Pipelined Architecture

Virendra Singh
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

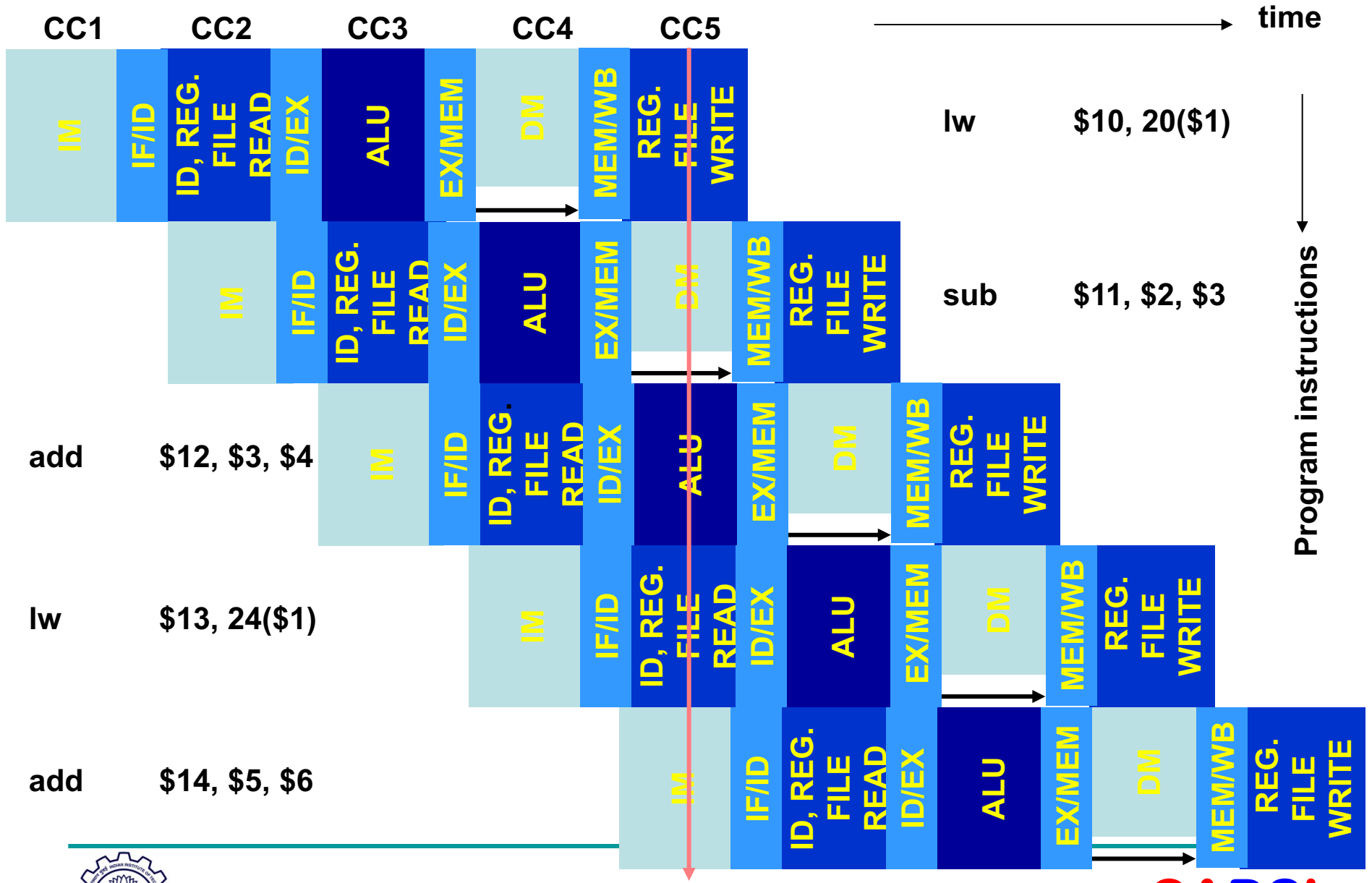
Lecture 7 (12 Aug 2021)

CADSL

Advantages of Pipeline

- After the fifth cycle (CC5), one instruction is completed each cycle; CPI ≈ 1, neglecting the initial pipeline latency of 5 cycles.
  - *Pipeline latency is defined as the number of stages in the pipeline, or*
  - *The number of clock cycles after which the first instruction is completed.*
- The clock cycle time is about four times shorter than that of single-cycle datapath and about the same as that of multicycle datapath.
- For multicycle datapath, CPI = 3. ….
- So, pipelined execution is faster, but . . .

Program Execution



Pipeline Hazards

- Definition: *Hazard in a pipeline is a situation in which the next instruction cannot complete execution one clock cycle after completion of the present instruction.*
- Three types of hazards:
  - Structural hazard (resource conflict)
  - Data hazard
  - Control hazard

- detect
- correct

UNIFIED PIPELINES

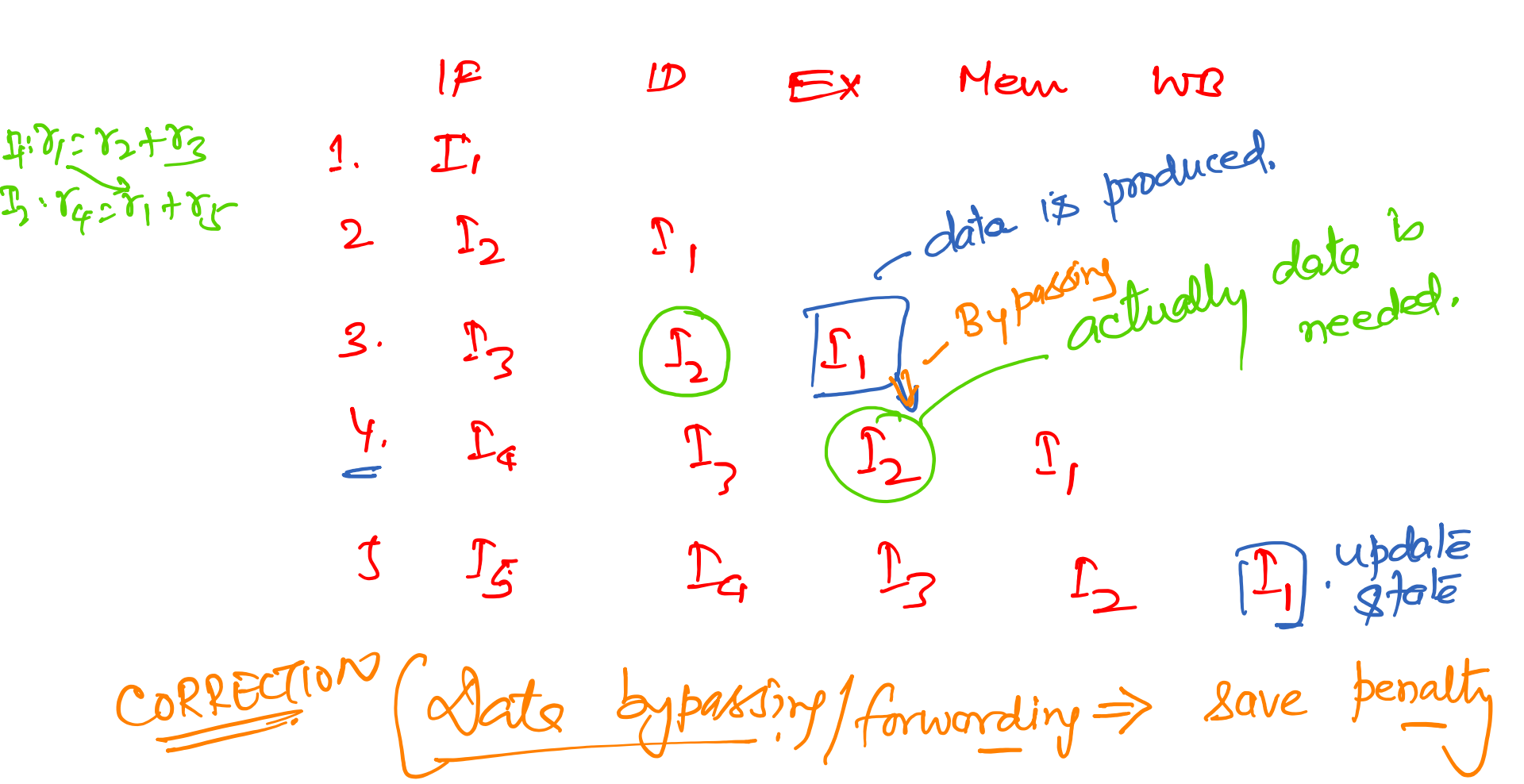
- fetch one instruction per cycle
- all instructions have to pass through all the pipeline stages

