RISC Design

Pipeline Hazards

Virendra Singh

Computer Architecture and Dependable Systems Lab

Department of Electrical Engineering

Indian Institute of Technology Bombay

http://www.ee.iitb.ac.in/~viren/

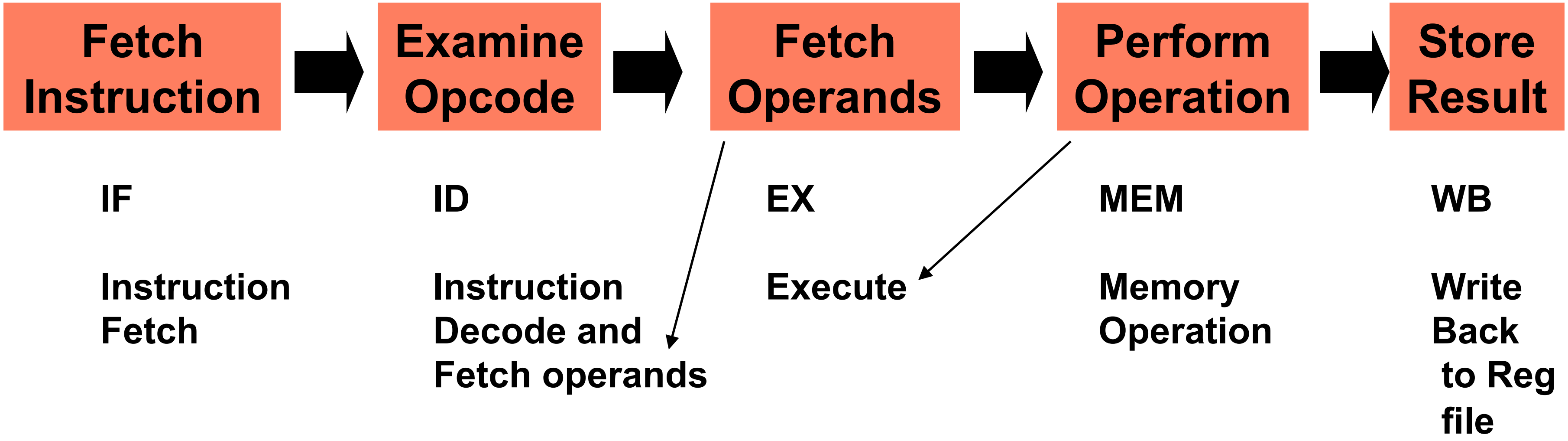
E-mail: viren@ee.iitb.ac.in

EE-309: Microprocessors

Lecture 37 (20 Oct 2015)

CADSL

Pipelining of RISC Instructions



Fetch Instruction → Examine Opcode → Fetch Operands → Perform Operation → Store Result

IF

Instruction Fetch

ID

Instruction Decode and Fetch operands

EX

Execute

MEM

Memory Operation

WB

Write Back to Reg file

Although an instruction takes five clock cycles, one instruction is completed every cycle.

Pipeline Hazards

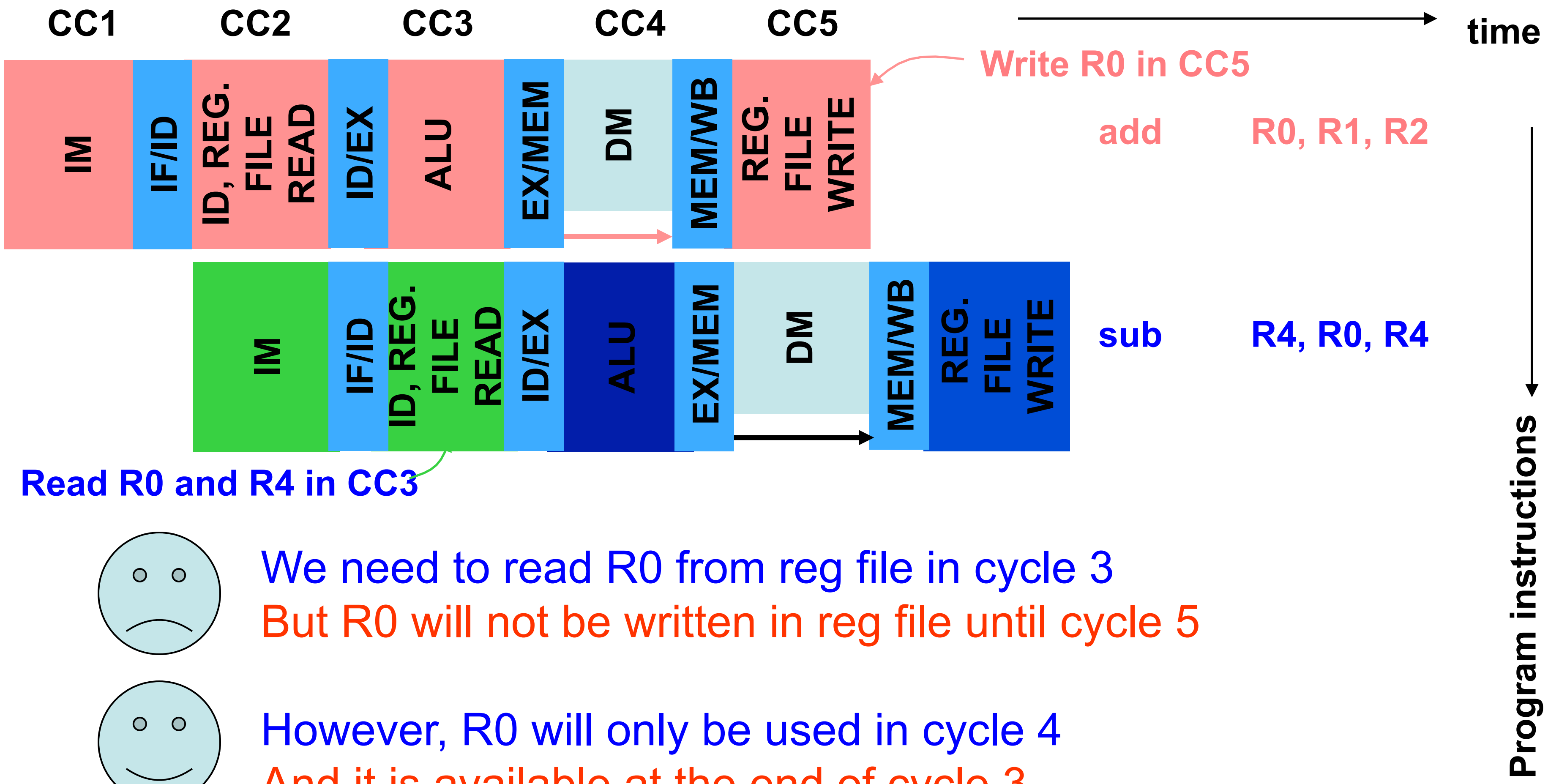
- Definition: *Hazard in a pipeline is a situation in which the next instruction cannot complete execution one clock cycle after completion of the present instruction.*
- Three types of hazards:
  - Structural hazard (resource conflict)
  - Data hazard
  - Control hazard

