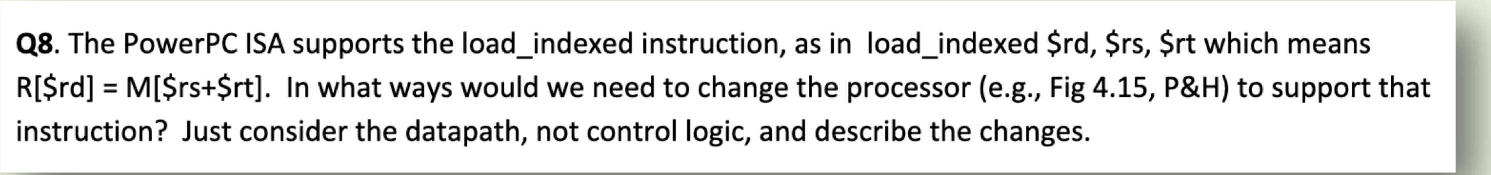
Announcements

- Reminder: Clocks change on Sunday
  - You get 2am twice!
- Midterm logistics
  - Following the path-of-most-success from other faculty
    - Will be on canvas
    - Will be timed
    - Will allow forward progress only
      - N.b. This is how many standardized tests you will encounter (e.g. GRE) work
    - We will give you, *in advance*, a list of questions and suggested timing
    - Will **not** use the (IMHO creepy) remote proctoring, etc services
    - Open book, not open Google [but you will not have **time** to look everything up!]
  - The participation quizzes will be configured this way so you can get comfortable
    - Only difference is participation quizzes will let you see correct answers when you're done

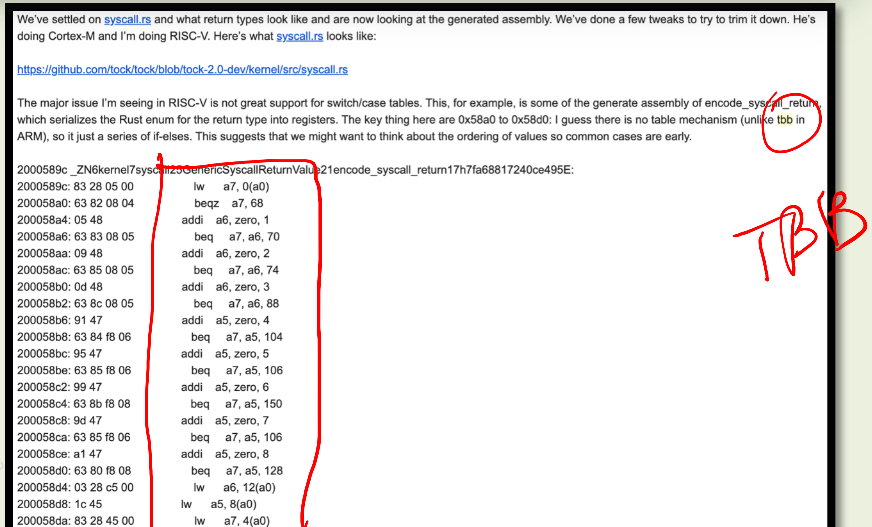
A brief aside: CSE 141 in the real world

- Literally, in the hour before lecture, during the TockOS project call…



Q8. The PowerPC ISA supports the load_indexed instruction, as in load_indexed $rd, $rs, $rt which means R[$rd] = M[$rs+$rt]. In what ways would we need to change the processor (e.g., Fig 4.15, P&H) to support that instruction? Just consider the datapath, not control logic, and describe the changes.

- RISC-V does not support a branch_indexed instruction, which makes switch/case expensive in code size, which may change how our OS works



We've settled on syscall.rs and what return types look like and are now looking at the generated assembly. We've done a few tweaks to try to trim it down. He's doing Cortex-M and I'm doing RISC-V. Here's what syscall.rs looks like:

https://github.com/tock/tock/blob/tock-2.0-dev/kernel/src/syscall.rs

The major issue I'm seeing in RISC-V is not great support for switch/case tables. This, for example, is some of the generate assembly of encode_syscall_return, which serializes the Rust enum for the return type into registers. The key thing here are 0x58a0 to 0x58d0: I guess there is no table mechanism (unlike tbb in ARM), so it just a series of if-elses. This suggests that we might want to think about the ordering of values so common cases are early.

```
2000589c _ZN6kernel7syscall25GenericSyscallReturnValue21encode_syscall_return17h7fa68817240ce495E:
2000589c: 83 28 05 00        lw      a7, 0(a0)
200058a0: 63 82 08 04        beqz    a7, 68
200058a4: 05 48              addi    a6, zero, 1
200058a6: 63 83 08 05        beq     a7, a6, 70
200058aa: 09 48              addi    a6, zero, 2
200058ac: 63 85 08 05        beq     a7, a6, 74
200058b0: 0d 48              addi    a6, zero, 3
200058b2: 63 8c 08 05        beq     a7, a6, 88
200058b6: 91 47              addi    a5, zero, 4
200058b8: 63 84 f8 06        beq     a7, a5, 104
200058bc: 95 47              addi    a5, zero, 5
200058be: 63 85 f8 06        beq     a7, a5, 106
200058c2: 99 47              addi    a5, zero, 6
200058c4: 63 8b f8 08        beq     a7, a5, 150
200058c8: 9d 47              addi    a5, zero, 7
200058ca: 63 85 f8 06        beq     a7, a5, 106
200058ce: a1 47              addi    a5, zero, 8
200058d0: 63 80 f8 08        beq     a7, a5, 128
200058d4: 03 28 c5 00        lw      a6, 12(a0)
200058d8: 1c 45              lw      a5, 8(a0)
200058da: 83 28 45 00        lw      a7, 4(a0)
```

TBB

Is it really that easy?

