Lab 2: Build a Cache Simulator
Due midnight of Thursday, Mar. 9

Introduction

In this project, a basic cache simulator will be implemented in C/C++. The simulator will take in several parameters that describe the desired cache (e.g. size, associativity, 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, total running time of the program etc.).

The project must be completed individually.

Configuration File

For the simulator, the configuration file is used which contains all the necessary parameters to setup your cache and calculate the stats. The layout of the configuration file has a specific format that must be adhered to. It contains 6 lines with the following order:

- ***Line size***: Specifies the line (block) size for the cache in bytes. This should always be non-negative power of 2 (i.e. 1, 2, 4, 8, etc).
- ***Associativity***: Specifies the associativity of the cache. A value of "1" implies a direct-mapped cache, while a "0" value implies fully-associative. Should always be a non-negative power of 2.
- ***Data size***: Specifies the total size of the data in the cache. This does not include the size of any overhead (such as tag size). It should be specified in KB and be a non-negative power of 2. For example, a value of "64" means a 64KB cache.
- ***Replacement Policy***: Specifies the replacement policy to be used. Should be either "0" for random replacement or "1" for FIFO. No other values are valid.
- ***Miss penalty***: Specifies the number of cycles penalized on a cache miss. May be any positive integer.
- ***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 through is used with no-write-allocate and write back is used with write-allocate***.

An example config file is shown in below. It specifies a 16KB direct-mapped cache with 8 byte blocks, an FIFO replacement policy, 100 cycle miss penalty, and following a write-allocate policy.

sample.cfg:

```
8
1
16
1
100
1
```

NOTE: The configs that you need to use for your report can be downloaded in Canvas. You need to include the results from all these configs in the report.

Trace File:

Your simulator will need to take a second command-line parameter that specifies the trace file that will be used to compute the output statistics. 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.

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 5 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.
- ***Total Run Time***: The total run time of the program, assuming that the last memory access was the last instruction of the program. The unit for this should be cycles.
- ***Average Memory Access Latency***: The average number of cycles needed to complete a memory access **(assume hit time is 1 cycle)**.

Compilation and Execution Environment:

The simulator should be written in either C or C++. It should be able to compile and run on the CSE machine using gcc (if written in C) or g++ (for C++).

Submission:

Lab Report:

Please follow the lab report format. The followings are specific requirements for this lab: The method part should include a testing plan on how to test the correctness of the programmed cache simulator. The result part needs to evaluate the provided cache configurations and their performance on the given trace files. You will need to generate graphs comparing the performance of each cache configuration on the given trace files. Specifically you will need to generate the following 2 graphs:

- ***Total Hit Rate***: A bar chart showing the total hit rate for each configuration on each input trace.
- ***Average Memory Access Latency***: A bar chart comparing this statistic on each cache configuration for each of the trace files

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:  cachesim.c
/
/       description:  Implements the cache simulator.
/
/       authors:  Last, First
/
/       class:       CSE 331
/       instructor:  Zheng
/       assignment:  Lab #2
/
/       assigned:  2/28/2023
/       due:      3/10/2023
/
*****************************************************************************/
```

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 lab report, executable, and source code in a gzipped tarball using the following naming convention: "`lastname-CSE331-S20-Lab2.tar.gz`" where `lastname` is your last name. Upload your tarball file in Canvas through the submission link.