Saeed Safari,

Advanced Computer Architecture

Pipeline – Basic Concepts

ECE School,

Faculty of Engineering,

University of Tehran

Outline

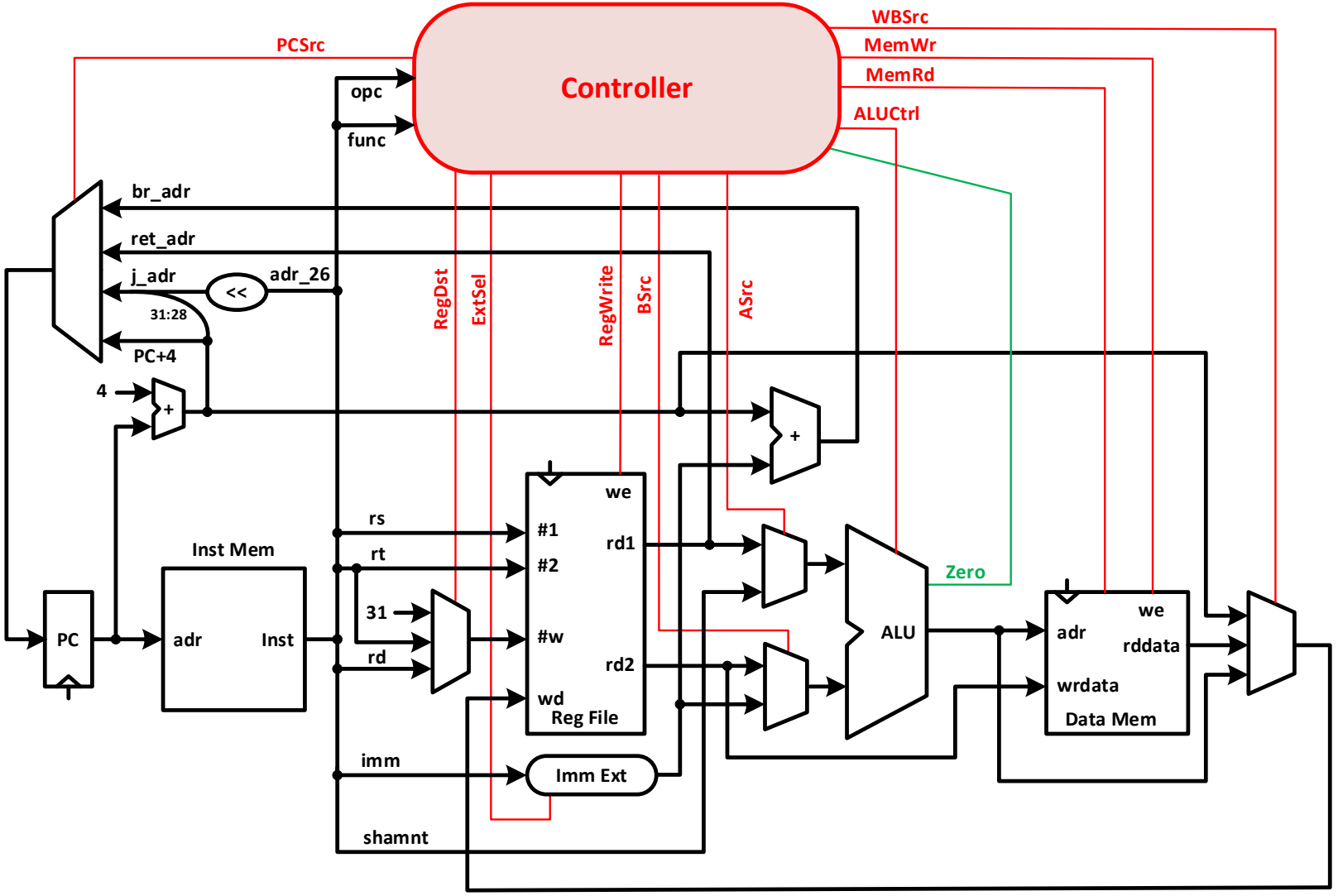
- MIPS implementation
  - Single-cycle
  - Multi-cycle
  - Pipeline
- Hazard
  - Structural hazard
  - Data hazard
  - Control hazard

MIPS ISA – An Overview

Registers	**32 32-bit integer registers (R0 – R31),** 32 32-bit floating point registers (F0 – F31), 2 32-bit integer register (hi, lo)
Addressing Modes	**Register, Immediate, PC Relative, Base**

R-Type	**add, addu, sub, subu, slt, sltu, and, or, xor, nor, sll, srl, sla, sllv, arlv, slav, jr, jalr,** mult, multu, div, divu, mfhi, mthi, mflo, mtlo
I-Type	**addi, addiu, slti, sltiu, andi, ori, xori, lui, lw, sw, beq, bne**
J-Type	**j, jal**
FR-Type	add.s, sub.s, mul.s, div.s, abs.s, neg.s, c.eq.s, c.lt.s, c.le.s add.d, sub.d, mul.d, div.d, abs.d, neg.d, c.eq.d, c.lt.d, c.le.d
FI-Type	l.s, s.s, l.d, s.d, bc1t, bc1f

MIPS – Single Cycle Implementation



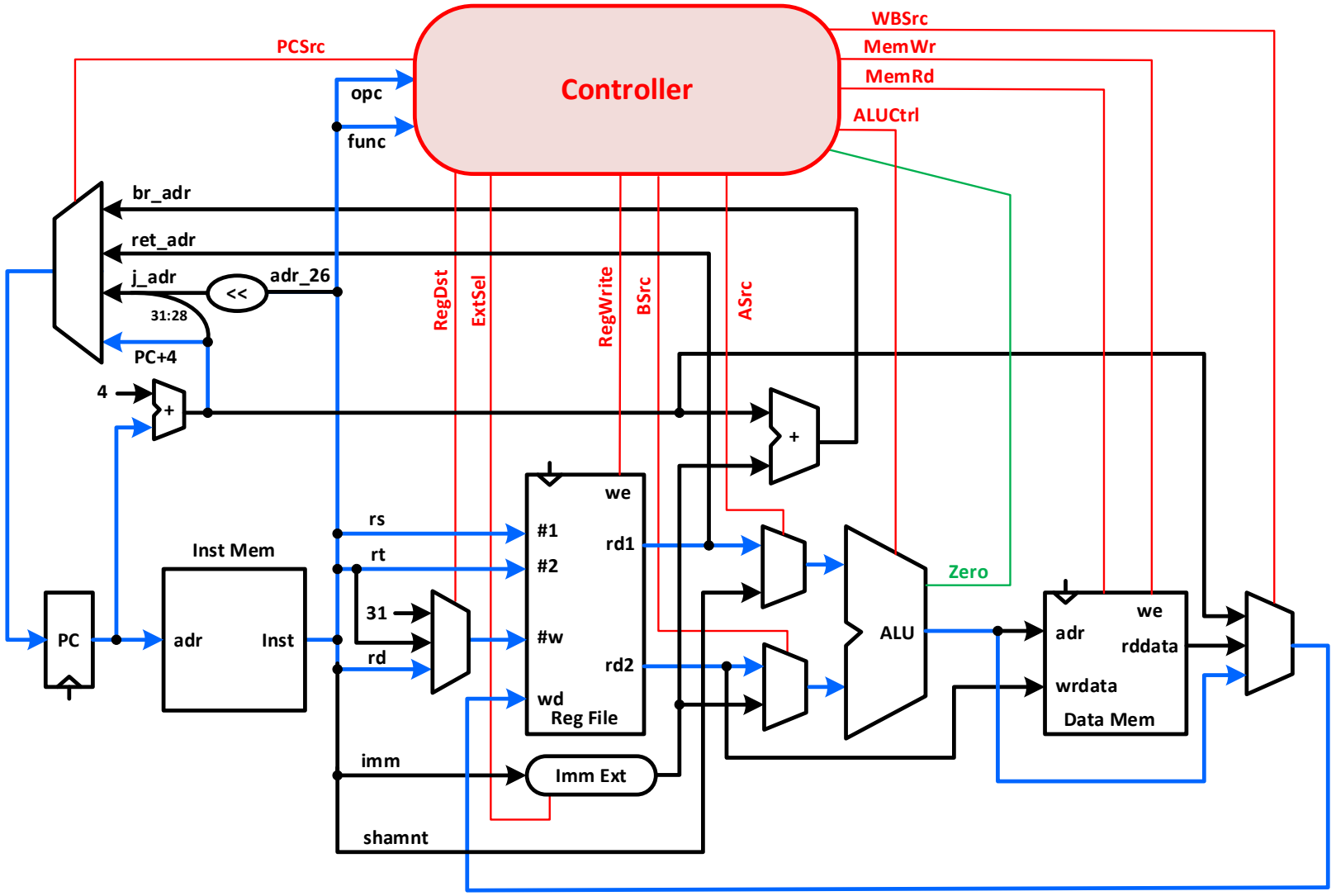
Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

MIPS – Single Cycle Implementation

- Key features:
  - Two different parts
    - Data-path
    - Controller
  - Controller is implemented as a combinational circuit
  - Each instruction needs one clock cycle to execute (single-cycle implementation)

