ECE2/401 Project 1
Fall 2018

October 9, 2018

1 Description

In this project, you are given a working MIPS 5 stage pipeline and are expected to add caching to it. Your job is scrutinizing the code in order to extract the existing interfaing ports and adding the required cache components.

2 Requirements

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

- Compile and run the provided code fo all the test programs and immediately report if you have a problem.
- Add seperate L1 caches for instruction and data memories as described below. Timing details are discussed in another section.
  - 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-allocte, 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.

- 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: Simple tests designed to help you debug your project. They are easily stepped through instruction by instruction so that you can make sense of the behavior of your desing.
2. cpp: Complex tests that consist of several compiled C++ programs. These are much harder to step through and can last several hundred thousand cycles. If all asm tests work, then cpp tests will be very close to working.

The multi-cycle and 5 stage pipelines will different in instruction count when executing the programs in cpp. As such, we provide two different instruction counts in order to verify that your program is working perfectly.

Application	IC_{5stage}
noio	2081
file	95215
hello	95705
class	97177
sort	104924
fact12	110820
matrix	137286
hanoi	201667
ical	216479
fib18	305749

4 Memory Timing

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 cycel, the cache checks to see if the data is available. Since it isn't it makes 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.

3. Return word 2 to cache Since the data bus is only wone word (32 bits / 4 bytes) wide, only one word is available on each cycle. To retrive the entire line will take another seven cycles.

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 can assume that this is because of an 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 Files

All source code will be provided on BlackBoard in a tarball/zip format. The source code contains the following folders.

- **verilog/** This contains the verilog design files for the processor. All necessary changes for this project will be made here. Additionallity, 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 direction of the makefile and typing 'make'. For those using linux systems, there should be no problem at all. For those using Mac systems, compilation will result in a lot of warnings due to deprecations, but they will not affect the functionality of you program. If you wish, you can request access to an ECE machine so that you may compile your project remotely (or in person) on a linux based machine. This also applies to Windows users, who will be unable to make the project without special tools.

Note for Windows users: You can compile this successfully using cygwin and by downloading the linux subsystem for Windows online, so it is not *strictly* necessary to compile on a remote system. If you have trouble getting this to work, we encourage you to come to office hours.

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 like so

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

to run for 12345 cycles.

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 Turn-in

Your submission should include the following files in your turn in, which should be made in the form of a tarball or zip file of some kind.

- **verilog/**: Naturally, you will need to submit your source code.

- **report.pdf**: You must provide a project report explaining what you did, and explaining 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. This must be submitted as a PDF.

- **sim_main/**: While you are not required to change anything about sim_main, you must still submit it as part of your project.

- **Makefile**: You must submit this for the same reason as above.