Principles of Computer Architecture

CSE 240A

Fall 2024

Hadi Esmaeilzadeh

hadi@ucsd.edu

University of California, San Diego

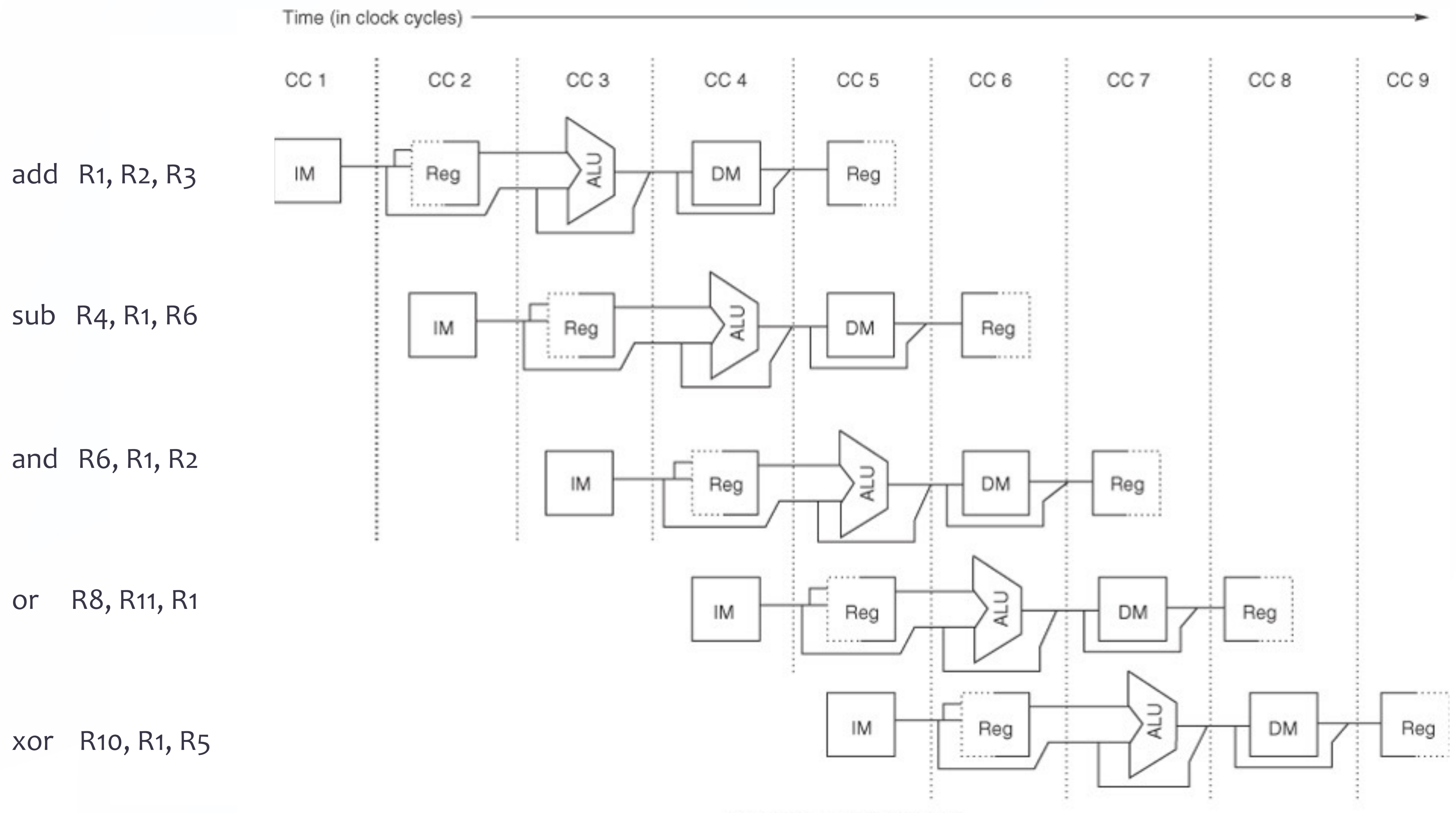
Pipeline Hazards

Data Hazards

```
ADD   R1, R2, R3
SUB   R4, R5, R1
AND   R6, R1, R7
OR    R8, R1, R9
XOR   R10, R1, R11
```

Data Dependence may result in data Hazard.

