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

CP-226: Computer Architecture

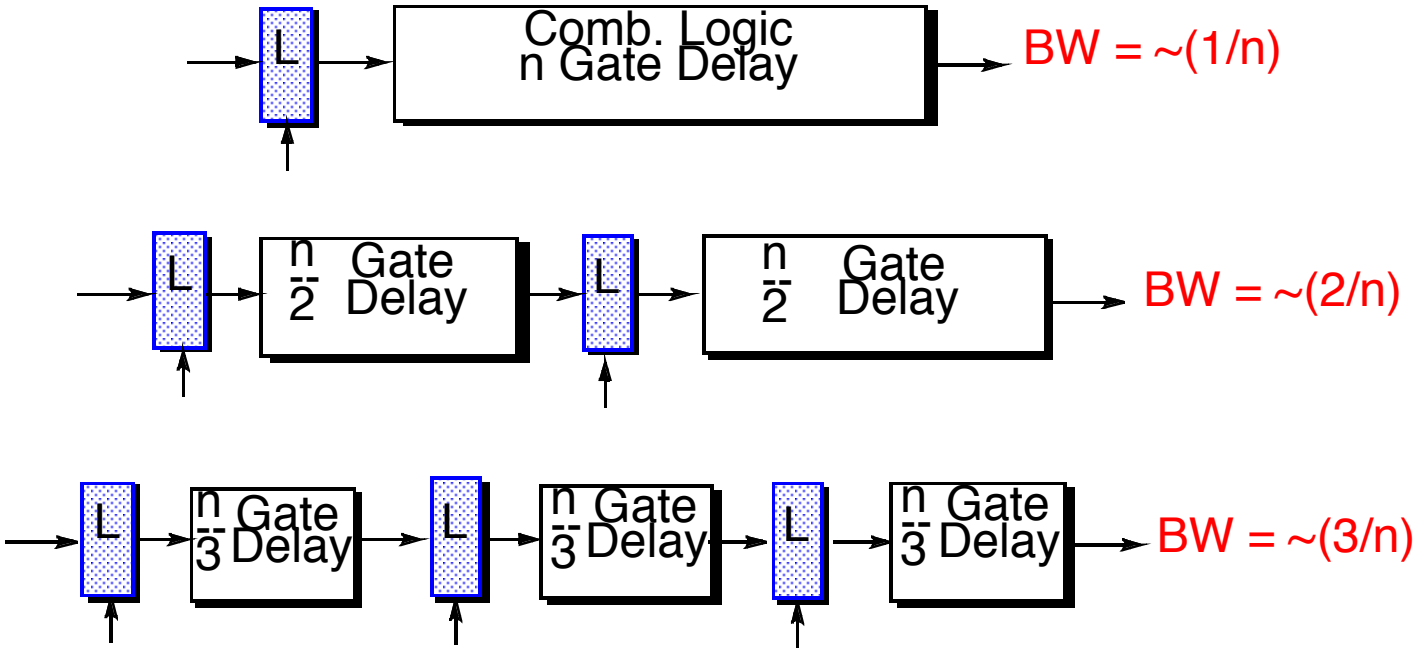
Lecture 12 (27 Feb 2013)

CADSL

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.

