Module 3: Processor Design
Pipelined MIPS

Module 3.1 Pipelining

1. Intro
2. MIPS
3. Pipelining. **Patterson. Section 4.6 or Hennessy. Apendix C.3**
4. Structural Hazards
5. Data Hazards. **Patterson. Section 4.7 or Hennessy. Apendix C.3**
6. Control Hazards. **Patterson. Section 4.8 or Hennessy. Apendix C.3**
7. Control Hazards with RAW
8. Summary

- **References**

– Hennessy, J. L., Patterson, D. “Computer Architecture: A Quantitative Approach”, 5th ed., Morgan Kaufmann, 2012. ISBN 978-0-12-383872-8. Apendix C.3

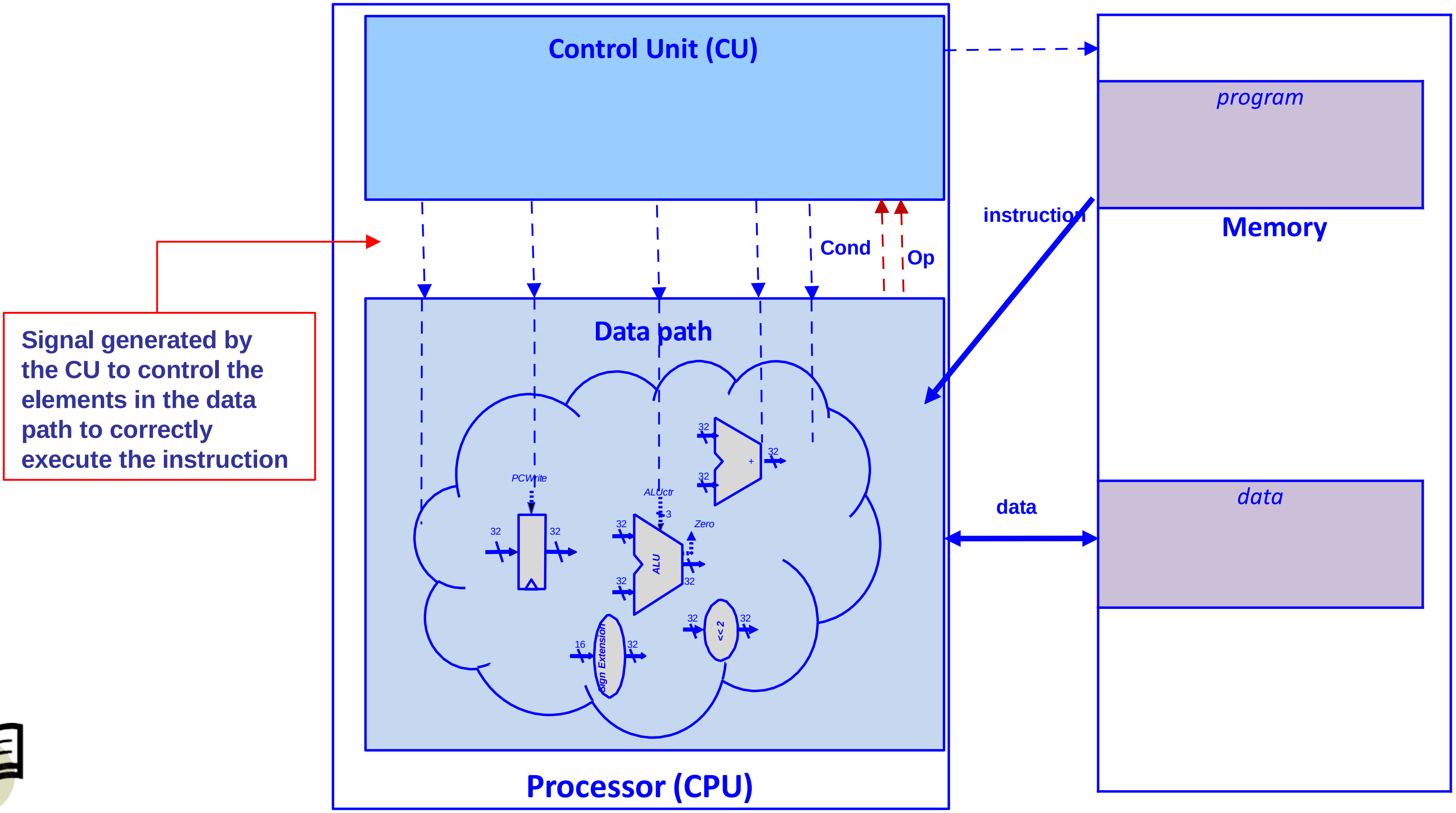
Patterson, D . Hennessy, J. L. “Computer Organization and Design” , 5th ed., Morgan Kaufmann, 2014, ISBN 0780124077263. Chapter 4.5-4.8

1. Intro

General diagram of the computer architecture: Von Neumann Model

Instructions are executed on the data path

The CU drives the signals that operate on the elements in the data path



Signal generated by the CU to control the elements in the data path to correctly execute the instruction

2. MIPS: Implemented Instruction Set

Data processing instructions: operands in registers (type R)

add rd, rs, rt	rd ←rs + rt, PC ←PC + 4
sub rd, rs, rt	rd ←rs - rt , PC ←PC + 4
and rd, rs, rt	rd ←rs and rt , PC ←PC + 4
or rd, rs, rt	rd ←rs or rt , PC ←PC + 4
slt rd, rs, rt	(if (rs < rt) then (rd ←1) else (rd ←0)), PC ←PC+4

Instructions with memory references (type I)

lw rt, inmed(rs)	rt ←Memory(rs + SignExt(inmed)) , PC ←PC + 4
sw rt, inmed(rs)	Memory(rs + SignExt(inmed)) ←rt , PC ←PC + 4

Conditional branches (type I)

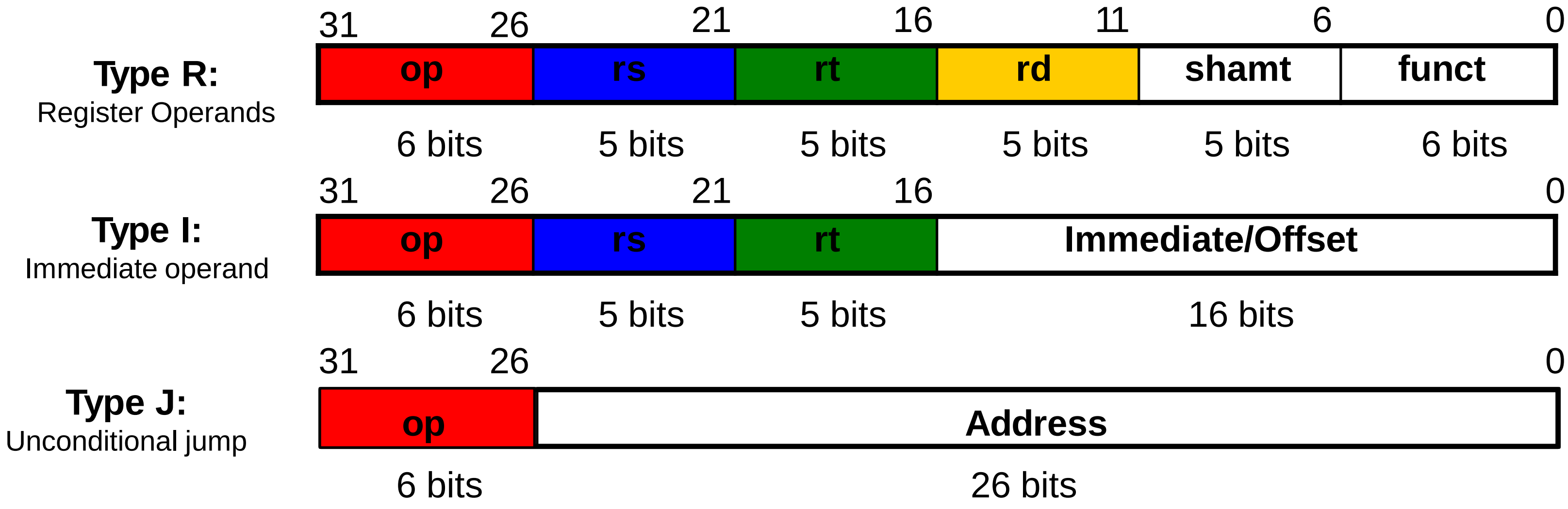
beq rs, rt, inmed	if (rs = rt) then (PC ←PC + 4 + 4·SignExt(inmed))
	else PC ←PC + 4

2. MIPS: Implemented Instruction Set

MIPS Architecture (DLX)

Fixed length instructions: 32 bits

Three different formats:



Type R: Register Operands

31 – 26	21	16	11	6	0
op	**rs**	**rt**	**rd**	**shamt**	**funct**
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Type I: Immediate operand

31 – 26	21	16	0
op	**rs**	**rt**	**Immediate/Offset**
6 bits	5 bits	5 bits	16 bits

Type J: Unconditional jump

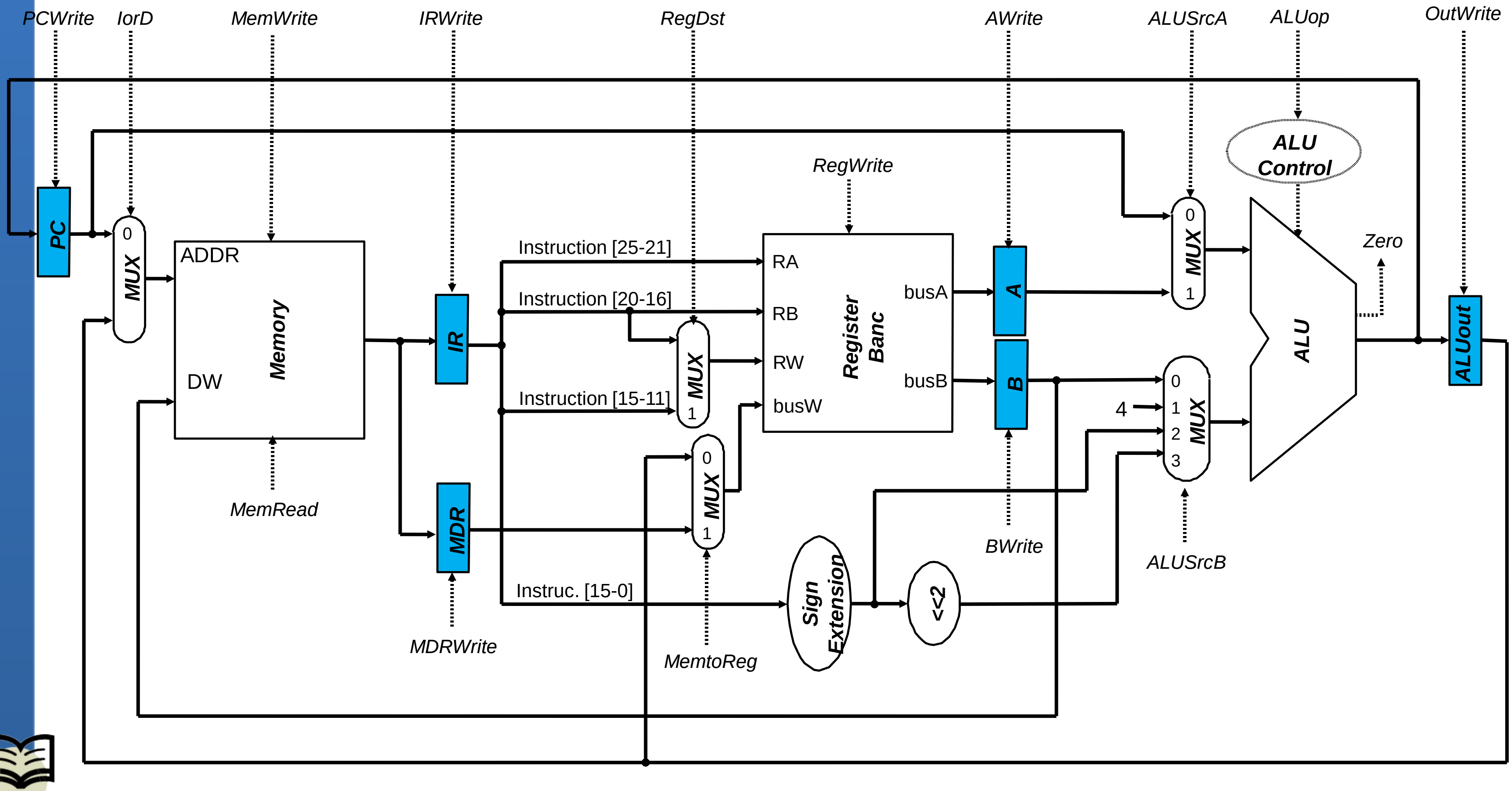
31 – 26	0
op	**Address**
6 bits	26 bits

Meaning of each field:

- **op:** operation code
- **rs, rt, rd:** identify the source and destination registers
- **shamt:** shift amount (for shift operations)
- **funct:** selects the arithmetic operation to perform with the ALU
- **Immediate/Offset:** immediate opperand or offset for indirect addressing
- **Address:** target address for a branch

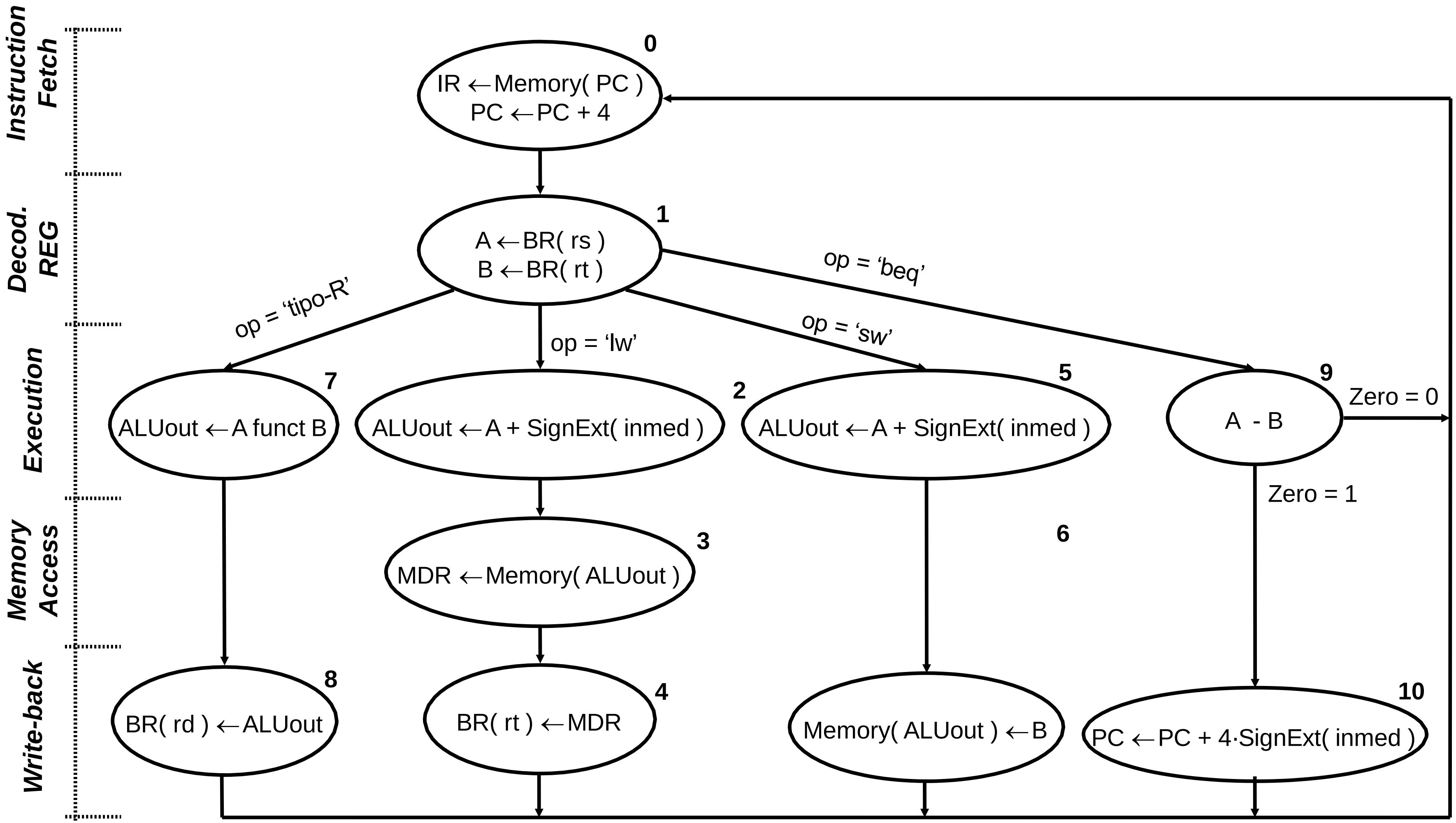
2. MIPS Multicycle implementation (introduction to computers)

Data path (Multicycle)



2. MIPS Multicycle control

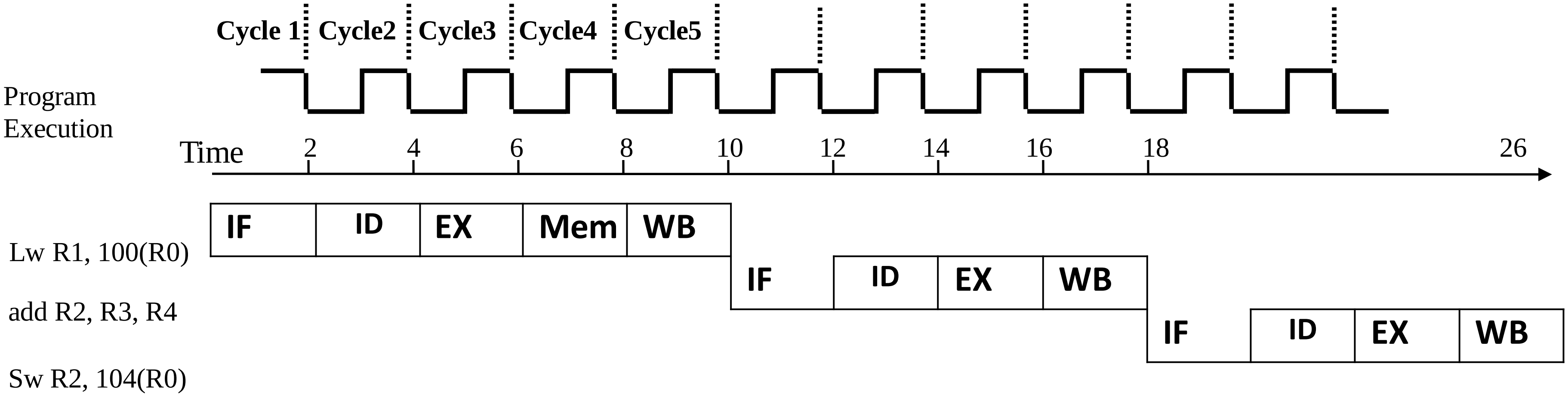
FSM diagram for the controller



2. MIPS Multicycle

Instruction execution

- **Load instructions 5 cycles**, they pass through all 5 the phases
- **The rest** of the instructions **4 cycles**, the do not pass through Memory Access Phase



Cycle 1 Cycle2 Cycle3 Cycle4 Cycle5

Program Execution

Time 2 4 6 8 10 12 14 16 18 26

Lw R1, 100(R0): IF ID EX Mem WB

add R2, R3, R4: IF ID EX WB

Sw R2, 104(R0): IF ID EX WB

26ns

3. Pipelining

How can we improve performance: Pipelining the processor

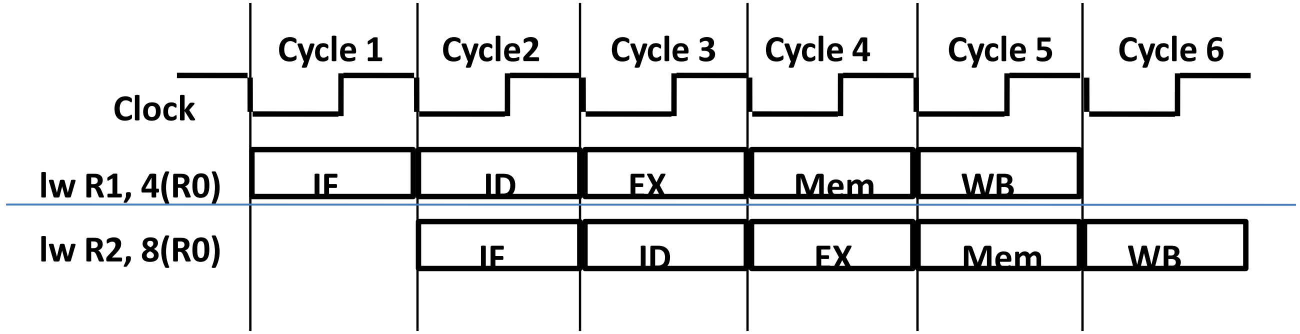
- Design technique that allows to overlap the execution of several instructions on different phases of their execution
  - **Each phase operates in parallel with the rest of the phases but on different instructions**

- It requires that each phases uses different hardware components from the data path
  - **It exploits instruction level parallelism**

3. Pipelining

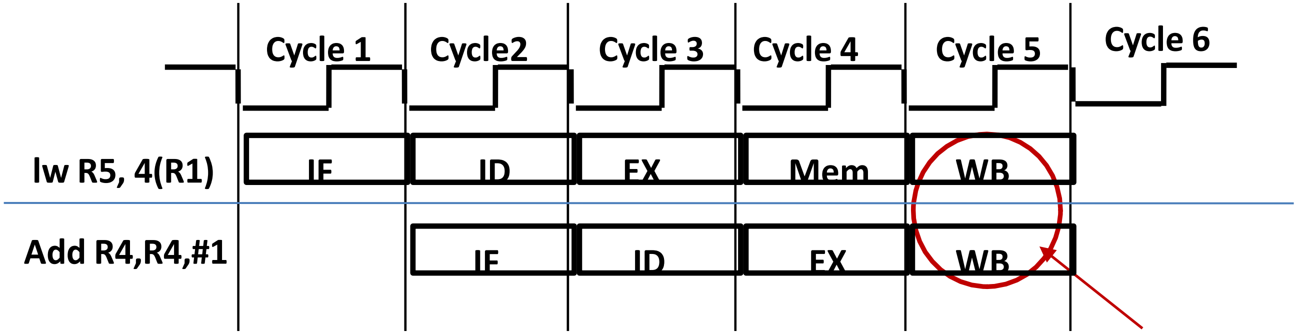
Instruction execution on pipelined processor

- Example 1



	Cycle 1	Cycle2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
Clock						
lw R1, 4(R0)	IF	ID	EX	Mem	WB	
lw R2, 8(R0)		IF	ID	EX	Mem	WB

- Example 2



	Cycle 1	Cycle2	Cycle 3	Cycle 4	Cycle 5	Cycle 6
lw R5, 4(R1)	IF	ID	EX	Mem	WB	
Add R4,R4,#1		IF	ID	EX	WB	

Conflict

3. Pipelining

Pipelined vs Multicycle

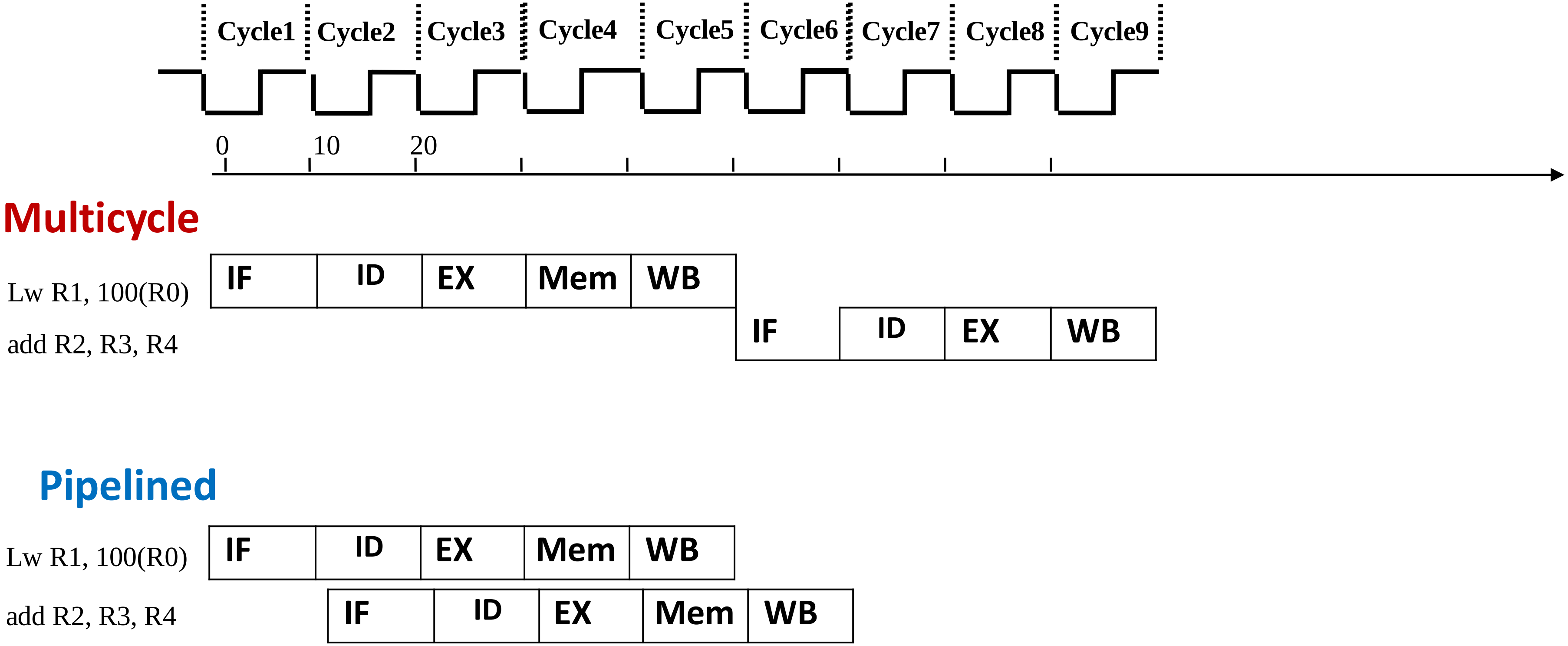
For the execution of **100 instructions**