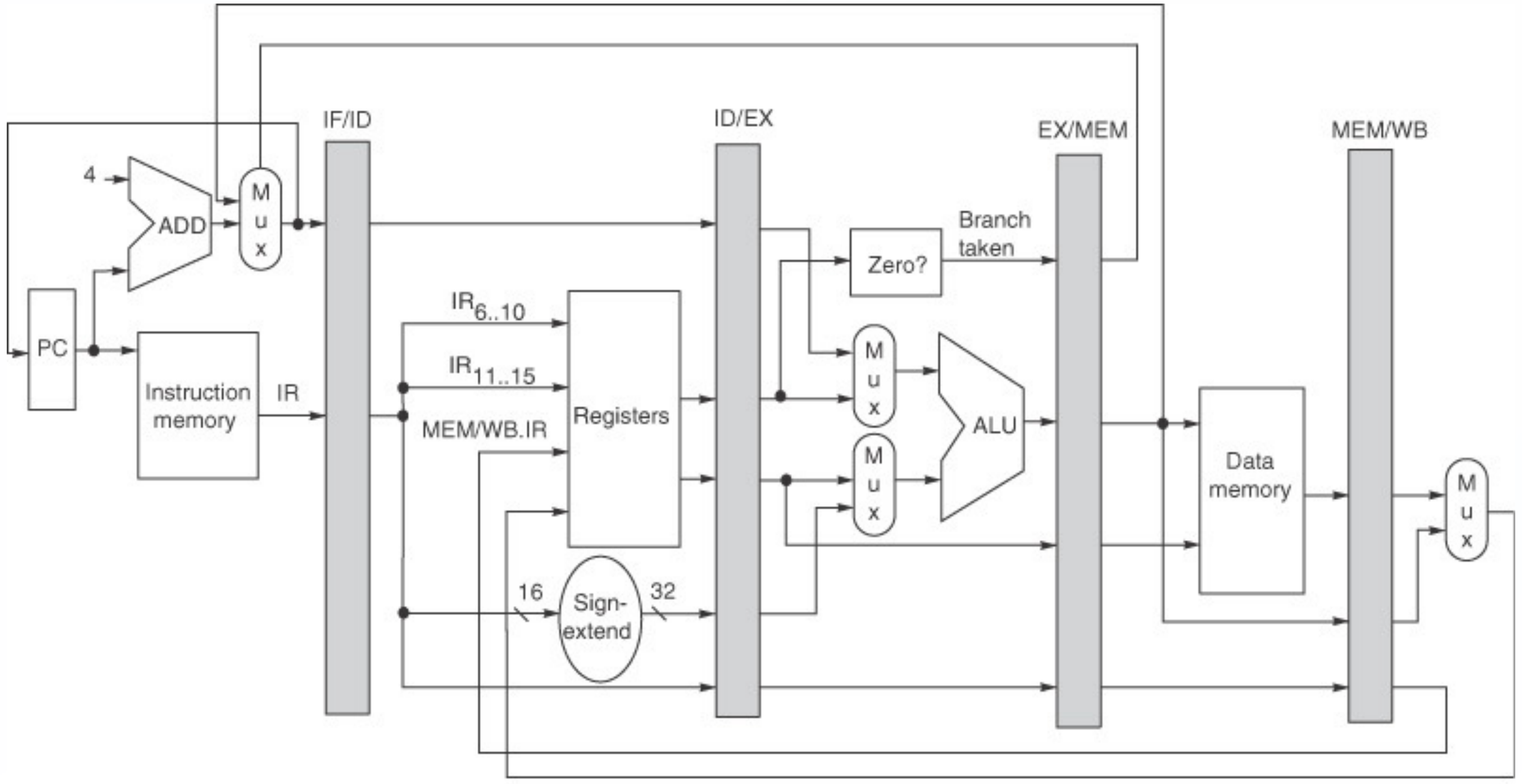
Data Hazards



Data Hazard

Let's see the wrong values on the wires

sub R7, R6, R3 add R6, R3, R1 addi R3, R1, #35



Data Dependence

- Data hazards are caused by data dependences
- Data dependences, and thus data hazards, come in 3 flavors (not all of which apply to this pipeline).
  - (read-after-write)
  - (write-after-write)
  - (write-after-read)

Which of the following pairs of instructions represent hazards which *only* apply if the instructions execute/finish out-of-order?

1

```
lw $1, 0 ($2)
add $1, $2, $3
```

2

```
lw $1, 0 ($2)
add $3, $2, $1
```

3

```
add $3, $2, $1
add $2, $1, $4
```

Selection	Instruction Pair
A	1
B	2
C	3
D	1 and 2
E	1 and 3

1: WAW
2: RAW
3: WAR

Let’s talk through reg rename (virtual registers in a way)

RAW Hazard

True, flow dependence

- later instruction tries to read an operand before earlier instruction writes it
- The dependence

