ECE 201/401: Project 2 MIPS Processor with Caches

Due Date: November 14, 2019 11:59p—Don't Be late!

Thursday 31st October, 2019

1 Introduction

In this project, you are given a working MIPS 5 stage pipeline and are expected to add a single level off caching to it. Your job s to scrutinize the code in order to extract the existing interfacing ports and adding the required cache components.

You may form a group of **at most two (2)** students to finish this project[1], but you don't have to if you feel more comfortable working by yourself. If you work in groups, please make sure both students do the same amount of work. You are encouraged to help each other. However, do not share your code (unless you are in the same group, of course).

2 Requirements

In this project, to fix the pipeline to an appropriate degree, you must:

- Compile and run the provided code of all the test programs and immediately report if you have a problem.
- Add separate L1 caches for instruction and data memories as described below. Timing details are discussed in Section 4.
  - Instruction Cache: 32KB, read-only, Direct Mapped, 32B block size, with 1 cycle access latency and 10 cycle miss penalty
  - Data Cache: 32KB, write-back, write-allocate, 2-way, 32B block size, 1 cycle access latency with 10 cycle miss penalty
- You can assume memory accesses resulting from cache misses by the instruction and data caches will not conflict with each other and can be serviced in parallel.
- Report all the issues regarding accessing to the memories that you solve in your project (and how).
- Assuming a memory access delay in the original processor (without caches) the same as miss penalty in the table above, compare the IPC for the entire test programs on both designs.

[1] This does not have to be the same partner as project 1

- Create a project report (described below)

Partial credit for completing significant fractions of these tasks (except for the first one) will be granted.

3 Testing

Your design will be tested using files provided to you, of which there are two types.

1. asm: These are similar to the tier 1 tests you had for project 1. They are aimed at targeting specific issues you may encounter and thus will allow you to debug your program one instruction at a time to iron out any bugs in your pipeline.

2. cpp: These are the same as some of the ones from the previous also with some additions. In order for full credit, these must run successfully. These are much harder to debug because they are more complex programs.

4 Memory Timings

The timing details for the cache assume that data present in the cache can be retrieved in one cycle. The ten cycle latency for cache misses can be construed as follows:

1. On the first cycle, the cache checks to see if the data is available. If it isn't, make a request to the lower-level cache (L2 or main memory)

2. Return word 1 to cache. On the second cycle, the lower-level cache finds the data and starts to return it. On a normal system, this would take longer than 1 cycle. Since the data bus is only one word (32 bits / 4 bytes) wide, only one word is available on each cycle. To retrieve the entire line will take another eight cycles.

3. Return word 2 to cache

4. Return word 3 to cache

5. Return word 4 to cache

6. Return word 5 to cache

7. Return word 6 to cache

8. Return word 7 to cache

9. Return word 8 to cache

10. The cache line is now populated, and the cache services to original request.

sim_main will simluate the time needed to send 8 words by refusing to process block read/write requests sooner than every 8 cycles. It is permissible to modify sim main to match timing requirements of your cache, but you must still maintain the 10-cycle-per-miss latency (You also must document all changes made to sim main and the reasons for them).

Note that writes to memory are not subject to the 10-cycle penalty. You may assume that this is because of an in infinite write buffer between the cache and *sim_main*, or you can assume that waiting 20 cycles to read data because of the need to evict a dirty line is just too long.

5 Getting Started

Download and unpack the project files using the command line below:
tar -xvzf ECE401-Project2-F19.tar.gz
The extracted directory will have the following file structure.

- verilog/
  This contains the verilog design files for the processor. All necessary changes for this project will be made here. Additionally, any additional verilog files you write should be placed here.

- sim_main/
  This contains the c++ source for the simulator that will run your processor. While you do not make any changes to this file, some cursory knowledge of how it works will be useful for the extra credit opportunities.

- tests/
  We provide several tests for you. Tests can be downloaded by, in terminal, switching directory to the same folder as the makefile and typing 'make tests'.

6 Compile

Compilation is made easy by the makefile, allowing you to compile by being in the directory of the makefile and typing 'make'. We encourage you to develop on the ECE cluster machines. You may develop on your personal machines, however if your code does not run on the ECE machines you will receive a score of zero (0).

7 Simulate

Once successfully built, you need to test your system. This can be done by running the executable that's now in your topmost directory 'VMIPS'. You will need to provide VMIPS a file to run as well. It should look something like this.

```
./VMIPS -f tests/cpp/class
```

Additionally, you can provide a number of cycles to run. For example, to run for 12345 cycles.

```
./VMIPS -f tests/cpp/class -d 12345
```

If you find that you are encountering a problem at a specific address, you may also provide a breakpoint to VMIPS:

```
./VMIPS -f tests/cpp/class -b 0x0413ABCD
```

After your program concludes execution, you may be left with several additional files.

- stdout.txt: This contains anything written to the stdout output stream during execution.
- stderr.txt: This contains anything written to the stderr output stream during execution.
- memwrite.txt: This will contain a list of reads and writes to main memory. Because sim main evaluates memory accesses twice per clock, each request will usually be shown twice.
- cachewrite.txt: Whether or not you even have a cache, this will contain a list of reads and writes to the data cache. It makes use of the various 2DC and fDC wires in MIPS.

8 Advice

You may find the following advice useful for doing this project.

1. Start the project early; right now is a good time. Otherwise you won't be able to finish it before the deadline.
2. Draw a timing diagram for how the caches will operate.
3. Iterate on modification and verification of the design using the provided test programs.
4. Use $display to print out messages on the screen as needed.
5. Take advantage of the asm test programs to diagnose why the pipeline may be malfunctioning.
6. You may reduce the total of code you need to write by creating parameterized modules (check http://www.asic-world.com/verilog/para modules1.html)
7. System calls will probably work best if caches are flushed first...
8. Both partners should contribute

9 Submission

You are to electronically turn in the following via Blackboard:

1. Include a README file, in text format (see next paragraph).
2. You may work in groups of up to two (2).
3. Only one partner should submit the code if working in a group[2].
4. You may submit multiple times. **Only the last submission before the deadline will be graded.**
5. Verify that your code runs on the ECE servers; If your code doesn't run you will receive a zero (0)

Write both partners' names. The README report is expected to contain an explanation of what you did, and explain how you did it. You should specify what is and is not working. If something does not work, attempt to explain why. If you have made changes to sim main, you must justify them in your report. **Be sure to discuss if caching helps or hurts performance compared to the given pipeline. Why?**

You are to compress and archive your project 2 folder using the following command and naming convention and upload the generated file, along with your report, onto blackboard.

tar -cvzf FirstInitialLastName_Project2.tar.gz {*Name of Directory to ZIP*}

10 Extra Credit

For this project you may earn up to 50 extra points by implementing a unified L2 cache. This means that the L2 services both the instruction and data L1 caches.

The cache parameters are as follows:

- 128 KB, write-back, write allocate, 8-way set associative, 32B block size

The databus between L1-L2, and L2-Main Memory are all 32-bits wide and can transfer 32-bits/cycle.

Timing for the L2 are as follows:

- The L1 hit time is the same as specified in section 4
- The L2 hit time is 25 cycles.
- The Main memory hit time is 50 cycles.

[2]Both partners will receive the same grade