Bilkent University
Department of Computer Engineering

CS 224 – Computer Organization

Design Report

Lab 05

Section 03

Arda İçöz

21901443

06/04/2021

List of all hazards in this pipelined processor

- Data hazards
    - Compute – use
    - Load – use
    - Load – store
- Control hazards
    - Branch

What are these hazards, when it happens and which pipeline stages are affected?

- Compute – Use: Read after write (RAW) hazard

```
add   $s0, $s2, $s3
and   $t0, $s0, $s1
```

  If we take these two instructions as examples, we can see that compute -use hazard happens in the second “and” instruction. “add” instruction is set to be complete in 5 cycles, so the value in register $s0 will change after 5 cycles. But, decoding stage of “and” happens in the third cycle, causing the wrong value (old value) of $s0 to be used. Thus, decode stage is being affected and also it affects the stages after them, execute and writeback in this case.

- Load – Use: Trying to use a value in memory before it is loaded into memory

```
lw    $s0, 40($0)
and   $t0, $s0, $s1
```

  Let’s have these two examples. Load word instruction “lw” completes loading the value to memory in end of cycle 4 but and instruction gets the $s0 register from register file at the beginning of cycle 4. So, “and” instruction tries to do calculations with the wrong value (or old value) of $s0 register. Thus, execute and memory stages

- Load – Store: Storing the value inside a register to our storage location, but the value inside that register is not yet loaded.

```
lw   $t3, 4($t2)
sw  $t3, 8($t1)
```

This hazard resembles the load-use hazard. Store word instruction "sw" tries to store the value inside $s3 to another register. The problem is that load word "lw" instruction isn't finished loading the value to $t3. So, wrong value (or old value) of $s3 is stored. Because the wrong value will be stored, it affects the memory stage.

- Branch: Branch decision is not yet decided but next instruction is already fetched.

In order a branch decision to be made, PCSrc needs to be calculated. But this decision happens in the memory stage of the processor, so another instruction can already be fetched while this branch instruction is still deciding. What if it decided to be branched? Then it should not have been executed the next instruction. That is where the problem comes. Thus, the memory stage will be affected by this.

How to solve these hazards?

- Compute – Use: We can use forwarding to solve this hazard. After the execution stage the value can be forwarded to just the beginning of the execution stage of the next instruction.

- Load – Use and Load – Store: Stalling can be used to solve it. Load word instruction (lw) is a *two-cycle* instruction because it finishes reading the data after the memory stage. So, the next instruction must wait for 2 cycles in order to get the correct value. That is also why forwarding

cannot be used in this hazard. We cannot forward the result of memory stage to execute stage of other instruction. Hence, pipeline must be at status-quo until the loading is completed. And it can be done with stalling.

- Branch: One way to solve branch hazard is using stalling. We can make the processor wait until PCSrc is computed in memory stage. But waiting for memory stage means waiting 3 cycles, which slows down the processor significantly.

  Other way is to use flushing. If the processor gets updated to predict whether the branch will take place or not early, it can solve this hazard with not slowing down that much (compared to stalling). If our prediction is correct, then everything should go on as how it was before. But if our prediction is wrong (branch is going to happen), we have to cancel the instructions which are after our actual instruction by flushing. Flushing is a better choice than stalling and if the prediction correctness becomes more successful, it becomes a very good option.

Logic equations

- 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

- Stalling and Flushing

lwstall = ((rsD == rtE) OR (rtD == rtE)) AND MemToRegE

StallF = StallD = FlushE = lwstall

MIPS Test Programs

(compute – use hazard)

MIPS	Machine code (hex)
add $s2, $s1, $s0	0x02309020
and $t1, $s2, $s3	0x02534824
or $t2, $s4, $s2	0x02925025
sub $t3, $s2, $s5	0x02555822

(load – use hazard)

MIPS	Machine code (hex)
lw $s0, 16($zero)	0x8c100010
and $t0, $s0, $s1	0x02114024

or $t1, $s4, $s0	0x02904825
sub $t2, $s0, $s5	0x02155022

(load – store hazard)

MIPS	Machine code (hex)
lw $t0, 8($t1)	0x8d280008
sw $t2, 0($t0)	0xad0a0000
addi $s0, $s0, 31	0x2210001f
addi $s1, $s1, 18	0x22310012
sub $s0, $s0, $s1	0x02118022

(branch hazard)

MIPS	Machine code (hex)
beq $s0, $s0, 2	0x12100001
addi $t0, $t0, 1	0x21080001
addi $t1, $t1, 2	0x21290002
addi $t2, $t2, 3	0x214a0003
addi $t3, $t3, 4	0x216b0004
addi $t4, $t4, 5	0x218c0005

(no hazard)

MIPS	Machine code (hex)
addi $t0, $t0, 8	0x21080008
addi $t1, $t1, 3	0x21290003
sub $t2, $t0, $t1	0x01095022
and $t3, $t1, $t0	0x01285824
subi $t0, $t0, 1	0x2108ffff
sub $t4, $t0, $t0	0x01086022
or $t5, $t0, $t4	0x010c6825