CSCE 312: Computer Organization - Final Project

Texas A&M University, Spring 2020

Blaine Britton

Clayton Stuhrenberg

Date: 04/30/2020

Clayton Stuhrenberg, Blaine Britton 4/30/2020

Final Project Report

Introduction:

The goal of this project was to create a cache simulator to depict memory transfer in a modern computing system. The simulator is coded in C++11 and makes use of the STL library data structures, additionally, we made our own custom data structures to represent RAM and cache respectively. The code can be compiled for both Unix and Windows systems. The body of this report starts by discussing the first design of the program, then the second design. At the end of the paper is the conclusion, where final thoughts, challenges, and possible improvements, are addressed.

Design Overview:

The first design of the program featured RAM built via a vector of integer values. The RAM had several constructors and setter methods. A notable feature of the RAM was that it did not initialize the vector with data until the initRAM() method was called. This was added so that a file could be passed to the RAM after it had been constructed. Additionally, when writing or reading from RAM, an address could be passed as a hex-formatted string or an integer value.

The cache was more complex; it had several integer values to represent set, associativity, and data block sizes. Additionally, the sets, blocks, and data were represented by a 3d vector of strings where the first 3 positions held the valid, dirty bits, and the tag. The cache also held an instance of the RAM class inside of it to streamline the RAM/cache communication process. Doing this eliminated the need for a mediator class to conduct communication.

Halfway through the project, we decided that two design choices we had made were complicating development. The first one was that the 3d vector in the cache needed to change to a more efficient data structure, and the second was that the RAM needed to use strings instead of integer values.

The RAM was converted to use strings easily and retained all previous functionality. The cache, however, needed a more involved redesign. We decided that we would have to make our own data structures for the set and block. Each would naturally represent their own pieces in the cache. Set had a vector of blocks and algorithms to handle LFU and LRU policies. Additionally, it was indexable for quick access to the blocks. Block had a string vector to hold data and keep information about the tag and valid/dirty bits separately. It was also indexable for quick data access. Inside the cache, the sets would be stored in a vector.

Conclusion:

Challenges in the design process slowed progress, but after rethinking our approach and finding a well thought out design, development was fast and effective. Improvements can be made to our simulation, such as improving the efficiency of the LRU and LFU algorithms. However since the program is smaller-scaled, this is not a critical fix. The cacheRead and cacheWrite functions could also be broken down into smaller helper functions, which would help readability given that the two are very similar and could share helper functions.