CS 224

Fall 2023–2024

Lab 5 Preliminary Report

Hüseyin Uzun

21702559

Section 3

Part 1b)

Data Hazard

i. Compute-use hazard

Memory and writeback stages are affected because store word instruction gets the register in the memory stage. Before the first instruction finished and the computed data is written to the register itself back, the register is used in the following instruction. Thus, the wrong data will be gotten from the register's destination.

The compute-use hazard happens when the following instruction uses the result of the R type instruction. As a solution, data forwarding is the most efficient one because there is no wasted time compared to stalling. These solutions can be both used.

ii. Load-store hazard

Execute and memory stages are affected. Just after the load instructions complete writing to the register, following instructions try to read it from the same register. Thus, the previous data will be read by the load word instruction. The solution should be stalling.

iii. Load-use hazard

Decode and execute stages are affected. Just after the store word instructions complete writing to the register, load word instructions try to read it from the same destination. Thus, the previous data will be read by the load word instruction. The solution should be stalling.

Control Hazard

Branch hazard

Branch not determined until memory stage and instructions after the branch are fetched before the branch occurs. The solution should be stalling.

Part 1c)

Forwarding logic:

```
if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
        then    ForwardAE = 10
else
        if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
                then    ForwardAE = 01
        else    ForwardAE = 00
```

```
if ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM)
        then    ForwardAE = 10
else
        if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)
                then    ForwardAE = 01
        else    ForwardAE = 00
```

Stalling Logic:

```
        lwstall = ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE
        StallF = StallD = FlushE = lwstall
```