Data Hazard

- Data hazard means that an instruction cannot be completed because the needed data, to be generated by another instruction in the pipeline, is not available.
- Example: consider two instructions:
  - add $s0, $t0, $t1
  - sub $t2, $s0, $t3 # needs $s0

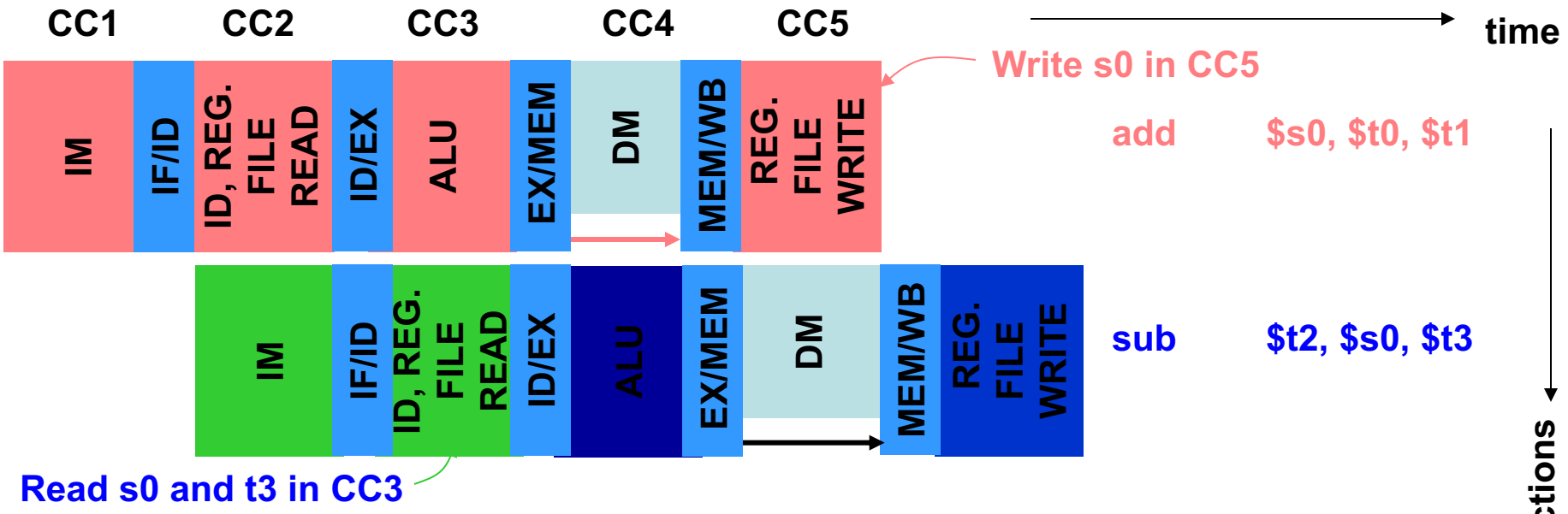
10% STALL – 3 cycle] penalty

CPI = 1 + .1 x 3 = 1.3



CORRECTION [Data bypassing/forwarding ⇒ save penalty]

Example of Data Hazard



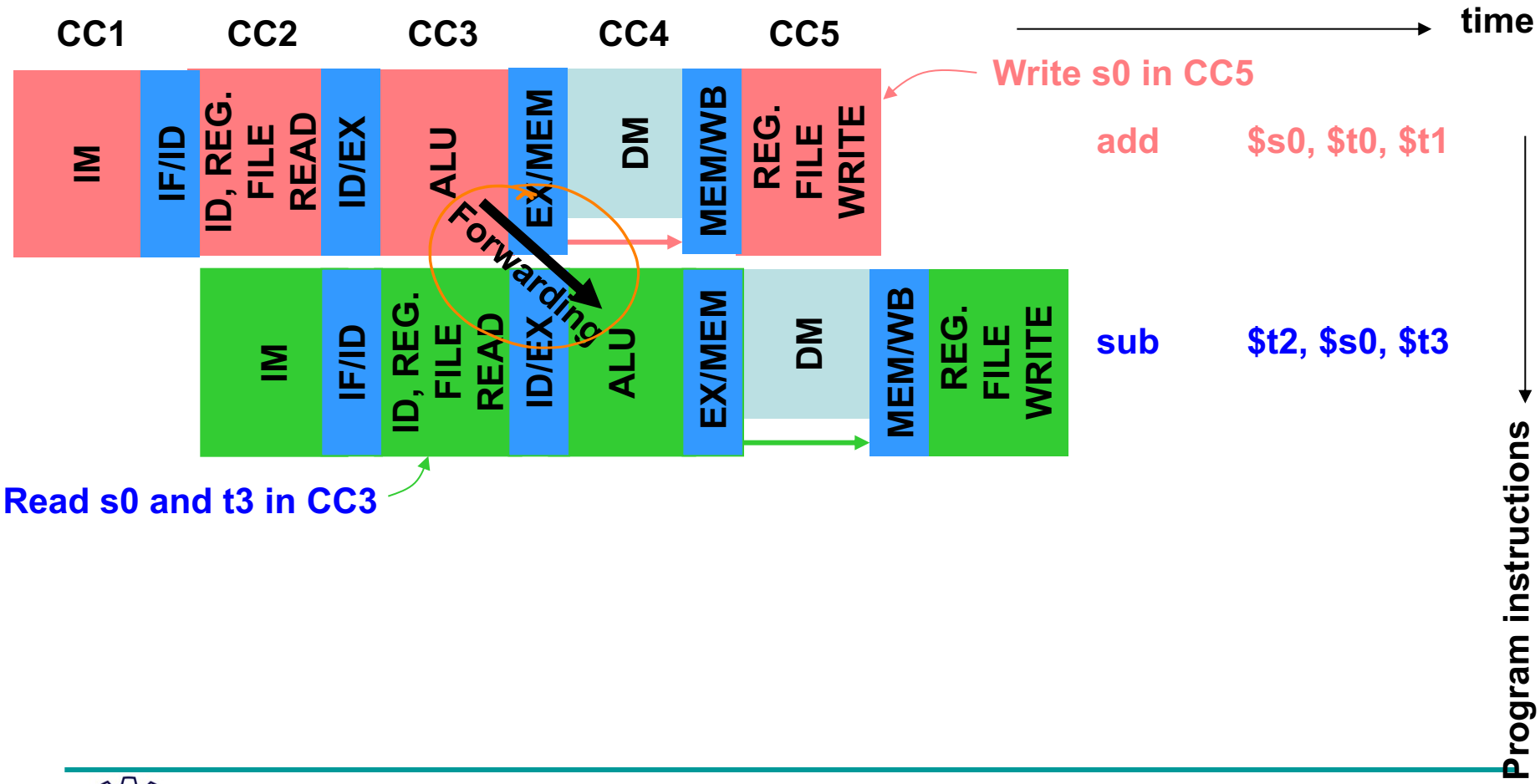
We need to read s0 from reg file in cycle 3
But s0 will not be written in reg file until cycle 5

However, s0 will only be used in cycle 4
And it is available at the end of cycle 3

