Dependency and hazard

Presented By

Dr. Banchhanidhi Dash

School of Computer Engineering
KIIT University

Dependencies and Hazards

Incorrect execution due to...

... *dependencies* in program...

...and *hazards* in hardware (pipeline).

Incorrect execution above is the "fault" of the hardware...

...because the ISA does not forbid dependencies.

Dependency:

A relationship between two instructions ...

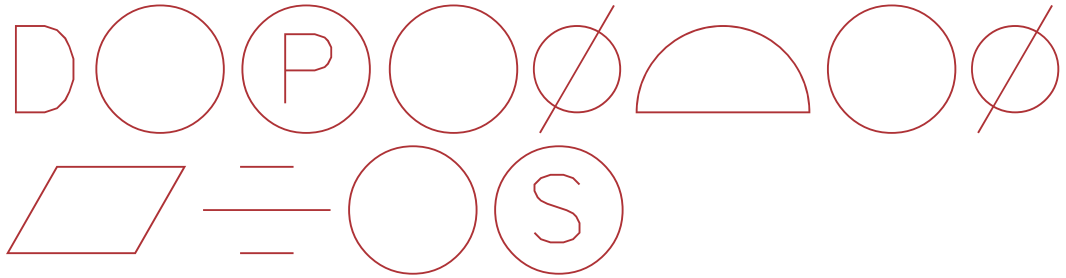
...indicating that their execution should be (or appear to be) in program order.

Hazard:

A potential execution problem in an implementation due to overlapping instruction execution.

There are several kinds of dependencies and hazards.

For each kind of dependence there is a corresponding kind of hazard.



Dependency:

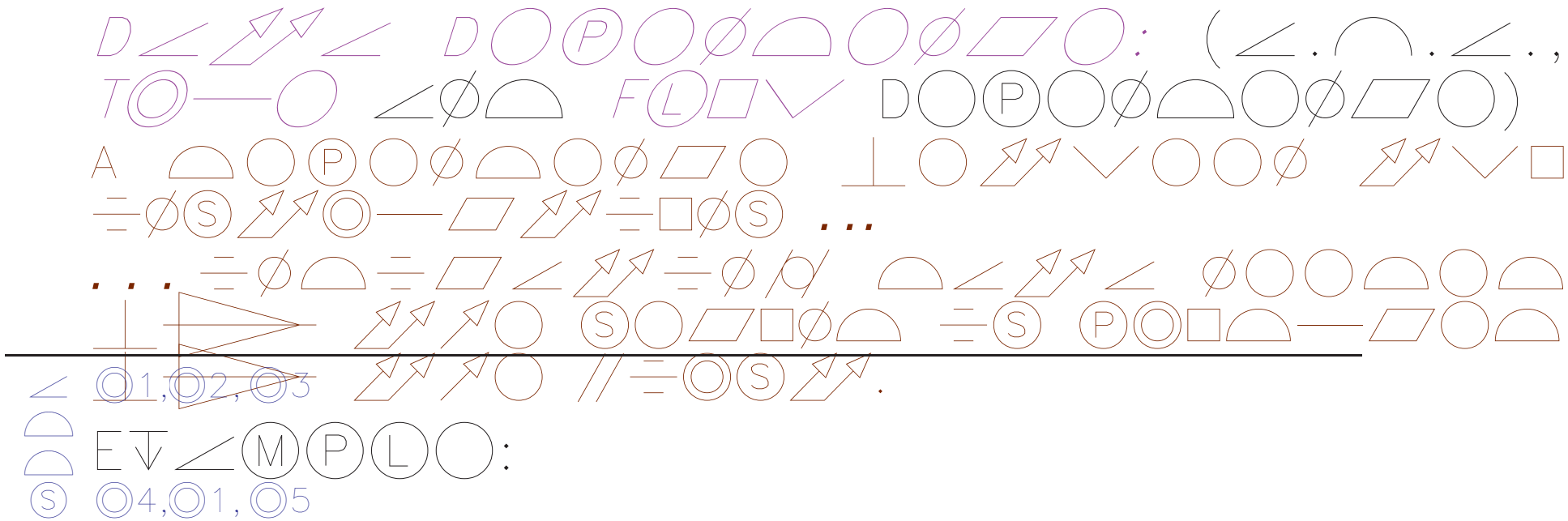
A relationship between two instructions ...

...indicating that their execution should be, or appear to be, in program order.

If *B* is dependent on *A* then *B* should appear to execute after *A*. Dependency Types:

- *True, Data, or Flow Dependence* (Three different terms used for the same concept.)
- *Name Dependence*
- *Control Dependence*

Data Dependence



The sub is dependent on add(via r1).

