RISC Design: Pipeline Hazards

Virendra Singh
Associate Professor
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-739: Processor Design

Lecture 15 (13 Feb 2013)

CADSL

ILP: Instruction Level Parallelism

- Single-cycle and multi-cycle datapaths execute one instruction at a time.
- How can we get better performance?
- Answer: Execute multiple instruction at a time:
  - Pipelining – Enhance a multi-cycle datapath to fetch one instruction every cycle.
  - Parallelism – Fetch multiple instructions every cycle.

Traffic Flow

Pipelining in a Computer

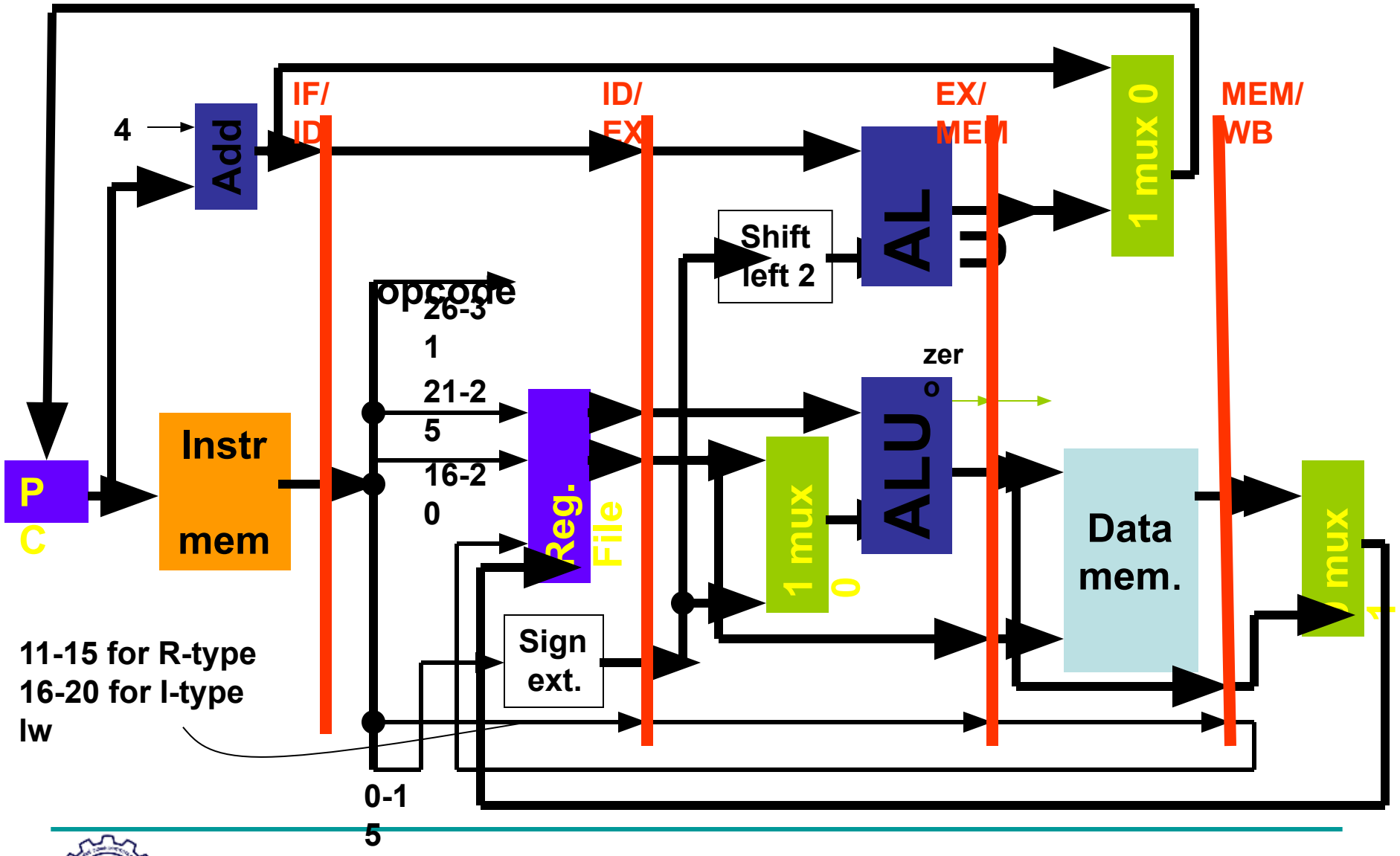
- Divide datapath into nearly equal tasks, to be performed serially and requiring non-overlapping resources.
- Insert registers at task boundaries in the datapath; registers pass the output data from one task as input data to the next task.
- Synchronize tasks with a clock having a cycle time that just exceeds the time required by the longest task.
- Break each instruction down into a fixed number of tasks so that instructions can be executed in a staggered fashion.

Pipelined Datapath

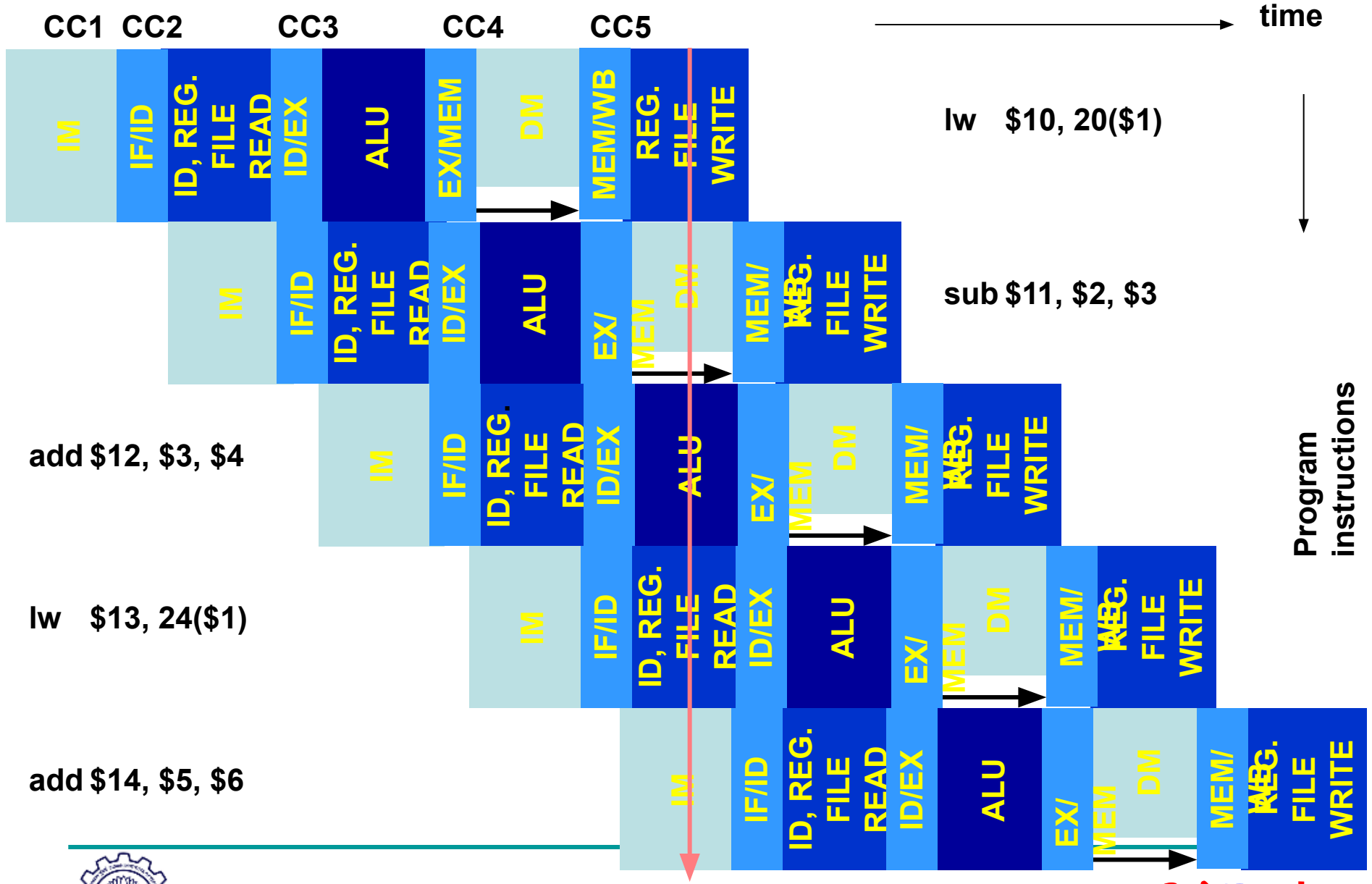
Instruction class	Instr. fetch (IF)	Instr. Decode (also reg. file read) (ID)	Execu-tion (ALU Opera-tion) (EX)	Data access (MEM)	Write Back (Reg. file write) (WB)	Total time
lw	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
sw	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
R-format: add, sub, and, or, slt	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns
B-format: beq	2ns	~~1ns~~ 2ns	2ns	2ns	~~1ns~~ 2ns	10ns

No operation on data; idle time inserted to equalize instruction lengths.