Arithmetic–Logic Instructions

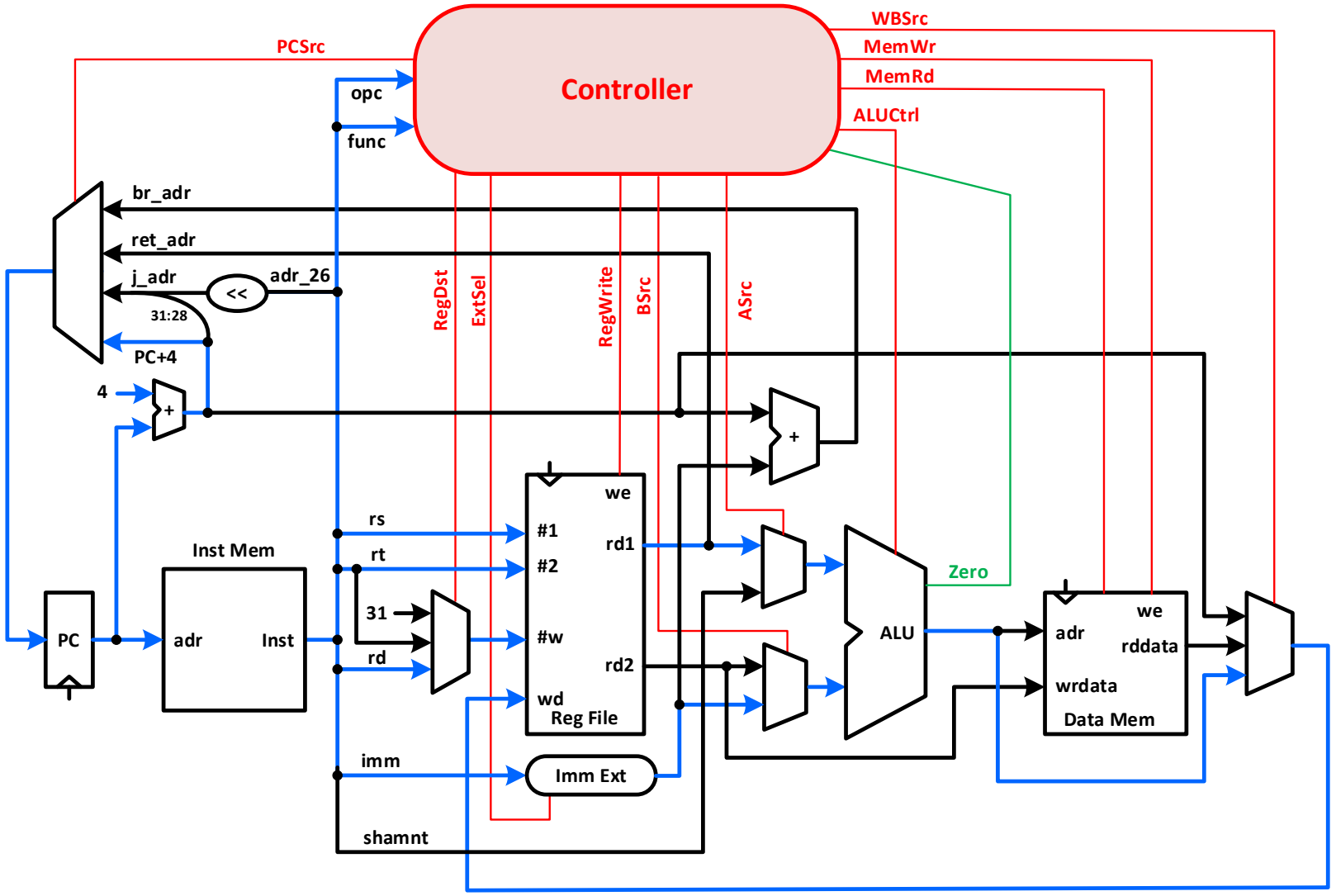
add,
addu,
sub,
subu,
slt,
sltu,
and,
or,
xor,
nor



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Immediate Instructions

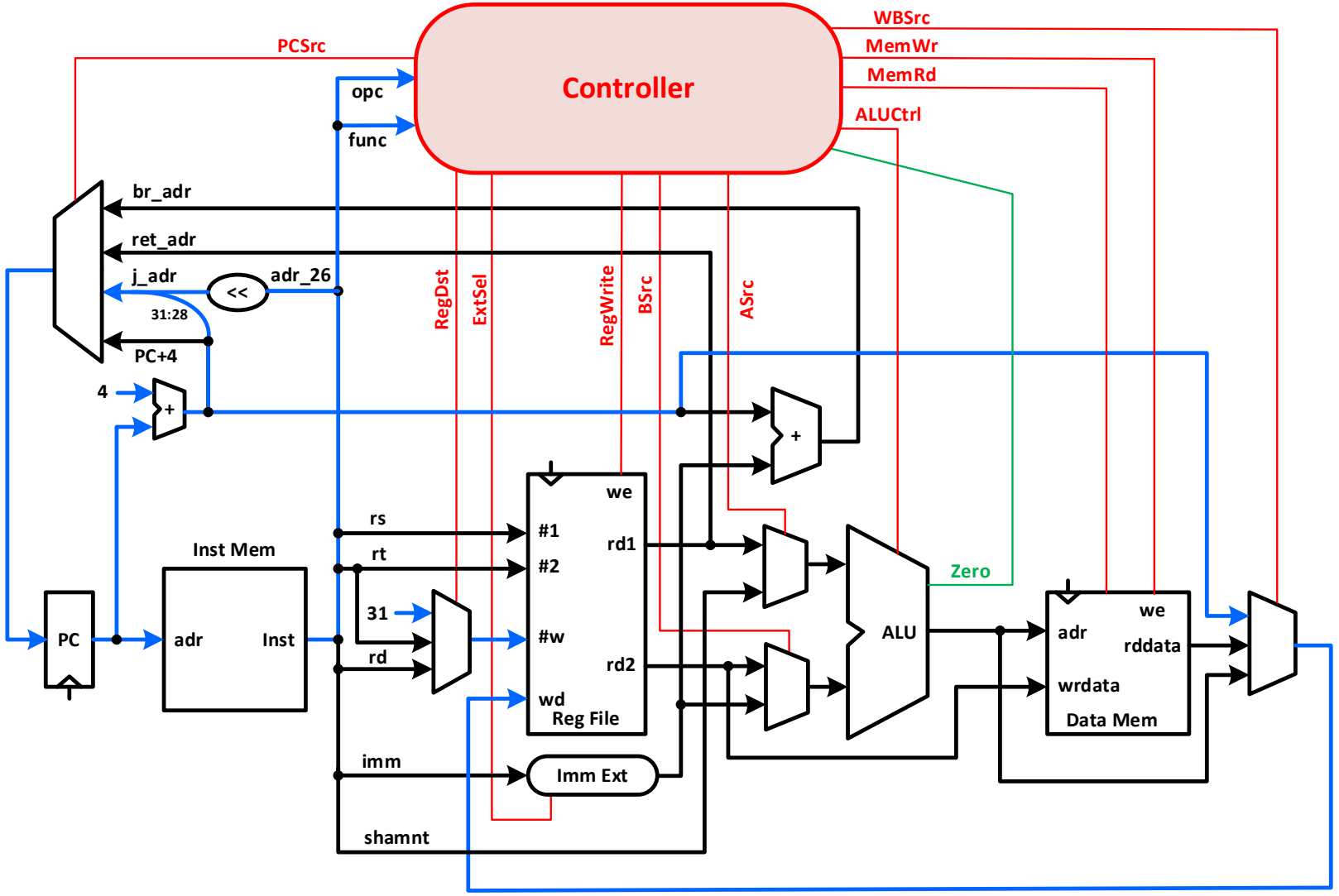
addi,
addiu,
slti,
sltiu,
andi,
ori,
xori,
lui