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     R0, R1, R2
– sub     R3, R0, R4              # needs R0
```

Example of Data Hazard



CC1 CC2 CC3 CC4 CC5 → time

Write R0 in CC5

IM | IF/ID | ID, REG. FILE READ | ID/EX | ALU | EX/MEM | DM | MEM/WB | REG. FILE WRITE — add R0, R1, R2

IM | IF/ID | ID, REG. FILE READ | ID/EX | ALU | EX/MEM | DM | MEM/WB | REG. FILE WRITE — sub R4, R0, R4

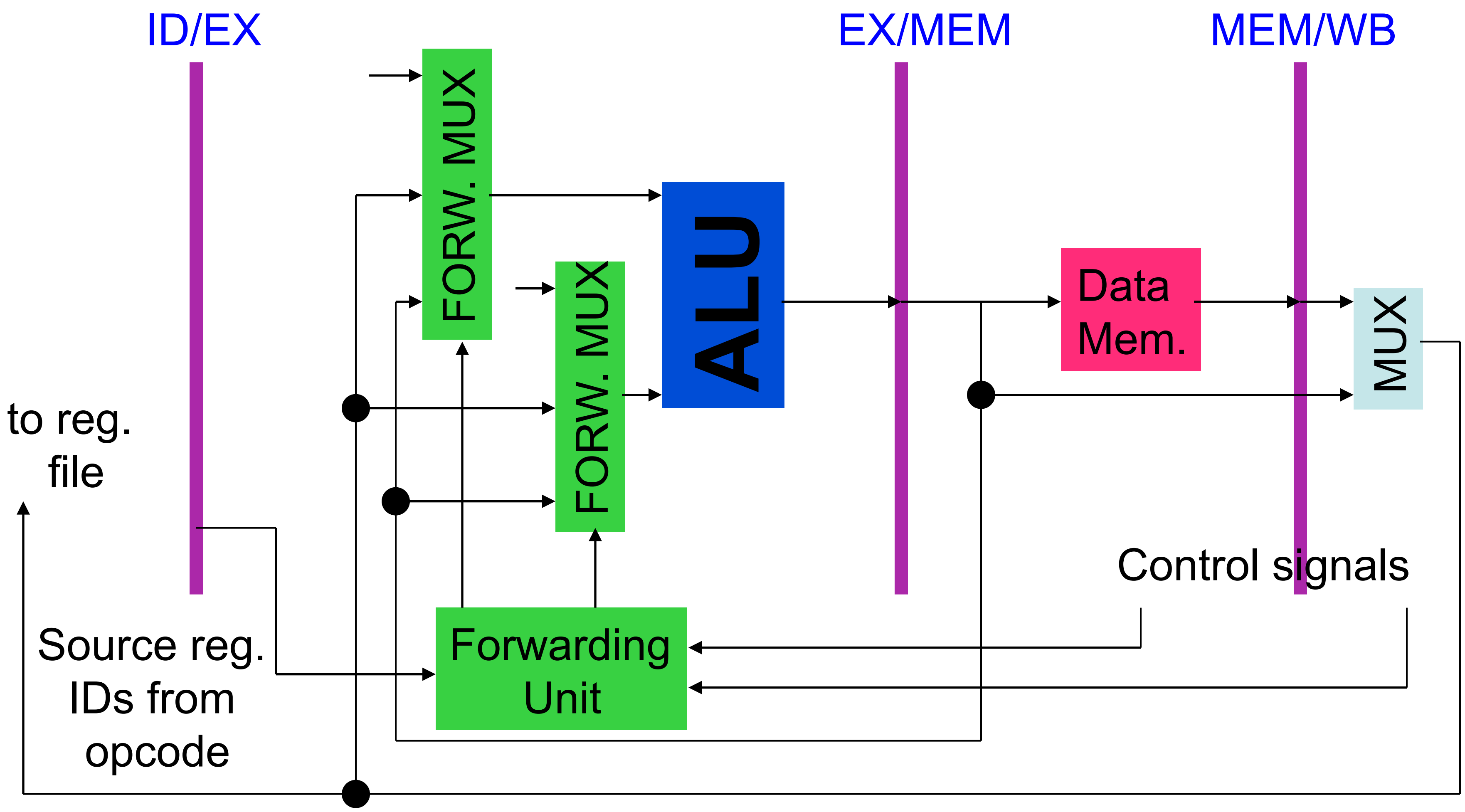
Read R0 and R4 in CC3

Program instructions