Pipelined Datapath



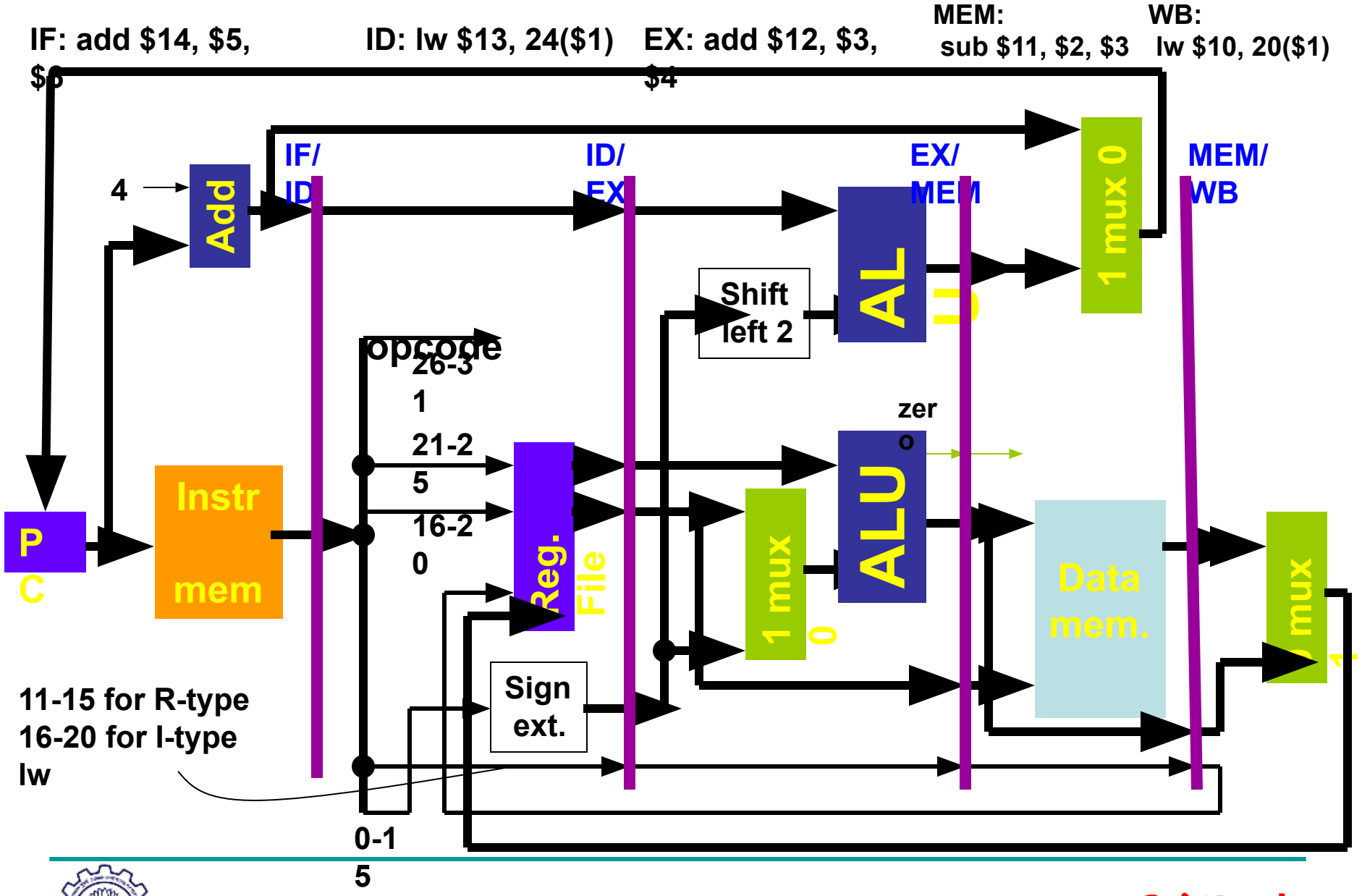
Program Execution



Single Lane Traffic

Wish list: Highway

CC5


IF: add $14, $5, $6
ID: lw $13, 24($1)
EX: add $12, $3, $4
MEM: sub $11, $2, $3
WB: lw $10, 20($1)
4
Add
IF/ID
ID/EX
EX/MEM
MEM/WB
1 mux 0
Shift left 2
ALU
opcode
26-31
21-25
16-20
0
zero
P C
Instr mem
Reg. File
1 mux 0
ALU
Data mem.
1 mux 0
Sign ext.
11-15 for R-type
16-20 for I-type
lw
0-15

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 . . .

Science is always wrong. It never solves a problem without creating ten more.

George Bernard Shaw

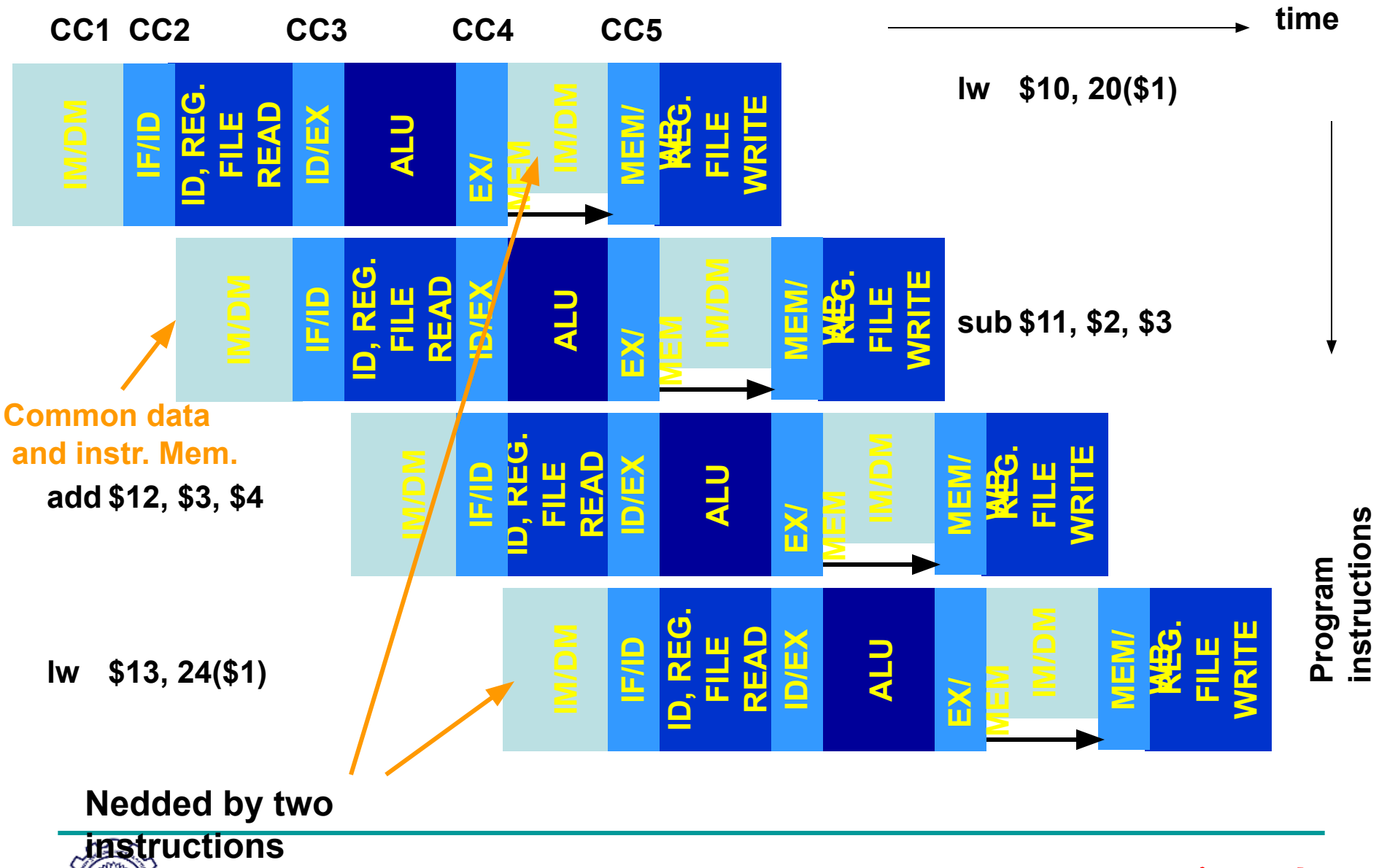
Pipeline Hazards

- 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

Structural Hazard

- Two instructions cannot execute due to a resource conflict.
- Example: Consider a computer with a common data and instruction memory. The fourth cycle of a *lw* instruction requires memory access (memory read) and at the same time the first cycle of the fourth instruction requires instruction fetch (memory read). This will cause a memory resource conflict.