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

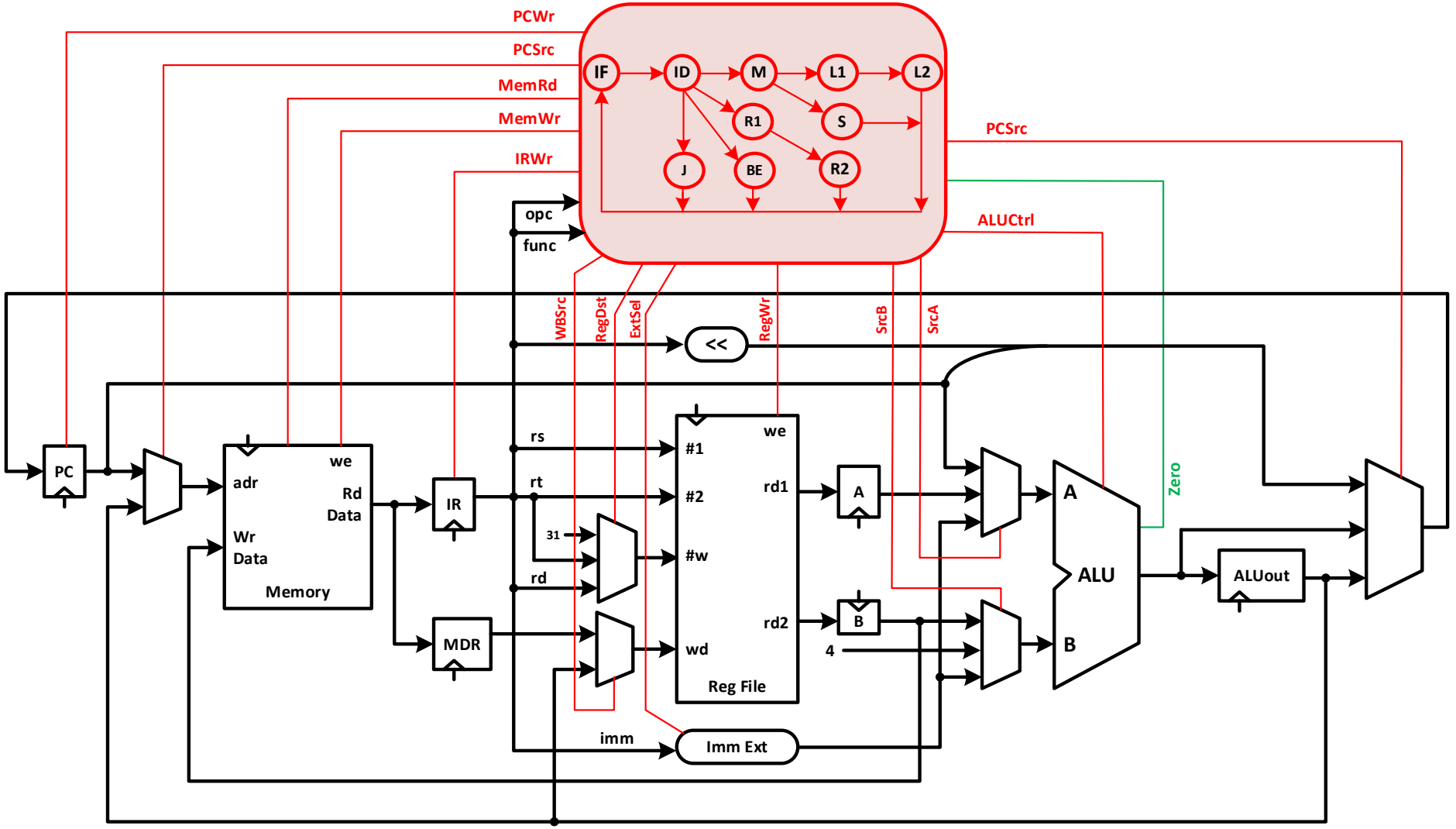
Jal Instruction

jal



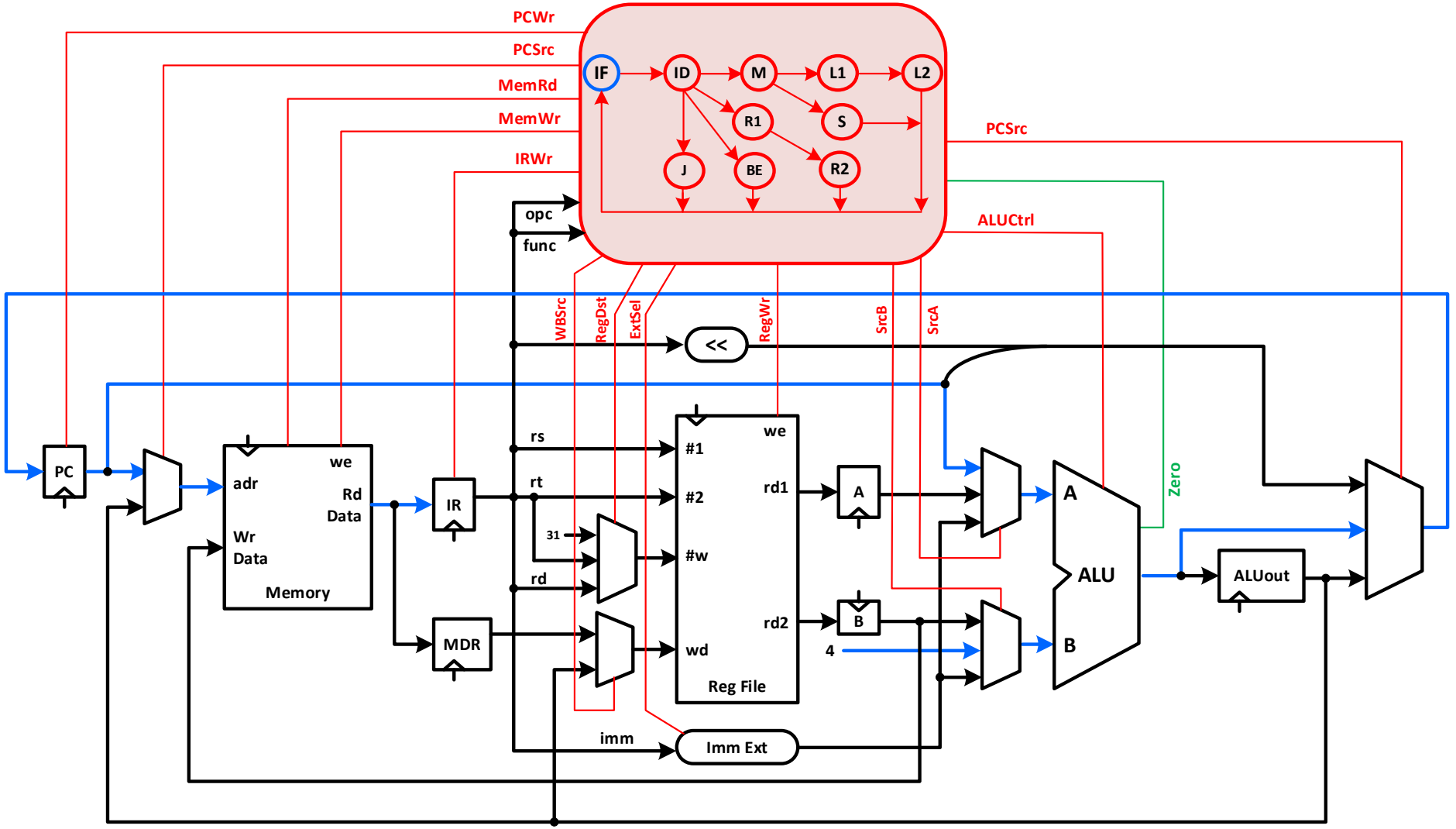
Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

MIPS – Multi-Cycle Implementation



Arithmetic–Logic Instructions (Clk#1)

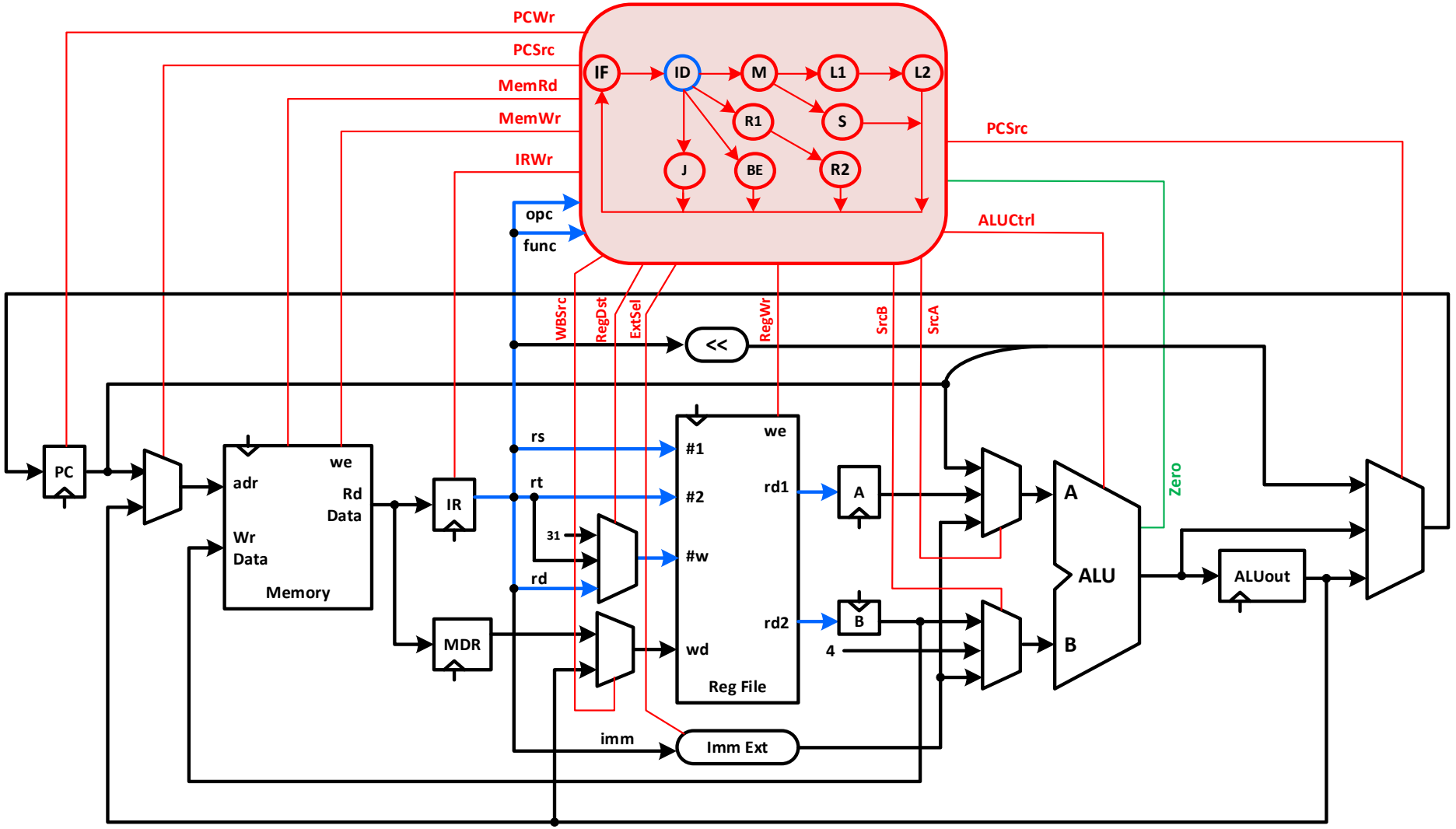
add,
addu,
sub,
subu,
slt,
sltu,
and,
or,
xor,
nor



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Arithmetic–Logic Instructions (Clk#2)

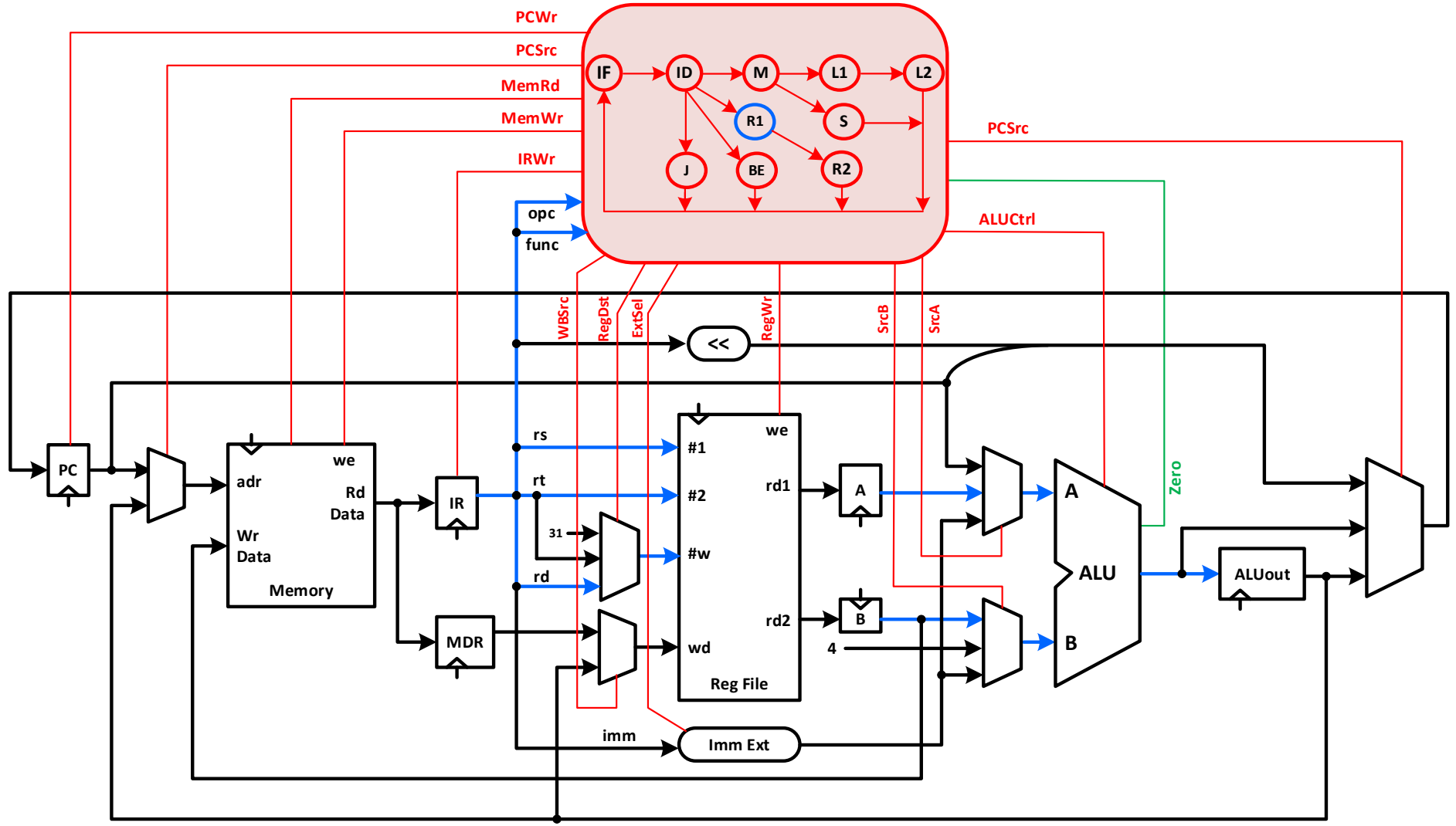
add,
addu,
sub,
subu,
slt,
sltu,
and,
or,
xor,
nor



