UNIVERSITY OF CENTRAL FLORIDA

EEL 4768: Computer Architecture ECE Department, UCF

Project 1

Due: Nov 4 2018

Project Objective:

Implementing a flexible cache simulator.

Rules and Regulations:

1. The work must be done individually.

2. Any form of cheating and plagiarism such as sharing of your code or using an available code is reported to the school for consequences and results in getting zero.

3. You can communicate with the instructor and the TA through WebCourses regarding your issues.

4. You can use any high-level programming language for implementation.

5. Implement your simulator in either Windows or Linux environment.

6. Your project report should be comprehensive and include the demanded results and their corresponding analyses and discussion.

7. Your simulator outputs should nearly match the reference outputs.

8. All your source codes along with the executable file must be delivered.

9. It is recommended to create a "**Makefile**" or the compilation command you used.

10. The configurations must be passed to your executable as "**arguments**".

11. You will be provided with traces of memory access for different applications.

Project Description:

In this project, you need to implement a simple cache simulator that takes as an input the configurations of the cache to simulate, such as: size, associativity and replacement policy. When you run you simulator, you need to additionally provide the path of the **trace file** that includes the memory accesses.

Your simulator will parse the trace file, which looks like:

R 0x2356257

W 0x257777

Each line consists of two parts, the operation type (read or write) and **byte address** in hexadecimal. After reading each line, the simulator will simulate the impact of that access on the cache state, e.g., the LRU state of the accessed set and the current valid blocks in the set. Your simulator needs to maintain information such as hits, misses and other useful statistics throughout the whole run.

In this project, you need to implement two different cache replacement policies: LRU and FIFO. In LRU, the least-recently-used element gets evicted, whereas in FIFO, the element that was inserted the earliest gets evicted.

Implementation hint: allocate your cache as a 2D array, where each row is a set. On each item of the array keep track of information like the tag of the data in this block. You can create multiple instances of such a 2D array for different purposes; for example, you can create another 2D array to track the LRU position of the corresponding block in the LRU stack of the set.

Assume 64B block size for all configurations

Inputs to Simulator

The name of your executable should be SIM, and your simulator should take inputs as following:

./SIM <CACHE_SIZE> <ASSOC> <REPLACEMENT> <WB> <TRACE_FILE>

<CACHE_SIZE> is the size of the simulated cache in bytes

<ASSOC> is the associativity

<REPLACEMENT> replacement policy: 0 means LRU, 1 means FIFO

<WB> Write-back policy: 0 means write-through, 1 means write-back

<TRACE_FILE> trace file name with full path

Example:

./SIM 32768 8 1 1 /home/TRACES/MCF.t

This will simulate a 32KB write-back cache with 8-way associativity and FIFO replacement policy. The memory trace will be read from /home/TRACES/MCF.t

Note: the trace file will contain addresses that can be for 64-bit system, so you might need data types that are large enough to read them correctly and bookkeep the metadata in your simulator. For example, if the tag is 9 bytes and you allocate your tag array bookkeeping array as an array of integers, you will not be able to store the whole 9 bytes; integer is only 4 bytes. Accordingly, use data types such as `long long int` and its equivalents in other languages.

Output from Simulator:

The following outputs are expected from your simulator:

a. The total miss ratio for L1 cache
b. The # writes to memory
c. The # reads from memory

Grading:

1- (20 points) A functional code that runs cleanly, i.e., showing serious efforts to implement the simulator.

2- (40 points) Your code simulates correctly the sample runs we provide and all output statistics should be within less than 0.001% error of our results. 2 tests will be provided along with the configurations and the expected output. Two tests will be not be provided to you and are used to make sure your simulator isn't tuned for our sample tests. Each test will weigh 10%.

3- (40 points) A report that contains the following:

A- Use the MiniFE and XSBench provided traces to analyze how the miss ratio of these workloads changes with cache size. Fix cache associativity at 4, write-back, replacement policy to be LRU, and vary the cache size from 8KB to 128KB in multiples of 2, i.e., 8KB, 16KB... 128KB.
B- Similar to part A, but change write policy and compare between write-back and write-through for each cache size by using the number of memory reads and writes.
C- Similar to part A, but instead of varying the cache size change the associativity. Fix the replacement policy to be LRU, cache size to be 32KB, associativity to change from 1 to 64, in multiples of 2, e.g., 1, 2, 4… 64.
D- Similar to part C, but now study the impact of replacement policy. Specifically, run the same experiments you ran in part B, but with FIFO replacement policy, write-back, then compare the results of part C and D to study the impact of replacement policy changes with associativity.

In each part of your report, plot the results and explain the reasons behinds the trend and compare the trends of MiniFE with XSBench.

Good Luck