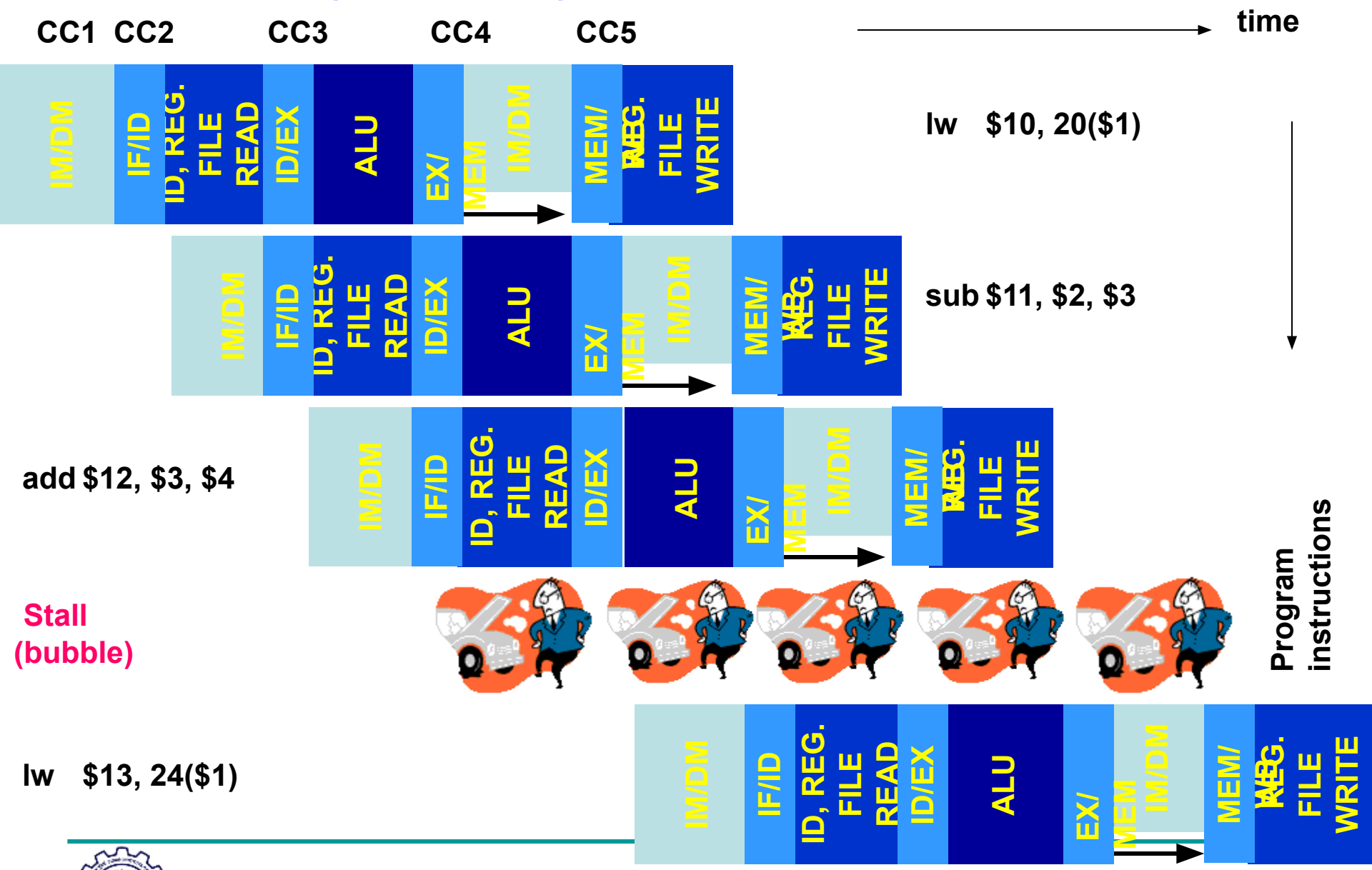
Example of Structural Hazard



Possible Remedies for Structural Hazards

- Provide duplicate hardware resources in datapath.
- Control unit or compiler can insert delays (no-op cycles) between instructions. This is known as pipeline *stall* or *bubble*.

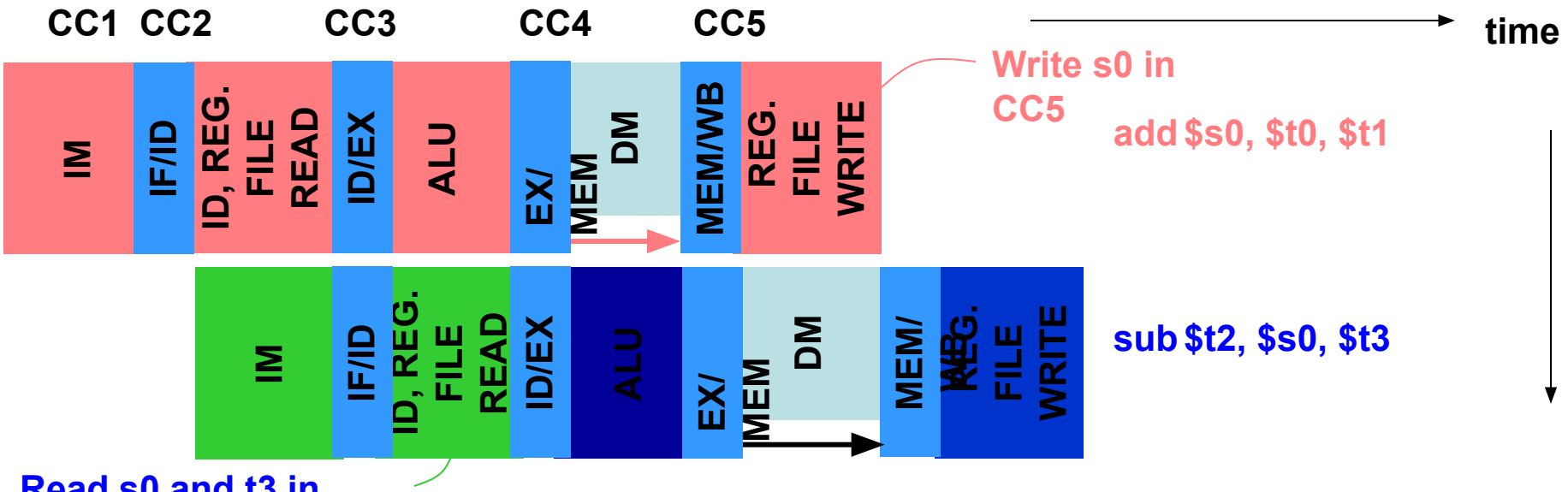
Stall (Bubble) for Structural Hazard



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 $s0, $t0, $t1
  - sub $t2, $s0, $t3 # needs $s0

Example of Data Hazard



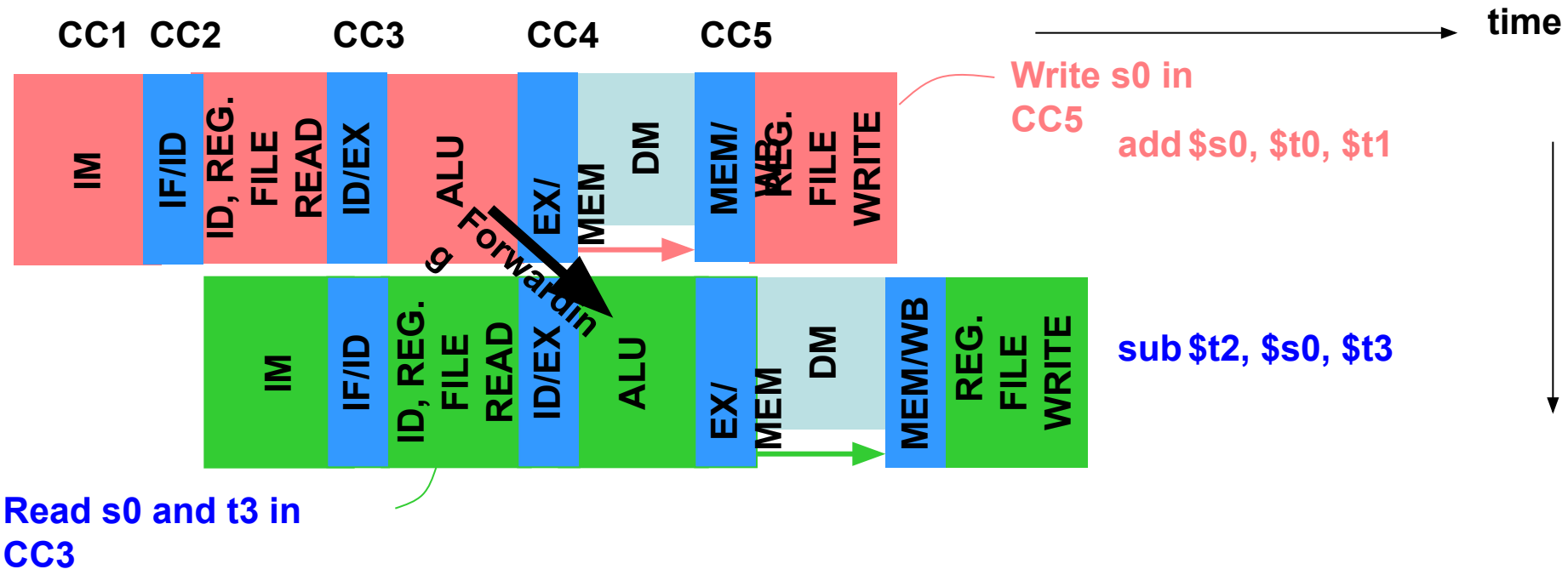
We need to read s0 from reg file in cycle 3
But s0 will not be written in reg file until cycle 5

