컴퓨터 구조

PA3 Write-up

2013-11826 임주경

1. Introduction

RISC-V (64-bit) pipeline 을 직접 구현하며, Pipeline 의 개념과 Hazard 를 해결하기 위한 Stall, Forwarding 을 이해한다.

2. Implementation

- Problem 1. Bit instructions

AND, OR, XOR operation 의 구현을 위해서 control.v 의 opcode == R_FORMAT 상황의 구문을 수정하였다. alu_operation 에 할당되는 값을 funct3 의 판별을 통해 AND, OR, XOR 의 ALU opcode 를 할당하였다.

ANDi, ORi, XORi operation 도 마찬가지로, control.v 의 opcode == I_FORMAT 상황의 구문을 수정하였다. alu_operation 에 할당되는 값을 funct3 의 판별을 통해 AND, OR, XOR 의 ALU opcode 를 할당하였다.

- Problem 2. Hazard detection

Problem 4 의 Forwarding 구현을 통해, Data Hazard 를 해결하였다. (Problem 4 구현 부분에서 설명) Load-Use Hazard 가 발생했을 때의 1-cycle stall 을 위해 control.v 에 새로운 input hazard_detected 를 추가하였고, hazard_detected 가 1 의 값을 가질 때, 모든 control output 의 값을 0 으로 할당하였다.

riscv_pipeline.v 의 코드도 수정하였다. 새로운 register hazard_detection 을 추가하였다.

다음과 같은 상황에서 hazard_detection 이 1 의 값을 갖고, control unit 에 전달하게 된다. (EX_mem_read == 1'b1) && (EX_rd != 4'b00) && ((EX_rd == ID_rs1) || (EX_rd == ID_rs2)))에서 Load-Use Hazard 가 발생한다고 판별한다. Hazard detection 이 판별되면, 1-cycle stall 을 위해 pc 의 Fetch 를 중단하고, ID 에 전달하는 instruction 도 변화없이 유지한다.

- Problem 3. Branch instruction (BEQ)

BEQ operation 구현을 위해 control.v 의 코드를 수정하였다. Control unit 의 input opcode 가 BEQ 와 일치하면, 그에 맞는 control output 값을 할당하며, alu_operation 은 두 register 값의 비교를 위해 SUB 연산 코드를 할당한다.

inst_decoder.v 의 코드는 다음과 같이 수정한다. 우선, instruction 을 읽고 branch 명령어인지 판별하여 pipeline 에 전달하는 output branch_check 값을 추가하였다. imm64 의 경우 BEQ instruction 의 규칙에 해당하게 값을 할당한다.

riscv_pipeline.v 의 코드는 다음과 같이 구현한다. 3-cycle stall 을 위해 instruction fetch 를 중단해주기 위한 control 변수 ID_branch_check, EX_branch_check, MEM_branch_check 과

branch의 jump를 위해 필요한 MEM_imm64 register를 추가하였다. ID_branch_check는 wire로 설정하여, instruction decoder unit의 output branch_check의 값을 할당하였고, 각 stage의 branch_check register는 1 clock시에 다음 stage에 값을 전달한다. Branch instruction이 판별되면, pc를 현재의 pc상태로 유지하도록하며, ID에 전달하는 instruction은 NOP이 실행되도록 32'h0을 할당하였다. MEM stage에서 branch_check와 MEM_zero가 모두 1이 될 때, jump가 실행되야 함을 판별하여, 다음 pc에 pc + 2 * imm을 전달한다. 만약 그렇지 않다면, pc + 4의 instruction이 실행되도록 구현하였다.

- Problem 4. Forwarding unit

Forwarding 구현을 위해 riscv_pipeline.v 의 코드에 3-bit register forward_A 와 forward_B를 추가하였다. 각각은 rs1, rs2의 forwarding control을 위한 register이며, 3'b000의 경우 default, 3'b001의 경우 MEM_alu_result를 EX_rs에 전달하며, 3'b010의 경우 EX_alu_result를 EX_rs에 전달하며, 3'b011의 경우 Load-Use Hazard의 forwarding을 위해, MEM_read_data의 값을 EX_rs에 전달한다. 그리고, 3'b100의 경우 WB_write_data의 값을 EX_rs에 전달한다.

3. Result

bit.dat, hazard_ctl.dat, branch.dat의 구현 결과, 올바른 register, memory 값을 얻을 수 있었다.

별도로, test 코드를 통해 beq, load와 같은 instruction의 forwarding과 stall을 구현해 본 결과, 올바른 값을 얻을 수 있었다.