Lab Project 2: Build a Victim Cache Simulator
Due Monday, Nov. 14 before class

Introduction

In this project, a victim cache simulator will be implemented in C/C++. The simulator will simulate a memory system that consists of a 32KB L1 direct mapped cache and a victim cache. The simulator take in several parameters that describe the system (e.g. size, replacement policy, etc.) along with a trace file describing data memory accesses for a particular program. The simulator will output the statistics (e.g. hit rate, average memory access time etc.).

The project must be completed individually.

Trace File:

Your simulator will need to take a command-line argument that specifies the trace file that will be used to compute the output statistics. The trace files used in this project are the same as project 1. The trace file will specify all the data memory accesses that occur in the sample program. Each line in the trace file will specify a new memory reference and have the following three fields:

- ***Access Type***: A single character indicating whether the access is a load ('l') or a store ('s').
- ***Address***: A 32-bit integer (in unsigned hexidecimal format) specifying the memory address that is being accessed. For example, "0xff32e100" specifies that memory address 4281524480 is accessed.
- ***Instructions since last memory access***: Indicates the number of instructions of any type that executed between since the last memory access (i.e. the one on the previous line in the trace). For example, if the 5th and 10th instructions in the program's execution are loads, and there are no memory operations between them, then the trace line for with the second load has "4" for this field.

Fields on the same line are separated by a single space. The trace files that you will need to do the simulation can be downloaded in Canvas.

Input Arguments

For the simulator, the simulator will take a set of arguments that correspond to all the necessary parameters to setup your cache and calculate the stats. They are as following:

- ***blocksize:*** Specifies the line (block) size for the L1 cache and the victim cache in bytes. This should always be non-negative power of 2 (i.e. 1, 2, 4, 8, etc).
- ***victimsize***: Specifies the total number of entries in the victim cache.

- ***Replacement Policy***: Specifies the replacement policy to be used by the victim cache. Should be either "0" for random replacement, "1" for FIFO, “2” for LRU. No other values are valid.
- ***Write allocate***: Specifies the cache policy on write misses. A value of "0" means no-write-allocate. A value of "1" means write-allocate. Any other value is invalid. ***Assume write back is used on write hits.***

For example, the following command line will run the simulator on the gcc trace and simulate a L1 (32 KB, direct-mapped, 8-byte block size), victim (8 entries, 8-byte block size) with random replacement policy, no-write-allocate.

```
%victimcache gcc.trace 8 8 0 0
```

Simulator Output:

The simulator should write to an output file file the ".out" extension, whose prefix is based on the name of the input trace file. For example, if you input "gcc.trace" then the output file should be named "gcc.trace.out". Your simulator will be calculating several statistics that will go into the output file. Much like the configuration file, the output file should contain one field per line. It should contain the following 4 lines:

- ***Total Hit Rate***: The percentage of memory ops (i.e. lines in the trace file) that were hits.
- ***Load Hit Rate***: The percentage of loads that were hits.
- ***Store Hit Rate***: The percentage of stores that were hits.
- ***Average Memory Access Time***: The average number of cycles needed to complete a memory access. (Assume hit time is 1 clock cycle and miss penalty is 100 clock cycle)

Compilation and Execution Environment:

The simulator should be written in either C or C++. It should compile and run on a TCC machine (rainbow.nmt.edu) using gcc (if written in C) or g++ (for C++).

Submission:

Report:

You will need to evaluate the effects of different parameters on the performance of the system. You will need to generate graphs comparing the performance on the given trace files. Specifically you will need to generate the following 5 graphs:

- ***Total Hit Rate vs. Number of Entries of Victim Cache***: A 2D line chart showing the total hit rate vs. number of entries (0 – 15) for each of the trace files. Assume 8B block size, FIFO, write-allocate.
- ***Average Memory Access Time vs. Number of Entries of Victim Cache***: A 2D line chart comparing this statistic for different number of entries of victim cache (0-15) for each of the trace files. Assume 8B block size, FIFO, write-allocate.
- ***Total Hit Rate vs***. ***Block size:*** A bar chart showing the total hit rate vs. block size (2, 4, 8, 16) for each of the trace files. Assume the number of entries of victim cache is 8, FIFO, write allocate.

- ***Total Hit rate vs*. *Replacement Policy:*** A bar chart showing the total hit vs. replacement policy (Random, FIFO, LRU) for each of the trace files. Assume the number of entries of victim cache is 8, 8B block size, write allocate.
- ***Total Hit rate vs*. *Write Policy:*** A bar chart showing the total hit vs. write policy (non-write-allocate, write allocate) for each of the trace files. Assume the number of entries of victim cache is 8, 8B block size, FIFO.

In your report, you should have a discussion for each graph.

A note on graphs: Graphs are supposed to make large amounts of information comprehensible very quickly. This means they must clear, uncluttered, easy to decipher, and well labeled. Perhaps the most important and most frequently broken rule for graphs is "Label your axes." Your graphs MUST have properly labeled axes, or they will receive no credit.

Executable and Source Code:

You should also turn in the source code and an executable for the cache simulator. See the note above about the compilation and execution environment. If you have any special compilation/execution notes, please include them in a README. Also, remember that your simulator should take only two parameters: the config file and the trace file (in that order). Make sure that all your source code files should have the following header filled-in appropriately.

```
/*****************************************************************
/
/       filename:  victimcachesim.c
/
/       description:  Implements the victim cache simulator.
/
/       authors:  Last, First
/
/       class:       CSE 331
/       instructor:  Zheng
/       assignment:  Lab Project #2
/
/       assigned:  Oct. 31, 2016
/       due:       Nov. 14, 2016
/
*****************************************************************/
```

A README file must be submitted with the executable and source code to explain how to compile the source code to an executable.

Tarball:

Please place your report, executable, and source code in a gzipped tarball using the following naming convention: "**lastname-cse331-F16-P2.tar.gz**" where lastname is your last name. Upload your tarball file in Canvas through the submission link.