Computer Organization HW 3: Developing the L1 Cache for the µRISC-V Processor

Computer Organization 2023 Programming Assignment III (toward "µRISC-V: An Enhanced RISC-V Processor Design using Spike")

Due Date: June 14, 2023 at 23:59

0. Overview

Cache algorithms are optimizing instructions, or algorithms, for managing a cache of information stored on central processing units (CPUs). CPU caches enhance the CPU performance by accommodating recent or often-used data items (that are originally stored in memory locations) and by allowing the data to be accessed faster than ordinary memory reads/writes. When a cache is full, the cache algorithm must choose which items to discard to make room for the new ones.

In this assignment, you have to develop **cache replacement policies**, and these policies should be implemented in the open-source RISC-V emulator, Spike. By implementing the policies with the cache simulation interfaces exposed by Spike,the RISC-V processor emulator can further estimate the performance delivered by various cache models (each with a distinct replacement policy or a cache configuration). Note that the original version of Spike has implemented the mechanism of cache simulations. That is, users can enable the cache simulations (e.g., L1 data and/or L1 instruction caches) by specifying which types of the cache simulations to be performed via parameter settings. Unfortunately, the current version of Spike only implements the *random* replacement policy and some other policies should be implemented in order to better evaluate different design alternatives.

Specifically, you are required to implement the three replacement policies to get the basic points, as described in the next section (1. Cache Replacement Policy). Moreover, you might like to design a novel policy that outperforms the three implemented policies to get bonus points of this assignment. Finally, you have to elaborate your design philosophy and elucidate your implementation code to get some points in the demonstration to the course TA.

1. The Three Cache Replacement Policies (The to-be-implemented policies)

As a cache replacement policy can greatly affect the cache hit/miss rate, it has been extensively studied and hence, many replacement polices have been proposed, each suitable for some purposes. The cache hit/miss rate is an important metric to evaluate the efficiency of a cache design. In this assignment, your designs are evaluated solely by the **miss rate**. You are required to implement the following three policies and to report their miss rates to get some scores of this assignment.

The three of the most commonly seen cache replacement policies are listed below. These three policies are served as the *baseline* policies to be implemented in this assignment.

- **First In First Out (FIFO)**: The cache behaves the same as a FIFO queue, where it evicts the cache blocks in the same order as they were added to the cache without considering any other factors.
- **Least Recently Used (LRU)**: Those the least recently used cache blocks are discarded first.

- **Least-Frequently Used (LFU)**: Those cache blocks that are used least often are replaced first.

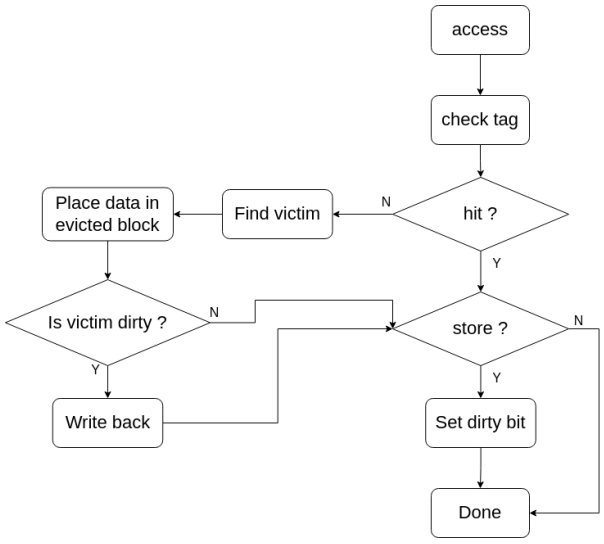
For more cache replacement policies (or the above three policies), please refer to the web page for more details.

