to the dataset directory and use that:

```
 cd ~                                          # Or place link elsewhere.
 ln -s /home/classes/ee4720/com/gp/08/hw2      # Do this just once.
 pse hw2/demo-4-gcc-4way.ds &                  # Run PSE in background.
```

All datasets are for simulations of an aggressive dynamically scheduled SPARC v8plus implementation.

`demo-4-gcc43.4way.ds`: Four-way superscalar, 64-entry reorder buffer, four floating-point units, and predict one block (branch) per cycle and fetch two non-contiguous regions per cycle.

`demo-4-gcc43.4way512rob.ds`: Four-way superscalar, 512-entry reorder, other parameters same as 4way.ds.

`demo-4-gcc43.16way.ds`: Sixteen-way superscalar, 1024-entry reorder buffer, 16 floating-point units, can predict three blocks (branches) per cycle and fetch up to four non-contiguous sets of instructions in a cycle.

The source code and assembler are in the hw2 directory.

Problem 1: Locate the loop that performs lighting calculations.

(*a*) Provide the address of an early instruction that does lighting calculations.

(*b*) How many instructions are executed per vertex (for lighting, of course)?

(*c*) What is the execution rate on the 4-way, 64-entry ROB system in vertices per cycle when there is one L1 cache miss (most iterations are like this)? (Since it will be less than one, this number is best given as a fraction, say $\frac{1}{123}$, where 123 is the time to light one vertex.)

(*d*) Estimate the minimum reorder buffer size needed to sustain a 4 IPC execution rate on a 4-way system when there is one cache miss. (To help answer the question look at the 512-entry ROB dataset.)

(*e*) Most (if not all) iterations suffer an L1 cache miss. Would prefetch help?

(*f*) Show why prefetch is easy for this loop (though not necessarily helpful). In your answer show the instructions that are currently present and how a prefetch instruction could be inserted.

Problem 2: Locate the inner loop of the rasterization loop nest in demo-4-lighting and find a place in execution where that inner loop executes for a large number of iterations. Note that the inner loop body sometimes does and sometimes does not update the frame buffer.

A large part of this loop body is spent preparing the color value to be written, starting with the red, green, and blue components expressed as floating point numbers. Suggest (or search for) new instructions that can perform the task faster. (This is a prime application of so-called multimedia instructions.)

For your answer you can make up credible instructions or choose some from a real ISA, such as SPARC VIS instructions.

Problem 3: Continuing to look at the rasterization part of execution, locate the branch mispredictions.

(*a*) Estimate the impact of branch mispredictions on the three system. Do so by examining the ROB plot. In your answer provide the segment numbers.

(*b*) Some of these branches resolve early, some resolve late. What part of the code is responsible for the late-resolving branches?

(*c*) Could those late-resolving branches be avoided? (Look at the code from the `clampi` routine.)