CS 211: Computer Architecture, Summer 2022
Programming Assignment 3: Cache Simulator (100 points)

Instructor: David Pham

Due: July, 31st 2022 at 11:55pm.

1 Overview

The goal of this assignment is to help you understand caches better. You are required to write a cache simulator using the C programming language. The programs have to run on iLab machines. We are providing real program memory traces as input to your cache simulator. The format and structure of the memory traces are described below.

We will not give you improperly formatted files. You can assume all your input files will be in proper format as described.

No cheating or copying will be tolerated in this class. Your assignments will be automatically checked with plagiarism detection tools that are pretty powerful. Hence, you should not look at your friend's code. See CS department's academic integrity policy at:
`https://www.cs.rutgers.edu/academic-integrity/introduction`

2 Memory Access Traces

The input to the cache simulator is a memory access trace, which we have generated by executing real programs. The trace contains memory addresses accessed during program execution. Your cache simulator will have to use these addresses to determine if the access is a hit or a miss, and the actions to perform in each case. The memory trace file consists of multiple lines. Each line of the trace file corresponds to a memory accesses performed by the program. Each line consists of multiple columns, which are space separated. The first column lists whether the memory access is a read (R) or a write (W) operation. And the last column reports the actual 48-bit memory address that has been accessed by the program. We have provided you four input trace files (some of them are larger in size).

Here is a sample trace file.
R 0x9cb3d40
W 0x9cb3d40
R 0x9cb3d44
W 0x9cb3d44
R 0xbf8ef498

3 Cache Simulator

You will implement a cache simulator to evaluate different congurations of caches. It should be able to run with different traces files. The followings are the requirements for your cache simulator:

- Simulate only one level cache; i.e., an L1 cache.
- The cache size, associativity, and the block size are input parameters. Cache size and block size are specfied in bytes.
- Replacement algorithm: First In First Out (FIFO). When a block needs to be replaced, the cache evicts the block that was accessed first. It does not take into account whether the block is frequently or recently accessed.
- You have to simulate a write through cache.

4 Cache Simulator Interface

You have to name your cache simulator first. Your program should support the following usage interface:

```
./first <cache size><associativity><block size><trace file>
```

where:
A) <cache size>is the total size of the cache in bytes. This number should be a power of 2.
B) <associativity> is one of:
direct - simulate a direct mapped cache.
assoc - simulate a fully associative cache.
assoc:n - simulate an n way associative cache. n will be a power of 2.
C) <block size> is a power of 2 integer that specifies the size of the cache block in bytes.
D) <trace file>is the name of the trace file.

Your program should check if all the inputs are in valid format, if not print error and then terminate the program.

5 Cache Replacement Policy

The goal of the cache replacement policy is to decide which block has to be evicted in case there is no space in the set for an incoming cache block. It is always preferable - to achieve the best performance - to replace the block that will be re-referenced furthest in the future. There are different ways one can implement cache replacement policy. Here we use FIFO replacement policy.

5.1 FIFO

Using this algorithm, you can always evict the block accessed first in the set without any regard to how often or how many times it was accessed before. So let us say that your cache is empty

initially and that each set has two ways. Now suppose that you access blocks A, B, A, C. To make room for C, you would evict A since it was the first block to be brought into the set.

6 Sample Run

Your program should print out the number of memory reads (per cache block), memory writes (per cache block), cache hits, and cache misses for normal cache. You should follow the exact same format shown below (pay attention to case sensitivity of the letters), otherwise, the autograder cannot grade your program properly.

$./first 32 assoc:2 4 trace2.txt
memread:3499
memwrite:2861
cachehit:6501
cachemiss:3499

In this example above, we are simulating 2-way set associate cache of size 32 bytes. Each cache block is 4 bytes. The trace file name is trace2.txt. As you can see, the simulator should simulate the cache and print out all designated metrics. Note: Some of the trace files are quite large. So it might take a few minutes for the autograder to grade for all the testcases.

7 Simulation Details

1. (a) When your program starts, there is nothing in the cache. So, all cache lines are empty (invalid).
(b) you can assume that the memory size is 2pow48 . Therefore, memory addresses are 48 bit (zero extend the addresses in the trace file if theyre less than 48-bit in length).
(c) the number of bits in the tag, cache address, and byte address are determined by the cache size and the block size.
2. For a write-through cache, there is the question of what should happen in case of a write miss. In this assignment, the assumption is that the block is fist read from memory (one read memory), and then followed by a memory write.
3. You do not need to simulate the memory in this assignment. Because, the traces doesnt contain any information on data values transferred between the memory and the caches.
4. You have to compile your program with the following flags:
-Wall -Werror -fsanitize=address

8 Submission

You have to e-submit the assignment using Canvas. Put all files (source code + Makefile) into a directory named first, which itself is a sub-directory under pa3 . Then, create a tar file (follow the instructions in the previous assignments to create the tar file). Your submission should be only a tar file named pa3.tar. You have to e-submit the assignment using Canvas. Your submission should be a tar file named pa3.tar. To create this file, put everything that you are submitting into

a directory named pa3. Then, cd into the directory containing pa3 (that is, pa3s parent directory) and run the following command: $tar cvf pa3.tar pa3 To check that you have correctly created the tar file, you should copy it (pa3.tar) into an empty directory and run the following command: $tar xvf pa3.tar This is how the folder structure should be.

- pa3
  - first
    - first.c
    - first.h
    - Makefile

Source code: all source code files necessary for building your programs. e.g. first.c and first.h. **Makefile:** There should be at least two rules in your Makefile: first: build the executables (first). clean: prepare for rebuilding from scratch.

Grading Guidelines

This is a large class so that necessarily the most significant part of your grade will be based on programmatic checking of your program. That is, we will build the binary using the Makefile and source code that you submitted, and then test the binary for correct functionality against a set of inputs. Thus:

- **You should not see or use your friend's code either partially or fully. We will run state of the art plagiarism detectors. We will report everything caught by the tool to Office of Student Conduct**.
- You should make sure that we can build your program by just running `make`.
- You should test your code as thoroughly as you can. For example, programs should *not* crash with memory errors.
- Your program should produce the output following the example format shown in previous sections. Any variation in the output format can result **in up to 100% penalty**. Be especially careful to not add extra whitespace or newlines. That means you will probably not get any credit if you forgot to comment out some debugging message.

Be careful to follow all instructions. If something doesn't seem right, ask on discussion forum.