Bilkent University
Computer Science
CS224 Computer Architecture

Design Report
Lab 05
Section 01

Berk Saltuk Yılmaz
21903419

April 12, 2021

Part 1.

b) **[10 points]** The list of all hazards that can occur in this pipeline. For each hazard, give its type (data or control), its specific name (“compute-use” “load-use”, “load-store”, “branch” etc.), the pipeline stages that are affected:

Data Hazards:

1. **load-store:** This hazard occurs when a value is loaded before the store instruction is done. The register is loaded with the wrong value and this affects the memory stage of the pipeline.

2. **load-use:** This data hazard occurs when a register is being used before it is loaded from the memory. The memory and execute stages will be affected since execution will be done with wrong values and the right value will not be fetched from the memory.

3. **compute-use:** This occurs when a register used again before it is written back to the register file after computation is done. This hazard affects decode stage since wrong value for the register is fetched and also execute step because register is being used before it is written.

Control Hazards:

1. **branch:** Branch will be taken in memory stage and the branch target address will be determined in this stage. This will delay the upcoming instruction by three clock cycles.

c) **[10 points]** For each hazard, give the solution (forwarding, stalling, flushing, combination of these), and an explanation of what, when, how:

Data Hazards:

1. **Solution for load-store:** Stalling is the solution for this hazard, the pipeline can be stalled to ensure the store operation is completed.
   what, when, how: Since loading cannot be done before storing is done, pipeline must be stalled such that the storing is done, this can be achieved by not updating the program counter.

2. **Solution for load-use:** Again stalling the pipeline is a solution for this type of hazard. Pipeline should be stalled till load operation is done before using this register.
   what, when, how: If the register used in second instruction is being loaded with another value in first instruction, we cannot use that register till the loading is done since we would use the previous and wrong version of this register in such case. Thus, pipeline must be stalled to be sure that the loading operation is done

3. **Solution for compute-use:** The next instruction can be delayed by adding two nop instructions or data might be forwarded at runtime or stall is another solution at runtime by keeping the PC same.
   what, when, how: After execution of first instruction, in the second instruction the previous value of register (which is changed in execute stage of first instuction) is being used. We can use a hazard unit and an R type instruction and compare first instruction's rd register to rs register of second instruction. After that by using a 2:1 MUX we can forward new value of register which causes an hazard.

Control Hazards:

1. **Solution for branch:** Since branch causes a delay for 3 clock cycles, the pipeline might be stalled for 3 clock cycles. Another solution for this hazard to make the branch decision earlier by using additional HW. In this situation flushing is also needed.
   what, when, how: Branch will be taken in memory stage and the branch target address will be determined in this stage. This will delay the upcoming instruction by three clock cycles.

d) **[10 points]** Give the logic equations for each signal output by the hazard unit, as a function of the input signals that come to the hazard unit. This hazard unit should handle all the data and control hazards that can occur in your pipeline (listed in b) so that your pipelined processor computes correctly:

Logic Equations for Data 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
```

Logic Equations for Stalling/Flushing:

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

Logic Equations for Branch Stalling:

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)

e) **[20 points]** Write small test programs, in MIPS assembly, that will show whether the pipelined processor is working or not. Each of your test programs should be designed to catch problems, if there are any, in the execution of MIPS instructions in your pipelined machine. Write: A test program with no hazards (to verify that there are no problems with the connections in your pipeline etc.) A test program that has one type of hazard, and another, and another...

1) With No Hazard:

```
addi $s0, $zero, 1
nop
nop
addi $s1, $s0, 2
addi $s2, $zero, 3
addi $s3, $zero, 4
and $s4, $s1, $s0
nop
or $s5, $s3, $s1
lw $s4, 0($s3)
nop
sw $s5, 0($s2)
addi $s4, $s4, 15
beq $zero, $zero, 15
addi $s6, $zero, 5
```

Machine Code of This:

```
8'h00: instr = 32'h20100001;
8'h04: instr = 32'h58000000;
8'h08: instr = 32'h58000000;
8'h0c: instr = 32'h22110002;
8'h10: instr = 32'h20120003;
8'h14: instr = 32'h20130004;
8'h18: instr = 32'h0230a024;
8'h1c: instr = 32'h58000000;
8'h20: instr = 32'h0271a825;
8'h24: instr = 32'h8e740000;
8'h28: instr = 32'h58000000;
8'h2c: instr = 32'hae550000;
8'h30: instr = 32'h2294000f;
8'h34: instr = 32'h10000001;
8'h38: instr = 32'h20160005;
```

2) Load-use Hazard:

```
addi $s0, $zero, 1
addi $s1, $zero, 3
addi $s2, $zero, 5
nop
lw $s0, 0($s1)
add $s2, $s0, $s2
```

Machine Code for This:

```
8'h00: instr = 32'h20100001;
8'h04: instr = 32'h20110003;
8'h08: instr = 32'h20120005;
8'h0c: instr = 32'h58000000;
8'h10: instr = 32'h8e300000;
8'h14: instr = 32'h02129020;
```

3) Load-store Hazard:

```
addi $s0, $zero, 1
nop
nop
sw $s0, 0($s1)
lw $s2, 0($s1)
```

Machine Code for This:

```
8'h00: instr = 32'h20100001;
8'h04: instr = 32'h58000000;
8'h08: instr = 32'h58000000;
8'h0c: instr = 32'hae300000;
8'h10: instr = 32'h8e320000;
```

4) Compute-use Hazard:

```
addi $s0, $zero, 5
addi $s1, $s0, 1
```

Machine Code for This:

```
8'h00: instr = 32'h20100005;
8'h04: instr = 32'h22110001;
```

5) Branch Hazard:

```
addi $s0, $zero, 5
addi $s1, $zero, 5
beq $s0, $s1, 3
addi $s3, $zero, 0
addi $s4, $zero, 10
```

Machine Code for This:

```
8'h00: instr = 32'h20100005;
8'h04: instr = 32'h20110005;
8'h08: instr = 32'h12110000;
8'h0c: instr = 32'h20130000;
8'h10: instr = 32'h2014000a;
```