- **Multicycle**

  10ns/cycle x 4.6 CPI x 100inst = **4600ns**

- **Pipelined**

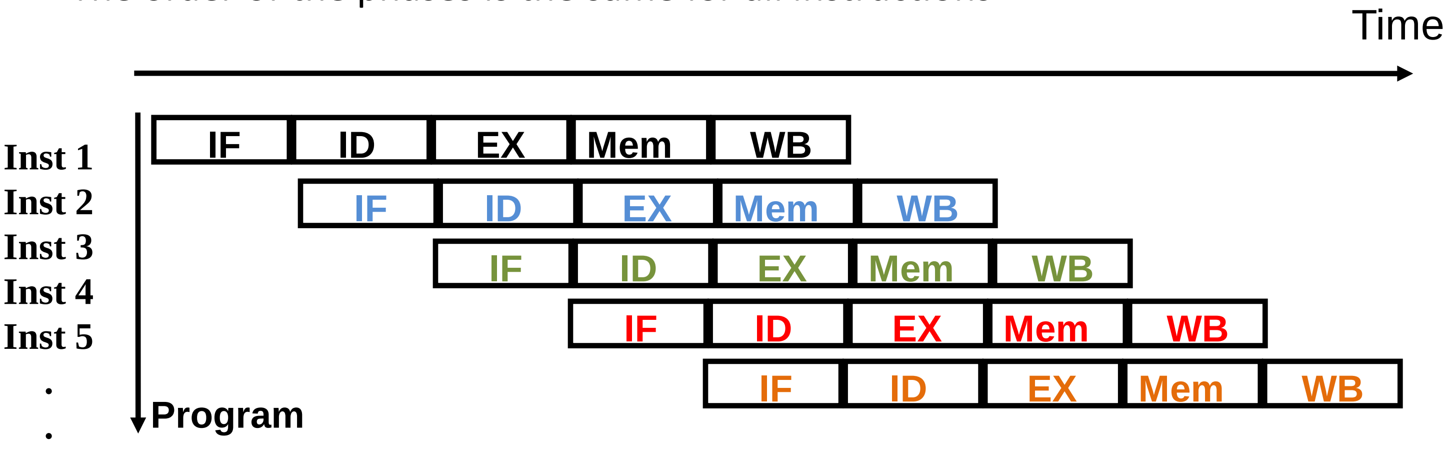
  10ns/cycle x (1CPIx100inst + 4 fill) =**1040ns**



3. Pipelining

How can we avoid this conflict?

- We cannot have more than one instruction on each phase
- **ALL instructions must go through ALL execution phases**
  - **The clock cycle is constraint by the slowest phase**
- The order of the phases is the same for all instructions



Time

Inst 1 IF ID EX Mem WB

Inst 2 IF ID EX Mem WB

Inst 3 IF ID EX Mem WB

Inst 4 IF ID EX Mem WB

Inst 5 IF ID EX Mem WB

.

.

Program

- From cycle 5
  - The processor finishes the execution of one instruction each cycle
  - **CPI=1**
- **The first 4 cycles are called filling cycles, and we can generally disregard them.**
- **CPI_{Ideal}=1**

3. Pipelining

What hinders pipelining?

- **Hazards**
  - Some situations appear that make it impossible to start the execution of an instruction every cycle
  - **Types:**
    - **Structural**
      - When two instructions want to use the same resource simultaneously
    - **Data**
      - An instruction tries to use a data that is not ready. Read-Write order must be preserved.
    - **Control**
      - An instruction tries make a decision based on a condition not yet evaluated

  Hazards must be *detected* and *solved*

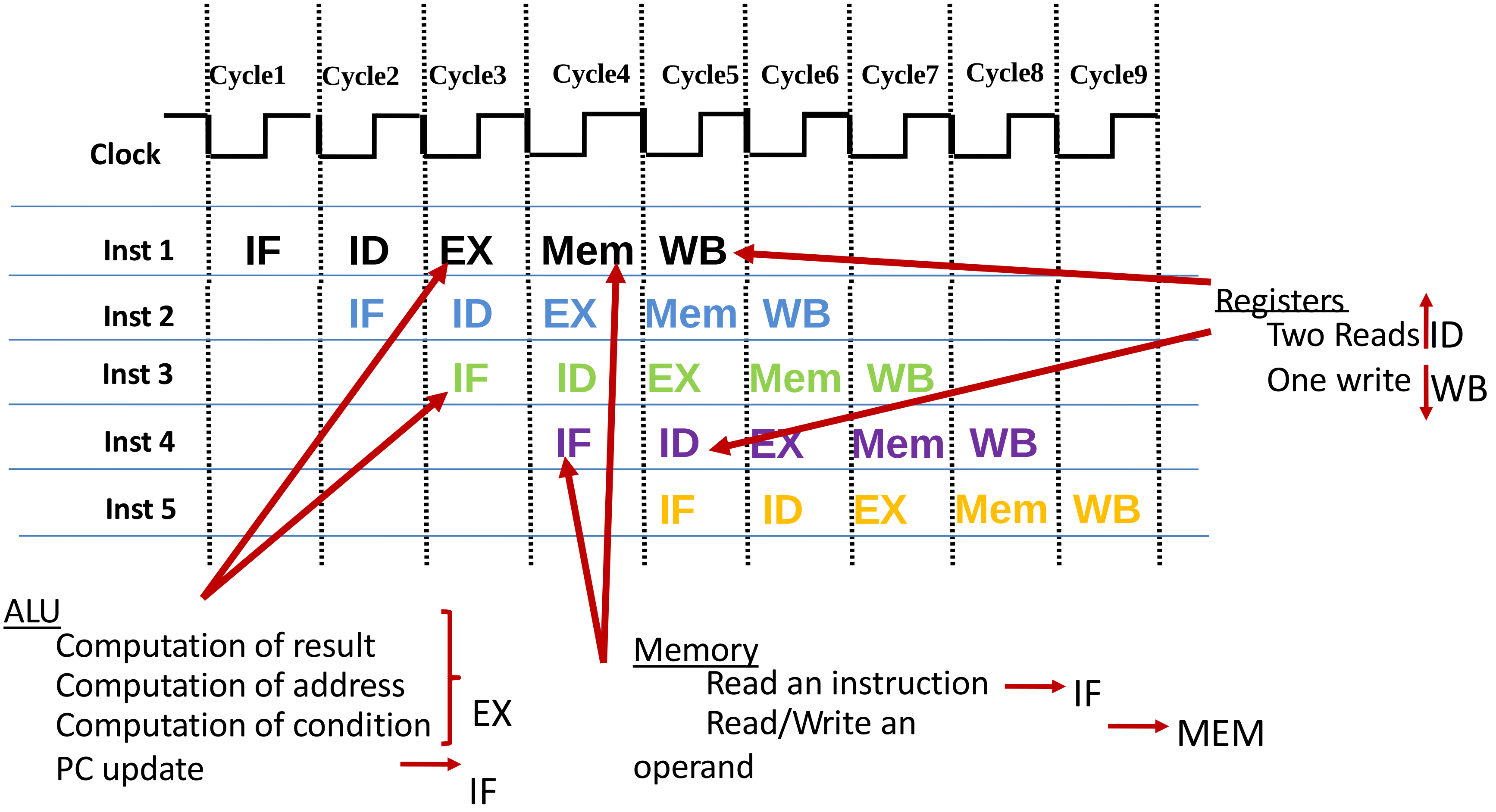
- **Interrupt management**

- **Out of order execution**

4. Pipelining: Structural Hazards

Structural Hazards

They happen when two instructions try to use the same resource simultaneously



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9
Clock									
Inst 1	IF	ID	EX	Mem	WB				
Inst 2		IF	ID	EX	Mem	WB			
Inst 3			IF	ID	EX	Mem	WB		
Inst 4				IF	ID	EX	Mem	WB	
Inst 5					IF	ID	EX	Mem	WB

ALU
Computation of result
Computation of address
Computation of condition
EX
PC update → IF

Memory
Read an instruction → IF
Read/Write an operand → MEM

Registers
Two Reads ID
One write WB

Goal:

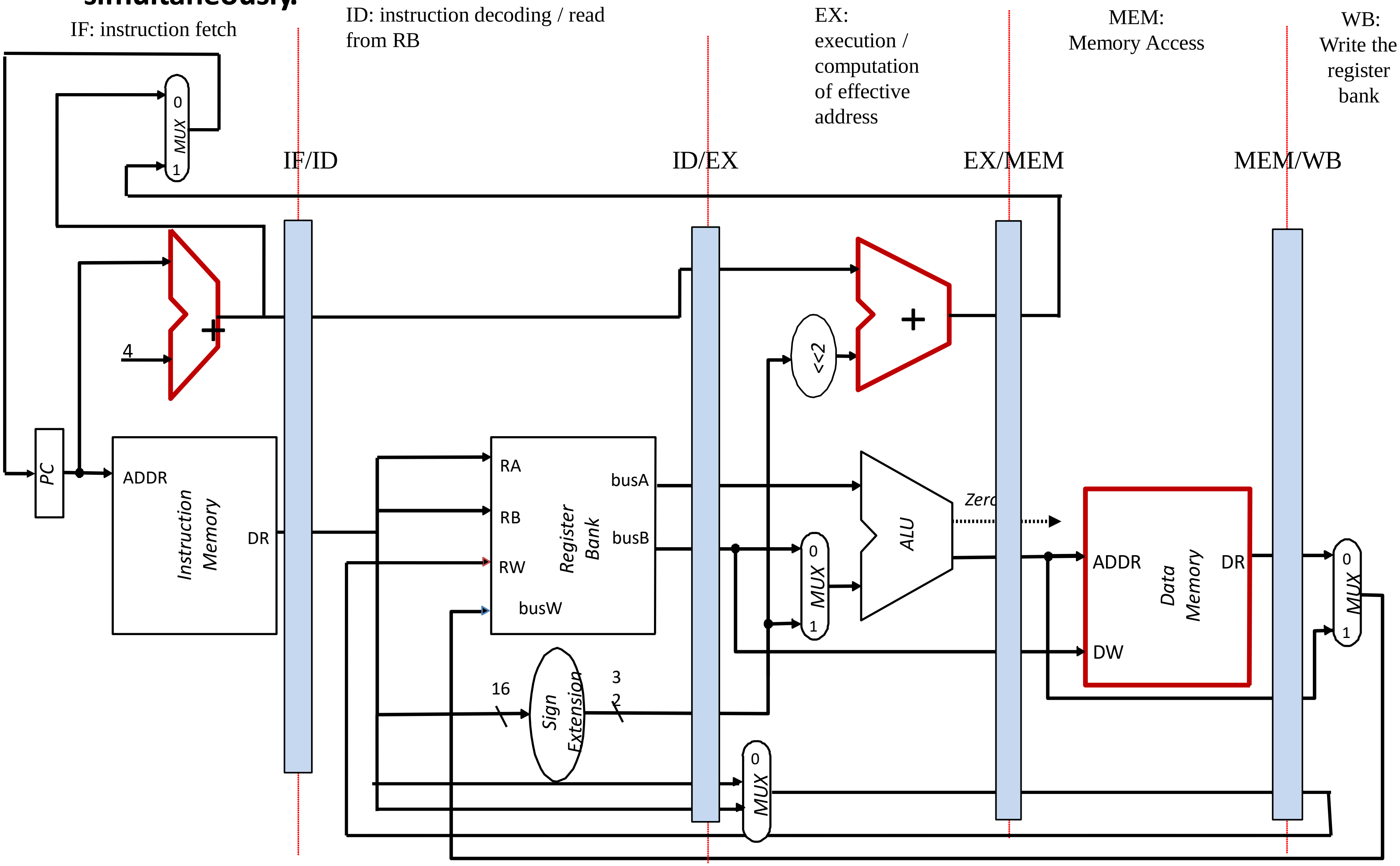
Execute without conflicts any combination of instructions

4. Pipelining: Structural Hazards

How to solve structural Hazards?

