CS311 Project 4: Cache Design

Due 11:59pm, December7 th
TA: Seungbeom Choi

1. Purpose

This project is intended to help you understand the principle of caching by implementing a data cache.

2. Overview

The main part of this project is to simulate a data cache from an input trace file. The cache should be configurable to adjust capacity, associativity, and block size with command-line options. You must support extra options to print out the content of the cache.

Please remember this is an alternative you may take but **you can receive only 80% of maximum score compared to the pipeline integrated cache version for project4.** In other words, your maximum score of completing **this project is 80 / 100 points**, whereas the maximum score **for pipeline-integrated data cache is 100 / 100 point**s.

2.1 Standalone Data Cache Simulation

In this project, you must design a trace-based cache simulator. An input trace file contains a sequence of read or write operations with addresses. For each step, an entry of the trace is fed to the cache simulator, to simulate the internal operation of the cache. The write policy of the cache must be *write-allocate* and *write-back*. The replacement policy must be the perfect LRU.

2.2 Cache Parameters

The capacity, associativity, and block size of the cache must be configurable. The parameters are specified with "–c" option: -c <capacity>:<assoc>:<blocksize>
Configurable parameters:

- Capacity: 4B (one word) - 8KB
- Associativity: 1 – 16 way
- Block size: 4B – 32B

When you specify both capacity and block size, you should specify the number with byte granularity and power of two.
Ex) Capacity 4KB, Associativity 4way, Block size 32B → **4096:4:32**

3. Simulator Options and Output

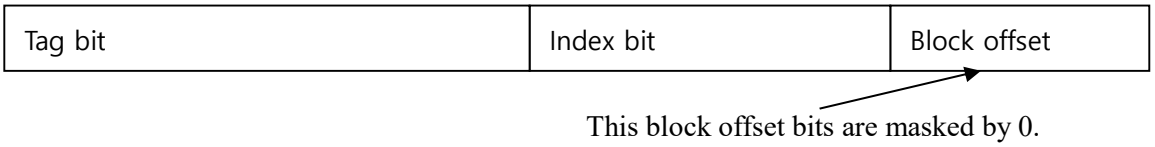
3.1 Options

```
cs311cache -c cap:assoc:block_size [-x] input_trace
```

- -c : cache configuration
- -x : dump the cache content at the end of simulation

Cache content is not data content but the address which is aligned with block size.
Ex) Block size 16B

When the address 0x10001234 is stored in cache, the content of the cache entry is 0x10001230.
The 4bits (block size bit) are masked by 0.



The TAs will provide skeleton code for displaying output format within “main.c”. You are free to change the parameters and corresponding parts of the code as long as it can print out the same format.

The example of output format with options ‘–c’ and ‘–x’ is attached as below.

```
Cache Configuration:
-------------------------------------
Capacity: 256B
Associativity: 4way
Block Size: 8B

Cache Stat:
-------------------------------------
Total reads: 0
Total writes: 0
Write-backs: 0
Read hits: 0
Write hits: 0
Read misses: 0
Write misses: 0

Cache Content:
-------------------------------------
          WAY[0]      WAY[1]      WAY[2]      WAY[3]
SET[0]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[1]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[2]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[3]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[4]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[5]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[6]:   0x00000000  0x00000000  0x00000000  0x00000000
SET[7]:   0x00000000  0x00000000  0x00000000  0x00000000
```

3.2 Input

The input trace contains a sequence of read or write operations with addresses as we mentioned.

Ex)
R 0x10000000
W 0x10003231
R 0x12341245
R 0x10003231
W 0x10023414
…

The TAs will provide several input traces which are both real world traces and the traces which we generate in order to check the cache operations.

3.3 Output

Your output must contain the following statistics:

- the number of total reads
- the number of total writes
- the number of write-backs
- the number of read hits
- the number of write hits
- the number of read misses
- the number of write misses

The order of output results is Cache Configuration (if –c option is on) → Cache Statistics → Cache Content (if –x option is on).The example of output format are attached as below.

4. Forking and Cloning your Repository

As with Project 1, we will fork the TA's Project4 repo to your team namespace. Then you will clone your team's repo into your local machines to work on the project.

4.1 Forking the TA's Repo

(1) Go to the following page: https://cs311.kaist.ac.kr/TAs/Project4-standalone . This page is the TA's repository.

(2) Click the fork button just like previous projects.

(3) Now select your Team (NOT YOUR USER) and the repo will be forked.

