İhsan Doğramacı Bilkent Üniversitesi

Computer Science

CS224 COMPUTER ORGANIZATION

Design Report

Lab 5

Section 3

İsmet Alp Eren

21703786

Sunday

Part-1

b-

Data Hazards: Occur when result of the instruction not written back to the register file and following instruction needs that result.

Computing(compute-use) instructions have that hazard. Add, addi, sub, and, or. These hazards occur in write-back and decode stages.

Second type of data hazard is (load-use), when lw is used program gets the result in memory stage, but if following instruction needs that lw result then hazard will occur because the result of lw not written the register file yet.

Third type of data hazard for sw (load-store or compute-store), when compute instruction followed by sw, data hazard occurs because before compute instruction write back the data to register file, sw will be need it.

Control Hazards: Occurs when next instruction not decided yet.

Since this is pipeline, instructions fetch before previous instruction not ended. And if branch occurs, then instruction already fetched, between branch instruction and instruction that branching will be, should be flushed because we don't need them. This is control hazard. (branching) It occurs in 4th stage which is memory stage

The case same for jump instruction. (jumping) I am not sure in which stage hazard occurs because we didn't learn that and not written in the book.

c-

Data Hazards: For compute-use and compute-store hazards we can use forwarding. What forwarding does is, since compute concludes in the 3rd stage and for following instruction, we need that result in the 3rd stage also, we forwarding result to directly the one of the alu's source registers. (forwarding occurs 4th stage to 3rd stage for following instruction but occurs 5th stage to 3rd stage for second following instruction)

For load-use and load-store hazard which is occurs when lw result is needed in following 2 instructions. Lw's result determining in the 4th stage and it passes the 5th stage. The result of the lw is needed in 3rd stage for next following 2 instruction. The solution is, we need to stall the program 1 clock cycle following 2 instruction and then forward data for first following instruction.

Control Hazards: Solution is simple, just flush the instructions between branch instruction and instruction that branching will be. Yet this solution causes time wasting. To prevent time wasting we use branch prediction.

d- forwarding for rs register

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

forwarding for rt register

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

stall for pipef (stallF), piped (stallD), flushE for branching

```
branchstall = BranchD AND RegWriteE AND (WriteRegE == rsD OR WriteRegE == rtD)
OR BranchD AND MemToRegM AND (WriteRegM == rsD OR WriteRegM == rtD);

lwstall = ((rsD==rtE) OR (rtD==rtE)) AND MemtoRegE
stallF = stallD = flushE = ( lwstall OR brachstall)
```

e-

No hazard:

```
addi v0,zero,1
addi v1,zero,2
addi a0,zero,3
addi a1,zero,4
sub  s0,v0,v1
```

```
8'h00: instr = 32'h20020001;
8'h04: instr = 32'h20030002;
8'h08: instr = 32'h20040003;
8'h0c: instr = 32'h20050004;
8'h10: instr = 32'h00438022;
```

Data Forwarding:

```
addi v0,zero,1
addi v1,zero,2
addi s0,v0,2
addi s1,v1,2
sub  s2,s0,v1
```

```
8'h00: instr = 32'h20020001;
8'h04: instr = 32'h20030002;
8'h08: instr = 32'h20500002;
8'h0c: instr = 32'h20710002;
8'h10: instr = 32'h02039022;
```

Flushing:

```
addi v0,zero,1
addi v1,zero,1
addi s0,zero,2
addi s1,zero,3
beq  v0,v1,8
addi s2,zero,4
addi s3,zero,5
addi s4,zero,6
addi s5,zero,7
```

```
8'h00: instr = 32'h20020001;
8'h04: instr = 32'h20030001;
8'h08: instr = 32'h20100002;
8'h0c: instr = 32'h20110003;
8'h10: instr = 32'h10430003;
8'h14: instr = 32'h20120004;
8'h18: instr = 32'h20130005;
8'h1c: instr = 32'h20140006;
8'h20: instr = 32'h20150007;
```

Stalling:

addi v0,zero,1	8'h00: instr = 32'h20020001;
addi v1,zero,4	8'h04: instr = 32'h20030004;
addi s0,zero,2	8'h08: instr = 32'h20100002;
addi s1,zero,3	8'h0c: instr = 32'h20110003;
sw v0,0(v1)	8'h10: instr = 32'hac620000;
addi s2,zero,4	8'h14: instr = 32'h20120004;
addi s3,zero,5	8'h18: instr = 32'h20130005;
lw s4,0(s1)	8'h1c: instr = 32'h8e340000;
addi s5,s4,6	8'h20: instr = 32'h22950006;