PIPELINE HAZARDS

Mahdi Nazm Bojnordi

Assistant Professor

School of Computing

University of Utah

Overview

- Notes
  - Homework 9 (deadline Apr. 9th)
    - Verify your submitted file before midnight

- This lecture
  - Pipeline Hazards
    - Structural
    - Data
    - Control

Recall: Pipeline Hazards

- Structural hazards: multiple instructions compete for the same resource
- Data hazards: a dependent instruction cannot proceed because it needs a value that hasn't been produced
- Control hazards: the next instruction cannot be fetched because the outcome of an earlier branch is unknown

Data Hazards

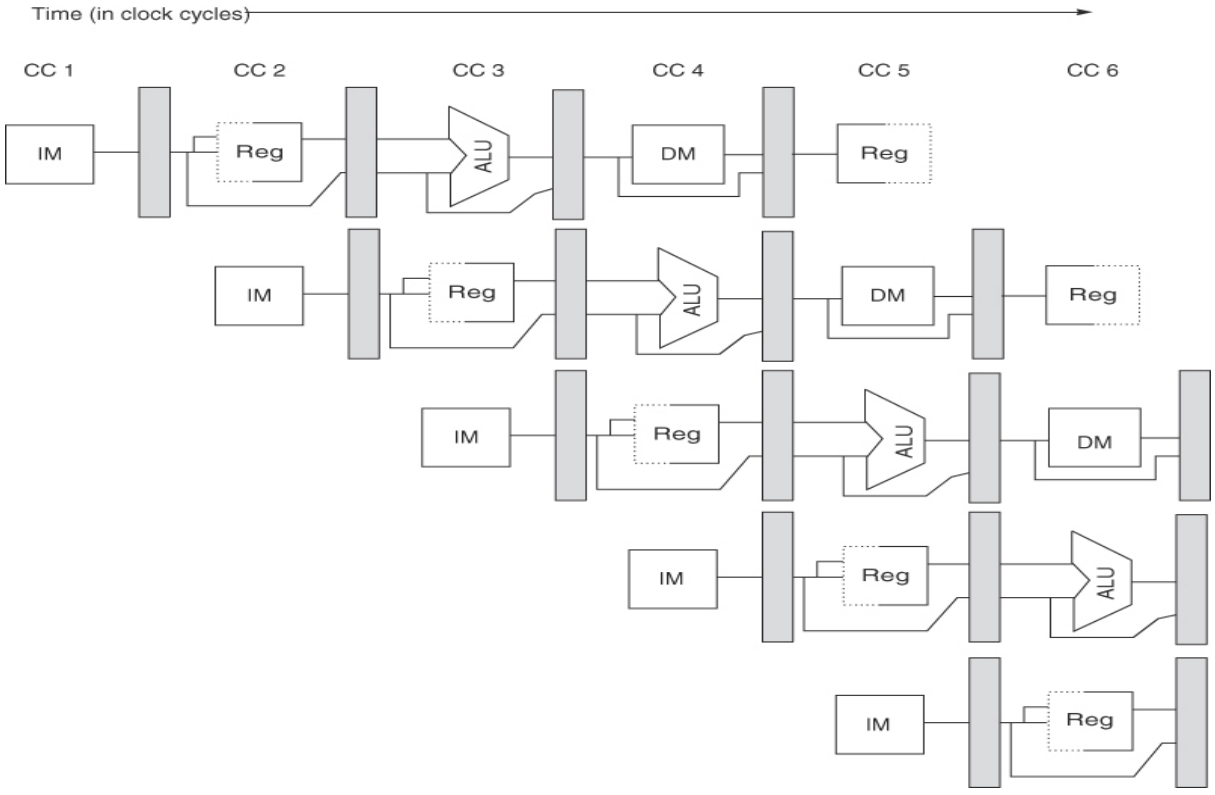
- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Loading data from memory.

lw $1, 0($2)

add $3, $1, $0

sub $4, $1, $3



Data Hazards

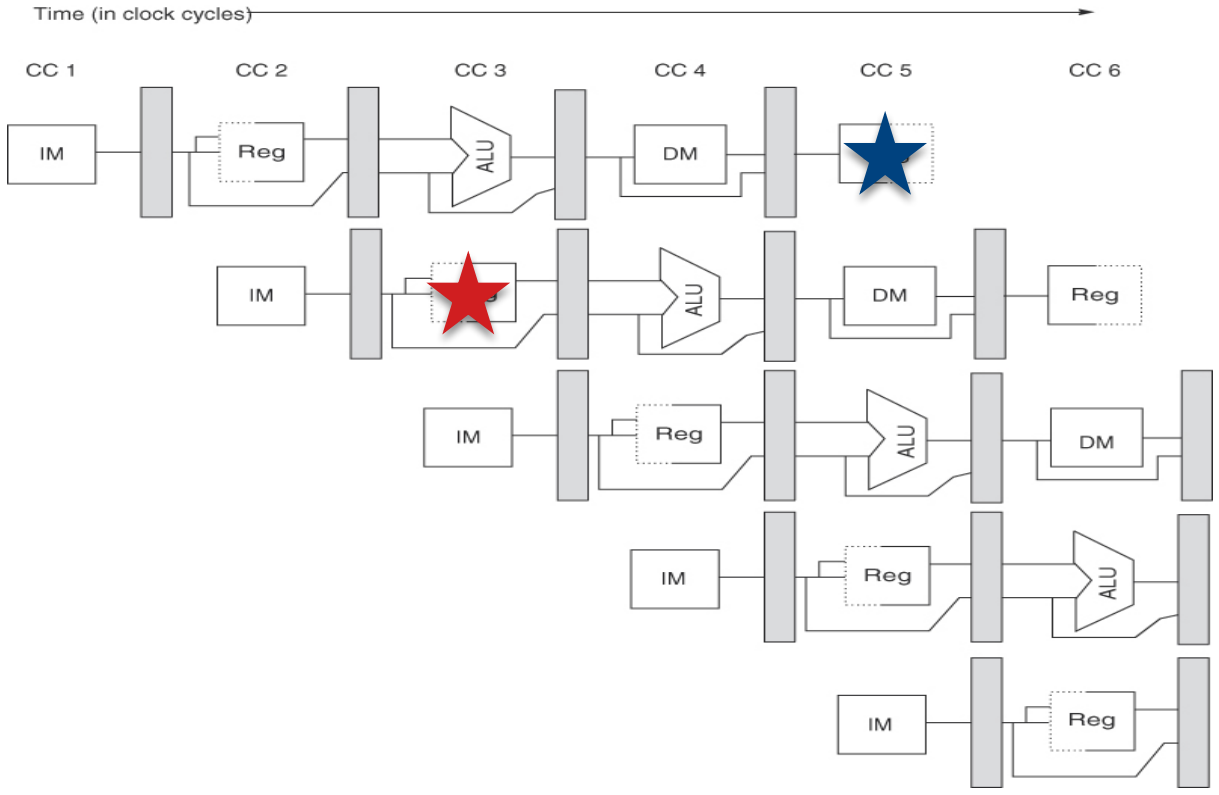
- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Loaded data will be available two cycles later.

lw $1, 0($2)

add $3, $1, $0

sub $4, $1, $3



Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

Inserting two bubbles.