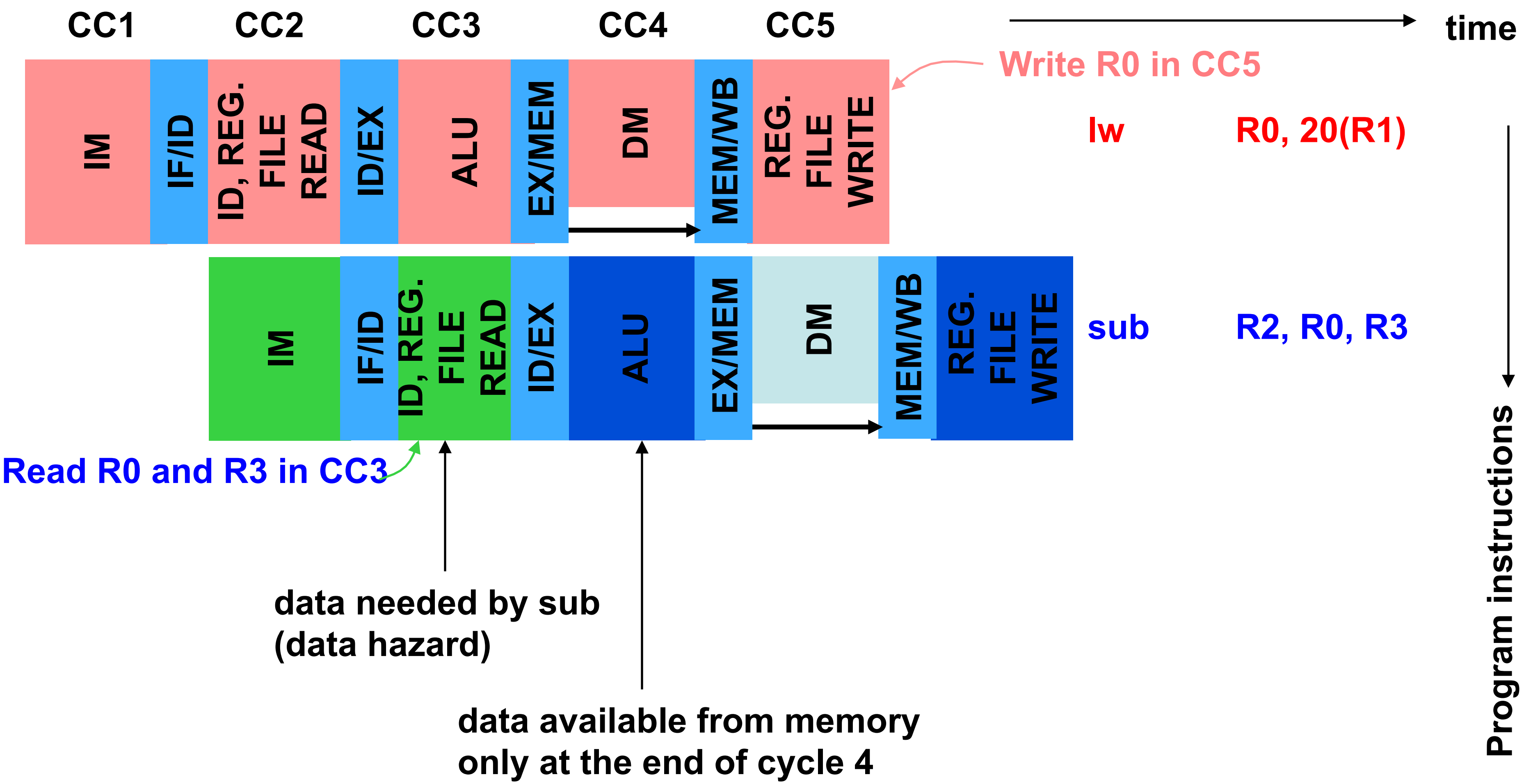
We need to read R0 from reg file in cycle 3
But R0 will not be written in reg file until cycle 5

However, R0 will only be used in cycle 4
And it is available at the end of cycle 3

Forwarding Unit Hardware



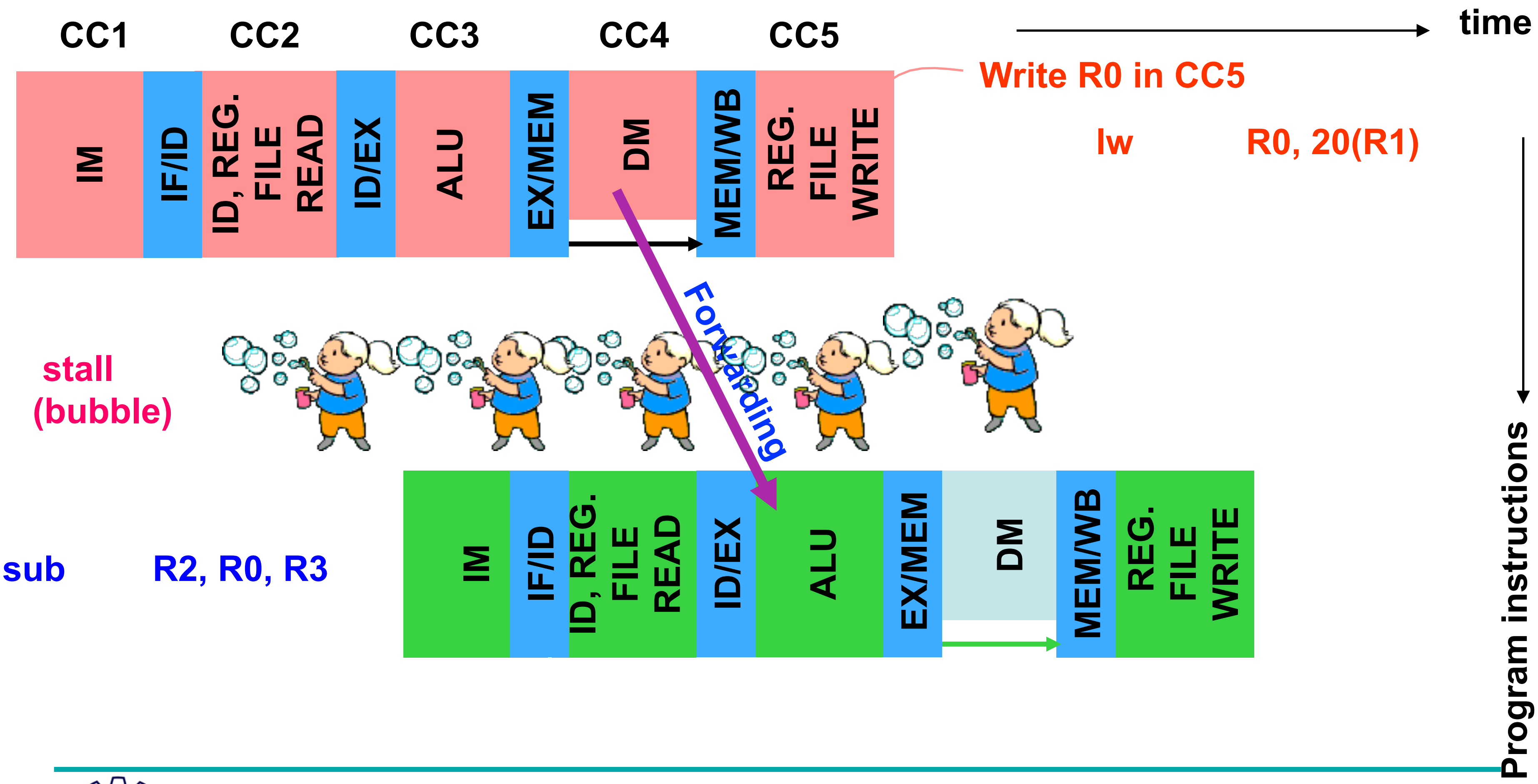
Forwarding Alone May Not Work



data needed by sub
(data hazard)

data available from memory
only at the end of cycle 4

Use Bubble and Forwarding



CC1 CC2 CC3 CC4 CC5

time

Write R0 in CC5

lw R0, 20(R1)

