

Homework #5 – Caches and Memory

Due Monday, April 18, at 5:00pm

You must do all work individually, and you must submit your work electronically via Gradescope.

- For short-answer questions, submit your answers in PDF format as a file called <NetID>-hw5.pdf. **Word documents will not be accepted.**
- For the programming question, submit your source file using the filename specified in the question.

All submitted programs will be tested for suspicious similarities to other programs (from this semester and recent semesters), and the test will uncover cheating, even if it is “hidden.”

1) [5 points] Why are write-through caches usually also write-no-allocate?

2) [5] Your L1 data cache has an access latency of 2ns, and your L2 cache has an access latency of 10ns. Assume that 90% of your L1 accesses are hits, and assume that 100% of your L2 accesses are hits. Assume (unrealistically) that all accesses are loads (i.e., there are no stores). What is the average memory latency as seen by the processor core? Show your work.

3) [20] You have a 64-bit machine and you bought 4GB of physical memory. Pages are 32KB. Remember that $1K=2^{10}$, $1M=2^{20}$, and $1G=2^{30}$.

[2] (a) How many virtual pages do you have per process?

[2] (b) How many physical pages do you have?

[2] (c) In the translation from a virtual address to a physical address, how many bits of VPN are you mapping to how many bits of PPN?

[2] (d) How big does a page table entry (PTE) need to be to hold just a single PPN? (Real PTEs hold some other stuff, too, like permission bits, but ignore those for this question.)

[2] (e) How many PTEs fit on a page, assuming PTEs are the size computed in part (d)? (Real PTEs need to be somewhat bigger than this, but don't worry about that detail here.)

[2] (f) How many pointers fit on a page?

[2] (g) How big would a flat page table be for a single process, assuming PTEs are the size computed in part (c)?

[3] (h) What are the virtual page offset bits for virtual address 15032? What are the physical page offset bits for virtual address 25012 after it has been translated?

[3] (i) Does a TLB miss always lead to a page fault? Why or why not?

4) [100 points] In Java *[or in C for up to 10 extra credit points]*, write a simulator of a cache. If you choose to write in C instead of Java, you will get 10 extra points added to your score as long as a good faith effort is made.

Submit your work as either `cachesim.java` OR `cachesim.c`, depending on which language you choose for writing the program. If you submit both, then `cachesim.java` will be graded and `cachesim.c` will be ignored.

The simulator, called `cachesim`, takes the following input parameters on the command line:

- name of the file holding the loads and stores
- cache configuration: size (not including tags or valid bits) in KB, associativity, and whether the cache is writeback and write-allocate (wb) or writethrough and write-no-allocate (wt)
- block size in bytes

The replacement policy is always LRU.

For example, `./cachesim tracefile 2 4 wt 32` should simulate a cache that is 2KB, 4-way set-associative, write-through/write-no-allocate, with 32B blocks

This cache hierarchy will be processing the loads and stores in the file called `tracefile`.

Important Assumptions: Addresses are 16-bits (2 bytes), and thus addresses range from 0 to $2^{16}-1$ (i.e., there is 64KB 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 16KB, 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 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 16-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 25bb 2 c77e
load d531 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 what happens (either "hit" or "miss"). For each load, it must also print out the value that is loaded (possibly after satisfying the miss from memory).

The output format must be as follows (for the auto-grader to accept it):¹

```
store 25bb miss
load d5e1 hit 7d2f13ac
```

Test kit instructions:

- To test your solution on your personal machine—e.g., your container (preferred) or your laptop—you're provided with a test kit (`hw5test.py`).
- You may run it as follows: `python3 hw5test.py ALL`
- By default, this will assume you're testing a Java-based solution and will run your `cachesim.class` file in your machine's preinstalled Java Runtime Environment; it assumes that you've already compiled your `cachesim.java` source file into `cachesim.class`.
- To test your C-based solution, you need to tell the tester to look for an **executable** called "cachesim" as opposed to a Java bytecode file called "cachesim.class" (which it looks for by default); to do this, run the tester as follows: `python3 hw5test.py ALL mode=exe`

¹ I'm assuming the load is a hit in this example, but that wouldn't have been the case if these were the only two memory operations in the entire input file.