Duplicate all resources needed simultaneously.

The RB is not a problem as it has two read ports and one write port that can be used simultaneously.



IF: instruction fetch

ID: instruction decoding / read from RB

EX: execution / computation of effective address

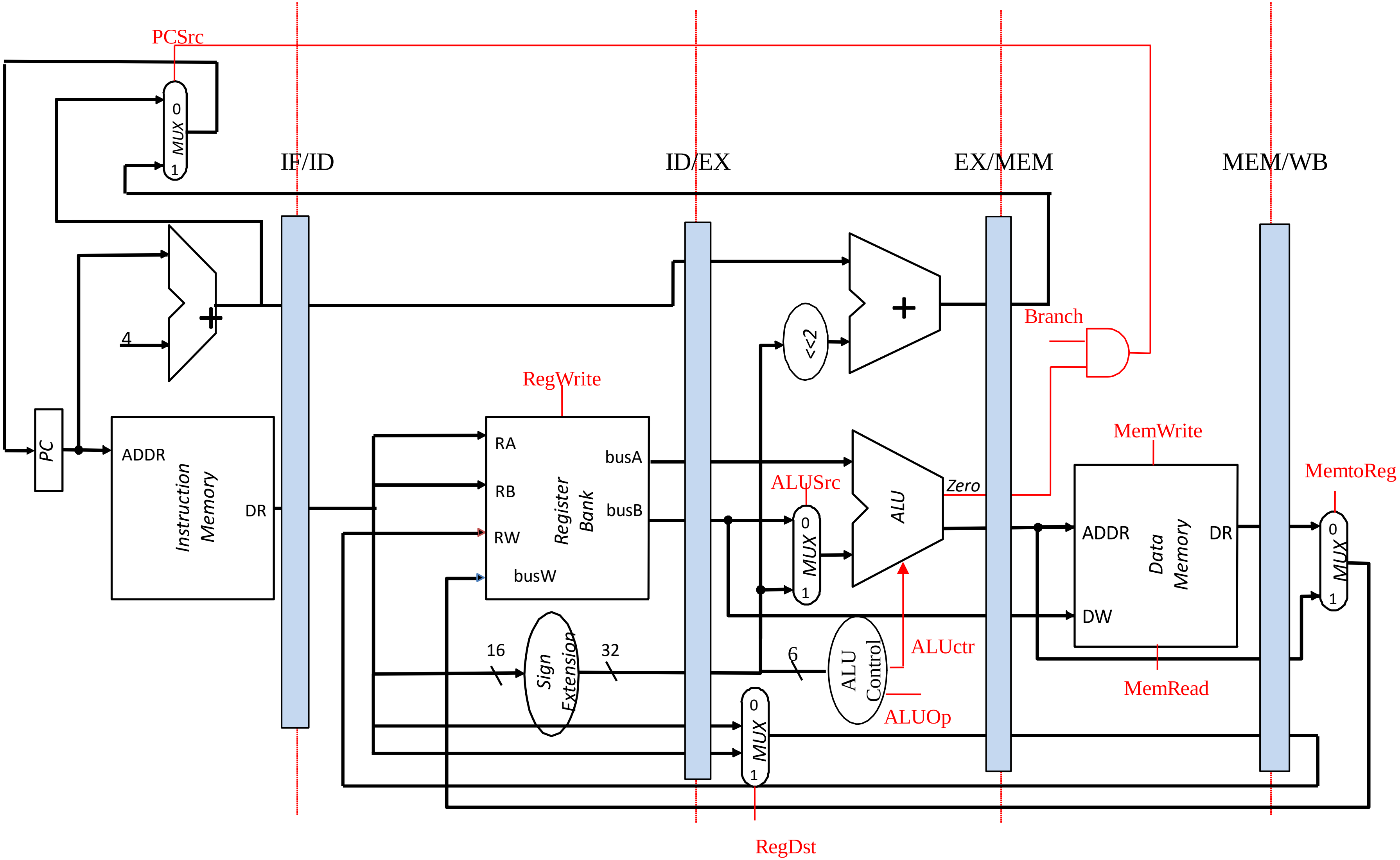
MEM: Memory Access

WB: Write the register bank

4. Pipelining: Structural Hazards

Control design on pipelined processor

The PC and the different decoupling registers (IF/ID …) must be loaded every cycle, so they do not need a specific load signal



Control unit design on pipelined processors

ALU control

op	funct	ALUop	ALUctr
100011 (lw)	XXXXXX	00	010+
101011 (sw)		00	010 +
000100 (beq)		01	110-
000000 (tipo-R)	100000 (add)	10	010+
	100010 (sub)	10	110-
	100100 (and)	10	000
	100101 (or)	10	001
	101010 (slt)	10	111

Main Control

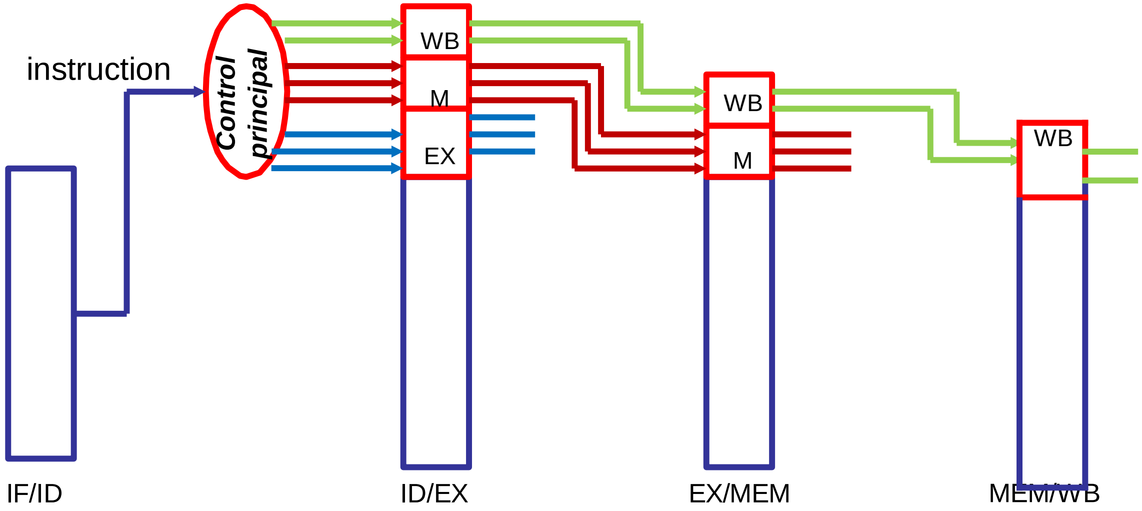
op	RegDst	ALUSrc	ALUop	MemRead	MemWrite	Branch	RegWrite	MemtoReg
100011 (lw)	0	1	00	1	0	0	1	0
101011 (sw)	X	1	00	0	1	0	0	X
000100 (beq)	X	0	01	0	0	1	0	X
000000 (tipo-R)	1	0	10	0	0	0	1	1

EX: Execution/ Address comput.

MEM: Memory Access

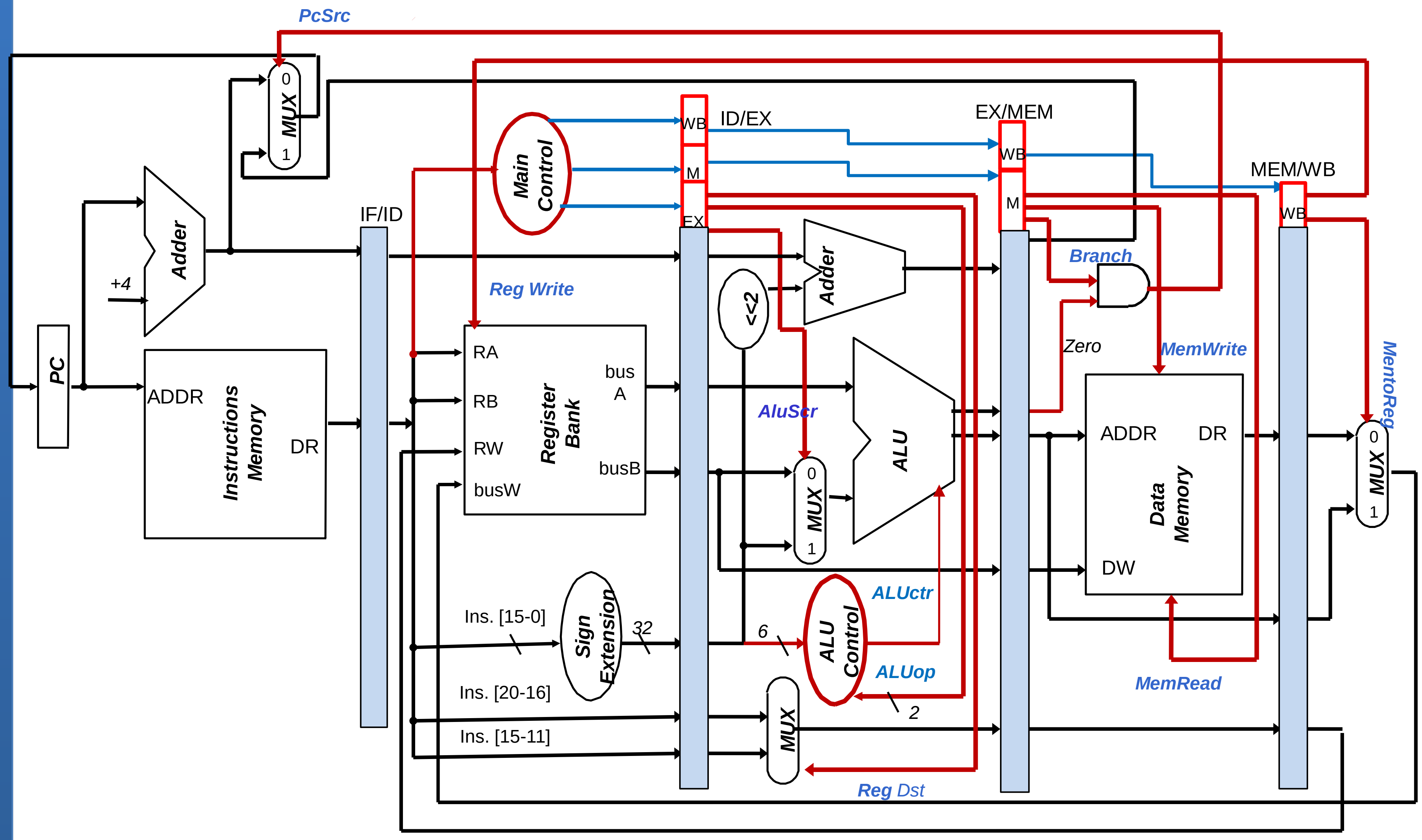
WB Write-Back

The ALU control is specified by ALUop (which depends on the instruction type) and the funct field of the R type instructions



4. Pipelining: Structural Hazards

Pipelined Datapath without structural Hazards



5. Pipelining : Data Hazards

Data Hazards:

- There is a hazard if a data dependency exists between instructions that exectue concurrently
- The number of data hazards increase with multicycle operations
- **Three** different types:
  - **Read After Write (RAW)**
  - **Write After Read (WAR)**
  - **Write After Write (WAW)**

5. Pipelining : Data Hazards

Read After Write (RAW)

*Add **r1**,r2,r3 – writes on register r1*

*Add r4,**r1**,r2— reads from register r1*

- There is a **hazard if r1 is read before the first instruction writes to it**

Write After Read (WAR)

*Add r1,**r4**,r3 – reads from register r4*

*Add **r4**,r5,r2—writes to register r4*

- There is a **hazard if r4 write to register r4 before the first instruction reads from it**

Write After Write (WAW)

*Add **r4**,r2,r3 – wirtes to register r4*

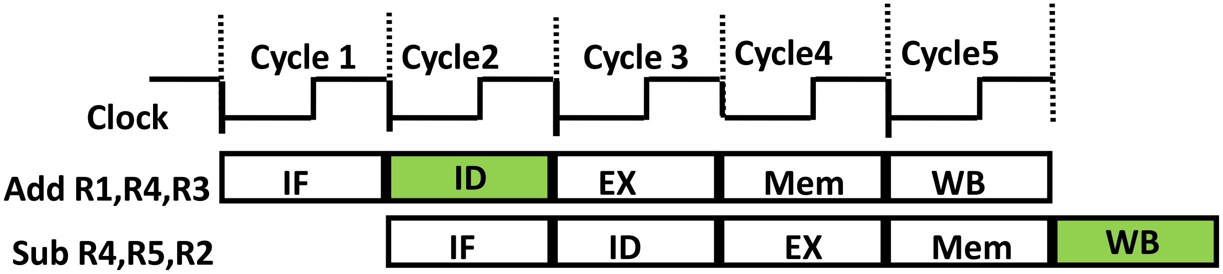
*Add **r4**,r1,r2—writes to register r4*

