Module 3: Processor Design
Multicycle Operations

Module 3 Processor Design

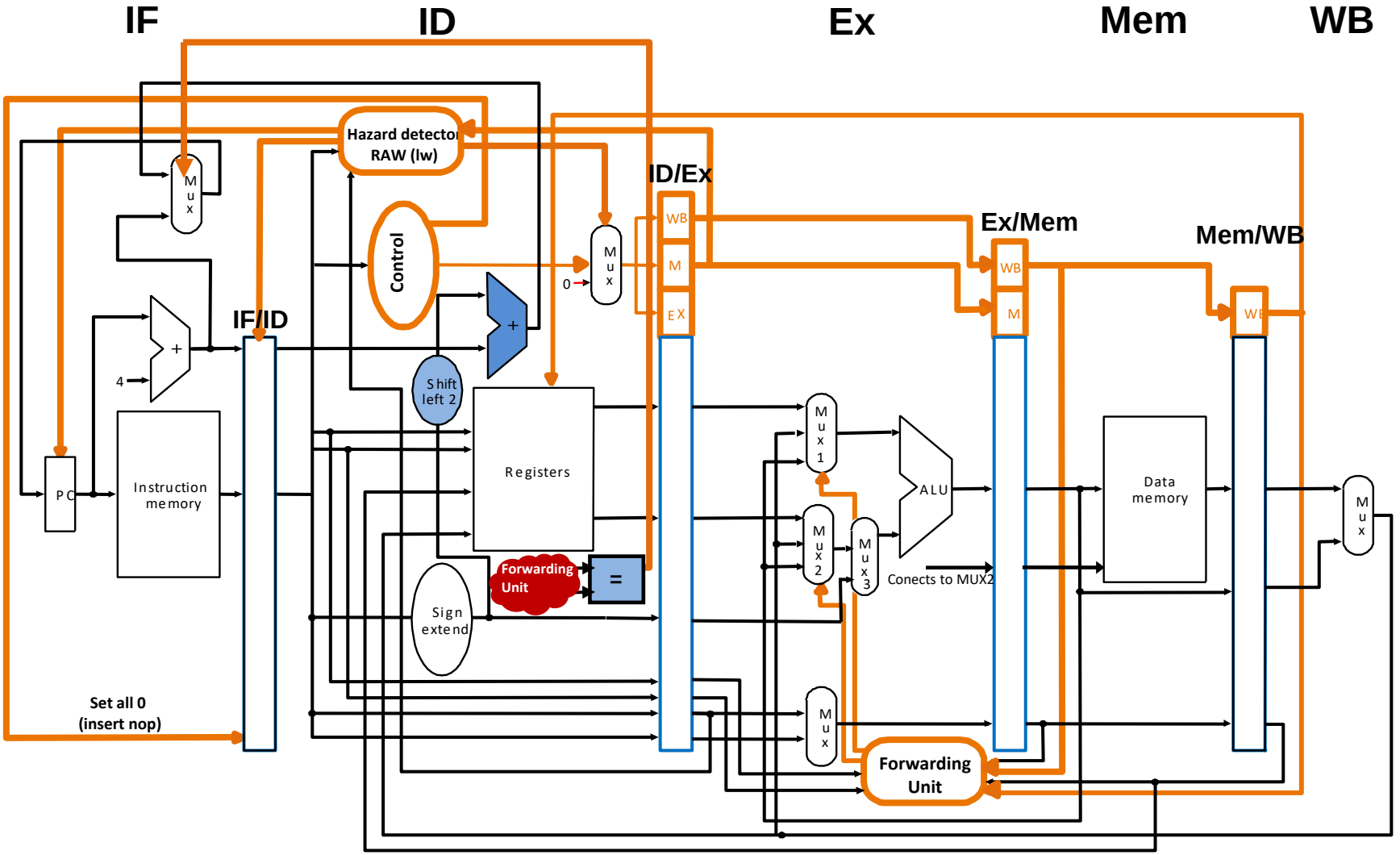
Index

- **References**
  - **Hennessy Patterson Apendix C.5**

1. Introduction

Complete data path of the pipelined processor

- Has implemented the optimal HW solutions for all hazards seen so far
- **The entries of the comparison module come from two Mux that choose between values from the RB and those from the forwarding units**



1. Introduction

Pipelined processor

- ✓ All instruction have the same duration
- ✓ Ideal performance: CPI=1
- ✓ WAW and WAR hazards are avoided
- ✓ RAW hazards are avoided using forwarding
- ✓ RAW hazards in load instructions generate pipeline stalls. The compiler can help by scheduling the instructions.
- ✓ Control hazards. Stalls and forwarding.
- ✓ **Instructions start and finish in order**

2. Floating point operations

We want to add floating point instructions to the instruction set.

- We need instructions for the basic operations (+, -, *, /)
- We need a different register file (F0…F15) to store floating point operands and load and store instructions that work on that register file.
- We need new functional units.