Ideal Pipelining

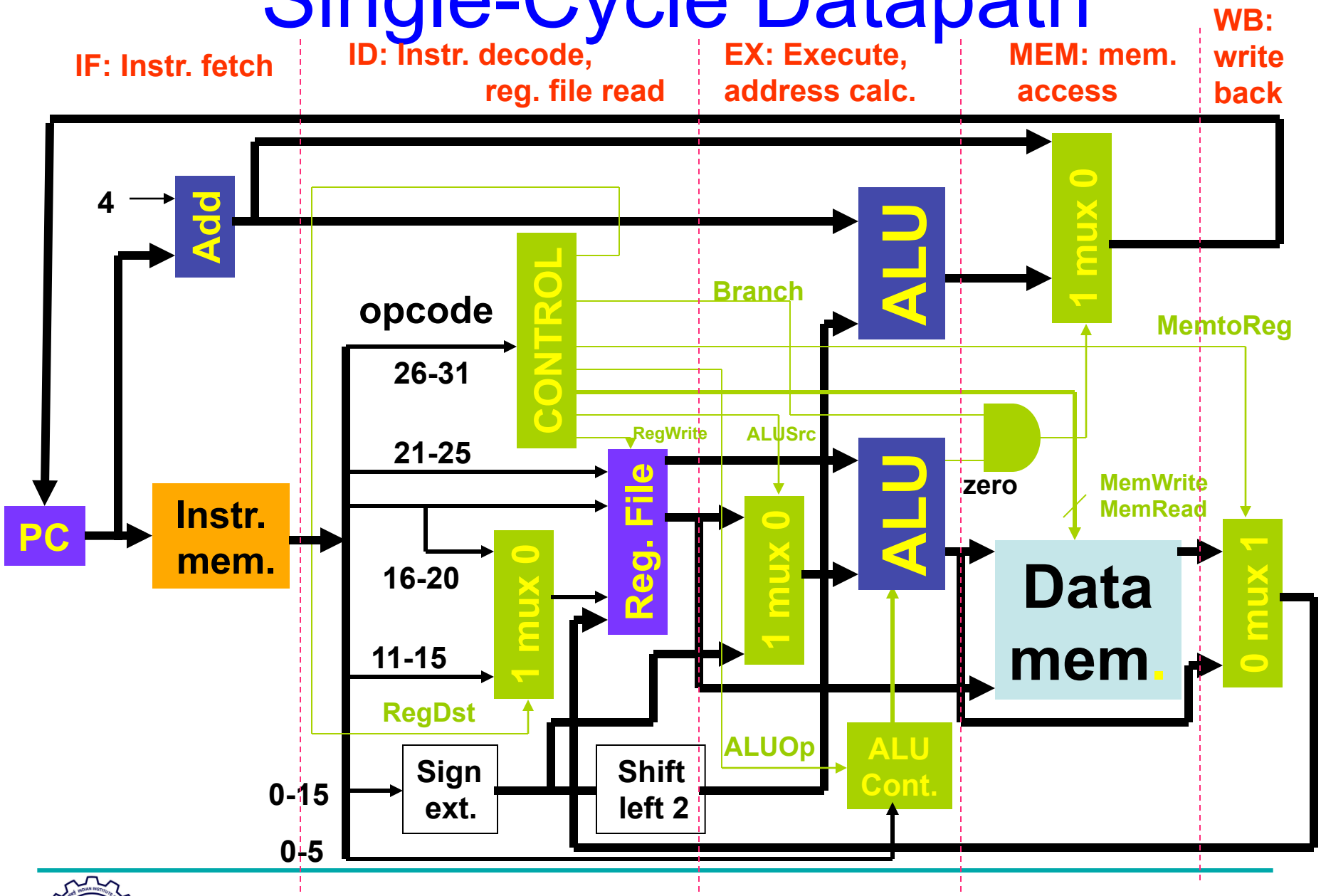


- Bandwidth increases linearly with pipeline depth
- Latency increases by latch delays

Pipelining Idealisms

- Uniform subcomputations
  - Can pipeline into stages with equal delay
  - Balance pipeline stages
- Identical computations
  - Can fill pipeline with identical work
  - Unify instruction types
- Independent computations
  - No relationships between work units
- Are these practical?
  - No, but can get close enough to get significant speedup

Single-Cycle Datapath

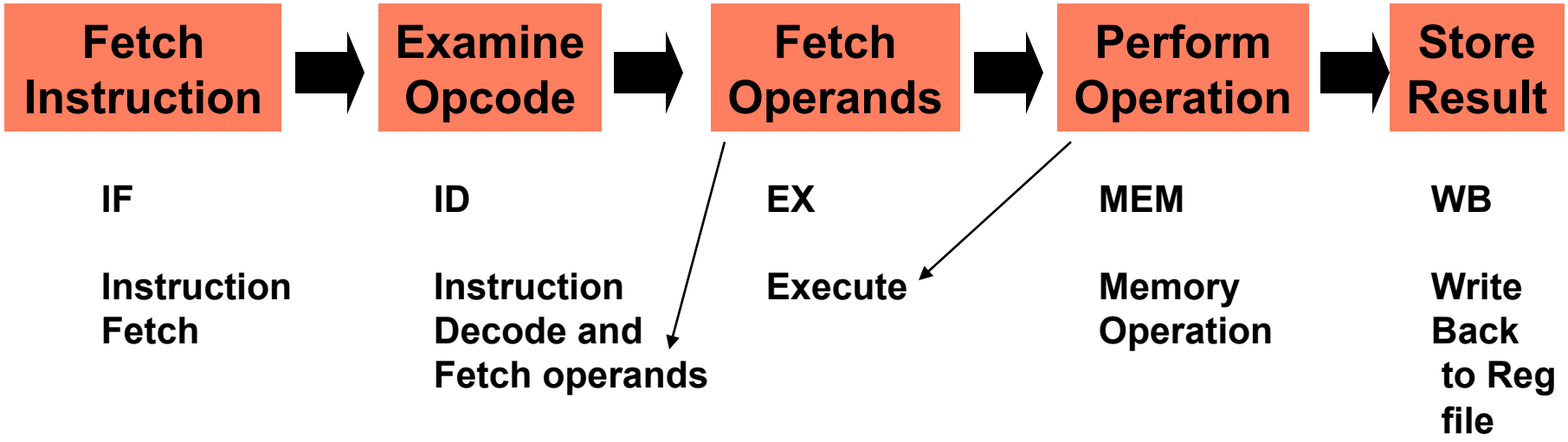


Pipelined Datapath

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.