- There is a **hazard if the first instruction writes to r4 after the second**

5. Pipelining : Data Hazards

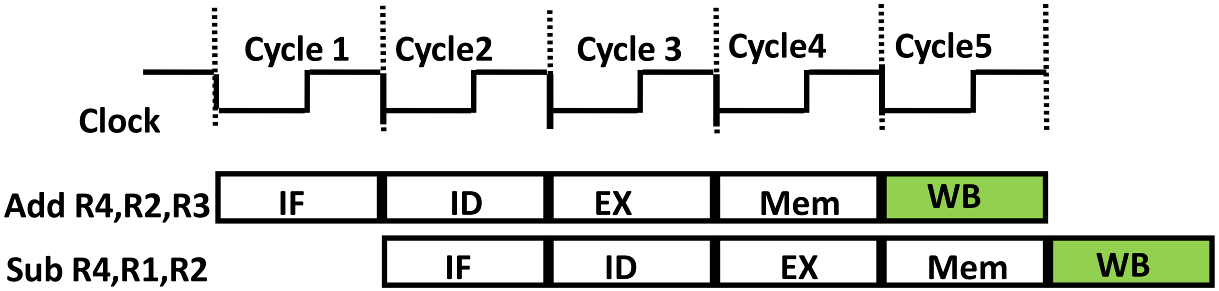
WAR and WAW:

- Write After Read: WAR



	Cycle 1	Cycle2	Cycle 3	Cycle4	Cycle5	
Clock						
Add R1,R4,R3	IF	ID	EX	Mem	WB	
Sub R4,R5,R2		IF	ID	EX	Mem	WB

- Write After Write: WAW



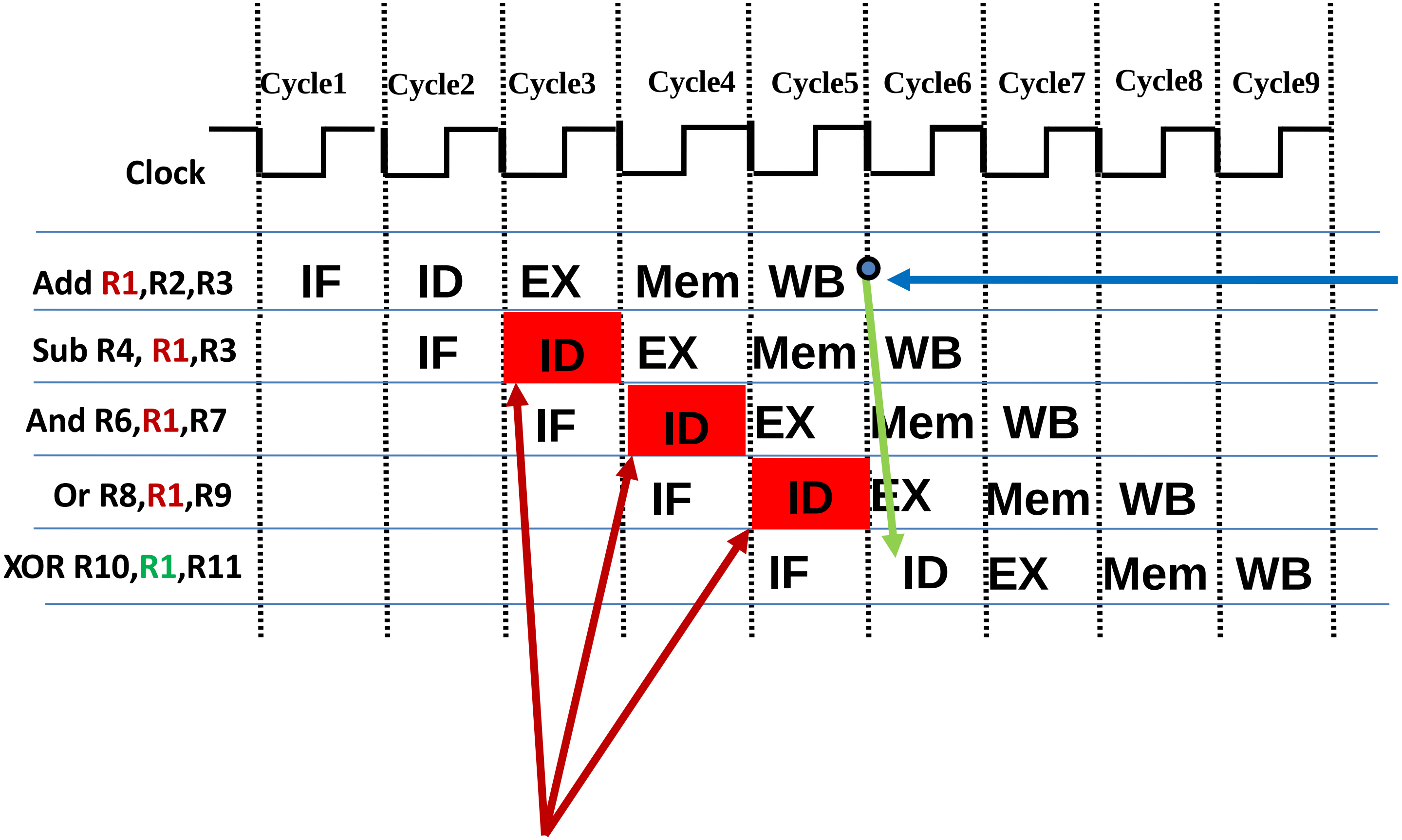
	Cycle 1	Cycle2	Cycle 3	Cycle4	Cycle5	
Clock						
Add R4,R2,R3	IF	ID	EX	Mem	WB	
Sub R4,R1,R2		IF	ID	EX	Mem	WB

- **These hazards do not appear in the integer DLX studied so far**
  - All registers are read after the second phase (ID)
  - Instructions do not write to the registers until the last phase (WB)
  - Instructions are executed and finalize in order

5. Pipelining: Data Hazards

RAW Hazards

- They appear in the following circumstances



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9
Add R1,R2,R3	IF	ID	EX	Mem	WB				
Sub R4, R1,R3		IF	ID	EX	Mem	WB			
And R6,R1,R7			IF	ID	EX	Mem	WB		
Or R8,R1,R9				IF	ID	EX	Mem	WB	
XOR R10,R1,R11					IF	ID	EX	Mem	WB

R1 is written in the rising edge that ends cycle 5 and starts cycle 6

The three instructions after the Add do not read the updated r1 value, they read the old value

5. Pipelining: Data Hazards

How can we solve the RAW hazards?

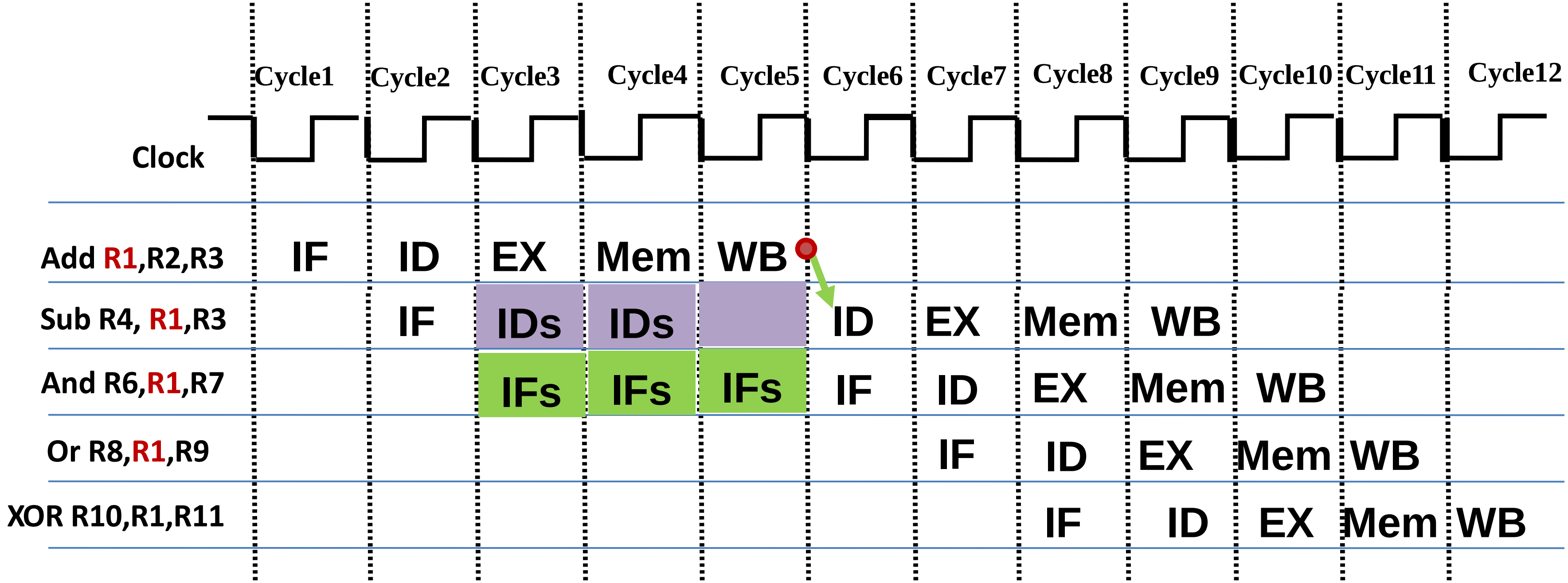
- **Solution 1: Stop/stall the pipeline**
  - **On which stage do we stop?**
    - Depends on the design of the data path
    - We will assume that we stop on Deco unless otherwise specified
  - **How do the pipeline stalls affect program execution?**
    - Instructions in stages before that of the stall must also stall
    - Instruction in stages that follow that of the stall do continue

- **Solution 2: Code reordering (compiler)**
  - **It is not always possible (can insert nops if no useful instructions can be found, which in the end is the same as stalling)**
  - Can minimize stalls, that is why compilers always try to reorder

5. Pipelining: Data Hazards

How can we solve RAW hazards?

- **Solution 1:** **Stall the pipeline** (in the decoding stage)



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9	Cycle10	Cycle11	Cycle12
Add R1,R2,R3	IF	ID	EX	Mem	WB							
Sub R4, R1,R3		IF	IDs	IDs		ID	EX	Mem	WB			
And R6,R1,R7			IFs	IFs	IFs	IF	ID	EX	Mem	WB		
Or R8,R1,R9							IF	ID	EX	Mem	WB	
XOR R10,R1,R11								IF	ID	EX	Mem	WB

3 wait cycles

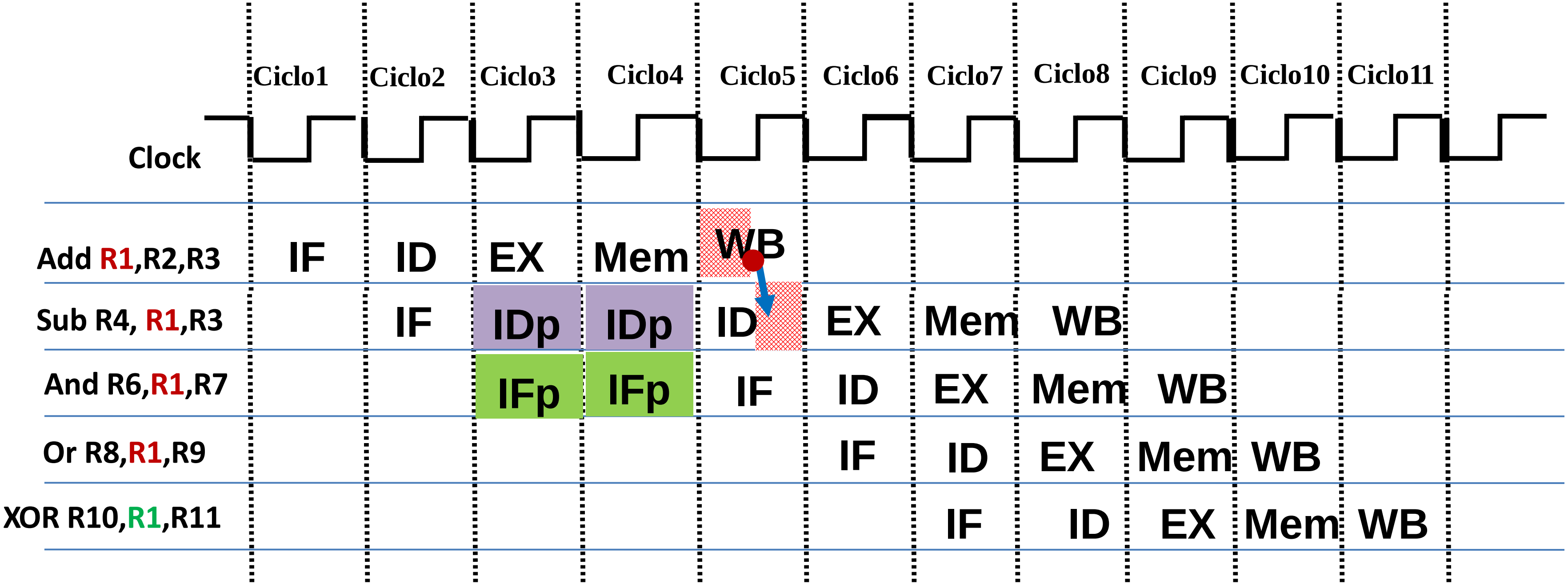
During which only the first instruction is executing

IDs stall because of RAW hazard between instruction 1 and 2

5. Pipelining: Data Hazards

How can we solve RAW hazards?