Single (32 bits)	ADDS	SUBS	MULTS	DIVS	LS	SS
Double (64 bits)	ADDD	SUBD	MULTD	DIVD	LD	SD

Example:

ADDS F0,F1,F2 adds F1 and F2 and stores the result in F0.

ADDD F0, F2,F4 adds the double number stored in F2-F3 to the double number in F4-F5 and stores the result in F0-F1.The least significant bit of the register number is ignored.

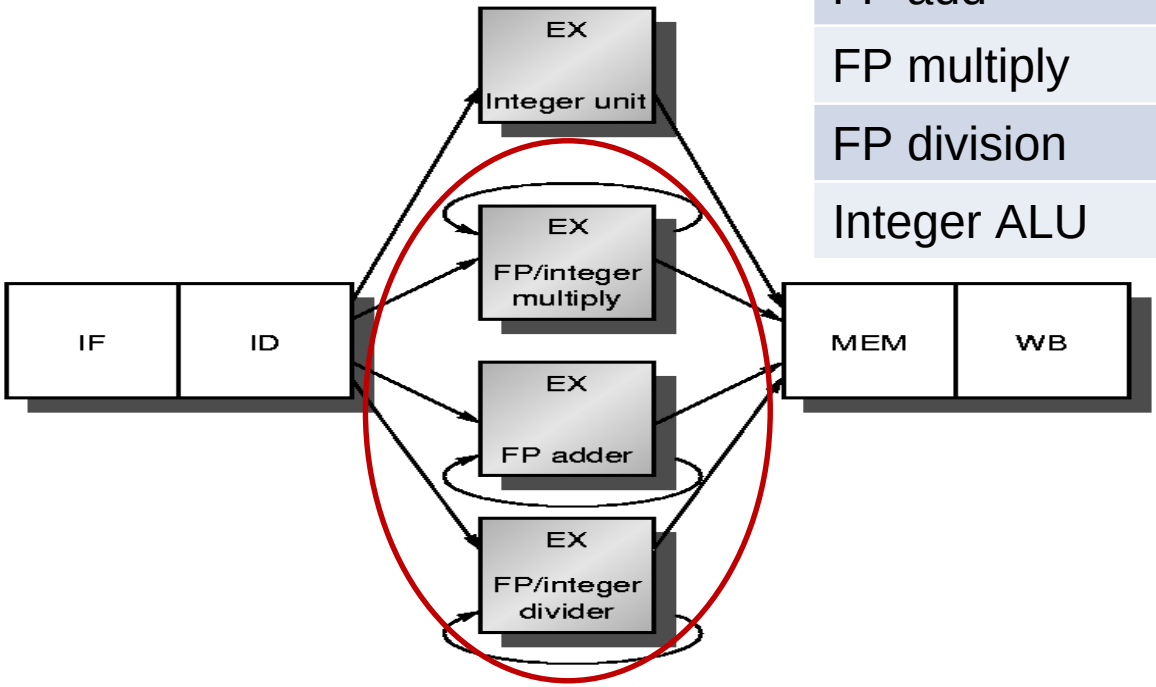
It is impractical to require that all floating point operations complete in 1 clock cycle.

2. Multicycle Operations

What happens if the instructions have different execution cycles?

- Floating points operations require more than one cycle
- Integer multiplication and division may also require more than one cycle

Functional Unit	Latency
FP add	4
FP multiply	7
FP division	24
Integer ALU	1



FU latency : Number of cycles to execute the instruction

2. Multicycle Operations

Problems

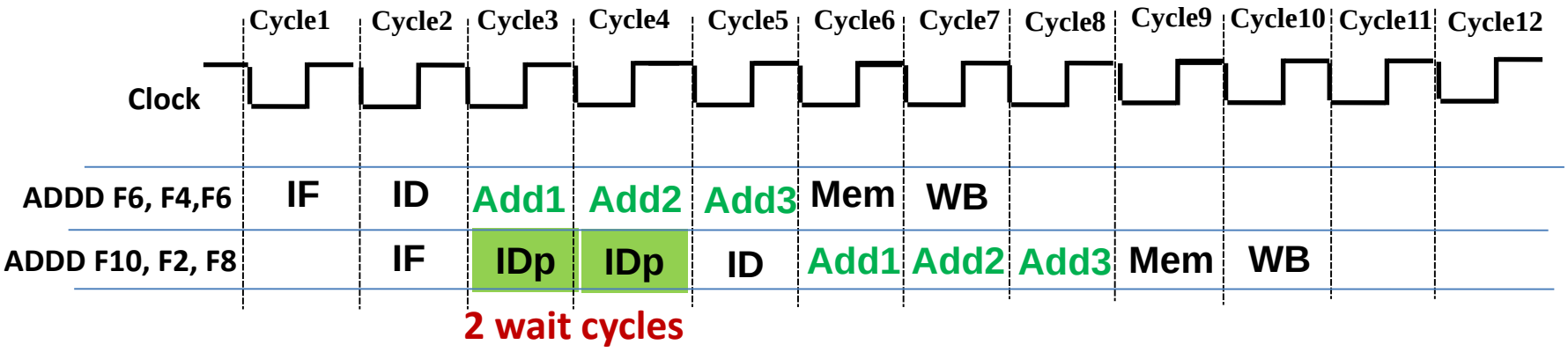
- Structural Hazards
- **Higher penalization** of RAW hazards
- WAW and WAR hazards appear
- Problems with **out of order commit**

