Pipeline Data Hazards

CSC 211 – December 2, 2020

Datapath Lab Q&A

jal isn't working. Any hints? – The mux that chooses between rd, 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.

Hazards 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 - an 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 written back.

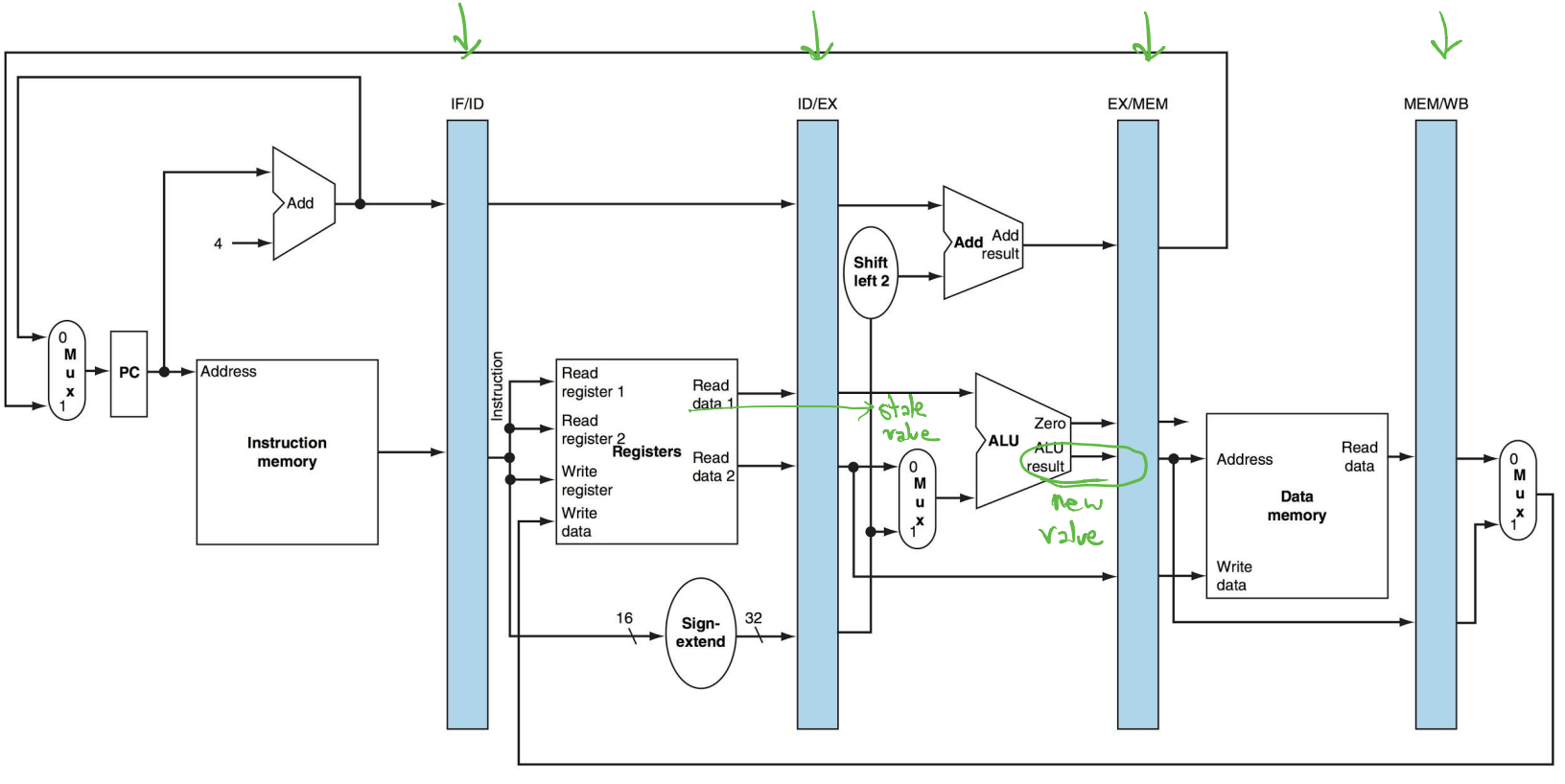
What are advantages and disadvantages of each?