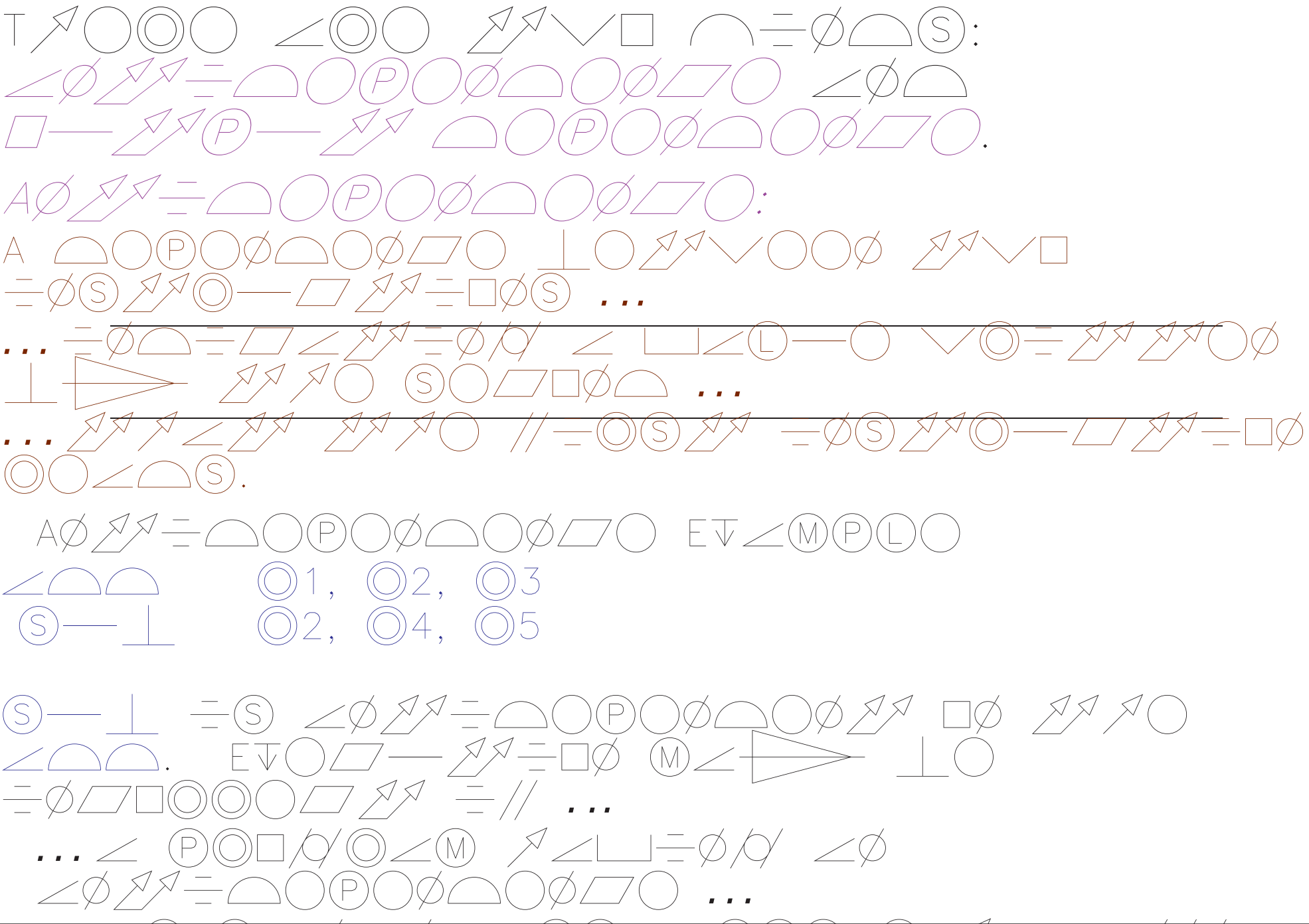
The and is dependent on sub(via r4). The and is dependent add(via sub).