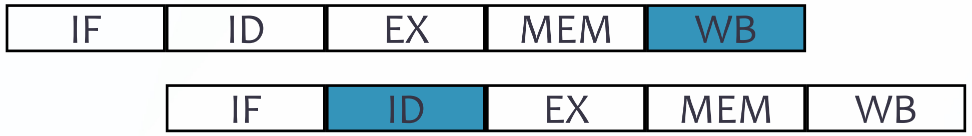
  add **R1,** R2, R3

  sub R5, **R1,** R4

- The hazard

  add **R1,** R2, R3

  sub R5, **R1,** R4



- RAW hazard is extremely common

WAW Hazard

false dependence,
Output dependence

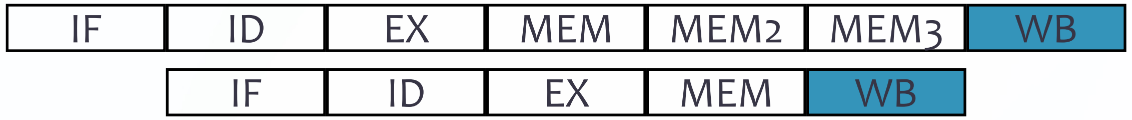
- later instruction tries to write an operand before earlier instruction writes it
- The dependence

  add **R1**, R2, R3

  sub **R1**, R2, R4
- The hazard

  lw **R1**, R2, R3

  sub **R1**, R2, R4



- WAW hazard possible in a reasonable pipeline, but not in the very simple pipeline we're assuming.

WAR Hazard

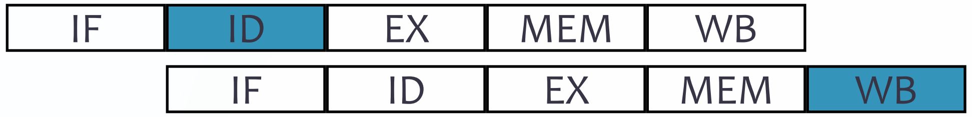
false dependence,
anti-dependence

- later instruction tries to write an operand before earlier instruction reads it
- The dependence

  add R1, **R2**, R3

  sub **R2**, R5, R4

- The hazard?



  add R1, **R2**, R3

  sub **R2**, R5, R4

- WAR hazard is uncommon/impossible in a reasonable (in-order) pipeline

RAW Solutions?

