BILKENT UNIVERSITY

COMPUTER ORGANIZATION

CS 224 – Section 05

LAB 05

Preliminary Design Report

Turgut Alp Edis

21702587

06/04/2021

Part b)

Possible Hazard List in The Pipeline:

1- Compute-use Hazard: This the type of data hazard. It affects decode, execute and writeback stages of the pipeline since the data will be used in decode is taken incorrectly because previous instruction used that data as well, then decode stage is dependent to previous instruction.
2- Load-use Hazard: It is data hazard. It affects decode and memory stages of the pipeline since the data written to the memory in memory stage is tried to be used by next instruction in decode stage and it creates hazard because of clock cycle latency.
3- Load-store Hazard: It is data hazard. It affects memory stage of the pipeline because the data loaded from memory in memory stage is tried to be used for store in memory by next instruction and it creates dependency.
4- Branch Hazard: It is control hazard. It can affect fetch and decode stages since the next instruction will be computed in decode and it goes to fetch stage but if there is a wrong computation, the pipeline continues in a wrong way.

Part c)

Solution of Compute-use Hazard: To solve compute use dependency, next instruction which uses same data as current instruction can be forwarded. Also, the next instruction can be stalled until its decode stage can be in same clock cycle as current instruction.

What, when and how is Compute-use hazard occur: It happens when between the decode and writeback stages. The destination register, rd in R-Type instructions, is not yet written by current instruction and same register is needed to be used by next instruction, then the next instruction is dependent to its previous instruction. If it does not wait writeback stage, it uses wrong data and all instruction will be miscalculated.

Solution of Load-use Hazard: This hazard cannot be solved by forwarding data, then it can be solved by stalling the instruction until it gets the same point as the memory stage of its previous instruction.

What, when and how is Load-use hazard occur: It happens when before the load instruction is not completely done, does not yet complete writeback stage, the next instruction is needed to use it in decode stage. If the load is not completed, the next instruction cannot receive this data and it creates hazard.

Solution of Load-store Hazard: Since it involves load, stalling is effective way to solve this hazard. Stalling the next instruction until it gets same cycle with memory stage of its previous instruction.

What, when and how is Load-store hazard occur: It happens when the data which is currently loaded from memory is tried to save to the memory by next instruction. Sequential load and store instructions causes this hazard.

Solution of Branch Hazard: To avoid mispredictions in decode stage, the new multiplexer can be added to determine the correct time for branch.

What, when and how is Branch hazard occur: It occurs when the branch is taken in beq instruction. Even if the conditions are not met for beq, the branch is taken and skips other instructions, then it may lead to loss of instructions, data, memory.

Part d)

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

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

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

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

StallF = StallD = FlushE = lwstall

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 e)

//With no hazard

```
8'h00: instr = 32'h2010000f;
8'h04: instr = 32'h2011001a;
8'h08: instr = 32'h20120038;
8'h0c: instr = 32'h00118025;
8'h10: instr = 32'h00119024;
8'h14: instr = 32'h0230902a;
8'h18: instr = 32'h1200fff9;
8'h1c: instr = 32'h02209020;
8'h20: instr = 32'h0230902a;
8'h24: instr = 32'hae110002;
8'h28: instr = 32'h8e300000;
8'h2c: instr = 32'h20120037;
8'h30: instr = 32'h20110059;

#Pipeline with no hazard
addi $s0, $0, 15
addi $s1, $0, 26
addi $s2, $0, 56
or $s0, $0, $s1
and $s2, $0, $s1
slt $s2, $s1, $s0
beq $s0, $0, 1
add $s2, $s1, $0
slt $s2, $s1, $s0
sw $s1, 2($s0)
lw $s0, 0($s1)
addi $s2, $0, 55
addi $s1, $0, 89
```

//With compute-use Hazard

```
8'h00: instr = 32'h2010004e;
8'h04: instr = 32'h2011002d;
8'h08: instr = 32'h20120063;
8'h0c: instr = 32'h00119024;
8'h10: instr = 32'h00118025;
8'h14: instr = 32'h02309020;

#Pipeline with compute-use hazard
addi $s0, $0, 78
addi $s1, $0, 45
addi $s2, $0, 99
and $s2, $0, $s1
or $s0, $0, $s1
add $s2, $s1, $s0
```

//With load-use Hazard

```
8'h00: instr = 32'h20100043;
8'h04: instr = 32'h3c011001;
8'h08: instr = 32'h34310000;
8'h0c: instr = 32'h20120000;
8'h10: instr = 32'h20120000;
8'h14: instr = 32'h20120000;
8'h18: instr = 32'h20120000;
8'h1c: instr = 32'h20120000;
8'h20: instr = 32'h20120000;
8'h24: instr = 32'h20120000;
8'h28: instr = 32'h20120000;
8'h2c: instr = 32'h20120000;
8'h30: instr = 32'h8e300000;
8'h34: instr = 32'h22120037;

#Pipeline with load-use hazard
.text
addi $s0, $0, 67
la $s1, array
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
lw $s0, 0($s1)
addi $s2, $s0, 55
.data
array: .word 56, 78
```

//With Load-Store Hazard

```
8'h00: instr = 32'h20100043;
8'h04: instr = 32'h3c011001;
8'h08: instr = 32'h34310000;
8'h0c: instr = 32'h20120000;
8'h10: instr = 32'h20120000;
8'h14: instr = 32'h20120000;
8'h18: instr = 32'h20120000;
8'h1c: instr = 32'h20120000;
8'h20: instr = 32'h20120000;
8'h24: instr = 32'h20120000;
8'h28: instr = 32'h20120000;
8'h2c: instr = 32'h20120000;
8'h30: instr = 32'h8e300000;
8’h34: instr = 32’hae320004;
8'h38: instr = 32'h20120037;
```

```
#Pipeline with load-store hazard
.text
addi $s0, $0, 67
la $s1, array
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
addi $s2, $0, 0
lw $s0, 0($s1)
sw $s2, 4($s1)
addi $s2, $0, 55
.data
array: .word 56, 78
```