CS152 Lab 3
Cache Architecture Exploration

Version 0.2. 2022-03-08

1. Overview and Guide

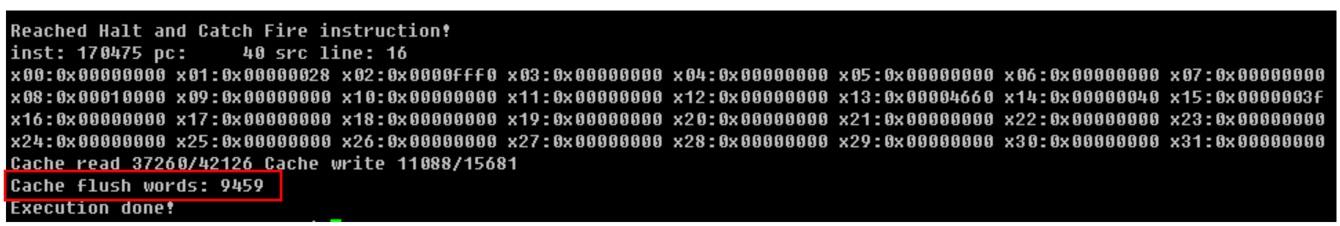
For this lab, you will implement a few different cache eviction policies in a software cache simulator, and measure their effectiveness on some small-scale benchmarks. You will again work with the rv32emulator infrastructure you have worked with for Lab 1. The rv32emulator code has been updated, so please obtain the newest code via "**git pull**"!

The cache relevant code is in the **cachesim.cpp** and **cachesim.h** files. At the top of **cachesim.h**, the number of sets, ways, and size of a cache line (in words) are specified via **#defines**. In **cachesim.cpp**, the relevant functions are **cache_read, cache_write**, and **cache_flush**. The **cache_flush** function chooses a cache way to evict, flushes the data into memory, and then clears the valid bit of the flag. Right now, the **cache_flush** function always chooses way 0 to evict, which is not very effective. Your task is to implement LRU, and measure the performance.

Two small benchmarks are provided

- example_questions/sort.s implements quicksort on 1024 elements
- example_questions/graph.s implements single-source shortest path on a graph with 64 nodes

Performance can be measured via the number of words flushed from the cache, as seen in the following figure:



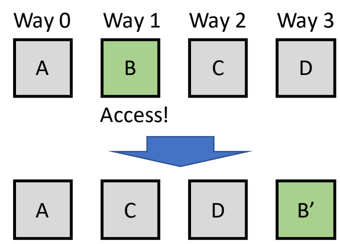
"Cache flush words" displays the number of words flushed from the cache.

2. Implementation steps

2.1. Try a random replacement policy by changing the **cache_flush** function to select **way = rand()%CACHE_WAYS;**

2.2. Implement LRU. This can be achieved in various different ways.

The most straightforward implementation would be augment **cache_read** and **cache_write** to move the accessed <cache line, flag, tag> values to the last way, shifting everything down by one slot. This way, the least recently used cache line will always be in way 0. The following figure illustrates this.



3. What to turn in

You will need to submit a single report, answering the questions listed in the following section.

Please submit in any major document format including .txt, .rtf, .doc, .docx, .odt, .abw, .wpd, or .pdf files.

4. Questions for the report

4.1. Evaluation of random replacement policy

4.1.1. Given a budget of 256 words in the cache (e.g., 256 sets, 1 way, 1 word per cache line, or 64 sets, 2 way, 2 words per cache line), what is the best set/way/line configuration for the two benchmarks? Is there a difference, why?

Remember, the cache configuration parameters are in **cachesim.h**, and are given in logarithmic terms. E.g., **CACHE_SETS_SZ 8** means the cache has 2^8 sets. *Do not modify CACHE_SETS and similar defines directly, as it may break the code.*

4.2. Evaluation of LRU replacement policy

4.2.1. Same as with the random replacement policy, what is the best configuration, and is there a difference between the two benchmarks, and why?

4.2.2. Is the LRU approach always superior to random?

4.3. Bonus: Pseudo-LRU replacement policy

4.3.1. Check out the "Bit-PLRU" approach in [https://en.wikipedia.org/wiki/Pseudo-LRU]. Using one or more bits in the flag, implement the Bit-PLRU policy. How well does the Bit-PLRU do, compared to LRU? Why would be use one or the other?