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Arithmetic–Logic Instructions (Clk#3)

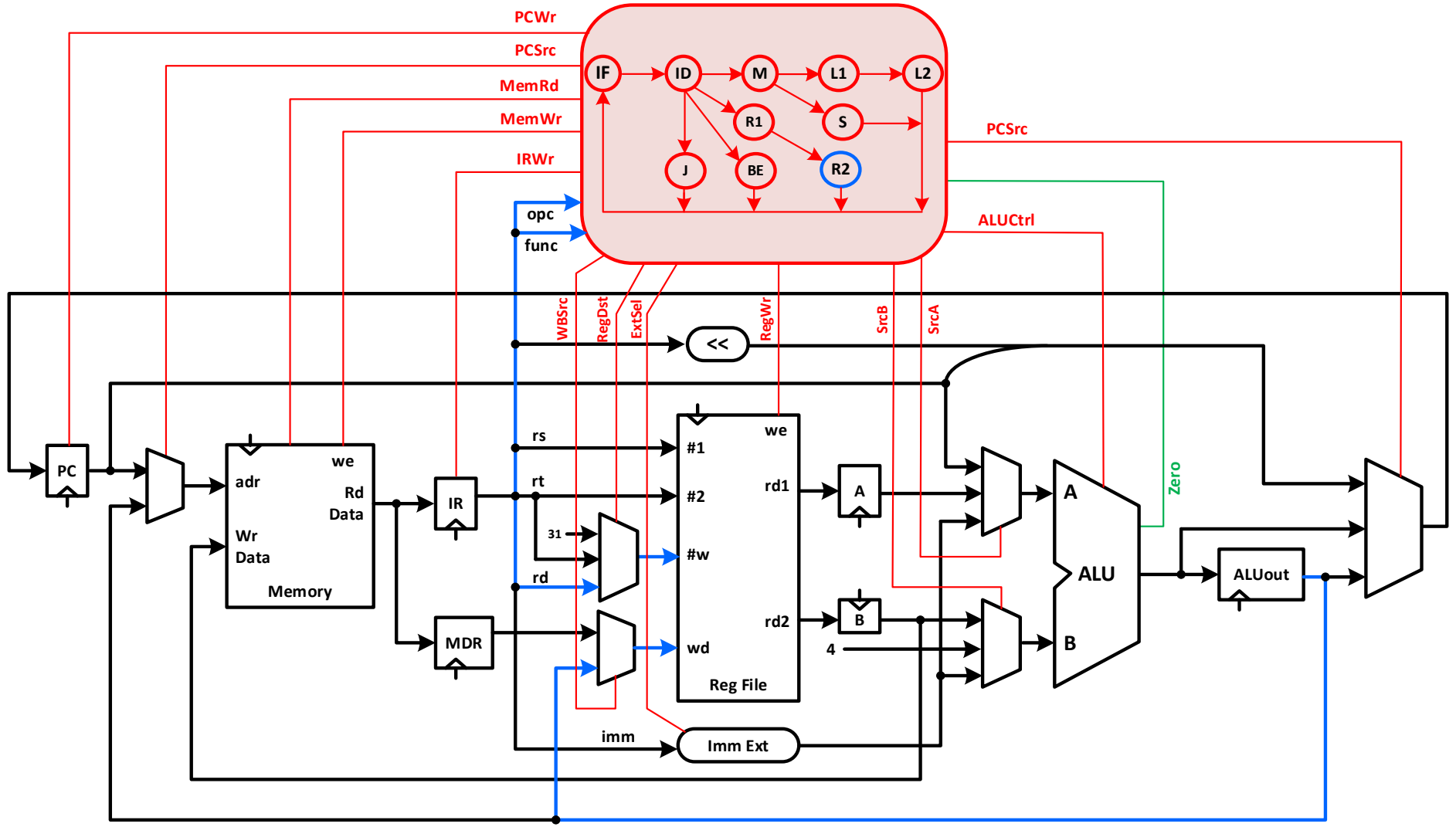
add,
addu,
sub,
subu,
slt,
sltu,
and,
or,
xor,
nor



Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Arithmetic–Logic Instructions (Clk#4)

add,
addu,
sub,
subu,
slt,
sltu,
and,
or,
xor,
nor

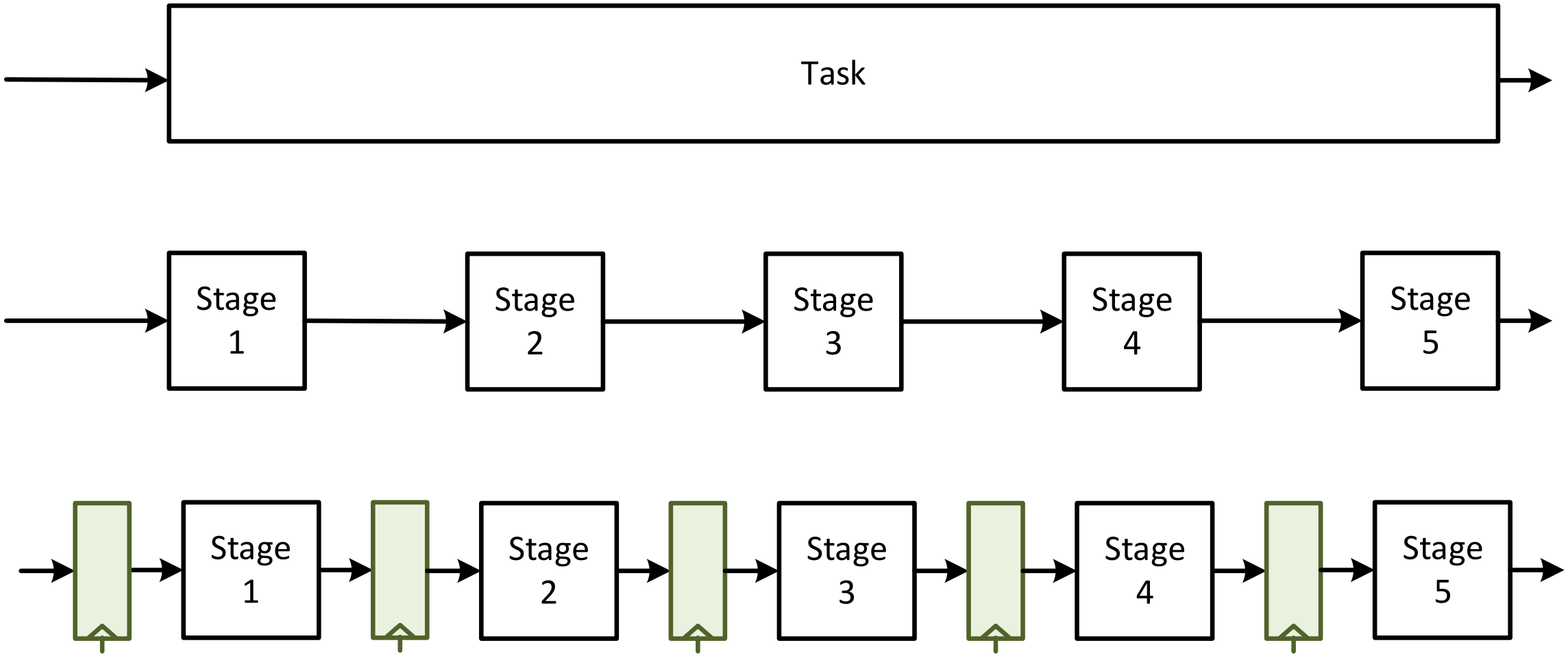


Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

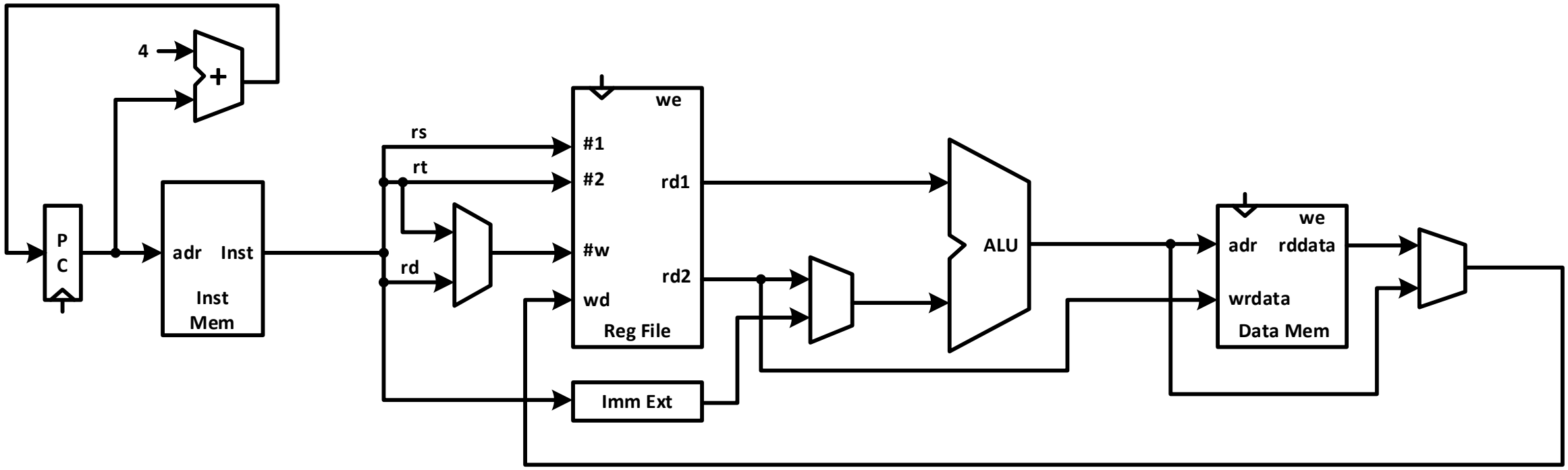
MIPS – Multi-Cycle Implementation

- Key features:
  - Data-path:
    - One ALU plays the role of ALU and two adders
    - One memory module plays the role of both data and instruction memory
  - Controller is implemented as a sequential circuit
  - Instructions need different number of clock cycles to execute (multi-cycle implementation)