IM | IF/ID | ID, REG. FILE READ | ID/EX | ALU | EX/MEM | DM | MEM/WB | REG. FILE WRITE

Forwarding

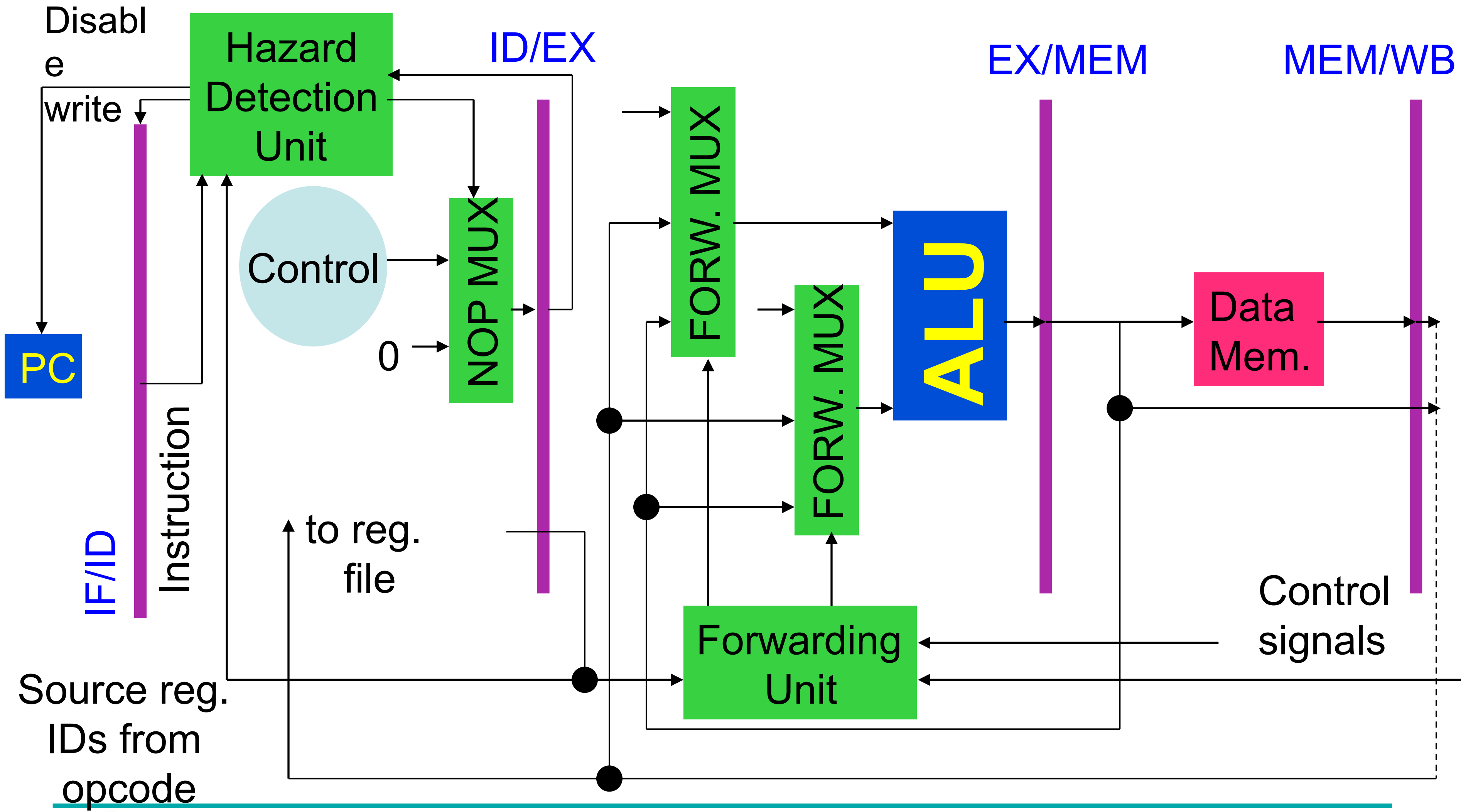
stall (bubble)

sub R2, R0, R3

IM | IF/ID | ID, REG. FILE READ | ID/EX | ALU | EX/MEM | DM | MEM/WB | REG. FILE WRITE

Program instructions

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.

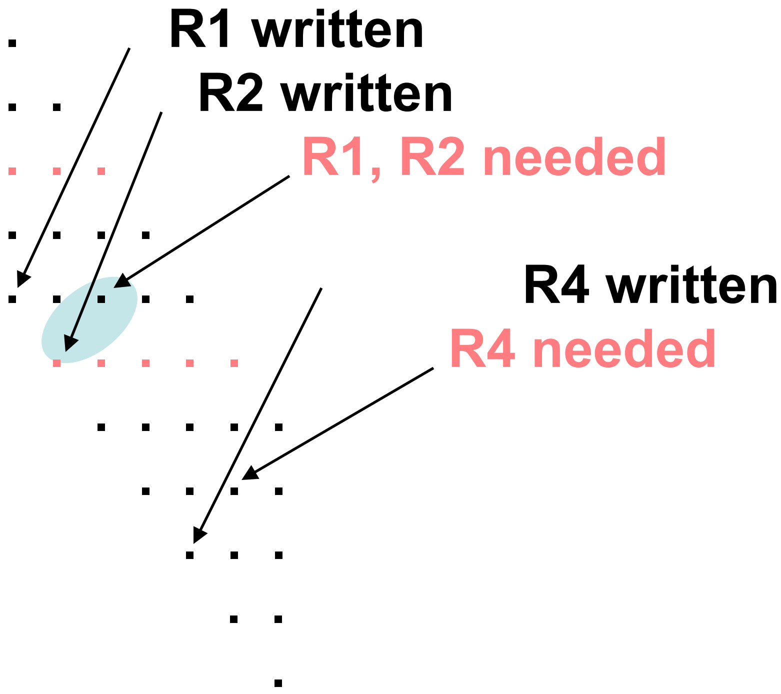
Avoiding Stall by Code Reorder

C code:

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

DLX code:

```
lw      R1,     0(R0)
lw      R2,     4(R0)
add     R3,     R1, R2
sw      R3,     12(R0)
lw      R4,     8(R0)
add     R5,     R1, R4
sw      R5,     16,(R0)
```

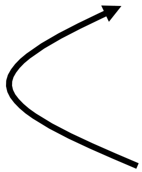


Reordered Code

C code:

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

DLX code:



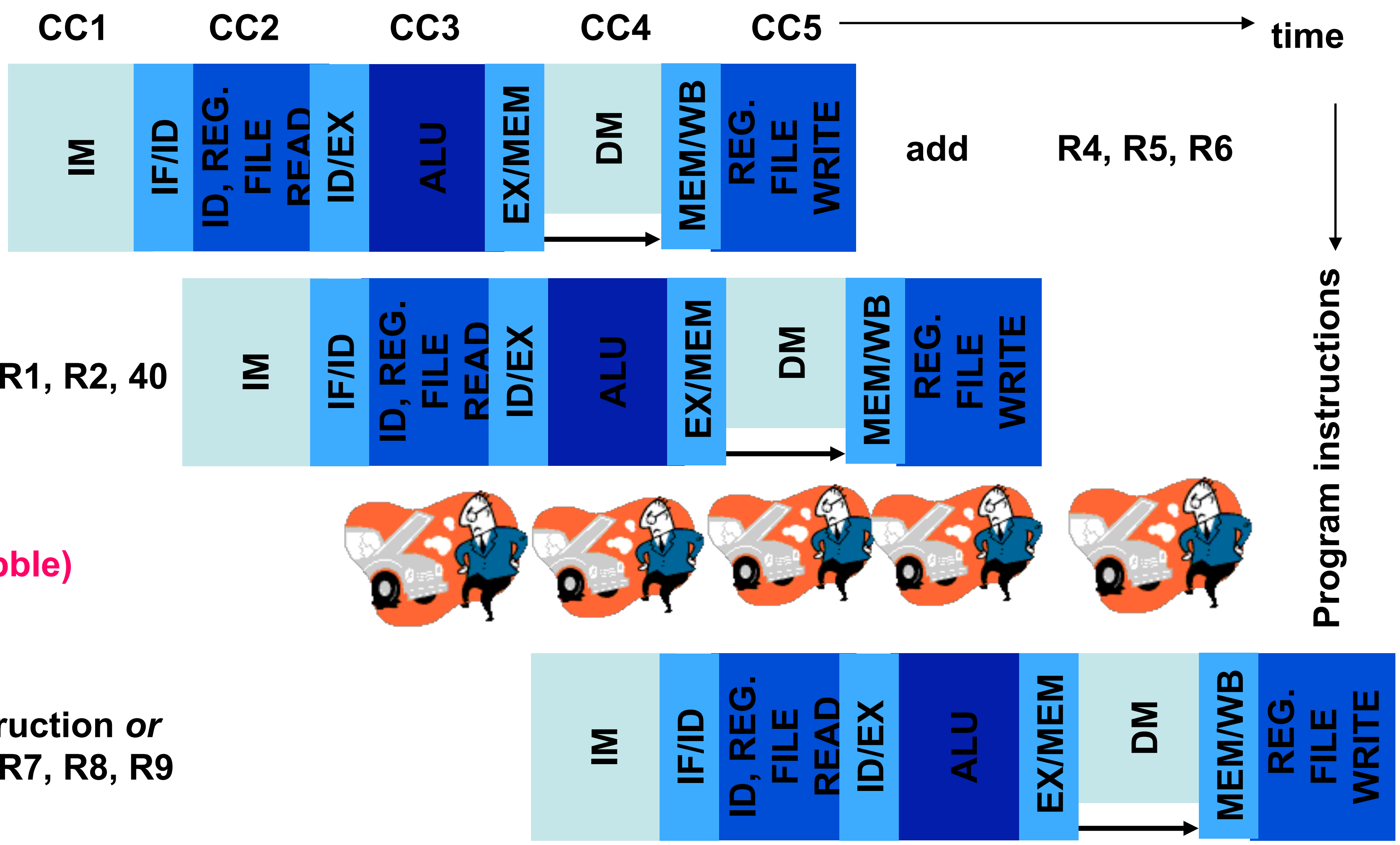
```
lw      R1,     0(R0)
lw      R2,     4(R0)
lw      R4,     8(R0)
add     R3,     R1, R2          no hazard
sw      R3,     12(R0)
add     R5,     R1, R4          no hazard
sw      R5,     16,(R0)
```

Control Hazard

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