The value is where we want, let's see the next slide for pipeline

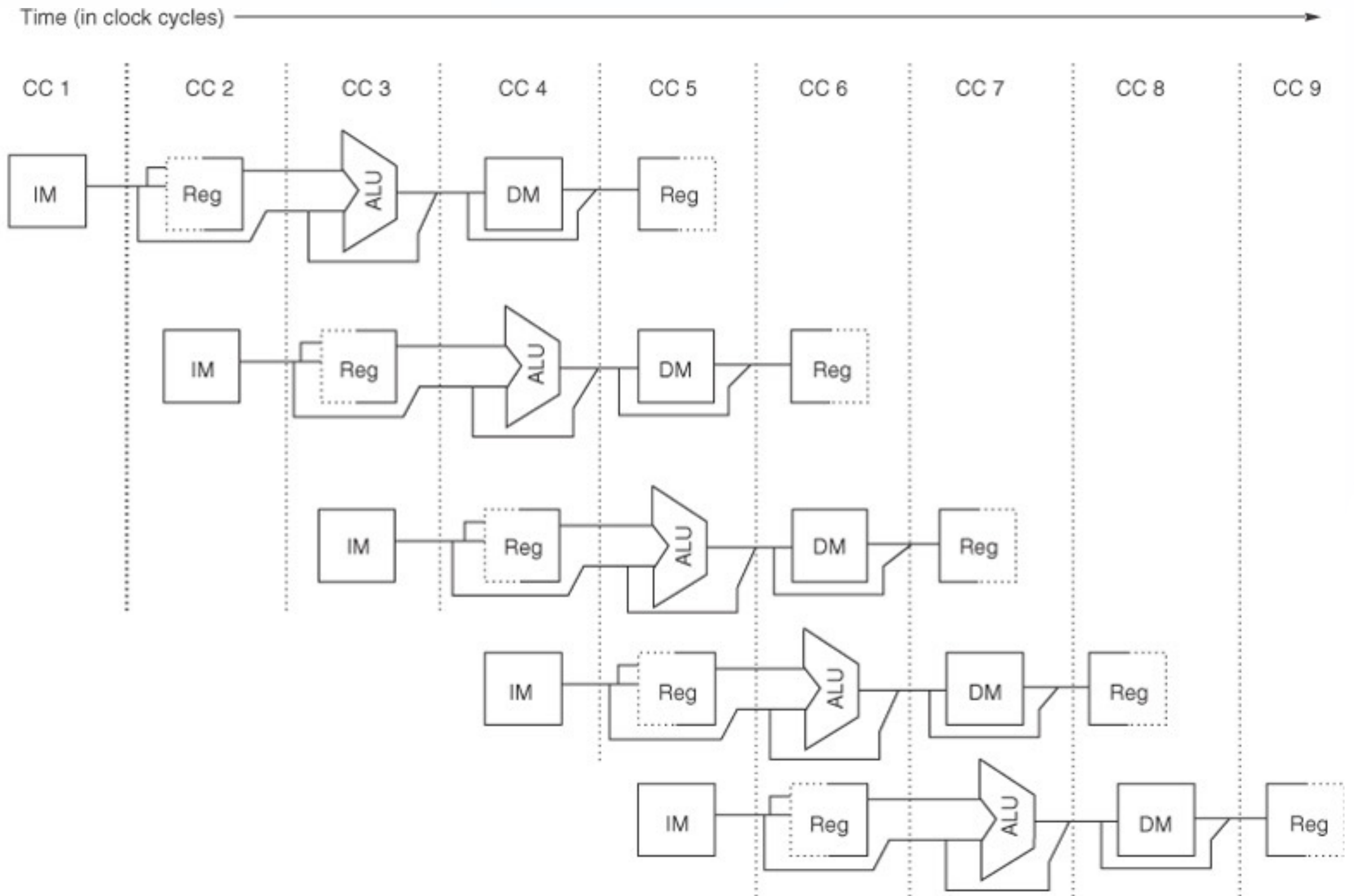
add R1, R2, R3

sub R4, R1, R6

and R6, R1, R2

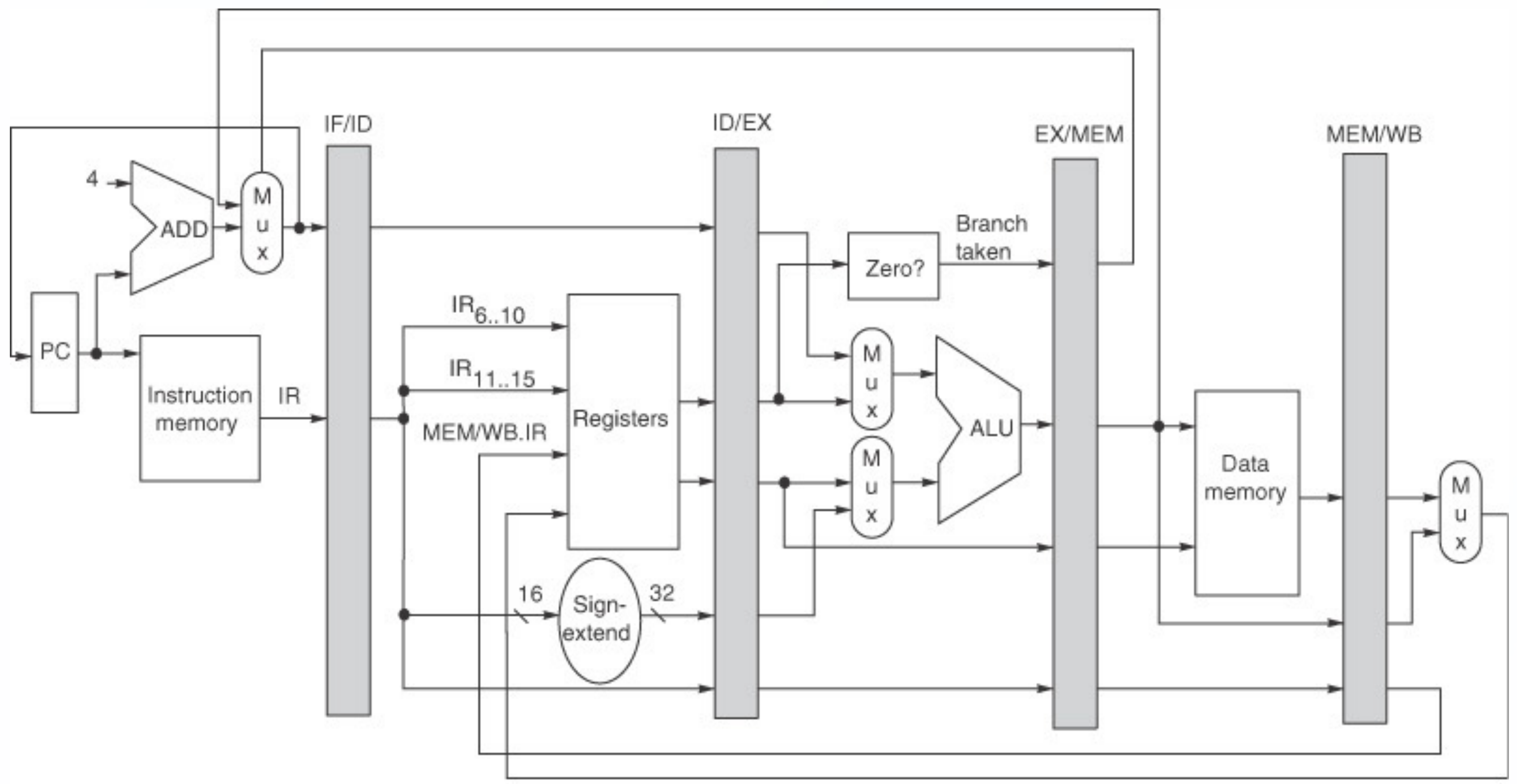
or R8, R11, R1