Part 1d)

```
module PipeDtoE(input logic RegWriteD, MemtoRegD, MemWriteD, AluSrcD, RegDstD,
BranchD,
                input logic[2:0] AluControlD,
                input logic[31:0] RD1D, RD2D, PCPlus4D,
                input logic[4:0] RsD, RtD, RdD,
                input logic[15:0] SignImmD,
           input logic CLR, clk,          // FlushE will be connected as this CLR
           input logic RegWriteE, MemtoRegE, MemWriteE, AluSrcE, RegDstE, BranchE,
                input logic[2:0] AluControlE,
                output logic[31:0] PCPlus4E, RD1E, RD1E,
                output logic[15:0] SignImmE,
                output logic[4:0] RsE, RtE, RdE);


           always_ff @(posedge clk, posedge CLR)
              if(CLR)
                 begin
                        RegWriteE<=0;
                                MemtoRegE<=0;
                                MemWriteE<=0;
                                AluControlE<=0;
                                AluSrcE<=0;
                                RegDstE<=0;
                                BranchE<=0;
                                RD1E<=0;
                                RD2E<=0;
                                RsE<=0;
                                RtE<=0;
                                RdE<=0;
                                SignImmE<=0;
                                PCPlus4E<=0;
                 end
                   else
                        begin
                                RegWriteE<=RegWriteD;
                                MemtoRegE<=MemtoRegD;
                                MemWriteE<=MemWriteD;
                                AluControlE<=AluControlD;
                                AluSrcE<=AluSrcD;
                                RegDstE<=RegDstD;
                                BranchE<=BranchD;
                                RD1E<=RD1D;
                                RD2E<=RD2D;
                                RsE<=RdD;
                                RtE<=RdD;
                                RdE<=RdD;
                                SignImmE<=SignImmD;
                                PCPlus4E<=PCPlus4D;
                        end
```

```
endmodule

module PipeEtoM(input logic RegWriteE, MemtoRegE, MemWriteE, BranchE, ZeroE,
            input logic[31:0] ALUOutE, PCBranchE, WriteDataE,
            input logic[4:0] WriteRegE,
        input logic clk,
            output logic RegWriteM, MemtoRegM, MemWriteM, BranchM, ZeroM,
            output logic[31:0] ALUOutM, PCBranchM, WriteDataM,
            output logic[4:0] WriteRegM);
        always_ff @(posedge clk,posedge reset)
          if(reset)
                  begin
                        RegWriteM<=0;
                        MemtoRegM<=0;
                        MemWriteM<=0;
                        BranchM<=0;
                        ZeroM<=0;
                        ALUOutM<=0;
                        PCBranchM<=0;
                        WriteDataM<=0;
                        WriteRegM<=0;
                  end
              else
            begin
                        RegWriteM<=RegWriteE;
                        MemtoRegM<=MemtoRegE;
                        MemWriteM<=MemWriteE;
                        BranchM<=BranchE;
                        ZeroM<=ZeroE;
                        ALUOutM<=ALUOutE;
                        PCBranchM<=PCBranchE;
                        WriteDataM<=WriteDataE;
                        WriteRegM<=WriteRegE;
                  end

endmodule
module PipeMtoW(input logic RegWriteM, MemtoRegM,
            input logic[31:0] ALUOutE, PCBranchE,
            input logic[4:0] WriteRegM,
        input logic clk,
            output logic RegWriteW, MemtoRegW,
            output logic[31:0] ALUOutW, ReadDataW,
            output logic[4:0] WriteRegW);

        always_ff @(posedge clk,posedge reset)
          if(reset)
                  begin
                        RegWriteW<=0;
                        MemtoRegW<=0;
                        ReadDataW<=0;
                        ALUOutW<=0;
                        WriteRegW<=0;
                  end
              else
            begin
                        RegWriteW<=RegWriteM;
                        MemtoRegW<=MemtoRegM;
                        ReadDataW<=ReadDataM;
                        ALUOutW<=ALUOutM;
                        WriteRegW<=WriteRegM;
```

```
                    End

assign PCSrcM = BranchM & ZeroM;
        mux2 #(32)  pcmuxInit(PCPlus4F, PCBranchM,PCSrcM,PCI);
        PipeWtoF pipewtof(PCI, StallF, clk, PCF);
        assign PCPlus4F = PCF + 4;

        PipeFtoD pipeftod(instr, PcPlus4F, StallD,clk, instrD, PcPlus4D);

        regfile     rf(clk, RegWriteW, instrD[25:21], instrD[20:16], writeRegW,
        // Instantiated register file.
             resultW, RD1D, RD2D);
        signext sn(instr[15:0], SignImmD);
        assign RsD = instrD[25:21];
        assign RtD = instrD[20:16];
        assign RdD = instrD[15:11];

        PipeDtoE pipedtoe(RegWriteD, MemtoRegD, MemWriteD, AluSrcD, RegDstD,
BranchD,
                AluControlD,
                RD1D, RD2D, PCPlus4D,
                RsD, RtD, RdD,
                SignImmD,
           CLR, clk,
           RegWriteE, MemtoRegE, MemWriteE, AluSrcE, RegDstE, BranchE,
                AluControlE,
                PCPlus4E, RD1E, RD1E,
                SignImmE,
                RsE, RtE, RdE);

        mux2 #(5) mux2EWriteReg(RtE, RdE,
            RegDstE,
             WriteRegE);

        mux4 #(32) mux3ESrcAE(RD1E, ResultW, ALUOutM, 0,
             ForwardAE,
             SrcAE);

        mux4 #(32) mux3EWriteDataE(RD2E, ResultW, ALUOutM, 0,
          ForwardBE,
           WriteDataE);

        mux2 #(32) mux2ESrcBE(WriteDataE, SignImmE,
           AluSrcE,
           SrcBE);
        sl2 shiftSIE(SignImmE,
           SignImmEOut);
        adder adderPCBranchM(SignImmEOut, PCPlus4E, PCBranchE);

        alu alu1(SrcAE, SrcBE,
          AluControlE,
          ALUOut,
          ZeroE, reset);

PipeEtoM pipeetom(RegWriteE, MemtoRegE, MemWriteE, BranchE, ZeroE,
                ALUOutE, PCBranchE, WriteDataE,
                WriteRegE,clk,RegWriteM, MemtoRegM, MemWriteM, BranchM, ZeroM,
                ALUOutM, PCBranchM, WriteDataM,WriteRegM);

        dmem dmemIn( clk, MemWriteM,
            ALUOutM, WriteDataM,
                ReadDataM);

        mux2 #(32) mux2WResultW( ALUOutW, ReadDataW,
           MemtoRegW,
           ResultW);

        PipeMtoW pipemtow(RegWriteM, MemtoRegM,ALUOutE, PCBranchE,
                WriteRegM,clk,RegWriteW, MemtoRegW,
                ALUOutW, ReadDataW,WriteRegW);

        HazardUnit hu(RegWriteW,WriteRegW,RegWriteM,MemToRegM,WriteRegM,
 RegWriteE,MemToRegE,rsE,rtE,rsD,rtD,ForwardAE,ForwardBE,FlushE,StallD,StallF);

if ((rsE != 0) AND (rsE == WriteRegM) AND RegWriteM)
        then     ForwardAE = 10
else
        if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)
                then     ForwardAE = 01
        else     ForwardAE = 00

if ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM)
        then     ForwardAE = 10
else
        if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)
                then     ForwardAE = 01
        else     ForwardAE = 00

assign lwstall = ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE;
        assign StallF = lwstall;
        assign StallD = lwstall;
        assign FlushE = lwstall;
```

Part 1e)

Compute use hazard:

```
        addi    $s0, $zero, 2
        addi    $s1, $zero, 4
        addi    $s2, $zero, 3
        lw      $t0, 0($s0)
        add     $t1, $s0, $s1
        and     $t2, $t1, $s2
```

Load use hazard:

```
        addi    $s0, $zero, 2
        addi    $s1, $zero, 4
        addi    $s2, $zero, 3
        lw      $t0, 0($s0)
        addi    $t1, $t0, 2
```

No hazard:

```
        addi    $s0, $zero, 2
        addi    $s1, $zero, 4
        addi    $s2, $zero, 3
        lw      $t0, 0($s0)
        sub     $s3, $s1, $s0
        lw      $t2, 0($s0)
        addi    $s4, $t0, 3
```

Branch hazard:

```
        addi    $s0, $zero, 2
        beq     $s0, $zero, end
        addi    $s1, $zero, 4
end:
```