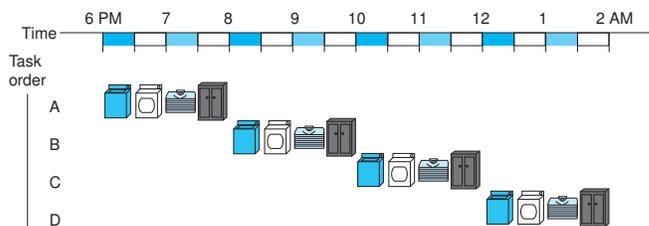

**IC220 Set #19:
Laundry, Co-dependency, and other Hazards
of Modern (Architecture) Life**

Return to Chapter 4

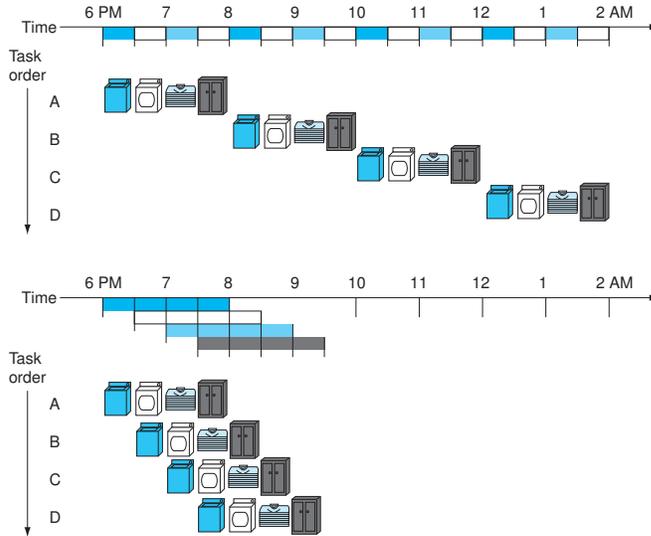
1

Midnight Laundry



2

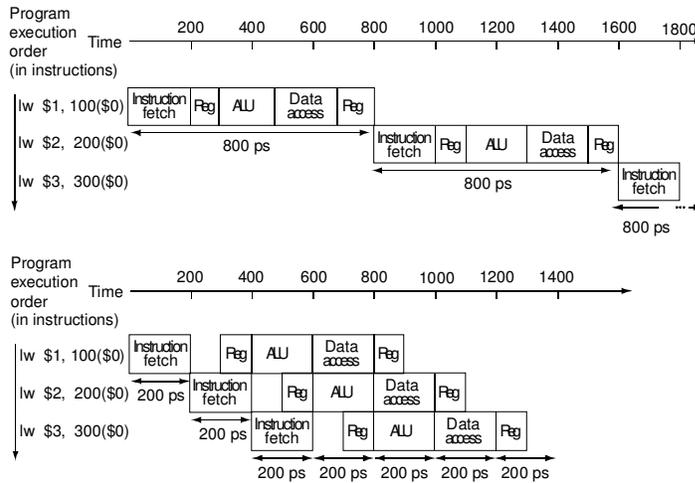
Smarty Laundry



3

Pipelining

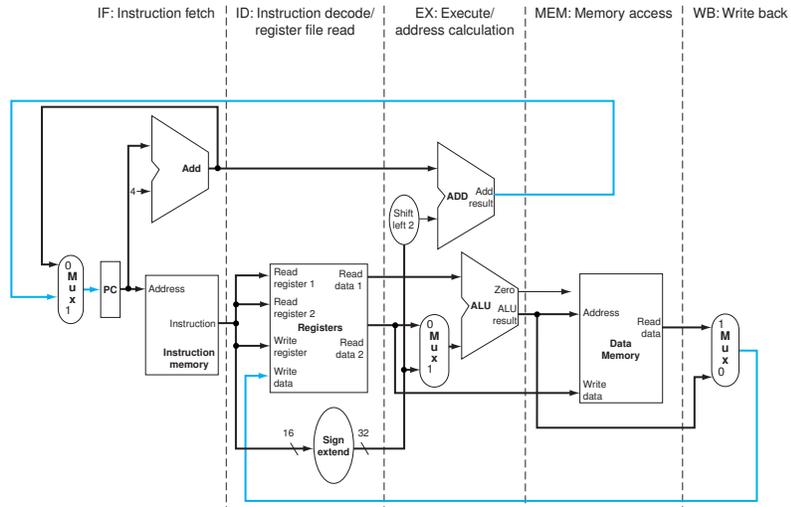
- Improve performance by increasing instruction throughput



Ideal speedup is number of stages in the pipeline. Do we achieve this?