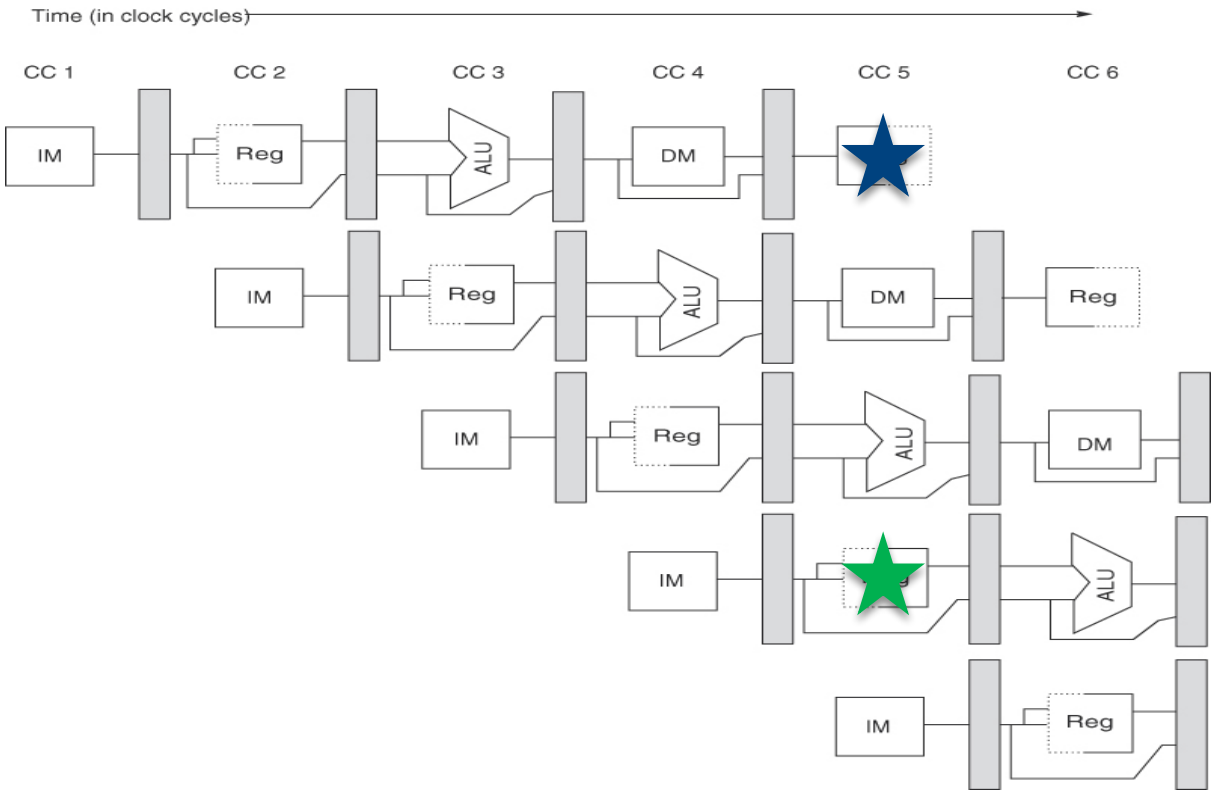
lw $1, 0($2)

Nothing

Nothing

add $3, $1, $0

sub $4, $1, $3



Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

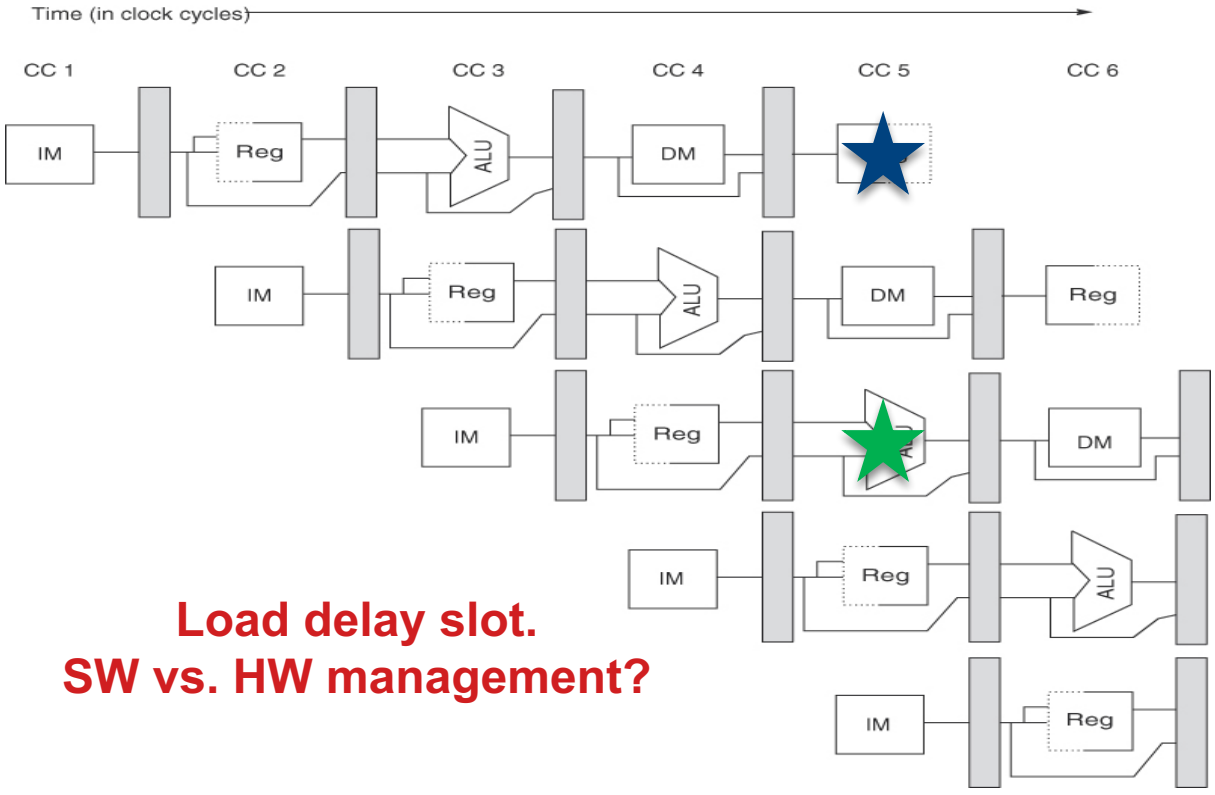
Inserting single bubble + RF bypassing.

