CSCI 2500 — Computer Organization
Homework 6 (document version 1.1)
Cache Simulator

Overview

- This homework is due by 11:59:59 PM on Wednesday, December 12, 2018.
- This homework is to be completed **individually**. Do not share your code with anyone else.
- You **must** use C for this homework assignment, and your code **must** successfully execute on Submitty to obtain full credit.

Homework Specifications

For this individual homework assignment, you will use C to implement a cache simulator. More specifically, your program will accept as input a series of non-zero memory addresses. Depending on the configuration of your simulated cache, some of these memory addresses will be cache hits, while others will be cache misses. You will keep track and display your results as shown on the next page.

For your simulation, you will simulate a direct-mapped, 2-way, or 4-way set associative cache. This is determined via the first command-line argument (i.e., either `1`, `2`, or `4`). Any other argument results in an error to `stderr`.

Further, you will simulate either the least recently used (LRU) or Bélády's algorithm. This is determined via the second command-line argument (i.e., either `LRU` or `Belady`). As above, any other argument results in an error to `stderr`. For LRU, when an item needs to be evicted from the cache, we select the item in our set that was used least recently, i.e., the longest time in the past.

Conversely, for Bélády's algorithm, when an item needs to be evicted, we select the item currently in the cache set that will be used furthest in the future. If an item is never accessed again, it will always be safe to evict it. Note that if multiple items are never accessed again, evict the item with the smallest memory address. This will ensure deterministic output on Submitty.

While Bélády's algorithm sounds impressive, in practice it is not possible to implement given its required foresight of memory accesses (though also in practice, there are techniques used to predict future memory accesses).

Regardless of the configuration from above, your cache must hold at most 256 items. In other words, 256×1 for a direct-mapped cache, 128×2 for a 2-way set associative cache, and 64×4 for a 4-way set associative cache. Initialize your cache lines with zeros, as the given memory trace will never contain address zero.

To keep your cache simulation relatively straightforward, finding your set will be as simple as taking the desired memory address and modding it by the set count, as in:

```
int set = mem_address % set_count;
```

Required Input File

Overall, your program requires three command-line arguments. In addition to the first two arguments described on the previous page, the third command-line argument specifies the name of the input file. This input file contains the sequence of memory address accesses to run through your simulator. Each memory address is a decimal (base 10) number on a line of its own. As noted above, this memory trace will never contain address zero.

Here is an example input file called `ex01.dat` that specifies only 10 memory accesses:

```
1
33
2
34
65
1
66
2
97
65
```

Required Output

For your output, you must echo the specified input parameters, then show each simulated memory access, indicating whether it results in a cache hit or a cache miss. At the end of your simulation, you must show totals and the overall hit rate.

The following example shows a sample run using the input file above. Use this initial example to better understand what you need to implement and track, how you should format your output, and how you could test your program.

```
bash$ ./a.out 2 LRU ex01.dat
Cache size: 256
Cache associativity: 2
Cache sets: 128
Cache algorithm: LRU
1 (miss)
33 (miss)
2 (miss)
34 (miss)
65 (miss)
1 (hit)
66 (miss)
2 (hit)
97 (miss)
65 (hit)
Cache accesses: 10
Cache hits: 3
Cache misses: 7
Overall hit rate: 0.300000
```

(v1.1) Assumptions

Recognizing that this is the last homework assignment, you can make the following assumptions:

- Assume the maximum number of lines (i.e., memory address accesses) in the input file is 1024.
- Assume the values in the input file are all valid 32-bit `unsigned int` values.

Error Checking

Be sure to verify that you have the correct number of arguments by checking `argc`; display an error message if argument(s) are missing.

In general, if an error occurs, use either `perror()` or `fprintf( stderr, "..." )`, depending on whether the global `errno` is set.

And be sure to return either `EXIT_SUCCESS` or `EXIT_FAILURE` upon program termination.

Submission Instructions

Before you submit your code, be sure that you have clearly commented your code (this should not be an after-thought). Further, your code should have a clear and logical organization.

To submit your assignment (and also perform final testing of your code), please use Submitty. Note that the test cases for this assignment will be available on Submitty a minimum of three days before the due date and will include hidden test cases.

Also as a reminder, your code **must** successfully execute on Submitty to obtain credit for this assignment.