Pipelining of RISC Instructions



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

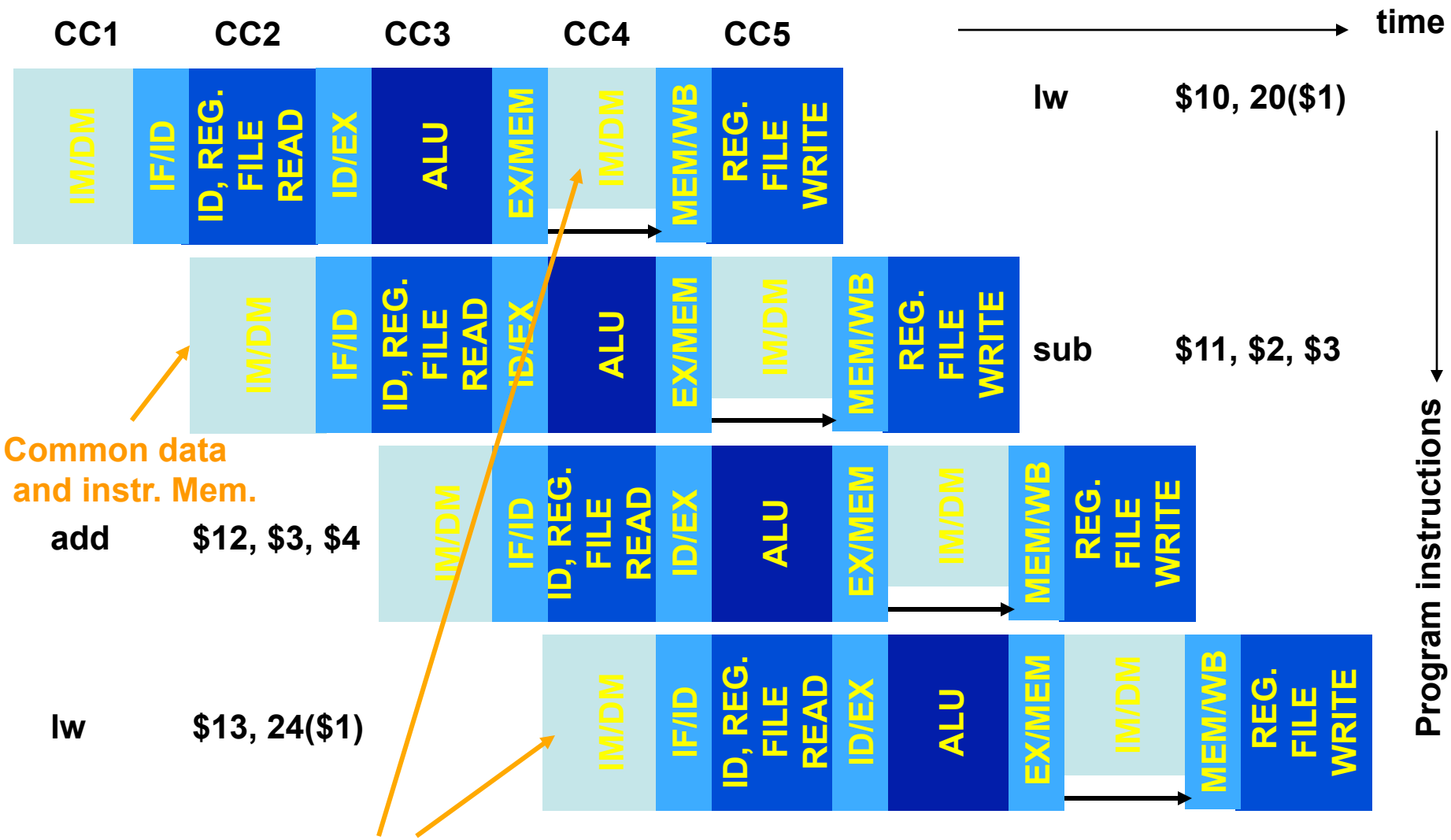
Pipeline Hazards

- 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

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.