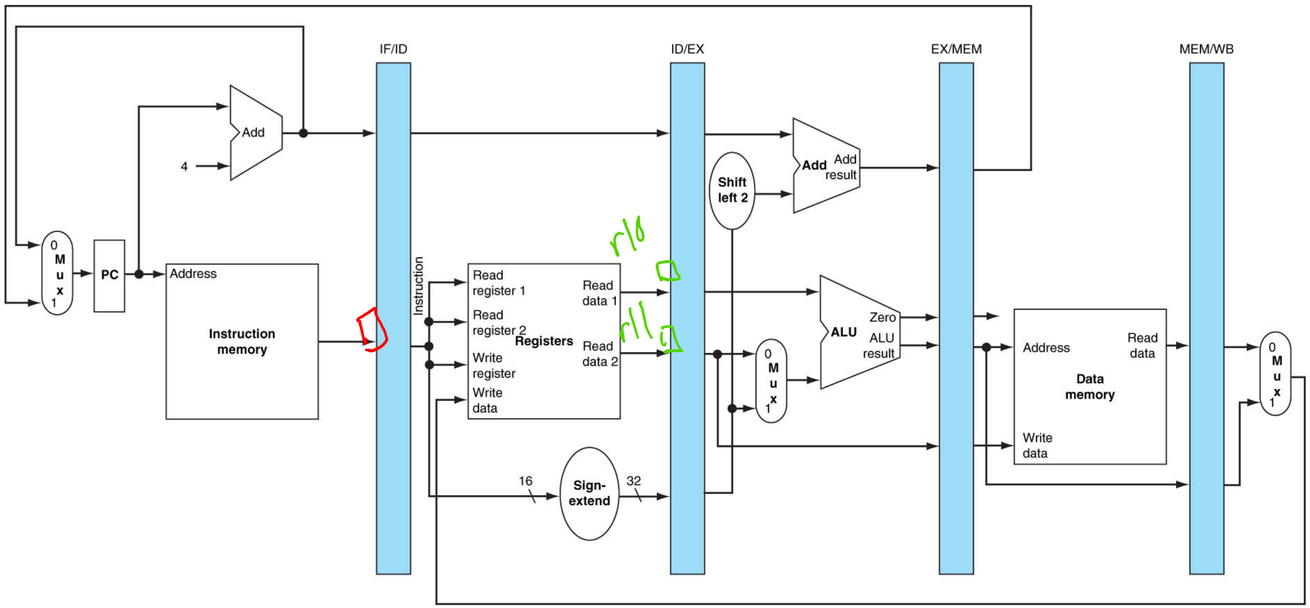
- What happens when…

```
add $3, $10, $11
lw $8, 1000($3)
sub $11, $8, $7
```

The Pipeline in Execution

lw $8, 1000($3) | add $3, $10, $11 | Execute/ Address Calculation | Memory Access | Write Back