3. Multicycle Operations: structural hazards

Structural Hazard: two instructions require the same FU

- **Stall required until the FU is free**

Functional Unit	Latency
FP add	3

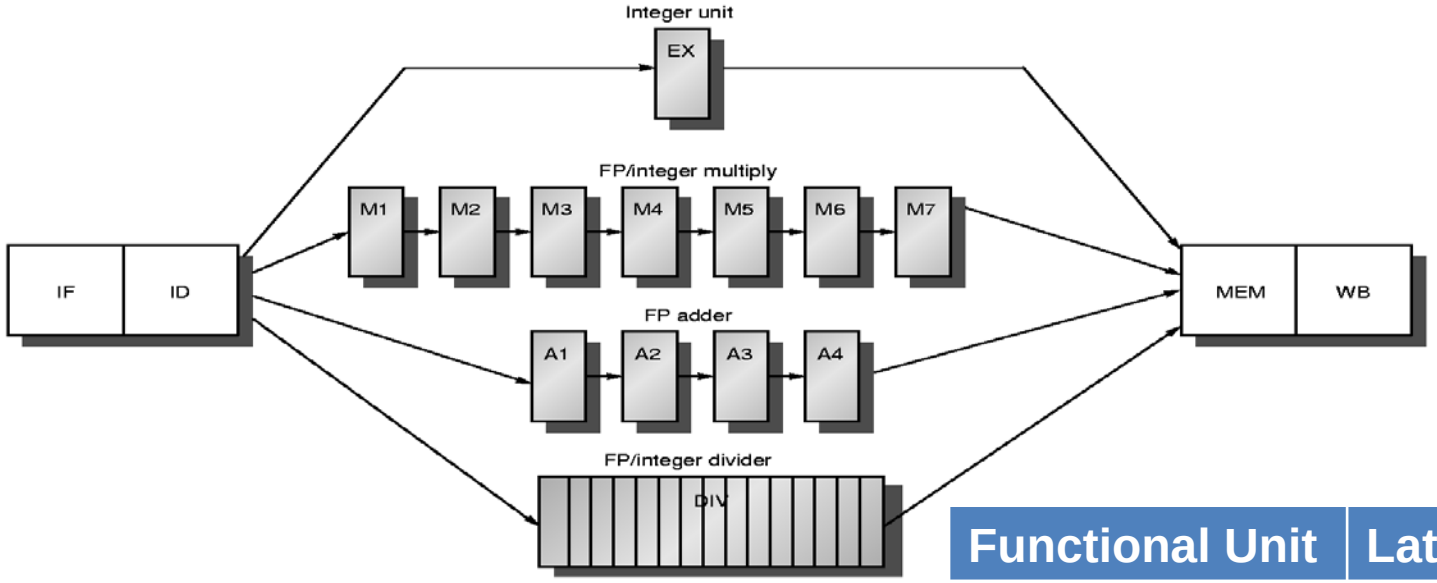


✓ **IDp** **Stall for Structural Hazard, the FP adder is needed but it is busy**

3. Multicycle Operations: structural hazards

Solution: **Pipeline the FUs with latency > 1**

- Division is usually not pipelined, the hazard must be detected and the pipeline stalled



