Preliminary Design Report

Lab 5

Section 1

Oğuz Kaan İmamoğlu

21702233

04.12.2019

b) The list of all hazards that can occur in this pipeline:

Data Hazards:

Load Store Hazard

Why: If a data is wanted to be stored into memory just after it was loaded from memory. Because we have to wait until writeback stage in order to load a data. If we try to use this data directly, the previous data will be used.

Affected Stages: Memory stage of the next instruction will be affected because the wrong data will be stored in the memory stage.

Load Use Hazard

Why: Instructions that need to read data from memory cannot read that data until the memory stage.

Affected Stages: Execute stage will be affected, if there is a store command memory stage will be affected also.

Compute Use Hazard

Why: After computation is completed, computed data will be return back to register file in writeback stage, which is a late stage for using new data in the next instructions. So, it causes usage of the old data in next instructions.

Affected Stages: Execute and writeback stage will be affected, if there is a store command memory stage will be affected also.

Compute Branch Hazard

Why: In this pipeline, branch is earlier than default pipeline. It can cause a hazard. Sometimes we want to branch according to result of execution stage. So early branch can be a hazard for us. This situation may lead to branching depends on wrong data.

Affected Stages: Since this will affect fetch stage, it will also affect all stages of the instruction.

c)

Solutions:

Load Store

Stalling is an effective strategy to allow loading from memory done until next instruction fetches data from same register at its decode stage.

Load Use

Forwarding cannot solve this problem because computation is already done before the new data is forwarded. So, we must stall the pipeline until the new data is available.

Compute Use

Instead of waiting for writeback stage, data must be forwarded to execution stage of next instruction, just after it is computed by alu.

Compute Branch Hazard

Stalling is the most convenient solution for this problem. We must stall the register before they enter the branching until the new data is computed and forwarded to this stage.

#d) Logic 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
```

#Logic for stalling and flushing:

lwstall =

((*rsD*==*rtE*) OR (*rtD*==*rtE*)) AND *MemtoRegE*

StallF = *StallD* = *FlushE* = *lwstall*

#e) Test Programs:

#Program 1 (No Hazard)

```
addi $t0, $zero, 7 ## 0x20080007
addi $t1, $zero, 5 ## 0x20090005
addi $t2, $zero, 0 ## 0x200A0000
addi $t3, $t0, 15 ## 0x210B0015

add $t2, $t0, $t1 ## 0x01095020

or $t2, $t0, $t1 ## 0x01095025

and $t2, $t0, $t1 ## 0x01095024

sub $t2, $t0, $t1 ## 0x01095022

slt $t2, $t0, $t1 ## 0x0109502A

sw $t0 4($t1) ## 0xAD280004

lw $t1 0($t0)  ## 0x8D090000
beq $t0 $zero 1  ## 0x11000001
```

<u>#Program 2 (Compute Use Hazard)</u>

```
addi $t2 $zero 7  ## 0x200A0007

addi $t1 $t2 5  ##  0x21490005

add $t3 $t1 $t2 ## 0x012A5820
```

<u>#Program 3 (Load Use Hazard)</u>

```
addi $t0, $zero, 7 ## 0x20080007

addi $t1, $zero, 5 ## 0x20090005

addi $t3, $zero, 15 ## 0x200B0015

sw $t0 0($t1) ## 0xAD280000

lw $t1 1($t0) ## 0x8D090001

add $t0 $t1 $t3 ## 0x012B4020
```

<u>#Program 4 (Load Store Hazard)</u>

```
addi $t0, $zero, 7 ## 0x20080007

addi $t1, $zero, 5 ## 0x20090005

addi $t2, $zero, 15 ## 0x200A0015

addi $t3, $zero, 25 ## 0x200B0025

sw $t0 0($t1) ## 0xAD280000

lw $t2 0($t1)  ## 0x8D2A0000

sw $t2 0($t3) ## 0xAD6A0000
```