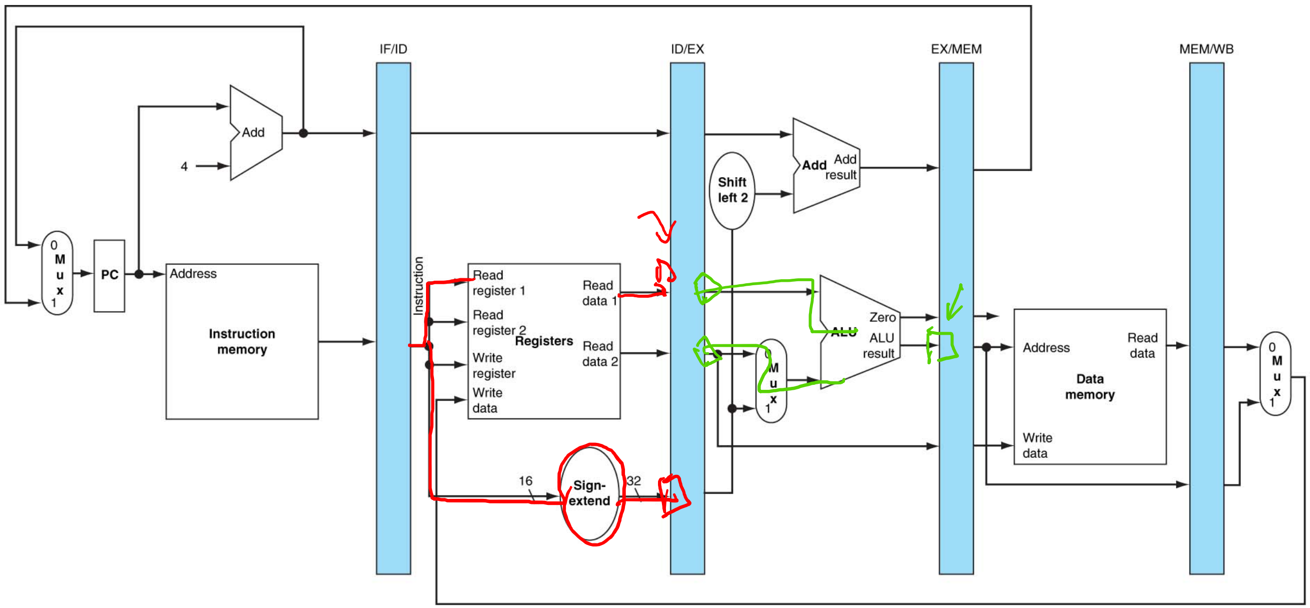
r3



The Pipeline in Execution

sub $11, $8, $7 lw $8, 1000($3) add $3, $10, $11 Memory Access Write Back

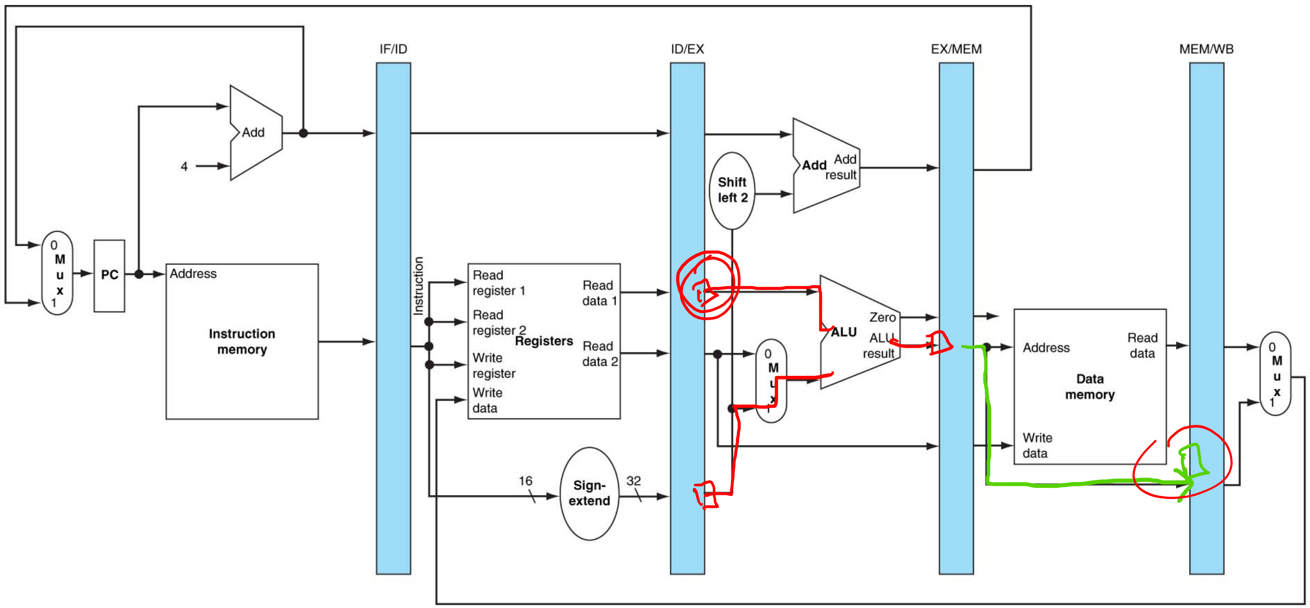
ID



The Pipeline in Execution

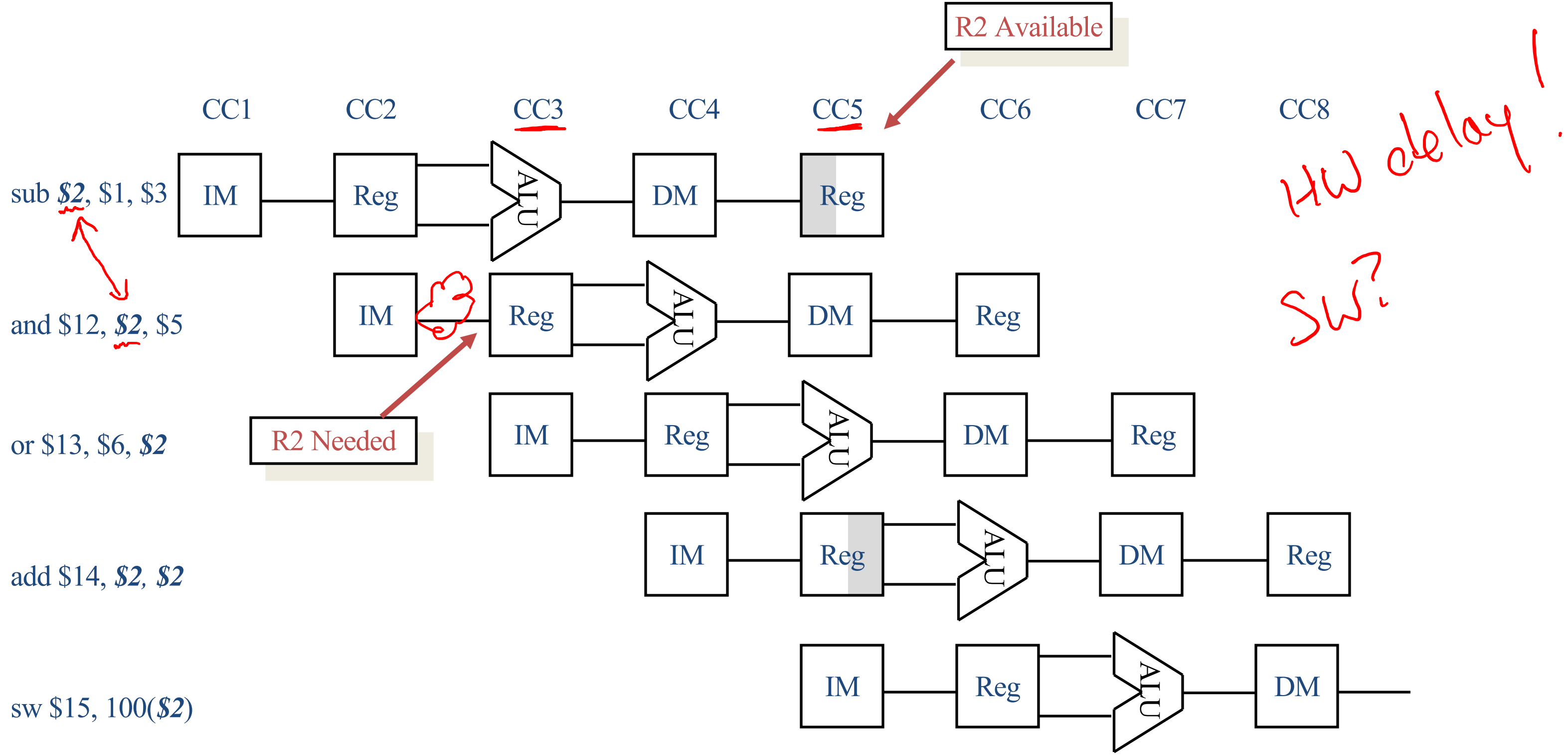
add $10, $1, $2   sub $11, $8, $7   lw $8, 1000($3)   add $3, $10, $11   Write Back

EX



Data Hazards

When a result is needed in the pipeline before it is available, a **data hazard** occurs. *What can we do?*



Data Hazards

```
sub $2, $1, $3
and $4, $2, $5
or  $8, $2, $6
add $9, $4, $2
slt $1, $6, $7
```

- Data Hazards are caused by data dependences
- Not all data dependences result in data hazards
- A data hazard results when there is a data dependence between two instructions that appear too close together in the pipeline

- We will define a data hazard as any data dependence that requires either the software or hardware to take special action to get correct

Dealing With Data Hazards – What can we do…

- …in Software?
  - add no-op (nop)
- …in Hardware?
  - use control signals to introduce ~~a delay~~ HW nops
  - "forward"

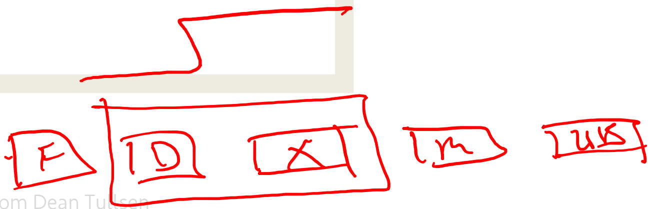
delay ☹

delay ☹

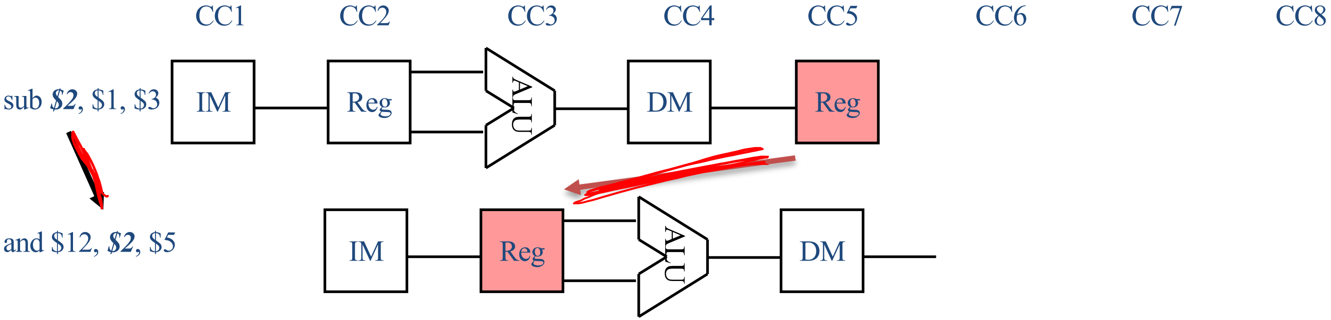
no delay? ☺

Data Hazards are caused by *instruction dependences*. For example, the add is data-dependent on the subtract:

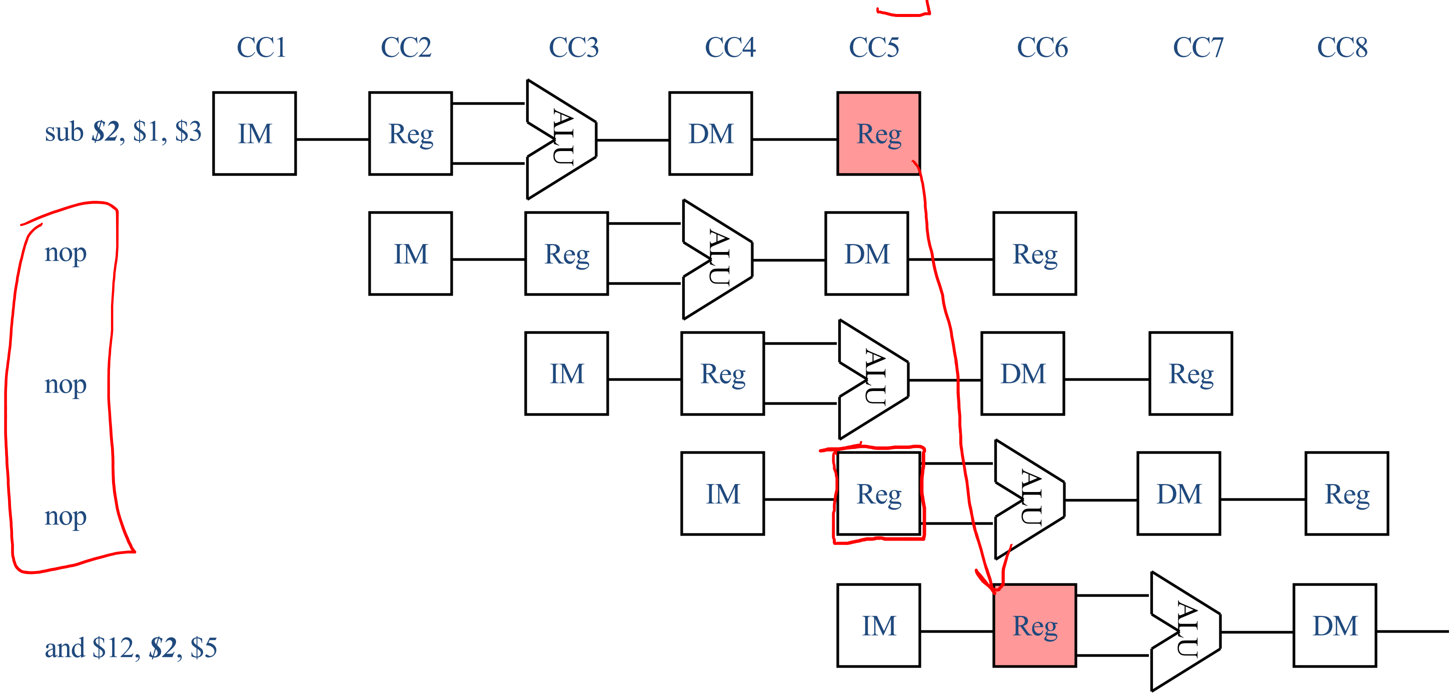
```
        subi $5, $4, #45
        add  $8, $5, $2
```



Dealing with Data Hazards in Software



Dealing with Data Hazards in Software



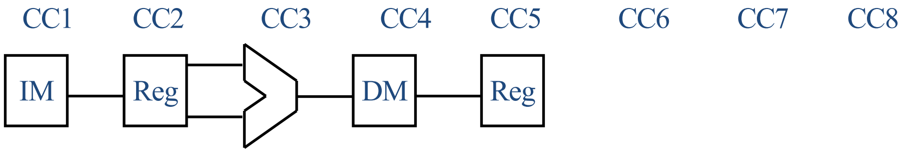
How Many No-ops?

```
sub $2, $1,$3
and $4, $2,$5
or   $8, $2,$6
add $9, $4,$2
slt   $1, $6,$7
```

Are No-ops Really Necessary?

```
sub $2, $1,$3
and $4, $2,$5
or   $8, $3,$6
add $9, $2,$8
slt   $1, $6,$7
```

Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls



sub ***$2***, $1, $3

and $12, ***$2***, $5

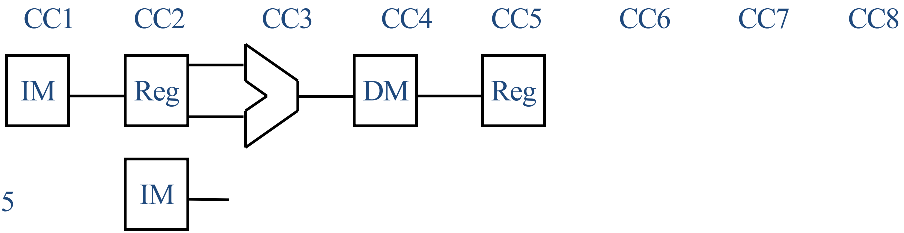
or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls



sub ***$2***, $1, $3

and $12, ***$2***, $5

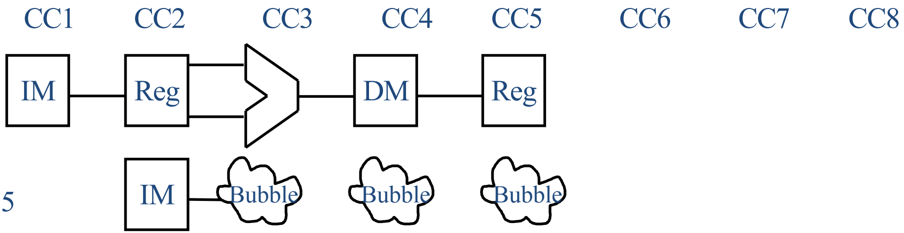
or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls



sub ***$2***, $1, $3

and $12, ***$2***, $5

or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

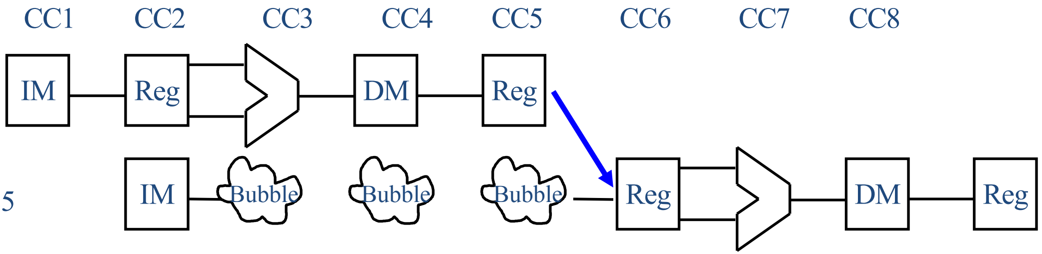
Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls

HW
delay

nop
[illegible]

sub ***$2***, $1, $3

and $12, ***$2***, $5



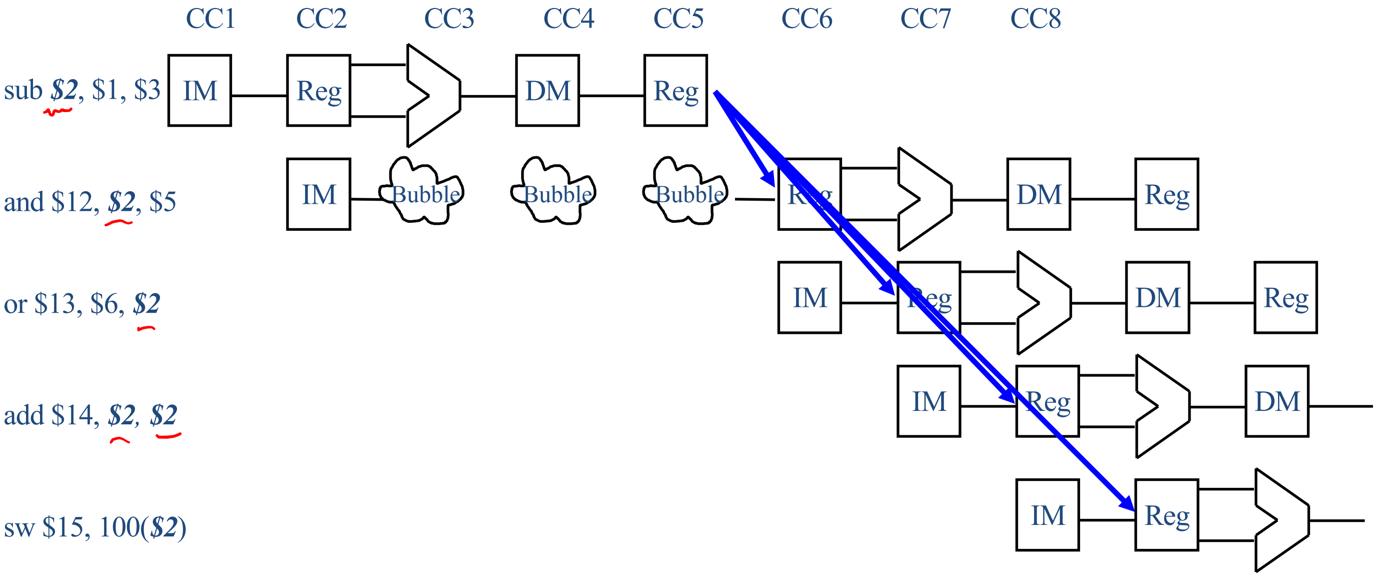
or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls



sub *$2*, $1, $3

and $12, *$2*, $5

or $13, $6, *$2*

add $14, *$2*, *$2*

sw $15, 100(*$2*)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls (alt. View)

CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8

sub ***$2***, $1, $3 IM

and $12, ***$2***, $5

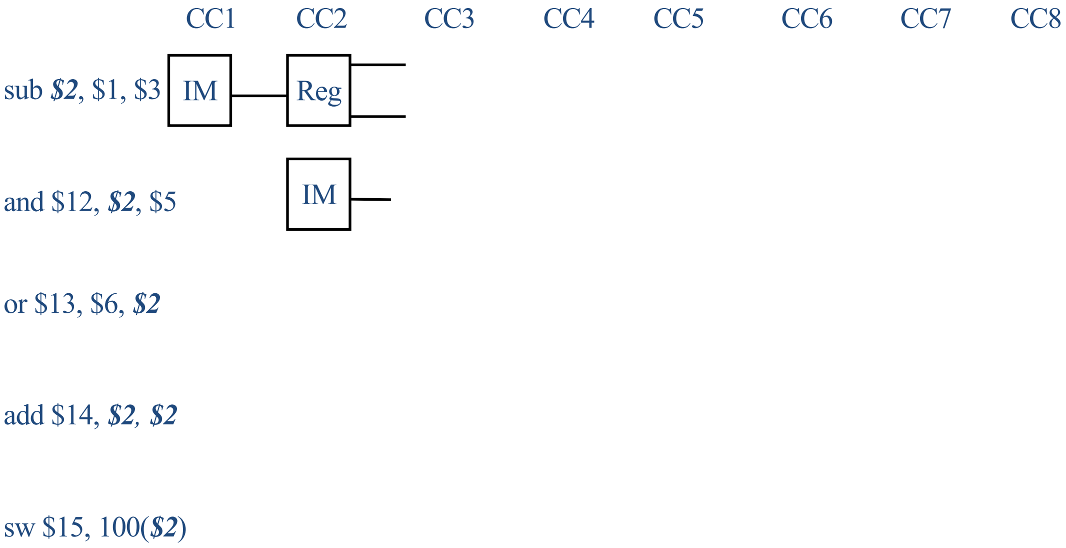
or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls (alt. View)



CC1 CC2 CC3 CC4 CC5 CC6 CC7 CC8

sub ***$2***, $1, $3 IM Reg

and $12, ***$2***, $5 IM

or $13, $6, ***$2***

add $14, ***$2**, **$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls (alt. View)