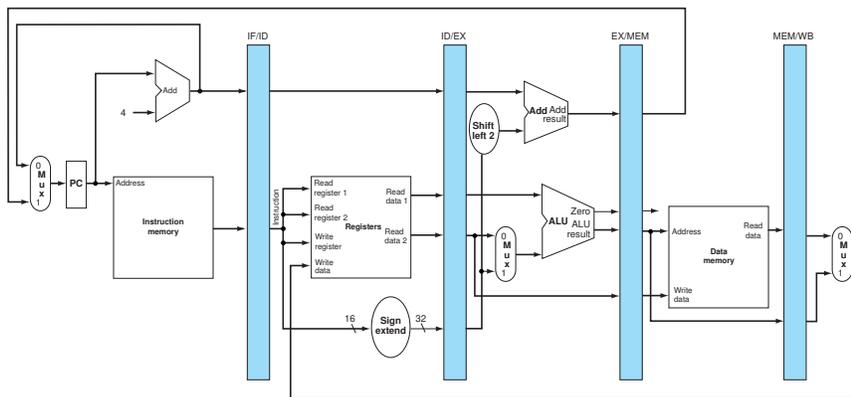
4

Basic Idea



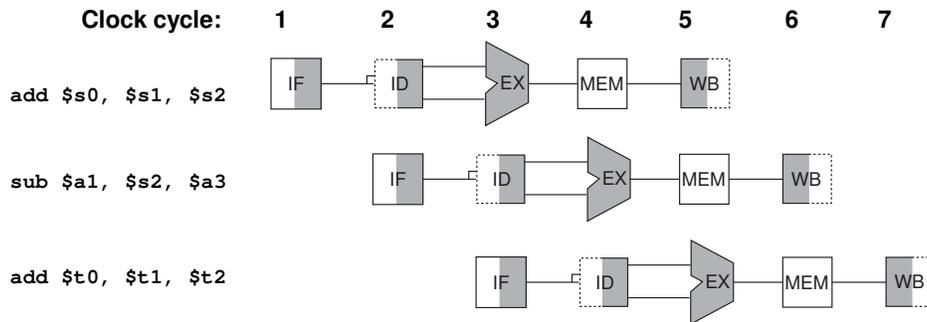
5

Pipelined Datapath



6

Pipeline Diagrams



Assumptions:

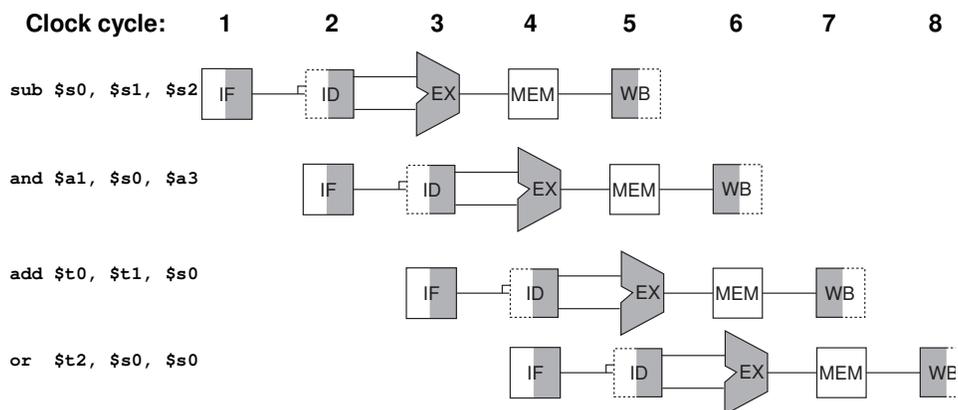
- Reads to memory or register file in 2nd half of clock cycle
- Writes to memory or register file in 1st half of clock cycle

What could go wrong?

