BILKENT UNIVERSITY

COMPUTER SCIENCE

CS224-Computer Organization

Design Report

ALP TUĞRUL AĞÇALI

21801799

SECTION 5

13.04.2022

PART 1.B

Data Hazards:

1-) Compute-use hazard: Compute – use hazard occurs when a register, which is output of an instruction(RF[rd] or RF[rt]) and result of a ALU, is used in RF[rt] or RF[rs] port of another instruction, one or two line after the first one.(Not three because result is ready at writeback stage (5^{th} clock cycle time) which is 3^{rd} instruction's decode stage. Because of the having RAW dependency (read after write) in 3^{rd} instruction hazard do not occur.) This hazard occurs because, the current value of the output register is written in writeback stage however, processor needs to read it in decode stage of next instruction.

```
Ex.     add $t0, $t1, $t2
        add $t3, $t0, $t2
```

2-) Load-use hazard: Load use hazard exists when a register is loaded with a word in memory using lw instruction. In lw instruction output register RF[rt] is ready at the end of WriteBack part of the datapath(5^{th} clock cycle time). Again result is not ready before the decode stage of the second instruction.

```
Ex.     lw $t0, 0($s0)
        addi $t0, $t0, 1
```

3-)Load-store hazard: Load store hazard occurs in similar situation, when after the lw instruction sw instruction exists. After lw instruction output value is updated in writeback stage however, processor needs it in decode stage.

```
Ex.     lw $t0, 0($t0)
        sw $t0, 40($t0)
```

Control Hazards:

1-)Branch hazard: Branch hazard occurs in the use of branch instructions. In this pipelined datapath whether the next address of the pc is bta or not is determined at the end of the decode stage(end of 2^{nd} clock cycle). However, new pc is needed in Fetch stage of other instruction.(start of 2^{nd} clock cycle)

```
Ex.     beq $t0, $t1, L1
        add $t0, $t1, $t2
        …
        …
        …
    L1:sub $t0, $t1, $t2
```

PART 1.C

Data Hazards:

1-)Solution of compute-use hazard: To solve compute use hazard we only need Data Forwarding. Because, the result of an first instruction is ready at the start of memory stage (4^{th} clock cycle) and processor needs it at the start of execute stage of second instruction (4^{th} clock cycle). So, we can forward it using the hazard unit only.

2-)Solution of load-use hazard: To solve load use hazard we need to stall it. Because the latest value of the output of first instruction is ready after memory stage(5^{th} clock cycle) and processor needs it at execute stage of second instruction (4^{th} clock cycle). So, we cannot forward the data.The solution of this type of hazard is stalling and flushing the execute stage with wrong data. After stall second instruction 1 clock cycle and determine the output's current data, we need to forward it to execute stage.

3-)Solution of load-store hazard: To solve load store hazard, we need data forwarding only. Because as mentioned before output of the lw instruction updated in 5^{th} clock cycle, and second instruction needs in memory stage(5^{th} clock cycle). So, we need to forward it.

Control Hazards:

1-)Solution of branch hazard: To solve the branch hazard we need to calculate whether the branch is taken or not earlier than the Memory Stage. To solve this in our datapath branch is calculated in decode stage. In addition, when the branch is calculated at the decode stage we have one clock cycle for second instruction. For this clock cycle we have to predict if branch is taken or not. If our prediction is wrong we flush the fetch stage we did.

PART 1.D

Data Hazards:

1-)

For rsE:

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

For rtE:

if ((rtE != 0) AND (rtE == WriteRegM) AND RegWriteM)

    ForwardBE = 10

else

    if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)

        ForwardBE = 01

    else

        ForwardBE = 00

2-)

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

StallF = StallD = FlushE = lwstall

//then forward

For rsE:

if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)

    ForwardBE = 01

else

    ForwardBE = 00

For rtE:

if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)

    ForwardBE = 01

else

    ForwardBE = 00

3-)

For rsE:

if ((rsE != 0) AND (rsE == WriteRegW) AND RegWriteW)

    ForwardBE = 01

else

ForwardBE = 00

For rtE:

if ((rtE != 0) AND (rtE == WriteRegW) AND RegWriteW)

ForwardBE = 01

else

ForwardBE = 00

Control Hazards:

1-)

//forward

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

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

//stall

branchstall = BranchD AND RegWriteE AND

(WriteRegE == rsD OR WriteRegE == rtD)

OR

BranchD AND MemtoRegM AND

(WriteRegM == rsD OR WriteRegM == rtD)

StallF = StallD = FlushE = (lwstall OR branchstall)

PART 1.E

Firstly, in order to implement the sracc instruction we need to have RD3 in register file. In addition, we need to have a mux in execution stage controlled by sracc input(if instruction is sracc output is rd3 else 0) and output of this mux should go to an adder with ALU output in memory stage, before ALU output go to the writeback stage or execute stage of next instruction.

First hazard that this instruction can cause is same with the compute use hazard. In this instruction RF[rd] is being updated so, if it is going to be used in a instruction one or two lines below it, it causes compute-use hazard.

Ex.
```
addi $t0, $zero, 5
addi $t1, $zero, 12
addi $t2, $zero, 2
sracc $t0, $t1, $t2
add $t3, $t0, $t1
```

Second hazard that this instruction can cause is in sracc RF[rd] is also a input of this instruction and it must have it's updated value if it was updated one or two lines before.

Ex.
```
addi $t2, $zero, 2
addi $t1, $zero, 12
addi $t0, $zero, 5
sracc $t0, $t1, $t2
```
or
```
addi $t1, $zero, 1
sw $t1, 0($s0)
lw t0, 0($s0)
sracc $t0, $t3, $t4
```