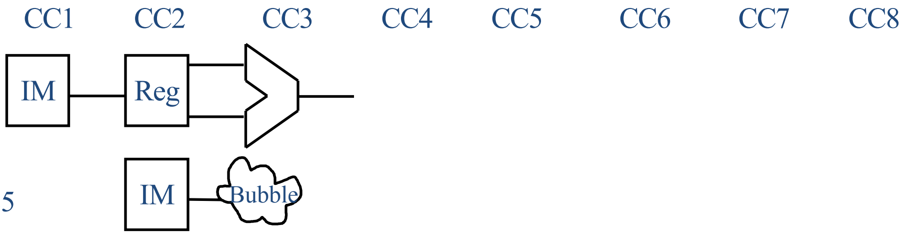
sub ***$2***, $1, $3

and $12, ***$2***, $5

or $13, $6, ***$2***

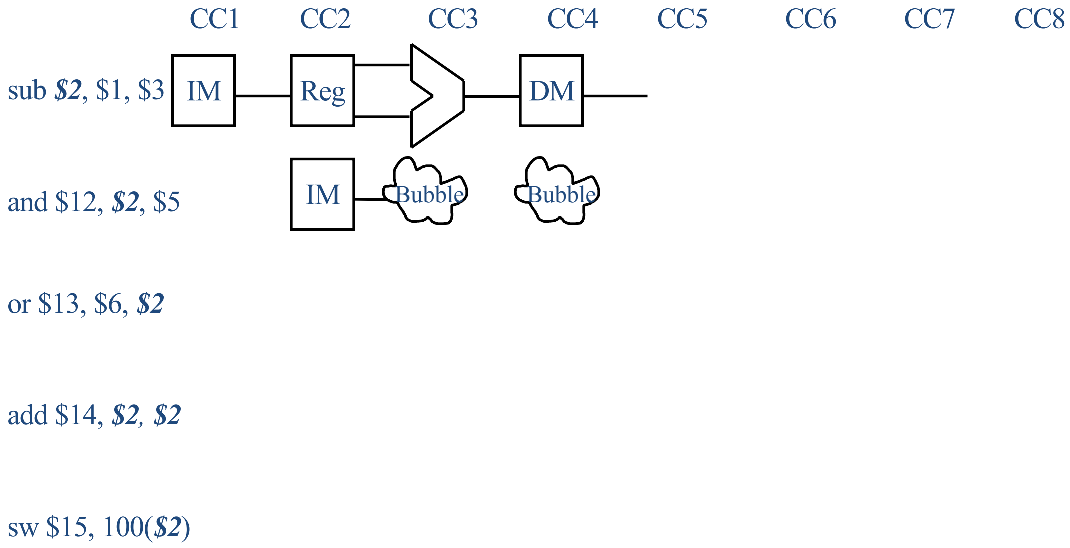
add $14, ***$2**, **$2***

sw $15, 100(***$2***)



Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls (alt. View)



sub ***$2***, $1, $3

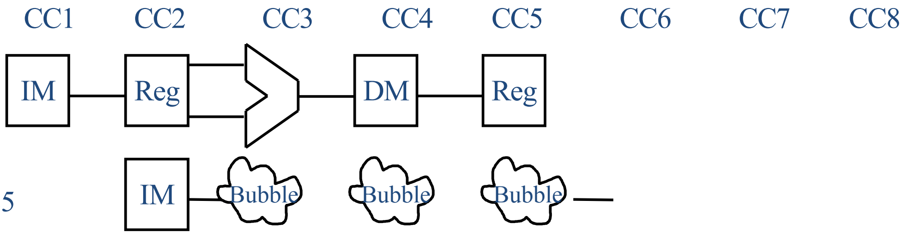
and $12, ***$2***, $5

or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware
Part II-Pipeline Stalls (alt. View)



sub ***$2***, $1, $3

and $12, ***$2***, $5

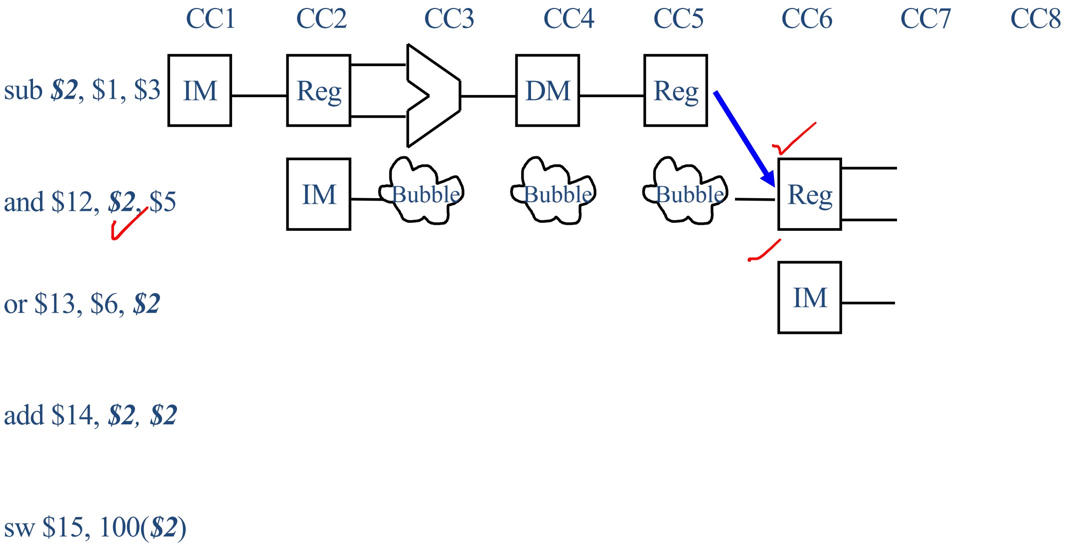
or $13, $6, ***$2***

add $14, ***$2***, ***$2***

sw $15, 100(***$2***)

Dealing with Data Hazards in Hardware

Part II-Pipeline Stalls (alt. View)



