2024 Spring CSED311 Lab 4-1 Report

Team ID: 67735
20220312 박준혁, 20220871 홍지우

명예서약 (Honor Code)

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

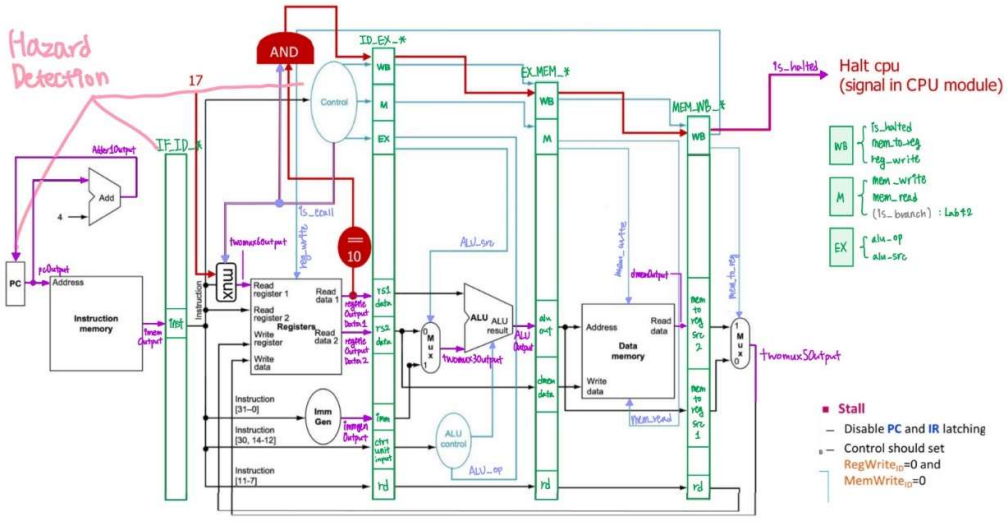
I completed this programming task without the improper help of others.

1 Introduction

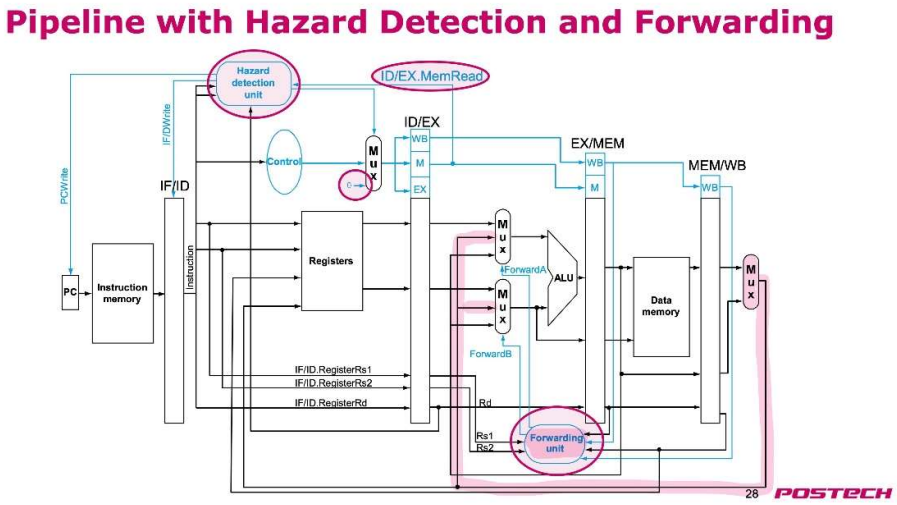
1.1 5-stage Pipelined CPU를 구현하였다. Hazard Detection이 이루어진다. Stall과 Data Forwarding이 적용된다.

2 Design

2.1 Hazard Detection은 아래 그림과 같이 설계하였다.



2.2 데이터 포워딩은 다음 그림을 따랐다.



3 Implementation

3.1 Pipeline 구현

우선 스테이지 사이마다 파이프라인 레지스터를 설치하였고,

다음 그림과 같이 매 clk에 맞추어 한 단계마다 값을 업데이트하며 넘겨주었다.

```
// Update ID/EX pipeline registers here
always @(posedge clk) begin
    if (reset) begin
        //ID_EX_alu_op <= 0;
        ID_EX_alu_src <= 0;
        ID_EX_mem_write <= 0;
        ID_EX_mem_read <= 0;
        ID_EX_mem_to_reg <= 0;
        ID_EX_reg_write <= 0;

        ID_EX_rs1_data <= 0;
        ID_EX_rs2_data <= 0;
        ID_EX_imm <= 0;
        ID_EX_inst <= 0;
        ID_EX_rd <= 0;
        ID_EX_is_halted <= 0;
        ID_EX_rs1 <= 0;
        ID_EX_rs2 <= 0;
    end
```

```
    else begin
        //ID_EX_alu_op <= ALU_op;
        ID_EX_alu_src <= ALU_src;
        ID_EX_mem_write <= mem_write;
        ID_EX_mem_read <= mem_read;
        ID_EX_mem_to_reg <= mem_to_reg;
        ID_EX_reg_write <= reg_write;

        ID_EX_rs1_data <= rs1_dout_forwarded;
        ID_EX_rs2_data <= rs2_dout_forwarded;
        ID_EX_imm <= immgenOutput;
        ID_EX_inst <= IF_ID_inst;
        ID_EX_rd <= IF_ID_inst[11:7];
        ID_EX_is_halted <= _is_halted;
        ID_EX_rs1 <= rs1;
        ID_EX_rs2 <= rs2;
    end
    if (is_hazard) begin
        ID_EX_reg_write <= 0;
        ID_EX_mem_write <= 0;
        ID_EX_rd <= 5'b0;
    end
nd
```