Stalling: − more instructions + simple

Static Scheduling: + no extra instructions − can't solve all hazards

Forwarding: + no impact on programmer − complicates datapath

Implementation Details
(briefly)


IF/ID
ID/EX
EX/MEM
MEM/WB
Add
4
M
u
x
0
1
PC
Address
Instruction memory
Instruction
Read register 1
Read register 2
Registers
Write register
Write data
Read data 1
Read data 2
Shift left 2
Add
Add result
ALU
Zero
ALU result
stale value
new value
16
Sign-extend
32
Address
Read data
Data memory
Write data

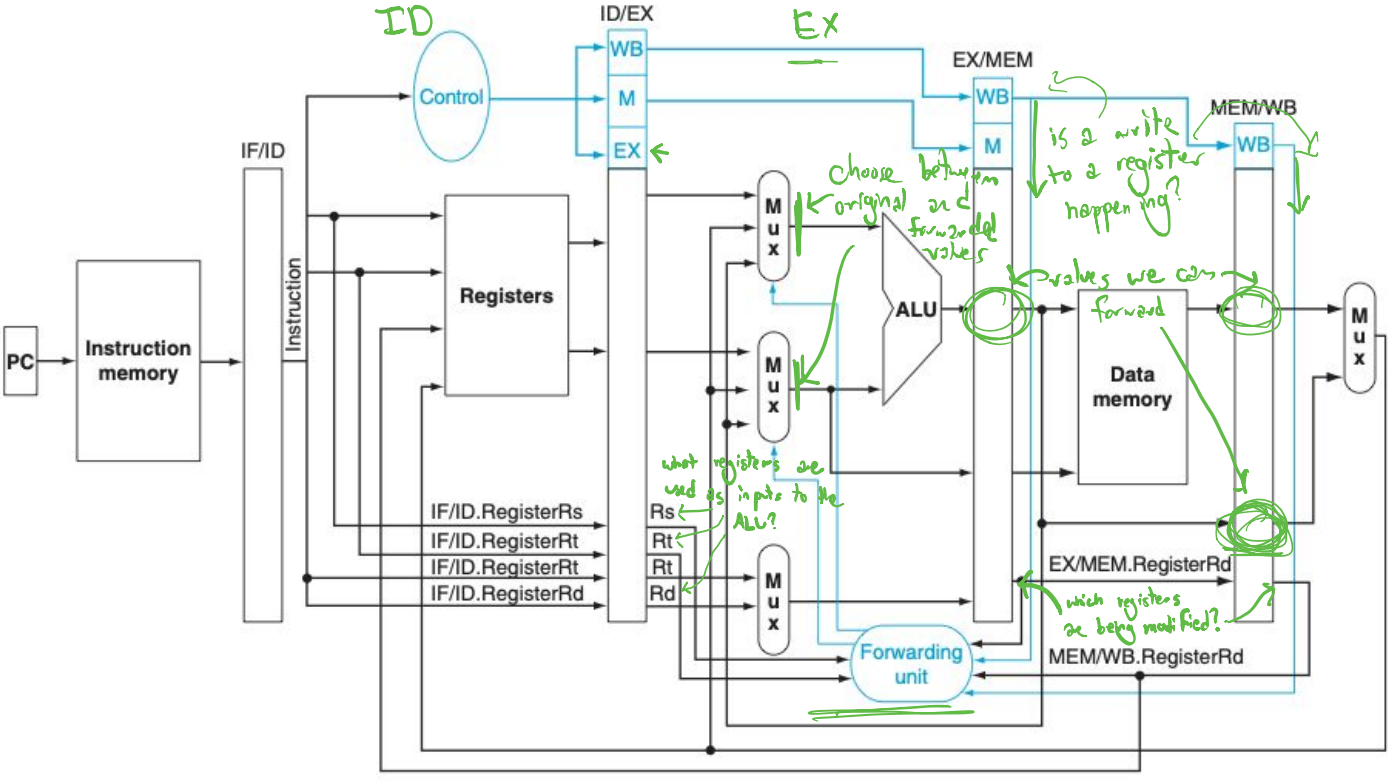
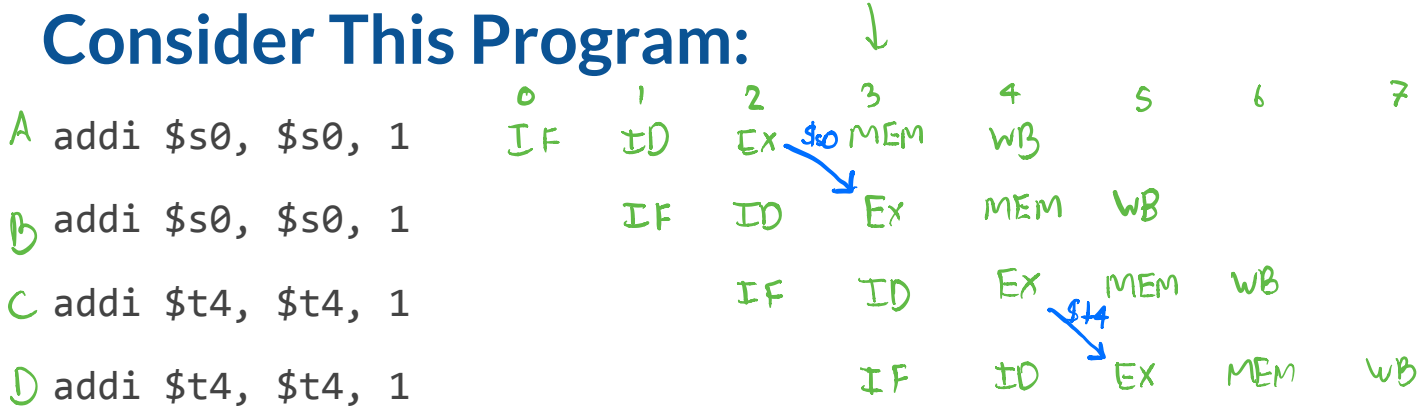