lw $1, 0($2)

Nothing

add $3, $1, $0

sub $4, $1, $3

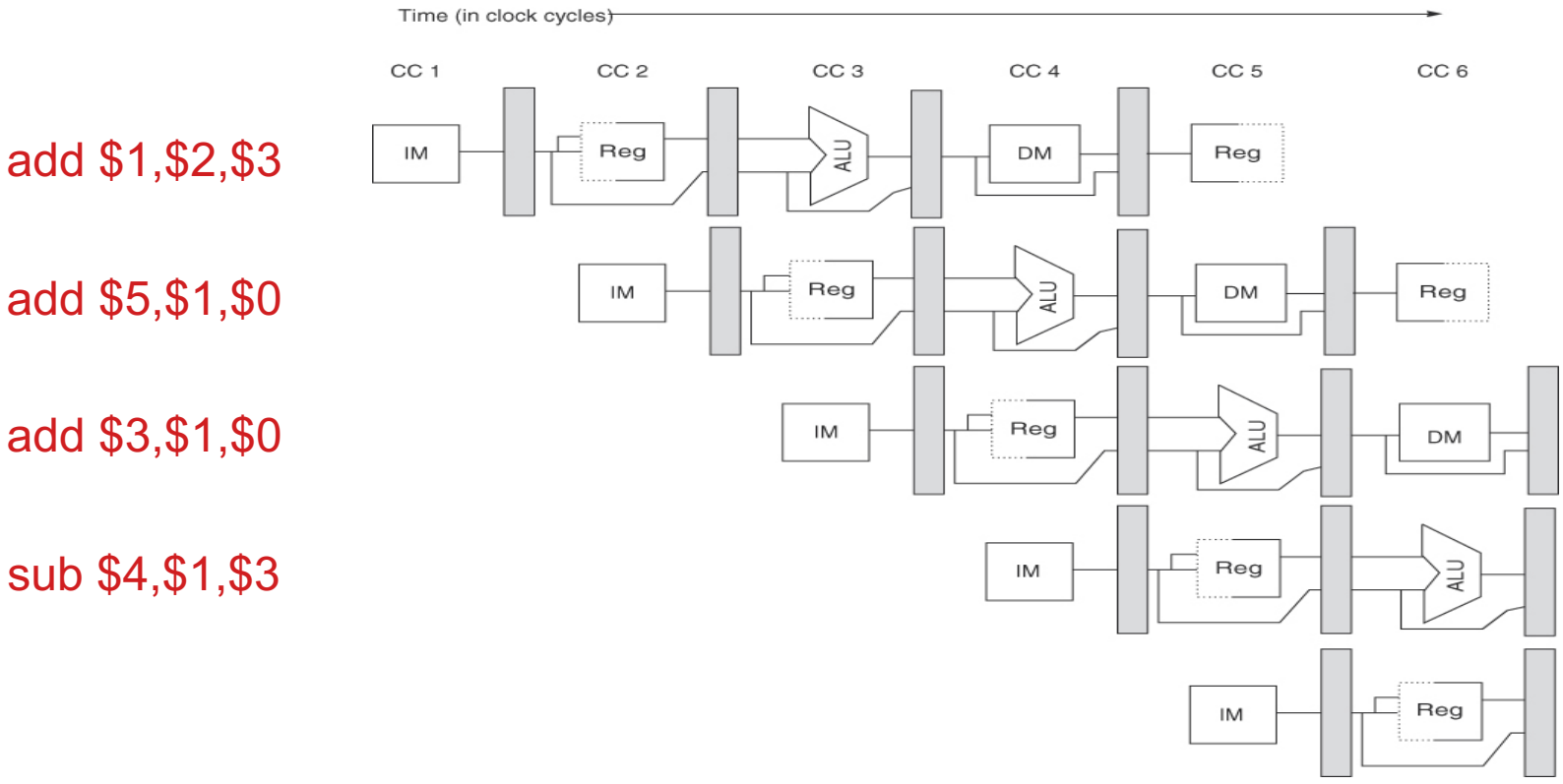


Load delay slot.
SW vs. HW management?

Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

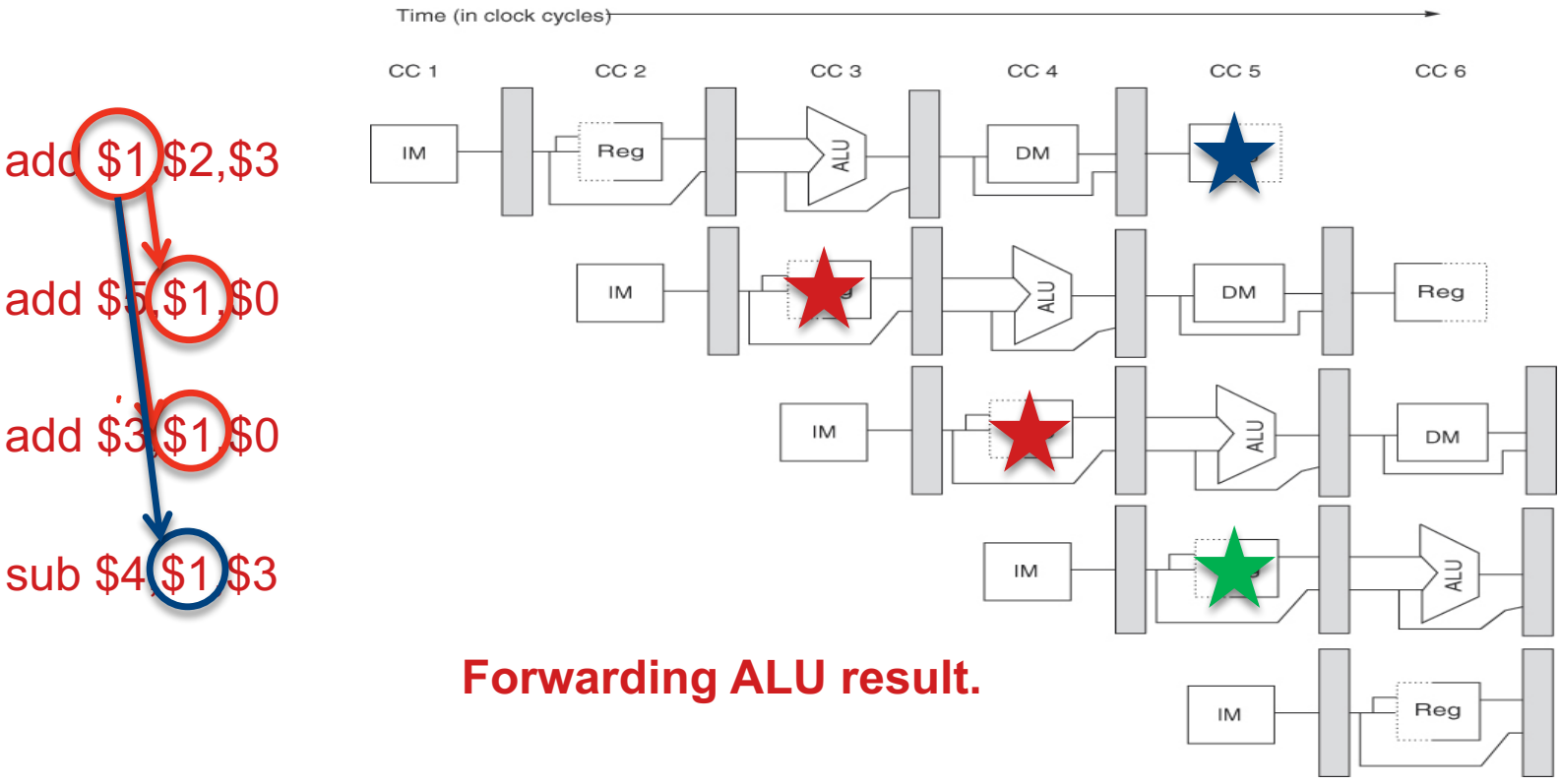
Using the result of an ALU instruction.