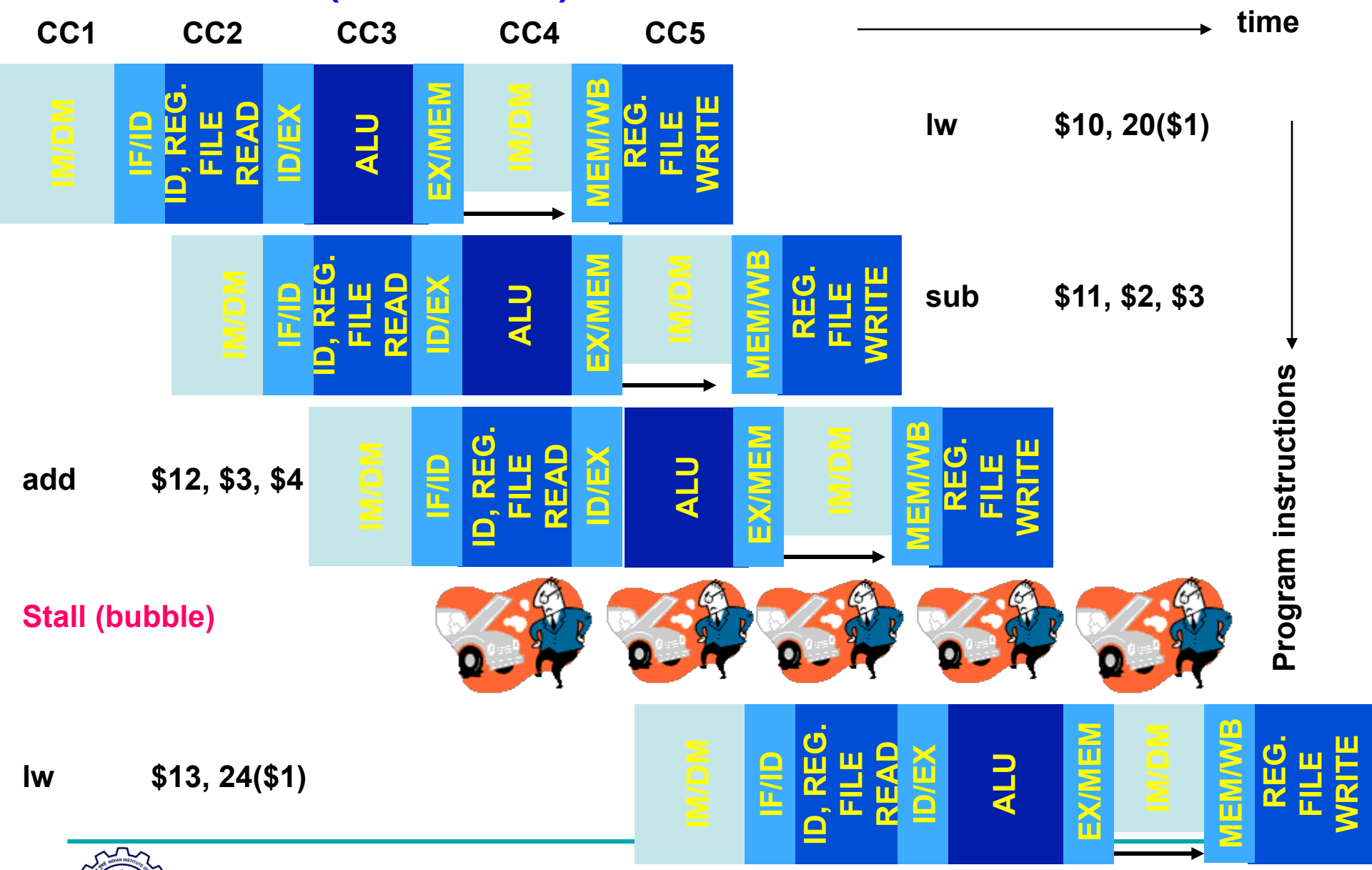
Example of Structural Hazard



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*.