Pipelining – Idea

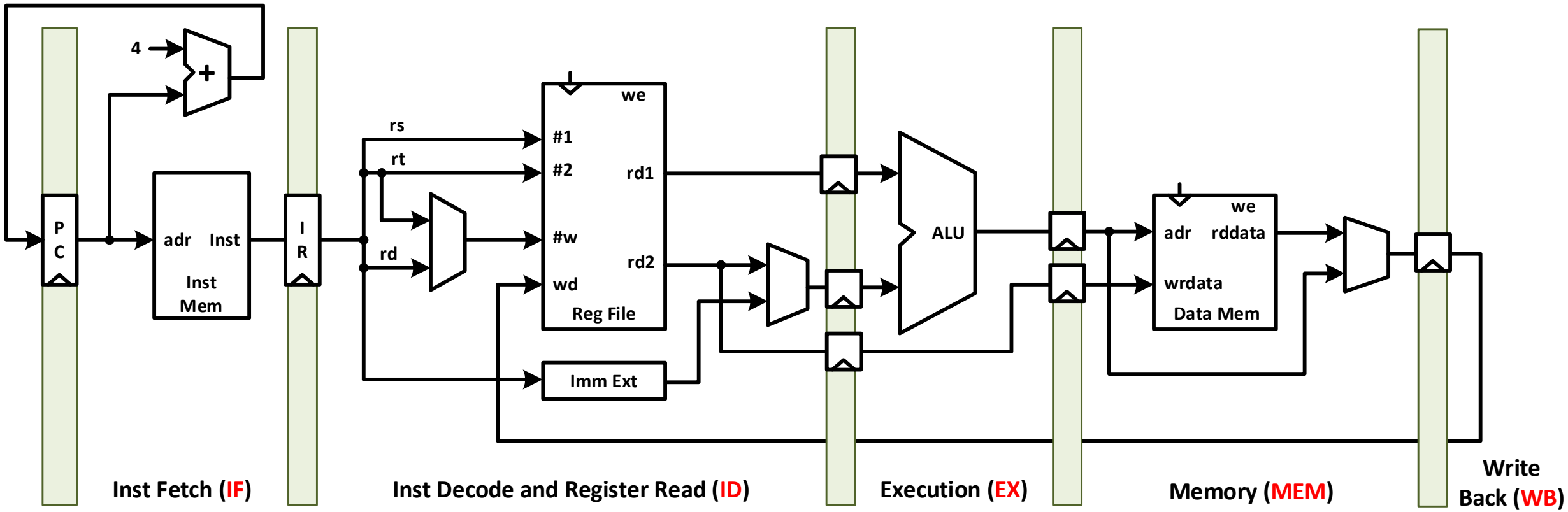


Simple Un-Pipelined Data-Path



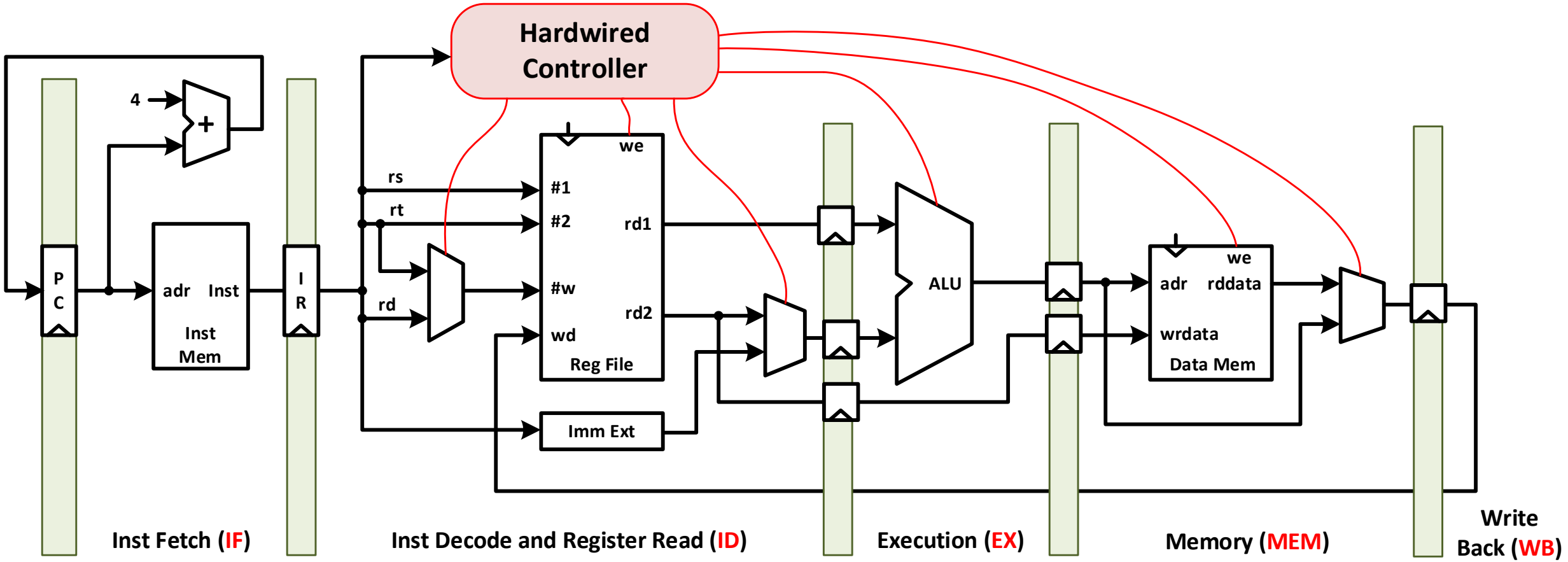
Inst Fetch (IF) | Inst Decode and Register Read (ID) | Execution (EX) | Memory (MEM) | Write Back (WB)

Pipelined Data-Path

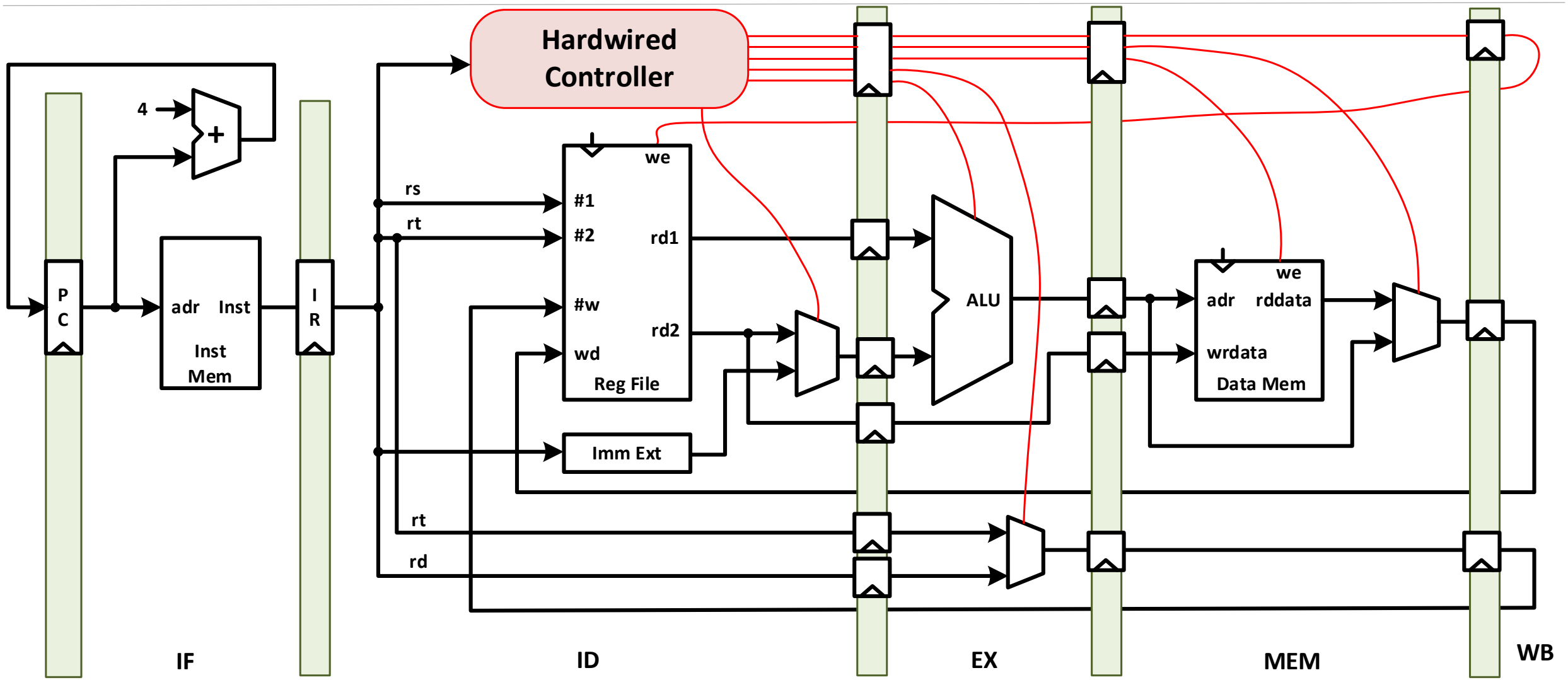


$$t_c > \max(t_{IM}, t_{RF}, t_{ALU}, t_{DM}, t_{WB}) \approx t_{DM}$$

Pipeline Controller



Pipeline Controller



Processor Performance

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

- **Inst/Prog**: Depends on source code, compiler technology, and ISA
- **Cycle/Inst (CPI)**: Depends on ISA and micro-architecture
- **Time/Cycle**: Depends on micro-architecture and base technology