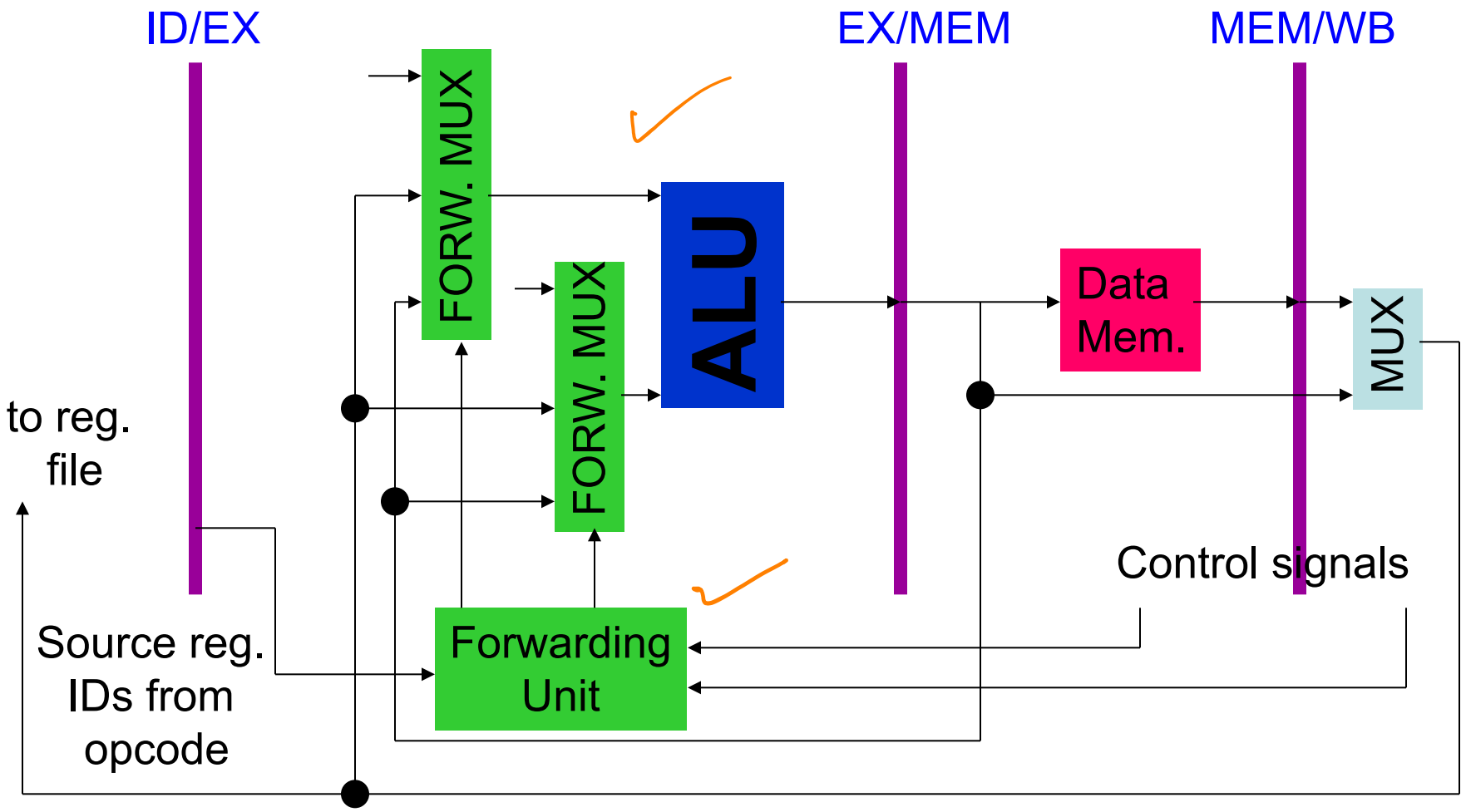
Forwarding or Bypassing

- Output of a resource used by an instruction is forwarded to the input of some resource being used by another instruction.
- Forwarding can eliminate some, but not all, data hazards.

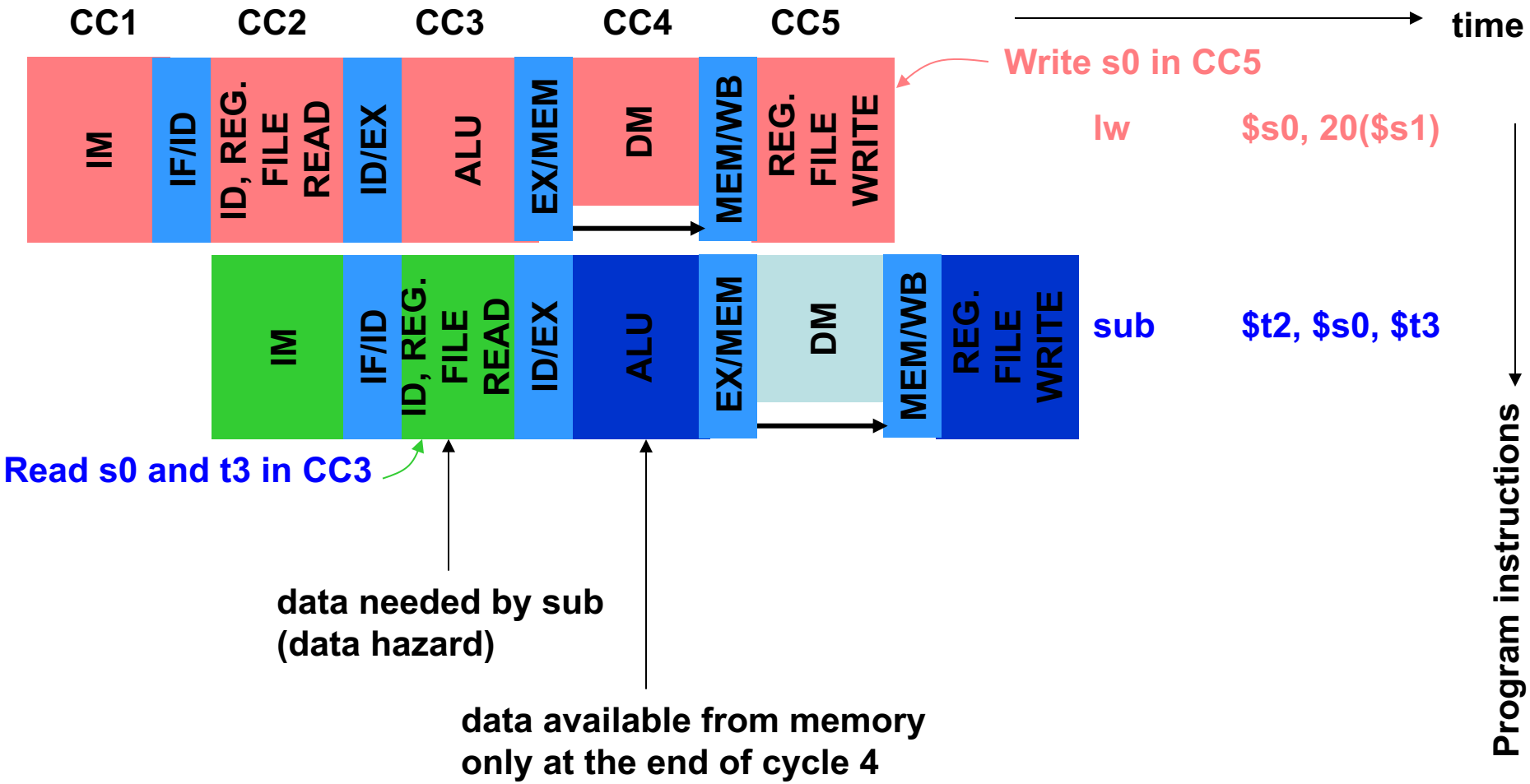
Forwarding for Data Hazard



Forwarding Unit Hardware



Forwarding Alone May Not Work



I_1 $r_1 \leftarrow a$

I_2 $r_3 = r_1 + r_2$

	IF	ID	Ex	Mem	WB
1.	I_1				
2.	I_2	I_1			
3.	I_3	I_2	I_1		
4	I_4	I_3	I_2	I_1	
5	I_4	I_3	I_2	–	I_1
6.	I_5	I_4	I_3	I_2	–
7					

data is needed.

data is produced.