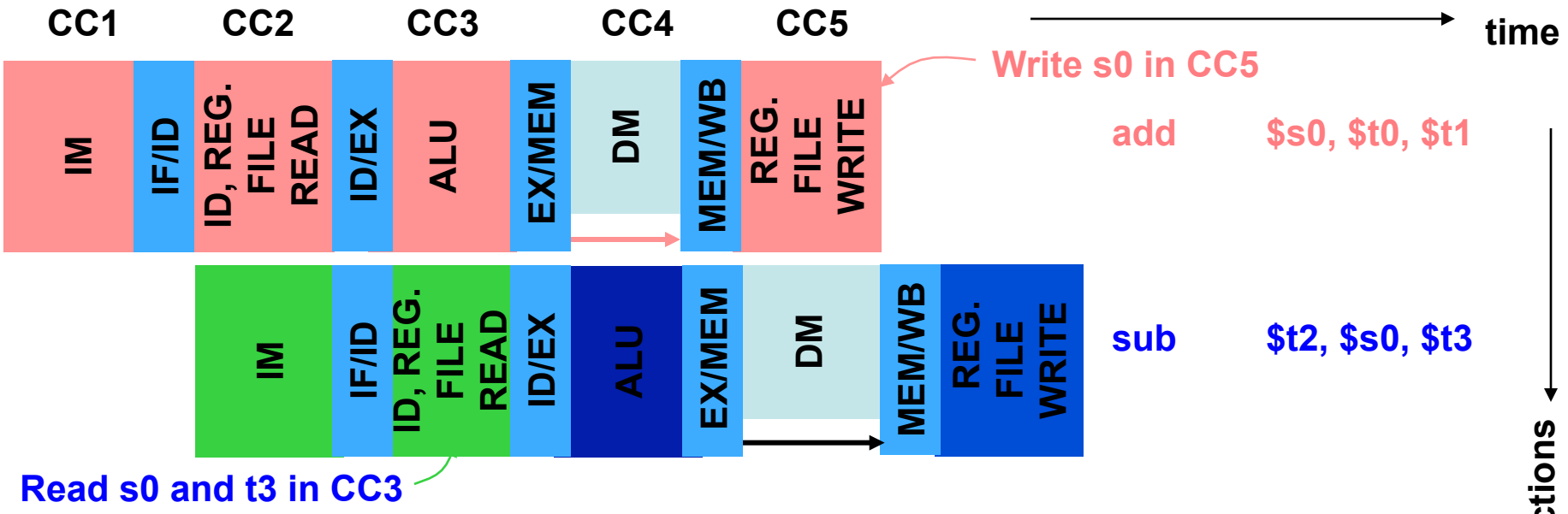
Stall (Bubble) for Structural Hazard



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



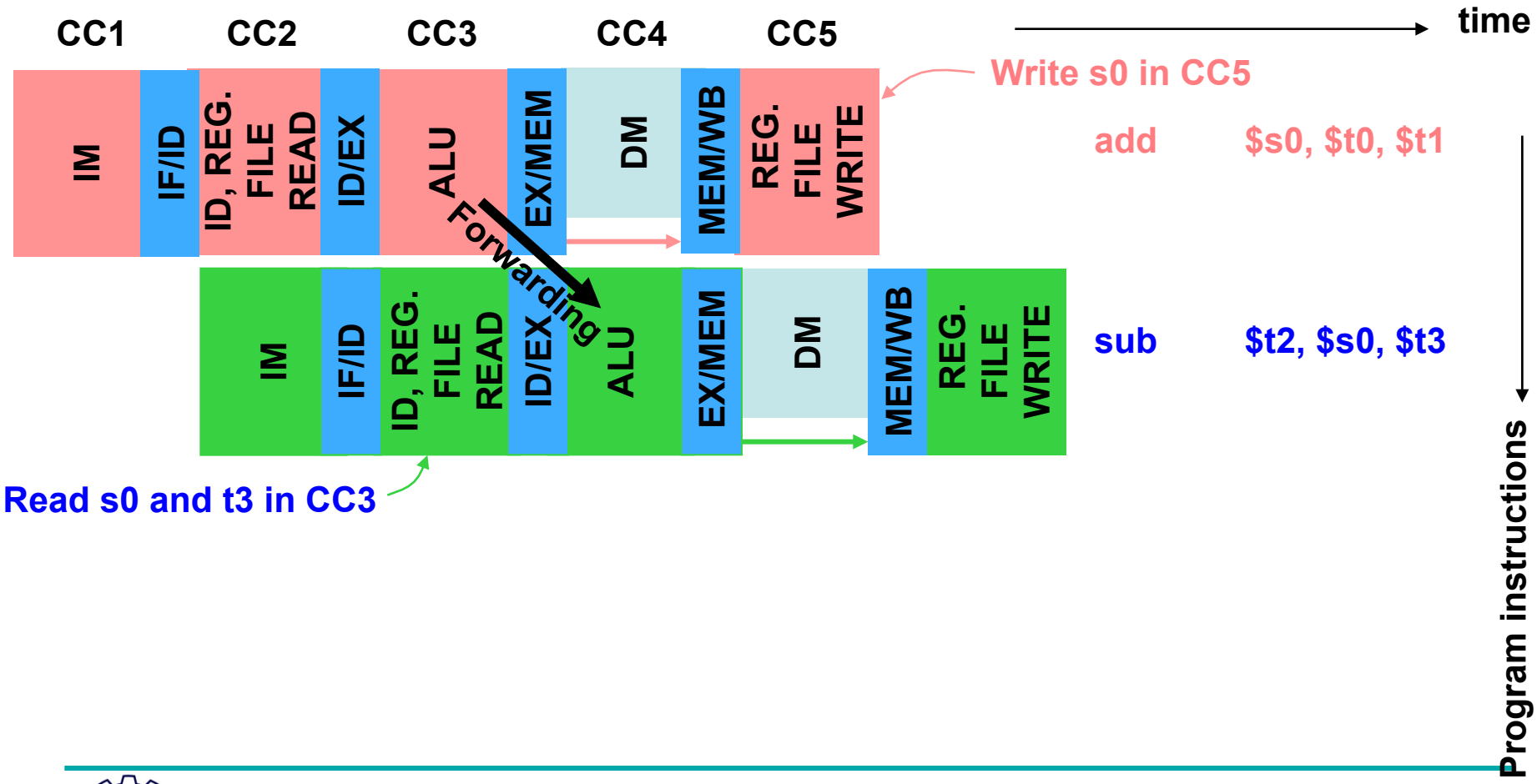
We need to read s0 from reg file in cycle 3
But s0 will not be written in reg file until cycle 5

