jal isn't working. Any hints? — The mux that chooses between d, and imm. doesn't really matter— we don't need the ALU. You'll need a different mux to choose PC+4 instead of the ALU result.
Pipeline Hazards
Basic Definitions

What is a pipeline hazard?

An “error” that prevents an instruction from executing in a specific stage of the pipelined datapath.

Hazard don’t mean your program is broken—we just need to deal with them somehow.

What are the three types of pipeline hazards?

Data hazards—a instruction needs data from another instruction, but it isn’t available yet.

Control hazards—branch instructions have a destination, and a condition. A control hazard happens when we run a branch instruction, but do not know the outcome of its condition.

Structural hazards—multiple instructions need the same part of the datapath.
Dealing with Data Hazards

What are three strategies for dealing with data hazards?

1. Stalling → add nops or bubbles
2. Static Scheduling → reorder instructions to avoid hazards
3. Forwarding → peek ahead in the pipeline to get values we need before they’ve been written back

What are advantages and disadvantages of each?

Stalling:  - more instructions  + simple

Static Scheduling:  + no extra instructions  - can’t solve all hazards

Forwarding:  + no impact on programmer  - complicates datapath
Implementation Details
(briefly)
FIGURE 4.56  The datapath modified to resolve hazards via forwarding. Compared with the datapath in Figure 4.51, the additions are the multiplexors to the inputs to the ALU. This figure is a more stylized drawing, however, leaving out details from the full datapath, such as the branch hardware and the sign extension hardware.
Practice
Consider This Program:

A  addi $s0, $s0, 1
B  addi $s0, $s0, 1
C  addi $t4, $t4, 1
D  addi $t4, $t4, 1

How long will this program take to run if we avoid data hazards by...

(a) static scheduling (no forwarding)  
(b) stalling (no forwarding)  
(c) forwarding and stalling

Order: A C nop B D

9 cycles to finish

A nop nop B C nop nop D
12 cycles

8 cycles
Consider This Program:

\[\text{A lw } \text{t0}, 0(a0)\]
\[\text{B lw } \text{t1}, 4(a0)\]
\[\text{C addi } \text{t0}, \text{t0}, 1\]
\[\text{D add v0, t0, t1}\]

How long will this program take to run if we avoid data hazards by...

- (a) static scheduling (no forwarding) \(\text{11 cycles}\)
- (b) stalling (no forwarding) \(\text{same!}\)
- (c) forwarding and stalling \(\text{8 cycles}\)
Consider This Program:

```
lw $t0, 0($a0)
sll $a1, $a1, 2
add $a0, $t0, $a1
lw $t1, 0($a0)
sub $v0, $t1, $t0
```

How long will this program take to run if we avoid data hazards by...

(a) statically scheduling instructions  
(b) stalling  
(c) forwarding and stalling
Consider This Program:

A  lw  $t0,  0($a0)
B  addi $t0,  $t0,  1
C  lw  $t1,  0($a1)
D  addi $t1,  $t1,  1
E  and  $t2,  $t0,  $t1

How long will this program take to run if we avoid data hazards by...

(a) statically scheduling instructions
(b) stalling
(c) forwarding and stalling
An Adversarial Input

We've seen some cases where pipeline stalls are the only option for avoiding a data hazard. While we usually think about making programs run faster, it's useful to think about a worst-case input.

Write a five-instruction MIPS assembly program that runs as slowly as possible.
You should maximize the number of pipeline stalls, even with forwarding.
Allowing instruction reordering should not improve your program's performance.