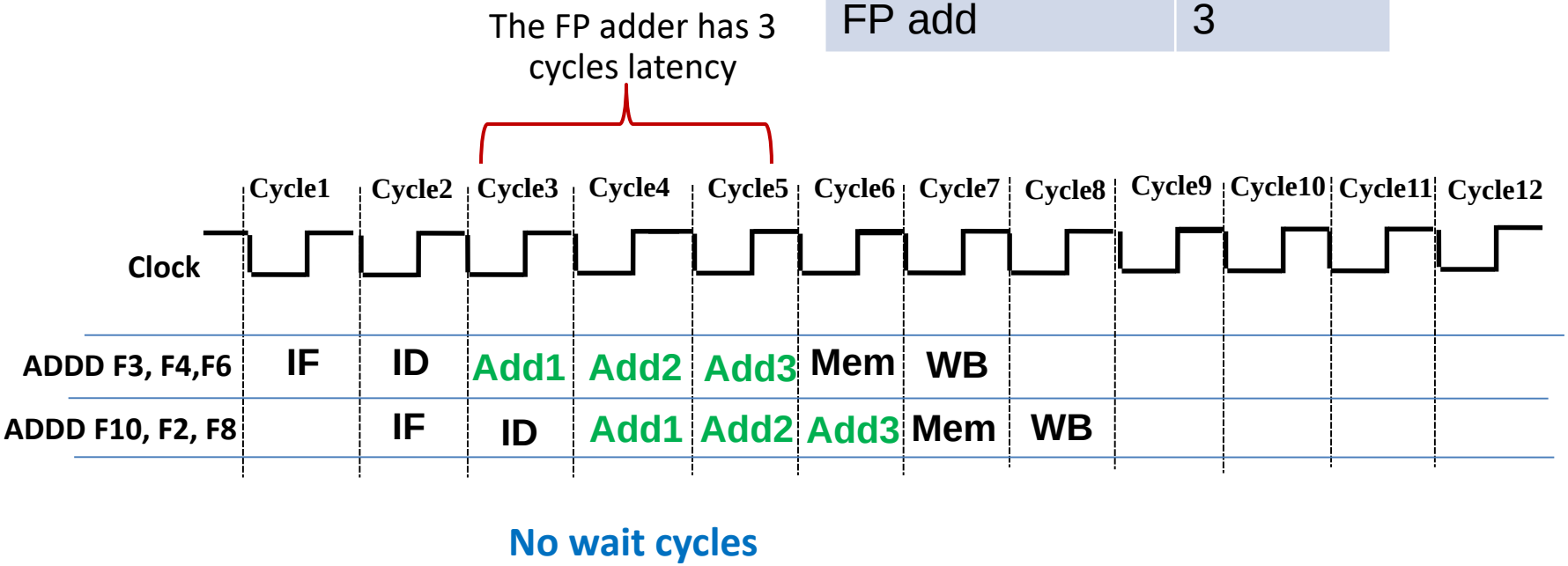
Functional Unit	Latency
FP add	4
FP multiply	7
FP division	24
Integer ALU	1

3. Multicycle Operations: structural hazards

Solution: **Pipeline the FUs with latency > 1**

- **Only wait until the first FU stage is free**

Functional Unit	Latency
FP add	3



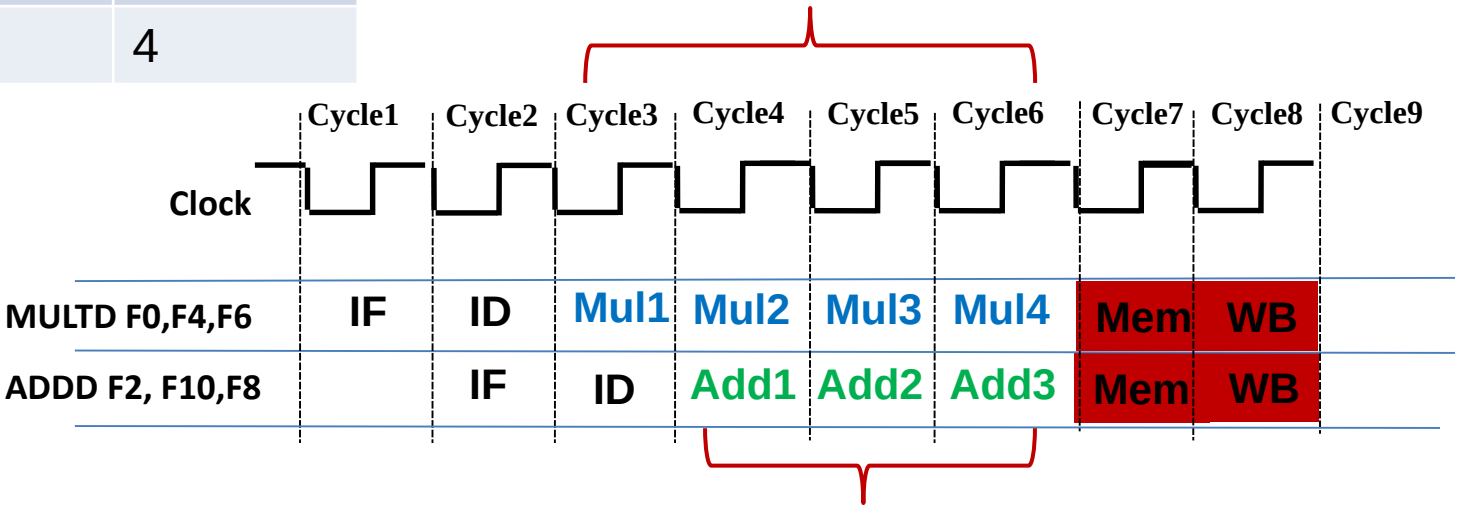
No wait cycles

3. Multicycle Operations: structural hazards

Structural Hazards

- Two instructions cannot reach the Mem stage simultaneously
- Two instructions cannot reach the WB stage simultaneously

