University of Connecticut
CSE 4302 / CSE 5302/ ECE 5402: Computer Architecture
Fall 2024

Programming Assignment 2:
Pipelined RISC-V Simulator with Control-Flow Instructions and Forwarding

Due October 3, 2024 @ 11:59 PM on HuskyCT

Ensure that your git repository is up-to-date by executing `git pull` within the `cse4302` directory. This will create a new `pa2` directory in the repository root that contains the materials for this programming assignment. There are several source code files in the `src` directory, but you will **only** modify `sim_stages.c` for this assignment; **you are not allowed to modify any other files in the `src` and `unittests` directories or the two Bash scripts.**

You will modify `sim_stages.c` to implement a fully functional 5-stage pipelined processor simulator for the *riscv-uconn* ISA. Your simulator implementation must be functional and terminates for all unit tests in the `unittests` directory. The `dump_all.sh` script will automatically execute each assembled unit test and gather the required outputs in a single `pa2_out.txt` file. You are encouraged to write and test your own unit tests, but they will not contribute to your grade for PA2.

In addition to the RISC-V instructions from PA1, this assignment will integrate the following instructions.

1. I-Type Instructions: JALR
2. B-Type Instructions: BEQ, BNE, BLT, BGE
3. J-Type Instructions: JAL

1 Support for Control-flow Instructions

The first part of this assignment is to to modify the `decode()`, `execute()`, `memory()`, and `writeback()` functions to support **ITYPE**, **BTYPE** and **JTYPE** instructions. Refer to the **Introduction to riscv-uconn** document for more details. To support control flow instructions, two new pipeline global variables are added (Figure 1.1). These variables are accessible to each pipeline stage and must be updated correctly to implement control flow instructions.

Pipeline Control Variables	Description
`j_taken`	Set when an unconditional branch is taken
`br_mispredicted`	Set when a conditional branch is taken

Figure 1.1: Global variables for control flow instructions.

The **fetch stage** must detect the change in control flow by observing the `j_taken` and `br_mispredicted` global signals. When any of these signals is set to '1', it implies that the fetch stage must create a pipeline bubble by injecting a `nop`.

The **decode stage** asserts `j_taken` when an unconditional branch instruction (JAL or JALR) changes the program control flow. When the global signal `br_mispredicted` is set to '1' the decode stage must create a pipeline bubble by injecting a `nop`.

The **execute stage** asserts `br_mispredicted` if the branch is taken for the B-type instructions. In this case the `advance_pc()` function is used to update the program counter to computed branch address.

Make sure the pipeline interlock logic from PA1 is updated for the control flow instructions.

2 Support for Forwarding

The second part of this assignment is to add forwarding to the RISC-V pipeline. When forwarding is enabled, the machine will enable forwarding paths from the execute, memory, and writeback stages to the decode stage. When forwarding is not enabled, the machine will interlock when a hazard is detected. Pipeline-related control variables responsible for forwarding are added in `sim_core.h` and listed in Figure 2.1.

Pipeline Control Variables	Description
`forwarding_enabled`	Enables / disables the forwarding paths
`dout_exe, dout_mem, dout_wb`	Holds the value of forwarded data
`lw_in_exe`	Set when a load word instruction in the execute stage

Figure 2.1: Global variables for forwarding logic.

When forwarding is enabled in the **decode stage**, the forwarded data values in `dout_exe`, `dout_mem`, or `dout_wb` are used to avoid interlocking when a RAW hazard is detected. A load word (LW) instruction in the execute stage does not have dependent data to forward. Thus, the decode stage must assert `pipe_stall` when `lw_in_exe` global signal is asserted, and the corresponding dependency is detected.

The **execute stage** sets `dout_exe` to the ALU output value so it is forwarded to the decode stage. Similarly, the **memory stage** sets its output in `dout_mem`, and **writeback stage** sets its output in `dout_wb` so the respective values are forwarded to the decode stage.

The simulator executable accepts a program argument that controls whether forwarding is enabled or not:

```
$ ./simulator OUT_FILE FORWARDING_ENABLED
```

where `OUT_FILE` is an assembled *riscv-uconn* program and `FORWARDING_ENABLED` is either 0 or 1 to indicate that forwarding is disabled or enabled respectively. The forwarding enabled status is visible in the `sim_stages.c` file through the `forwarding_enabled` variable declared in `sim_core.h`.

To capture the output for PA2, run the following command:

```
./dump_all.sh
```

When you have completed the programming assignment, submit your `sim_stages.c` and `pa2_-out.txt` files via HuskyCT.