Processor Performance

Micro-Architecture	CPI	Cycle Time
Multi-Cycle	> 1	Short
Single Cycle (unpipelined)	1	Long
Pipelined	1	Short

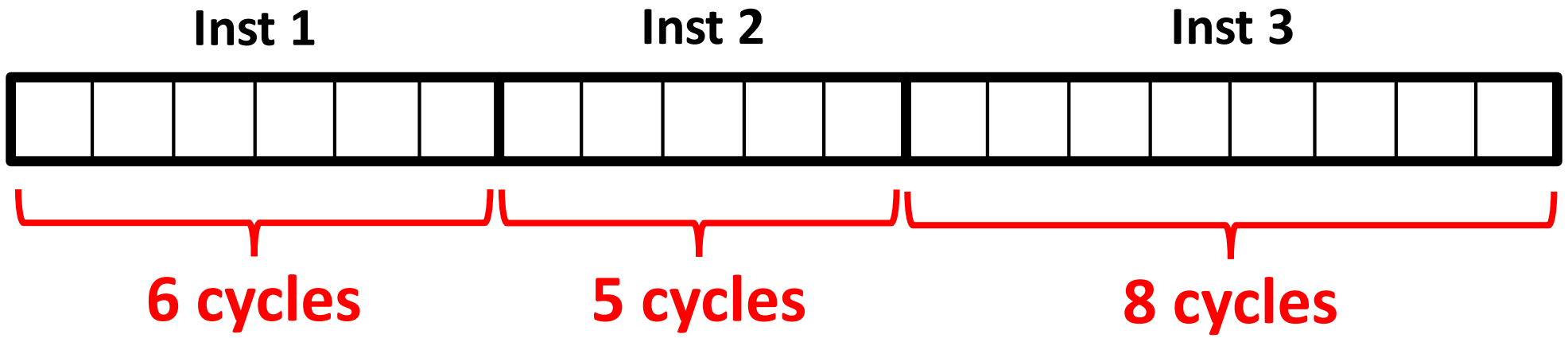
CPI Examples

- Single-cycle implementation

- 3 instructions, 3 cycles, CPI = 1

CPI Examples

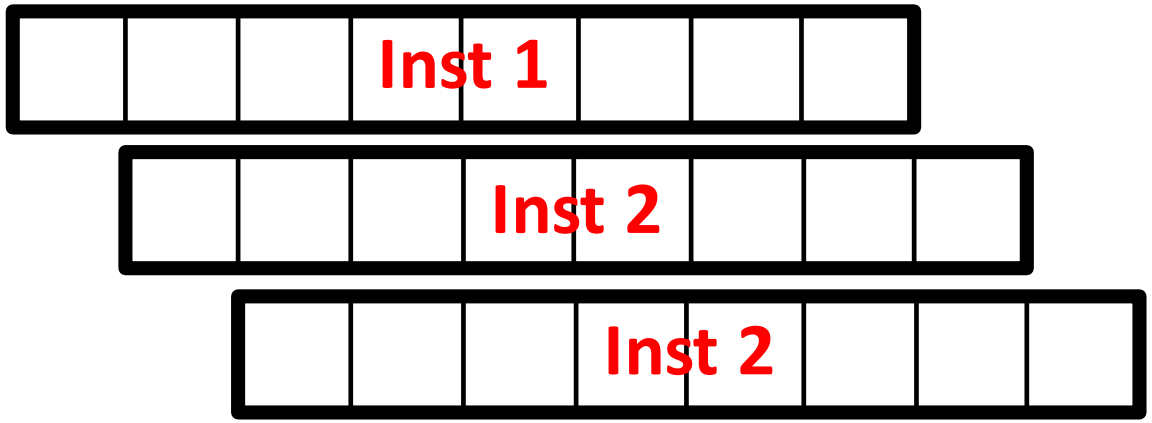
- Multi-cycle implementation



- 3 instructions, 19 cycles, CPI = 6.33

CPI Examples

- Pipeline implementation



- 3 instructions, 3 cycles, CPI = 1

Technology Assumption

- A small amount of very fast memory (Cache)
- Fast ALU
- Reasonable assumption:

$$t_{IM} \approx t_{RF} \approx t_{ALU} \approx t_{DM} \approx t_{WB}$$

- We focus on a 5-stage pipeline
  - Some commercial processors use 30-stage pipeline!

Technology Assumption

- Let's assume:

$$t_{IM} = t_{RF} = t_{ALU} = t_{DM} = t_{WB} = 200^{ps}$$

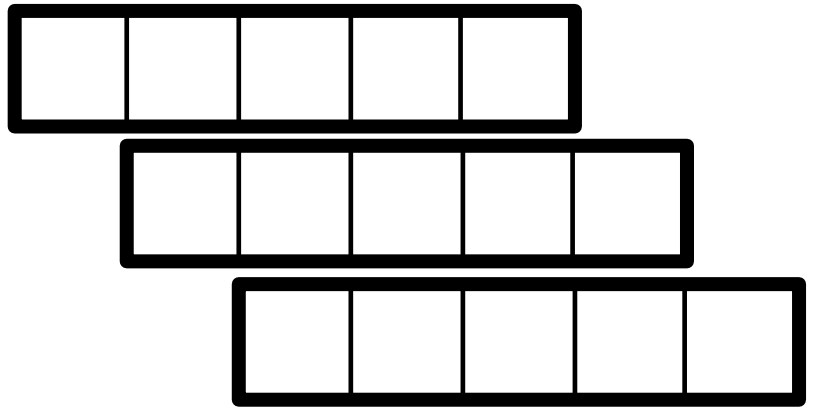
- Use a program with n instruction as a benchmark. Find the speedup of the pipelined implementation over SC implementation.

Technology Assumption

$$T_{SC} = (5 \times 200) \times n = 1000n$$

$$T_{Pipe} = (5 \times 200) + (n - 1) \times 200 = 800 + 200n$$

$$Speedup = \frac{T_{SC}}{T_{Pipe}} = \frac{1000n}{800 + 200n}$$



$$Speedup \xrightarrow[n \to +\infty]{} \frac{1000n}{200n} = 5$$

Generalization

- Let's assume a k-stage pipeline with the delays $t_1, t_2, \ldots, t_k$, respectively
- Also assume: $t_1 + t_2 + \cdots + t_k = T$
- Use a program with n instruction as a benchmark. Find the speedup of the pipelined implementation over SC implementation.

Generalization

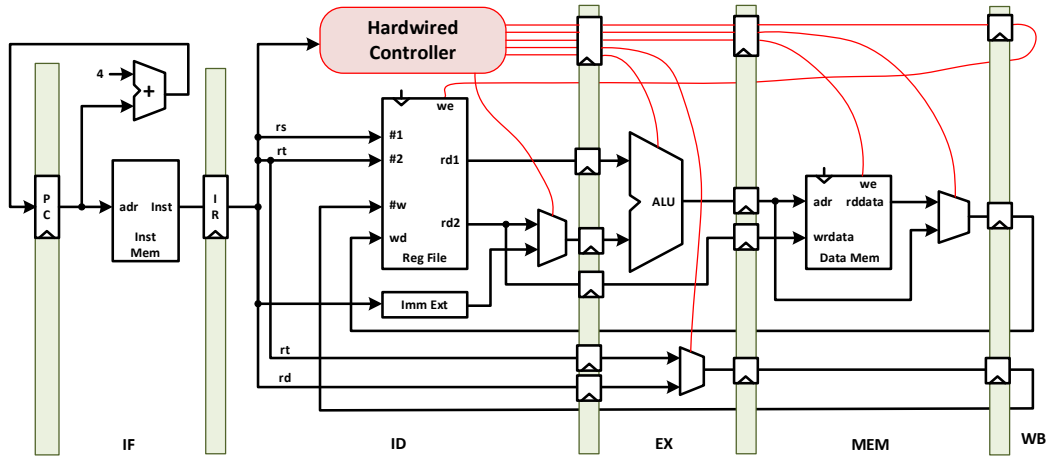
$$T_{SC} = T \times n$$

$$T_{Pipe} = k \times Max(t_1, \dots, t_k) + (\mathrm{n} - 1) \times Max(t_1, \dots, t_k)$$

$$Speedup = \frac{T_{SC}}{T_{Pipe}} = \frac{T \times n}{k \times Max(t_1, \dots, t_k) + (\mathrm{n} - 1) \times Max(t_1, \dots, t_k)}$$

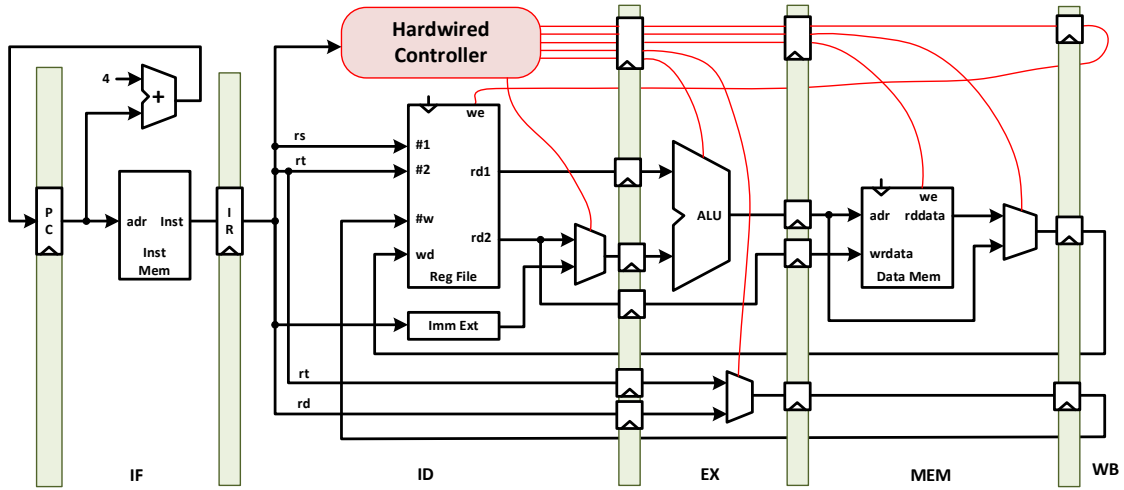
$$Speedup \xrightarrow[n \to +\infty]{} \frac{T}{Max(t_1, \dots, t_k)}$$

