Out-of-Order Execution

Exercise #6

Due 5:00pm Friday, Dec. 7, 2001

This exercise is meant to help you better understand the implementation of out-of-order issue and register renaming

Implementation

In this exercise, you will implement and evaluate a simplified out-of-order processor. It will be based on the simple, single-issue pipeline from Exercise 3 (pipelining, with forwarding, caches, and branch prediction).

You MUST use the reference simulator we provide. It is available in **~cs654/fall2001/assign6/assign6.tar.gz**. For automated grading purposes, you MUST name your simulator **sim-out_of_order.c.**

NOTES:

1. Please have your trace generation code be a command-line parameter (i.e. controlled by the verbose parameter). We will be selectively viewing the traces on some of the longer tests. Also, make sure when you submit your code that any INTERACTIVE debug code, such as scanf()'s/printf()'s or the like, has been commented out.
2. Comment your code clearly.

In this assignment, we will assume perfect branch prediction (all control-flow directions and targets are predicted omnisciently), a perfect instruction cache (*i.e.,* it never misses), single-cycle execution latency for all instructions, and a variable data-cache miss penalty (please see Evaluation section). Cache misses are the only source of variable latency.

Please implement the following:

- Perfect branch prediction (this is most easily done by using the reverse-order traversal of the stages to resolve branch directions/targets in decode and feed these to the fetch stage)
- Single level data cache. (See "evaluation" below for configurations.)
- An instruction window between ID and EX. Its size should be a command-line parameter. This window will serve as both an active list and an issue queue, so it will contain a mixture of instructions that have completed execution and are waiting to commit, instructions in execution, and instructions waiting to issue. (See below for sizes.)
- In-order commit; one instruction can be committed per cycle
- Register renaming: speculative results are stored in the instruction window and written to the architectural register file at commit.
- Out-of-order issue; one instruction can be issued every cycle. If any instruction in the instruction window has its operands satisfied, the oldest (in terms of program order) one should issue.

Evaluation

Write a report discussing the results of a number of experiments, to be discussed further in this section.

Baseline for comparison: The baseline for comparison is the unmodified in-order pipeline **sim-pipebase.c**, with a data cache configured as specified in each part. Please assume the following for your baseline:

- Perfect branch prediction
- Perfect instruction cache
- Single-cycle execution latency for all instructions

In your report, please describe your out-of-order implementation thoroughly. Then describe the performance relative to the baseline, in-order simulator as a function of the following 3 categories of experiments:

- Instruction window size (from 1 entry to 64 entries in steps of 2X, *i.e.* 1, 2, 4, 8, 16 entries…).
- Data cache latency (from perfect D-cache, *i.e.* no miss penalty, to 64 cycle miss penalty in steps of 2X, *i.e.*, 0, 1, 2, 4,…); use an 8 KB, direct-mapped cache with 32B lines.
- Data cache associativity (from 1-way to 8-way); use a miss penalty of 8 cycles and a total cache size of 8 KB with 32B lines.

These should be done as separate experiments. Use graphs as appropriate.

Also please perform the following measurement:

- For a window size of 64 entries and an 8KB, direct-mapped cache with a miss penalty of 8 cycles, measure the average number of instructions in the instruction window that are:
  - completed and waiting to commit
  - ready for execution
  - waiting for operands

Use gcc as your benchmark. As in the previous assignment, run your simulations for 250 million instructions after fast-forwarding for 300 million instructions.

As usual, please turn your report into the CS 654 folder in Brenda Perkins' office Olsson 204, and send e-mail to cs654 with a pointer to your code.

Grading Criteria

Your report will be graded on the following general criteria:

- Thoroughness and organization (description of implementation, methodology, experimental parameter, etc.).
- Clarity and accuracy of description of implementation
- Clarity of graphs
- Clarity and correctness of analysis
- Insights/inferences from the experiments

Your code will be graded on the following general criteria:

- Clear and descriptive commenting of any added sections of code
- Code compiles in our simulation framework
- Code behaves as you describe in the report for any tests that we give it and according to the behavior outlined in this document.