sub *$2*, $1, $3

and $12, *$2*, $5

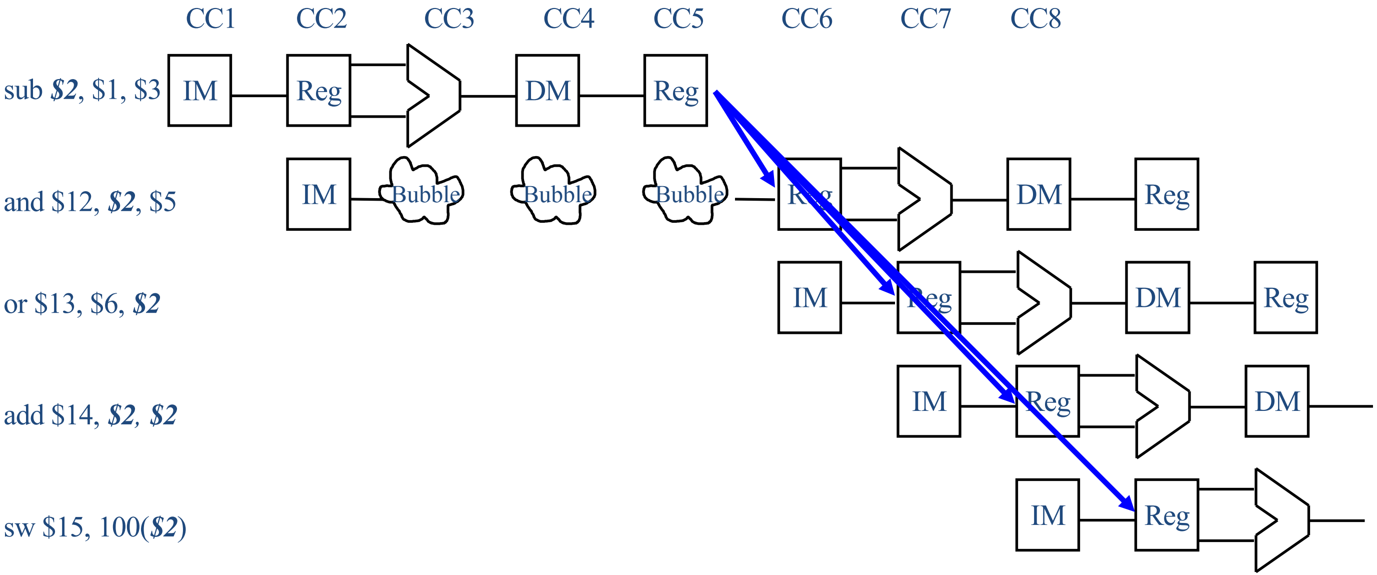
or $13, $6, *$2*

add $14, *$2*, *$2*

sw $15, 100(*$2*)

Dealing with Data Hazards in Hardware

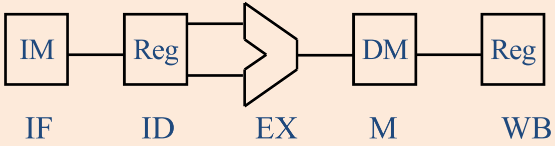
Part II-Pipeline Stalls (alt. View)



Poll Q: Try it yourself

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
sub $2, $1, $3	IF	ID	EX	M	WB			
add $12, $3, $5								
or $13, $6, $2								
add $14, $12, $2								
sw $14, 100($2)								

	How many bubbles?
A	5
B	6
C	7
D	8
E	None of the above



Working this example...

	CC1	CC2	CC3	CC4	CC5	CC6	CC7	CC8
sub $2, $1, $3	IF	ID	EX	M	WB			
add $12, $3, $5		IF	ID	EX	M	WB		
or $13, $6, $2			IF			ID	EX	M
add $14, $12, $2						IF	ID	X
sw $14, 100($2)							IF	

#2

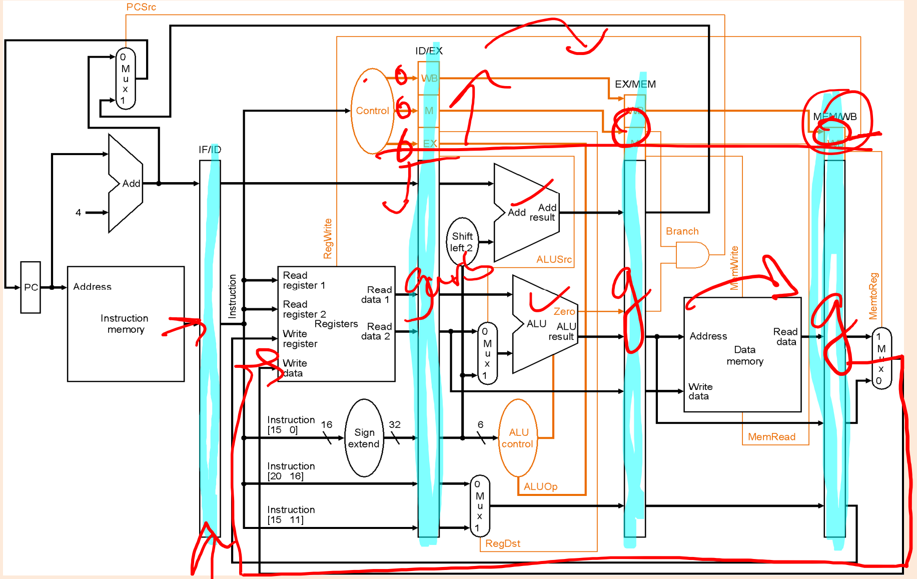
R12

Poll Q: How to actually implement this in hardware?

Once you detect the hazard in ID – what must you do to **insert the nop and "stall"**?

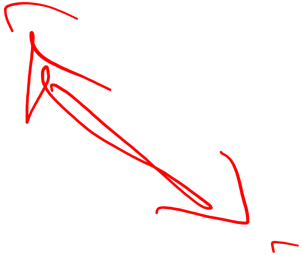
1. Flush all instructions in the pipeline (set control signals to 0).
2. Set all control signals going to ID/EX register to zero.
3. Set PCWrite to zero.
4. Set IF/ID register write to zero.