hazard가 발생했을 때 stall을 적용하는 코드도 나타나 있다.

3.2 Hazard Detection

발생 조건은 Data Forwarding이 적용되어 있기 때문에 아래 로직을 따랐다.

- Even with data-forwarding, a hazard occurs for RAW dependence on an immediately preceding LW instruction
- Stall = { [($rs1_{ID}$ == rd_{EX}) && use_rs1(IR_{ID})] || [($rs2_{ID}$ == rd_{EX}) && use_rs2(IR_{ID})] } && $MemRead_{EX}$
  - i.e., op_{EX}=LW/LH/..

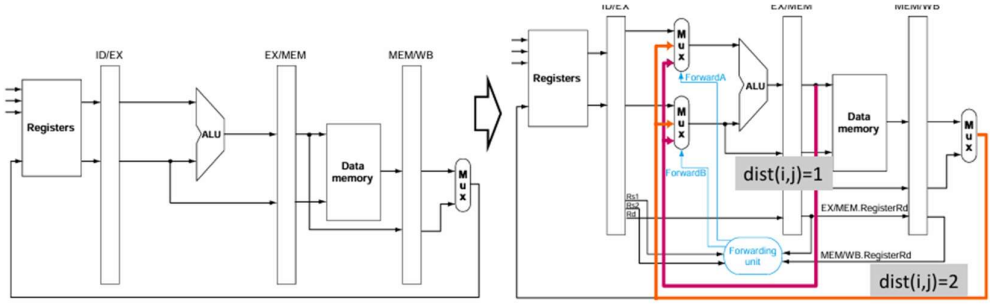
코드는 다음과 같다.

ecall이 발생했을 때, rs1과 rs2에 대해 hazard가 발생하는 경우를 체크해 주었다.

```
always @(*) begin
    is_hazard = 0;

    // ecall hazard check
    if (is_ecall) begin // (IF_ID_opcode == `ECALL) begin
        if (ID_EX_rd == 17 && ID_EX_reg_write || EX_MEM_rd == 17 && EX_MEM_reg_write) begin
            is_hazard = 1;
        end else begin
            is_hazard = 0;
        end
    end

    // rs1 hazard check
    if (IF_ID_opcode == `ARITHMETIC || IF_ID_opcode == `ARITHMETIC_IMM || IF_ID_opcode == `LOAD || IF_ID_opcode == `STORE) begin
        if (IF_ID_rs1 == ID_EX_rd && ID_EX_rd != 0 && ID_EX_mem_read) begin
            is_hazard = 1;
        end else begin
            is_hazard = 0;
        end
    end

    // rs2 hazard check
    if (is_hazard == 1) begin
        // skip rs2 hazard check if rs1 hazard is detected
    end else if (IF_ID_opcode == `ARITHMETIC || IF_ID_opcode == `LOAD || IF_ID_opcode == `STORE) begin
        if (IF_ID_rs2 == ID_EX_rd && ID_EX_rd != 0 && ID_EX_mem_read) begin
            is_hazard = 1;
        end else begin
            is_hazard = 0;
        end
    end
end
```

3.3 Data Forwarding

포워딩 조건은 다음과 같다.



No forwarding　　With forwarding

23 POSTECH

Forwarding Logic

if ($rs1_{EX}$!=x0) && ($rs1_{EX}$==rd_{MEM}) && $RegWrite_{MEM}$ then
    forward operand from MEM stage　　// dist=1

Can this be done always?

Else if ($rs1_{EX}$!=x0) && ($rs1_{EX}$==rd_{WB}) && $RegWrite_{WB}$ then
    forward operand from WB stage　　// dist=2

else
    use the operand from register file　　// dist≥3

◆ Do the same for rs2

위 그림대로 작성한 코드는 다음과 같다.

```
always @(*) begin
    // rs1 Forwarding
    if (opcode == `ARITHMETIC || opcode == `ARITHMETIC_IMM || opcode == `LOAD || opcode == `STORE) begin
        if (rs1 == dist1_rd && dist1_rd != 0 && dist1_reg_write) begin
            forwardA = 2'b01; // Distance 1 Forwarding
        end else if (rs1 == dist2_rd && dist2_rd != 0 && dist2_reg_write) begin
            forwardA = 2'b10; // Distance 2 Forwarding
        end else begin
            forwardA = 2'b00; // No Forwarding
        end
    end else begin
        forwardA = 2'b00;
    end

    // rs2 Forwarding
    if (opcode == `ARITHMETIC || opcode == `LOAD || opcode == `STORE) begin
        if (rs2     == dist1_rd && dist1_rd != 0 && dist1_reg_write) begin
            forwardB = 2'b01; // Distance 1 Forwarding
        end else if (rs2 == dist2_rd && dist2_rd != 0 && dist2_reg_write) begin
            forwardB = 2'b10; // Distance 2 Forwarding
        end else begin
            forwardB = 2'b00; // No Forwarding
        end
    end else begin
        forwardB = 2'b00;
    end
end
endmodule
```