7

Problem: Dependencies

- Problem with starting next instruction before first is finished



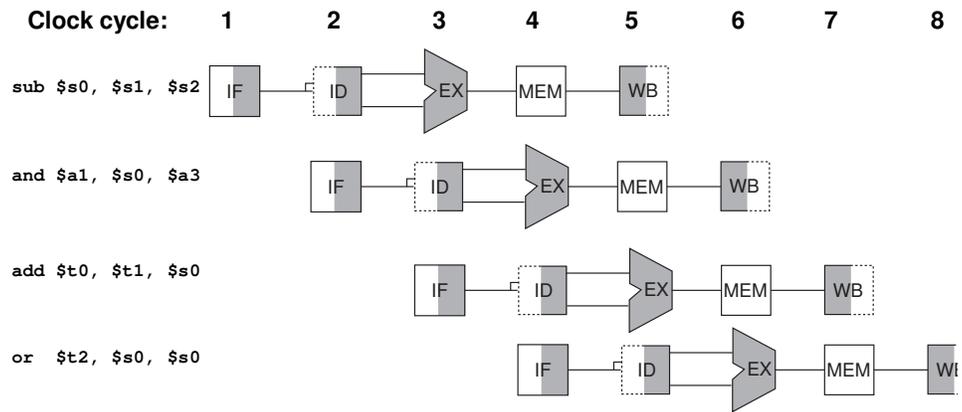
Dependencies that “go backward in time” are _____

Will the “or” instruction work properly?

8

Solution: Forwarding

Use temporary results, don't wait for them to be written

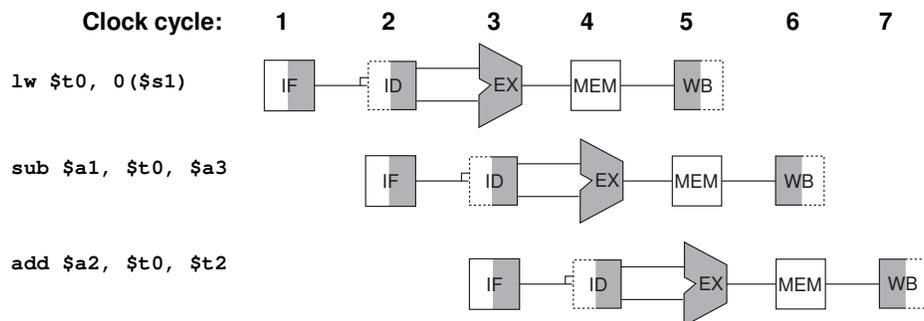


Where do we need this?

Will this deal with all hazards?

9

Problem?



Forwarding not enough...

When an instruction tries to _____
 a register following a _____
 to the same register.

10

Solution: “Stall” later instruction until result is ready

Clock cycle:	1	2	3	4	5	6	7
<code>lw \$t0, 0(\$s1)</code>							
<code>sub \$a1, \$t0, \$a3</code>							
<code>add \$a2, \$t0, \$t2</code>							

Why does the stall start after ID stage?

11

Assumptions

- For exercises/exams/everything assume...
 - The MIPS 5-stage pipeline
 - That we have forwarding
- ...unless told otherwise

12

Exercise #1 – Pipeline diagrams

- Draw a pipeline stage diagram for the following sequence of instructions. Start at cycle #1. You don't need fancy pictures – just text for each stage: ID, MEM, etc.

```
add $s1, $s3, $s4
lw  $v0, 0($a0)
sub $t0, $t1, $t2
```
- What is the total number of cycles needed to complete this sequence?
- What is the ALU doing during cycle #4?
- When does the sub instruction writeback its result?
- When does the lw instruction access memory?

13

Exercise #2 – Data hazards

- Consider this code:
 1. `add $s1, $s3, $s4`
 2. `add $v0, $s1, $s3`
 3. `sub $t0, $v0, $t2`
 4. `and $a0, $v0, $s1`
- 1. Draw lines showing all the data dependencies in this code
- 2. Which of these dependencies do not need forwarding to avoid stalling?