- **We modify the Register Bank**: writes are completed at mid cycle, on the falling edge of the clock, and half cycle is enough to read the updated value



	Ciclo1	Ciclo2	Ciclo3	Ciclo4	Ciclo5	Ciclo6	Ciclo7	Ciclo8	Ciclo9	Ciclo10	Ciclo11
Clock											
Add R1,R2,R3	IF	ID	EX	Mem	WB						
Sub R4, R1,R3		IF	IDp	IDp	ID	EX	Mem	WB			
And R6,R1,R7			IFp	IFp	IF	ID	EX	Mem	WB		
Or R8,R1,R9						IF	ID	EX	Mem	WB	
XOR R10,R1,R11							IF	ID	EX	Mem	WB

2 wait cycles

Only first instruction executing during those cycles

IDs Stall because of RAW between instruction 1 & instruction 2

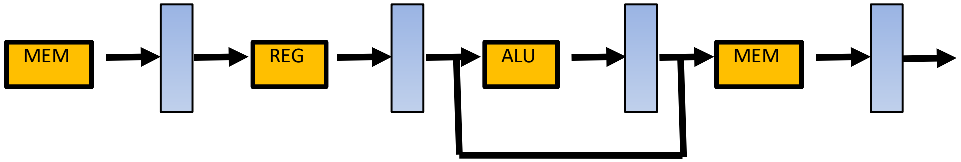
5. Pipelining : Data Hazards

How can we solve RAW Hazards?

- **Solution 3:** **Forwarding (or bypassing)**
  - **Send data as soon as it is computed to the stages that need it without waiting for it to be written on to the register bank**

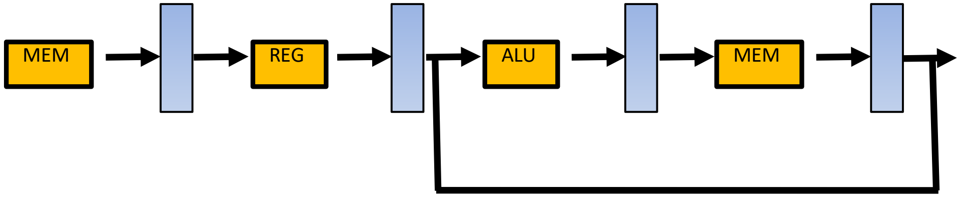
Case 1:

add **r1**,r2,r3
sub r4,**r1**,r3



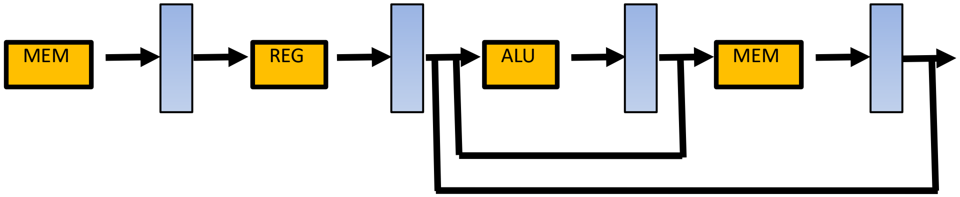
Case 2:

add **r1**,r2,r3
sub r4,r5,r3
and r6,**r1**



Case 3:

add **r1**,r2,r3
sub r4,**r1**,r3
and r6,**r1**



To implement forwarding we need two changes on the data path:

- From the pipeline EX/MEM register (ALU output) to the ALU input
- From the pipeline MEM/WB register (Mem output) to the ALU input

5. Pipelining : Data Hazards

How can we solve RAW Hazards?

– **Solution 3: Forwarding**

Information needed to implement forwarding:

Destination register in last stage (Rd for Type-R, Rt for Lw)

EX/MEM.Rd

MEM/WB.Rd

Registers read on the second stage (Rs & Rt)

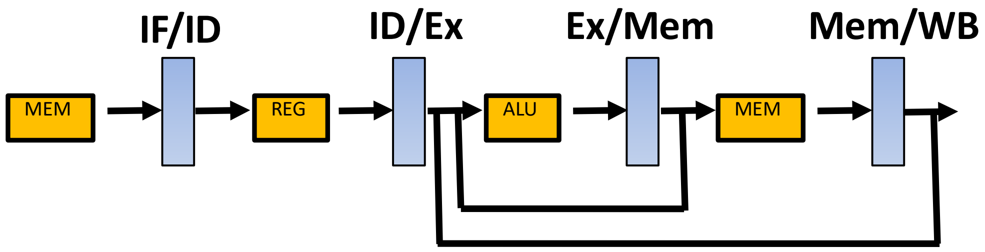
ID/EX.Rt

ID/EX.Rs

Indication of wether the instruction writes to the RB

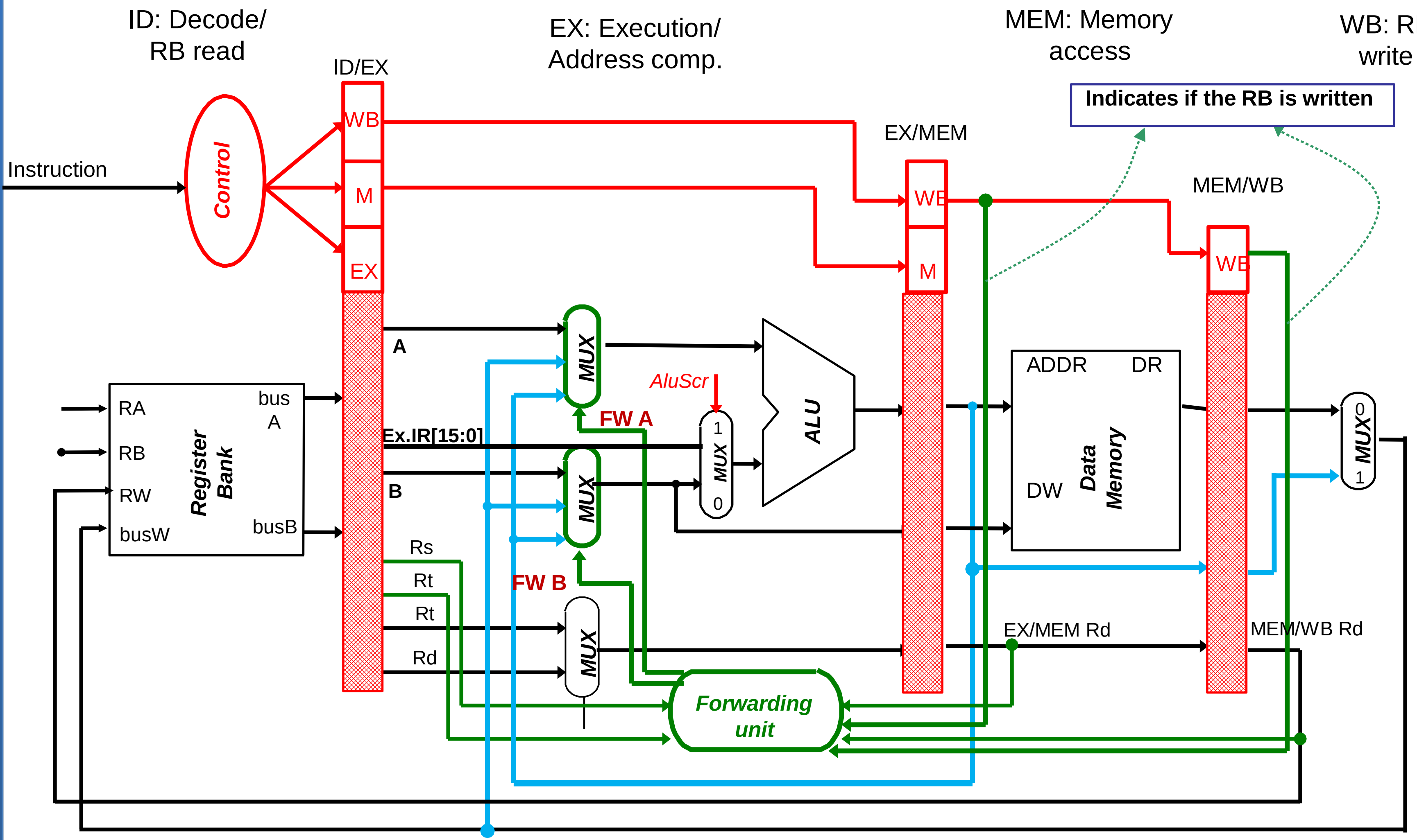
EX/MEM.RegWrite

MEM/WB.RegWrite



5. Pipelining : Data Hazards

RAW Hazards: Forwarding implementation

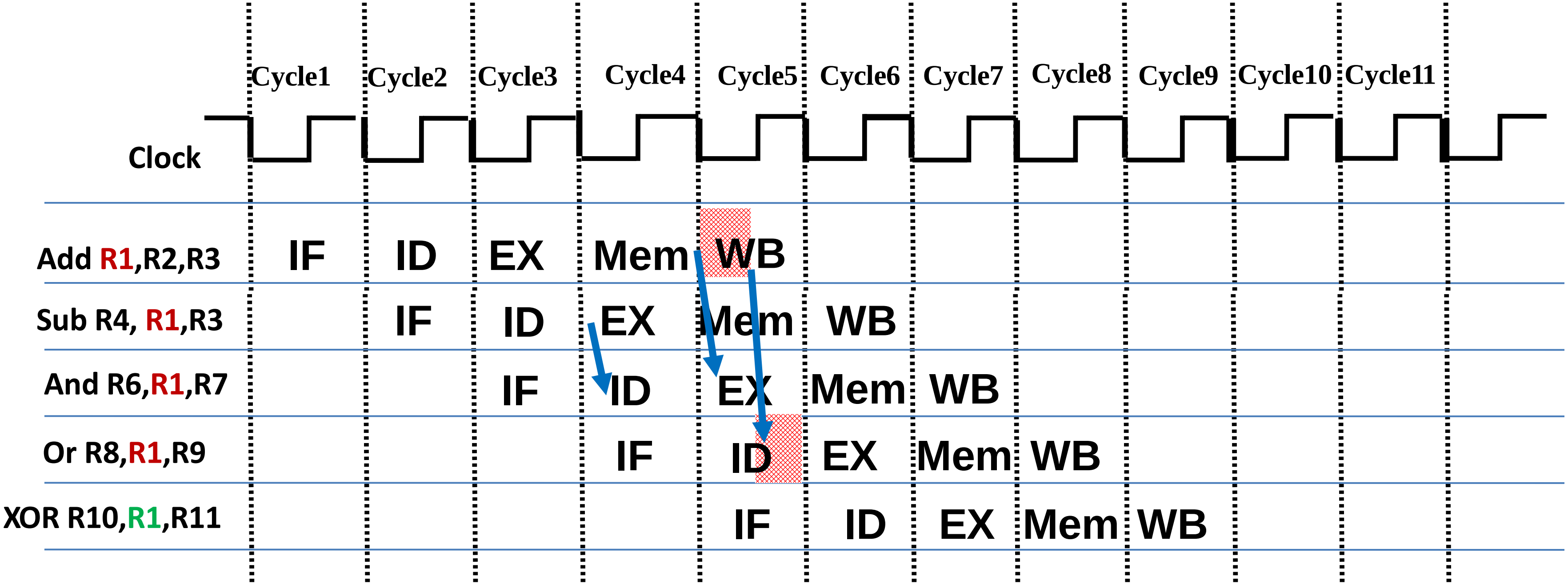


Control information in green, data in light blue

5. Pipelining: Data Hazards

How can we solve RAW hazards?

