ECEC 355 – Computer Architecture

Pipelining RISC-V Simulation with Data Forwarding

Instructor: Dr. Anup Das

Distributed, Intelligent, and Scalable COmputing (DISCO) Lab

ECE Department

Drexel University

February 11, 2019

1. Objective

This project is intended to be a comprehensive introduction to pipelining RISC-V simulation. There are three parts to this project:

- In part one, you will design a pipelining RISC-V simulator to be able to simulate a non-hazard program.
- In part two, you will extend your simulator with a hazard detection unit to detect control hazard as well as data hazard.
- In part three, you will further extend your simulator with a forwarding unit.

Please submit your work by March 15, 2019, 11:59 pm, via Bblearn. You may work on this project in teams of up to two people.

2. Required Reading

Chapter 4, The Processor, Sections 4.5 – 4.7

3. Tutorials on Pipelining RISC-V Simulator Code Template

3.1 Overview

The code template of a pipelining RISC-V simulator under *src/pipeline* has been given. The only difference compared to the single-cycle RISC-V simulator is the change of hierarchical level of Core. Instead of designing and simulating the circuit in Core, there are five new Classes – IF_Stage, ID_Stage, EX_Stage, MEM_Stage and WB_Stage (all defined in Stages.h) that take over this responsibility.

Stages{.h, .cpp} defines all the five stages. A partially functioning hazard detection unit and partial prototype of stage registers have been provided, as mentioned in the comments, for demonstration, we assume all the instructions are R-type, so it will be your responsibility to complete them in order to support all types of instructions and get a completely functioning pipeline in the end.

If you examine all of the stage definitions, you will find each stage holds an Instruction-type pointer (iterator), for example, Stages.h, line 102. The reason is that it makes the adjustment of end execution time easily, for example, Stages.cpp, line 89, once a stall is detected, the end execution time of the corresponding instruction should be incremented by one.

3.2 Compile and Run

Navigate to the root directory of DREXEL-DISCO-RISC-V-Simulator and type bash compile.bash pipeline. Run the simulator with the sample trace file ***cpu_traces/pipeline_debugging_with hazard*** to visualize how hazard detection unit works. You can disable this visualization by setting the DEBUG flag to 0 (In Core.h, line 12).

4. What to do

- Complete all the five stages and simulate cpu_traces/task_0 with initialization:
  - x1 = 0; x2 = 10; x3 = -15; x4 = 20; x5 = 30; x6 = -35
  - 40(x1) = -63, 48(x1) = 63
- Integrate your pipelining simulator with a hazard detection unit to detect control hazard as well as data hazard. Conventionally, the zero signal is generated in the EX stage leading to two potential flushes for instructions followed by a conditional jump. Please add a comparator and modify corresponding stages to make decisions in ID stage, in this case, only one flush is needed.
  - Simulate cpu_traces/task_1 with initialization:
    - x1 = 8; x3 = -4; x5 = 255; x6 = 1023
  - Simulate cpu_traces/task_3 with initialization:
    - x1 = 0; x2 = -5; x5 = -10; x6 = 25
    - 100(x7) = -100
  - Simulate cpu_traces/task_3 with initialization:
    - x1 = 8; x2 = -5; x5 = -10; x6 = 25
    - 100(x7) = -100

- Integrate your pipeline simulator with a forwarding unit.
  - Simulate cpu_traces/task_1 with initialization:
    - x1 = 8; x3 = -15; x5 = 255; x6 = 1023
  - Simulate cpu_traces/task_2 with initialization:
    - x5 = 26; x6 = -27;
    - 20(x1) = 100

5. Submissions

Zip the followings and submit through Bblearn:

- Report on how you complete the five stages
- Report on how you implement the hazard detection unit
- Report on how you implement the forwarding unit
- src/pipeline