Data Hazards

- True dependence: read-after-write (RAW)
  - Consumer has to wait for producer

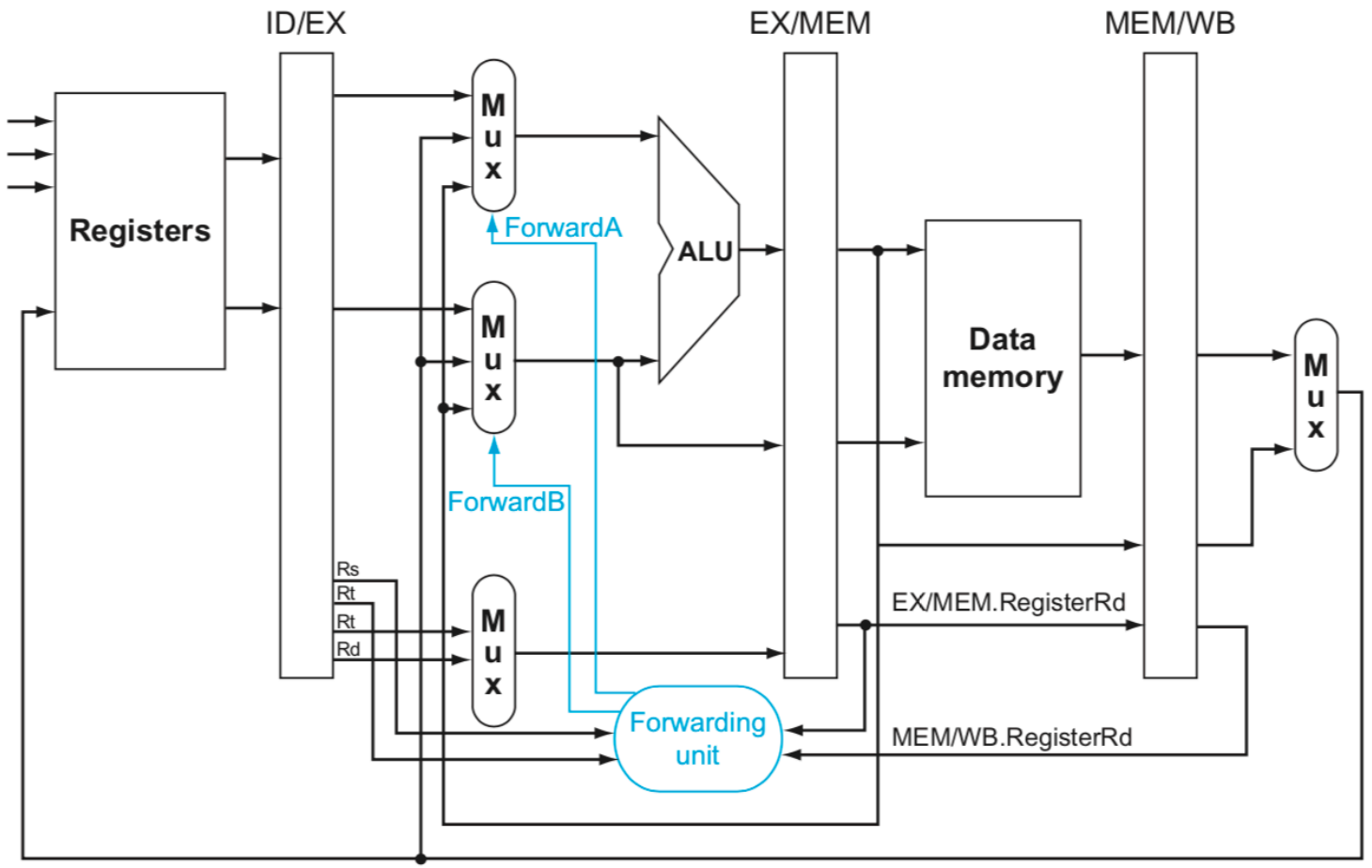
Using the result of an ALU instruction.



Forwarding ALU result.

Data Hazards

- Forwarding with additional hardware



Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below

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

add $2, $1, $0

sub $1, $1, $2

sw $2, 0($3)
```

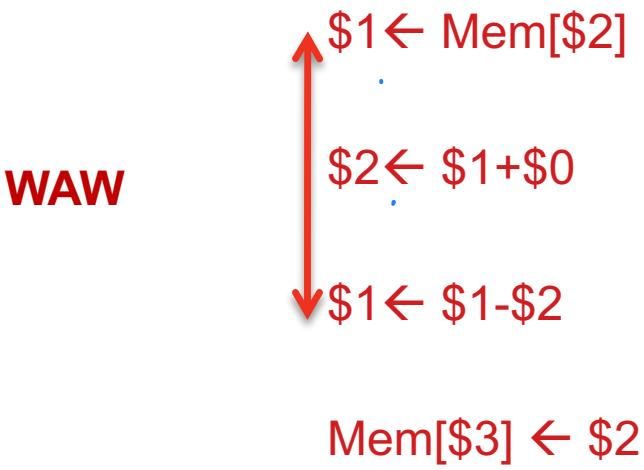
Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below

lw $1, 0($2)	$1← Mem[$2]
add $2, $1, $0	$2← $1+$0
sub $1, $1, $2	$1← $1-$2
sw $2, 0($3)	Mem[$3] ← $2

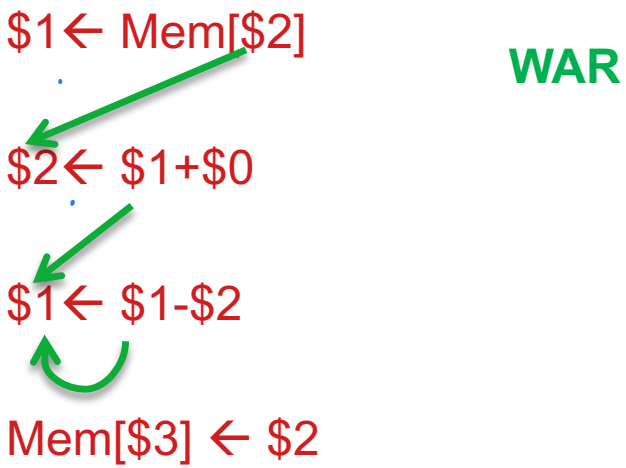
Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below



Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below



Data Hazards

- How to detect and resolve data hazards
  - Show all of the data hazards in the code below

Control Hazards

- Sample C++ code