Pipeline Diagrams: Instructions vs. Time



Inst	C1	C2	C3	C4	C5	C6	C7	C8	…
Inst1	IF_1	ID_1	EX_1	MEM_1	WB_1				
Inst2		IF_2	ID_2	EX_2	MEM_2	WB_2			
Inst3			IF_3	ID_3	EX_3	MEM_3	WB_3		
Inst4				IF_4	ID_4	EX_4	MEM_4	WB_4	
Inst5					IF_5	ID_5	EX_5	MEM_5	WB_5

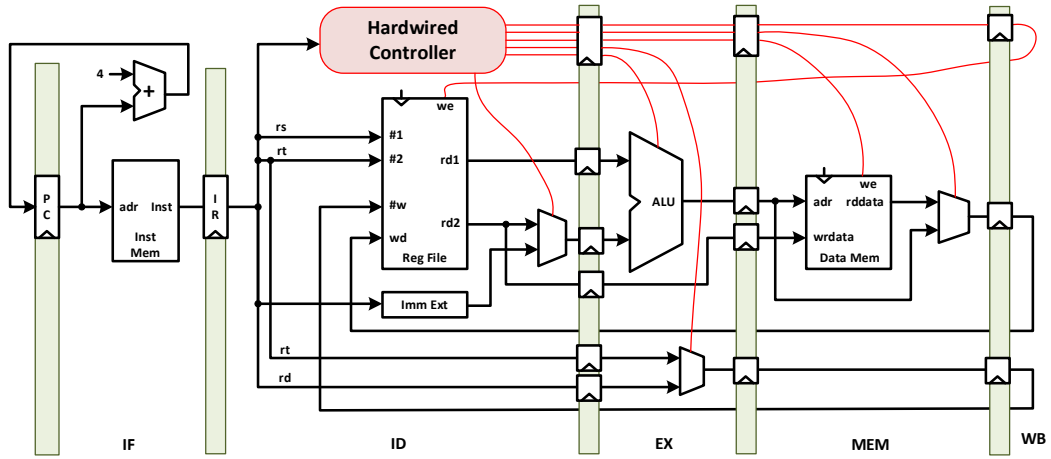
Pipeline Diagrams: Space vs. Time



Time	C0	C1	C2	C3	C4	C5	C6	C7	…
IF	$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$				
ID		$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$			
EX			$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$		
MEM				$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$	
WB					$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$

Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Pipeline Diagrams: Space vs. Time



Inst	C1	C2	C3	C4	C5	C6	C7	C8	…
ID	$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$				
ID		$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$			
EX			$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$		
MEM				$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$	
WB					$Inst_1$	$Inst_2$	$Inst_3$	$Inst_4$	$Inst_5$

Hazards

- **Structural hazard:** An instruction in the pipeline needs a resource being used by another instruction in the pipeline
- **Data hazard:** An instruction depends on a data value produced by an earlier instruction in the pipeline
- **Control hazard:** An instruction depends on a control decision made by an earlier instruction in the pipeline

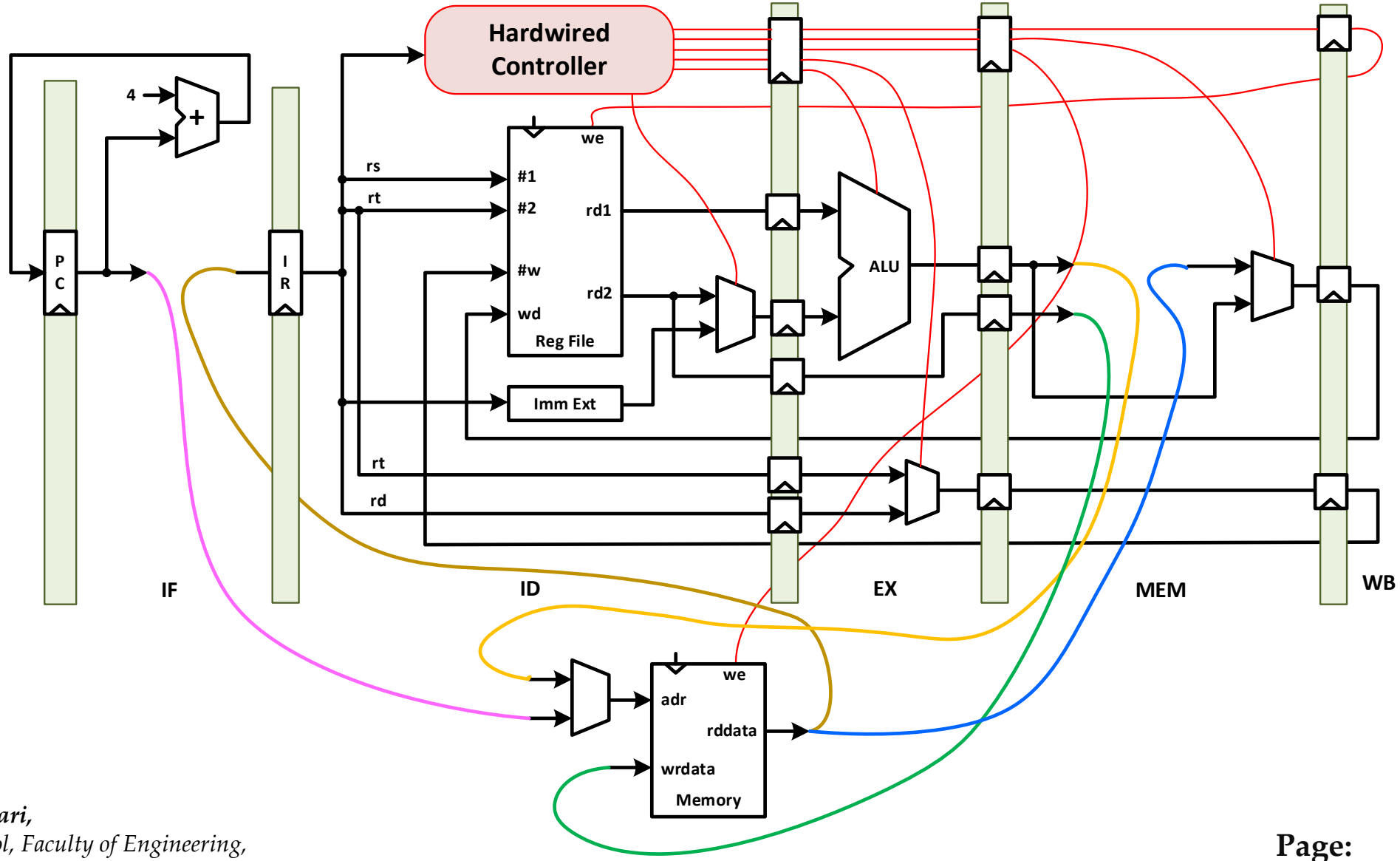
Structural Hazard

- **Schedule:** Programmer explicitly avoids scheduling instructions that would create structural hazards
- **Stall:** Hardware includes control logic that stalls until the earlier instruction is no longer using contended resource
- **Duplicate:** Add more hardware so that each instruction can access independent resources at the same time

Structural Hazard

- Simple 5-stage MIPS pipeline has no structural hazards specifically because ISA was designed that way

Example 1: Unified Memory

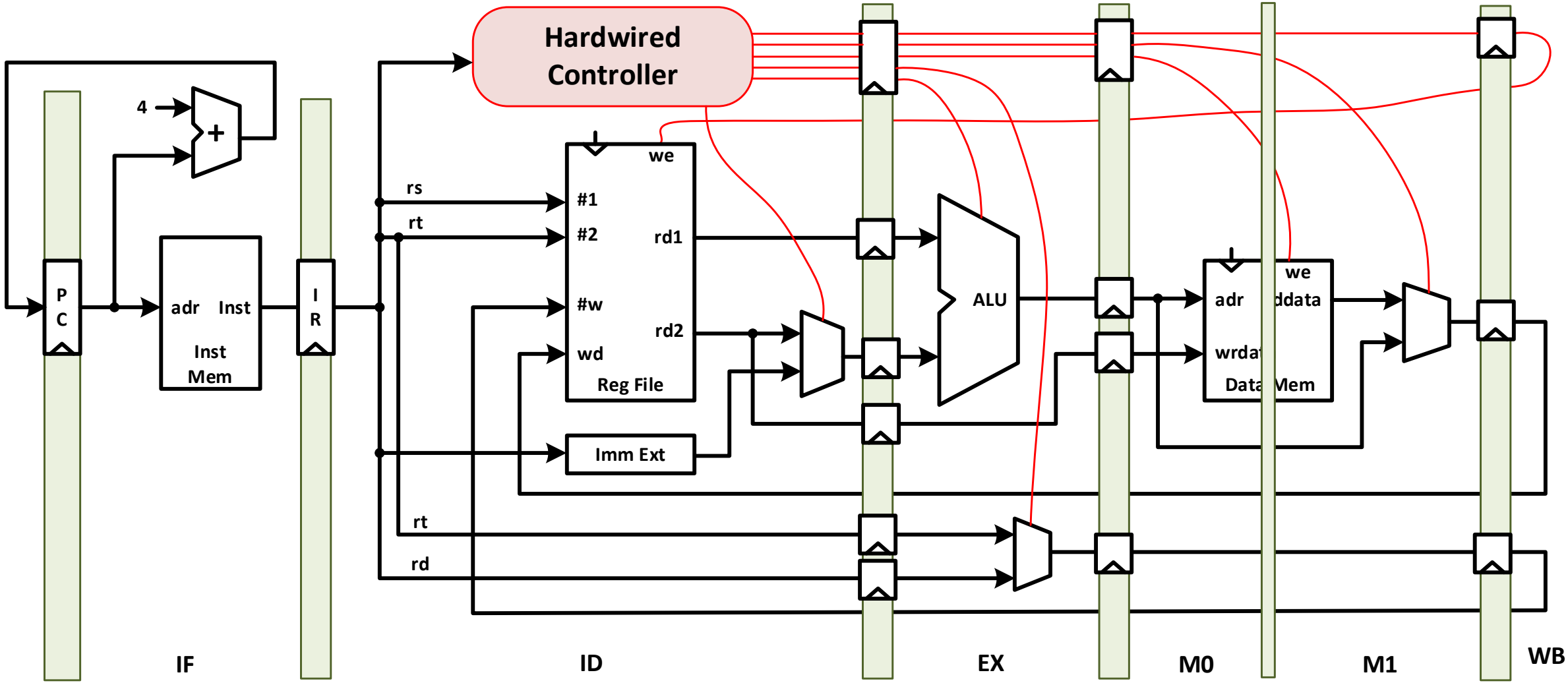


Saeed Safari,
ECE School, Faculty of Engineering,
University of Tehran

Example 1: Unified Memory

Inst	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
LW	F	D	X	M	W							
ADD		F	D	X	M	W						
ADD			F	D	X	M	W					
ADD				–	F	D	X	M	W			

