EC 413

Lab 9 Write Up

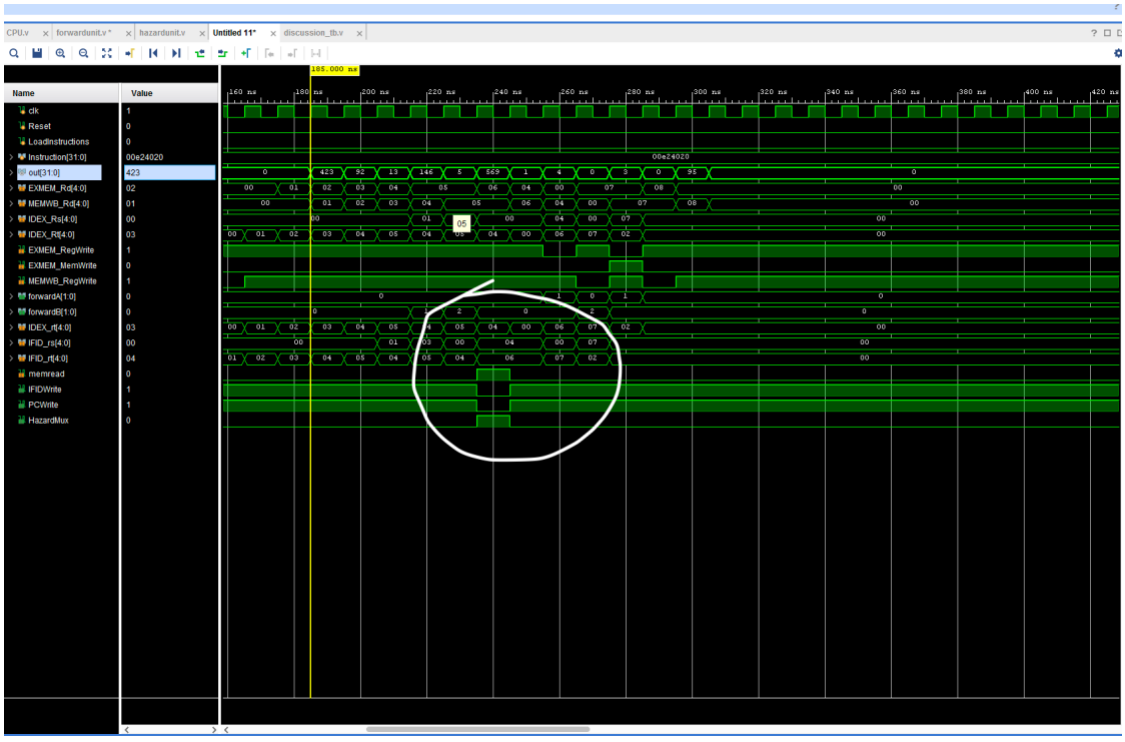
Dabin Jang

- Hazard Detection Unit

  For Hazard Detection Unit, I slightly modified from lab8 because I only need to detect when forwarding is not helpful. That's when load word instruction is used. Therefore, I used this logic that we learned from the class to catch the hazard.

```
assign IFIDWrite = (memread && (IDEX_rt == IFID_rs || IDEX_rt == IFID_rt)) ? 0 : 1;
assign PCWrite = (memread && (IDEX_rt == IFID_rs || IDEX_rt == IFID_rt)) ? 0 : 1;
assign HazardMux = (memread && (IDEX_rt == IFID_rs || IDEX_rt == IFID_rt)) ? 1 : 0;
```

  Basically, when **memread** is happening (which tells us that it's lw instruction) we compare rt value of ID/EX stage with rs, rt values of IF/ID stage. If at least one of them are equal, then there is a hazard that forwarding cannot help. We must stall the stage to solve the problem.



The waveform shows that there is one hazard detection which is after lw instruction. The hazard detection unit works properly.

