ECE 201/401: Project 3 MIPS Out-of-Order Core

Due Date: December 18, 2019 11:59p—Don't Be late!

Tuesday 12th November, 2019

1 Introduction

In this project you are going to get familiar with designing an out of order MIPS processor. The code for in-order processor with caches is provided. You are free to use your platform from the previous project if you made it pass all the test programs [1]. An illustrative example of the desired architecture is shown in Table 1:

Fetch/ Decode/ Rename/ Issue/ Commit Widths	1/1/1/1
Branch Prediction	Always Not Taken
Decode Queue	8-entry, FIFO
Rename Queue	8-Entry, FIFO
Renaming	Dual-RAT, No Checkpointing
Issue Queue	16-entry, compacting
Load/Store Queue	16-Entry, FIFO
Physical Register File	64 entry
Functional Units	1 ALU
ROB	64 entries

You may form a group of **at most two (2)** students to finish this project [2], 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.
- Convert the existing in-order core to an out-of-order core.

[1] If you do use your code from project 2, be sure to modify the Makefile to set PROJECT=4

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

- Report all issues regarding implementing out of order that you solve in your project (and how).
- Create a project report (details are enumerated 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.

Here are the instruction counts that each program should execute[3].

Application	Instruction Count
noio	2081
file	95215
hello	95705
class	97177
sort	104924
fact12	110820
matrix	137286
hanoi	201667
ical	216479
fib18	305749

4 Getting Started

Download and unpack the project files using the command line below:
tar -xvzf ECE401-Project3-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.

[3]This does not equate to number of cycles, as those will be different due to your core.

- 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'.

5 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).

6 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.

7 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. Build the necessary data structures (FIFOs, RATs, Issue Queues, ROB, and Physical Register File) early, and extend them as needed
3. Draw a timing diagram for how the core will operate.
4. Iterate on modification and verification of the design using the provided test programs.
5. Use $display to print out messages on the screen as needed.
6. Take advantage of the asm test programs to diagnose why the pipeline may be malfunctioning.
7. 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)
8. System calls will probably work best if caches are flushed first...
9. Both partners should contribute

8 Submission

You are to electronically turn in the following via Blackboard:

1. Include a README file, in pdf 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[4].
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)

[4]Both partners will receive the same grade

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.

You are to compress and archive your project 3 folder using the following command and naming convention and upload the generated file, along with your report, onto blackboard.

tar -cvzf FirstInitialLastName_Project3.tar.gz {*Name of Directory to ZIP*}

9 Extra Credit

For this project, there are three (3) extra credit opportunities each worth 250 points each (100% of the project-the total value of this project). (recall the project accounts for roughly 25% of your total course grade).

- Dual-Issue super scalar processor
- Lock-up Free/non-blocking caches
- Separate load and store queues, with load-store forwarding and speculation

These are simply some examples, if you have more ideas that you would like to implement, talk to Ryan.