- Solution 3: Forwarding



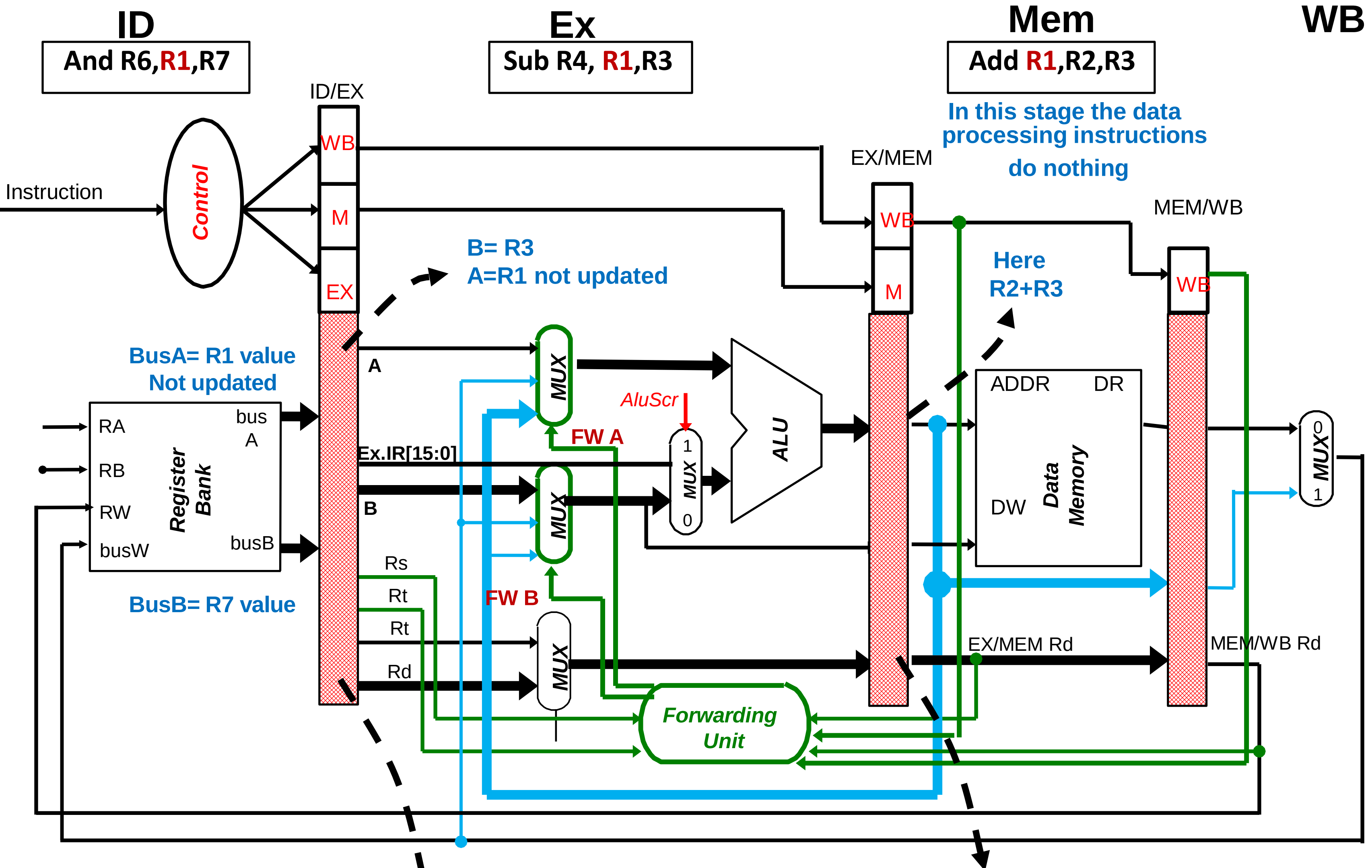
	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9	Cycle10	Cycle11
Add R1,R2,R3	IF	ID	EX	Mem	WB						
Sub R4, R1,R3		IF	ID	EX	Mem	WB					
And R6,R1,R7			IF	ID	EX	Mem	WB				
Or R8,R1,R9				IF	ID	EX	Mem	WB			
XOR R10,R1,R11					IF	ID	EX	Mem	WB		

No wait cycles

We can execute all the instructions without wait cycles

5. Pipelining : Data Hazards

Execution of the example: Cycle 4



In this stage the data processing instructions do nothing

B= R3
A=R1 not updated

Here R2+R3

BusA= R1 value Not updated

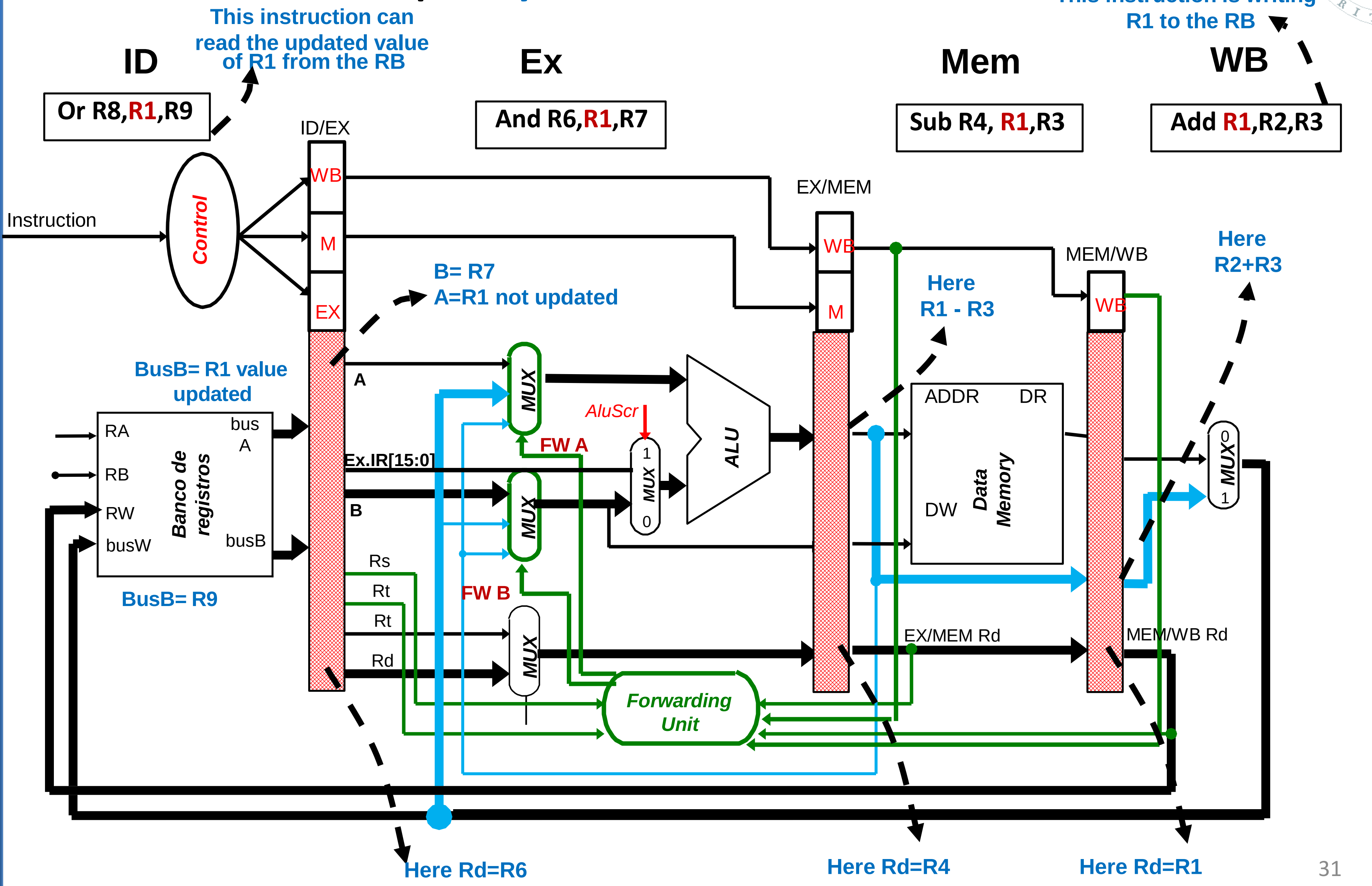
BusB= R7 value

Here Rd = R4

Here Rd = R1

5. Pipelining : Data Hazards

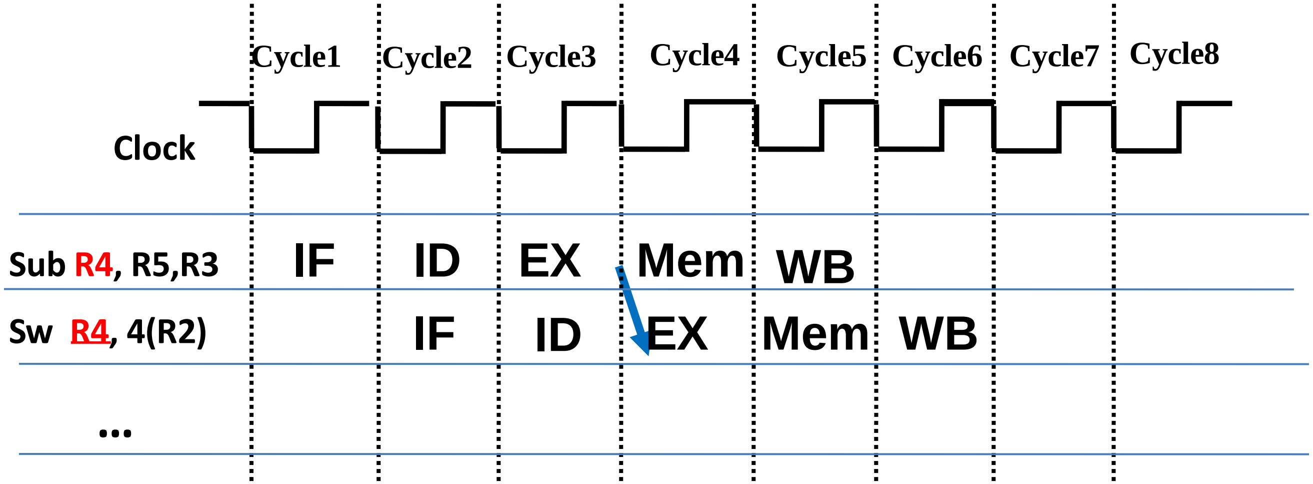
Execution of the example: Cycle 5



5. Pipelining: Data Hazards

RAW with a store instruction

- The store instruction writes to memory in the Mem stage, but the data written is read from the RB in the ID stage
  - **Can make use of Forwarding, replacing in ex the value read in ID**

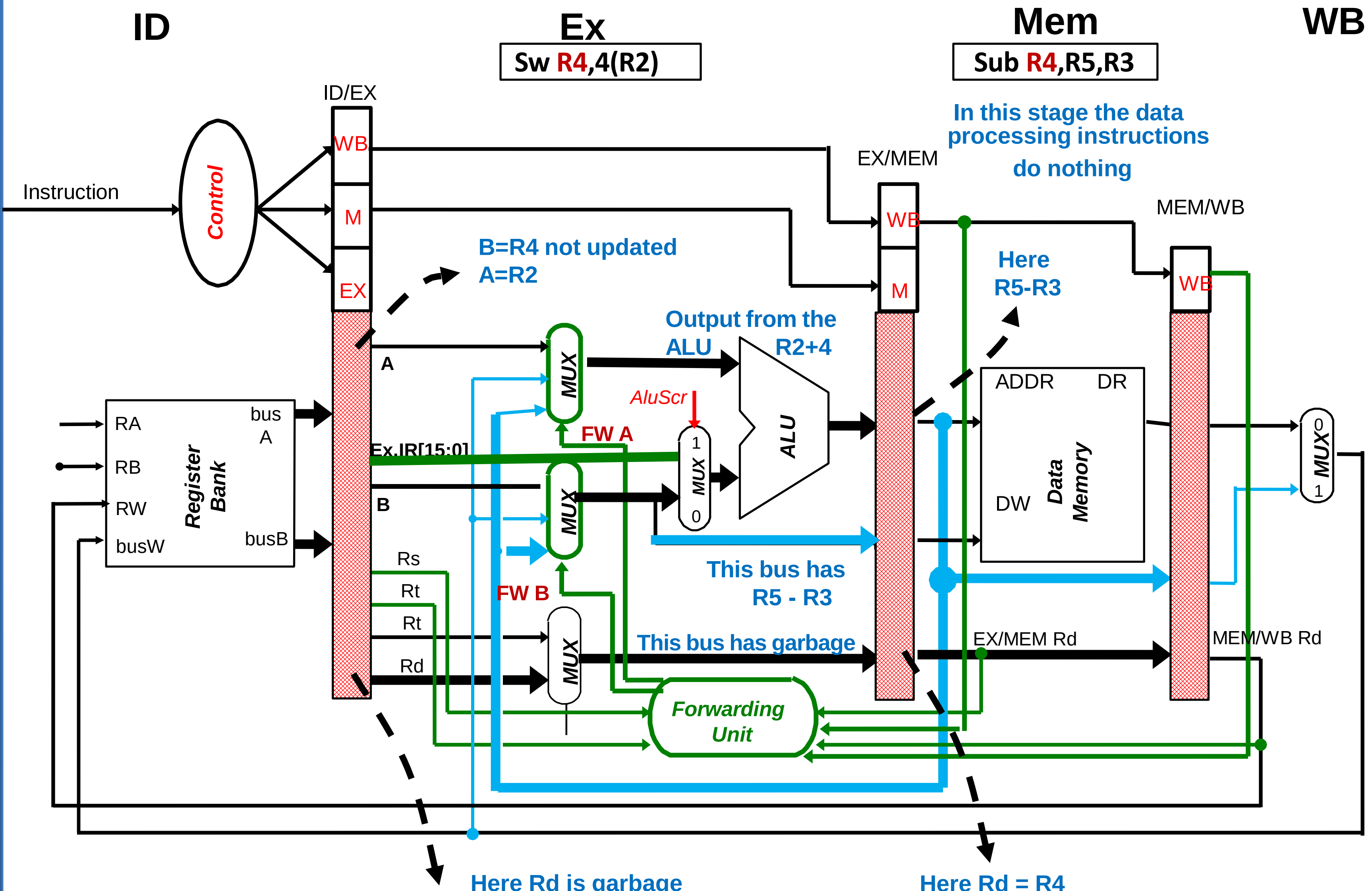


	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
Clock								
Sub R4, R5,R3	IF	ID	EX	Mem	WB			
Sw R4, 4(R2)		IF	ID	EX	Mem	WB		
…								