Functional Unit	Latency
FP add	3
FP multiply	4

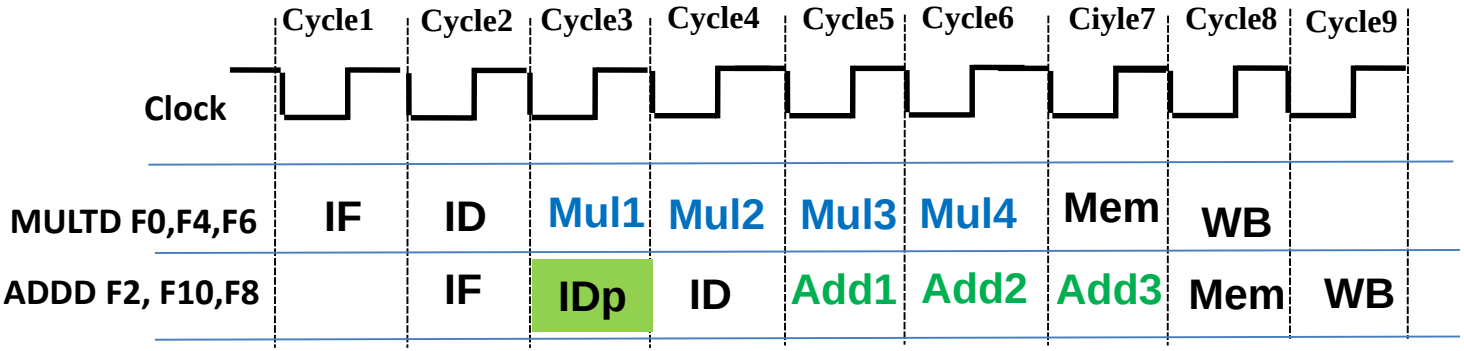


3. Multicycle Operations: structural hazards

Solution 1:

- **Stop** the second instruction in the **decoding stage**

Functional Unit	Latency
FP add	3
FP multiply	4



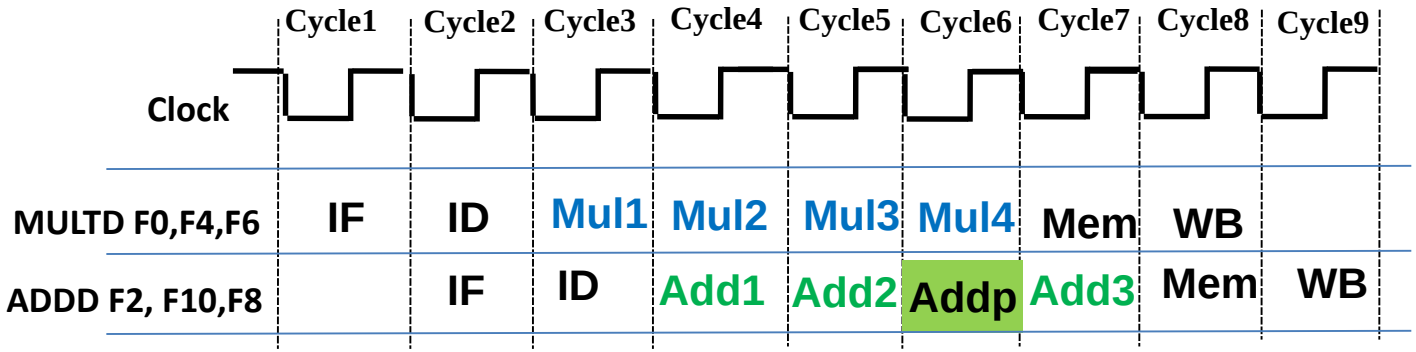
IDp **Stall due to structural hazard, two instructions reaching Mem in the same cycle**

3. Multicycle Operations: structural hazards

Solution 2:

- **Stop** the conflicting instructions **at the end of their Ex stage**
  - Requires to establish a priority scheme
    - For example, giving more priority to large latency units
  - Split detection and generation logics in two different poins of the data path

Functional Unit	Latency
FP add	3
FP multiply	4



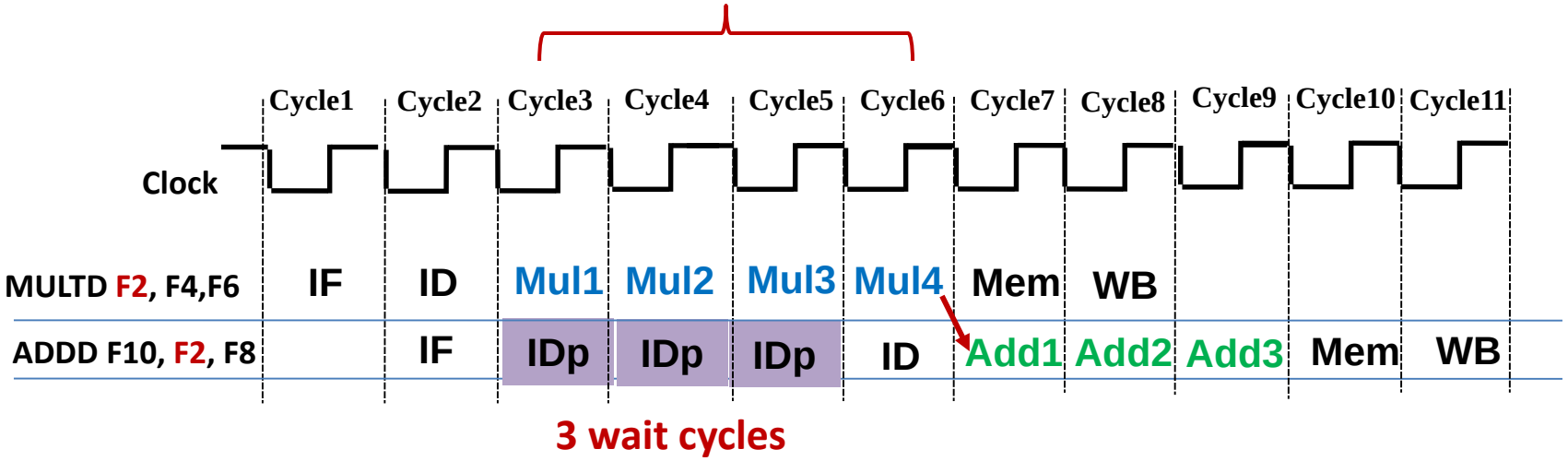
Addp **Stall due to a structural hazard, two instructions reaching the Mem stage in cycle 7**

