CS224
Lab No: 5
Section No: 4

Emine İrem Esendemir
22202103

b) LIST OF HAZARDS

- COMPUTE-USE:
  - Hazard type: Data
  - Affected Pipeline Stages: Execute, Memory, Writeback (Depends on the instructions order)
  - Solution: Forwarding

- LOAD-USE:
  - Hazard type: Data
  - Affected Pipeline Stages: Decode, Execute, Memory
  - Solution: Forwarding, Stalling (Depending on the lw’s position compared to the current instruction)

- J-TYPE JUMP:
  - Hazard type: Data
  - Affected Pipeline Stages: Fetch, Decode, Execute
  - Solution: Stalling, Flushing

- BRANCH:

- Hazard type: Data
- Affected Pipeline Stages: Fetch, Decode, Execute
- Solution: Stalling, Flushing

c) Hazard Unit Logic

```
//For data hazards
if ((rsE != 0) && (rsE == WriteRegM) && RegWriteM):
      ForwardAE = 10
else if ((rsE != 0) && (rsE == WriteRegW) && RegWriteW):
      ForwardAE = 01
else:
      ForwardAE = 00
if ((rtE != 0) && (rtE == WriteRegM) && RegWriteM):
      ForwardBE = 10
else if ((rtE != 0) && (rtE == WriteRegW) && RegWriteW):
      ForwardBE = 01
else:
      ForwardBE = 00

lwstall = ((rsD = = rtE) || (rtD = = rtE)) && MemtoRegE

//For control hazards
ForwardAD = (rsD != 0) && (rsD == WriteRegM) && RegWriteM
ForwardBD = (rtD != 0) && (rtD == WriteRegM) && RegWriteM

branchstall = (BranchD && (RegWriteE && ((WriteRegE == rsD) || (WriteRegE
== rtD))) || (MemToRegM && ((WriteRegM == rsD) || (WriteRegM == rtD))));

StallF = StallD = (lwstall || branchstall);
FlushE = (lwstall || branchstall||  JumpD);
```

d) Test Programs in MIPS

- A test program with no hazards:
- A test program with compute-use hazard:
- A test program with load-use hazard:
- A test program with J-type hazard:
- A test program with branch hazard

e) SystemVerilog Modules

• Hazard Unit

```
module HazardUnit( input logic RegWriteW,
                    input logic [4:0] WriteRegW,
                    input logic RegWriteM,MemToRegM,
                    input logic [4:0] WriteRegM,
                    input logic RegWriteE,MemToRegE,
                    input logic [4:0] rsE,rtE,
                    input logic [4:0] rsD,rtD,
                    input BranchD, JumpD,
                    output logic [2:0] ForwardAD,ForwardBD,
                    output logic [2:0] ForwardAE,ForwardBE,
                    output logic FlushE,StallD,StallF);

    logic lwstall, branchstall

    always_comb begin
        if ((rsE != 0) && (rsE == WriteRegM) && RegWriteM)
        ForwardAE = 10;
        else if ((rsE != 0) && (rsE == WriteRegW) && RegWriteW)
        ForwardAE = 01;
        else
        ForwardAE = 00;

        if ((rtE != 0) && (rtE == WriteRegM) && RegWriteM)
        ForwardBE = 10;
        else if ((rtE != 0) && (rtE == WriteRegW) && RegWriteW)
        ForwardBE = 01;
        else
        ForwardBE = 00;

        lwstall = MemToRegE && ((rsD == rsE) || (rsD == rsE));
        branchstall =(BranchD && RegWriteE && ((WriteRegE == rsD) ||
(WriteRegE == rtD))) ||
(BranchD && MemtoRegM && ((WriteRegM == rsD) || (WriteRegM == rtD)));

        ForwardAD = RegWriteM & (rsD != 0) & (rsD == WriteRegM );
        ForwardBD = RegWriteM & (rtD != 0) & (rtD == WriteRegM );

        StallF = (lwstall || branchstall);
        StallD = (lwstall || branchstall);
        FlushE = (lwstall || branchstall || JumpD);

    end
endmodule
```

• PipeDtoE

```
module PipeDtoE(
    input logic clk, reset, clear, // clear -> FlushE

    input logic RegWriteD, MemToRegD, MemWriteD, AluSrcD, RegDstD,
    input logic [2:0] AluControlD,
    input logic[31:0] RD1D, RD2D, SignImmD,
    input logic[4:0] RsD, RtD, RdD,

    output logic RegWriteE, MemtoRegE, MemWriteE, ALUSrcE, RegDstE,
    output logic[2:0] ALUControlE,
    output logic[31:0] RD1E, RD2E, SignImmE
    output logic[4:0] RsE, RtE, RdE
    );

    always_ff @(posedge clk or posedge reset) begin
        if(reset || clear) begin
            RegWriteE <= 0;
            MemtoRegE <= 0;
            MemWriteE <= 0;
            ALUSrcE <= 0;
            RegDstE <= 0;
            ALUControlE <= 0;
            RD1E <= 0;
            RD2E <= 0;
            SignImmE <= 0;
            RsE <= 0;
            RtE <= 0;
            RdE <= 0;
        end
        else begin
            RegWriteE <= RegWriteD;
            MemtoRegE <= MemToRegD;
            MemWriteE <= MemWriteD;
            ALUSrcE <= AluSrcD;
            RegDstE <= RegDstD;
            ALUControlE <= AluControlD;
            RD1E <= RD1D;
            RD2E <= RD2D;
            SignImmE <= SignImmD;
            RsE <= RsD;
            RtE <= RtD;
            RdE <= RdD;
        end
    end
endmodule
```

• PipeEtoM

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