2. Cache Simulations Supported by Spike

During the program emulation by Spike, the emulated instructions (the addresses of the instructions) are fed into the L1 *instruction cache* simulator and the data required by the emulated instructions (e.g., accessed via the memory instructions) are fed to the L1 *data cache* simulator. The cache simulator(s) are represented as the `cache_sim_t` class objects in Spike. The creation of the cache simulators is determined by command line parameters of Spike; for example, the `dc` flag is used to create a L1 data cache simulator, and its parameters are used to specify the attributes of the created data cache, such as *set* and *ways*. Please refer to the following section (3. What Should You Do in this Assignment?) for more details.

The `cache_sim_t` class defines the behaviors of the cache simulators created by Spike, where the prototypes are defined in `riscv/cachesim.h` and the implementation is in `riscv/cachesim.cc`. Both data and instruction caches inherit the interfaces defined in the `cache_sim_t` class. There are some important member functions related to manipulate the created cache(s), such as `access`, `check_tag`, and `victimize`. In the following paragraphs, we briefly introduce the high-level concepts of `access`, and `check_tag`. **You should trace the source code to figure out more details of the member functions so as to understand the mechanism and workflow of the caches simulated by the `cache_sim_t` objects.**

The workflow of the data cache simulation in Spike is highlighted as follows. The data cache simulation starts when a memory read/write is emulated by Spike, and the `access` function is invoked. The prototype of the `access` member function is `cache_sim_t::access(uint64_t addr, size_t bytes, bool store)`, where the parameter `addr` contains the address to the to-be-accessed data, `bytes` represent the size of the to-be-accessed data, and `store` refers to its a read or store operation. The high level workflow is illustrated in the following image.



As for the `cache_sim_t::check_tag(uint64_t addr)` function, it is used to check the existence of a to-be-accessed data in the simulated data cache (i.e., `check tag` rectangle in the workflow image). The index to the *set* for the to-be-accessed data is calculated by the `addr` argument via the formula: `idx = (addr >> idx_shift) & (sets-1)`. If you have configured a four-set data cache, the `idx` value is the remainder of the dividing by 4 operation; that is, the value range of `idx` is from 0 to 3. Next, all of the *tag* in

the corresponding *ways* are compared with the input argument `addr` to find the existence of the data with the `addr` in the data cache. If a match is found, it means a cache it; otherwise, it is a cache miss.

When a cache miss occurs, the `victimize` function is invoked to find a victim data cache block to store the new input data. Currently, Spike implements only the *random* algorithm to randomly select the victim data block. When cache miss occurrs, the victim block should be selected to store the input data. In particular, Spike leverage the LFSR to implement a pseudo-random cache replacement policy which use `way = lfsr.next() % ways` to choose victim block. Once the victim block has been selected, input data can be stored in it.

Note that computer simulations do not always replicate real hardware specifications and behaviors. Instead, they focus on some designated problems and use different workloads and simulation configurations to evaluate the tradeoffs among different designs. For example, in this assignment, we focus on the L1 data cache designs and try to find out a good cache design that can lead to the lowest data cache miss rate of the given workload. Furthermore, the overhead incurred by the cache misses data is not taken into account, and the implementation overhead (e.g., hardware area, power consumption, computation latency) of different replacement policies is not calculated, too.

3. What Should You Do in this Assignment?

You are responsible for **implementing at least three cache replacement policies** (FIFO, LRU, LFU) into the cache simulation module of Spike. To do so, you should use the Spike source code from the repository `riscv-isa-sim` that has been installed in *HW #0*. Particularly, you need to revise the following two source files that are used for cache simulations.

- Please add your implementations of the cache replacement policies in the following files. Note that the implementation of each replacement policy has its own file name. Please refer to Section 5. Submission of Your Assignment for more details.
    - **riscv/cachesim.h**: You may like to add the required data structures for the cache simulation in this file, so as to keep essential information to facilitate cache simulation.
    - **riscv/cachesim.cc**: You are required to add your code into this file to implement the mechanism of the replacement policies. For example, you need to inject your code to the member functions of the `cache_sim_t` class to do what is necessary for the cache simulation in different stages (of the above workflow image). You might like to refer to cache replacement policies first to understand the mechanisms of the to-be-implemented replacement policies.

