EE108B
Winter 2012-13

Prof. Kozyrakis

EE 108B Lab Assignment #4
Caches
Due: Thursday, March 14, 2013

1. Introduction

At this point you have created a single-cycle microprocessor and added pipelining to it. Up to now we have modeled memory accesses simply as a direct access to RAM that can occur within one clock cycle, and you have been working with relatively small data sets. In Lab 4, we will add one level of hardware cache to the memory stage of the pipeline. This cache will be between the processor and the main memory, which we will give an artificially high latency to simulate the characteristics of a real computer system, where the memory is usually far from the CPU and takes many cycles to access. Your task in lab 4 is to implement this cache.

2. Requirements

You must implement a direct-mapped, write-through, no-write allocate data cache in your microprocessor. You may implement more complex caches for extra credit.

3. Implementation Details

It is up to you to decide the associativity and replacement policy of the cache. The cache you implement may not exceed 2 kilobytes in size, (16,384 bits), including the data, tag, and valid bit storage elements that are already declared in the starter code (9664 bits). You may adjust or remove any of these storage elements, but the total size of your final implementation must respect the 2kB limit.

The cache and main memory are set up to use a 4-word block size. Reads and writes to main memory will occur in 4-word blocks and will be delayed by an adjustable number of cycles. Read src/dataram.v to understand how the memory module works.

The only file you should need to edit in this lab is src/cache.v. The starter code contains a very basic implementation, which essentially sends all cache accesses directly to main memory. You should remove this code as you implement your own cache.

4. Submission Requirements

You will demonstrate your lab simulations and synthesis to the TA during office hours on the due date (or at an alternate time that you must arrange in advance).

You must submit your implementation and lab report to the drop-off box on CCNet by 11:59PM on the due date. Please refer to the lab report guidelines handout for what must be in your report.

In addition to your lab report, you must include the following in your submission:

1. The whole project directory with your modifications (the Makefile should still work).
2. All instruction ROMs you wrote.

5. Extensions

As we have discussed in lecture, more advanced caches can exhibit better performance for common access patterns.

You may implement either or both of the following extensions for extra credit:

1. Make your cache n-way set associative with LRU replacement. (You may choose n to be any number as long as your final design respects the 2kB size limit.)
2. Make your cache write-back, write-allocate.

These two extensions do not depend on one another at a high level, but they may have complex interactions in your implementation. No points will be awarded for extensions if the basic caching functionality does not work. We strongly suggest starting with a direct-mapped, write-through cache and saving it before starting to work on extensions.

6. Additional Questions

Please answer the following questions in your lab report (make a clearly labeled additional section in your report):

1. What kinds of memory access patterns will your cache perform well on? What kinds will exhibit poor performance?
2. Will your implementation of Pong have good cache performance on your implementation?

7. Grading Rubric

Lab Report:
5 points – following report guidelines (no missing sections, title page, etc.)
10 points – thorough design discussion
10 points – answers to additional questions

Code:
5 points – turning in your code
45 points – passing automated tests

Demonstration:
25 points – To be determined.

Extensions:
+5 points – n-way set associative
+5 points – write-back, write-allocate

Total: 100 (110 possible with extensions)