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

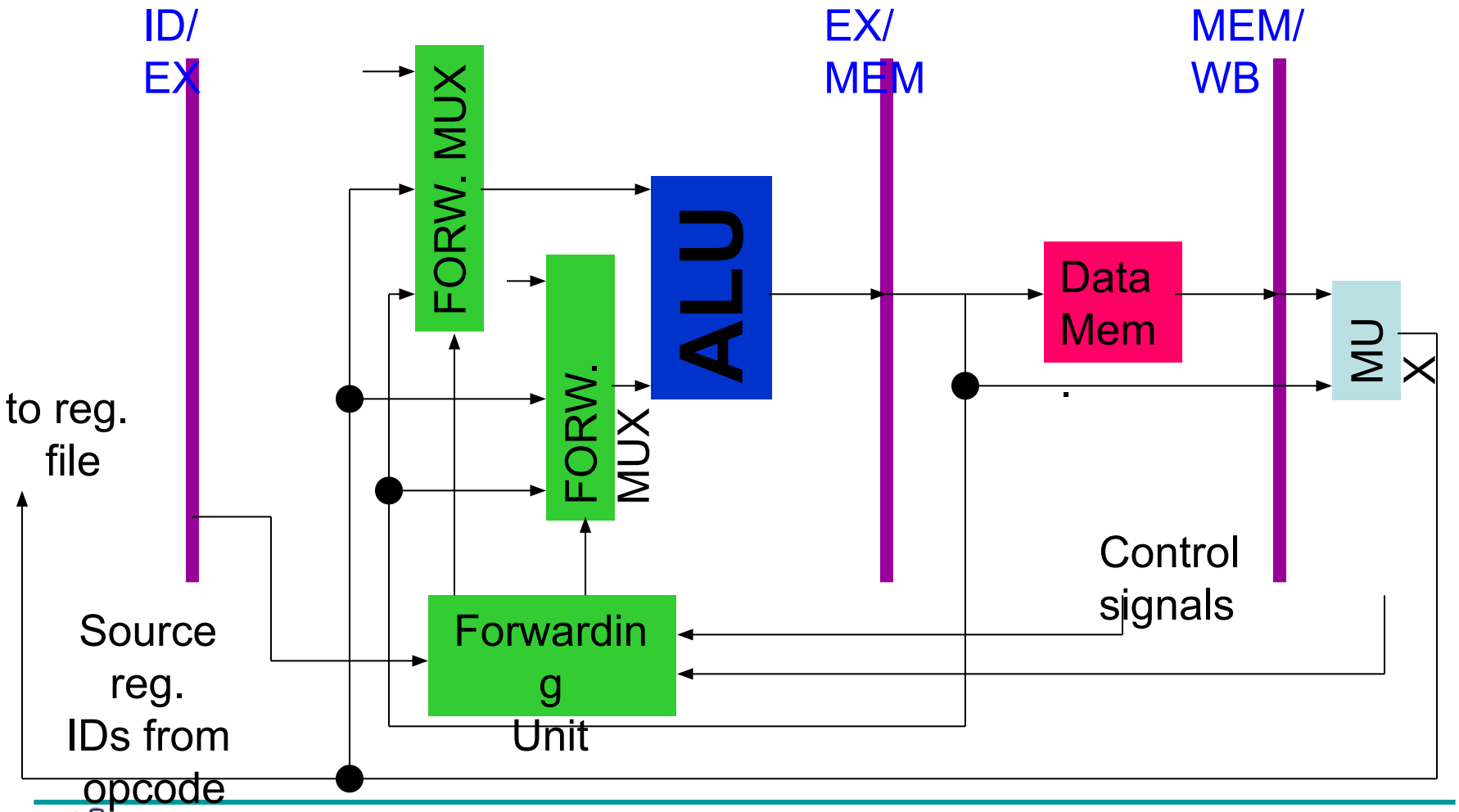
Forwarding or Bypassing

- 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.