This assignment has two major parts, each of which has different requirements. Please check the instructions below carefully.

- The first part is to implement the three *baseline* cache replacement policies, as specified in 1. The Three Cache Replacement Policies. The scoring conditions are listed in the following subsection.
- The second part includes the code explanation (to TA during your demo session) and the implementation of another cache replacement policy (in addition to the three policies implemented in the first part).

Please submit your developed code before the assignment deadline specified at the very beginning of this document. You can get some credits of this assignment only if you submit the code to the NCKU Moodle and *correctly* answer to the TA questions during your demo session.

Before your start, please prepare your development environment

To make sure your installed environment behaves correctly, you need to run the following commends in the Linux shell.

1. The first command is to build an RISC-V executable.
2. The second command is to feed the built executable to the Spike simulator and activate the L1 data cache simulation. The cache miss rate (of the default *random* cache replacement policy) will be reported. XXX: Is there any output example for the L1 data cache simulation?

Use the following commands to compile an input program and to do the instruction emulation and the L1 data cache simulation for the input program.

```
$ riscv64-unknown-elf-gcc -static -o qr ./benchmakr/qrcode.c
$ spike --isa=RV64GC --dc=2:4:8 /home/ubuntu/riscv/riscv64-unknown-
elf/bin/pk qr
```

The argument `--dc=<sets>:<ways>:<linesz>` allows you to turn on the L1 data cache simulation and to set the number of **sets**, **ways**, **block size** for the activated cache simulation, where these three argument should be the power of two. If you use the setting: `--dc=2:4:8`, it means that the data cache will be configured with 2 sets, 4 ways and 256 bytes each block. You can apply different settings to observe the different miss rates of these settings and think about the reasons of higher/lower miss rates. **Note that** you can use up to 6 bits at most in your setting. For example, there is 6 bits in the `--dc=2:4:8` setting, and the calculation of the setting and the required bits are listed as below.

- sets = $2 = 2^x$ where $x = 1$
- ways = $4 = 2^y$ where $y = 2$
- block size = $8 = 2^z$ where $z = 3$ (block size need to be greater than and equal to 8)
- $x + y + z = 6 \leq 6$

A. Baseline: Implementation of the three cache replacement policies (40 %)

This baseline exercise is scored by the delivered L1 data cache miss rate after the program emulation done by Spike. In particular, you need to report *which cache replacement policy* and *the setting* of the data cache (e.g., `--dc=2:4:8`) that can deliver the **lowest** L1 data cache miss rate. The *answer* should be set in the `config.conf`. Please check the information below to see the format of your provided answers. **Note that** in this assignment, the L1 data cache miss rate is the major focus, and you do not have to take some other factors into consideration (e.g., the cache hit time).

You need to implement the three replacement policy, and you run the Spike simulations with different configurations of **replacement policies** and **data cache settings**. An intuitive way is the brute force method to exhaustively search all of the possible configurations to find your answer: **the exact configuration delivering the lowest L1 data cache miss rate**.

Your obtained score of this exercise depends on the miss rate below. TAs will build a new Spike executable with your code and run with your configuration to obtain the *lowest miss rate* of the test programs.

