Bilkent University

Department of Computer Science

CS 224

Design Report

Lab 5

Section 2

Kutay Tire

22001787

Date of the lab: 13th of April, Wednesday, 13:30 – 17:20

b.)

Specific Name	Type	Affected Stages
Compute-Use	Data Hazard	Decode Stage, Execute Stage, subsequently Memory and WriteBack Stages as wrong values are fetched.
Load-Store	Data Hazard	Memory and Execute Stages, because wrong data will be stored.
Load-Use	Data Hazard	Execute and Decode Stages. Also, if a sw instruction comes than Memory is affected.
Branch	Control Hazard	Unnecessary 3 instructions are fetched. This can be reduced to one with earlier branch decision, but RAW hazards must be checked if then.

c.)

1. **Compute-Use Hazard**

Solution: Data can be forwarded from Memory or WriteBack stages depending on the cycle the below instructions are. Also, the stalling can be used.

What is the Hazard: Compute-Use hazard is one of the three data hazards that can be seen in pipelines. It can be associated with RAW type hazards as a data is read from register before the WriteBack stage is completed.

When it occurs: It occurs when an R type instruction such as sub or and is executed and if the subsequent instructions use the result in destination register as a source register (rs or rt) without waiting for it to be written back.

How it occurs: This hazard occurs when a subsequent instruction tries to use the destination register during decode stage without waiting for it to be written back. This would cause a hazard because the result of a computation is not immediately written back to the destination register. It must go through WriteBack stage. So, if the subsequent instructions try to use this register as a source before the WriteBack, it will cause a compute-use hazard.

2. Load-Store Hazard

Solution: Stalling the pipeline until decode stage is the solution as the load-word instruction will have time to finish.

What is the Hazard: It is again one of the three control hazards that happens when a data from memory is required to be stored right after the load.

When it occurs: It happens when a store-word instruction comes right after load-word instruction.

How it occurs: As the load-word instruction needs time to be completed, the store-word instruction may use the wrong data value to store in the memory.

3. Load-Use Hazard

Solution: Similar to load-store, pipeline needs to be stalled until data reached.

What is the Hazard: It is the last type of data hazard occurs when the data from load-word instruction is immediately required to be used.

When it occurs: It happens when a subsequent instruction uses the value in the source registers rs or rt previously updated by the load-word instruction.

How it occurs: As the load-word instruction needs time to be completed, the subsequent instruction needs to wait until memory stage is completed to access the data from the memory. As a result, the following instruction cannot reach the data of the destination register at execute stage.

4. Branch Hazard

Solution: There are couple of solutions to fix the branch hazard. First of all, the pipeline can be stalled for three cycles, but it is not practical. Another approach is to predict whether branch will occur or not and calculate the BTA. Then, when branch instruction comes, the other instructions can be flushed. This

also requires three instructions to be flushed and it can be reduced to one by using early branch decision.

What is the Hazard: It is the only control hazard. As the decision is made in memory stage with the calculation of PCSrcM, the processor does not know which instruction to fetch.

When it occurs: It happens when a branch instruction is present and the branch decision is made regularly in the memory stage with PCSrcM.

How it occurs: As the new instruction is fetched with the value of PCSrcM found in the memory stage, the processor does not know which instruction to fetch in the fetch stage that causes a branch hazard.

d.)

Logic for Forwarding Data

These are for early branch prediction.

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

ForwardBD=(rtD != 0) AND (rtD == WriteRegM) AND RegWriteM;

For rs

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
```

For rt

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

For rd (sracc)

```
if ((rdE != 0) AND (rdE == WriteRegM) AND RegWriteM) then
        ForwardCE = 10
else if ((rdE != 0) AND (rdE == WriteRegW) AND RegWriteW) then
        ForwardCE = 01
else
        ForwardCE = 00
```

Logic for Stalling and Flushing

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 branchstall

e.)

The sracc instruction cannot cause a load-store or a branch hazard because there is no value saved to the memory and there isn't any input or select signal to make the processor branch. However, since it uses rs, rt and even rd registers as sources, load-use can be caused if sracc instruction comes right after a lw instruction and compute-use hazard can be caused if the destination register of sracc is used as a source register for the following instruction. Similarly, if sracc instruction uses the data of the destination register obtained from previous instruction as source, again compute-use hazard occurs.

Solutions to these hazards are the same as listed in part c. For load-use hazard, the pipeline can be delayed. For the compute-use hazard, the pipeline can either be delayed or the data can be forwarded with the help of hazard unit. However, these solutions need a little bit updating. For the forwarding, the same equations in part d must be written for rdE. Moreover, the hazard unit needs to supply another select signal ForwardCE for the third multiplexer that

will be added to get the correct value from rd as it is also a source register in the newly modified datapath.

Test Programs

For Load-Use Hazard

```
addi $t5, $zero, 5
addi $t4, $zero, 3
addi $t3, $zero, 12
lw $t2, 0($t5)
sracc $t4, $t2, $t3
```

- This causes a load-use hazard as the sracc instruction uses $t2 as a source register, but is just recently loaded from data memory. Since sracc uses all rs, rt and rd registers as source, putting $t2 first or third still causes a hazard.

For Compute-Use Hazard

```
addi $t5, $zero, 5
addi $t4, $zero, 3

addi $t1, $zero, 3
add $t3, $t5, $t4
```

sracc $t4, $t2, $t3

- In this example, again sracc uses the register $t3 as one of the source registers. However, the write back stage for the previous instruction was not completed. So, the $t3 register contains the wrong value that causes a compute-use hazard.

No Hazards

addi $t5, $zero, 5

addi $t4, $zero, 4

addi $t2, $zero,7

addi $t1, $zero, 3

add $t3, $t5, $t4

or $s0, $t2, $t4

sw $t1, 0($t2)

sracc $t3, $t4, $t2

add $s0, $zero, $t1