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

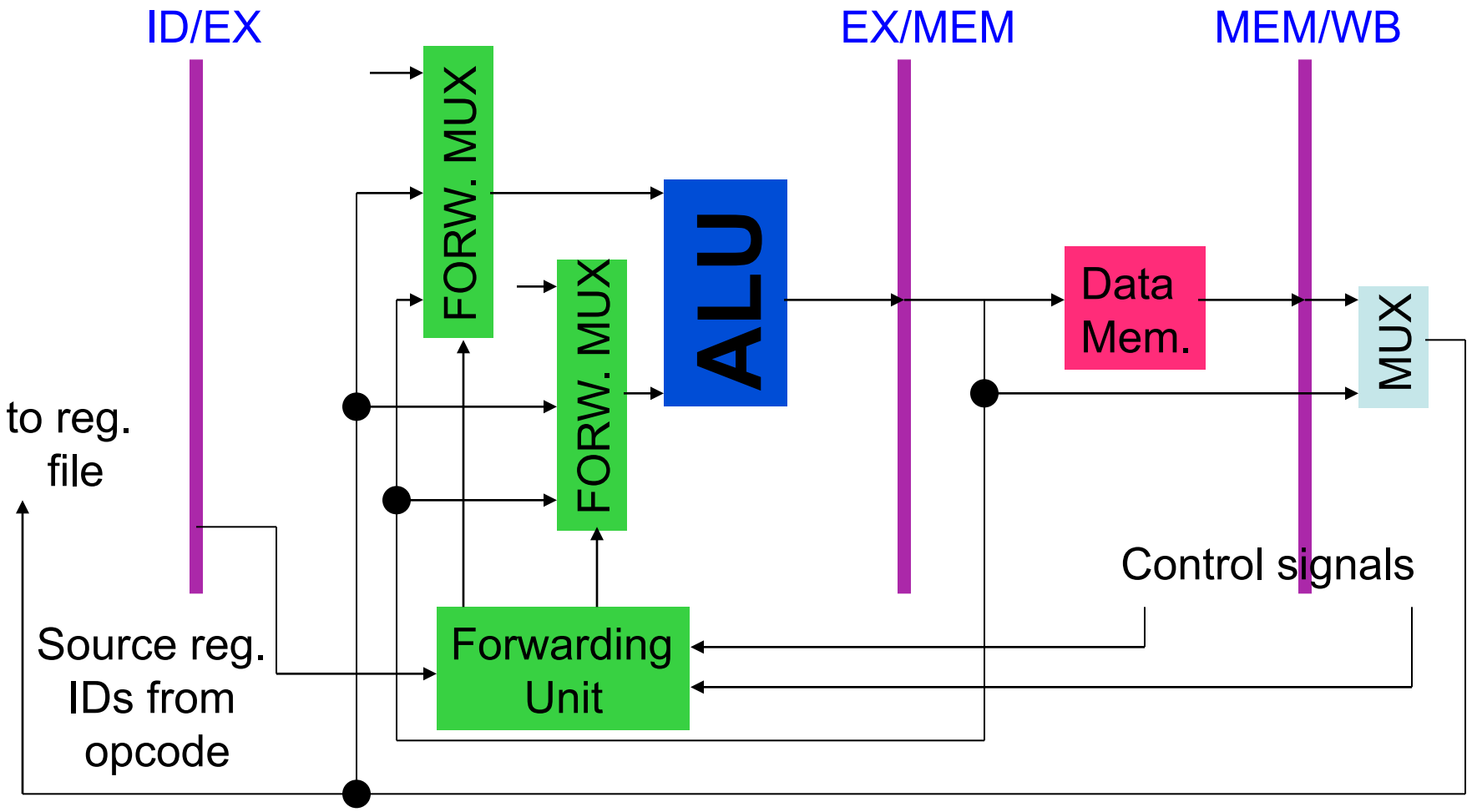
Forwarding or Bypassing

- 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.