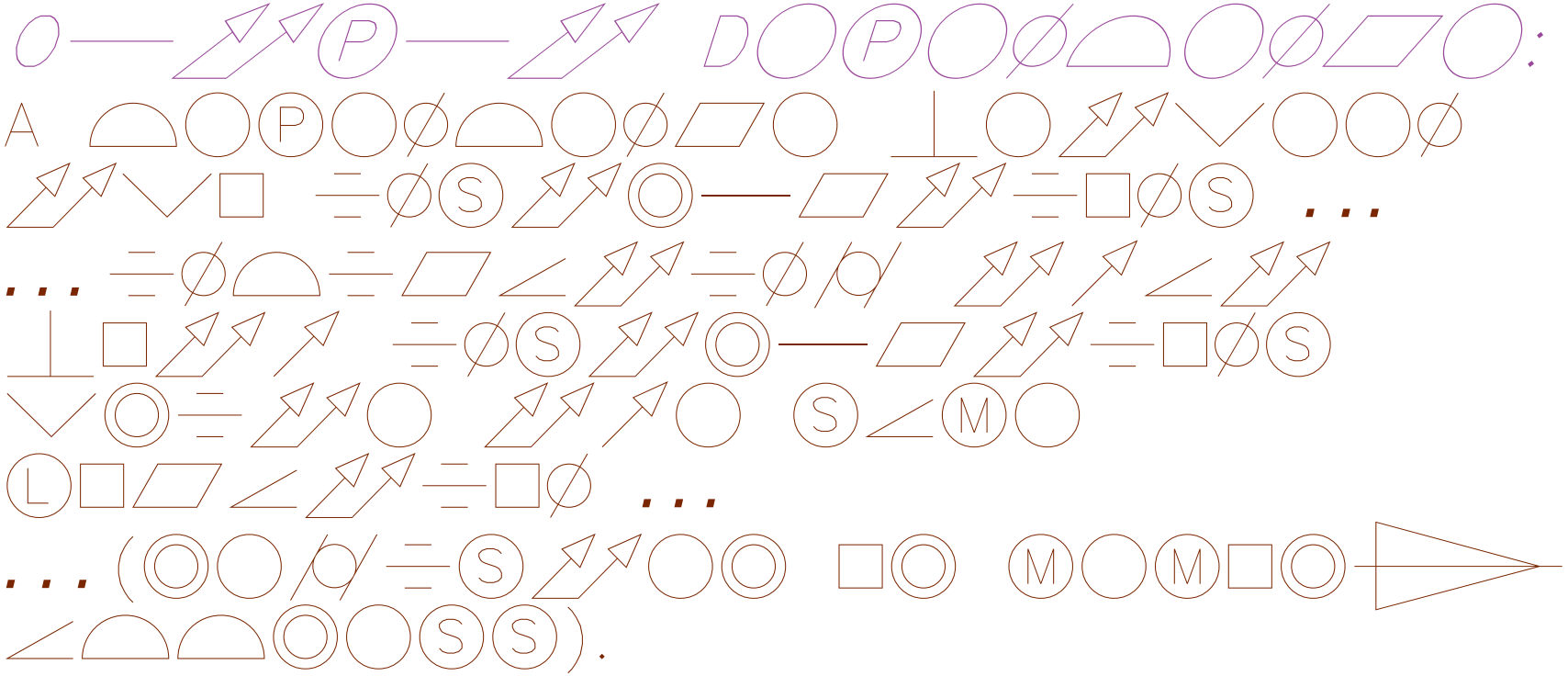
Execution may be incorrect if ...

... a program having a data dependence ...

... is run on a processor having an uncorrected RAW hazard.

Name Dependencies





control Dependency

An instruction B has a control dependency on a preceding instruction A if the outcome of A determines wheather B should be executed or not.

Example:

```
A:    BEQ  $s1, $s2,Label
B:    add $s3,$s4,$s5
C:    Label:  sub $t0,$t1,$t2
```

Pipeline Hazards

What is a pipeline hazard?

A situation that prevents an instruction from executing during its designated clock cycles.

There are 3 classes of hazards:

Structural Hazards

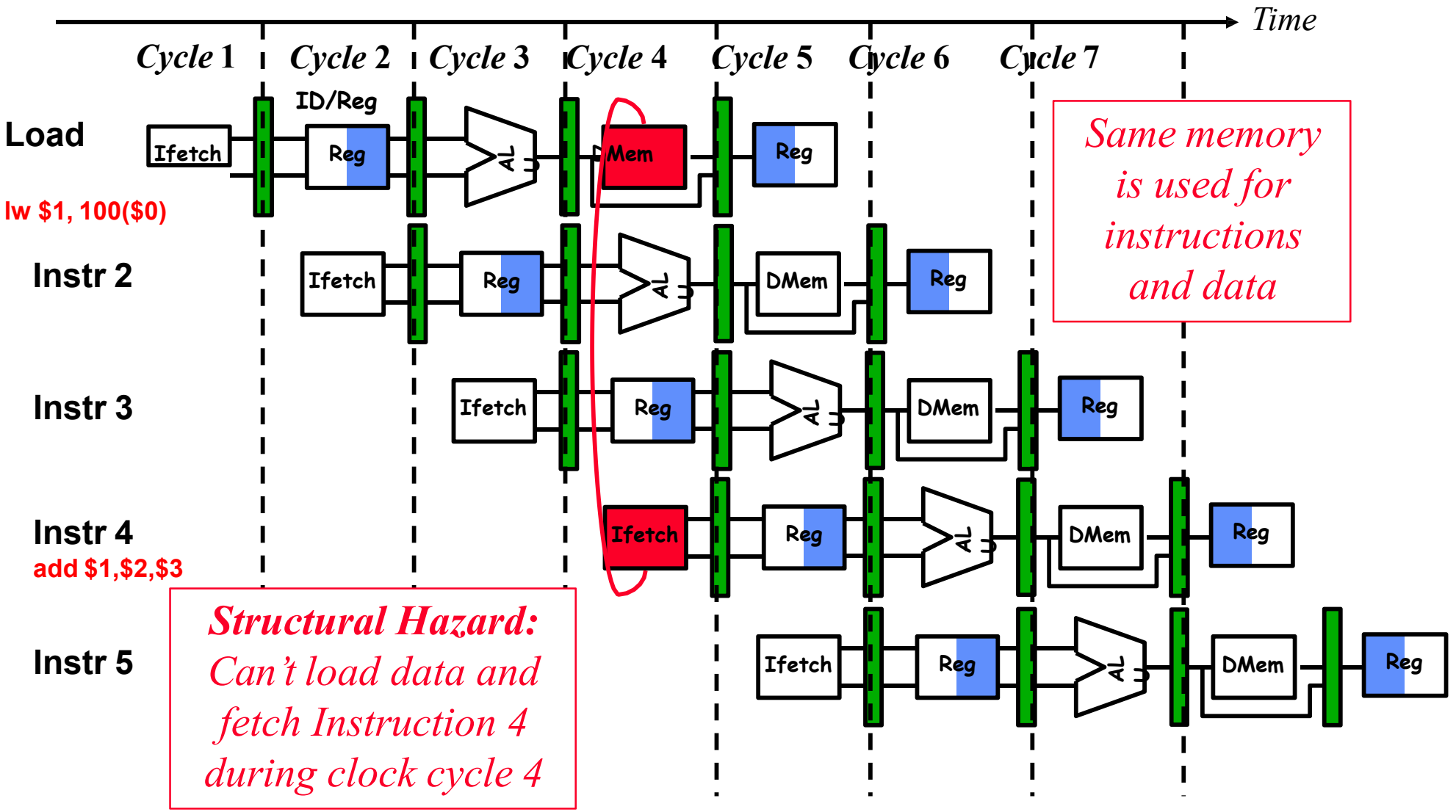
data hazard

Control Hazards

Pipeline Hazards

- Hazards can result in incorrect operations:
  - Structural hazards: Two instructions requiring the same hardware unit at same time.
  - Data hazards: Instruction depends on result of a prior instruction that is still in pipeline
    - Data dependency
  - Control hazards: Caused by delay in decisions about changes in control flow (branches and jumps).
    - Control dependency