```
for (i=100; i != 0; i--) {
    sum = sum + i;
}
total = total + sum;
```

How many branches in this code?

Control Hazards

- Sample C++ code

```
for (i=100; i != 0; i--) {
    sum = sum + i;
}
total = total + sum;
```

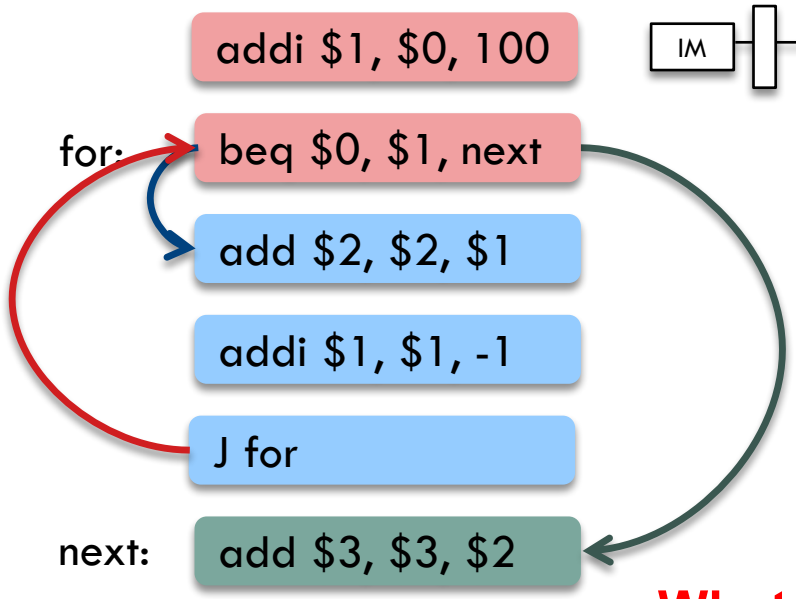
```
        addi $1, $0, 100
for:    beq $0, $1, next
        add $2, $2, $1
        addi $1, $1, -1
        J for
next:   add $3, $3, $2
```

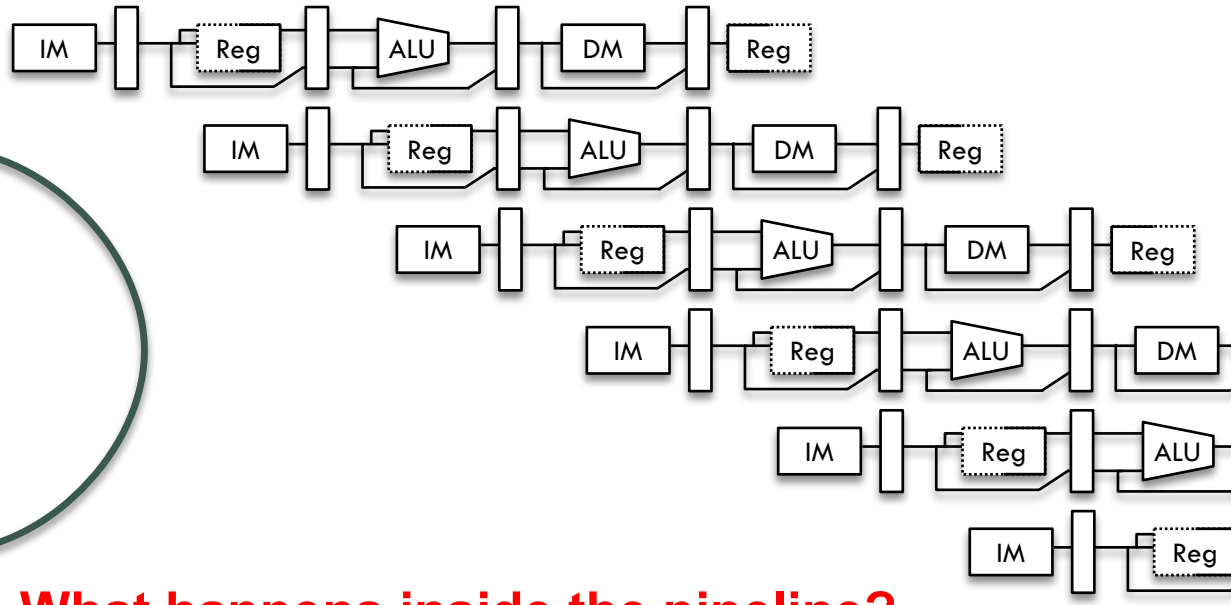
What are possible target instructions?

Control Hazards

- Sample C++ code