아래는 포워딩되는 부분의 예시이다.

addi x15 x0 14

sw x15 -24 x8

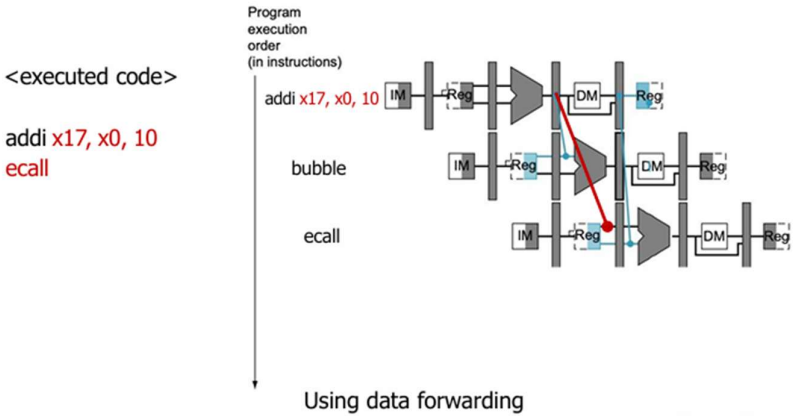
lw x14 -24 x8

이때 x14에 들어갈 값이 앞에서 포워딩되어 (forwardB=01: dist 1 forwarding)

stall 없이 들어간 모습이다.

3.4 Ecall Forwarding

아래 그림과 같이 동작하도록 하였다.



코드는 다음과 같다.

```
always @(*) begin
    if((rs1 == rd) && (rd != 0)) begin
        rs1_dout_forwarded = rd_din;
    end
    else if((EX_MEM_rd == 5'd17) && is_ecall) begin
        rs1_dout_forwarded = EX_MEM_alu_out;
    end
    else begin
        rs1_dout_forwarded = rs1_dout;
    end

    if((rs2 == rd) && (rd != 0)) begin
        rs2_dout_forwarded = rd_din;
    end
    else begin
        rs2_dout_forwarded = rs2_dout;
    end
end
```

4 Discussion

4.1 레지스터를 새로 만들고 매 클락마다 업데이트하면서도 많은 조건들을 체크하는 것이 어려웠다.

4.2 single cycle에서보다 8사이클정도 많이 나왔는데, 중간중간 stall도 있고 5단계로 나뉘어 앞뒤에 추가 사이클이 필요하기 때문이다. (2개 inst도 6사이클이나 필요할 수 있음) 그러나 한 사이클당 하나의 단계만큼의 시간이 걸리므로 multi-cycle의 짧은 cycle 주기와 single-cycle의 적은 사이클 수를 둘 다 가지고 있다고 할 수 있다.

5 Conclusion

5.1 테스트벤치 결과

```
### SIMULATING ###
TEST END
SIM TIME : 96
TOTAL CYCLE : 47 (Answer : 46)
FINAL REGISTER OUTPUT
 0 00000000 (Answer : 00000000)
 1 00000000 (Answer : 00000000)
 2 00002ffc (Answer : 00002ffc)
 3 00000000 (Answer : 00000000)
 4 00000000 (Answer : 00000000)
 5 00000000 (Answer : 00000000)
 6 00000000 (Answer : 00000000)
 7 00000000 (Answer : 00000000)
 8 00000000 (Answer : 00000000)
 9 00000000 (Answer : 00000000)
10 0000000a (Answer : 0000000a)
11 0000003f (Answer : 0000003f)
12 fffffff1 (Answer : fffffff1)
13 0000002f (Answer : 0000002f)
14 0000000e (Answer : 0000000e)
15 00000021 (Answer : 00000021)
16 0000000a (Answer : 0000000a)
17 0000000a (Answer : 0000000a)
18 00000000 (Answer : 00000000)
19 00000000 (Answer : 00000000)
20 00000000 (Answer : 00000000)
21 00000000 (Answer : 00000000)
22 00000000 (Answer : 00000000)
23 00000000 (Answer : 00000000)
24 00000000 (Answer : 00000000)
25 00000000 (Answer : 00000000)
26 00000000 (Answer : 00000000)
27 00000000 (Answer : 00000000)
28 00000000 (Answer : 00000000)
29 00000000 (Answer : 00000000)
30 00000000 (Answer : 00000000)
31 00000000 (Answer : 00000000)
Correct output : 32/32
```