RPTU Kaiserslautern-Landau
Fachbereich Elektrotechnik und Informationstechnik
Entwurf informationstechnischer Systeme

Assignment 5 Pipelined RISC-V Core with Forwarding

The fifth assignment adds a forwatrding unit and a logic for hazard detection to the 5-stage pipeline from the previous task. The implemented ISA will stay the same simplified subset of RV32I as before. The pipeline is extended to auomatically resolve certain types of hazards and execute the given RV32I subset without any stalling.

Structure of the Project

Before you start with the tasks, make yourself familiar with the directory for the fifth assignment (05_pipelined_RISC-V_core_hazardDetection) in your forked repository. The Chisel project already contains a skeleton of the core. The actual implementation will be done in the core module (core.scala), while HazardDetectionRISCV32I.scala serves as a wrapper to connect the core tile to the outside world.

The wrapper interacts with the testbench in HazardDetectionRISCV32I_tb.scala. The testbench runs a simplified software program by passing the BinaryFile to the core. The core reads the content of the binary file into its instruction memory and executes it line by line. The test program in BinaryFile consists of 32-bit RISC-V assembly instructions encoded in HEX. BinaryFile_dump does not contribute to the testbench, but lists the assembly instructions in a human-readable way.
Remember, the testbench itself (HazardDetectionRISCV32I_tb.scala) only checks the expected results of the instructions and runs the clock. When adding new instructions to the test program, think of the result you expect from the core and add the corresponding check to the testbench.

Specification

In a pipelined processor, instructions that follow each other may have data dependencies. For instance, one instruction might write to a register that a subsequent instruction reads from. Without a way to detect and resolve these hazards, this would cause pipeline stalls, delaying execution until the data is available. A forwarding unit resolves data hazards in pipelined RISC-V cores by forwarding values from later pipeline stages (EX, MEM, WB) to earlier ones, allowing instructions to execute without stalling, even when they depend on the results of previous instructions.

Add a forwarding unit to your pipelined design from task 4, and stick to the design presented in the lecture (slide 6-24ff, schematic on slide 6-26)

Task 5.1 Preparation

Before you start with the implementation, please answer the following questions:

1. What signals are required as inputs / outputs to the forwarding unit?
2. What types of hazards can be solved by forwarding?
3. Which pipeline stages are connected to the forwarding unit and how can potential hazards be detetced?
4. Take a look at ADS I slide 6-24. What does it tell you about simultaneous read and write requests to the register file? How can the desired behaviour be implemented in the register file from task 4?
5. Are there any other potential hazards left in your specific implementation that would require stalling?

Task 5.2 Implementation

Implement the forwarding unit according to the specification in the lecture by using the provided Chisel project.

Task 5.3 Test your Implementation

Check whether your design runs and produces correct results. The test bench is located in HazardDetectionRISCV32I_tb.scala, test cases are provided as HEX instructions ("Binary File") in the "programs" folder.