CS/ECE 250: Homework 5

Due: Dec 1st, 2015 at 5 pm through Sakai. This homework is to be done individually.

Section-I Cache Organization

Problem 1 [10 points]: Consider a CPU that has a maximum of 4 GB of addressable main memory. This CPU has a Level 1 cache of 32 KB with a 64 byte block size. For each of the following associativities:

a. **[3 points]** Direct mapped
b. **[4 points]** 8-way set associative
c. **[3 points]** Fully associative

Show the size of the tag, index, and offset fields in a memory address. Also, compute the total number of bits needed to implement the entire cache for each of the three associativities. Assume one valid bit and refer to the table on slide 36 of the "Cache and Memory Hierarchies" lecture slides to see what needs to be included in this total.

Problem 2 [5 points]: Describe the general characteristics of a program that exhibits little temporal and little spatial locality with regard to data accesses. Provide some example code that reflects this (pseudocode is acceptable).

Problem 3 [5 points]: Describe the general characteristics of a program that exhibits high temporal locality but very little spatial locality with regard to data accesses. Provide some example code to reflect this (pseudocode is acceptable).

Problem 4 [10 points]: Find the average memory access time (*t_avg*) for a machine with:

a. **[5 points]** A single level cache with a 1 cycle hit time, a 200 cycle miss penalty, a 5% miss rate, and a 2ns clock.

b. **[5 points]** A two level cache hierarchy with a 2 cycle L1 hit time, a 10 cycle L2 hit time, a 100 cycle L2 miss penalty, a 1ns clock, a 10% L1 miss rate, and a 5% L2 miss rate.

Section-II Cache Operation

Problem 5 [20 points]:

a. **[10 points]** Assume a 128 Byte direct-mapped cache with one word (4B) block size, and LRU replacement policy. Given this sequence of memory accesses (32-bit addresses):

   5, 1, 8, 20, 3, 16, 19, 56, 12, 11, 4, 52, 5, 6, 9, 16.

   where each address references a **byte**; write down each address in binary, then show the tag and index of each reference. Then label each reference as a hit or a miss and show the final contents of the cache. Assume the cache is initially empty.

b. **[10 points]** Assume a 2 way set-associative cache with two-byte blocks, a total size of 32 bytes, and LRU replacement policy. Given this sequence of memory accesses (32-bit addresses):

   172, 44, 4, 172, 104, 32, 192, 88, 200, 16, 56,184, 52, 196, 254

   where each address references a **byte**; write down each address in binary, then show the tag and index of each reference. Then label each reference as a hit or a miss and show the final contents of the cache. Assume the cache is initially empty.

Problem 6 [50 points] Write a simulator of a single-level cache. You may use Java (or C for up to 10 extra credit points) to do this, but are limited to **one code file** (so use inner classes in Java if you want). The simulator, called **Cachesim**, takes the following input parameters on the command line:

1. Name of the file holding the loads and stores
2. Cache size (not including tags or valid bits) in KB
3. Associativity
4. Block size in bytes

For example,

```
java Cachesim tracefile 1024 4 32
```
(for Java)

```
./cachesim tracefile 1024 4 32
```
(for C)

should simulate a cache that is 1024 KB (=1 MB), 4-way set-associative, has 32-byte blocks. This cache will process the loads and stores from the file named tracefile.

The replacement policy is always LRU.

Important Assumptions: Addresses are 24-bits (3 bytes), and thus addresses range from 0 to 2^{24}-1 (i.e., 16MB of address space). The machine is byte-addressed and big-endian. The cache size, associativity, block size, and access size will all be powers of 2. Cache size will be no larger than 2MB, block size will be no larger than 64B, and no access will be larger than the block size. **No cache access will span multiple blocks** (i.e., each cache access fits within a single block).

All cache blocks are initially invalid. All cache misses are satisfied by the main memory (and you must track the values written through to memory in case they are subsequently loaded). If a block has never been written before, then its value in main memory is zero. The cache is **write-back** and **write-allocate**.

The trace file will be in the following format. There will be some number of lines. Each line will specify a single load or store, the 24-bit address that is being accessed (in base-16), the size of the access in bytes, and the value to be written if the access is a store (in base-16).

For example:

```
store 0x1a25bb 2 c77e
load 0xd5316f 4
```

Your simulator must produce the following output. For every access, it must print out what kind of access it is (load or store), what address it's accessing (in base-16), and whether it is a hit or a miss. For each load, it must print out the value that is loaded (possibly after satisfying the miss from memory). The output format must be as follows (to help the grader):

For example:

```
store 0x1a25bb miss
load 0xd5e16f hit 7d2f13ac
```

Notes:

- References to KB, MB, GB, etc. in this homework are used to refer to their 2^x meanings rather than 10^x. This means that 1 KB = 1024 B.
- Grading will on the Teer machines, so it is critical that your **code works on the Teer machines**. Please be sure to test your code running from the terminal instead of only from an IDE like Eclipse.
    - Currently, only Java 6 is installed, but a request has been made to update those machines to Java 8 (still pending). Please make sure your programs run on Java 6 until further announcement.
- Please make sure your project is in the default package (i.e. does not use package statements). In Eclipse, if you don't enter a package name when you create your classes, your classes should be in the default package by default.
- If you want to use classes, please use inner classes to keep all your code within one code file.

Testing Cachesim:

You will be provided with some sample test files with sample outputs. Please use these to make sure your code is working. To run these test files:

1. Compile your code and place the test files in the same directory
2. Run the following command from the terminal

    `java Cachesim tracefile_1_64_16 1 64 16 > out` (for Java)

    `./cachesim tracefile_1_64_16 1 64 16 > out` (for C)

    Notice that the numbers given after tracefile are the command line arguments that should be used.
3. Now you should have a file named out which has the output or your simulator. Compare this (diff or manually or with another tool) against the corresponding key. You may use the –b option to ignore spacing changes.

    ```
    diff -b out key_1_64_16
    ```
4. Please make sure that your formatting is the same as the key files, including capitalization and the inclusion of leading 0s if needed.

Submission Instructions:

Please submit a single PDF document titled **hw5-netid.pdf** with your answers. Please submit a **single code file** (.c or .java file). Do not zip any of your submission files, just upload directly as individual attachments.

Please title your code file **Cachesim.java** or **cachesim.c**. Please also include a README.txt file to note any bugs or special notes about your code. If you believe your code works, just note that in the README.