Structural Hazard - Conflict due to Memory Access

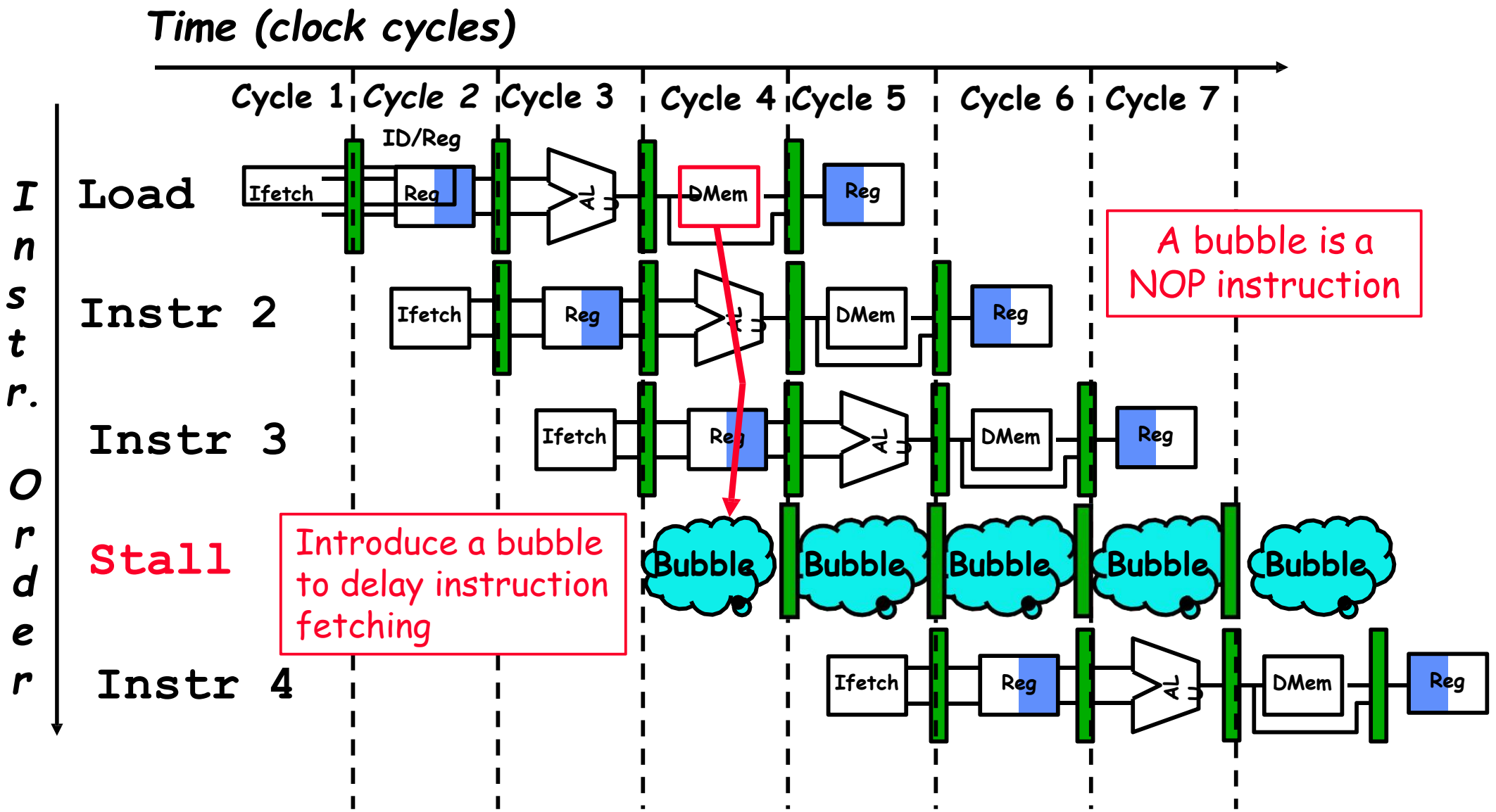


Resolving structural hazards

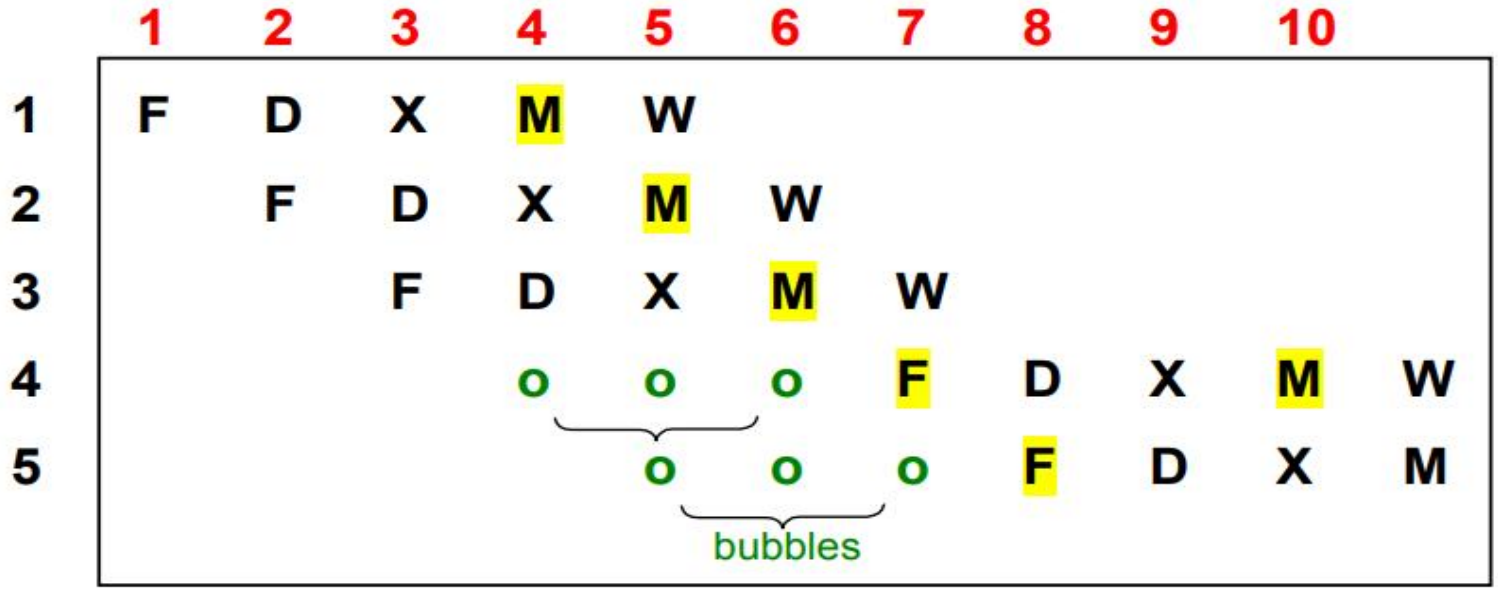
- Problem
  - Attempt to use the same hardware resource (Memory) by two different instructions during the same cycle
- Solution 1: Wait
  - Must detect the hazard
  - Must have mechanism to delay (stall) instruction access to resource (Introduce bubble / NOP)
  - Serious: hazard cannot be ignored
- Solution 2: Redesign the pipeline
  - Add more hardware to eliminate the structural hazard
  - In our example: use two memories with two memory ports
    - Instruction Memory
    - Data Memory

