Homework #4: Cache Simulation

Problem 1: Cache Simulator (90%)

You will design and implement a **cache simulator** that can be used to study and compare the effectiveness of various cache configurations. Your simulator will read a **memory access trace** from standard input, simulate what a cache based on certain parameters would do in response to these memory access patterns, and finally produce some summary statistics to standard output. Let's start with the file format of the memory access traces:

```
s 0x1fffff50 1
l 0x1fffff58 1
l 0x1fffff88 6
l 0x1fffff90 2
l 0x1fffff98 2
l 0x200000e0 2
l 0x200000e8 2
l 0x200000f0 2
l 0x200000f8 2
l 0x30031f10 3
s 0x3004d960 0
s 0x3004d968 1
s 0x3004caa0 1
s 0x3004d970 1
s 0x3004d980 6
l 0x30000008 1
l 0x1fffff58 4
l 0x3004d978 4
l 0x1fffff68 4
l 0x1fffff68 2
s 0x3004d980 9
l 0x30000008 1
```

As you can see, each memory access performed by a program is recorded on a separate line. There are three "fields" separated by white space. The first field is either l or s depending on whether the processor is "loading" from or "storing" to memory. The second field is a 32-bit memory address given in hexadecimal; the 0x at the beginning means "the following is hexadecimal" and is not itself part of the address. You can **ignore** the third field for this assignment.

Your cache simulator will be configured with the following cache design parameters which are given as command-line arguments (see below):

- number of sets in the cache (a positive power-of-2)
- number of blocks in each set (a positive power-of-2)
- number of bytes in each block (a positive power-of-2, at least 4)
- write-allocate or no-write-allocate
- write-through or write-back
- lru (least-recently-used), fifo, or random evictions

Note that certain combinations of these design parameters account for direct-mapped, set-associative, and fully associative caches:

- a cache with n sets of 1 block each is direct-mapped
- a cache with n sets of m blocks each is m-way set-associative
- a cache with 1 set of n blocks is fully associative

The smallest cache you must be able to simulate has 1 set with 1 block with 4 bytes; this cache can only remember a single 4-byte memory reference and nothing else; it can therefore only be beneficial if consecutive memory references in a trace go to the exact same address. **You should probably use this tiny cache for basic sanity testing.**

A few reminders about the other three parameters: The **write-allocate** parameter determines what happens for a **cache miss** during a **store**:

- for write-allocate we bring the relevant memory block into the cache *before* the store proceeds
- for no-write-allocate a cache miss during a store does *not* modify the cache

Note that this parameter interacts with the following one. The **write-through** parameter determines whether a store **always** writes to memory **immediately** or not:

- for write-through a store writes to the cache as well as to memory
- for write-back a store writes to the cache *only* and marks the block *dirty*; if the block is evicted later, it has to be written back to memory before being replaced

It doesn't make sense to combine no-write-allocate with write-back because we wouldn't be able to actually write to the cache for the store!

The last parameter is only relevant for associative caches: in direct-mapped caches there is no choice for which block to evict!

- for lru (least-recently-used) we evict the block that has not been **accessed** the longest
- for fifo (first-in-first-out) we evict the block that has been **in the cache** the longest
- for random we evict a block uniformly at random regardless of "age"

Your cache simulator should assume that loads/stores from/to the cache take **one** processor cycle; loads/stores from/to memory take **100** processor cycles for **each** 4-byte quantity that is transferred. There are plenty of things about caches in real processors that you do **not** have to simulate, for example write buffers or smart ways to fill cache blocks; implementing all the options above correctly is already somewhat challenging, so we'll leave it at that.

We expect to be able to run your simulator as follows:

./csim 256 4 16 write-allocate write-back lru <sometracefile

This would simulate a cache with 256 sets of 4 blocks each (aka a 4-way set-associative cache), with each block containing 16 bytes of memory; the cache performs write-allocate but no write-through (so it does write-back instead), and it evicts the least-recently-used block if it has to. (As an aside, note that this cache has a total size of 16384 bytes (16 kB) if we ignore the space needed for tags and other meta-information.)
After the simulation is complete, your cache simulator is expected to print the following summary information in **exactly** the format given below:

Total loads: 318197
Total stores: 197486
Load hits: 314798
Load misses: 3399
Store hits: 188250
Store misses: 9236
Total cycles: 9344483

Hints

Your simulation is only concerned with hits and misses, at no point do you need the **actual** data that's stored in the cache; that's the reason why the trace files do not contain that information in the first place.

Don't try to implement all the options right away, start by writing a simulator that can only run direct-mapped caches with write-through and no-write-allocate. Once you have that working, extend step-by-step to make the other design parameters work. Also, sanity-check your simulator frequently with simple, hand-crafted traces for which you can still derive manually what the behavior should be.

Problem 2: Best Cache? (10%)

For the second problem, you'll use the memory traces as well as your simulator to determine which cache configuration has the **best overall effectiveness**. You should take a variety of properties into account: hit rates, miss penalties, total cache size (including overhead), etc. In your report describe in detail what experiments you ran (and why!), what results you got (and how!), and what, in your opinion, is the best cache configuration of them all.