Bilkent University
Computer Science
CS224 - Computer Organization

Design Report
Lab # 5
Section 6

Osama Tanveer
21801147

15 May 2020

Question b

<u>List of Possible Hazards that might occur in this Pipeline</u>:

Data Hazards:

1. **Type:** Compute-use
   Affected Pipeline Stages:
   The affected pipeline stage is the decode stage, because here the wrong data will be fetched from the register since the updates register value is not written yet. The Execute and Writeback stage perform operations on the data that has not been updated yet, resulting in wrong values.

2. **Type:** Load-use
   Affected Pipeline Stages:
   Due to two-cycle latency instructions and if memory is written, Execute and Memory stages will be affected.

3. **Type:** Load-store
   Affected Pipeline Stages:
   The Memory stage of the instruction storing data into memory will be affected because the wrong data is stored into memory from the register.

Control Hazards:

1. **Type:** Branch
   Affected Pipeline Stages:
   Due to branch decisions being made in the Memory stage, 3 instructions have been fetched and if branch is taken, these instructions have been computed unnecessarily.

Question c

Data Hazards:

1. **Type:** Compute-use
   What this Hazard is and How this Hazard occurs:
   After the execute stage of an instruction, the newly computed data is not written in the destination register until the end of the writeback stage. This might result in a subsequent instruction reading data from a register that was written by a previous instruction, in the same register. The value to be used by the next instruction has not been written into the destination register for it to be read.

   When this Hazard occurs:

This hazard occurs when an instruction writes a register in the Writeback stage, but a subsequent instruction accesses the data in the same register before the Writeback stage. For example, in case of the add instruction, the rd register is written in the Writeback stage. Now suppose that the previous instruction was an or instruction that has written into the same register the add instruction is supposed to write into. Lastly, suppose that the instruction following the or instruction is a sub instruction. The sub instruction will read the data written by the or instruction, instead of accessing the data that was supposed to be written by the add instruction. Hence, rs or rt registers of the subsequent instruction access the wrong data if they access the rd register.

Solution:
Data can be forwarded from the Execute or Writeback stage of the pipelines, instead of waiting for the Writeback stage to be over. Stalling the pipeline can also be used.

2. **Type:** Load-use
   Why and How this Hazard occurs:
   The instructions that load data from the memory cannot load data from the memory until the end of the memory stage. The subsequent instruction cannot read data from the destination register of this instruction because in the Execute stage of the subsequent instruction, this data has not been written in the register.

   When this Hazard occurs:
   This occurs when an instruction accesses data in memory and places the data in a register, but the subsequent instruction accesses this register and gets the wrong data value.

   Solution:
   Stalling the pipeline until data is available is a viable option.

3. **Type:** Load-store
   Why and How this Hazard occurs:
   This hazard occurs when the data loaded from the memory is immediately required to be stored in the memory.

   When this Hazard occurs:
   Subsequent lw and sw instructions can cause this hazard if they use the same rt register.

   Solution:
   Stalling the pipeline until the Decode stage of the subsequent instruction could provide a solution.

Control Hazards:

1. **Type:** Branch

   Why and How this Hazard occurs:

   Whenever a branch decision is supposed to be made, it is made in the Memory stage of the pipeline. This results in the next instructions being fetched. A delay is caused due to the decision of the branch being made in the Memory stage.

   When this Hazard occurs:

   This hazard occurs when a branch instruction is loaded and the branch is taken according to the branch decision at the end of Memory stage.

   Solution:

   This hazard can be fixed by make the branch decision earlier in the pipeline by using additional hardware such as to reduce the number of instructions that unnecessarily execute, hence by reducing the branch misprediction penalty. Unnecessarily executed instructions need to be flushed if the branch is taken. The other solution is to stall the pipeline for 3 clock cycles whenever a branch instruction is encountered.

Part d

Hazard Unit Logic for Forwarding Data

```
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
```

Hazard Unit Logic for Stalling & Flushing

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

Part E

No Hazard

```
addi $t0, $zero, 0x0010
addi $t1, $zero, 0x0015
addi $t2, $zero, 0x0012
addi $t3, $zero, 0x0010
addi $t4, $zero, 0x0018
addi $s0, $t0, 0x0007
sw $t0, 0x0004($zero)
add $s1, $t0, $t1
or $t2, $t2, $t1
slt $s5, $t0, $zero
and $s6, $t3, $t4
sub $t1, $t1, $t0
lw $ra, 0x0004($zero)
beq $t2, $s0, 0xFFFB
```

Compute Use Hazard

```
addi $t0, $zero, 0x0015
addi $t1, $t0, 0x0010
add $t2, $t1, $t0
```

Load-Use & Load-Store Hazard

```
addi $t0, $zero, 0x0015
addi $t1, $zero, 0x0010
addi $s0, $zero, 0x0007
addi $s1, $zero, 0x0008
sw $t0, 0x0000($zero)
lw $v1, 0x0000($zero)
sub $v0, $v1, $s0
```

Branch Hazard

```
addi $t0, $zero, 0x0000
beq $t0, $zero, 0x0002
addi $t1, $zero, 0x0004
addi $t1, $t0, 0x0001
addi $t1, $t0, 0x0002
addi $t1, $zero, 0x0003
sw $t1, 0x0000($zero)
```