xor R10, R1, R5



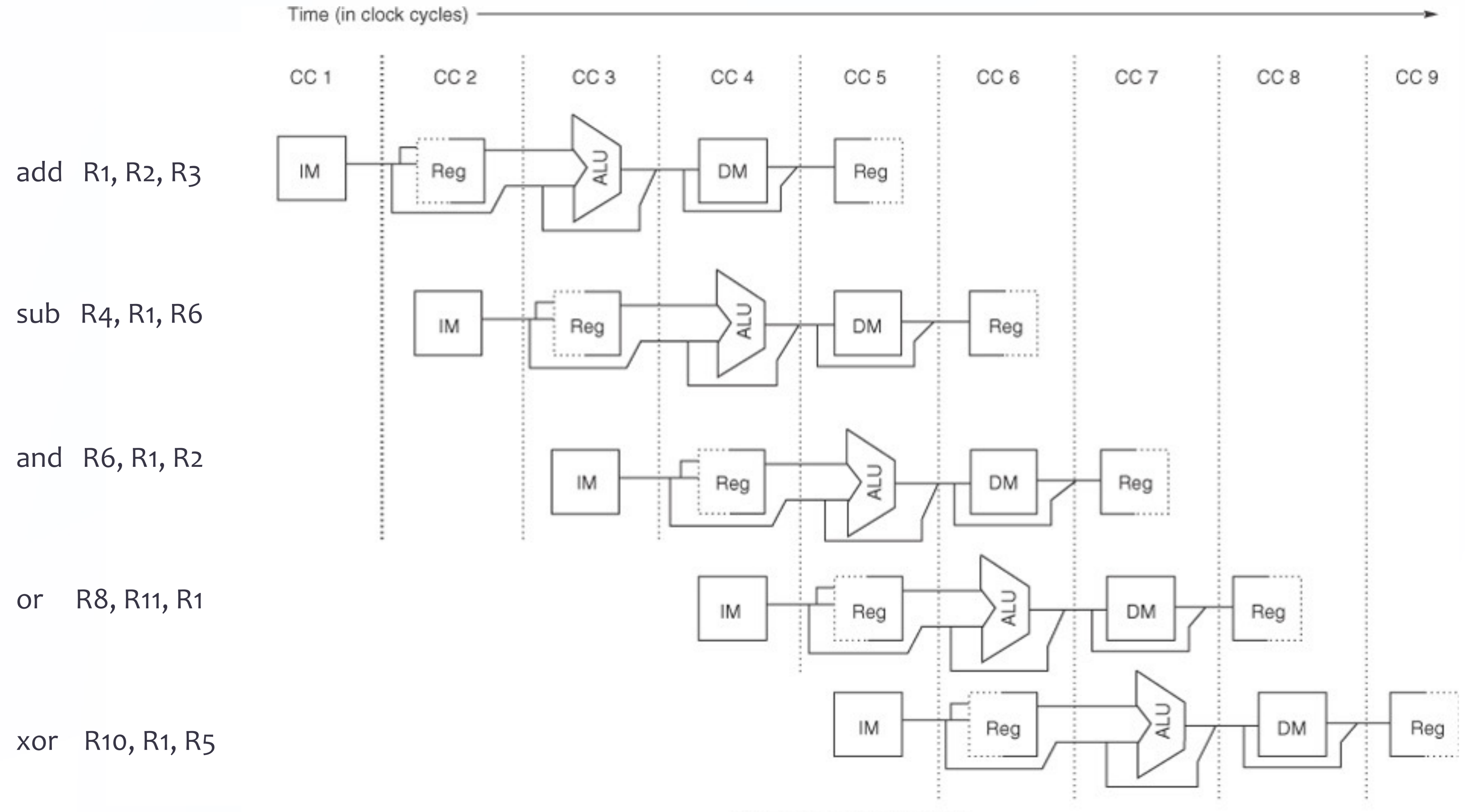
Dealing with Data Hazards through Forwarding

AND R6, R4, R7   SUB R4, R1, R5   ADD R1, R2, R3



The value we want is already there!

RAW Solutions?