4. Multicycle Operations: RAW hazards

What happens with RAW hazards when data is provided by an A-L instruction

– Forwarding does not remove all stalls

Functional Unit	Latency
FP add	3
FP multiply	4



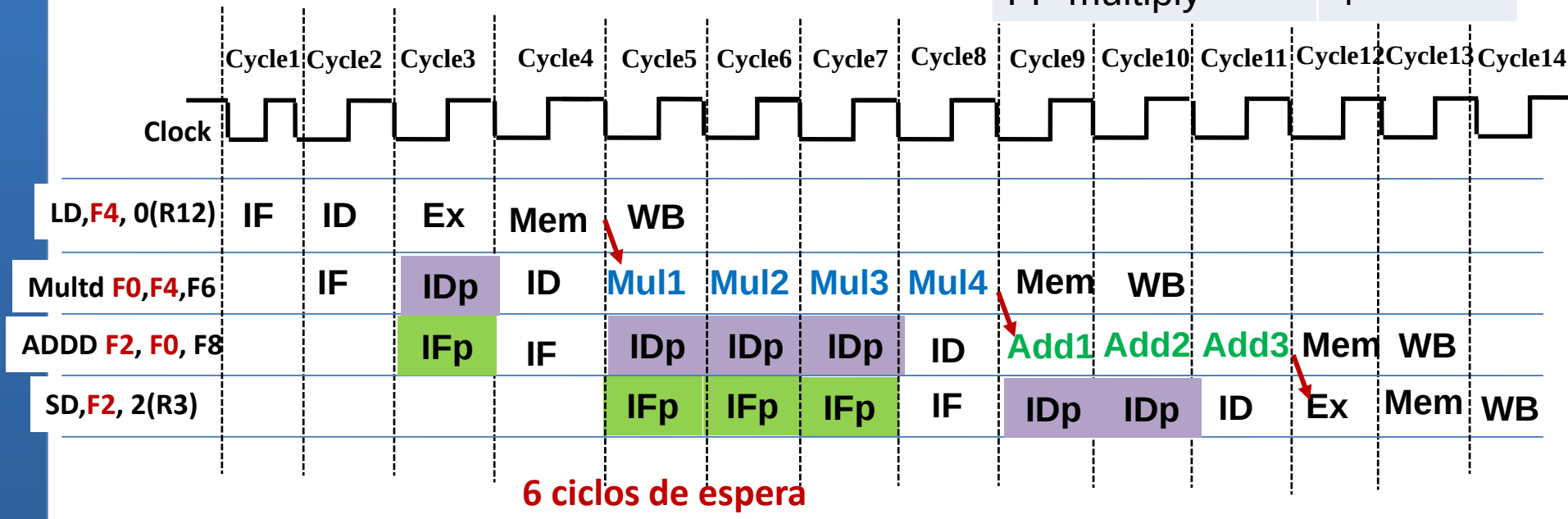
✓ IDp Stall due to RAW hazard

4. Multicycle operations: RAW hazards

What happens with RAW when data is provided by a lw instruction

- The instruction depending on the lw suffers one stall
- The forwarding logic does not remove all stalls

