CS224
Lab No.:5
Section No.: 1
Name: Atakan Akar
Bilkent ID: 22203140
Date: 27.11.2024

A)
Data Hazards:

TYPE	affected pipeline stages	the solution	What, when, how	reason
Compute-use	Decode stage of the instruction	Fowarding the data directly to execute stage of the next instruction	When r-type instruction is executed, and the subsequent instruction tries to access its destination register before its written.	If data computed in execute is not written back befor the subsequent ins. tries to read
Load-use	Execute and memory stages	Stalling the pipeline	If a data still being loaded is tried to be used by a subsequent instruction there fore error in that register that is aimed	If memory cannot be accessed
Load-store	The store instruction's memory stage	The pipeline is stalled until the load operation is fully completed.	when successive lw and sw use same rt	Data is loaded but stored in another memory location using the same register immitiately

Control Hazards:

TYPE	affected pipeline stages	the solution	What, when, how	reason
Branch	Three consecutive instructions	The pipeline is stalled until the branch decision is resolved to avoid fetching incorrect instructions.	Occurs during conditional branching when the branch decision is not resolved before the pipeline proceeds with subsequent instruction fetches	Branching desicions are not made in time therefore the process of pipelining is deficted
Jump	Two pipeline stages (IF,ID)	The pipeline is flushed to discard incorrectly fetched instructions when a jump instruction computes its target address late.	When the jump is to where the address is computed late	Target adress is not available

C)
Forwarding Equations:

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

if ((rsE != 5'd0) & (rsE == WriteRegM) & RegWriteM)
 ForwardAE = 2'b10
else if ((rsE != 5'd0) & (rsE == WriteRegW) & RegWriteW)
 ForwardAE = 2'b01
else
 ForwardAE = 2'b00

if ((rtE != 5'd0) & (rtE == WriteRegM) & RegWriteM)
 ForwardBE = 2'b10
else if ((rtE != 5'd0) & (rtE == WriteRegW) & RegWriteW)
 ForwardBE = 2'b01
else
 ForwardBE = 2'b00
```

Stall:

```
StallF = StallD = FlushE = (((rsD == rtE) | (rtD == rtE)) & MemtoRegE) |
        (BranchD & MemtoRegM & (WriteRegM == rsD | WriteRegM == rtD)) |
        (BranchD & RegWriteE & (WriteRegE == rsD | WriteRegE == rtD))
```

D)

Test1 (Data Hazard)):
Without Hazard:

```
And $s1,$t0,$t0
Add $s3,$s6,$s7

0x01088024
0x02D73820
```

With Hazard:

```
And $s1,$t0,$t0
Add $s3,$s1,$s7


0x01088024
0x02373820
```

Test2 (Data Hazard)):
Without Hazard:

Ori $t1,$t2,2
Add $t3,$s1,$t2

0x35490002
0x023A5820

With Hazard:
Ori $t1,$t2,2
Add $t3,$s1,$t1

0x35490002
0x02395820

Test3 (Branch Hazard)):

Without Hazard:
xor $t3, $a0, $a1
beq $t0, $t1, Label

0x00a54024
0x00a54826

With Hazard:
beq $t0, $t1, Label
and $t2, $a0, $a1
xor $t3, $a0, $a1
Label: add $t4, $t2, $t3

0x11090002
0x00a54024
0x00a54826
0x014b5020

Test4 (Load-Store Hazard)):
Without Hazard:
sw $t0, 0($t1)
lw $t2, 1($t0)
or $t3, $s2, $a0
xor $t4, $s2, $a1

0xad280000
0x8d0a0001
0x014a5025
0x014a5826

With Hazard:
sw $t0, 0($t1)

```
lw $t1, 1($t0)
or $t2, $t1, $a0
xor $t2, $t1, $a1

0xad280000
0x8d090001
0x01245025
0x01255026
```

E)

Hazard Unit)

```
module HazardUnit(
    input logic RegWriteW,
    input logic BranchD,
    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,
    output logic [2:0] ForwardAE, ForwardBE,
    output logic FlushE, StallD, StallF, ForwardAD, ForwardBD
);

    logic loadUseStall, branchStall;

    always_comb begin
        loadUseStall = MemToRegE & ((rsD == rtE) | (rtD == rtE)) & (rsD | rtD);
        branchStall = BranchD & (
                (RegWriteE & ((WriteRegE == rsD) | (WriteRegE == rtD))) |
                (MemToRegM & ((WriteRegM == rsD) | (WriteRegM == rtD)))
              );
        StallF = loadUseStall | branchStall;
        StallD = StallF;
        FlushE = StallF;

        ForwardAE = (rsE != 5'd0) ? (
                (rsE == WriteRegM) & RegWriteM ? 3'b010 :
                (rsE == WriteRegW) & RegWriteW ? 3'b001 :
                3'b000
            ) : 3'b000;

        ForwardBE = (rtE != 5'd0) ? (
                (rtE == WriteRegM) & RegWriteM ? 3'b010 :
                (rtE == WriteRegW) & RegWriteW ? 3'b001 :
                3'b000
            ) : 3'b000;
        ForwardAD = (rsD != 5'd0) & (rsD == WriteRegM) & RegWriteM;
        ForwardBD = (rtD != 5'd0) & (rtD == WriteRegM) & RegWriteM;
    end

endmodule
```

PipeDtoE)

```
module PipeDtoE(
    input logic clk, FlushE,
    input logic [31:0] RD1, RD2, SignImmD,
    input logic [4:0] RsD, RtD, RdD,
    input logic [2:0] ALUControlD,
    input logic RegWriteD, MemToRegD, MemWriteD,
    input logic ALUSrcD, RegDstD,
    output logic [31:0] RD1E, RD2E, SignImmE,
    output logic [4:0] RsE, RtE, RdE,
    output logic [2:0] ALUControlE,
    output logic RegWriteE, MemToRegE, MemWriteE,
    output logic ALUSrcE, RegDstE
);

    always_ff @(posedge clk) begin
        if (FlushE) begin
            RegWriteE <= 1'b0;
            MemToRegE <= 1'b0;
            MemWriteE <= 1'b0;
            ALUSrcE <= 1'b0;
            RegDstE <= 1'b0;
            ALUControlE <= 3'b0;
            RD1E <= 32'b0;
            RD2E <= 32'b0;
            SignImmE <= 32'b0;
            RsE <= 5'b0;
            RtE <= 5'b0;
            RdE <= 5'b0;
        end else begin
            RegWriteE <= RegWriteD;
            MemToRegE <= MemToRegD;
            MemWriteE <= MemWriteD;
            ALUSrcE <= ALUSrcD;
            RegDstE <= RegDstD;
            ALUControlE <= ALUControlD;
            RD1E <= RD1;
            RD2E <= RD2;
            SignImmE <= SignImmD;
            RsE <= RsD;
            RtE <= RtD;
            RdE <= RdD;
        end
    end

endmodule
```

PipeEtoM)

```
module pipEtoM(
    input logic clk,
    input logic [31:0] aluout_e, writedata_e,
    input logic [4:0] writereg_e,
    input logic regwrite_e, memtoreg_e, memwrite_e,
    output logic [31:0] aluout_m, writedata_m,
    output logic [4:0] writereg_m,
    output logic regwrite_m, memtoreg_m, memwrite_m
);
    always_ff @(posedge clk) begin
        writedata_m <= writedata_e;
        memwrite_m <= memwrite_e;
        aluout_m <= aluout_e;
        regwrite_m <= regwrite_e;
        writereg_m <= writereg_e;
        memtoreg_m <= memtoreg_e;
    end
endmodule
```