SCE212 Project 4: Cache Design

Due 11:59pm, January 26th, 2018

1. Overview

This project is intended to help you understand the principle of caching by implementing a data cache. The main part of this project is to add a data cache to the five stage pipeline implemented for project 3. Cache is considered to be sitting between the processor and memory. Therefore, whenever your simulator executes instructions such as `lw` or `sw`, which access memory, your simulator must access the data cache for data. Your simulator must also support extra options to print out various content related to the newly added cache.

2. Cache Implementation

In this project, you must add a data cache to the pipelined MIPS simulator you have implemented for project 3. If you have not yet finished your pipelined simulator, we strongly recommend that you finish it before proceeding with this project. Assume the memory reads from `IF`, access the memory directly in a cycle in the same way as the project 3 model assumes. You only need to add a data cache. The write policy of the cache must be `write-allocate` and `write-back`. The replacement policy must be the `perfect LRU`.

If an access is a cache hit, it takes a cycle to finish the MEM stage. If an access is a cache miss, it takes `30 cycles` to finish the MEM stage. *While handling the miss, the pipeline must be stalled.* During the 30 cycle period, the missed cache block is moved from the memory to the cache, and the access is completed. **After 30 cycles of handling the miss, the requested block must be present in the cache and before then the contents of data cache remain the same**. **Also, your data cache must be storing actual data used in the provided inputs**.
If you need a concrete example, please run the reference simulator with "-X" and "-p" option.

```
Hit : IF – DE – EX – MEM - WB
Miss: IF – DE – EX - MEM1- … - MEM30 - WB
```

2.1 Cache Parameters

For this project, you must implement a **4-way set associative cache**. We will be using a fixed set of parameters as following:

```
Associativity:     4-way
Capacity:          64 Byte
Block Size:        8 Byte
```

3. Simulator Options and Output

3.1 Options

The following options must be supported in your simulator:

```
$ ./sce212sim [-nobp] [-nof] [-m addr1:addr2] [-d] [-p] [-x] [-X] [-n
 num_instr] [-mc cycle_miss] [-c] inputBinary
```

- `-m`: Dump the memory content between addr1 to addr2
- `-d`: Print the register file content every cycle.
- `-p`: print the PCs of instructions in each pipeline stage at *every cycle*
  ex) CYCLE N: x3004 | x3003 | x3002 | x3001 | x3000
- `-c`: shows cache configuration at the start of simulation
- `-x`: dump the cache content at the end of simulation
- `-X`: dump the cache content every cycle
- `-n`: simulate for only a 'num_instr' instructions.
- `-mc`: Set the number of cycles to stall when handling cache miss. Default is set to 30
  (Although -mc option is not used for grading, it will help you debug your code)

The TAs will provide the skeleton code for displaying output format. The code for new options can be found in `util.c` and you are free to change the parameters and corresponding parts of the code as you wish 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: 64B
Associativity: 4way
Block Size: 8B

Simulating for 100 instructions...

Current Cache state:
-------------------------------------
                WAY[0]              WAY[1]              WAY[2]              WAY[3]
SET[0]:   WORD[0]: 0x00000000  WORD[0]: 0x00000000  WORD[0]: 0x00000000  WORD[0]: 0x00000000
          WORD[1]: 0x00000000  WORD[1]: 0x00000000  WORD[1]: 0x00000000  WORD[1]: 0x00000000

SET[1]:   WORD[0]: 0x00000000  WORD[0]: 0x00000000  WORD[0]: 0x00000000  WORD[0]: 0x00000000
          WORD[1]: 0x00000000  WORD[1]: 0x00000000  WORD[1]: 0x00000000  WORD[1]: 0x00000000
```

3.2 Reference simulator

We will be providing a reference simulator (without the code, of course) so that you may compare the execution of your simulator to the reference. You will be able to dump debug messages, pipeline outputs, and cache contents of both the reference and your simulators to check the execution every cycle.

The TA's answer may contain some incorrect executions. If you do find any upon your debugging and comparing, please drop us an e-mail of the expected behavior, and the actual behavior shown by the TA's answer. We'll check it out and push fixed reference outputs.

4. Forking and Cloning your Repository

As with Project 3, we will fork the TA's Project4 repository 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: `http://beehive.ajou.ac.kr:10080/SCE212/Project4`.
(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://sce212.ajou.ac.kr/Team[your number]/Project4`

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)

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@sce212.ajou.ac.kr:10022/Team[your number]/Project4.git
```

Change directory to the location you want to clone your project and clone!

```
$ git clone ssh://git@sce212.ajou.ac.kr:10022/Team[your number]/Project4.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 `grading_input` directory. Your simulator should print the exactly same output as the files in the `grading_output` directory.

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

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

```
$ ./sce212sim -p grading_input/various_inst.o > my_output
$ diff -Naur my_output grading_output/various_inst
```

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 5 code segments to be graded and you will be granted 20% of total score 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.

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 (do not 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
```

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(`XXX@ajou.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 Ajou BB, and will send you an e-mail using the Ajou BB system. **Frequently check your Ajou BB linked e-mail account or the Ajou BB 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 Ajou BB notice board and mailing system from Ajou BB. 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@sce212.ajou.ac.kr:10022/TAs/Project4.git

# This may ask for you to write a merge commit
# Just save and exit
$ git pull upstream master
```

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.