} Can be implemented as caches

Solution 1 : Detect Structural Hazard and Delay



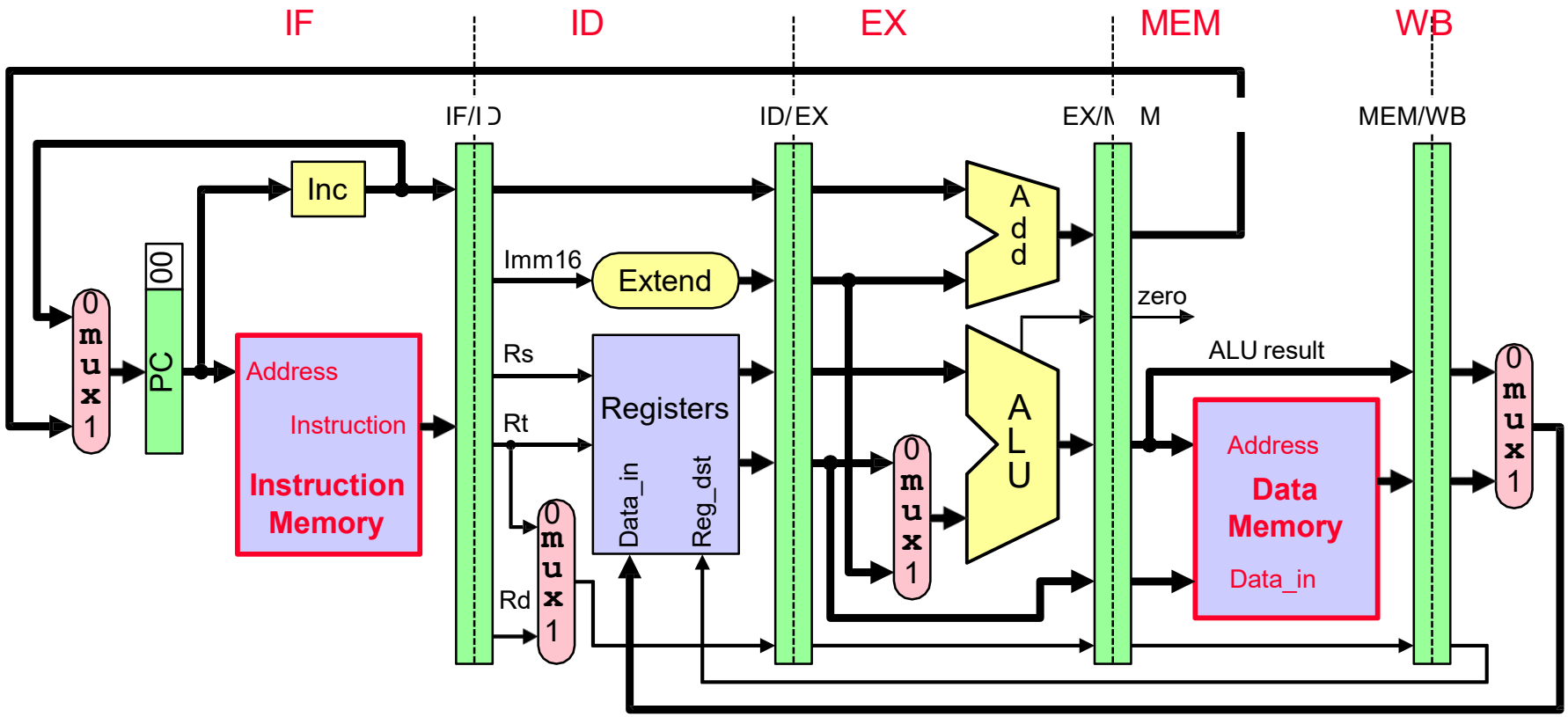
Can we fetch an instruction (F) and read data from the memory at the same time? For a single port memory, the answer is NO.



These bubbles (stall cycles) will appear if we use a single-port memory to store both instructions and data.

Solution 2: Add More Hardware
(Use Instruction and data memory)