```
for (i=100; i != 0; i--) {
    sum = sum + i;
}
total = total + sum;
```

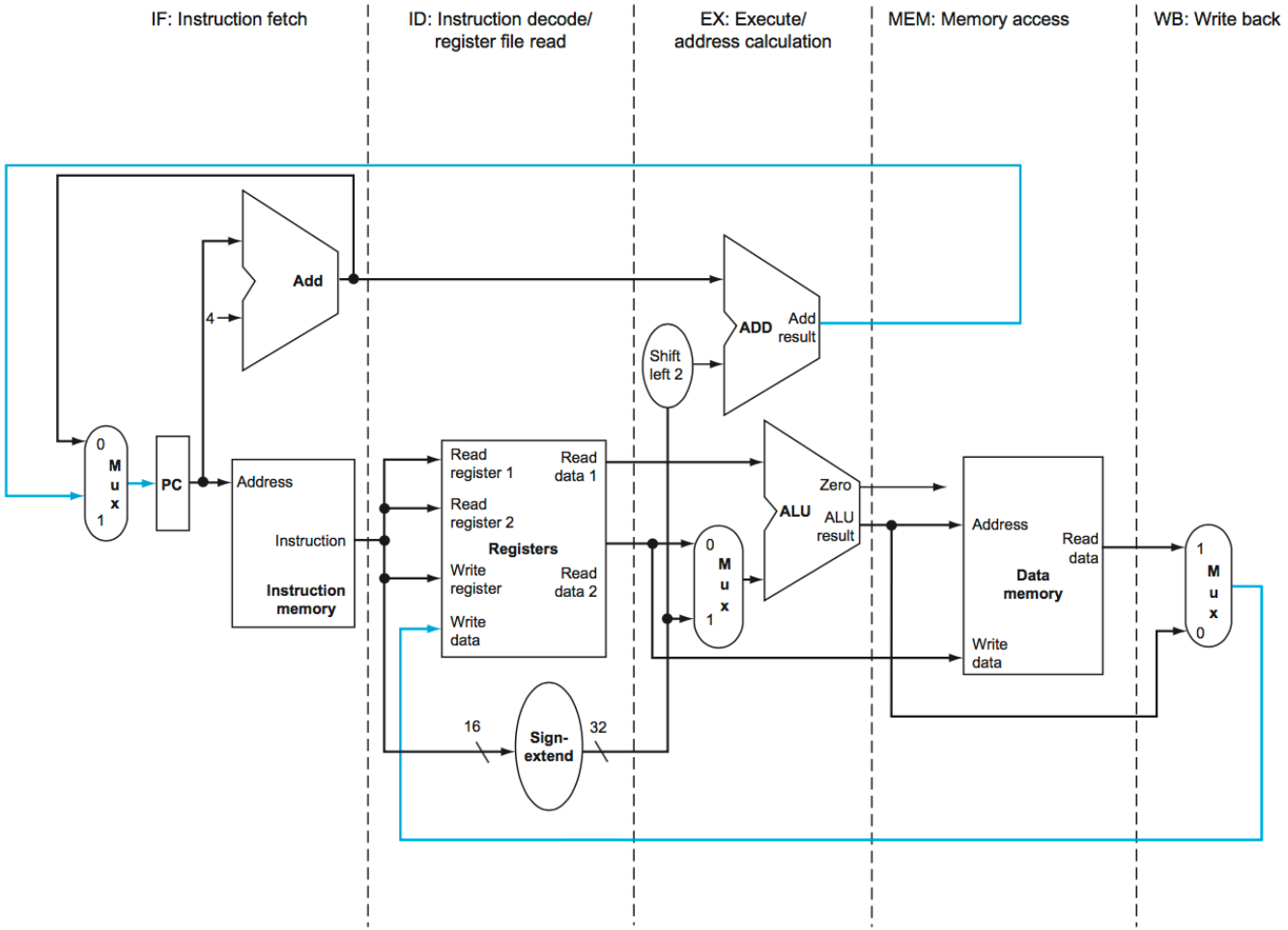




What happens inside the pipeline?

Control Hazards

- The outcome of the branch

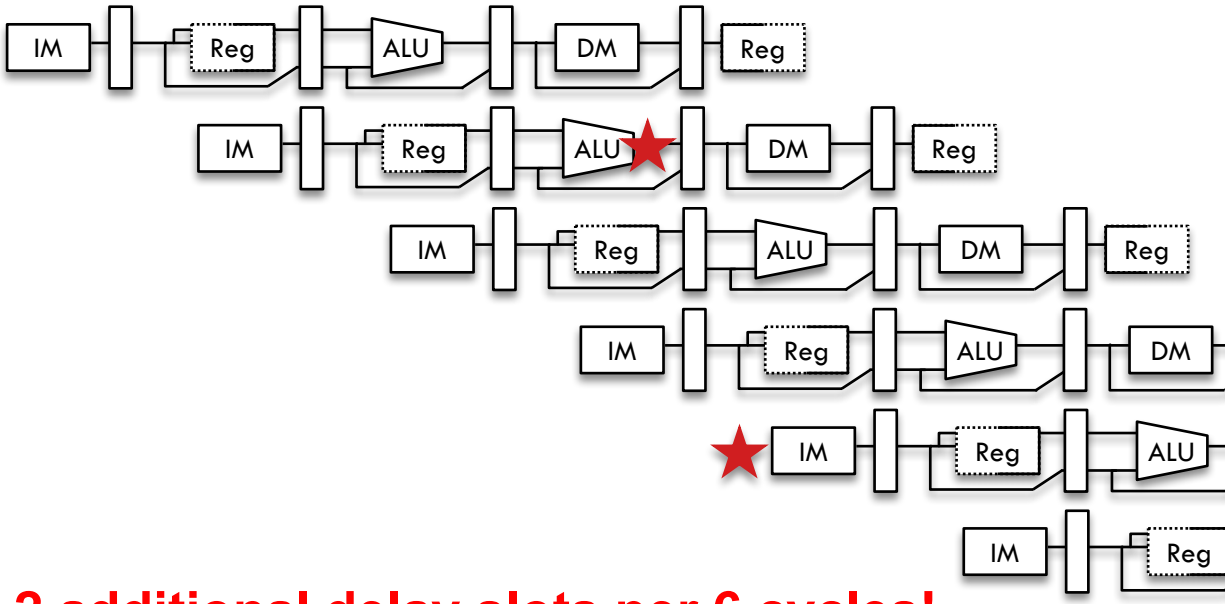


Handling Control Hazards

- 1. introducing stall cycles and delay slots
  - How many cycles/slots?
  - One branch per every six instructions on average!!

for:

```
addi $1, $0, 100
beq $0, $1, next
nothing
nothing
add $2, $2, $1
addi $1, $1, -1
J for
```



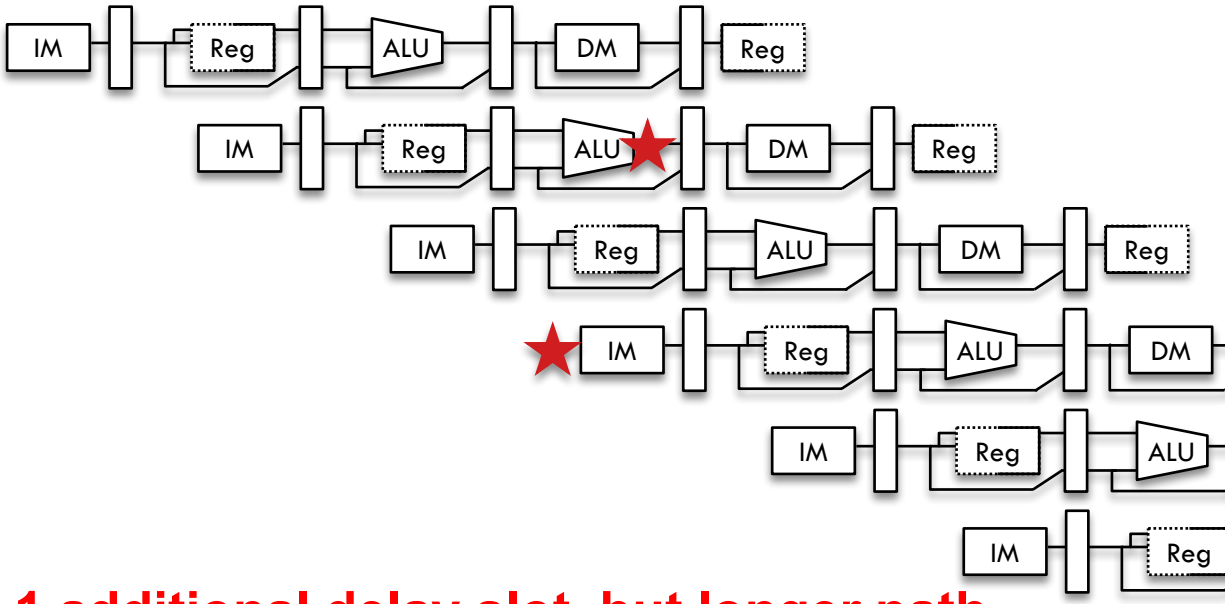
2 additional delay slots per 6 cycles!

Handling Control Hazards

- 1. introducing stall cycles and delay slots
  - How many cycles/slots?
  - One branch per every six instructions on average!!

for:

```
addi $1, $0, 100
beq $0, $1, next
nothing
add $2, $2, $1
addi $1, $1, -1
J for
nothing
```

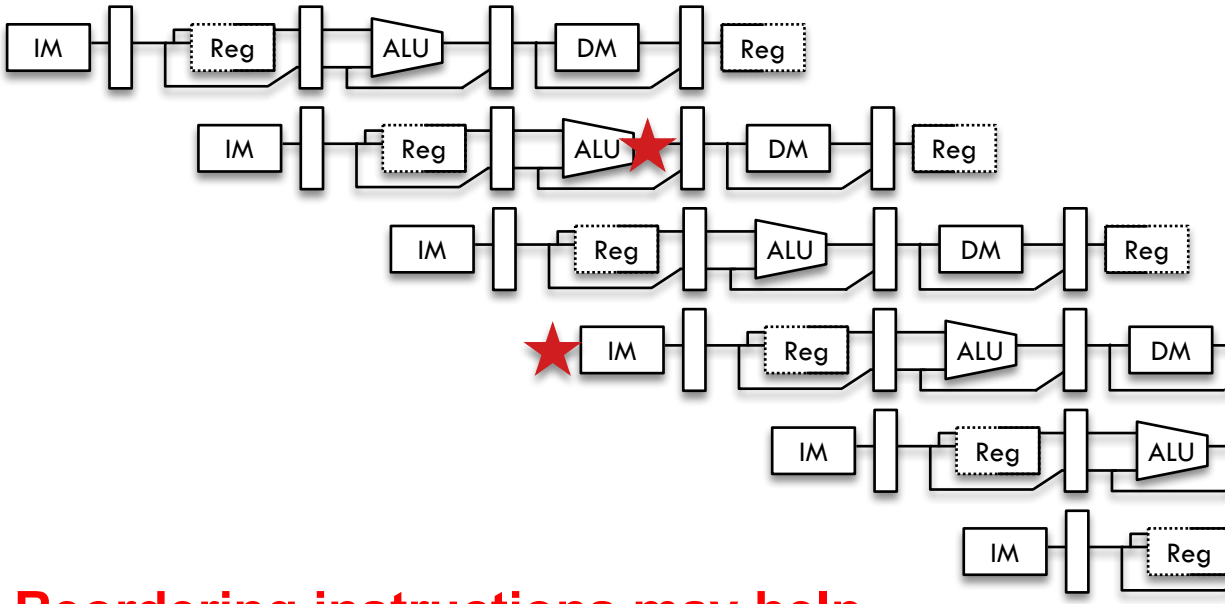


1 additional delay slot, but longer path

Handling Control Hazards

- 1. introducing stall cycles and delay slots
  - How many cycles/slots?
  - One branch per every six instructions on average!!

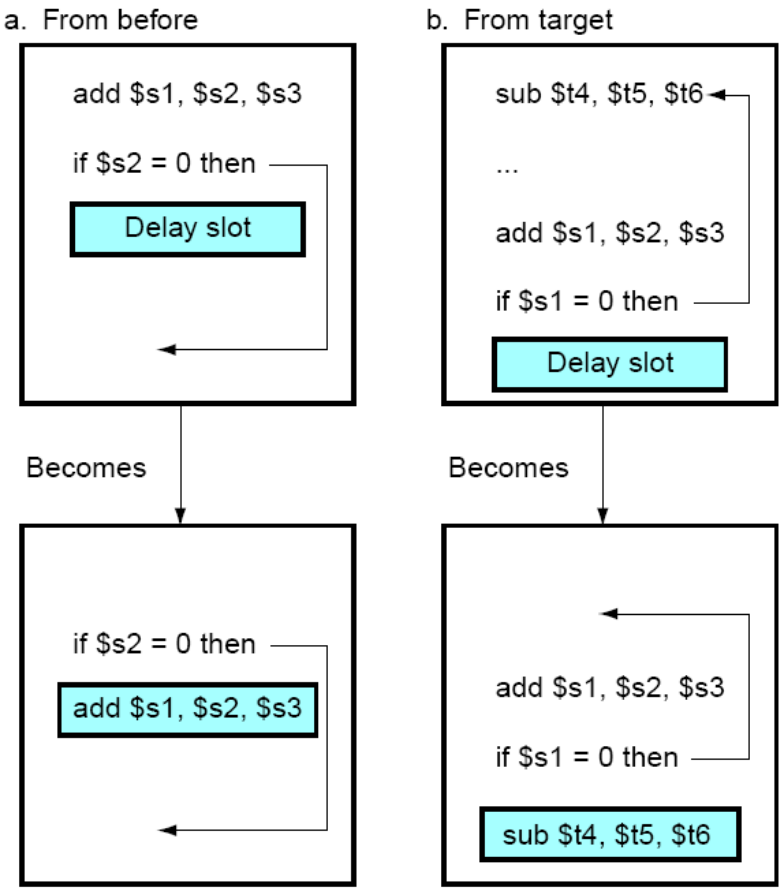
```
        addi $1, $0, 100
for:    beq $0, $1, next
        nothing
        add $2, $2, $1
        J for
        addi $1, $1, -1
next:   add r3, r3, r2
```



Reordering instructions may help

Handling Control Hazards

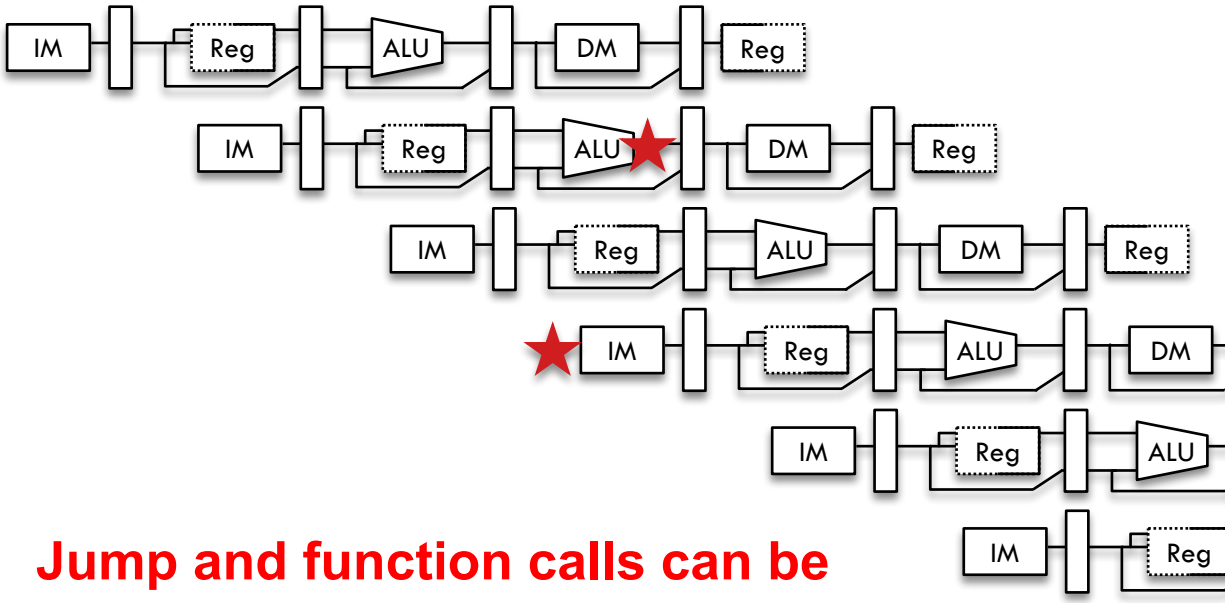
- Branch delay slot



Handling Control Hazards

- 1. introducing stall cycles and delay slots
  - How many cycles/slots?
  - One branch per every six instructions on average!!