Selection	Changes
A	1, 3, 4
B	1, 2, 3
[✓ C]	2, 3, 4
D	1
E	None of the above

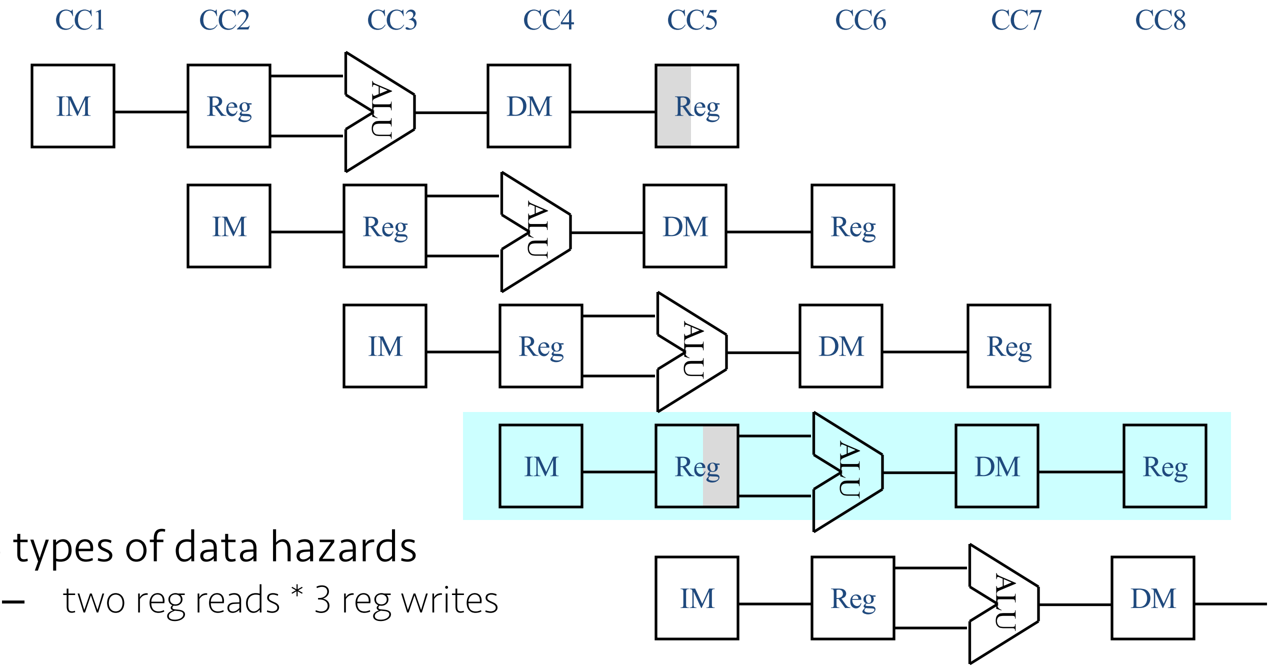


Pipeline Stalls

- To ensure proper pipeline execution in light of register dependences, we must:
  - detect the hazard
  - stall the pipeline

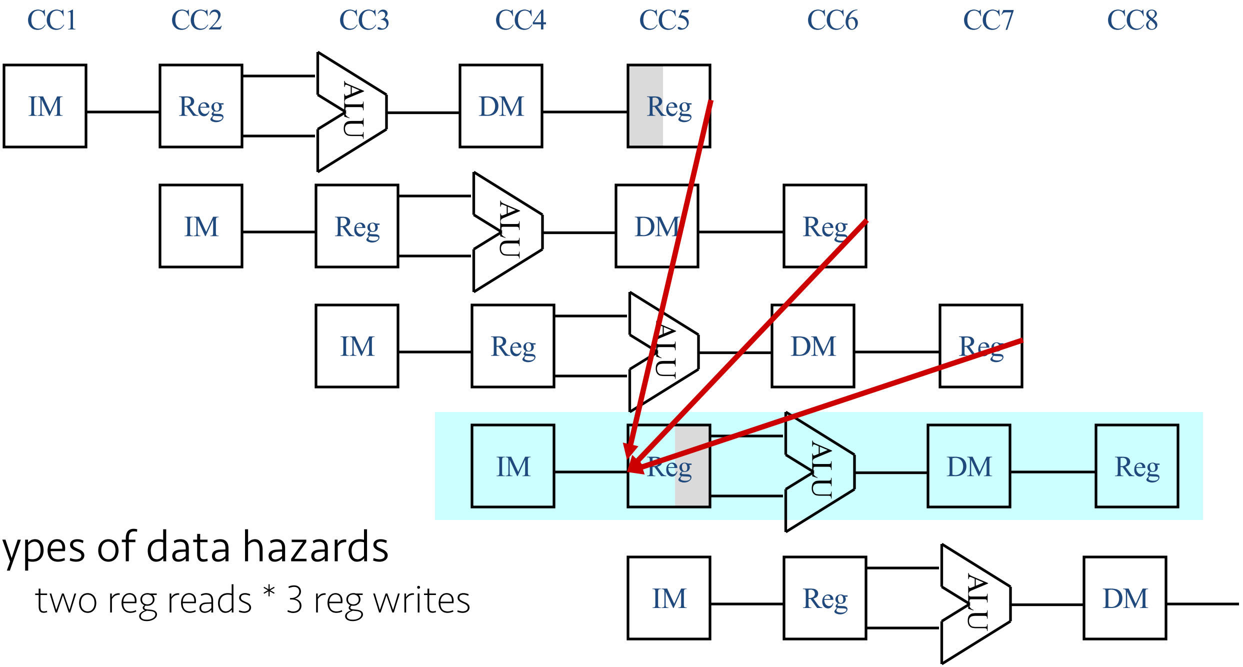


Knowing When to Stall



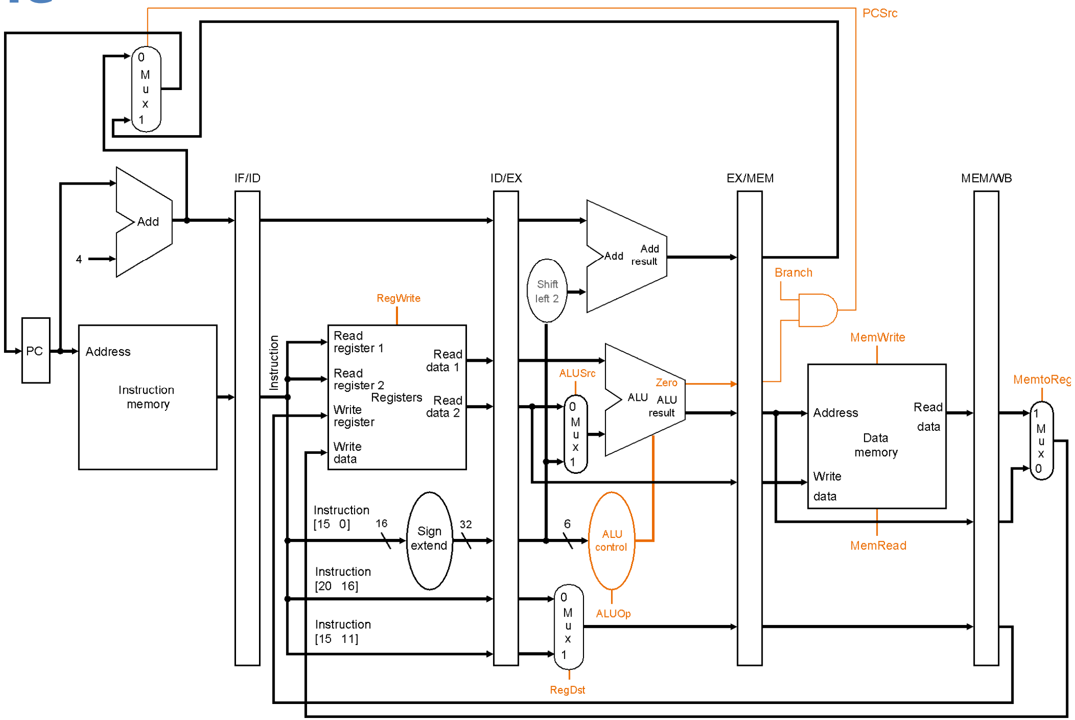
- 6 types of data hazards
  - two reg reads * 3 reg writes

Knowing When to Stall



- 6 types of data hazards
  - two reg reads * 3 reg writes

The Pipeline



- What comparisons tell us when to stall?