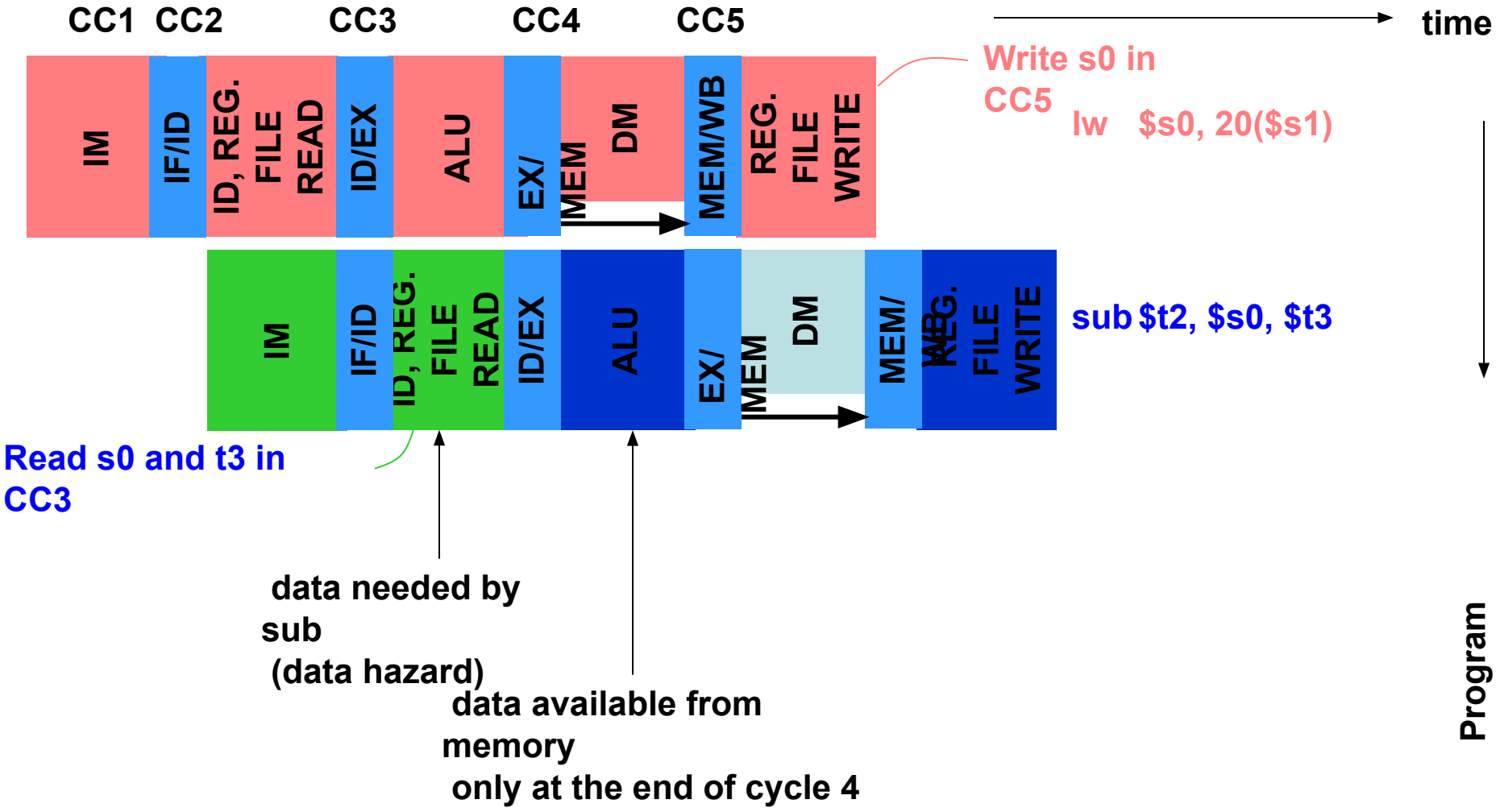
Forwarding for Data Hazard



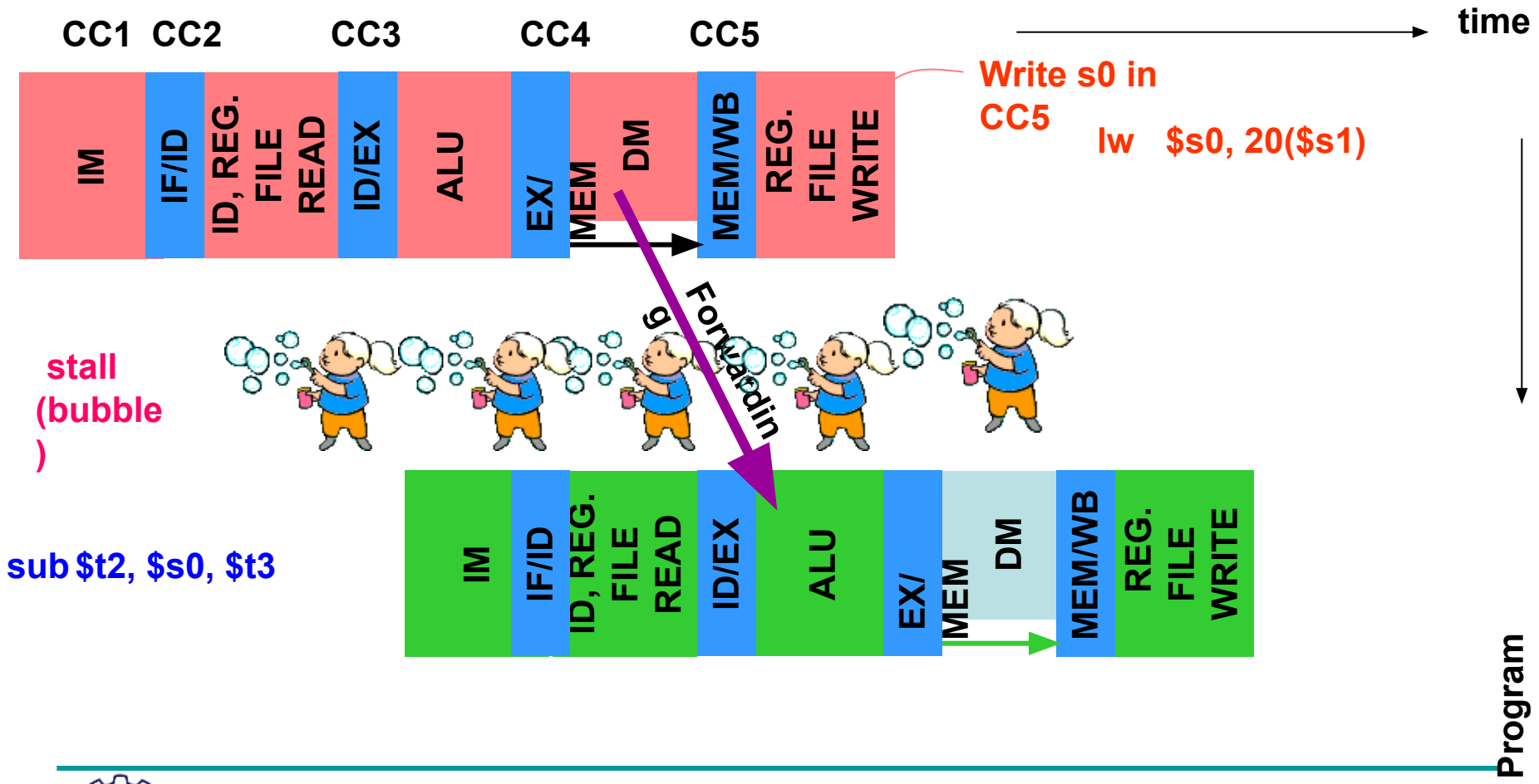
Forwarding Unit Hardware



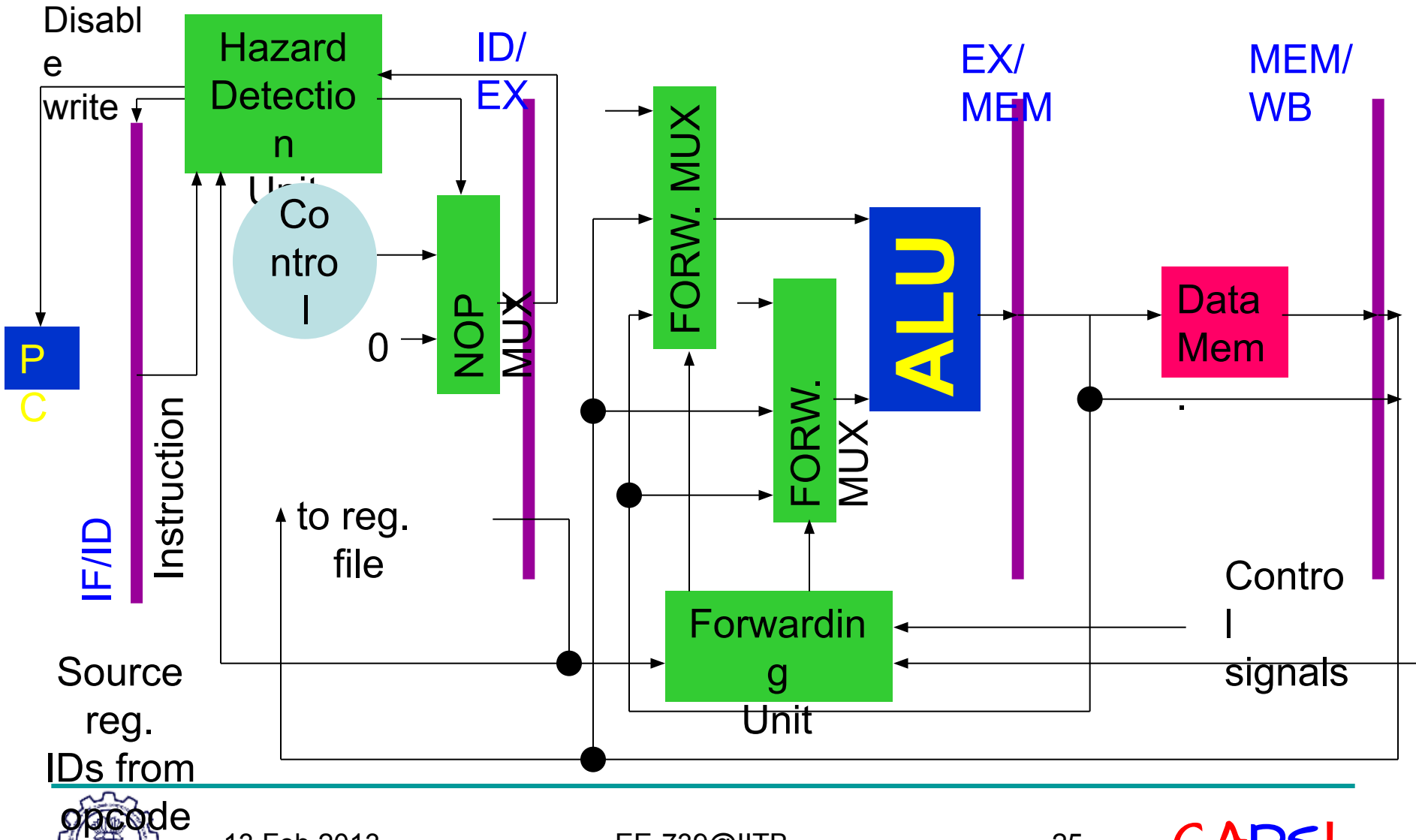
Forwarding Alone May Not Work



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.

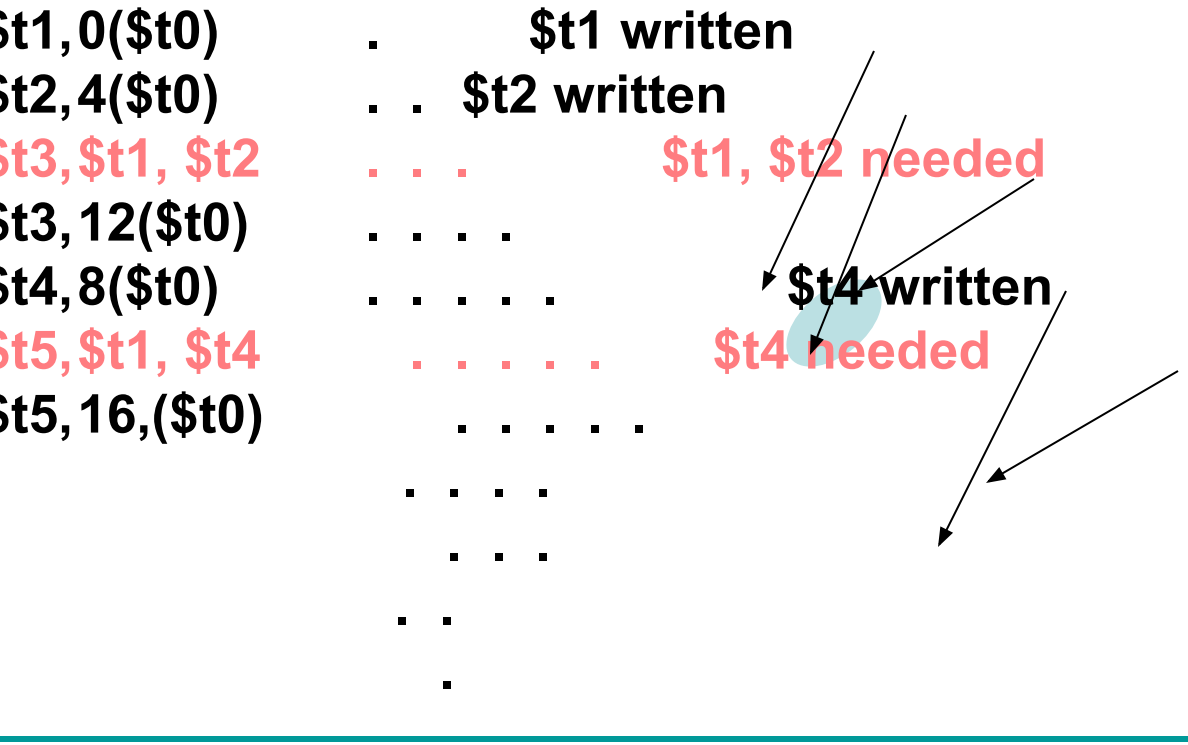
Avoiding Stall by Code Reorder

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

C code:

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

MIPS code:

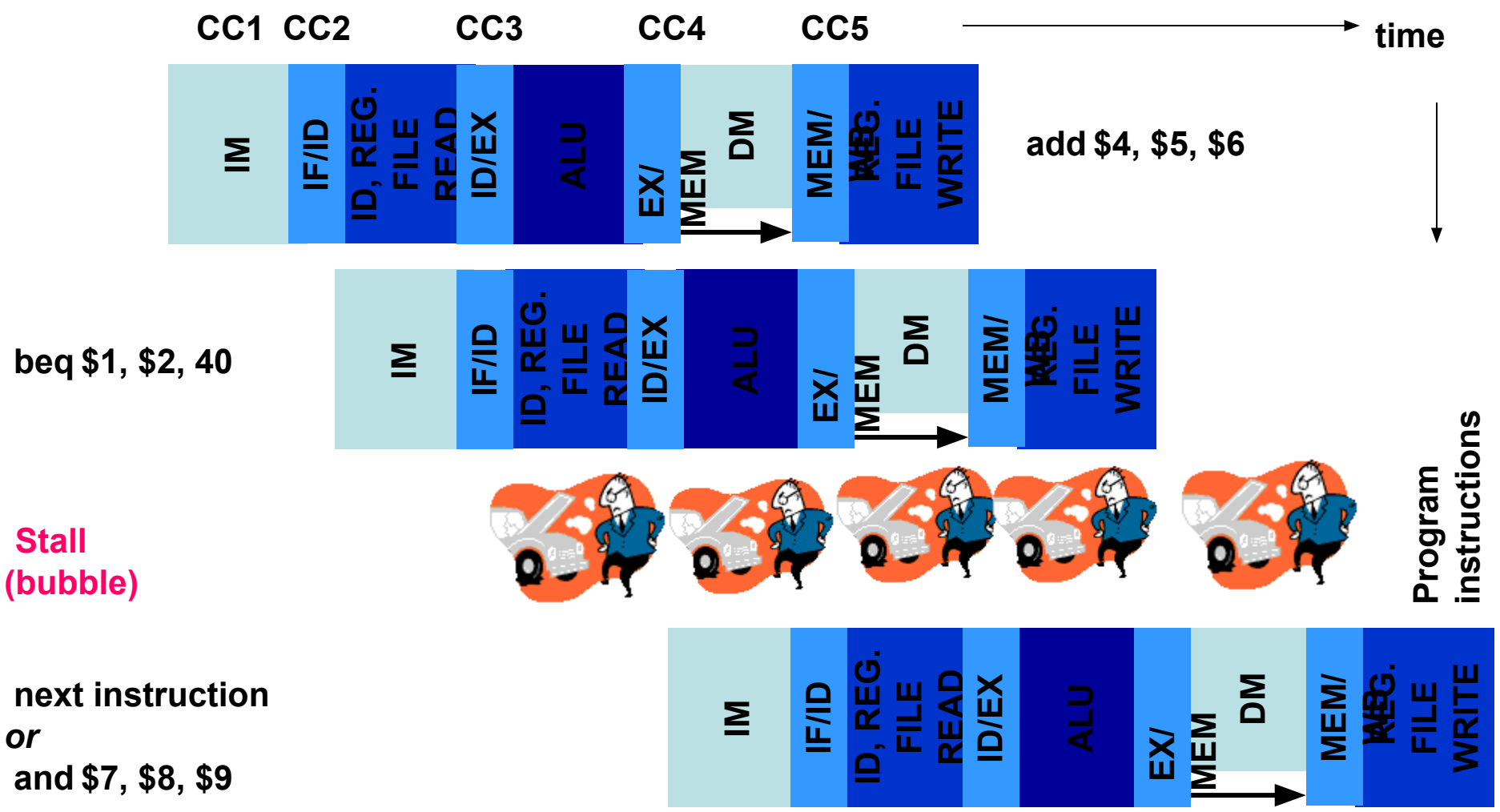
```
lw   $t1,0($t0)
lw   $t2,4($t0)
lw   $t4,8($t0)
add$t3,$t1, $t2      no hazard
sw  $t3,12($t0)
add$t5,$t1, $t4      no hazard
sw  $t5,16,($t0)
```