```
    add   R4, R5, R6
    beq   R1, R2, 40
    next instruction
    . . .
  40and   R7, R8, R9
```

Stall on Branch



beq R1, R2, 40

Stall (bubble)

next instruction *or*
and R7, R8, R9

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

Delayed Branch Example

- Stall on branch

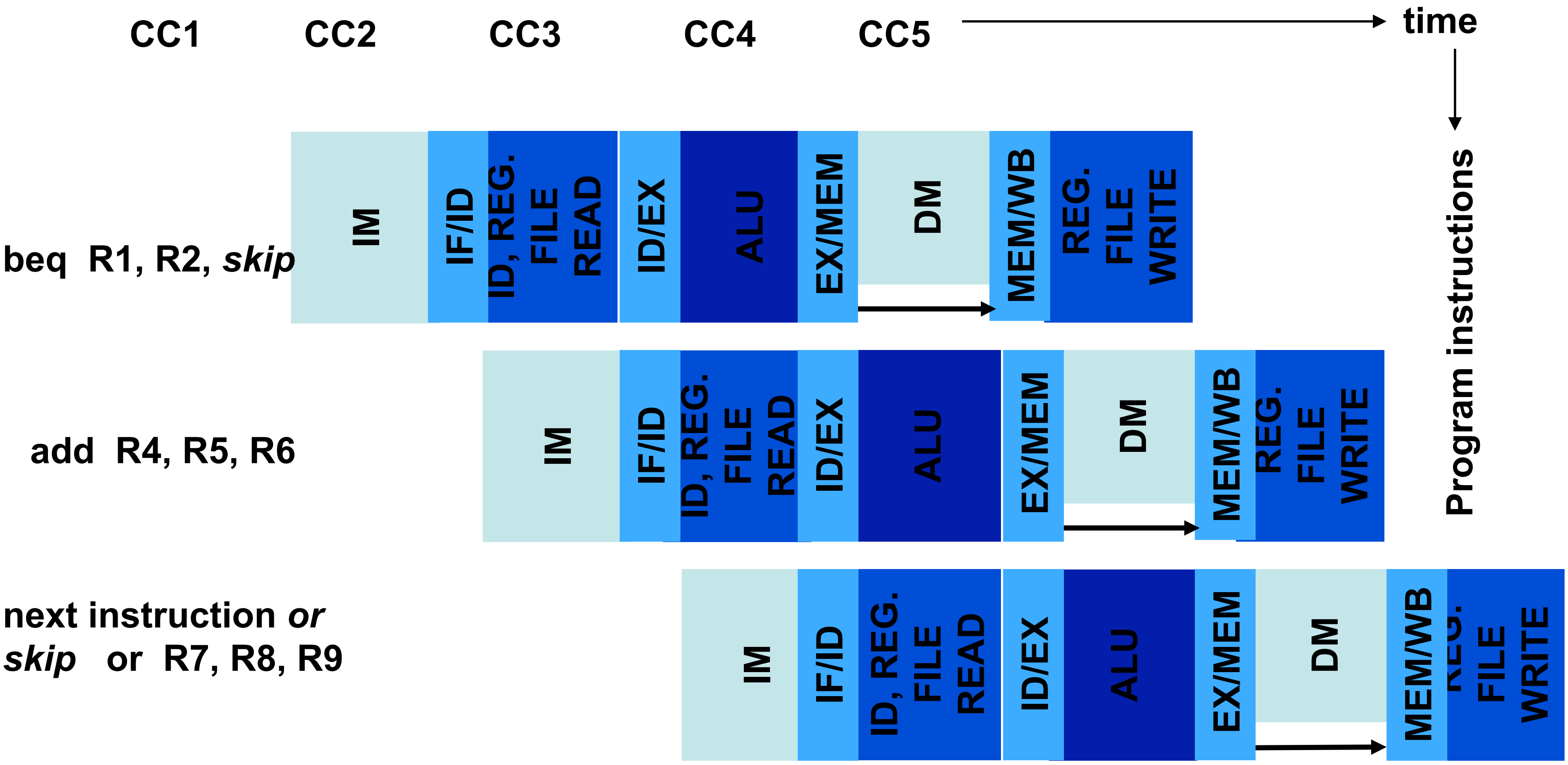
```
      add R4, R5, R6
      beq R1, R2, skip
      next instruction
      . . .
skip  or R7, R8, R9
```

- Delayed branch

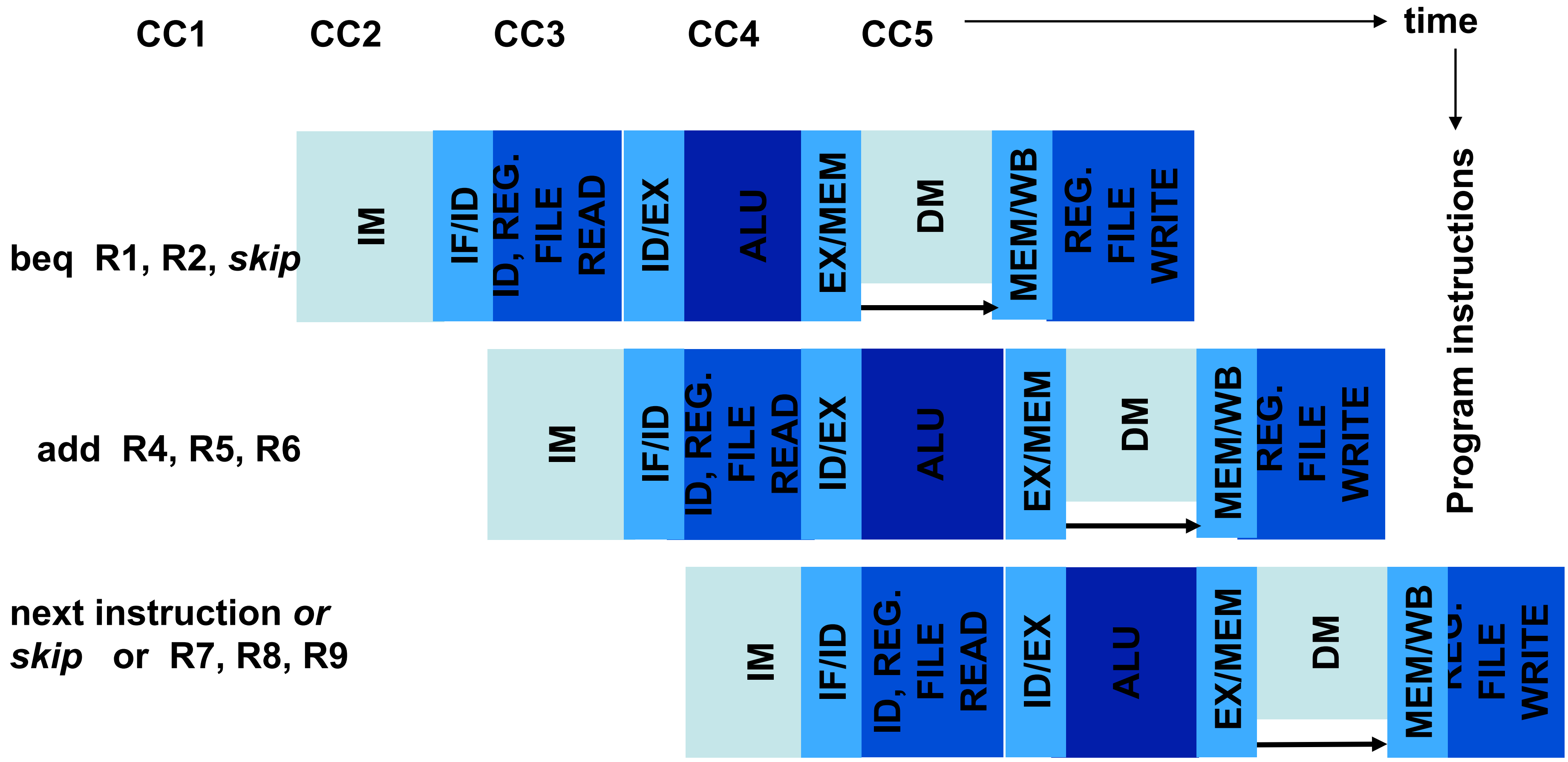
```
      beq R1, R2, skip
      add R4, R5, R6
      next instruction
      . . .
skip  or R7, R8, R9
```

Instruction executed irrespective of branch decision

Delayed Branch



Delayed Branch



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 to Z
A
B
C
D
Z

cycle b	cycle b+1	cycle b+2	cycle b+3	cycle b+4
Branch fetched	Branch decoded	Branch decision	PC keeps D (br. not taken)	
	A fetched	A decoded	A executed	A continues
		B fetched	B decoded	B executed
			C fetched	C decoded
				D fetched

Branch Taken

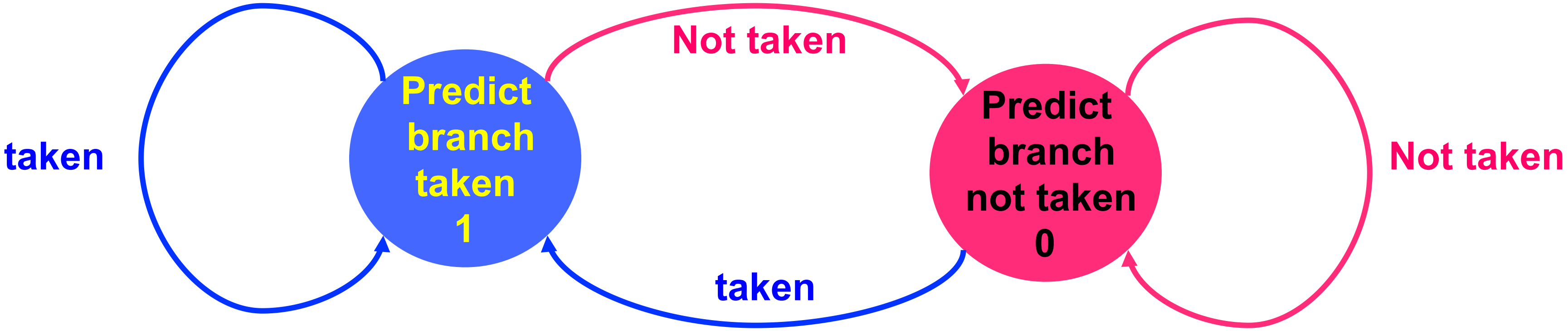
Branch to Z
A
B
C