- Forwarding Unit

  For forwarding unit, I use the same logic that we learned from the lecture. When there is one ahead and two ahead at the same time, then we always want one ahead value because we want the latest value.

```
    // output
    output reg [1:0] forwardA;
    output reg [1:0] forwardB;

    always@(*)
    begin
    forwardA = 2'b00;
    forwardB = 2'b00;

    if (EXMEM_RegWrite && (EXMEM_Rd != 0) && EXMEM_Rd == IDEX_Rs) // One Ahead
        begin
            forwardA = 2'b10;
        end
    else if (MEMWB_RegWrite && (MEMWB_Rd != 0) && MEMWB_Rd == IDEX_Rs) // Two Ahead
        begin
            forwardA = 2'b01;
        end
        else begin
            forwardA = 2'b00;
        end

    if (EXMEM_RegWrite && (EXMEM_Rd != 0) && (EXMEM_Rd == IDEX_Rt)) // One Ahead
        begin
            forwardB = 2'b10;
        end
    else if (MEMWB_RegWrite && (MEMWB_Rd != 0) && MEMWB_Rd == IDEX_Rt) // Two Ahead
        begin
            forwardB = 2'b01;
        end
    else
        begin
            forwardB = 2'b00;
        end
    end
endmodule
```

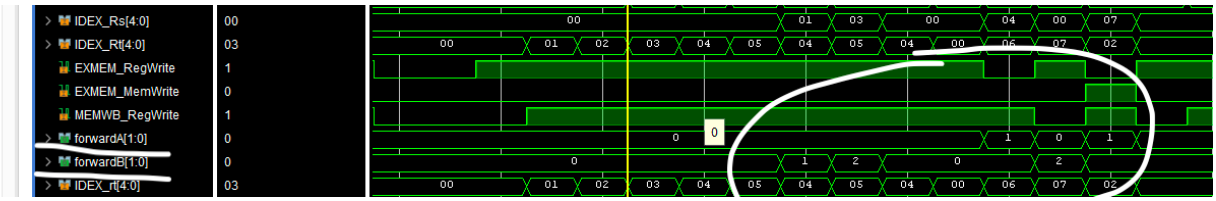
Basically, everything is the same as the lecture note. But there might be some cases where forwardA and forwardB are both not equal to 2'b00. That's why I separated into two if parts. One only deals with forwardA and the other one only deals with forwardB.

I wrote down where I expect one ahead and two ahead data dependecy on test bench.

This is the screenshot of my testbench.

```
        LoadInstructions = 1;
        #10;
        // begin by loading values into registers
        Instruction = 32'b001000_00000_00001_0000000110100111; //addi $R1, 423
        #10;
        Instruction = 32'b001000_00000_00010_0000000001011100; //addi $R2, 92
        #10;
        Instruction = 32'b001000_00000_00011_0000000000001101; //addi $R3, 13
        #10;
        Instruction = 32'b001000_00000_00100_0000000010010010; //addi $R4, 146
        #10;
        Instruction = 32'b001000_00000_00101_0000000000000101; //addi $R5, 5
        #10;
        Instruction = 32'b000000_00001_00100_00101_00000_100000; //add  $R5, $R1, $R4,   (569) two ahead - one NOP
        #10;
        Instruction = 32'b000000_00011_00101_00110_00000_101010; //slt  $R6, $R3, $R5,  (1) one ahead - two NOP
        #10;
        Instruction = 32'b100011_00000_00100_00000_00000_000100;    // LW $4, 4(R0)  (4)
        #10;
        Instruction = 32'b000000_00100_00110_00111_00000_100010; //sub  $R7, $R4, $R6
        #10;
        Instruction = 32'b101011_00000_00111_00000_00000_000000;    // SW R7, 0(R0) 0
        #10;
        Instruction = 32'b000000_00111_00010_01000_00000_100000;    // add R8, R7, R2  92  Two ahead - one NOP
        #10;
        LoadInstructions = 0;
        Reset = 1;
        #10;
        Reset = 0;
        #100;
        // Add stimulus here
```

And whenever I expect one ahead and two ahead data dependecy, there is proper value of 2 bits forwardA and forwardB go into the mux.



For Example, after addi $R5, we have add instruction of $R5 $R1 $R4. It has two ahead data dependency. And MEMWB_Rd = IDEX_Rt. Therefore, forwardB must 01 and forwardA must be 00. It's cleary shown in the waveform. Everything works properly.