14

Exercise #3 – Data hazards

- Draw a pipeline diagram for this code. Show stalls where needed.
 1. `add $s1, $s3, $s4`
 2. `lw $v0, 0($s1)`
 3. `sub $v0, $v0, $s1`

15

Exercise #4 – More Data hazards

HW: 4-81 to 4-82

- Draw a pipeline diagram for this code. Show stalls where needed.
 1. `lw $s1, 0($t0)`
 2. `lw $v0, 0($s1)`
 3. `sw $v0, 4($s1)`
 4. `sw $t0, 0($t1)`

16

The Pipeline Paradox

- Pipelining does not _____ the execution time of any _____ instruction
- But by _____ instruction execution, it can greatly improve performance by _____ the _____

17

Structural Hazards

- Occur when the hardware can't support the combination of instructions that we want to execute in the same clock cycle
- MIPS instruction set designed to reduce this problem
- But could occur if:

18

Control Hazards

- What might be a problem with pipelining the following code?

```
    beq $a0, $a1, Else
    lw  $v0, 0($s1)
    sw  $v0, 4($s1)
Else:  add $a1, $a2, $a3
```

- What other kinds of instructions would cause this problem?

19

Control Hazard Strategy #1: Predict not taken

- What if we are wrong?
- Assume branch target and decision known at end of ID cycle. Show a pipeline diagram for when branch is taken.

```
    beq $a0, $a1, Else
    lw  $v0, 0($s1)
    sw  $v0, 4($s1)
Else:  add $a1, $a2, $a3
```

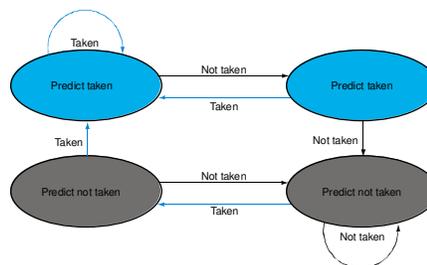
20

Control Hazard Strategies

1. Predict not taken
One cycle penalty when we are wrong – not so bad
Penalty gets bigger with longer pipelines – bigger problem
- 2.
- 3.

21

Branch Prediction



With more sophistication can get 90-95% accuracy
Good prediction key to enabling more advanced pipelining techniques!

22

Code Scheduling to Improve Performance

- Can we avoid stalls by rescheduling?

```
lw  $t0, 0($t1)
add $t2, $t0, $t2
lw  $t3, 4($t1)
add $t4, $t3, $t4
```

- Dynamic Pipeline Scheduling
 - Hardware chooses which instructions to execute next
 - Will execute instructions out of order (e.g., doesn't wait for a dependency to be resolved, but rather keeps going!)
 - Speculates on branches and keeps the pipeline full (may need to rollback if prediction incorrect)

23

Dynamic Pipeline Scheduling

- Let hardware choose which instruction to execute next (might execute instructions out of program order)
- Why might hardware do better job than programmer/compiler?

Example #1

```
lw  $t0, 0($t1)
add $t2, $t0, $t2
lw  $t3, 4($t1)
add $t4, $t3, $t4
```

Example #2

```
sw  $s0, 0($s3)
lw  $t0, 0($t1)
add $t2, $t0, $t2
```

24

Exercise #1

- Can you rewrite this code to eliminate stalls?
 1. `lw $s1, 0($t0)`
 2. `lw $v0, 0($s1)`
 3. `sw $v0, 4($s1)`
 4. `add $t0, $t1, $t2`

25

Exercise #2

HW: 4-86 to 4-87

- Show a pipeline diagram for the following code, assuming:
 - The branch is predicted not taken
 - The branch actually is taken

```
lw $t1, 0($t0)
beq $s1, $s2, Label2
sub $v0, $v1, $v2
Label2: add $t0, $t1, $t2
```

26

Pipeline Control

- Generate control signal during the _____ stage
- _____ control signals along just like the _____

Instruction	Execution/Address Calculation stage control lines				Memory access stage control lines			stage control lines	
	Reg Dst	ALU Op1	ALU Op0	ALU Src	Branch	Mem Read	Mem Write	Reg write	Mem to Reg
R-format	1	1	0	0	0	0	0	1	0
lw	0	0	0	1	0	1	0	1	1
sw	X	0	0	1	0	0	1	0	X
beq	X	0	1	0	1	0	0	0	X