D
Z

cycle b	cycle b+1	cycle b+2	cycle b+3	cycle b+4
Branch fetched	Branch decoded	Branch decision	PC gets Z (br. taken)	
	A fetched	A decoded	A executed	Nop
		B fetched	B decoded	Nop
			C fetched	Nop
				Z fetched

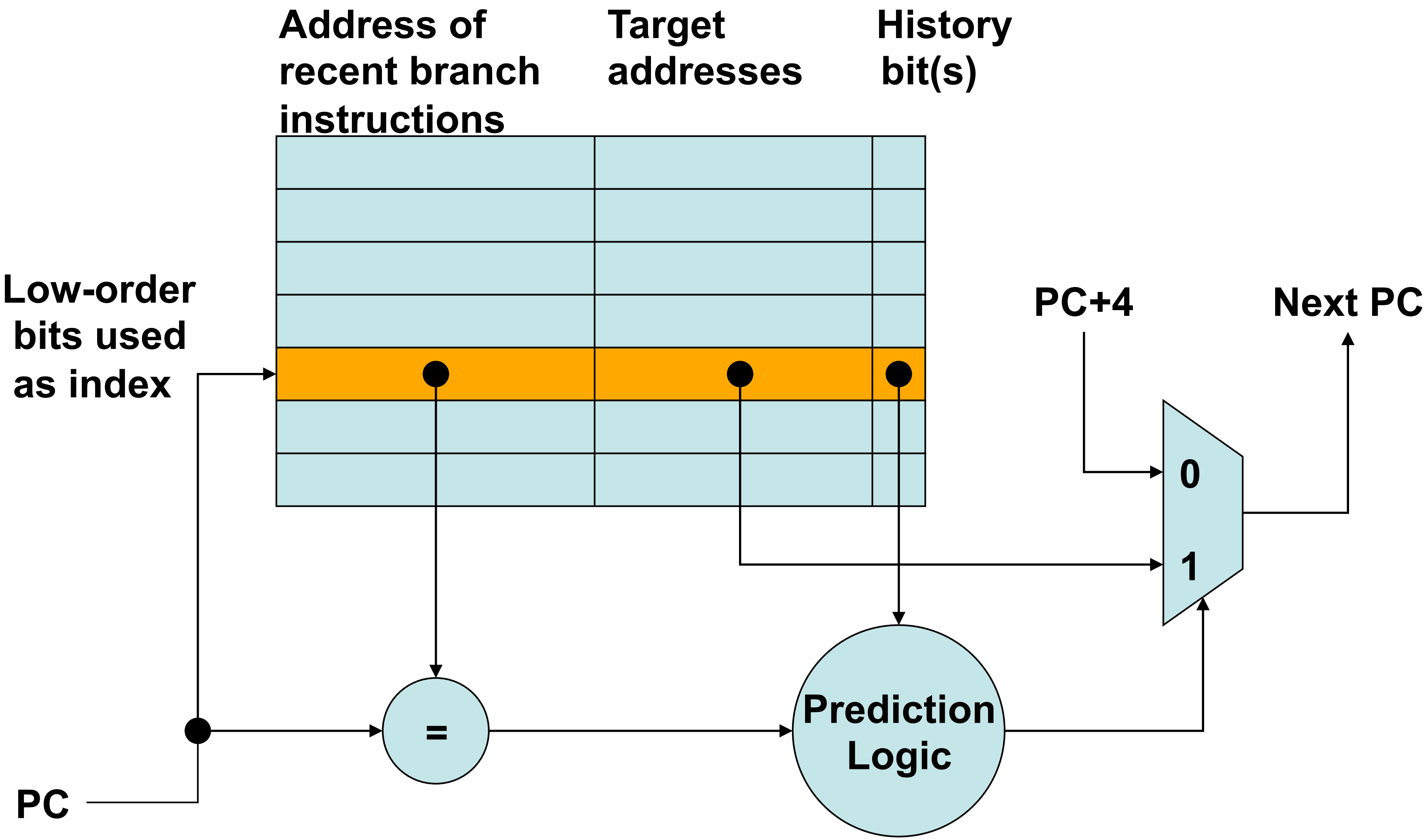
Three instructions are flushed if branch is taken

Branch Prediction

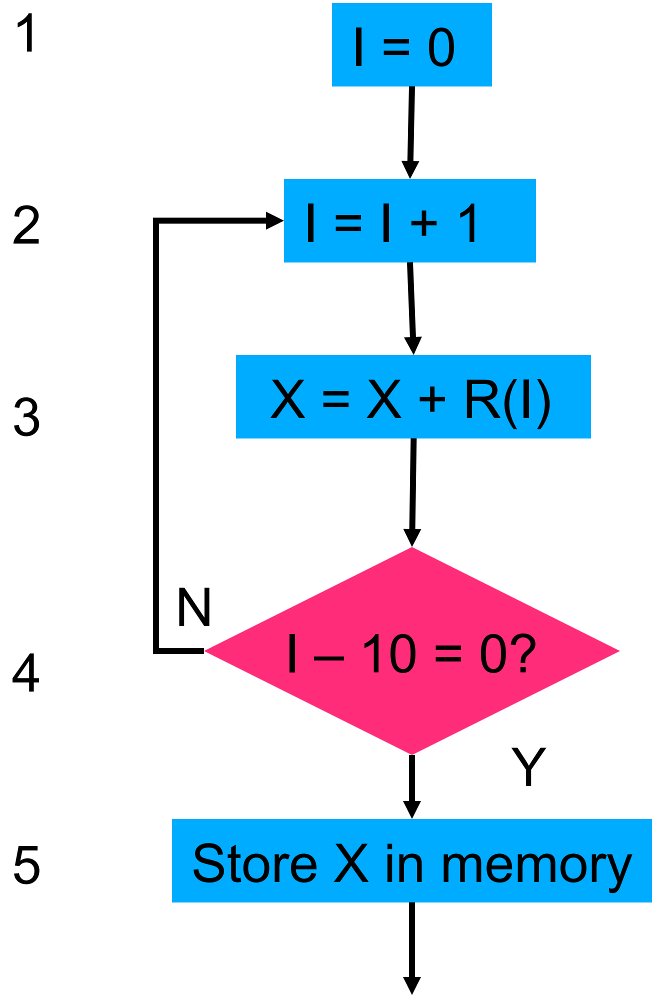
- Useful for program loops.
- A one-bit prediction scheme: a one-bit buffer carries a “history bit” that tells what happened on the last branch instruction
    - History bit = 1, branch was taken
    - History bit = 0, branch was not taken



Branch Prediction


Address of recent branch instructions
Target addresses
History bit(s)
Low-order bits used as index
PC+4
Next PC
0
1
=
Prediction Logic
PC

Branch Prediction for a Loop



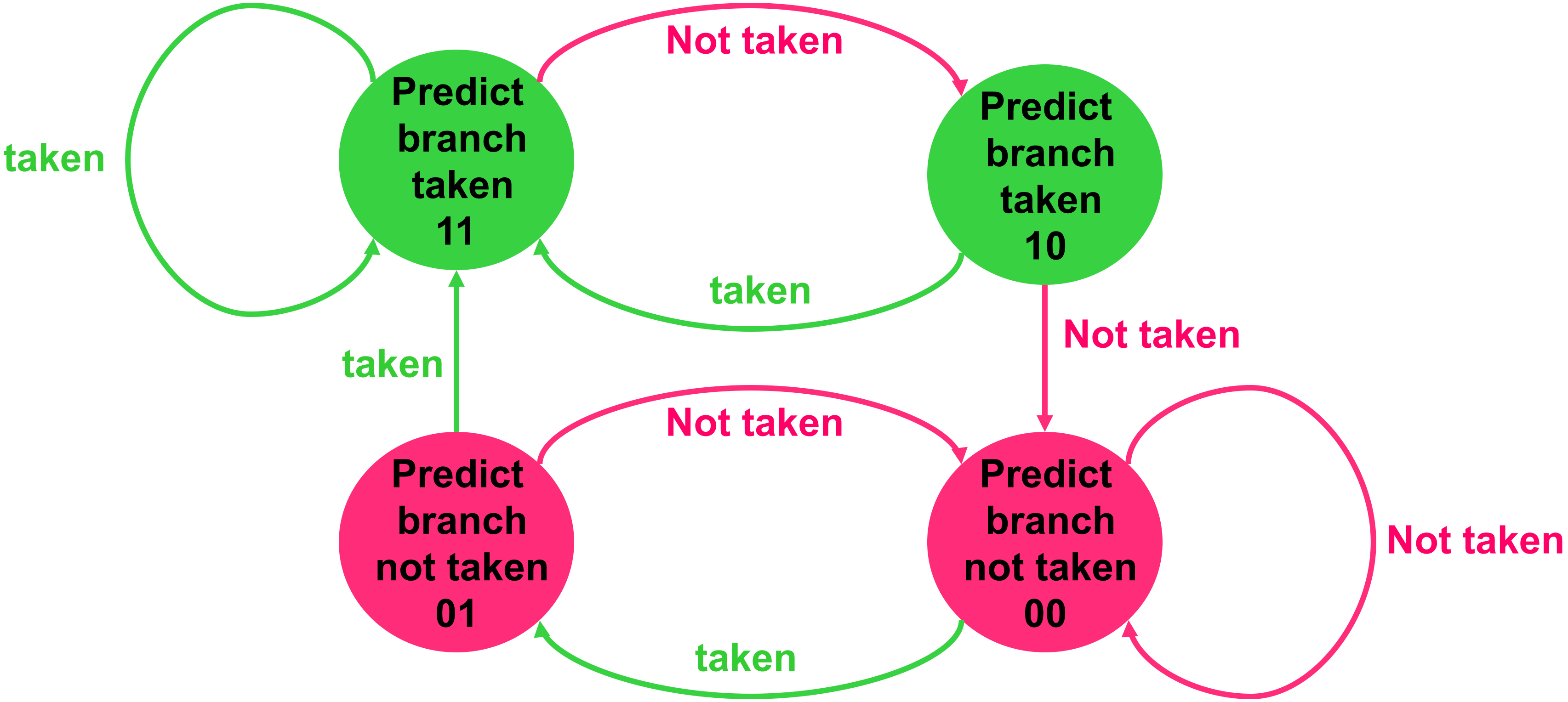
Execution of Instruction 4

Execu-tion seq.	Old hist. bit	Next instr.			New hist. bit	Prediction
		Pred.	I	Act.		
1	0	5	1	2	1	Bad
2	1	2	2	2	1	Good
3	1	2	3	2	1	Good
4	1	2	4	2	1	Good
5	1	2	5	2	1	Good
6	1	2	6	2	1	Good
7	1	2	7	2	1	Good
