Program 4: Cache Simulator

Dr. William Kreahling

November 9, 2020

Overview

You will be reading from standard input and writing to standard output. If you use input redirection from Linux you can run the program like so:
`java myProgram < inputFile`.
Your program will accept one command line argument: A character 'f' or 'F'. If this is present you program will perform a final cache state report. (see the examples below).

Requiring configuration input from either the command line or keyboard will result in lost points.

Your assignment is to write a program that simulates the behavior of an unified cache. The program will read data from *standard input* and produce statistics about a cache 'trace' to standard output. The input will contain cache configuration information followed by memory references consisting of hexadecimal addresses, one per line. The first three lines of the input specifies the number of cache sets, the associativity level of each set, and the size of a cache line, in bytes. Following the first three lines each line consists of 3 colon separated values representing the memory address (in hexadecimal notation), type of memory access (R or W), and the size, in bytes, to read or write, *You should check if each reference's address is properly aligned.* Your cache simulator should handle load byte, load halfword, load word, and load doubleword (Assume word size is 32 bits).

```
#Number of sets: 2
#Set size: 2
#Line size: 8
58:R:4
68:R:4
58:R:4
68:R:4
40:R:4
c:R:4
40:R:4
48:R:4
```

Details:

- The program should be written in C, C++, Java, Python, Go, or your choice of language. It must run successfully on agora. It must compile with no errors, warnings or memory leaks. It must be modular and well designed. If you use and object-oriented language, it must also be object oriented.
- You should check that the line size is at least 4 and that the number of sets and line size will be a power of 2.
- Your cache simulator should use an LRU replacement algorithm.
- **This cache is a write-back, write allocate cache.**
- NOTE: There is **no** actual data for this cache. This is a simulation. You will be given enough information to determine if a memory address would map into a cache of a given size. Then you will keep enough information to track what blocks in the cache *would* have data stored inside them. However, there is no data being provided!

- The cache simulator output should output the following information (in order):
  1. print information about the cache configuration used.
  2. It should indicate if this cache is fully associative or direct mapped.
  3. print information for each reference.

     This information is: access type, the address, tag, index, offset, result (hit or miss) and number of memory references to the level of memory below the cache. **Note: the address and the tag should be printed in hex.** The index, offset, and memory references should be printed in decimal.
  4. After the last reference the cache simulator should output the following summary information:
     (a) the number of hits
     (b) the number of misses
     (c) total accesses to the cache
     (d) total memory references
     (e) hit ratio
     (f) miss ratio
  5. If the final cache state report is requested is should print every set, all lines associated within that set. If the line is valid print: the byte addresses range (in hex) for each line, the current tag, and LRU value.

The output below contains the output of my cache simulator after processing the configuration information and the references in the trace file *trace.dt*. Your output should match match mine in for full credit (this includes style, number format, alignment, etc).

```
Cache Configuration

        2-way set associative entries
        2 sets total
        2 words per set


Results for Each Reference

Access Address    Tag   Index Offset Result Memrefs
------ -------- ------- ----- ------ ------ -------
  read       58       5     1      0   MISS       1
  read       68       6     1      0   MISS       1
  read       58       5     1      0    HIT       0
  read       68       6     1      0    HIT       0
  read       40       4     0      0   MISS       1
  read        c       0     1      4   MISS       1
  read       40       4     0      0    HIT       0
  read       48       4     1      0   MISS       1


Simulation Summary Statistics
-----------------------------
Total hits                 : 3
Total misses               : 5
Total accesses             : 8
Total memory references    : 5
Hit  ratio              : 0.375000
Miss ratio              : 0.625000


   Final Data Cache State
```

```
-----------------------------
set 0
   line 0 = byte addresses 40-47, tag 4, lru 0
   line 1 = invalid
set 1
   line 0 = byte addresses 8-f, tag 0, lru 1
   line 1 = byte addresses 48-4f, tag 4, lru 0
```

- You should use good programming style, which includes descriptive variable names, abstraction, modularization, and comments.
- No global variables or public static variables (with the exception of constants)
- You should comment your program so that others (e.g. the instructor) can understand it.
- You should have comments before each function/method and each major block of code.
- You should comment every variable in your code.
- You should also have comments at the top of the file indicating your name, this course, the assignment, and the command used to compile/interpret your program.

Submit:

Due: November 24th at 11:59 PM. Submit via handin: **All** files needed to compile and run your program.

```
handin.<course #>.<section #> <assignment #> <list of files>

handin.350.1 4 *.[c|C|cpp|java|pl]
```