- miss rate < 50 % (10 %)

- 40 % < miss rate <= 50 % (15 %)
- 35.5 % < miss rate <= 40 % (20 %)
- miss rate <= 35.5 % (40 %)

The example format of `config.conf`

The example of the cache setting in the `config.conf` file is as below. This example content means that the policy `LRU_cachesim` and the setting `--dc=1:2:32` give the lowest miss rate.

```
[cache]
Set = 1
Way = 2
BlockSize = 32
Policy = "lru"
```

> Policy can be configured to `origin`, `fifo`, `lru`, `lfu`, `self`

Build the revised Spike to activate your implemented cache replacement policies

Your should refactor the corresponding file (`FIFO_cachesim.h`, `FIFO_cachesim.cc`, `LRU_cachesim.h`, `LRU_cachesim.cc`, `LFU_cachesim.h`, and `LFU_cachesim.cc`) to implement different policies. You can recover to original code with `ORIG_cachesim.cc` and `ORIG_cachesim.h`. Lastly, if you implement other policy you can put it into `SELF_cachesim.cc` and `SELF_cachesim.h`. **You have to rebuild the Spike with the following commands and config.conf.**

After completing the above tasks, you can begin to run the benchmark to evaluate the miss rate with :

```
$ make score
```

Note that you have to revise riscv-isa-sim path in makefile if you build the environment by yourself.

And everyone have to revise setting in `config.conf` to find the best policy and configuration.

> benchmark refer to rv32emu/tests with
>
> - qrcode.c
> - nyancat.c
> - captcha.c

B. Demo: Elaboration of your implemented code and extra efforts (80 %)

Your job is to register a demo session (which will be announced later) and elaborate your design philosophy of your implemented replace policies. Your obtained score is based on the answer you provided to TA. Besides, you can earn a bonus score if you implemented another cache replacement policy (the fourth cache replacement policy). Moreover, an extra score can be earned if the fourth policy has a lower miss rate (less than 35.5%).

A more detailed explanation of the to-dos in the demo session and their credits.

- Describe the workflow and mechanism in Spike, related to cache simulation. (20 %)
    - You are required to open the source files and to elaborate the functionalities of the Spike source code (not limiting to the cache simulation code).
- Elucidate the design philosophy, mechanism, and the developed code for implementing the three policies. (50 %)
- Bonus. (10 %)
    - The elaboration of your implemented another policy (e.g., the fourth policy). (5 %)
        - The fourth policy can be a variant (e.g., mixing) of the above three policies or a totally different policy from the three policies.
    - Run the simulation of the fourth policy and it delivers the miss rate that is lower than 35.5%. (5 %)

4. Test Your Developed Code

Run the benchmark programs for your assignment

```
$ make score
Policy: "origin"
Data Cache Setting with: 1:8:8
Miss Rate: 50.30 %
```

5. Submission of Your Assignment

At least 7 file you should revise

- FIFO_cachesim.cc
- FIFO_cachesim.h
- LRU_cachesim.cc
- LRU_cachesim.h
- LFU_cachesim.cc
- LFU_cachesim.h
- config.conf (the best configuration you find)

(Optional)

- makefile (if your riscv-isa-sim and pk path is different to default)
- SELF_cachesim.cc
- SELF_cachesim.h

We assume your developed source files are put under the folder: `CO_StudentID_HW3`. Please follow the instructions below to submit your programming assignment.

1. Compress your source code into a zip file.
2. Submit your homework with NCKU Moodle.
3. The zipped file and its internal directory organization of your developed code should be identical to the example below.

- **NOTE**: Replace all following "*StudentID*" with your student ID number.

```
CO_StudentID_Hw3.zip/
└── CO_StudentID_Hw3/
    └── benchmark/
        ├── captcha.c
        ├── nyancat.c
        └── qrcode.c
    ├── config.conf
    ├── FIFO_cachesim.h
    ├── FIFO_cachesim.cc
    ├── LRU_cachesim.h
    ├── LRU_cachesim.cc
    ├── LFU_cachesim.h
    ├── LFU_cachesim.cc
    ├── ORIG_cachesim.cc
    ├── ORIG_cachesim.h
    ├── SELF_cachesim.cc
    ├── SELF_cachesim.h
    ├── makefile
    └── test.py
```

!!! Incorrect format (either the file structure or file name) will lose 10 points. !!!

6. Reference

- Cache replacement policy
- Linear-Feedback Shift Register (LFSR)