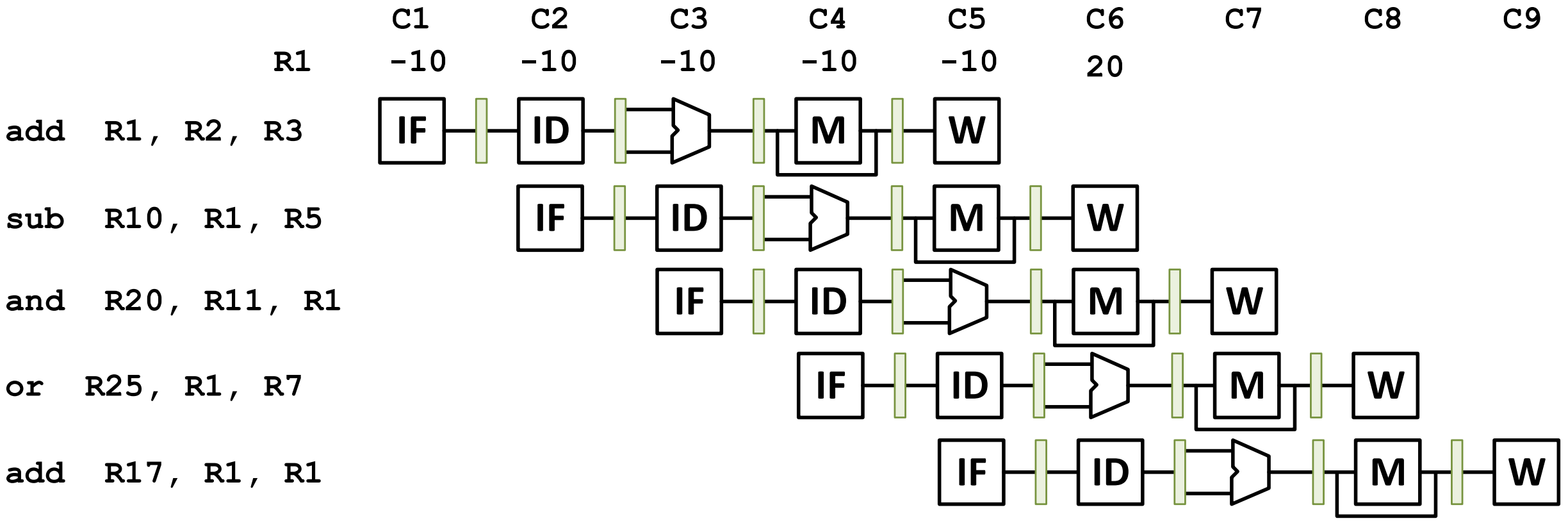
Example 2: 2-Cycle Memory



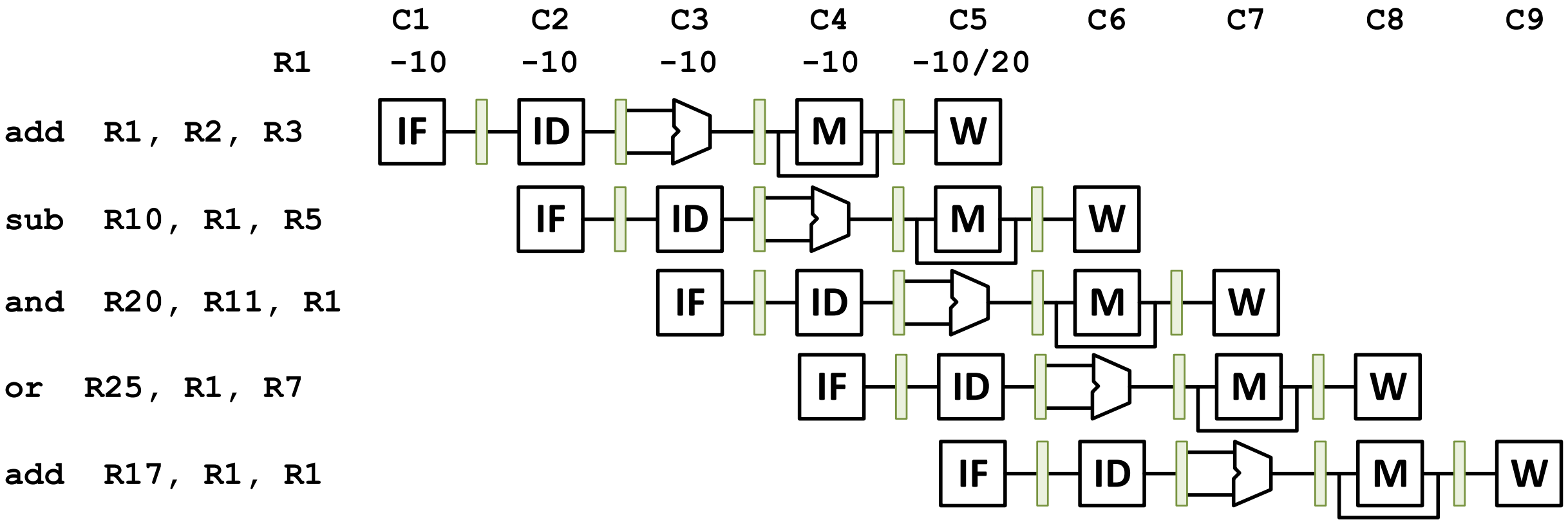
Example 2: 2-Cycle Memory

Inst	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12
ADD	F	D	X	M0	M1	W						
ADD		F	D	X	M0	M1	W					
LW			F	D	X	M0	M1	W				
LW				F	D	X	–	M0	M1	W		
ADD					F	D	–	X	M0	M1	W	

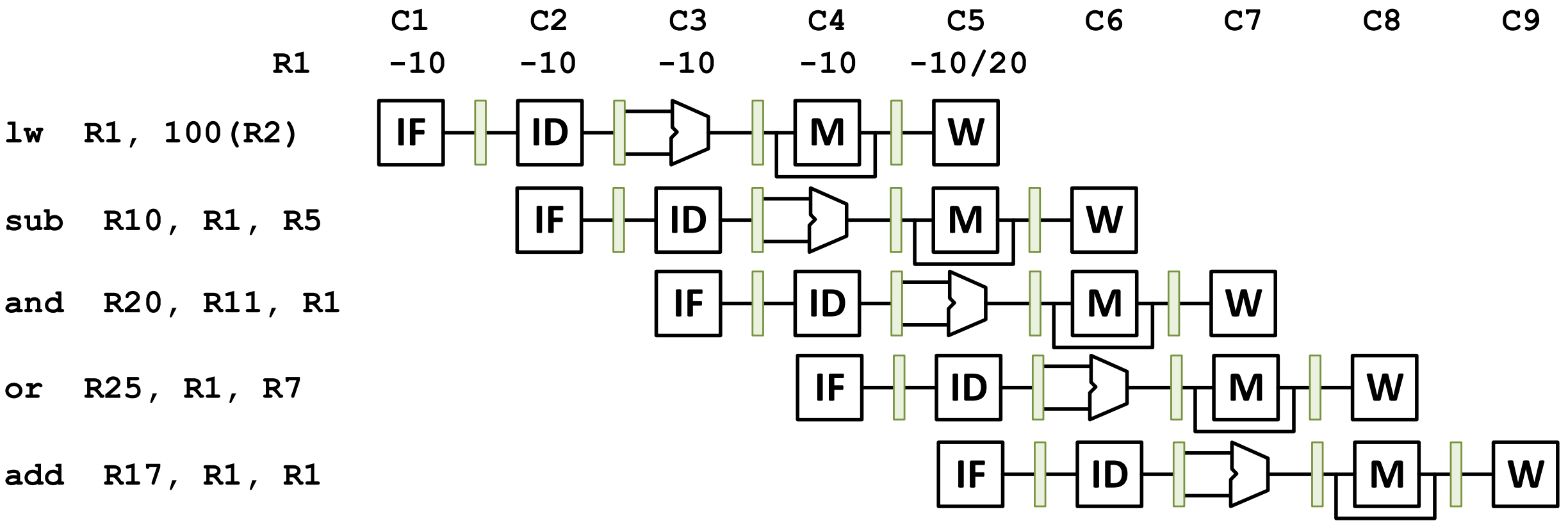
Data Hazard


C1 C2 C3 C4 C5 C6 C7 C8 C9
R1 -10 -10 -10 -10 -10 20
add R1, R2, R3
sub R10, R1, R5
and R20, R11, R1
or R25, R1, R7
add R17, R1, R1
IF ID M W

Data Hazard


C1
C2
C3
C4
C5
C6
C7
C8
C9
R1
-10
-10
-10
-10
-10/20
add R1, R2, R3
sub R10, R1, R5
and R20, R11, R1
or R25, R1, R7
add R17, R1, R1
IF
ID
M
W

Data Hazard


C1
C2
C3
C4
C5
C6
C7
C8
C9
R1
-10
-10
-10
-10
-10/20
lw R1, 100(R2)
sub R10, R1, R5
and R20, R11, R1
or R25, R1, R7
add R17, R1, R1
IF
ID
M
W

Data Hazard

- **Schedule:** Programmer explicitly avoids scheduling instructions that would create data hazards
- **Stall:** Hardware includes control logic that freezes earlier stages until preceding instruction has finished producing data value
- **Bypass:** Hardware datapath allows values to be sent to an earlier stage before preceding instruction has left the pipeline
- **Speculate:** Guess that there is not a problem, if incorrect kill speculative instruction and restart