Functional Unit	Latency
FP add	3
FP multiply	4

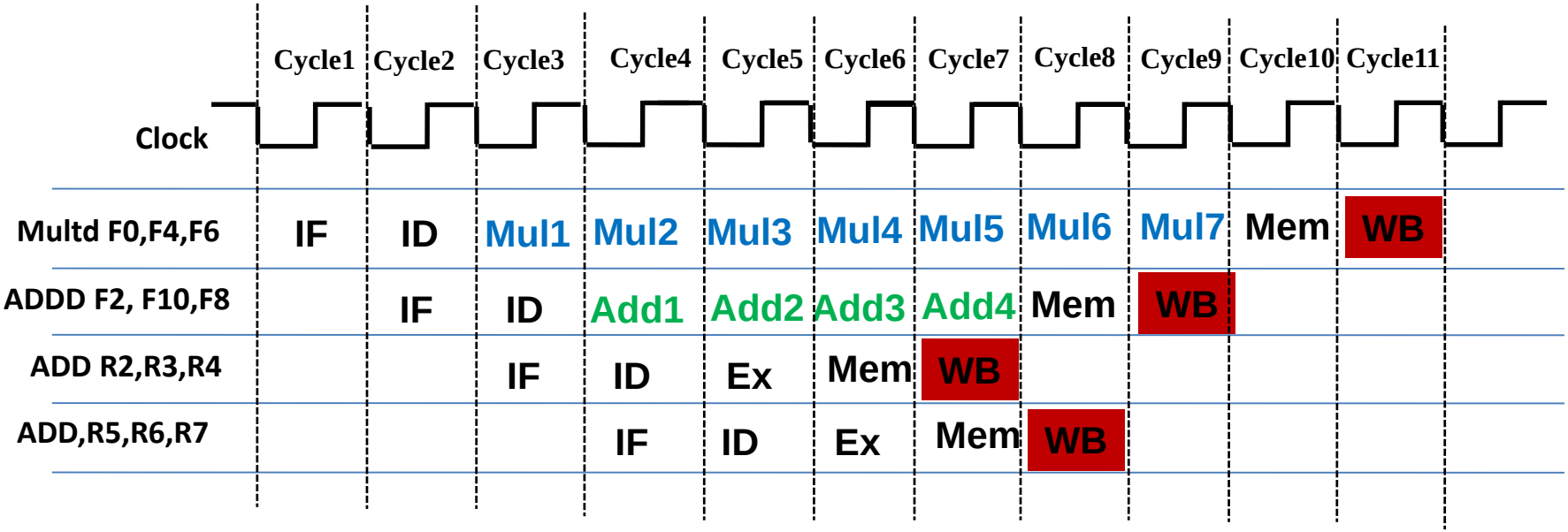


✓ IDp Stall because of RAW hazard

5. Multicycle Operations: out of order commit

- The finishing order may differ from the starting order
- **Problems**:
  - Conflicts due to simultaneous writes to the RB (structural hazard)
  - WAW hazards appear

Functional Unit	Latency
FP add	4
FP multiply	7

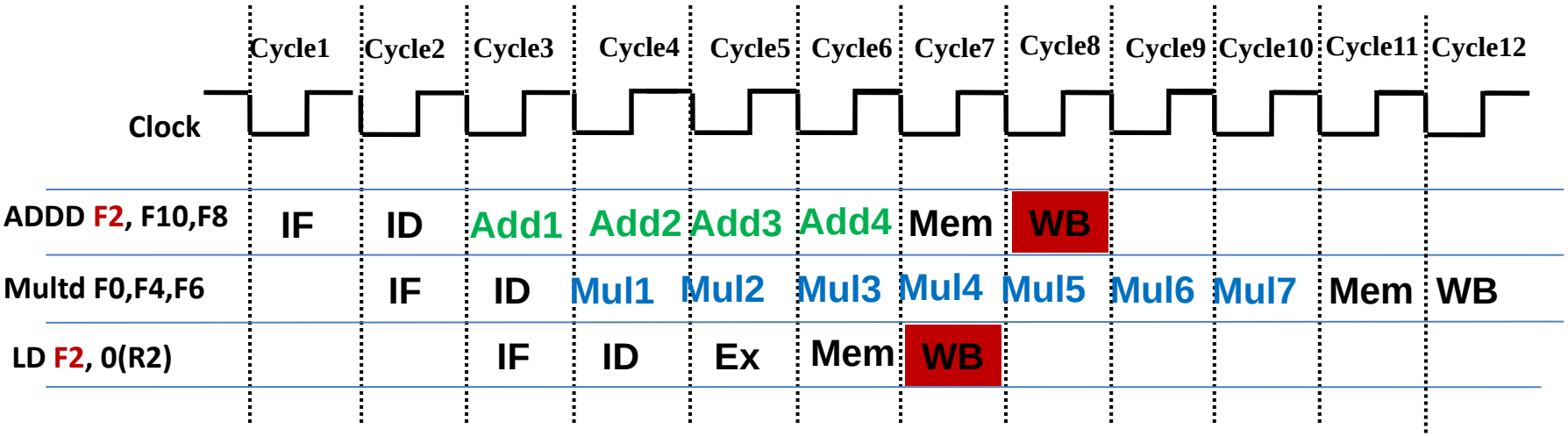


6. Multicycle operations: WAW hazards

What happens when a WAW hazard appears

- A wrong value is stored on the register
- Unlikely situation: in the example the ADDD F2, F10,F8 instruction is useless