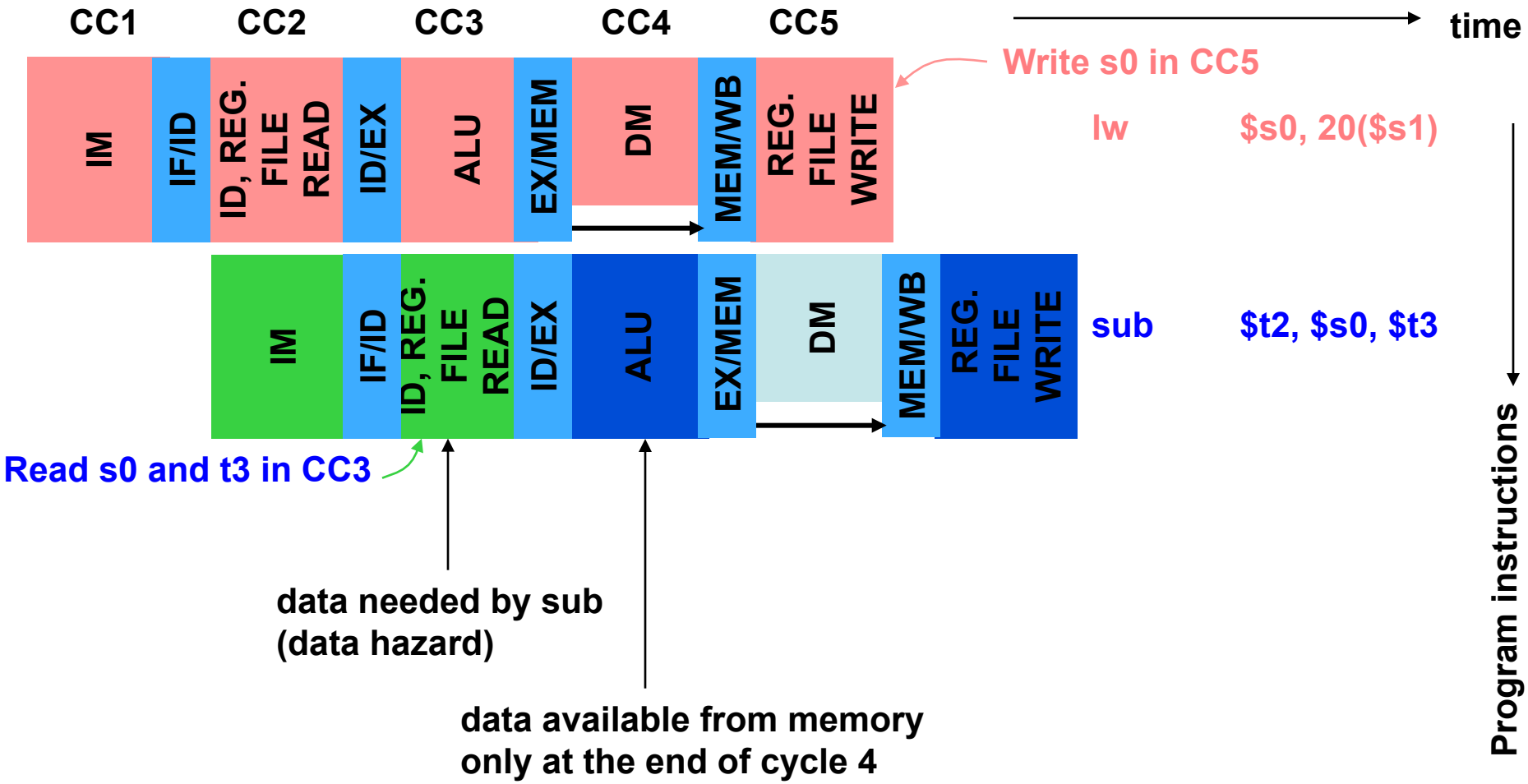
Forwarding for Data Hazard



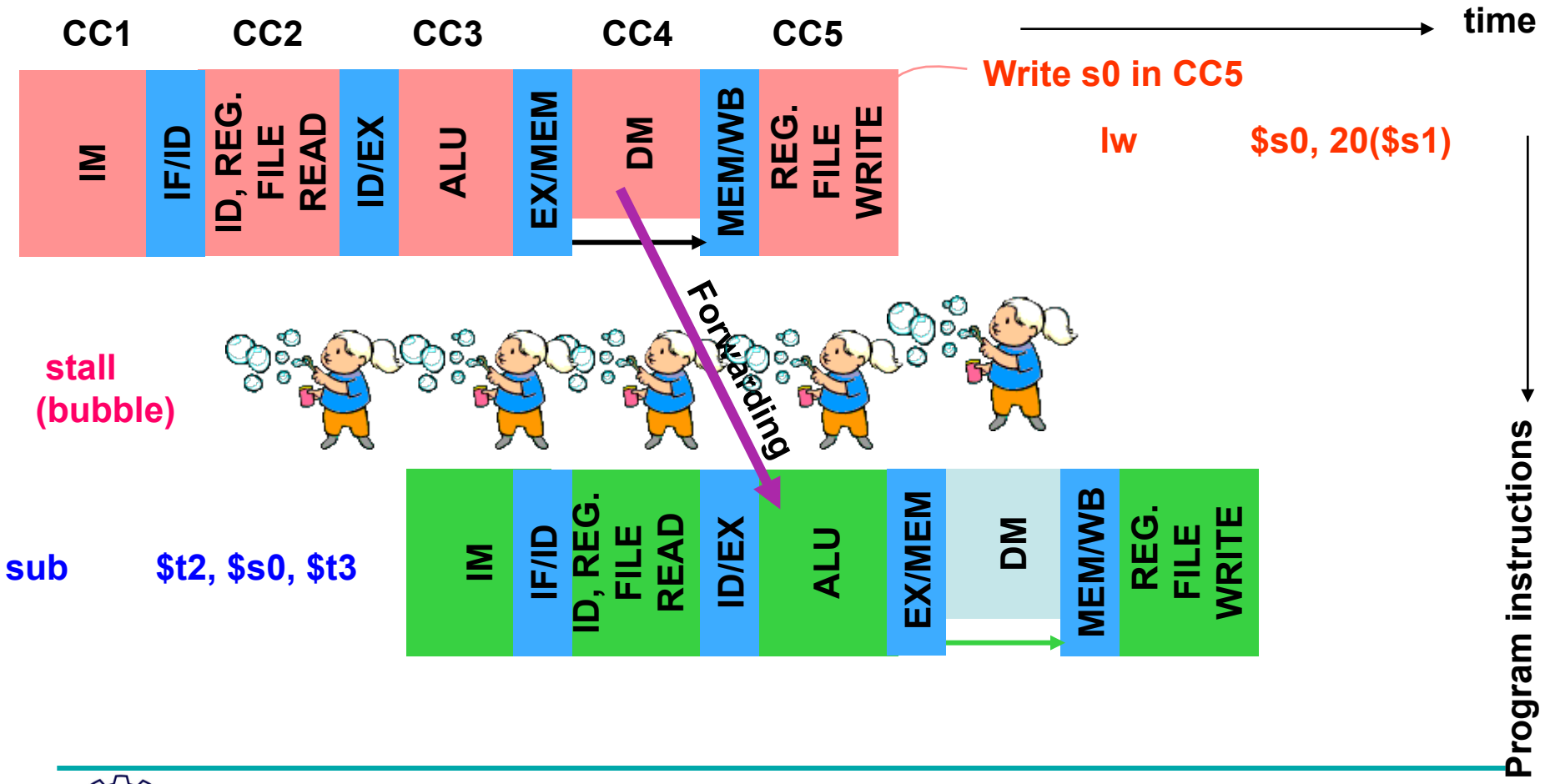
Forwarding Unit Hardware



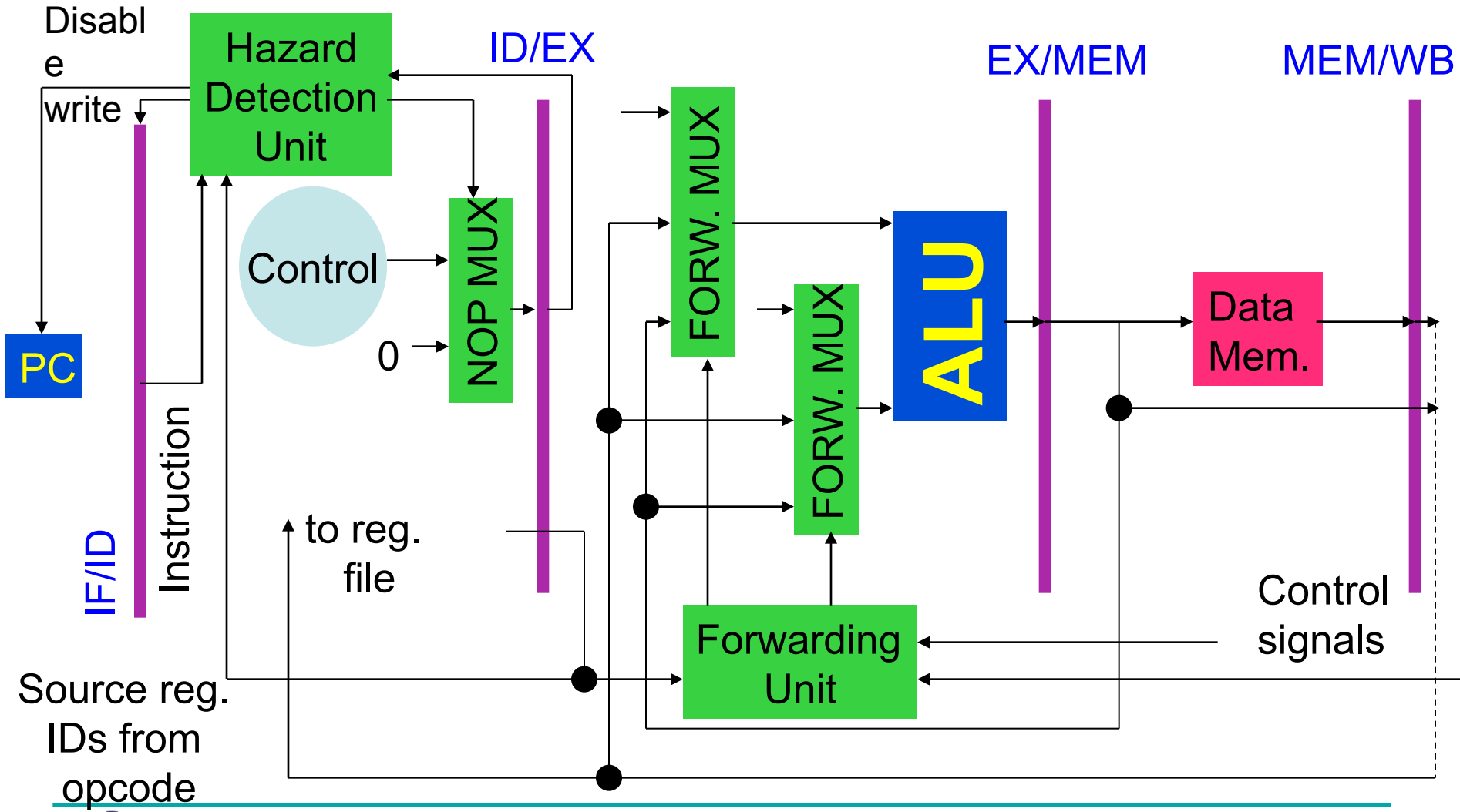
Forwarding Alone May Not Work



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.