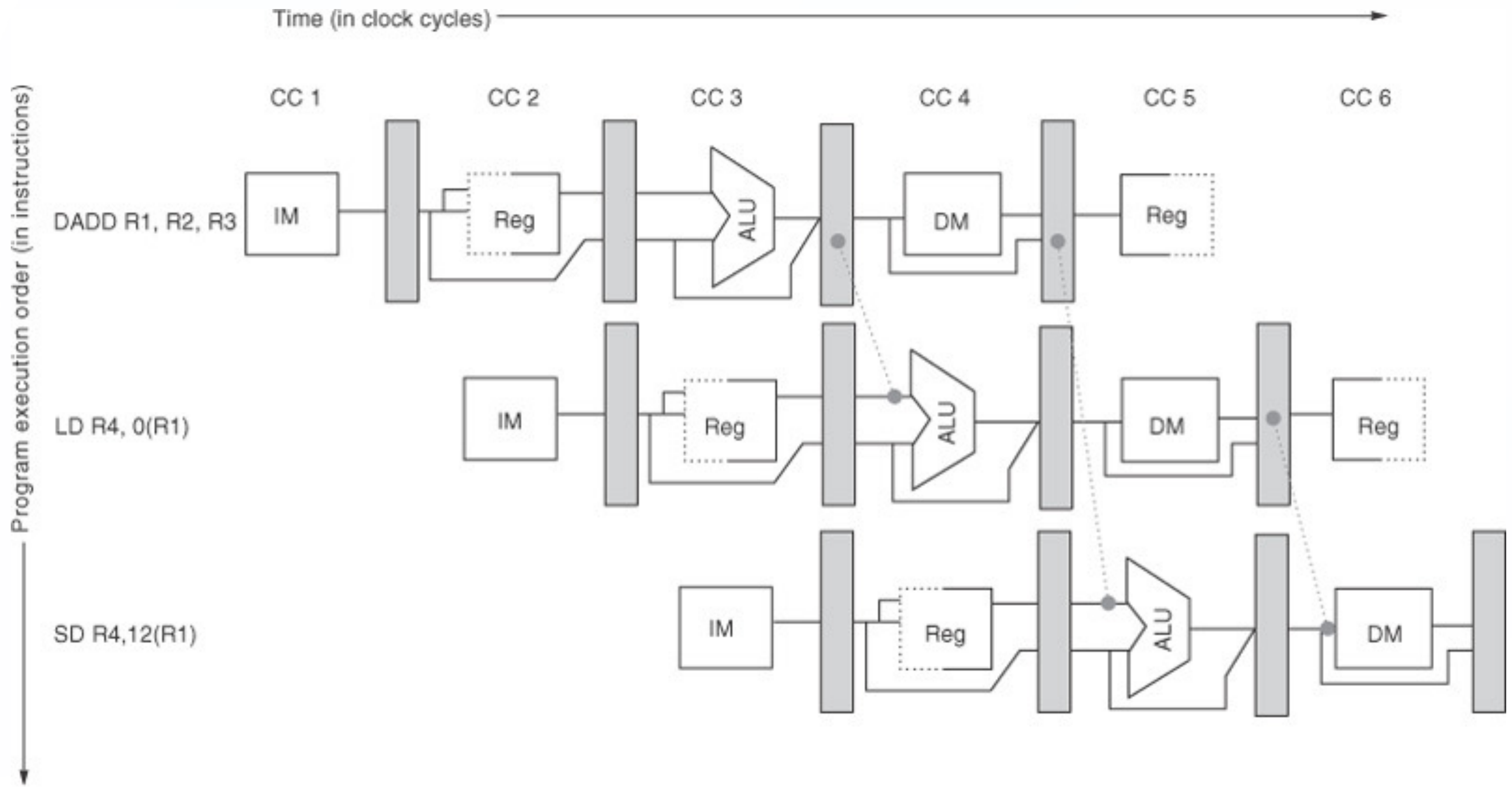


Forwarding Options

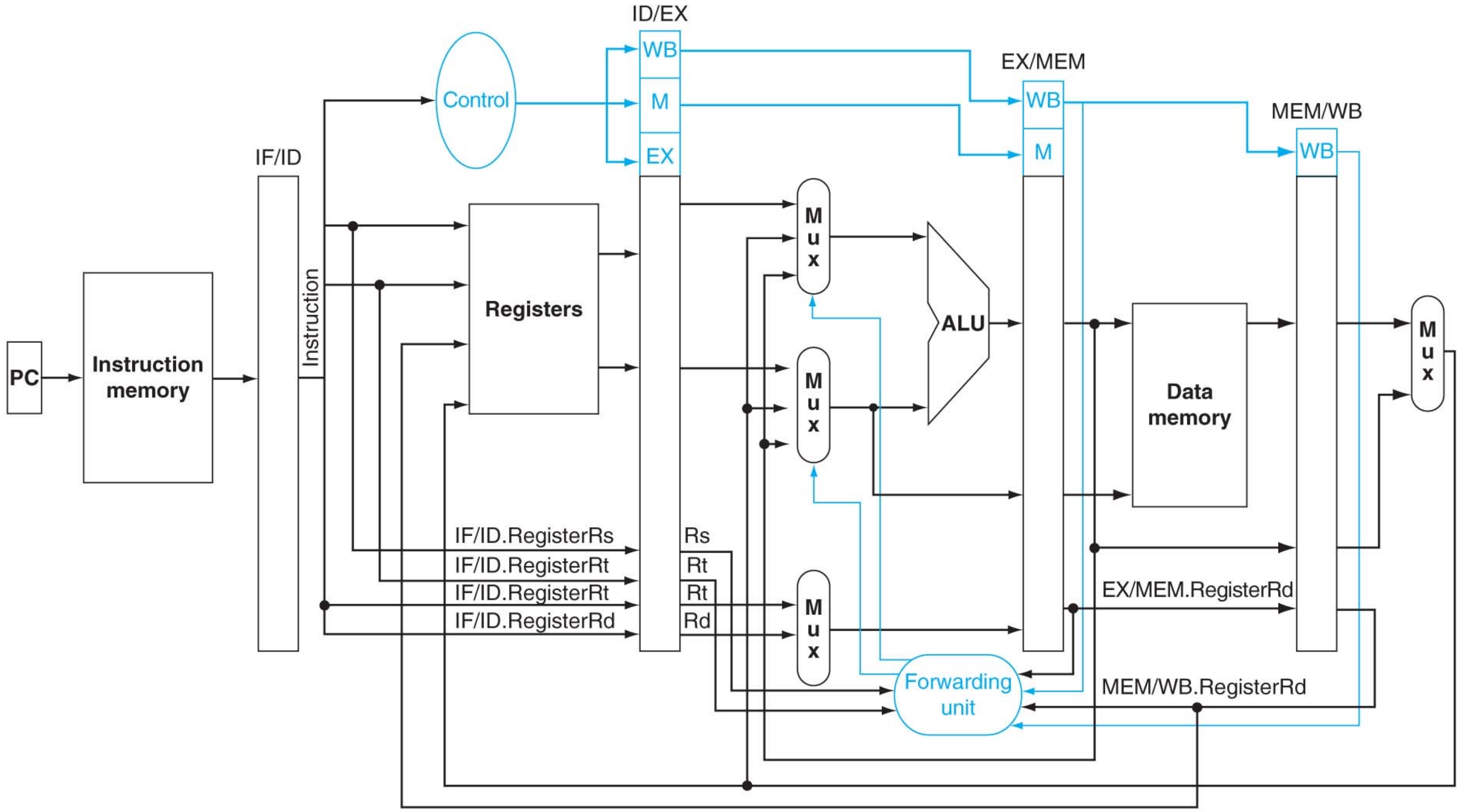
- ADD -> ADD
- ADD -> LW
- ADD -> SW (2 operands)
- LW -> ADD
- LW -> LW
- LW -> SW (2 operands)

(I’m letting ADD stand in for all ALU operations)