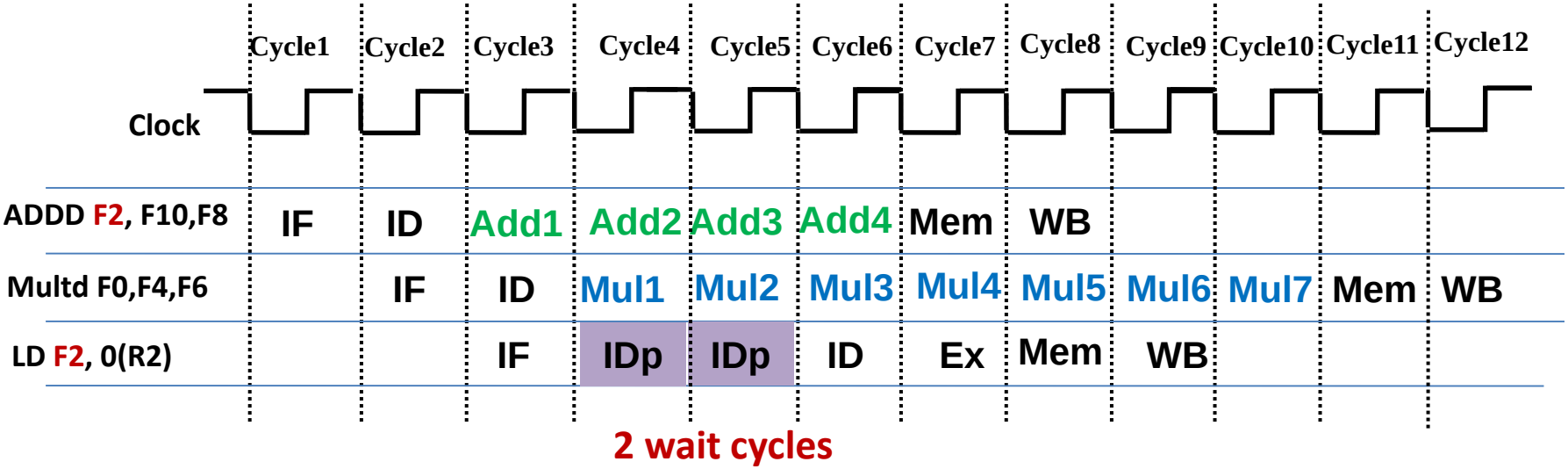
Functional Unit	Latency
FP add	4
FP multiply	7



6. Multicycle operations: WAW hazards

Solution1

- **Stop the instruction that produces the WAW hazard** (the second one)
- The number of stalls needed depends on the "length" of the first instruction and the distance between both of them.

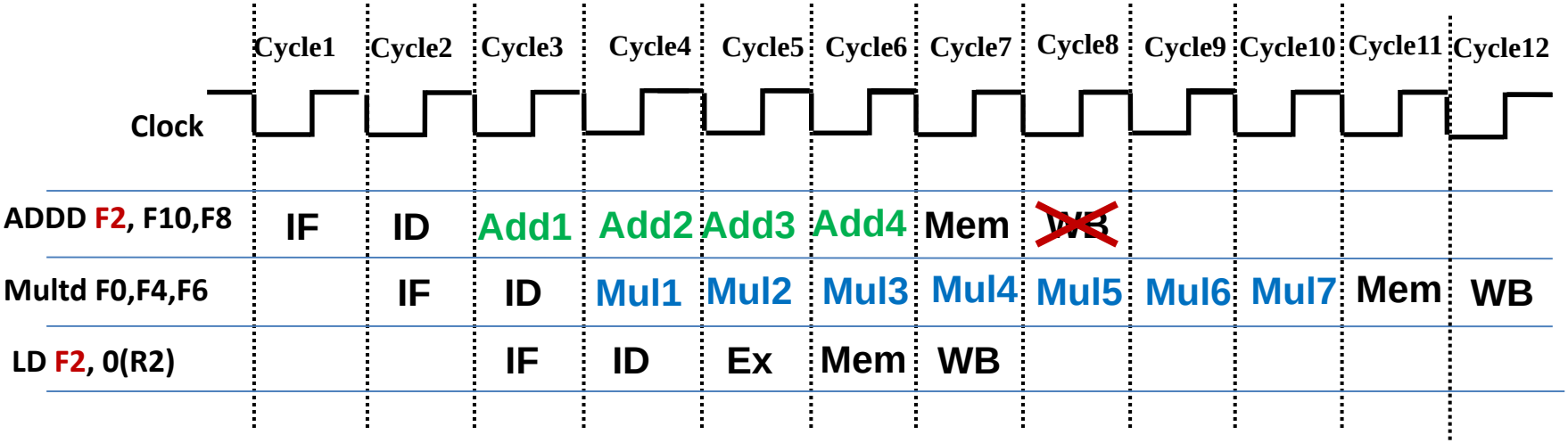


✓ IDp Stall to solve a WAW hazard

6. Multicycle Operations: WAW hazards

Solution 2

- Inhibit the RB write for the first instruction (clearing its write enable signal on the pipelining register)



7. Multicycle operations: WAR hazards

These hazards **do not appear** due to the pipeline design

- RF is read always in the ID stage
- RF is written always in the WB stage (last)
- Until ID everything is in order