8	1	2	8	2	1	Good
9	1	2	9	2	1	Good
10	1	2	10	5	0	Bad

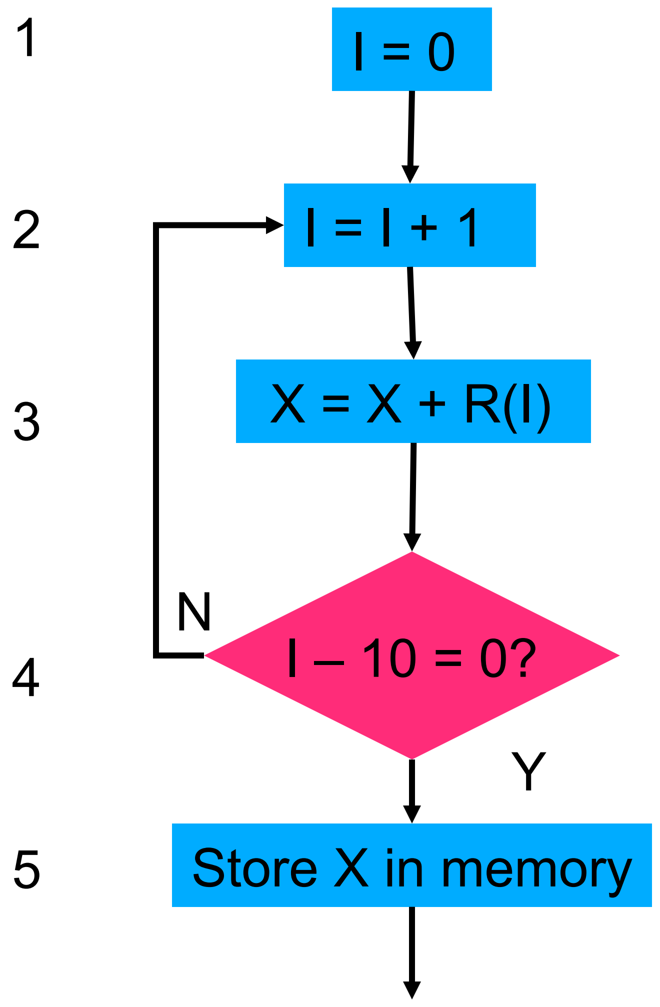
h.bit = 0 *branch not taken*, h.bit = 1 *branch taken*.

Two-Bit Prediction Buffer

- Can improve correct prediction statistics.



Branch Prediction for a Loop


1 I = 0

2 I = I + 1

3 X = X + R(I)

4 I – 10 = 0?

N

Y

5 Store X in memory

Execution of Instruction 4

Execu -tion seq.	Old Pred. Buf	Next instr.			New pred. Buf	Predi ction
		Pred.	I	Act.		
1	10	2	1	2	11	Good
2	11	2	2	2	11	Good
3	11	2	3	2	11	Good
4	11	2	4	2	11	Good
5	11	2	5	2	11	Good
6	11	2	6	2	11	Good
7	11	2	7	2	11	Good
8	11	2	8	2	11	Good
9	11	2	9	2	11	Good
10	11	2	10	5	10	Bad

Summary: Hazards

- Structural hazards
  - Cause: resource conflict
  - Remedies: (i) hardware resources, (ii) stall (bubble)
- Data hazards
  - Cause: data unavailablity
  - Remedies: (i) forwarding, (ii) stall (bubble), (iii) code reordering
- Control hazards
  - Cause: out-of-sequence execution (branch or jump)
  - Remedies: (i) stall (bubble), (ii) branch prediction/pipeline flush, (iii) delayed branch/pipeline flush

Thank You