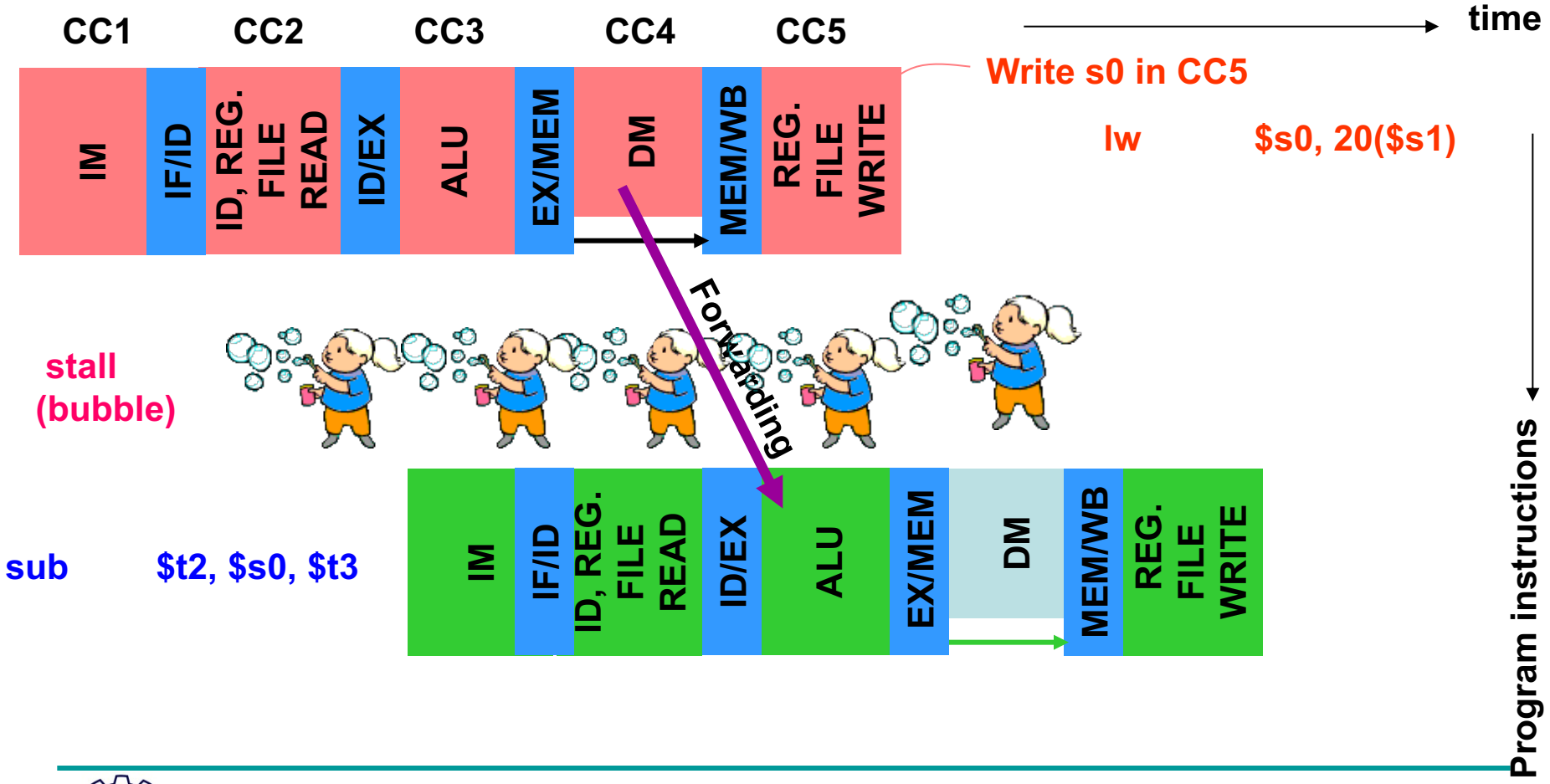
STALL for one cycle.

Is it the bottleneck?

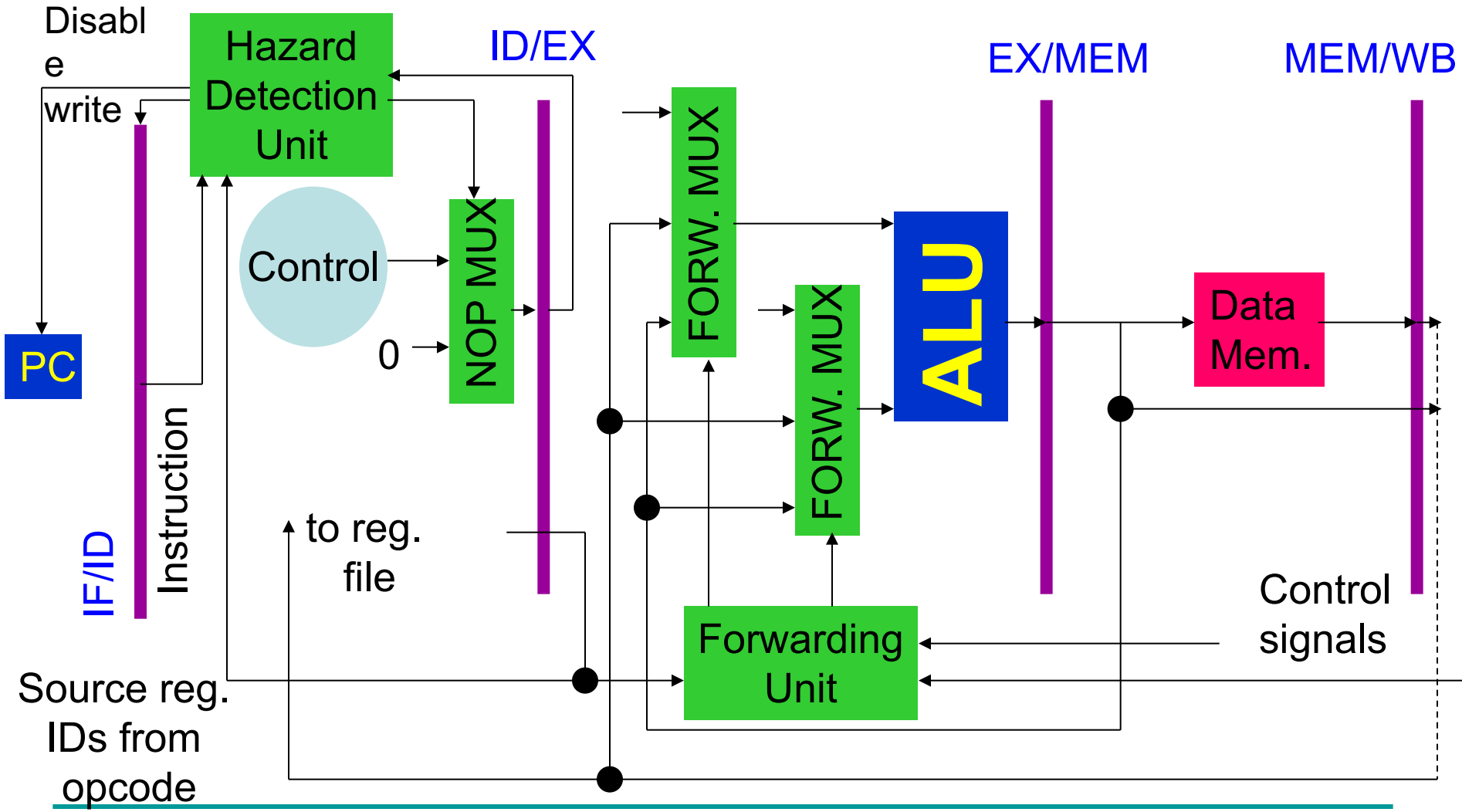
CPI: $1 + .3 \times .4 \times 1$

$= 1 + .12 = 1.12$

Use Bubble and Forwarding



Hazard Detection Unit Hardware



Resolving Hazards

- Hazards are resolved by Hazard detection and forwarding units.
- Compiler's understanding of how these units work can improve performance.