More Forwarding

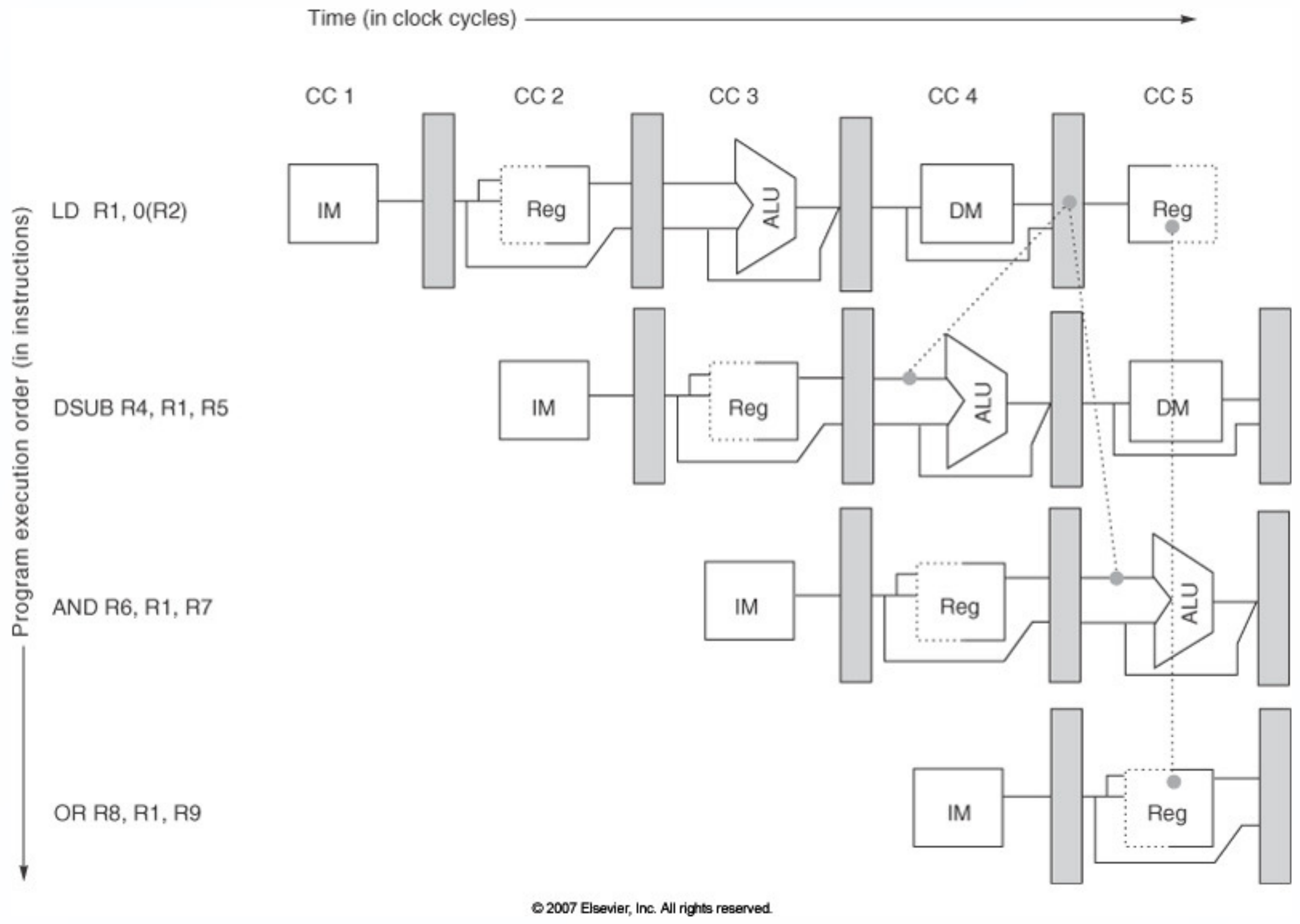


Forwarding in the Pipeline



(Picture from undergrad text)

More Forwarding



Forwarding and Stalling

```
ld    R1,  0(R2)     IF    ID    EX    WB    ID

dsub   R4,  R1, R6

dand   R6,  R1, R7

dor     R8,  R1, R9
```

Example

```
DADD  R1, R2, R3
SD  R1, 1000(R2)
LD  R7, 2000(R2)
DADD R5, R7, R1
LD  R8, 2004(R2)
SD R7, 2008(R8)
DADD R8, R8, R2
LD  R9, 1012(R8)
SD R9, 1016(R8)
```

Suppose IF and M are the longest stages. To reduce CT, we split each in half. Given the following pipeline:

IF1 IF2 ID EX M1 M2 WB

Let's check our understanding

Assume the input data must be available at the start of IF1/M1 and the output is available after IF2/M2. How many hardware stalls would be required in the following code (assuming hardware forwarding wherever possible)?

```
ld r1, 8 (r2)
sd r1, 0 (r3)
```

Selection	Number of stalls
A	0
B	1
C	2
D	3
E	4

IF = 200ps
ID = 100ps
EX = 100ps
M = 200ps
WB = 100ps

Hardware engineers determine these to be the execution times per stage of the MIPS 5-stage pipeline processor. Consider splitting IF and M into 2 stages each. (So IF1 IF2 and M1 M2.) The most important code run by the company is (assume branch is taken most of the time):

```
Loop:  ld r1, 0 (r2)
       dadd r2, r3, r4
       dsub r5, r1, r2
       beq r5, $zero, Loop
```

What would be the impact of the new 7-stage pipeline compared to the original 5-stage MIPS pipeline.. Assume the pipeline has forwarding where available, predicts branch not taken, and resolves branches in ID.

Selection	CPI	CT
A	Increase	Increase
B	Increase	Decrease
C	Decrease	Increase
D	Decrease	Decrease
E	Increase	No Change

Avoiding Pipeline Stalls (MIPS)

```
ld R1, 1000(R2)
ld R3, 2000(R2)
dadd R4, R1, R3
ld R1, 3000(R2)
add R6, R4, R1
sd R6, 1000(R2)
```

Activity: Find a way to avoid stalling in the 5-stage MIPS pipeline. You may need to relax some constraints…. Work with your group for ~5 minutes.

Avoiding Pipeline Stalls

```
ld R1, 1000(R2)
ld R3, 2000(R2)
dadd R4, R1, R3
ld R1, 3000(R2)
add R6, R4, R1
sd R6, 1000(R2)
```

Can’t do it without
Sw register renaming.
Focus on the 2nd load to
rename R1 and we are Good.

- this is a compiler technique called *instruction scheduling*.

How big a problem are these pipeline stalls?

Effect on CPI?

- 13% of the loads in floating point programs
- 25% of the loads in integer programs

Key Points

- Pipeline improves throughput rather than latency
- Pipelining gets parallelism without replication
- ET = IC * CPI * CT
- Keeping the pipeline full is no easy task
  - structural hazards, data hazards, control hazards
- Data Hazards require dependent instructions to wait for the producer instruction
  - Most of the problem handled with forwarding (bypassing)
  - Sometimes stall still required (especially in modern processors)