CSCE 312: Computer Organization - Final Project

Texas A&M University, Spring 2020

Date: 04/30/2020

Student 1 name: Ashok Meyyappan **Student 2 name: Mahir Pirmohammad**	**Student 1 UIN: 827001687** **Student 2 UIN: 927006039**

Introduction

The purpose of the project was to get familiar with cache memory organization and configuration, and replacement policies, in which a minimized command-line based cache simulator was built using the Python programming language. The simulator uses the cache and physical memory, assuming the use of an 8-bit address. The program first initialized the physical memory in the input file, then the cache was configured with user-inputted numbers. The program finally requests the user to pick one of the provided cache simulation options and performs the operation.

Design and implementation

Step 1. This step of the program involves the initialization of the physical memory. During compilation the program file and the input file is called for the initialization. The input file is read line by line, and added to the RAM. This function uses the global count variable as a key in the RAM address in hex form to contain the memory read in from the file. Following completion, a message is given to the user of a successful initialization.

Step 2. This step of the program involves the configuration of the cache with the use of user input. This part of the program requests the user to input numbers for certain conditions for proper configuration. Once these values are requested from the user and received, the program performs mathematical computations to initialize the cache before exiting the function. Following completion and prior to the computations, a message is given to the user of a successful configuration.

Step 3. This step of the program involves the simulation of the cache with the use of multiple commands. The simulation function provides the user with options for various commands. Once the user types in a command, the code goes from the cachesimulator.py file to the defined function of the command in the menu.py file. For the cache-read, the function first detects whether there's a cache hit/miss. After converting the address into a binary string, the index and tag values are printed. The hit/miss is decided based on the tag from the address and the valid bit. If there's a hit, the program prints the provided eviction line and the RAM address. Then it also prints the data by accessing the cache. Based on the line hit, the LFU global variable updates accordingly. If there's a miss, the values are printed after checking multiple cases for the eviction line and RAM address with the access of the cache. For the cache-write, the program is very similar to the cache-read, except that when there's a hit, the program either does a write-through or a write-back. When there's a miss, the program either does a write-allocate or a no write-allocate. For the cache-flush, the program goes through a looping structure and sets all indices to 0 or 00. For cache-view, the program converts configuration cache numbers for choosing values to their respective meaning for printing, and prints the cache list structure. For the memory-view, the program simply prints the information on the memory. For the cache- and memory-dump, the program writes the cache data into cache.txt and data in the RAM data structure into ram.txt.

Conclusion

The experience writing this program was dealing with cache and physical memory, and reading/writing cache. It provided experience in dealing with cache hits/misses, and dealing with the additional cases to perform specific operations depending on those cases. To summarize the code, it simulates the cache based on the command provided by the user. The challenges that were encountered during the work were the accessing/reading of the wrong cache during the earlier stages of the project, and needing to add additional cases during the testing stages of the program. Possible improvements that could be added are coded improvisations that would enhance the running time and reduce the amount of if-statements.