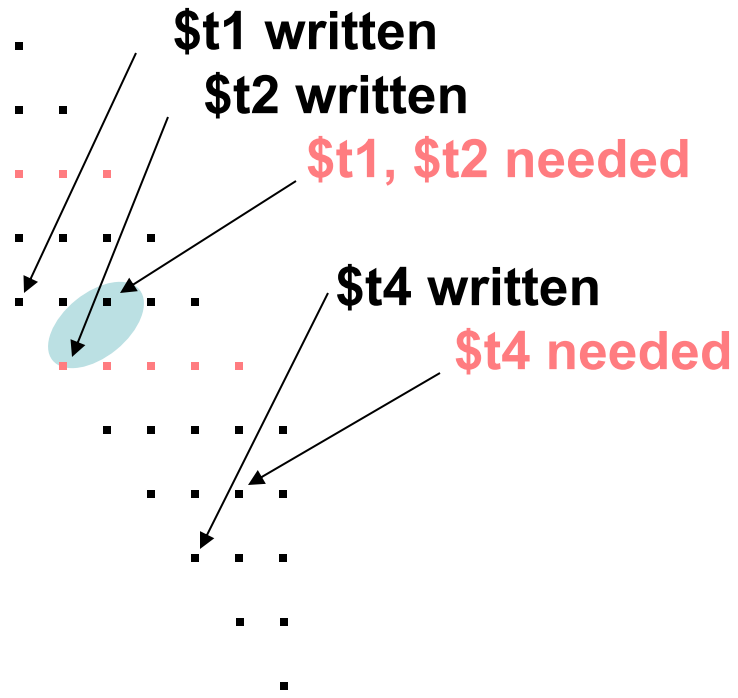
Avoiding Stall by Code Reorder

C code:

```
A = B + E;
C = B + F;
```

MIPS code:

```
lw    $t1,  0($t0)
lw    $t2,  4($t0)
add   $t3,  $t1, $t2
sw    $t3,  12($t0)
lw    $t4,  8($t0)
add   $t5,  $t1, $t4
sw    $t5,  16,($t0)
```



Reordered Code

- DATA DEPENDENCY

* Detect
* Correct

Stall | forwarding | Reorder

CORRECTNESS

C code:

```
A = B + E;
C = B + F;
```

MIPS code:

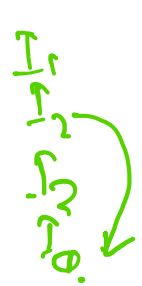
```
lw    $t1,   0($t0)
lw    $t2    4($t0)
lw    $t4,   8($t0)
add   $t3,   $t1, $t2      no hazard
sw    $t3,   12($t0)
add   $t5,   $t1, $t4      no hazard
sw    $t5,   16,($t0)
```

Reduce the stalls due to dependency

Control Hazard

- Instruction to be fetched is not known!
- Example: Instruction being executed is branch-type, which will determine the next instruction:

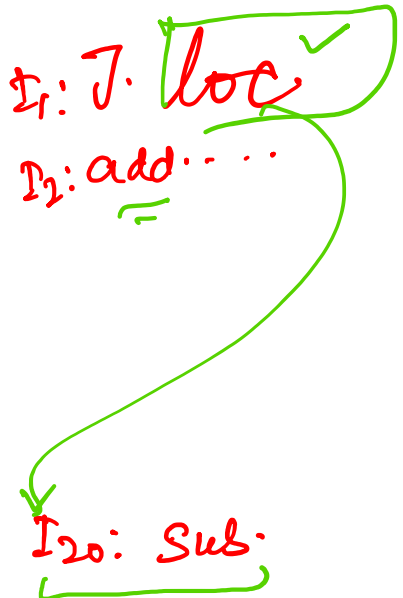
```
    add    $4, $5, $6
    beq    $1, $2, 40
    next instruction
    . . .
40and    $7, $8, $9
```



- detect
- correct.

Branch ⇄ Conditional.
Unconditional ✓

$1 == $2



unconditional.

every branch

⇒ 1 cycle ≤ STALL

↓

penalty

IF ID Ex Mem WB

1. I_1

2. ~~I_2~~ I_1 ← detect a branch.

3. ~~I_3~~ I_{20} ~~I_2~~ I_1

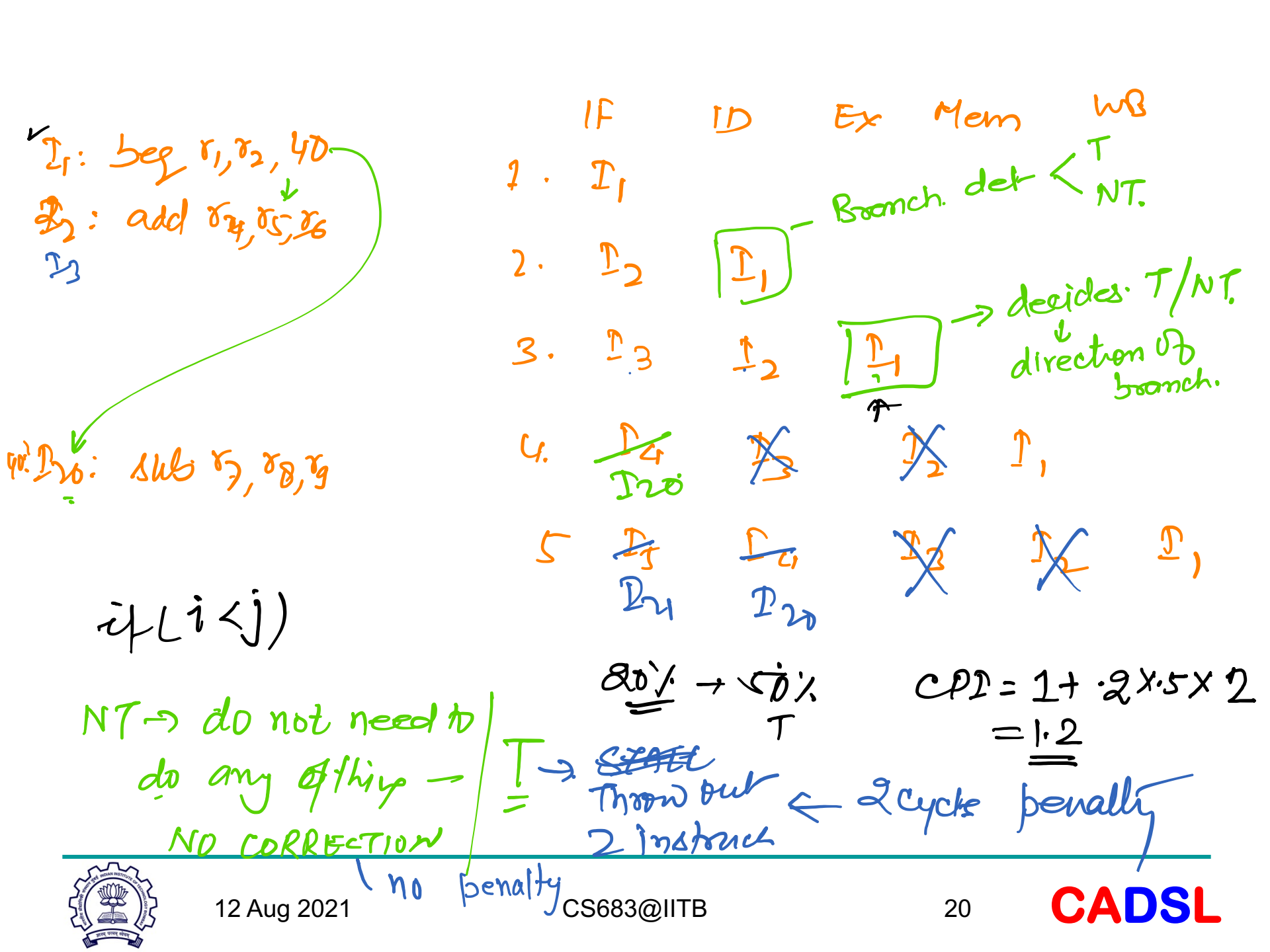
4. ~~I_4~~ I_{21} ~~I_3~~ I_{20} ~~I_2~~ I_1

5. ~~I_5~~ I_{22} ~~I_4~~ I_{21} ~~I_3~~ I_{20} ~~I_2~~ I_1.

20%.