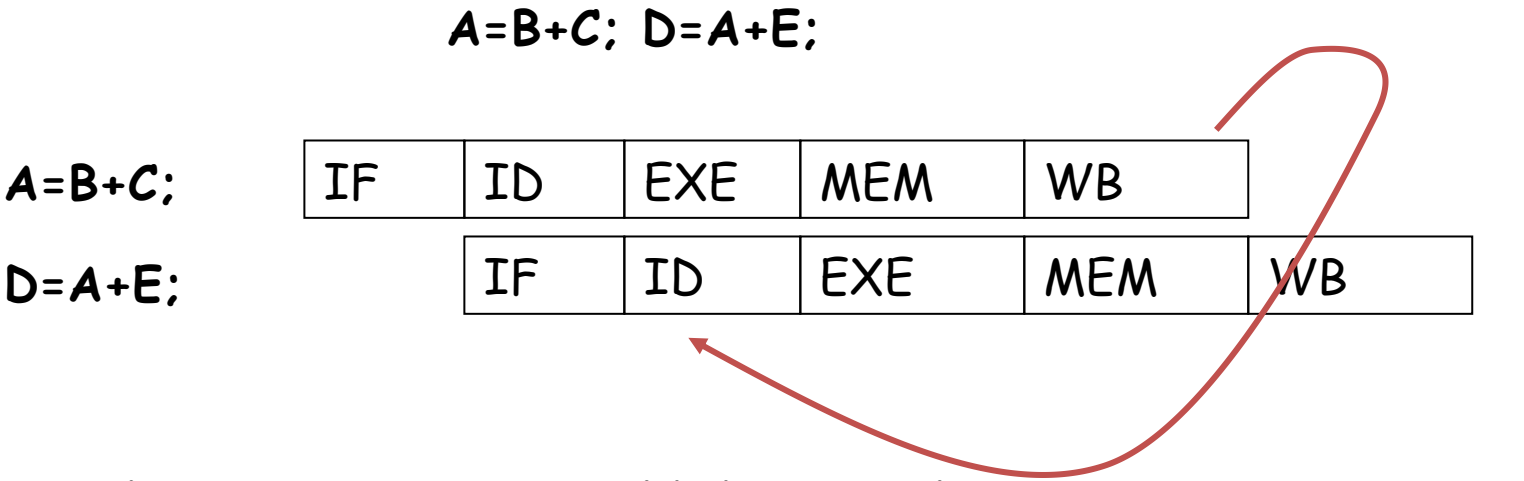
- Eliminate structural hazard at design time
- Use two separate memories with two memory ports
  - Instruction and data memories can be implemented as caches



Data Hazards

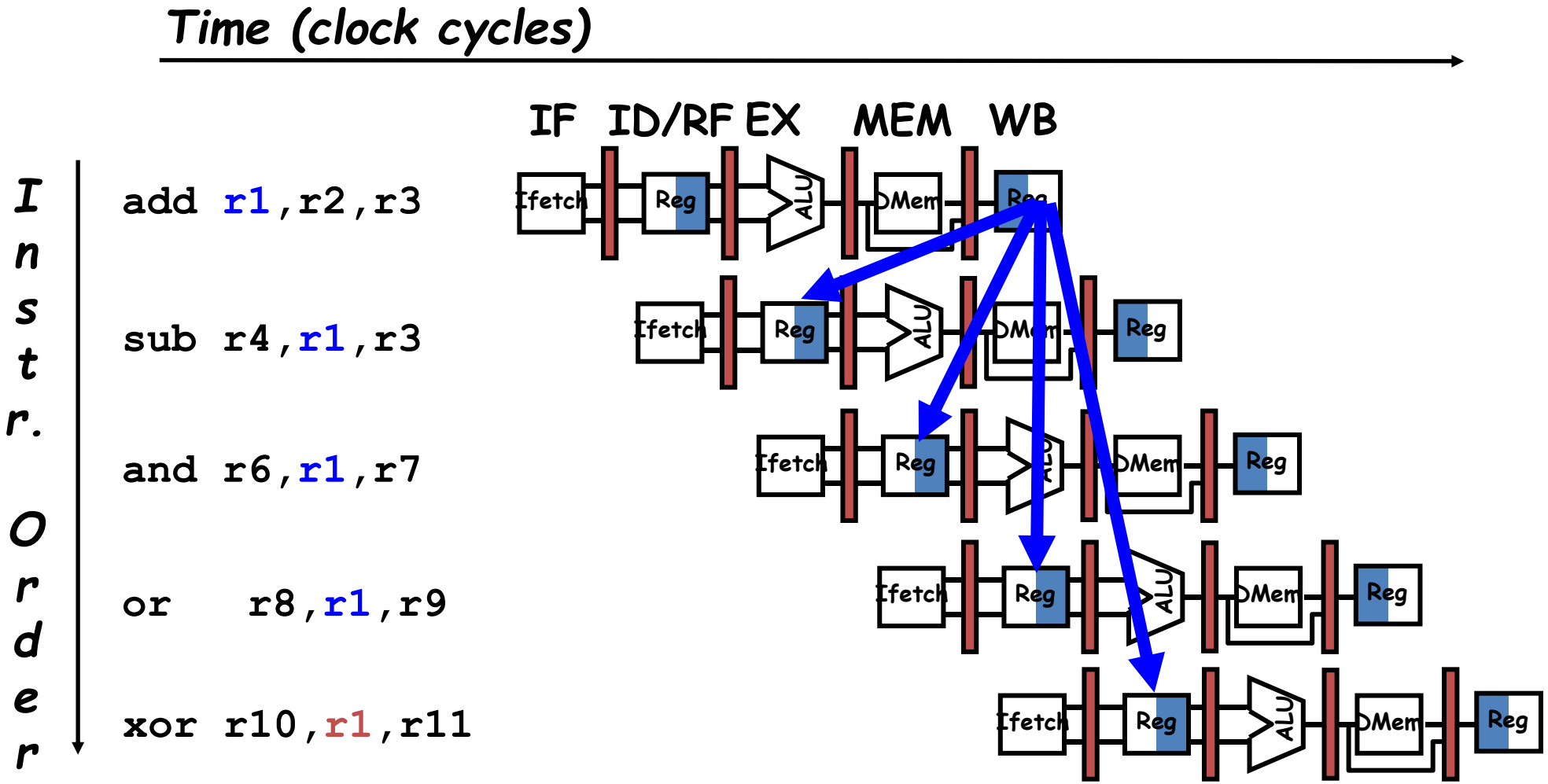
Occur when an instruction under execution depends on: Data from an instruction ahead in pipeline.

