HighRISCHighReward

CP2 Progress Report

Work Distribution

One person works on implementation of data forwarding, one person works on stalling, including control hazards and cache miss and data hazards. Another person works on the arbiter. We all work to combine the codes together and test our code. We all work together to design advanced features.

Implementation

For the forwarding, we add a module called forward_control_unit.sv to receive some control words from pipeline stage DE, EX, MEM, WB and decide the mux control signals in the EX, MEM, and ID stages, which decide whether those stages will utilize forwarded values or not.
The forwarding paths we implemented were MEM-EX, WB-EX, EX-ID, MEM-ID. From the MEM stage, we forwarded ALU-out, for ALU operations, BR-en, for SLT and its variants, as well as PC + 4, for a JALR.
For the stalling detection, we have a stall_control_unit.sv module to help us find all the conditions that we should stall. For example, when we are reading a register for branch instruction that is the destination register for previous load or ALU instruction, we determine there should be stalling. To implement this, bubbles are pushed in from the execution stage when the branch instruction is in the decode stage until the load instruction is in the WB stage and has finished reading in MEM stage, and then our transparent register file will remove the LD-BR hazard. For ALU operations which have a destination register that is the same as a Branch source register, we ‘insert’ one bubble, and then use our MEM-ID forwarding.

For the advanced features, these are the ones we have designed in this checkpoint. We did the paper design together.

1. Pipelined L1 caches [6]
2. L2+ cache system [2]
3. Local branch history table [2]
4. Global 2-level branch history table [3]
5. Tournament branch predictor [5]

6. RISC-V M Extension: A basic multiplier design is worth [3] while an advanced multiplier is worth [5]

Testing and Verification

First, we combine our code for data forwarding and hazard detection and stalling algorithms together and run with magic memory. We also run the code with our MP2 code and take a snapshot of final stages' register values in regfile for each test case. Then, we compare these snapshots with the final stages' register value executed from our pipelined code and see if we got them correct. After we passed the first two competition codes and stuck on the third one, we used the correctly passed test cases to hook up the RVFI correctly first. Then, we use the RVFI monitor to compare with our result in each instruction committed, under the third test case, and in this way, the debugging is easier.

Fmax & Energy report from Design Compiler

Combinational Delay per Stage: 4.3 ns
Latch Delay per Stage: 0.14 ns
Total Delay per Stage: 4.44 ns
Fmax = 1/(3.92 ns) = 226 MHz