Machine Organization and Assembly Language Programming

Problem Set #6

Due: Wednesday June 1st

In this assignment you will write in the C language a simple trace-driven simulator for assessing the performance of a cache. You can do this assignment in groups of 2 if you so desire.

Trace-driven simulation is a widely used technique to assess the performance of the components of the memory hierarchy. For this assignment, the trace will consist of a string of data memory references (we will simulate only data caches) with the following format:< *operation* > < *address* > where *operation* is "1" for a read and "2" for a write, and *address* is a 32-bit byte address.
The trace ends with the pair 0 0.

The basic idea is to process each memory reference in turn. First you decompose the address in (tag,index,displacement) components. Then you check if there is a hit/miss (note that you don't need to simulate bringing data in the cache; it is sufficient to know which blocks are currently mapped in the cache). At this point, you need to record the statistics needed for the desired outputs and take appropriate actions (e.g., setting dirty bits, LRU information, replacements in case of a miss etc.).

Because the trace is short, we'll have ridiculously small caches so that you don't have only compulsory misses. The caches that you'll have to consider are:

1. Cache 1: 256 bytes, 2-way set-associative, line size 8 bytes. It uses a write-through, write-around policy.
2. Cache 2: 256 bytes, 2-way set-associative, line size 8 bytes. It uses a write-back, write-allocate policy and an LRU replacement algorithm.
3. Cache 3: 256 bytes, direct-mapped, line size 8 bytes backed up by a 2-entry victim cache using an LRU replacement algorithm. Cache 3 uses a write-back, write-allocate policy.

The output statistics that you should generate are:

- The number of data references that you processed.
- For each of the 3 cache configurations, the number of hits in the cache (in the case of Cache 3, count separately the number of hits in the cache and the number of hits in the victim cache)
- For each of the 3 cache configurations, the number of words you needed to write back. You can assume that all load and store are to words. In the write-through policy only the word being accessed is written to memory. In the write-back policy, the whole line is written back.

(don't draw any definite conclusion on the advantages/drawbacks of one configuration based on the results of this single minuscule and totally unrepresentative experiment!)

A trace, a C-skeleton for this program (of course you don't have to print each address like in the skeleton), and instructions for turnin will be available soon.

(A good warm-up would be to do Exercises 7.9 and 7.10 in your book by hand; you might also want to do these exercises by having a 2-way set-associative cache).