```
        addi $1, $0, 100
for:    beq $0, $1, next
        nothing
        add $2, $2, $1
        J for
        addi $1, $1, -1
next:   add r3, r3, r2
```

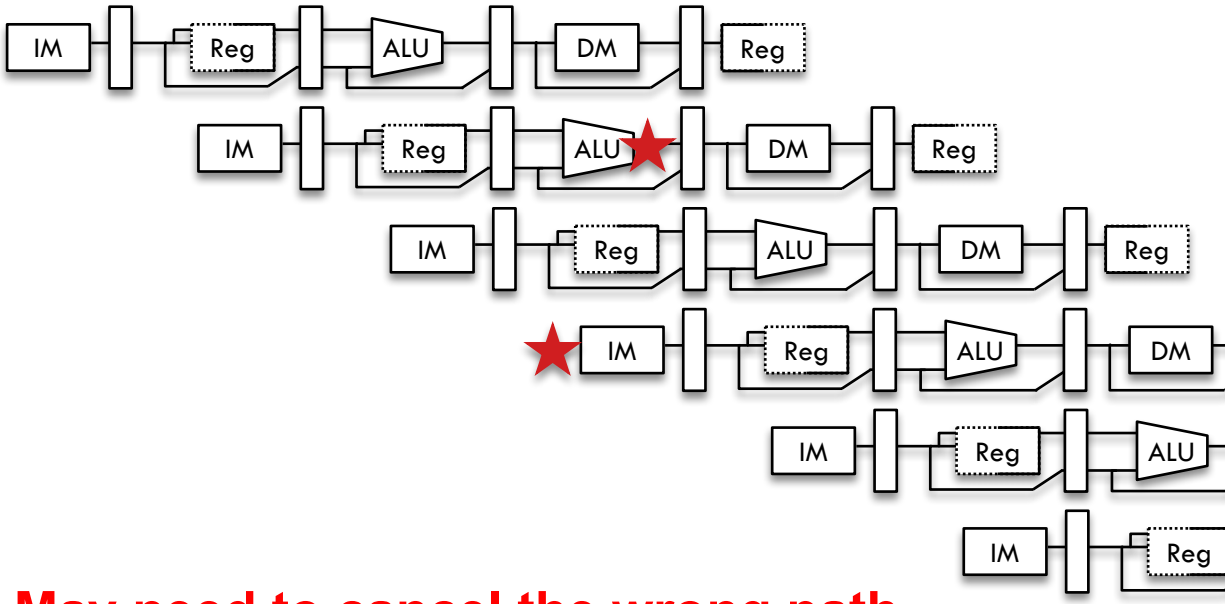


Jump and function calls can be resolved in the decode stage.

Handling Control Hazards

- 1. introducing stall cycles and delay slots
- 2. predict the branch outcome
  - simply assume the branch is taken or not taken
  - predict the next PC

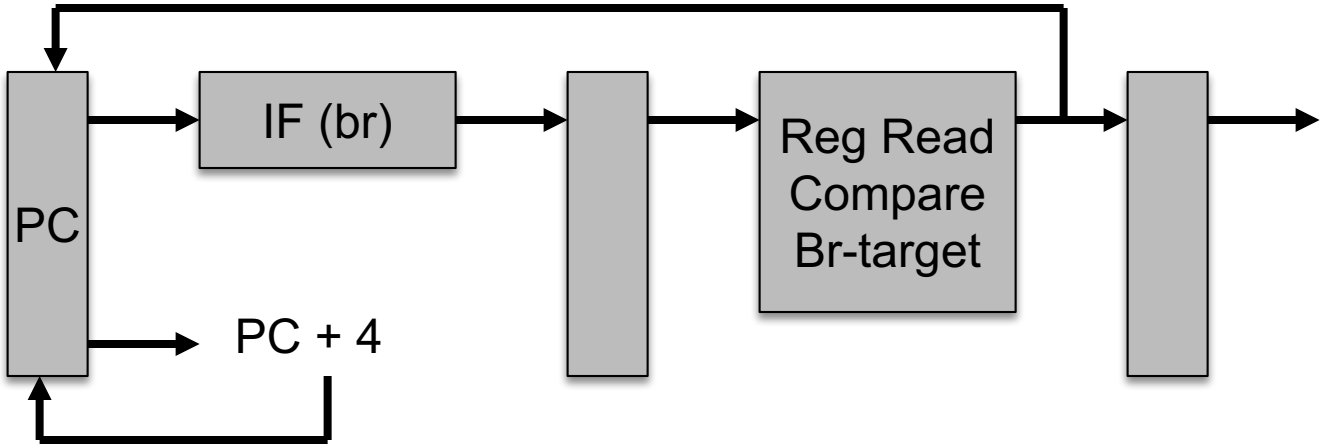
```
        addi $1, $0, 100
for:    beq $0, $1, next
        add $2, $2, $1
        addi $1, $1, -1
        J for
next:   add r3, r3, r2
```



May need to cancel the wrong path

Handling Control Hazards

- Pipeline without branch predictor



Handling Control Hazards

- Pipeline with branch predictor