1 Hazards

One of the costs of pipelining is that it introduces three types of pipeline hazards: structural hazards, data hazards, and control hazards.

Structural Hazards

Structural hazards occur when more than one instruction needs to use the same datapath resource at the same time. There are two main causes of structural hazards:

Register File The register file is accessed both during ID, when it is read, and during WB, when it is written to. We can solve this by having separate read and write ports. To account for reads and writes to the same register, processors usually write to the register during the first half of the clock cycle, and read from it during in the second half. This is also known as double pumping.

Memory Memory is accessed for both instructions and data. Having a separate instruction memory (abbreviated IMEM) and data memory (abbreviated DMEM) solves this hazard.

Something to remember about structural hazards is that they can always be resolved by adding more hardware.

Data Hazards

Data hazards are caused by data dependencies between instructions. we will always assume that instructions are always going through the processor in order, we see data hazards when an instruction **reads** a register before a previous instruction has finished **writing** to that register.

Forwarding

Most data hazards can be resolved by forwarding, which is when the result of the EX or MEM stage is sent to the EX stage for a following instruction to use.

2.1 Look for data hazards in the code below, and figure out how forwarding could be used to solve them.

Instruction	C1	C2	C3	C4	C5	C6	C7
1. `addi t0, a0, -1`	IF	ID	EX	MEM	WB		
2. `and s2, t0, a0`		IF	ID	EX	MEM	WB	
3. `sltiu a0, t0, 5`			IF	ID	EX	MEM	WB

There are two data hazards, between instructions 1 and 2, and between instructions 1 and 3. The first could be resolved by forwarding the result of the EX stage in C3 to the beginning of the EX stage in C4, and the second could be resolved by forwarding the result of the MEM stage in C4 to the beginning of the EX stage in C5.

Stalls

3.3 Look for data hazards in the code below. One of them cannot be solved with forwarding—why? What can we do to solve this hazard?

Instruction	C1	C2	C3	C4	C5	C6	C7	C8
1. `addi s0, s0, 1`	IF	ID	EX	MEM	WB			
2. `addi t0, t0, 4`		IF	ID	EX	MEM	WB		
3. `lw t1, 0(t0)`			IF	ID	EX	MEM	WB	
4. `add t2, t1, x0`				IF	ID	EX	MEM	WB

There are two data hazards in the code. The first hazard is between instructions 2 and 3, from t0, and the second is between instructions 3 and 4, from t1. The hazard between instructions 2 and 3 can be resolved with forwarding, but the hazard between instructions 3 and 4 cannot be resolved with forwarding. This is because even with forwarding, instruction 4 needs the result of instruction 3 at the beginning of C6, and it wont be ready until the end of C6.

Extra for Experience

3.6 Given the RISC-V code above and a pipelined CPU with no forwarding, how many hazards would there be? What types are each hazard? Consider all possible hazards from all pairs of instructions.

How many stalls would there need to be in order to fix the data hazard(s)?

Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
1. `sub t1, s0, s1`	IF	ID	EX	MEM	WB				
2. `or s0, t0, t1`		IF	ID	EX	MEM	WB			
3. `sw s1, 100(s0)`			IF	ID	EX	MEM	WB		
4. `beq s0, s2, 1`				IF	ID	EX	MEM	WB	
5. `add t2, x0, x0`					IF	ID	EX	MEM	WB

There are four hazards: between instructions 1 and 2 (data hazard from t1), instructions 2 and 3 (data hazard from s0), instructions 2 and 4 (from s0), and instructions 4 and 5 (a control hazard).

Assuming that we can read and write to the RegFile on the same cycle, two stalls are needed between instructions 1 and 2, and two stalls are needed between instructions

2 and 3.

4. Consider the 5-stage **pipelined** MIPS datapath consisting of Instruction Fetch (IF), Instruction Decode (ID), Execution (EX), Memory (MEM), and Write-Back (WB).

You are given the following MIPS instruction sequence:

```
# $s0 to $s3 = 56, 30, 30, 7
# $t0 to $t4 = 7, 7, 7, 7, 7
        add $t0, $s0, $0
         and $t1, $t0,$s1 or $t2, $t0, $s2 sub $t3, $t0, $s3 srl $t4,
         $t0, 2
```

Start numbering the cycles with 1 when the add instruction enters the IF stage.

For part i. to iii. assume that the datapath is broken and there is no forwarding and no stalling.

i. What are the values of $t0 to $t4 at the end of Cycle 7?

(a) 56, 24, 62, 7, 7
(b) 56, 24, 62, 49, 7
(c) 56, 6, 31, 7, 7
(d) 56, 6, 7, 7, 7
(e) None of the above

ii. What are the values of $t0 to $t4 at the end of cycle 8?

(a) 56, 24, 62, 49, 7
(b) 56, 24, 62, 49, 14
(c) 56, 6, 31, 49, 7
(d) 56, 6, 31, 7, 7
(e) None of the above

iii. What are the values of $t0 to $t4 at the end of cycle 9?

(a) 56, 24, 62, 49, 7
(b) 56, 24, 62, 49, 14
(c) 56, 6, 31, 49, 14
(d) 56, 6, 31, 0, 7
(e) None of the above

iv. What instruction(s) is/are computing the wrong result(s) (choose the answer that includes ALL faulty instructions)?

(a) add
(b) and, or
(c) and, or, sub
(d) and, or, sub, srl
(e) add, and, or, sub, srl

v. Say we want to completely fix the problem from part iv. using forwarding. Which forwarding path(s) do we need to provide in order to execute the code sequence correctly (it is implied that multiplexers are inserted to join the forwarded signals with the original signals)?

(a) Output of ALU in the EX stage back to the input of the ALU in the EX stage.
(b) Output of ALU in the MEM stage back to the output of Register File in the ID stage.
(c) Output of ALU in the MEM stage back to the input of ALU in the EX stage.
(d) Both (a) and (b).
(e) Both (a) and (c).