Bilkent University

Computer Science Department

CS224

Design Report

Lab 5

Section 3

Murat Furkan Uğurlu

Id: 21802062

April 13, 2021

PART I

B) The list of all hazards that can occur in this pipeline is the following:

Data Hazards:

- **Name:** Compute Use
  The pipeline stages that are affected: Decode stage is affected.

- **Name:** Load Use
  The pipeline stages that are affected: Execute stage and memory stage are affected.

- **Name:** Load Store
  The pipeline stages that are affected: Memory stage is affected.

Control Hazards:

- **Name:** Branch
  The pipeline stages that are affected: Memory stage is affected.

C) The solution and explanation of each hazard is the following:

Compute Use

Solution: The data can be forwarded to next instruction's execute stage so that the next instruction can use it. Stalling might be another way to solve the problem.

Explanation: This type of a hazard occurs when the next instruction tries to access the register which is being processed by the current instruction that makes a computation. Therefore, the previous value of the register is used by the next instruction and this causes a problem because a wrong result will be obtained. For instance, "add" instruction can be an example for this. If the next instruction tries to reach the register which is destination register of the current instruction, then there occurs a problem.

Load Use

Solution: Stalling until the data becomes accessible is a way to solve the problem.

Explanation: This type of a hazard occurs when the next instruction tries to access the register which is being processed by the current instruction that makes a load operation. In the execute stage of the next instruction, thus, the wrong data is retrieved since the new data has still not been written to memory in the current instruction.

Load Store

Solution: Stalling until the decode stage of the next instruction is a way to solve the problem.

Explanation: This type of a hazard occurs if there is a load operation followed by a store operation which uses the same register. This causes a problem since the value is not loaded yet while store instruction is executing.

Branch

Solution: Stalling for 3 cycles is a way to solve this problem. Another way is to add a hardware component to control the branch without waiting for the next stages. Flushing the stages is also another way to solve this hazard.

Explanation: This type of a hazard occurs when there is a branch instruction. Since it is done in memory stage, there occurs a delay. At the same time, the next instruction is fetched, so the delay causes a problem.

D) Logic equations for each signal output by the hazard unit

Logic equations for Forwarding

```
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
```

Logic equations for Stalling

```
lwstall = ( ( rsD == rtE ) OR ( rtD == rtE ) ) AND MemtoRegE

StallF = StallD = FlushE = lwstall
```

Control Forwarding & Stalling Logic

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

ForwardBD = ( rtD != 0 ) AND ( rtD == WriteRegM ) AND RegWrite


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 in MIPS that will show whether the pipelined processor is working or not

Test program with no hazard

```
addi    $t0, $zero, 19
addi    $t1, $zero, 3
addi    $t2, $zero, 7
addi    $t3, $zero, 12
addi    $t4, $zero, 3
sw      $t1, 0($s3)
add     $t5, $t0, $t1
sub     $s0, $t5, $t2
and     $t6, $t1, $t4
slt     $s1, $t1, $t2
slt     $s2, $t3, $t2
lw      $s4, 0($s3)
beq     $t1, $t4, 2
```

Machine Code

```
8'h00: instr = 32'h20080013;
8'h04: instr = 32'h20090003;
8'h08: instr = 32'h200a0007;
8'h0c: instr = 32'h200b000c;
8'h10: instr = 32'h200c0003;
8'h14: instr = 32'hae690000;
8'h18: instr = 32'h01096820;
8'h1c: instr = 32'h01aa8022;
8'h20: instr = 32'h012c7024;
8'h24: instr = 32'h012a882a;
8'h28: instr = 32'h016a902a;
8'h2c: instr = 32'h8e740000;
8'h30: instr = 32'h112cfff5;
```

Test program with Compute Use hazard

```
addi    $t0, $zero, 5
addi    $t1, $zero, 6
add     $t2, $t0, $t1
```

Machine Code

```
8'h00: instr = 32'h20080005;
8'h04: instr = 32'h20090006;
8'h08: instr = 32'h01095020;
```

Test program with Load Use hazard

```
addi    $t0, $zero, 5
addi    $t2, $zero, 4
addi    $t3, $zero, 1
sw      $t0, 0($s0)
addi    $t4, $zero, 3
add     $t0, $t0, $t1
addi    $t5, $zero, 7
lw      $t1, 0($s0)
slt     $t6, $t1, $t2
```

Machine Code

```
8'h00: instr = 32'h20080005;
8'h04: instr = 32'h200a0004;
8'h08: instr = 32'h200b0001;
8'h0c: instr = 32'hae080000;
8'h10: instr = 32'h200c0003;
8'h14: instr = 32'h01094020;
8'h18: instr = 32'h200d0007;
8'h1c: instr = 32'h8e090000;
8'h20: instr = 32'h012a702a;
```

Test program with Load Store hazard

```
addi    $t0, $zero, 5
addi    $t1, $zero, 4
addi    $t2, $zero, 1
addi    $a0, $zero, 7
addi    $t3, $zero, 3
addi    $t4, $zero, 7
lw      $t6, 0($t0)
sw      $t6, 0($t0)
```

Machine Code

```
8'h00: instr = 32'h20080005;
8'h04: instr = 32'h20090004;
8'h08: instr = 32'h200a0001;
8'h0c: instr = 32'h20040007;
8'h10: instr = 32'h200b0003;
8'h14: instr = 32'h200c0007;
8'h18: instr = 32'h8d0e0000;
8'h1c: instr = 32'had0e0000;
```

Test program with Branch hazard

```
addi    $t0, $zero, 0
addi    $t1, $zero, 4
addi    $t2, $zero, 1
beq     $t0, $zero, 1
addi    $t3, $zero, 5
addi    $t4, $t0, 12
addi    $t5, $zero, 1
slt     $t6, $zero, $t1
```

Machine Code

```
8'h00: instr = 32'h20080000;
8'h04: instr = 32'h20090004;
8'h08: instr = 32'h200a0001;
8'h0c: instr = 32'h1100fffd;
8'h10: instr = 32'h200b0005;
8'h14: instr = 32'h210c000c;
8'h18: instr = 32'h200d0001;
8'h1c: instr = 32'h0009702a;
```