CS 341L Fall 2009 Lab 5b

Assigned: Wednesday, 21 October 2009

Due: 11:59pm, Tuesday 27 October 2009 by attachment as an email to labturnin.cs341l@gmail.com

You should attach your program with your last name and the lab number in the name of the file, e.g., "knockel-lab5b.c". Be sure to include the 'b' on the end.

No late assignments will be accepted unless circumstances warrant an extension for the whole class (e.g., server is down all weekend, etc.). You are expected to do your own work as an individual effort, copying the code of others is considered cheating.

All code must be implemented in C. No C++.

The purpose of this lab is to learn about how caches work and build a cache simulator for lab 5c, where we will explore some of the tradeoffs.

First, add the following options and arguments to your cache simulator:

-h N The hit time—this will be the hit time in clock cycles, must be a positive integer

-m N The miss penalty—this will be the miss penalty in cycles, must be a positive integer

-s N The store miss penalty—this will be the extra miss penalty in cycles for stores, i.e., you'll add this much more penalty whenever you evict a cache line with its dirty bit set, must be a positive integer

Your cache simulator should always assume an LRU replacement policy and a write-back scheme.

The file format for the cache trace file has changed to distinguish between loads and stores, and now looks like this:

```
lw 0x1ffffdb0
lw 0x300088a0
sw 0x1ffffd00
...
```

The first two characters of each line indicate whether the cache access is a load (lw) or store (sw), and then the address is separated by a space.

Each cache line in your simulated cache should have a dirty bit. You should simulate the way a typical cache with the given parameters would operate. For writes to a cache line, hit vs. miss and the eviction strategy remain the same but you should set the dirty bit. When a cache miss (either a load or a store) causes eviction of a cache line with its dirty bit set you should add an extra penalty based on the store miss penalty option (on top of the regular miss penalty option which you will always penalize misses in the cache with).

Your cache simulator should do the following things in this order:

1. Print a summary of all options given (don't forget to mention LRU and write-back in your summary even though the user can't change these); and also some additional information you can readily calculate based on the user's options: the number of sets in the cache and the required size in bits for each tag.
2. Simulate the cache behavior based on the trace file given. Your cache should initially be empty with all valid bits cleared so that you will get some compulsory misses, i.e., don't “prime the cache”.
3. Print the results, including: the hit rate for all accesses, the hit rate for loads, the hit rate for stores, and the overall effective average access time (usually equal to *HitTime* + *MissRate* * *MissPenalty* but remember that, in your write-back scheme, every time you evict a dirty block you pay an extra penalty). All of these should be printed to 3 decimal places past the decimal point.

The trace files and some example executions and output will be posted on the TA blog.

Note that a real cache does not get an access every cycle, so just calculate that access times but keep in mind that this is a cache trace and not a CPU trace, so each line may be several cycles apart in reality. Also, for superscalar, some cache accesses can be hidden through ILP, but we won't simulate CPI in anyway, just average access times and hit rates. Also note that we're ignoring virtual memory and assuming the trace files are the physical addresses most caches use and not the virtual addresses the process would actually use (doesn't actually make much difference since conceptually each page is made up of several cache blocks, typically).

<u>IMPORTANT</u>: Your code should use arrays in two separate places and pointers in two separate places. It doesn't matter where, but clearly mark these four places in your C code with comments like this:

```
/*** POINTER USAGE ***/
```

or

```
/*** ARRAY USAGE ***/
```

We'll use this in lab 6.