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
```

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
- 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

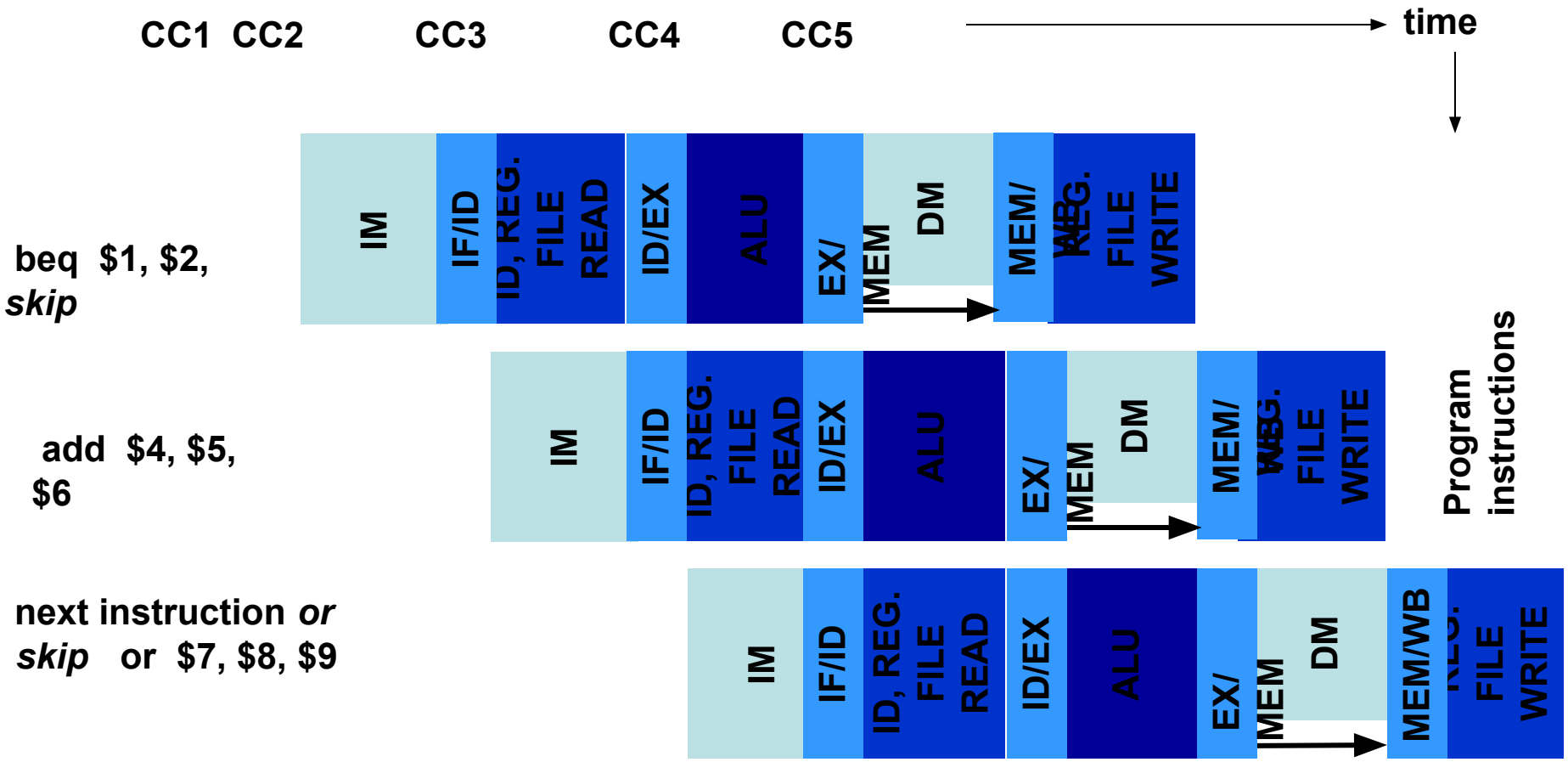
```
        add $4, $5, $6
        beq $1, $2, skip
        next instruction
        . . .
skip    or $7, $8, $9
```

- Delayed branch

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

Instruction executed irrespective of branch decision

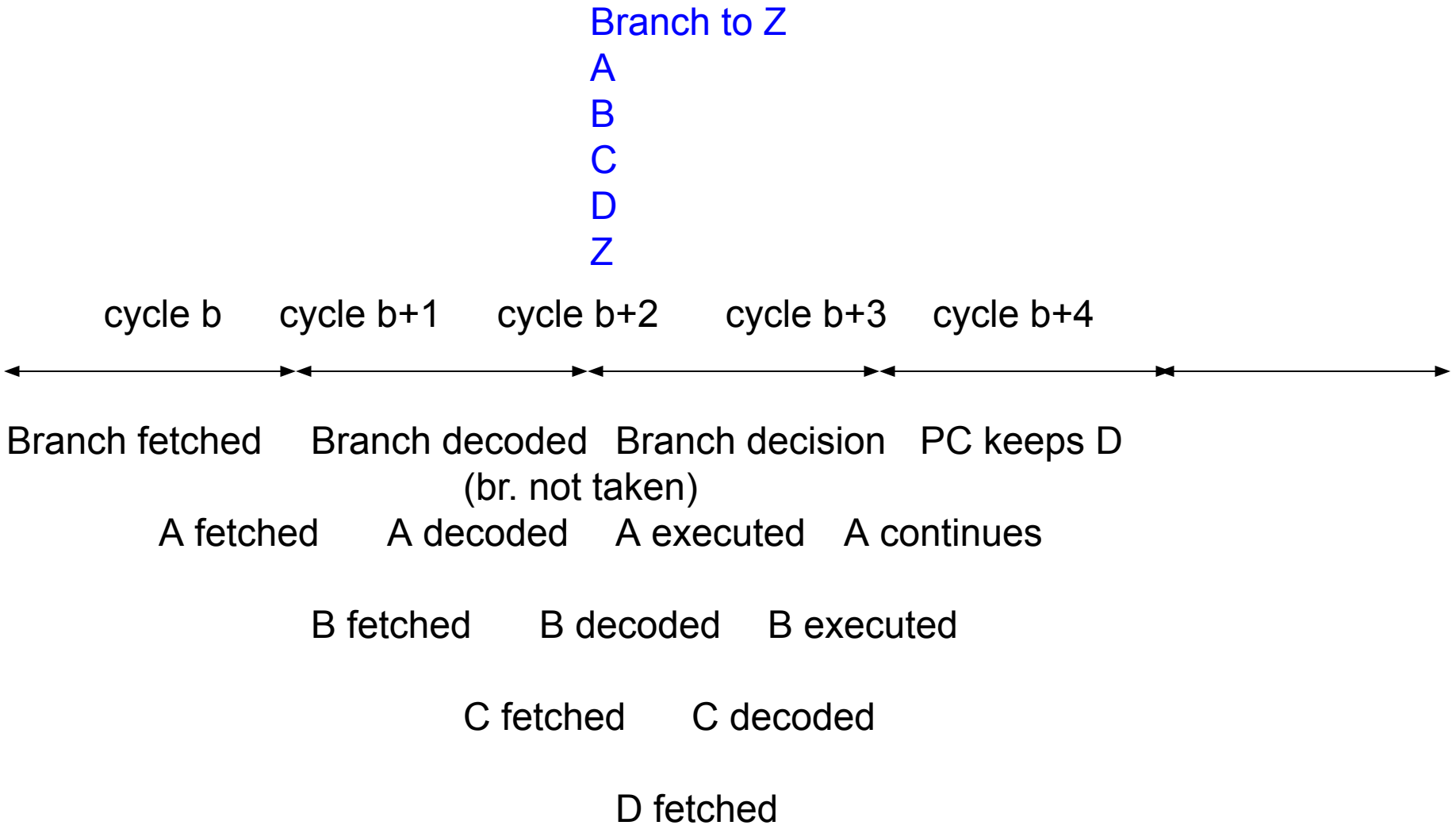
Delayed Branch



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 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 (br. taken) | Branch decision | PC gets Z

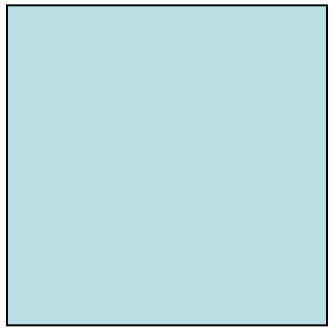
A fetched | A decoded | A executed | Nop

B fetched | B decoded | Nop

C fetched | Nop

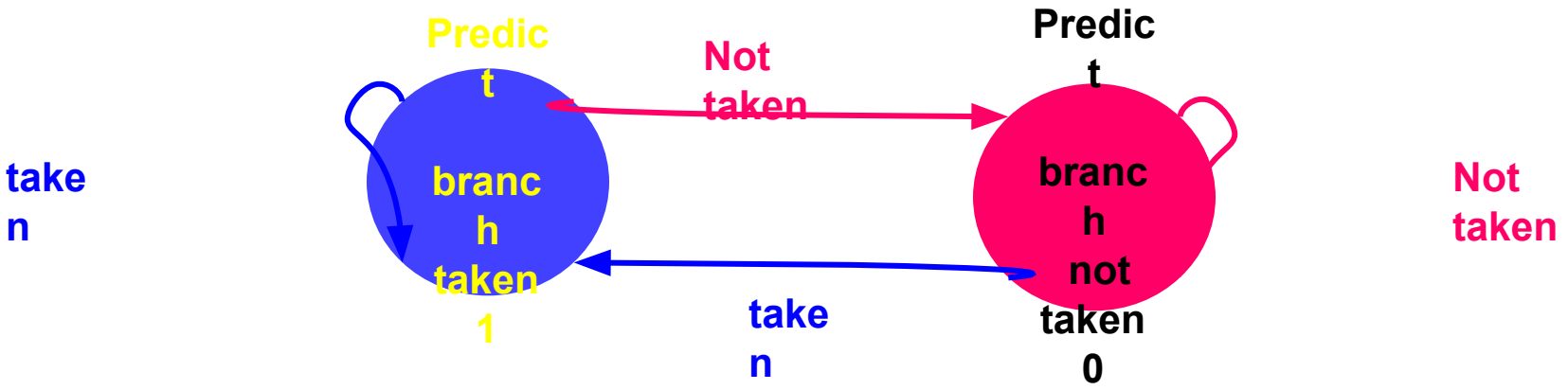
Z fetched

Three instructions are flushed if branch is taken

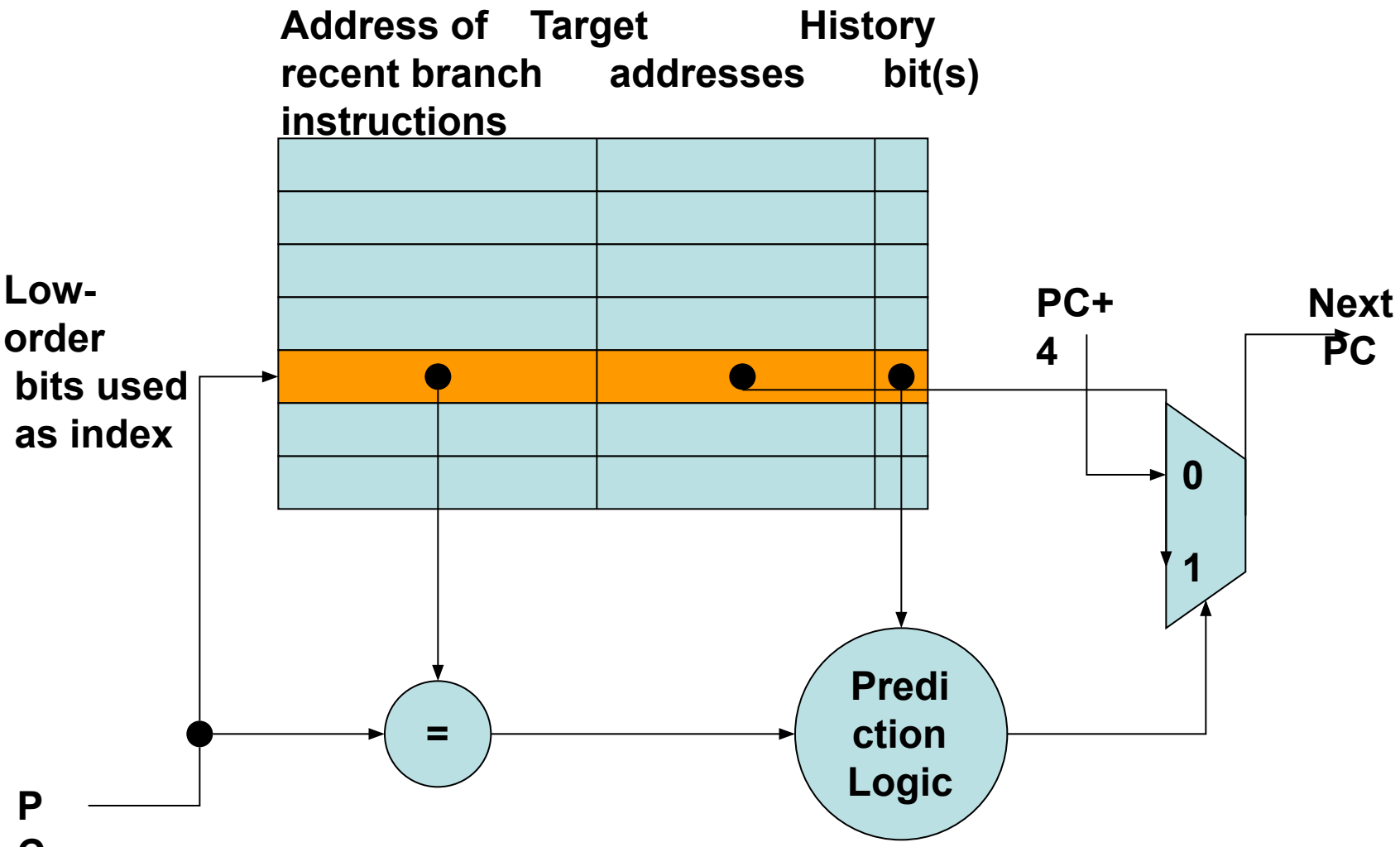


Branch Prediction

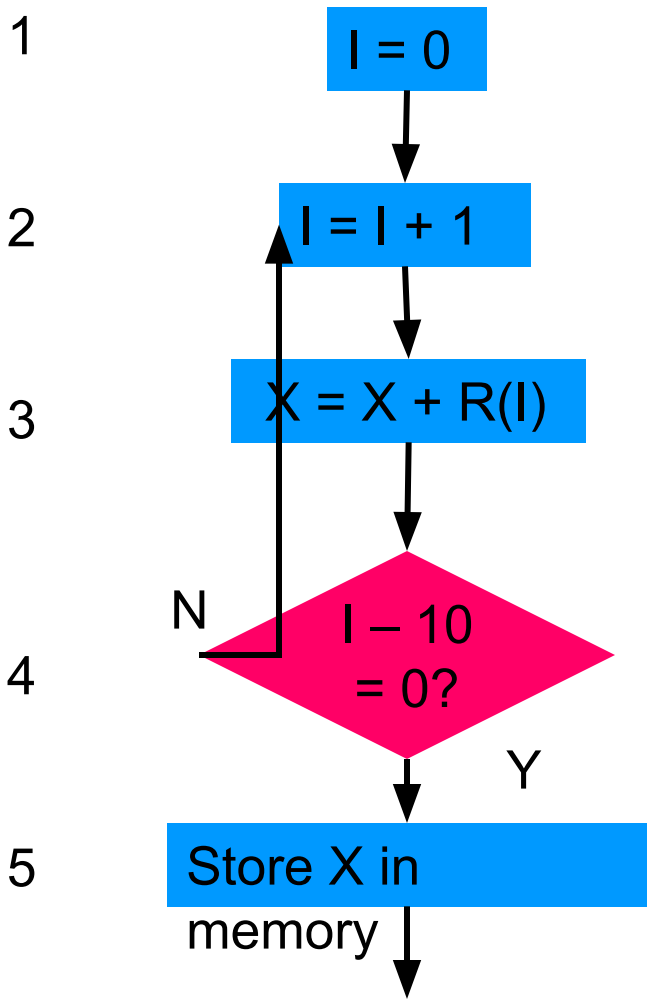
- Useful for program loops.
- A one-bit prediction scheme: a one-bit buffer carries a “history bit” that tells what happened on the last branch instruction