FIGURE 4.56 The datapath modified to resolve hazards via forwarding. Compared with the datapath in Figure 4.51, the additions are the multiplexors to the inputs to the ALU. This figure is a more stylized drawing, however, leaving out details from the full datapath, such as the branch hardware and the sign extension hardware.

Practice

Consider This Program:



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

(a) static scheduling (no forwarding) order: A C nop B D 9 cycles to finish

(b) stalling (no forwarding) A nop nop B C nop nop D 12 cycles

(c) forwarding and stalling 8 cycles

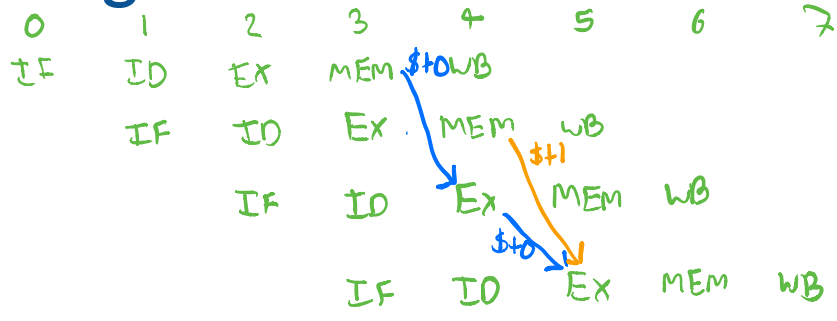
Consider This Program:

A `lw $t0, 0($a0)`

B `lw $t1, 4($a0)`

C `addi $t0, $t0, 1`

D `add $v0, $t0, $t1`



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

(a) static scheduling (no forwarding) A B nop C nop nop D 11 cycles

(b) stalling (no forwarding) same!

(c) forwarding and stalling 8 cycles

skipped

Consider This Program:

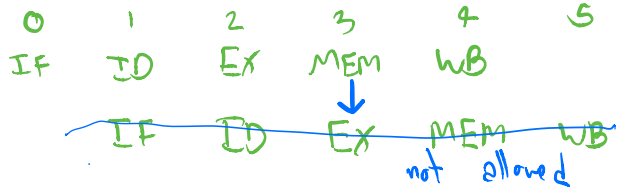
```
lw $t0, 0($a0)

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 A C nop B D nop nop E 12 cycles

(b) stalling A nop nop B C nop nop D nop nop E 15 cycles

(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.