Bilkent University

Computer Engineering Department

CS224

Design Report

Lab 5

Section 3

Muhammet Abdullah KOÇ

ID: 21802832

Lab Date: 13.04.2021

Pre-work Date: 07.04.2021

b) **Type of hazards**

1) **Data Hazards**

Compute – use: In this type, decode stage is affected because old and wrong instruction comes to next instruction's decode stage (Former instruction's writeback stage is not started). Execute and writeback stages are affected implicitly.

Load – use: In this type, execute and memory stages are affected.

Load – store: In this type, memory stage is affected because of storing of wrong value.

2) **Control Hazards**

Branch: Because of delay, 3 instructions will be fetched wrongly.

c) **Solutions and Other Details**

1) **Data Hazards**

Compute – use: This type of hazard occurs when the calculation of former instruction has not been written to register file yet. For example, if former instruction is add $t0, $t1, $t2; and next instruction is add $t3, $t0, $t4; the new value of $t0 is not written yet.

There are several solutions to handle this problem. One of solutions is forwarding. Thus, the result of previous instruction can be forwarded to next instruction directly. Stalling also works for this hazard.

Load – use: If the previous instruction needs to read data from data memory, it needs to reach the end of memory stage. Therefore, next instruction cannot get the data.

This type of hazard cannot be handled by forwarding. Stalling pipeline until the data is ready can be a solution.

Load – store: This hazard occurs when a data will be stored after loaded with the same rt register. For example, if consecutive instructions are: lw $t0, 10($t1) and sw $t0, 0($t2), wrong value will be stored.

Stalling can be an effective solution for solving this problem. Pipeline can be stalled until the next instruction reads data.

2) **Control Hazards**

Branch: In the pipelined MIPS architecture, branch decision is made at memory stage. Therefore, it causes delay. Until branch decision is made, next instruction is at execute stage.

A solution of this hazard can be stalling the pipeline for 3 cycles. Another solution is flushing redundant instructions. Last solution is to change hardware and alter branch predictions to be made early.

Note: All of hazards can be handled by using nop's, but it will be less effective.

d) **Logic Equations**

Forwarding:

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


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

Stalling and Flushing:

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

Control Forwarding and Stalling

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




branchstall = BranchD AND RegWriteE AND
                   (WriteRegE == rsD OR WriteRegE == rtD)
          OR
          BranchD AND MemtoRegM AND
                   (WriteRegM == rsD OR WriteRegM == rtD)
StallF = StallD = FlushE = (lwstall OR branchstall)
```

e) **Test Programs**

#case 1 – no hazard (to check pipelined processor is correctly written)

addi $t0, $zero, 1	instr = 32'h20080001;
addi $t1, $zero, 2	instr = 32'h20090002;
addi $t2, $zero, 3	instr = 32'h200a0003;
or $t3, $t1, $t2	instr = 32'h012a5825;
add $t4, $t0, $t1	instr = 32'h01096020;
sub $t5, $t2, $t0	instr = 32'h01486822;
sw $t0, 10($t4)	instr = 32'had88000a;
lw $t5, 8($zero)	instr = 32'h8c0d0008;
beq $t1, $zero, 1	instr = 32'h1120fff8;
sub $t1, $t1, $t2	instr = 32'h012a4822;
and $t4, $t2, $t3	instr = 32'h014b6024;

#case 2 – compute – use hazard

addi $t0, $zero, 8	instr = 32'h20080008;
addi $t1, $zero, 9	instr = 32'h20090009;
sub $t2, $t1, $t0	instr = 32'h01285022;

#case 3 – load – use hazard

addi $t0, $zero, 5	instr = 32'h20080005;
addi $t1, $zero, 10	instr = 32'h2009000a;
addi $t2, $zero, 15	instr = 32'h200a000f;
lw $t0, 10($t1)	instr = 32'h8d28000a;
add $t3, $t0, $t2	instr = 32'h010a5820;
sub $t4, $t2, $t0	instr = 32'h01486022;

#case 4 – load – store hazard

addi $t0, $zero, 8	instr = 32'h20080008;
addi $t1, $zero, 7	instr = 32'h20090007;
addi $t2, $zero, 6	instr = 32'h200a0006;
lw $t0, 6($t1)	instr = 32'h8d280006;
sw $t0, 7($t3)	instr = 32'had680007;

#case 5 – branch hazard

```
addi $t0, $zero, 4                    instr = 32'h20080004;
addi $t1, $zero, 4                    instr = 32'h20090004;
addi $t2, $zero, 3                    instr = 32'h200a0003;
addi $t3, $zero, 5                    instr = 32'h200b0005;
beq $t0, $t1, Continue                instr = 32'h1109ffff;
addi $t5, $zero, 9                    instr = 32'h200d0009;
add $t0, $t1, $t2                     instr = 32'h012a4020;
or $t3, $t2, $t1                      instr = 32'h01495825;
addi $t6, $zero, 7                    instr = 32'h200e0007;
Continue:     and $t0, $t0, $t1       instr = 32'h01094024;
```