Computer Science

Computer Organization - CS 224

Design Report

Lab - 5

Section – 1

Erdem Ege Eroğlu

21601636

21 DEC 2020

b) The list of hazards with their types and pipeline stages that are affected.

Data Hazards

- Compute-use Hazards

  The pipeline stages that are affected: Decode stage is affected, since fetch stage come up with wrong data. Subsequently, execute and writeback stages work wrongly because they receive data from decode stage.

- Load-use Hazards

  The pipeline stages that are affected: Execute and memory stages are affected because of 2 cycle latency.

- Load-store Hazards

  The pipeline stages that are affected: Memory stage is affected because register brings about wrong data.

c) Solutions and explanations (what, when, how) of hazards

- Compute-use Hazards

Solution: Data can "forward" to execute and memory stages of next instruction rather than waiting until writeback stage is over.

Explanation: In pipelined processors, each clock cycle starts new instructions however they do not finish in the same cycle. Thus, there can be several instructions processing at the same time. Those several instructions might have the same registers. Obviously, after each instruction's registers should be updated with new data. But those instruction have not been finished so they cannot be updated. That is why the next instruction starts with old and wrong data. When one of the registers that is used as rd, and again used as rs or rt before the first instruction is finished, then compute-use hazard occurs.

- Load-use Hazards

Solution: Waiting until the data become available. This solution called Stalling.

Explanation: If the instruction includes reading data from memory it cannot load because it has to wait until the end of memory stage. That is why next instruction would not be able to access that data. That sort of hazards occurs when next instruction wants to reach data that is supposed to be loaded by the previous instruction.

- Load-store Hazards

Solution: Stalling the pipeline until next instruction's decode stage.

Explanation: If one instruction tries to store data in a memory address and the next instruction tries to load data. Because it has old and wrong data. This kind of hazards occurs when load-word and store-words instructions are consecutive and they have the same "rt" register.

d) Logic equations that makes pipeline processor computes correctly.

- 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
```

- Stalling & Flushing

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

e) Test programs in mips with their hex codes

- <u>No Hazard:</u>

```
addi    $s0,    $zero,  3
addi    $s1,    $zero,  7
addi    $s2,    $zero,  0
addi    $s3,    $zero,  5
addi    $s4,    $zero,  9
add     $s2,    $s1,    $s0
and     $s2,    $s1,    $s0
or      $s2,    $s1,    $s0
slt     $s2,    $s1,    $s0
sw      $s1,    4($s0)
lw      $s0,    0($s1)
```

```
8'h04: instr = 32'h20100003;
8'h08: instr = 32'h20110007;
8'h0c: instr = 32'h20120000;
8'h10: instr = 32'h20130005;
8'h14: instr = 32'h20140009;
8'h18: instr = 32'h02309020;
8'h1c: instr = 32'h02309024;
8'h20: instr = 32'h02309025;
8'h24: instr = 32'h0230902a;
8'h28: instr = 32'hae110004;
8'h2c: instr = 32'h8e300000;
```

- Compute-use Hazard:

```
addi    $s0,    $zero,  3
addi    $1,     $zero,  7
add     $s2,    $s0,    $s1
```

```
8'h04: instr = 32'h20100003;
8'h08: instr = 32'h20110007;
8'h0c: instr = 32'h02119020;
```

- Load-use Hazard:

```
addi     $s0,   $zero,  3
addi    $s1,    $zero,  7
sw      $s0,    0($s1)
lw      $s1,    1($s0)
add     $s2,    $s1,    $s0
```

```
8'h04: instr = 32'h20100003;
8'h08: instr = 32'h20110007;
8'h0c: instr = 32'hae300000;
8'h10: instr = 32'h8e110001;
8'h14: instr = 32'h02309020;
```

- Load-store Hazard:

```
addi     $s0,   $zero,  3
addi    $s1,    $zero,  7
sw      $s0,    0($s1)
lw      $s1,    0($s1)
```

```
8'h04: instr = 32'h20100003;
8'h08: instr = 32'h20110007;
8'h0c: instr = 32'hae300000;
8'h10: instr = 32'h8e310000;
```