(4) Your Team repo will have the following URL: https://cs311.kaist.ac.kr/Team[your number]/Project4-standalone
NOTE: We will be running automated scripts to download your work and grade your projects. **Please do not change the name of your project paths**. (Keep the project name & path as Project4-standalone)

4.2 Cloning the team repository to your local machine

From the website of your team repo copy the SSH or HTTPS URL of the git repository
the SSH URL will look something like the following:
ssh://git@cs311.kaist.ac.kr:10022/Team[your number]/Project4-standalone.git

Change directory to the location you want to clone your project and clone!
$ git clone ssh://git@cs311.kaist.ac.kr:10022/Team[your number]/Project4-standalone.git

Be sure to read the README.md file for some useful information. It includes the explanation of each file and which files you are allowed to modify for this project.

5. Grading Policy

Grades will be given based on the examples provided for this project provided in the `sample_input` directory. Your simulator should print the exactly same output as the files in the `sample_output` directory.

We will be automating the grading procedure by seeing if there are any difference between the files in the `sample_output` directory and the result of your simulator executions.
Please make sure that your outputs are identical to the files in the sample_output directory.

You are encouraged to use the *`diff`* command to compare your outputs to the provided outputs.

```
$ ./cs311cache -c 1024:8:8 sample_input/simple > my_output
$ diff -Naur my_output sample_output/simple
```

If there are any differences (including whitespaces) the diff program will print the different lines. If there are no differences, nothing will be printed. Furthermore, we have provided a simple checking mechanism in the Makefile. Executing the following command will automate the checking procedure.

```
$ make test
```

There are 4 code segments to be graded and you will be granted 20 points for each correct binary code and **being "Correct" means that every digit and location is the same** to the given output of the example. If a digit is not the same, you will receive **0 score** for the example. As mentioned in 'Overview' section, the maximum points you can receive for this project is 80 points.

6. Submission (Important!!)

6.1 Make sure your code works well on your allocated Linux server

In fact, it is highly recommended to work on your allocated server throughout this class.
Your project will be graded on the same environment as your allocated Linux server.

6.2 Summarize the contribution of each team member

If you are working with your teammate, you need to summarize your contributions to each project. Add a 'contribution.txt' file in your repository (don't forget to commit it). If you use good commit messages, this can be done in a simple step. **"git shortlog"** summarizes commit titles by each user and will come in handy (especially if your commit titles have useful information).

```
$ git shortlog > contribution.txt
$ git add contribution.txt
```

```
$ git commit
```

If you want to add commit messages, please fill in the part after the option '–m' when committing.

```
$ git commit
```

A text editor will pop up with some information about the commit. Fill out your commit message at the top. The first line is the subject line of the commit. The second line should be blank, the third line and onwards will be the body of your commit message.

6.3 Add the 'submit' tag to your final commit and push your work to the gitlab server

The following commands are the flow you should take to submit your work.

```
$ git tag submit
$ git push
$ git push --tags
```

If there is no "submit" tag, your work will not be graded so please remember to submit your work with the tag.

If you do not `push` your work, we will not have the visibility to your work. Please make sure you push your work before the deadline

6.4 Updating Your Submit Tag

If you decide after tagging your commit and pushing, that you want to update your submission, you will need to remove the existing tag and retag & repush your submit tag.

```
$ git tag -d submit          # Deletes the existing tag
$ git push origin :submit # Removes the 'submit' tag on the server
```

Now you may re-tag your work and submit using the instruction in Section 7.3

7. Late Policy

You will lose **50%** of your score on the **first day** (Dec 8th 0:00 ~23:59). We will **not accept** works that are submitted after then.

Be aware of plagiarism! Although it is encouraged to discuss with others and refer to extra materials, copying other students or opened code is strictly banned.

The TAs will compare your source code with open source codes and other team's code. If you are caught, you will receive a penalty for plagiarism.

If you have any requests or questions regarding administrative issues (such as late submission due to an unfortunate accident, GitLab is not working) please send an e-mail to the TAs(cs311_ta@calab.kaist.ac.kr).

8. Updates/Announcements

If there are any updates to the project, including additional tools/inputs/outputs, or changes, we will post a notice on the Notice board of KLMS, and will send you an e-mail using the KLMS system.

Frequently check your KLMS linked e-mail account or the KLMS notice board for updates.

8.1 Merging any updates into your repository

During the course of the project, the TA may need to update some sample_input/sample_output files. If so the TA will notify you via KLMS notice board and mailing system from KLMS. If the TA instructs you to pull any new changes you can do the following.

Move to your local git directory. Commit any uncommitted changes.

```
$ git remote add upstream ssh://git@cs311.kaist.ac.kr:10022/TAs/Project4-standalone.git
$ git pull upstream master          # This may ask for you to write a merge commit
                                    # Just save and exit
```

Now you should have caught up with the TA's change. Check with `git log` and check if you see the TA's commit at the very top. If the TA sends you another notification asking for merging any new changes, you can skip the first command and only execute the second `git pull upstream master` command.