CS3339
Homework 3: Cache Simulator
Spring 2023
Due: Friday, Apr 7 11:59 PM

Description

For this homework, you will implement a cache simulator. The simulator will take as input (i) the configuration parameters of a cache and (ii) a sequence of memory addresses requested by the CPU. It will then *simulate* the behavior of the configured cache on the given memory references. Upon completion, the simulator will report for each reference whether it is a hit or a miss in cache.

Each cache entry consists of a valid bit and a tag. The tag should be extracted from the memory address in the customary way. For the purposes of this homework, no actual data will be stored in the cache. Hits and misses will be determined by analyzing the memory addresses.

You may assume

- at the outset, the cache is empty (i.e., all entries are invalidated)
- each cache entry/block contains only one word
- the given memory references are specified as *word* addresses

Input

The simulator will be invoked with three command-line arguments as follows

```
./cache_sim num_entries associativity memory_reference_file
```

`num_entries` is the total number of entries and `associativity` is the associativity of the cache to be simulated. You may assume that `num_entries` is a power-of-two and that `associativity` evenly divides `num_entries`. The `memory_reference_file` will contain a list of memory references, separated by spaces. There is no limit to the number of memory references contained in the file.

Output

The output should be placed in a file called `cache_sim_output`. To facilitate testing, each line of the output file should have information about a single memory reference and should be formatted in the following way.

```
ADDR : HIT/MISS
```

`ADDR` is a memory address in the input file printed as an integer. `HIT/MISS` indicates the status of the memory reference. You can create more verbose output for your own benefit, if you like.

Sample I/O

Reference input file

```
1 3 5 1 3 1
```

Invocation

```
./cache_sim 4 2 memory_reference_file
```

Output file

```
1 : MISS
3 : MISS
5 : MISS
1 : MISS
3 : MISS
1 : HIT
```

Implementation Guidelines

The code should be written in C/C++. Although you will not be graded on style, you are expected to follow good design principles (e.g., create a class for cache) and adhere to coding best practices (e.g., the style guide from Google). If you want some skeleton code to get you started, let me or Chase know.

Submission

Your submission should consist of a text file with a link to a Texas Git repository that hosts your code. The repo should contain a README file describing how to build and run your code. The README should also list any known bugs or limitations.

For this homework, you can work in pairs. There should be a single submission per team. Clearly include the names of both team members in the submission.

Extra Credit

- (10 points) Implement multi-word blocks in your simulator
- (10 points) Implement multi-level cache (e.g., L1 and L2)
- (10 points) Classify each miss as compulsory, capacity or conflict