CSCE 312: Computer Organization - Final Project

Texas A&M University, Spring 2020

Date: 04/19/2020

Student 1 name: Joseph Chieng **Student 2 name: Marcel Thio**	**Student 1 UIN: 327004707** **Student 2 UIN: 327004640**

Introduction

The purpose of this project is to build a cache simulator from scratch given an input file that can be used as Physical Memory(RAM). This report presents the procedures to create the cache, editing the cache and RAM data, and outputting the cache and RAM into new files. This reports will take an overview over each class, and an indepth look towards the input and outputs of the simulator.

Design and implementation

The driver class is used to control the cache configurations as well as to determine what input is used in the cache simulator menu. Driver was designed by reading in a file <input.txt> and stores the data inside that file into a ram object. It then initializes cache using the configurations inputted, before revealing the menu to allow the user to interact with the simulation. The RAM class creates a vector that stores each byte in the <input.txt> file. In addition, it grabs the corresponding block for the cache-read and cache-write function to cache, as well as writing the data into the specific address with cache-write. The memory-dump option is also implemented in the RAM class, which prints out the data in the RAM vector into a file “ram.txt”. The cache class creates the cache vector by using the inputs obtained during the cache configuration process. By using the given input values, as well as the cache parameter equations, the structure of the cache is built. As the least recently used(LRU) policy is an option, each block is pushed into a new vector, which keeps track of the block that was last accessed. This is done by popping the block index from the vector and pushing it back in to reset the access lineup. The cache-read function is implemented by separating the input command into two substrings. The first determines the menu choice, and the second is the address. By converting the address from hexadecimal to binary, the tag bits, the tag value, and which set the address belongs in, is obtained. It then checks the cache for the tag value, as well as whether the valid bit is true, before determining whether it is a cache miss or cache hit. If it ends up being a cache hit, then it outputs the corresponding information, as well as resetting the LRU vector for that block. If it is a cache-miss, then it replaces the block either through LRU or by random, depending on the cache configuration. The cache-write is similar to cache-read, except if the first substring of the input command is determined to be “cache-write”, then the second substring splits into two more substrings, to differentiate between the address accessing, and the data replacing that address. If it is a cache-hit, then it will write the data into the corresponding address in cache. Depending on the cache configuration for the write hit policy, it may also write the data into RAM. If it is a cache-miss, then it will either load the block from RAM and write it into cache, or simply write the data into RAM, depending on the write miss policy. Cache-flush is implemented by simply writing the dirty bits to ram and then deleting all the data, replacing them with zeros as placeholders. Cache-dump will simply grab the cache vector and print it into a file “cache.txt”. Both the RAM class and the cache class are capable of displaying their respective vectors by simply iterating through it and printing out each line. Frequency.h was used specifically for LFU and was initialized in the cache configuration.

Conclusion

The overall programming process was simple, but tedious. Most of the code was straightforward, but implementing different cache configurations resulted in some code either being changed, or forced if statements to account for each option. Improvements to the code would be to remove the parts that is simply duplicated for another configuration, rather than creating certain variables that can allow the same code to work for different configurations