CPI: 1 + .2 × 2 × 1

CPI. = 1.4

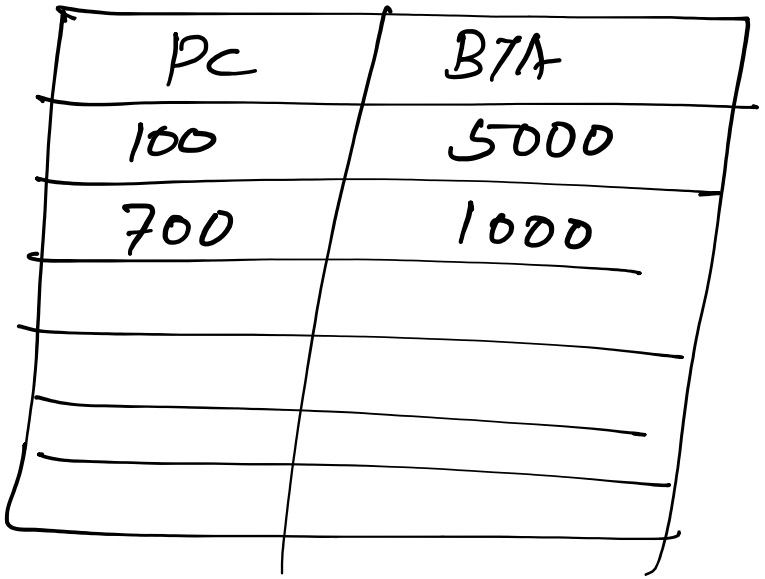

I1: beq r1, r2, 40
I2: add r4, r5, r6
I3
I10: sub r7, r8, r9
IF ID Ex Mem WB
1. I1
2. I2 I1
3. I3 I2 I1
Branch det < T / NT.
decides T/NT.
direction of branch.
4. I4 I3 I2 I1
I10
5 I5 I4 I3 I2 I1
I11 I10
if (i<j)
80% → 60%
T
CPI = 1 + .2 x .5 x 2
= 1.2
NT → do not need to do any thing — NO CORRECTION
no penalty
T → Throw out 2 instruct
2 cycle penalty

How to reduce

→ Static → Branch. 70% T & 30% NT

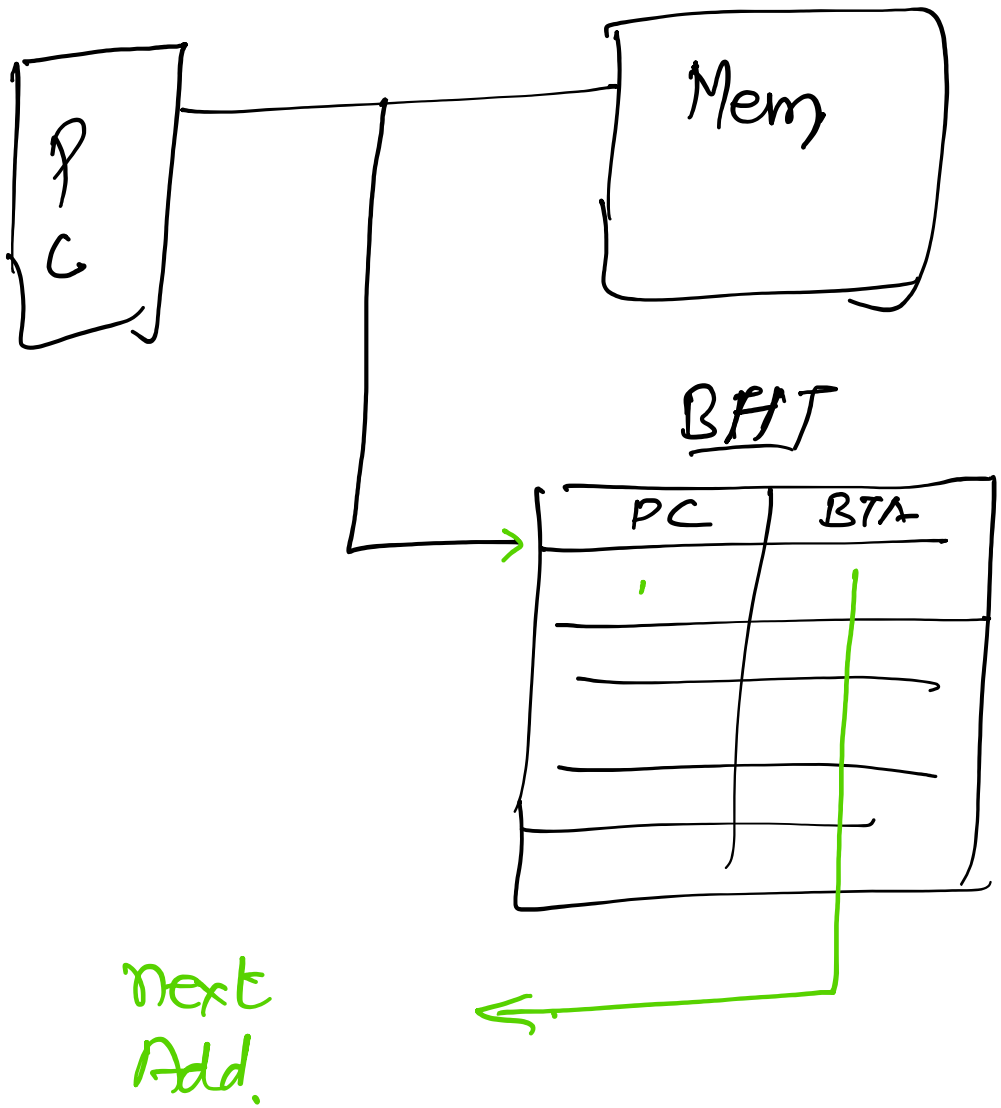
↓ Address

Branch History Table



Fetch.

PC Add


P
C
Mem
BHT
PC
BTA
Next
Add.

I_1 : beq. $r_1 r_2$ 40

I_2 : add r_3, r_2, r_4

I_{20}:

IF ID Ex MEm WB

1. I_1

2. I_{20} I_1

3 I_{21} I_{20} I_1

4 I_{22} I_{21} I_{20} I_1

Validated

15%

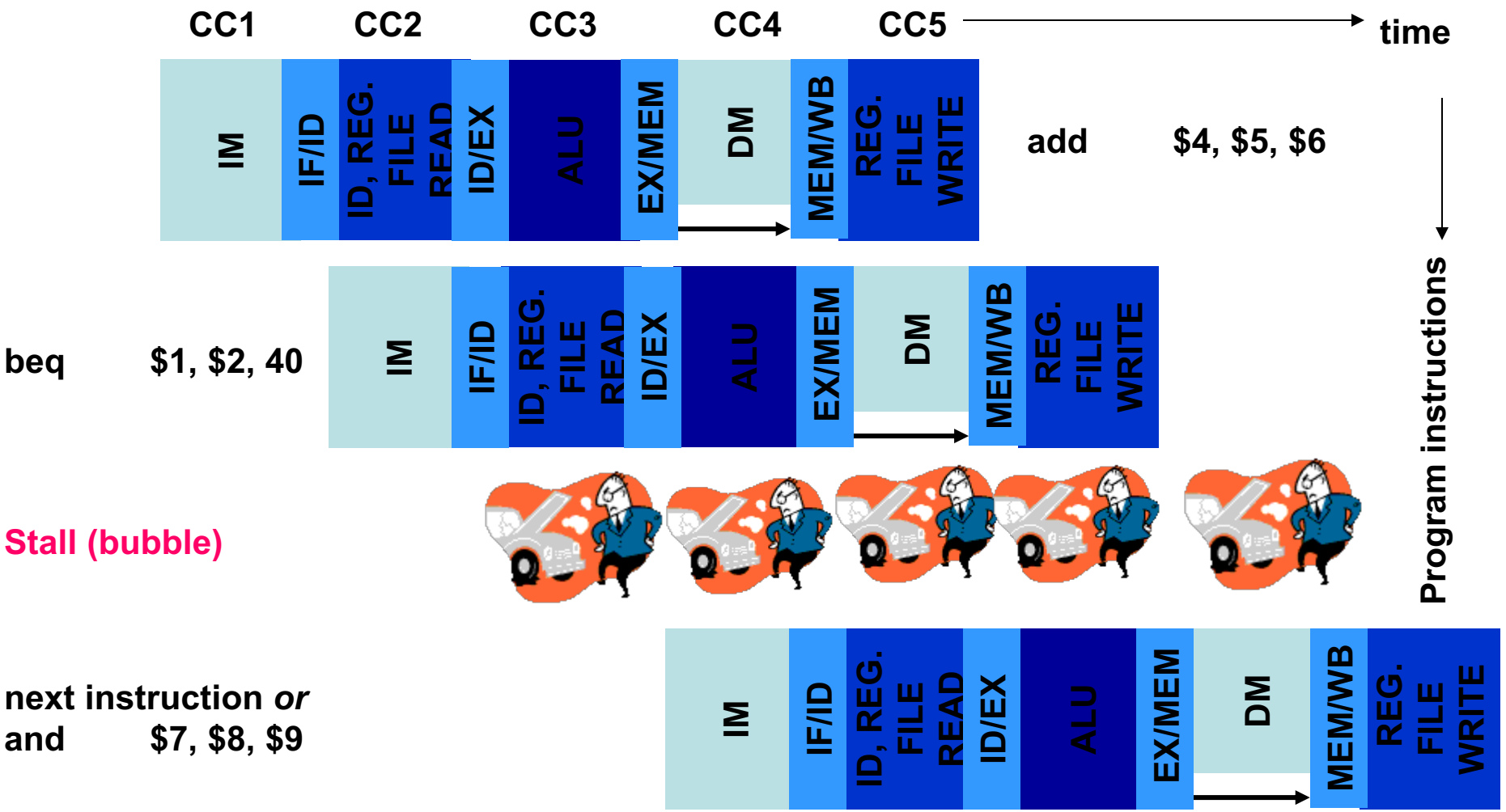
70%

CPI = 1 + .15 x 3 x 2

= 1 + .09 = 1.09

SAFE

Stall on Branch



Why Only One Stall?

- Extra hardware in ID phase:
  - Additional ALU to compute branch address
  - Comparator to generate zero signal
  - Hazard detection unit writes the branch address in PC

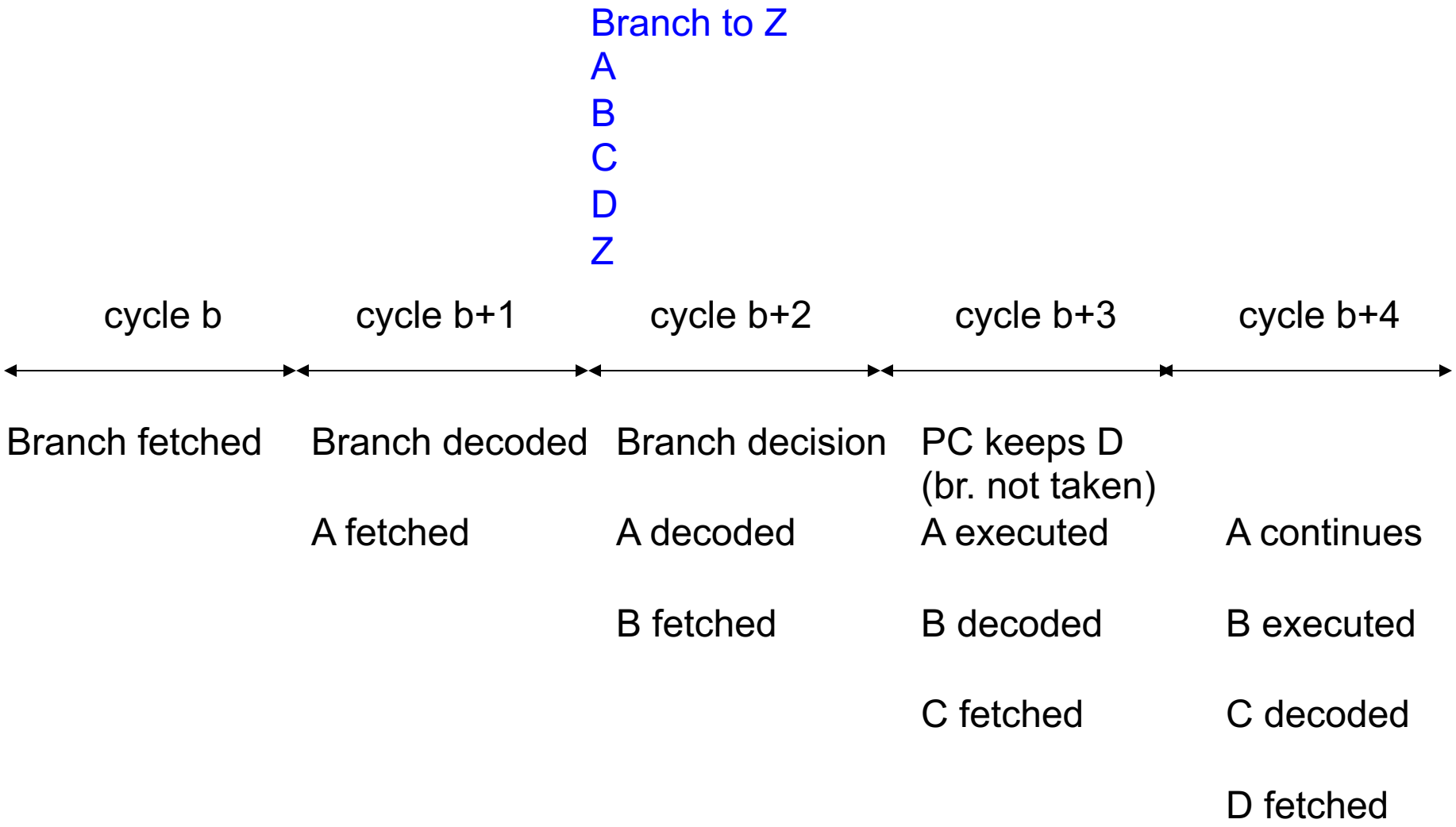
Ways to Handle Branch

- Stall or bubble
- Delayed branch
- Branch prediction:
  - Heuristics
    - Next instruction
    - Prediction based on statistics (dynamic)
    - Hardware decision (dynamic)
  - Prediction error: pipeline flush