Example:



Dependent instruction uses old data:Results in wrong computations

Data Hazards



The use of the result of the ADD instruction in the next three instructions causes a hazard, since the register is not written until after those instructions read it.

Data hazards

Data hazards occur when instructions that exhibit data dependence modify data in different stages of a pipeline. There are three situations in which a data hazard can occur:

read after write (RAW), a true dependency
write after read (WAR), an anti-dependency
write after write (WAW), an output dependency

Read after read (RAR) is not a hazard case.

Execution Order is:
$Instr_I$
$Instr_J$

Read After Write (RAW)

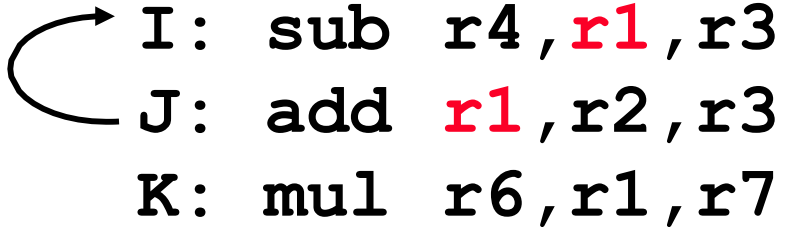
$Instr_J$ tries to read operand before $Instr_I$ writes it

Execution Order is:
$Instr_I$
$Instr_J$

Write After Read (WAR)

$Instr_J$ tries to write operand *before* $Instr_I$ reads i

- **Gets wrong operand**



- **Called an "anti-dependence" by compiler writers. This results from reuse of the name "r1".**

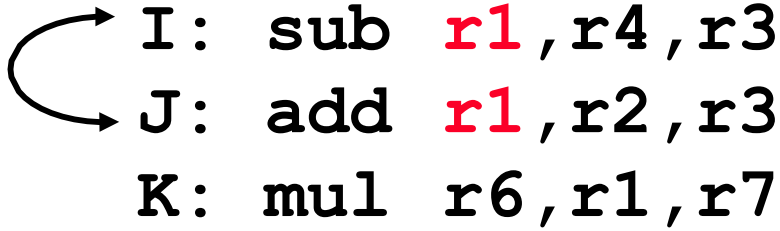
- **Can't happen in MIPS 5 stage pipeline because:**
  - **All instructions take 5 stages, and**
  - **Reads are always in stage 2, and**
  - **Writes are always in stage 5**

Execution Order is:
$Instr_I$
$Instr_J$

Write After Write (WAW)

$Instr_J$ tries to write operand *before* $Instr_I$ writes it

- **Leaves wrong result ($Instr_I$ not $Instr_J$)**



- **Called an "output dependence" by compiler writers This also results from the reuse of name "r1".**
- **Can't happen in MIPS 5 stage pipeline because:**
  - **All instructions take 5 stages, and**
  - **Writes are always in stage 5**
-

Q.Consider the following MIPS assembly code:

```
LW R1, 10(R2)
ADD R7, R1, R5
SUB R8, R7, R6
MUL R6, R4, R8
```

Identify each dependency by type and list the two instructions involved.

Solution

LW& ADD	(R1- True),
ADD & SUB	(R7 - True),
SUB & MUL	(R8 - True),
SUB & MUL	(R6- Anti)

Q.Consider the following MIPS assembly code:

```
LW R4, 10(R2)
LW R5, 0(R4)
MUL R7, R5, R4
DIV R3, R7, R6
```

Identify each data hazard by type and list the two instructions involved in it.

RAW :- 1st LW - 2nd LW (due to R4)
RAW :- 1st LW - MUL (due to R4)
RAW :- 2nd LW - MUL (due to R5)
RAW :- MUL – DIV (due to R7)