  - History bit = 1, branch was taken
  - History bit = 0, branch was not taken



Branch Prediction



Branch Prediction for a Loop



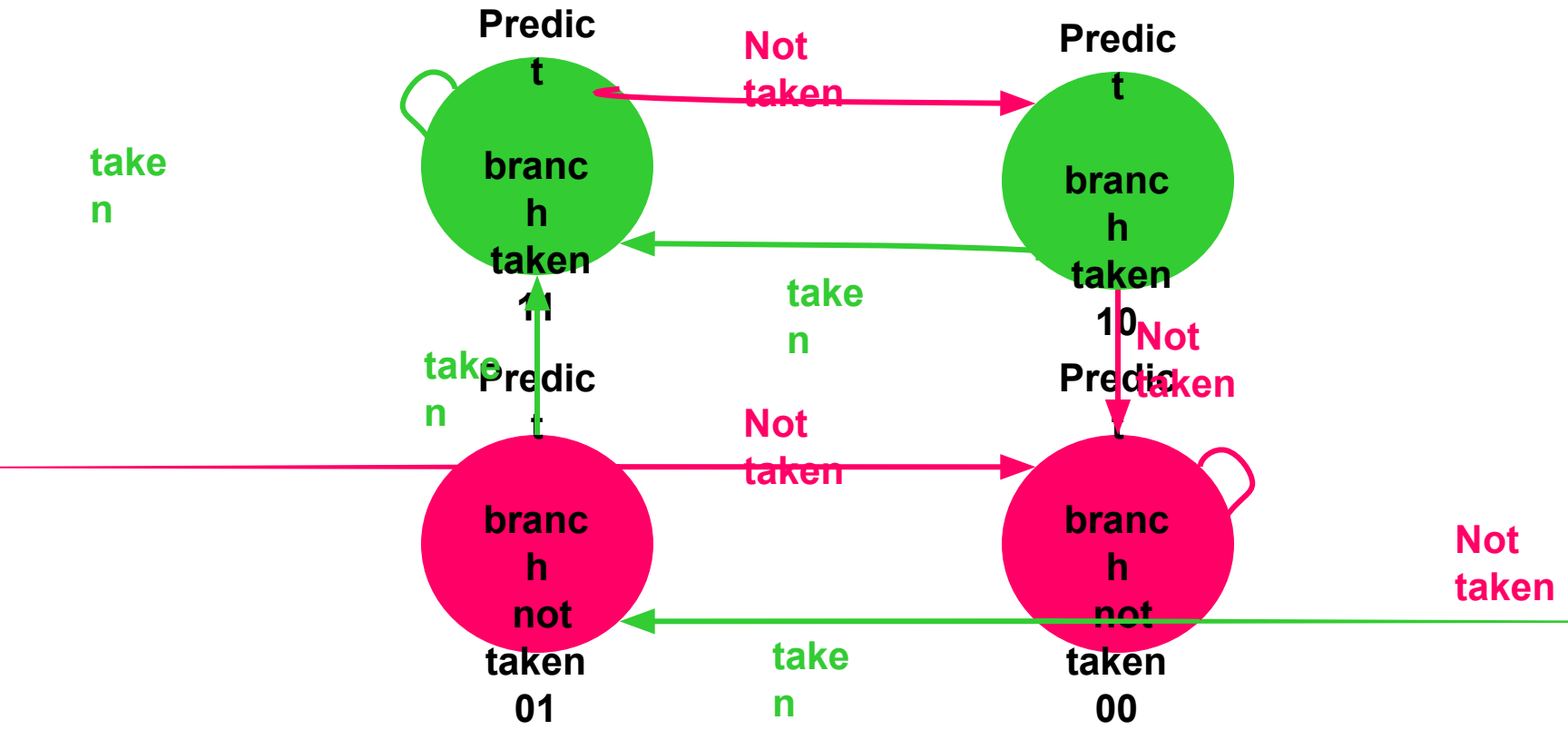
Execution of Instruction 4

Execu-tion seq.	Old hist. bit	Next instr.			New hist. bit	Predi ction
		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

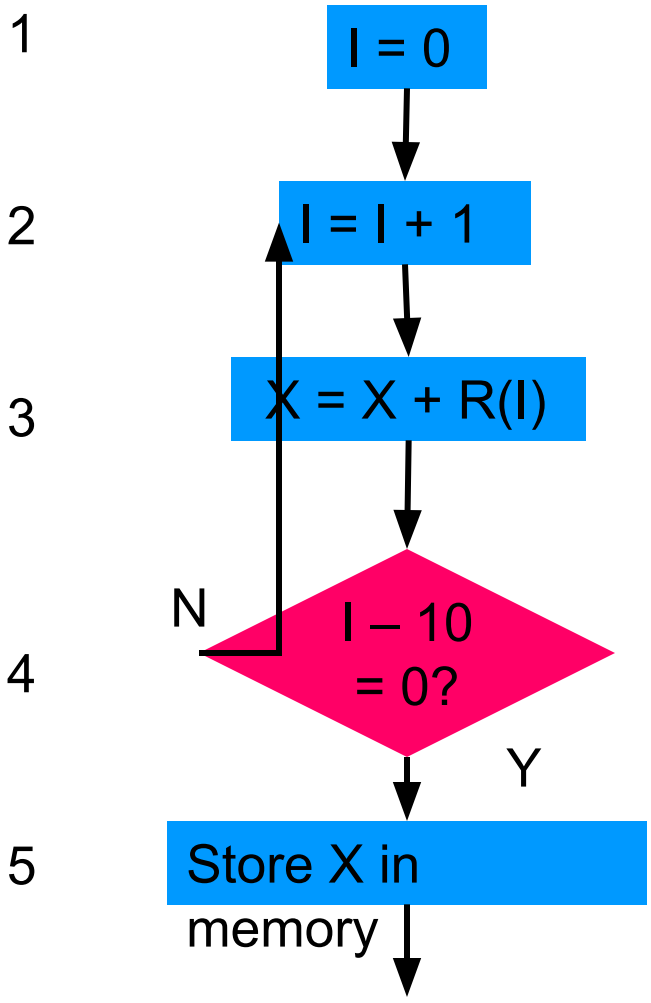
h.bit = 0 *branch not taken*, h.bit = 1 *branch taken*.

Two-Bit Prediction Buffer

- Can improve correct prediction statistics.



Branch Prediction for a Loop



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

Limits of Pipelining

- IBM RISC Experience
  - Control and data dependences add 15%
  - Best case CPI of 1.15, IPC of 0.87
  - Deeper pipelines (higher frequency) magnify dependence penalties
- This analysis assumes 100% cache hit rates
  - Hit rates approach 100% for some programs
  - Many important programs have much worse hit rates

Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

- In the 1980's (decade of pipelining):
  - CPI: 5.0 => 1.15
- In the 1990's (decade of superscalar):
  - CPI: 1.15 => 0.5 (best case)
- In the 2000's (decade of multicore):
  - Marginal CPI improvement