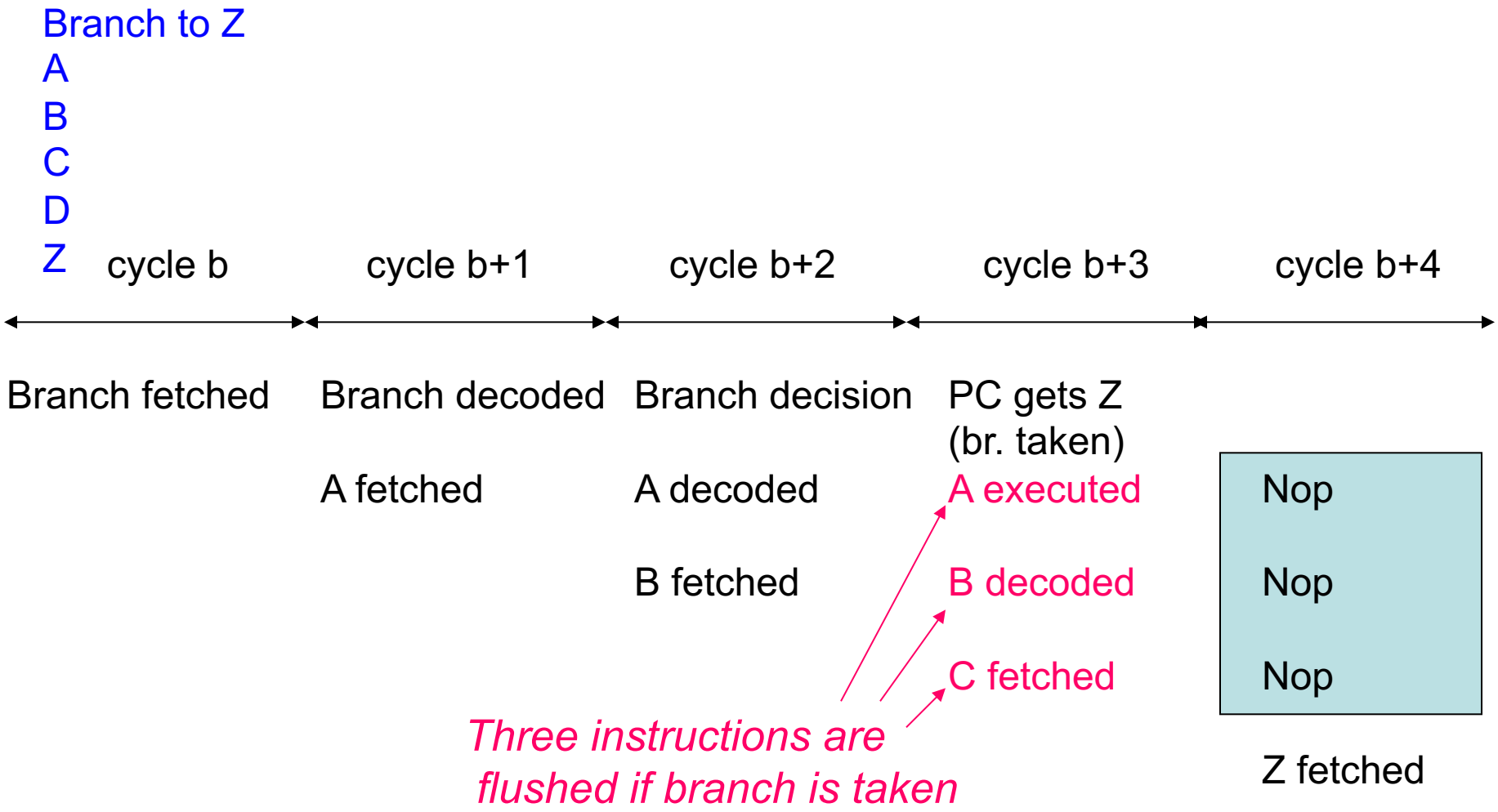
Branch Hazard

- Consider heuristic – branch not taken.
- Continue fetching instructions in sequence following the branch instructions.
- If branch is taken (indicated by *zero* output of ALU):
  - Control generates *branch* signal in ID cycle.
  - *branch* activates *PCSource* signal in the MEM cycle to load PC with new branch address.
  - *Three instructions in the pipeline must be flushed if branch is taken – can this penalty be reduced?*

Branch Not Taken

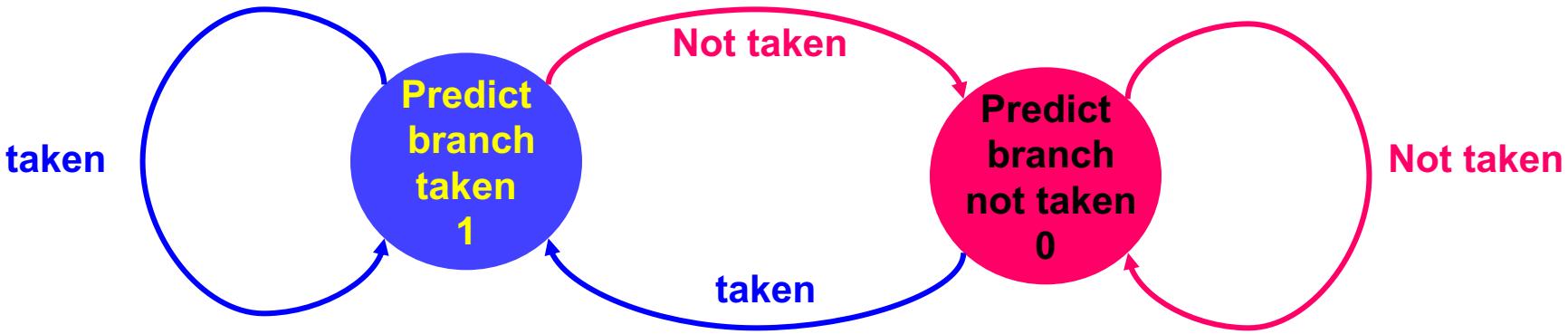


Branch Taken

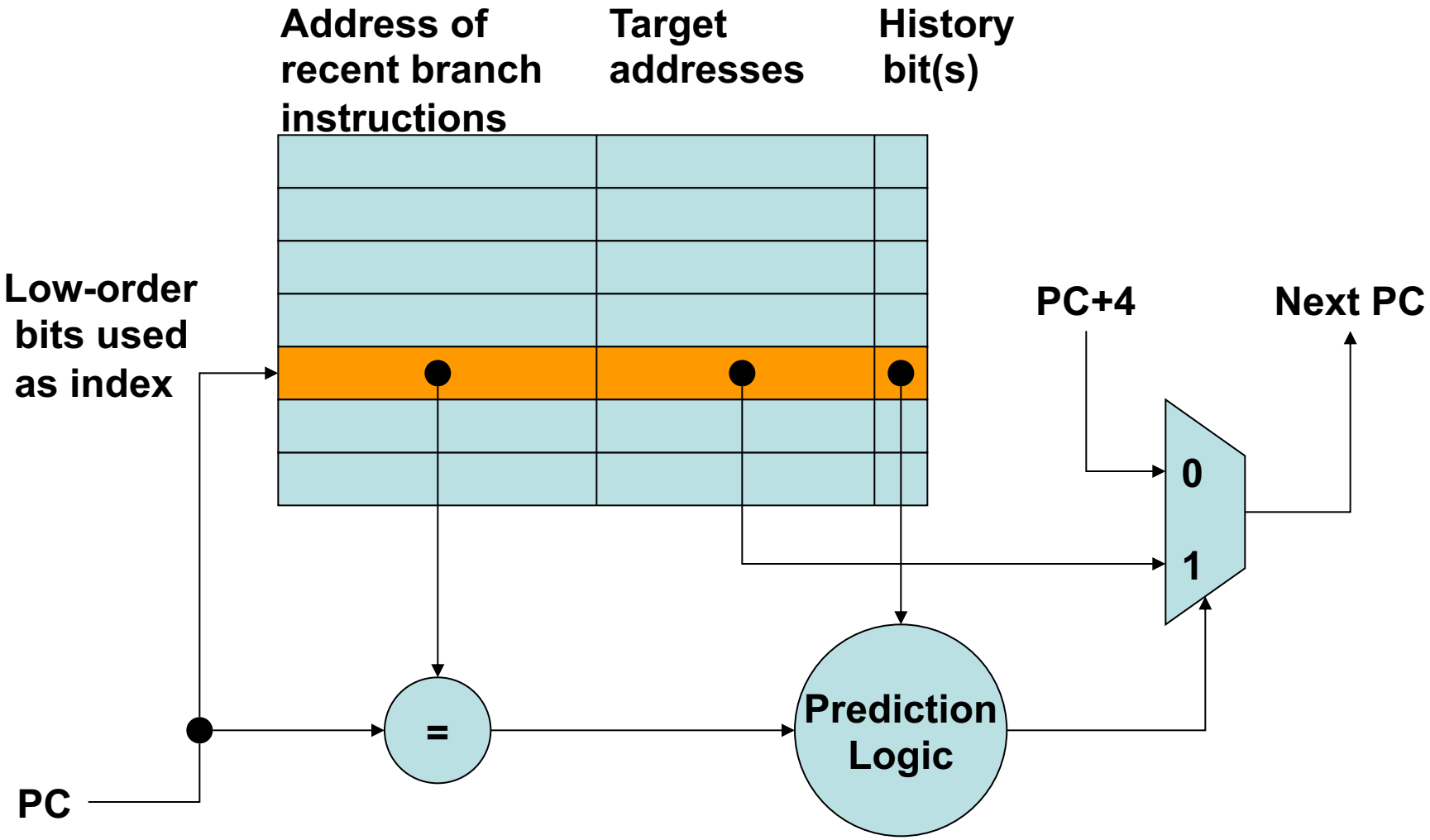


Branch Prediction

- Useful for program loops.
- A one-bit prediction scheme: a one-bit buffer carries a "history bit" that tells what happened on the last branch instruction
  - History bit = 1, branch was taken
  - History bit = 0, branch was not taken



Branch Prediction



Thank You