Bilkent University
Computer Engineering
CS-224

Design Report
Lab 5
Section 5

Suleyman Hanyyev
21901009

09/04/2021

Part 1:

B)

The list of hazards that can occur in this pipeline:

Data Hazards:

1. Compute-use or RAW: This occurs when data at decode stage is read from the register. If some instruction has to overwrite registers at Write Back stage and it has not done it before another instruction at decode stage that is trying to read the data from that register. This situation is a problem since the last instruction is reading the old value.

2. Load-use: This hazard may occur when the data loaded into register has not finished loading and another instruction is trying to access that register. Execute stage is affected.

3. Load-store: Load-store hazard occurs when the wrong register data is trying to be put into the data memory.

Control Hazards:

1. Branch: Even though early-branch prediction exists, it is still a hazard. We lose 1 clock cycle in case the prediction is wrong.

C) The solution:

1. Compute-use: Data forwarding is a possible hardware solution. Also, using nops or rearranging the code is a possible software solution. Another possible hardware solution is stalling. However, the first maybe more efficient in most cases, since in the others we are potentially losing some amount of clock cycles.

As an example, let's look at two add instructions. First instruction is writing the result into $t0 and the second is reading from $t0. In order for the second instruction to get the latest value of $t0, two nops may be used.

2. Load-use: Similarly, here we could use nops or stalling. Stalling is an effective solution, it waits till the load instruction writes the data into the register (write back stage) and then continues with the next instruction.

3. Load-store: The solution is similar to load-use. Using stalls could prevent from load-store hazards. These situations occur when an instruction tries to access the data of a register however that register has not been yet updated by the sw.

4. Branch: Stall the pipeline, Fetch and Decode stages for one clock cycle. Flush the execute stage. This is often needed if the branch predicts wrongly.

D)

```
if ((rsE!= 0) AND (rsE== WriteRegM) AND RegWriteM)
        ForwardAE= 10
else if (( rsE != 0) AND (rsE== WriteRegW) AND RegWriteW)
        ForwardAE= 01
else
        ForwardAE= 00

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

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

E) NO HAZARDS:

add $t0, $0, $0	0x00004020
sub $t1, $0, $0	0x00004822
addi $s1, $t0, -1	0x2111FFFF
addi $s2, $t1, -5	0x2132fffb
sw $s2, 0($s3)	0xAE720000
addi $s5, $0, 15	0x2015000f
add $s6, $s5, $t0	0x02A8B020
sub $s3, $s6, $s2	0x02D29822

Compute-use Hazard:

add $s0, $0, $0	0x00008020
addi $t0, $0, 34	0x20080022
sub $s1, $t0, $s0	0x01108822

Load-use Hazard:

lw $s3, 0($t0)	0x8D130000
add $s0, $s3, $0	0x02608020

Load-store

add $t3, $t1, $0	0x01205820
sw $t3, 0($s0)	0xAE0B0000