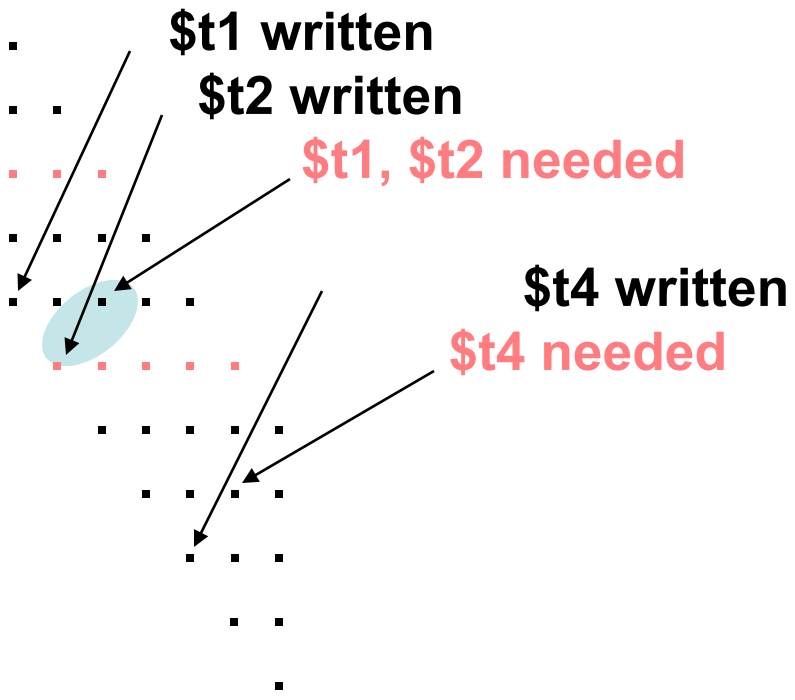
Avoiding Stall by Code Reorder

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)
```

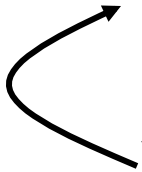


Reordered Code

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)
```

Thank You