ECSE 425: Computer Organization and Architecture
Deliverable 5: Optimized Processor

Due April 11th, 2017, 11:59 PM

For this deliverable, you will optimize your processor using some of the techniques covered in the lectures. You will evaluate the effect of your optimization by simulating the performance of your processor under different circumstances, as determined by the particular optimization(s) you choose to explore.

Change Log

- 21-Mar Initial revision.

Overview

Your tasks for this deliverable are:

1. Select an optimization; send email to the course instructors to confirm your choice. Include sufficient detail to make it possible for us to judge the design complexity of your choice (e.g., which branch predictors, or cache organizations, you will compare).
2. Conduct your VHDL design work. Be sure to keep your git repository up to date.
3. Perform your evaluation. You are expected to perform 12 distinct experiments. For example, if you are comparing (a) a system without caches and (b) a system with caches, six benchmarks must be evaluated for each. If you additional evaluate (c) a system with associative caches, four benchmarks must be evaluated for each.
4. Report the result of your experiments in the final report. The specific expectations for this document will be provided separately.

Possible Optimizations

Your first task is to pick an optimization or optimizations to implement and evaluate. Here is a non-exhaustive list of optimizations you might consider implementing.

Caching: Instead of using separate memories for instructions and data, use a single main memory with an instruction cache and data cache. You will need to create a simple arbiter to coordinate access to the main memory by the two caches. It would make sense to set the delay of your memory model to a suitably high number of clock cycles, to simulate the effect of a miss penalty. You could also use the CACTI tool to optimize the size and associativity of your cache.

Branch prediction: Implement a branch target buffer for branch prediction in IF, 1-bit, 2-bit, tournament predictors, or go even deeper and use a neural network to predict branches!

Multiple-issue: For this optimization, you could implement either static scheduling or dynamic scheduling. If you go with dynamic scheduling, you could implement the Tomasulo architecture covered in class or a more modern architecture.

SIMD: You could implement arithmetic instructions to operate on multiple words in parallel (or four 8-bit wide operands, to save memory bandwidth).

Please email us a description of the optimization you would like to implement so we can judge whether it is of appropriate difficulty.

Benchmarking

It does not suffice to point to your optimized processor and say, "it has two integer units, therefore it will go fast!" You need to perform some kind of benchmarking to measure how much faster code running on the optimized processor will go as a result of the optimization. For example, the processor you implemented for PD4 is not well suited for linear algebra operations, such as matrix multiplication. You could add a custom SIMD instruction for computing dot products and compare the number of clock cycles required to multiply two matrices for the original processor and the optimized processor for matrices of different sizes. Remember, 12 distinct experiments are required: expand your set of assembly programs to test the performance of your processor in different situations. What assembly produces the best results? What produces the worst?

Grading

The instructional staff will test and grade your project using the same procedure as for PD4. You should include a `testbench.tcl` similar to the one you provided for the baseline processor. We will run the same test programs on your optimized processor, and it should of course produce the same output files, which we will check for correctness.

As before, clearly document your processor, indicating where you've made changes to address problems with your PD4 submission, and where you've added optimizations. Select a configuration for us to perform our tests with, but clearly indicate how we might change the configuration, if applicable.

Hand In Procedure

Hand in, via MyCourses, in a single ZIP file:

- `testbench.tcl`
- All VHDL files required to simulate your processor.
- All code required to reproduce your benchmark results, along with instructions explaining how to compile and run your code.

As for PD4, please indicate how we can find your GitHub repository. You can invite us to a private repository, or include the link to your public Github repo in your submission comments.