There are no wait cycles

5. Pipelining : Data Hazards

Execution of RAW with a store instruction: Cycle 4



In this stage the data processing instructions do nothing

B=R4 not updated
A=R2

Output from the ALU R2+4

Here R5-R3

This bus has R5 - R3

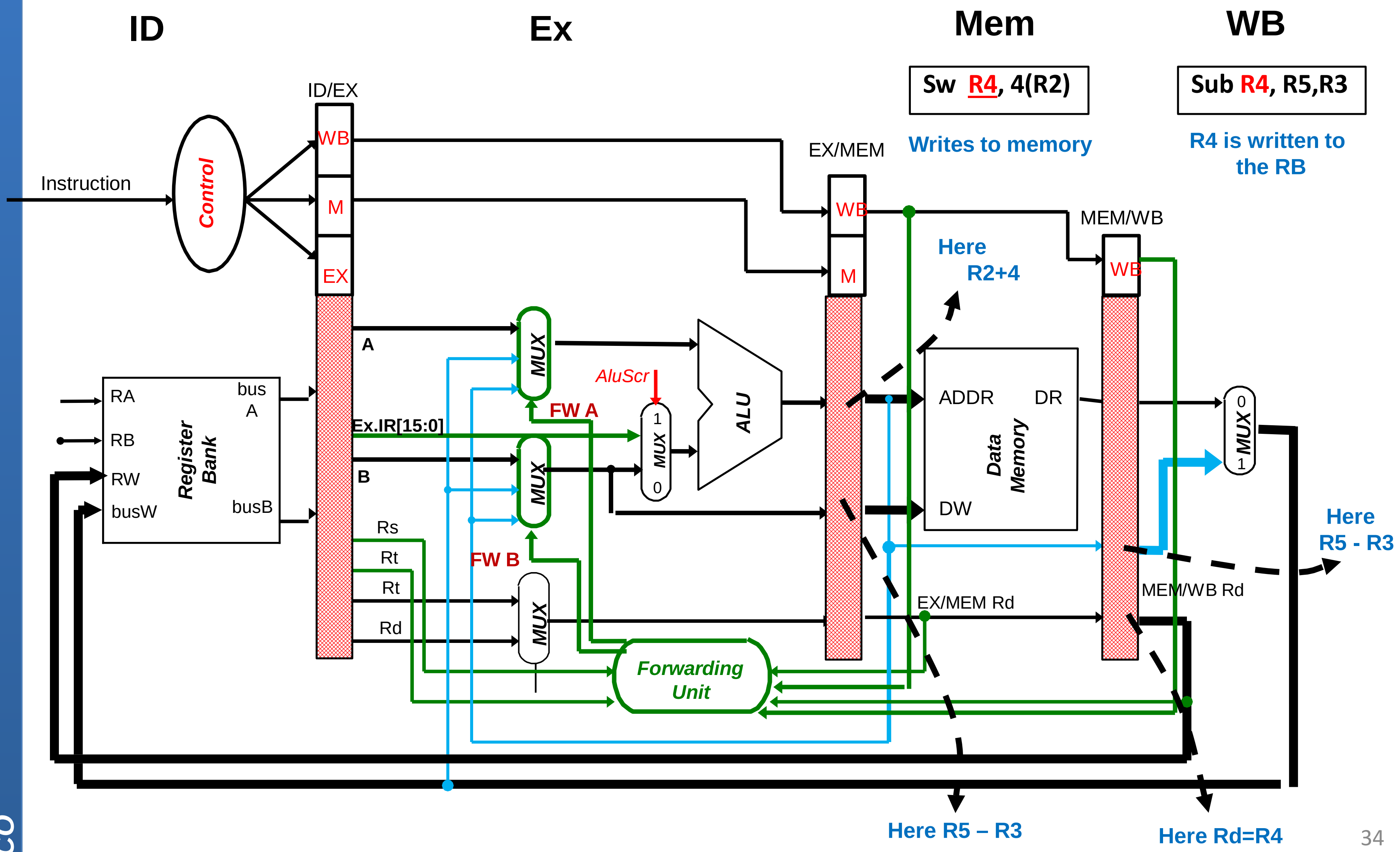
This bus has garbage

Here Rd is garbage

Here Rd = R4

5. Pipelining : Data Hazards

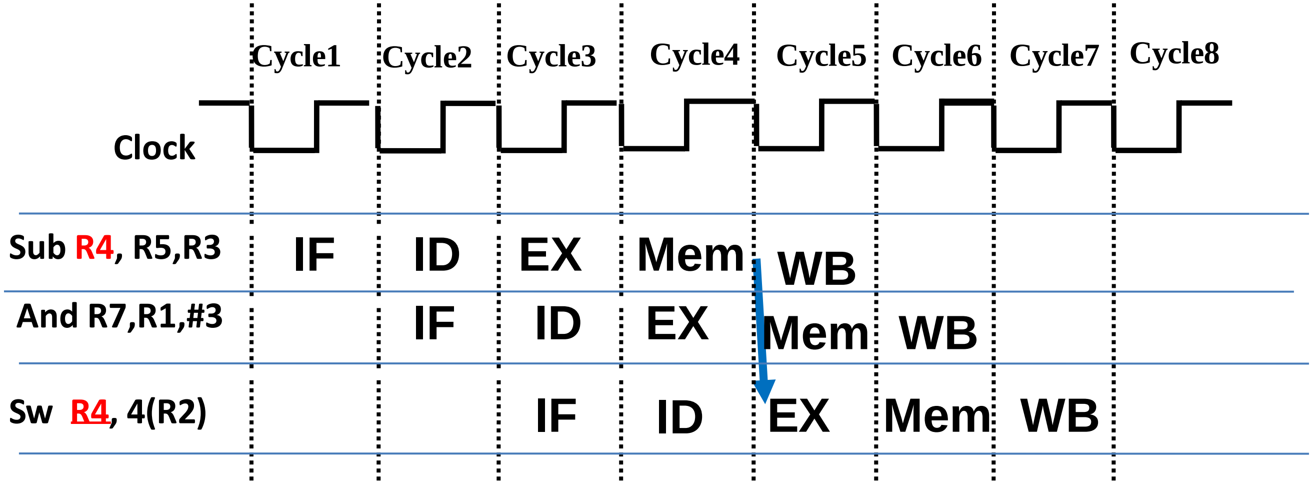
Execution of RAW with a store instruction: Cycle 5



5. Pipelining: Data Hazards

RAW Hazard with store instructions

- The store instruction writes to memory in the Mem stage, but the data written is read from the RB in the ID stage
  - **Can make use of Forwarding, replacing in ex the value read in ID**



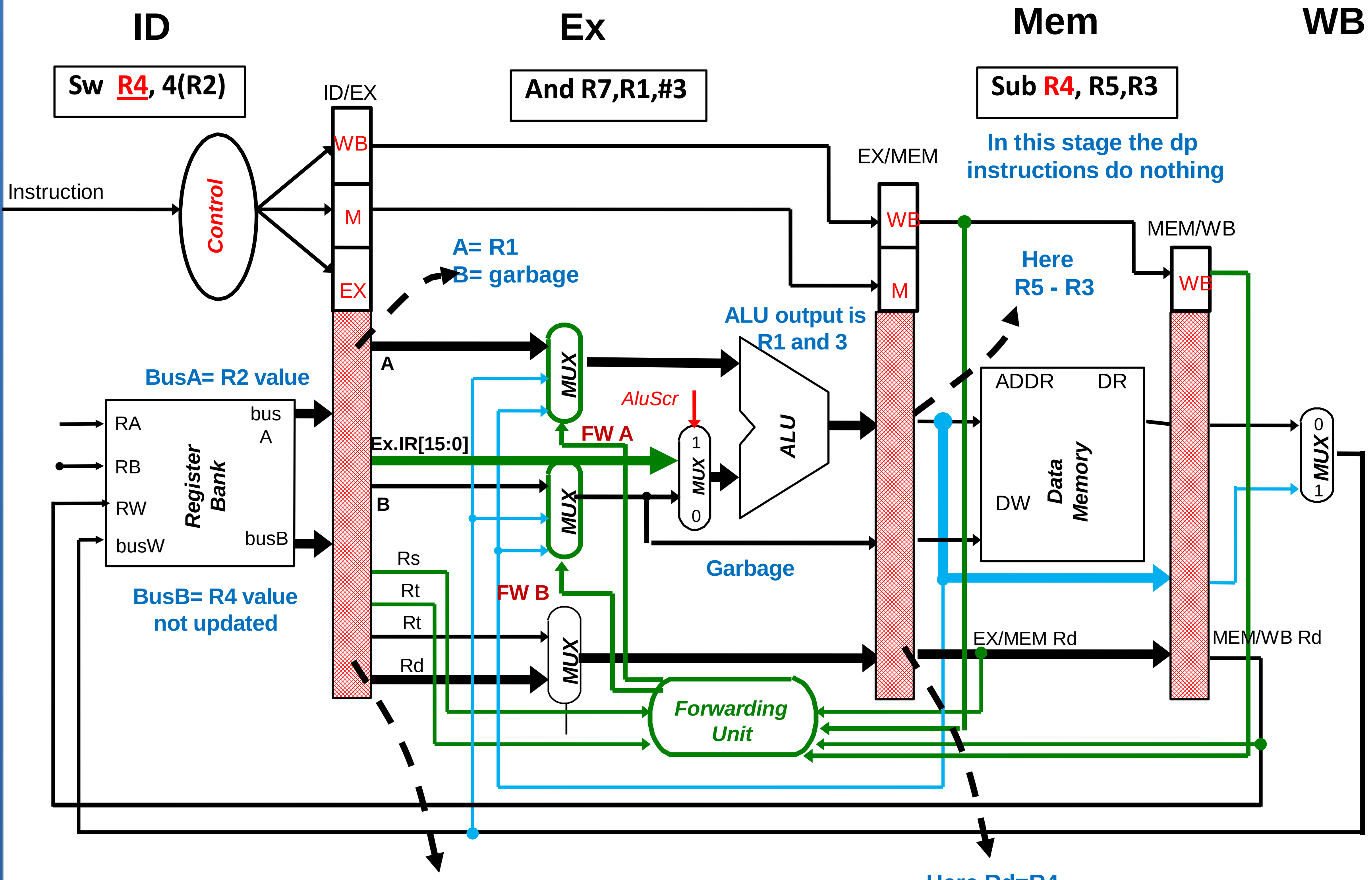
	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
Sub R4, R5,R3	IF	ID	EX	Mem	WB			
And R7,R1,#3		IF	ID	EX	Mem	WB		
Sw R4, 4(R2)			IF	ID	EX	Mem	WB	

…

No wait cycles

5. Pipelining : Data Hazards

Execution of example: Cycle 4



ID

Ex

Mem

WB

Sw R4, 4(R2)

And R7,R1,#3

Sub R4, R5,R3

In this stage the dp instructions do nothing

A= R1
B= garbage

Here
R5 - R3

ALU output is R1 and 3

BusA= R2 value

BusB= R4 value
not updated

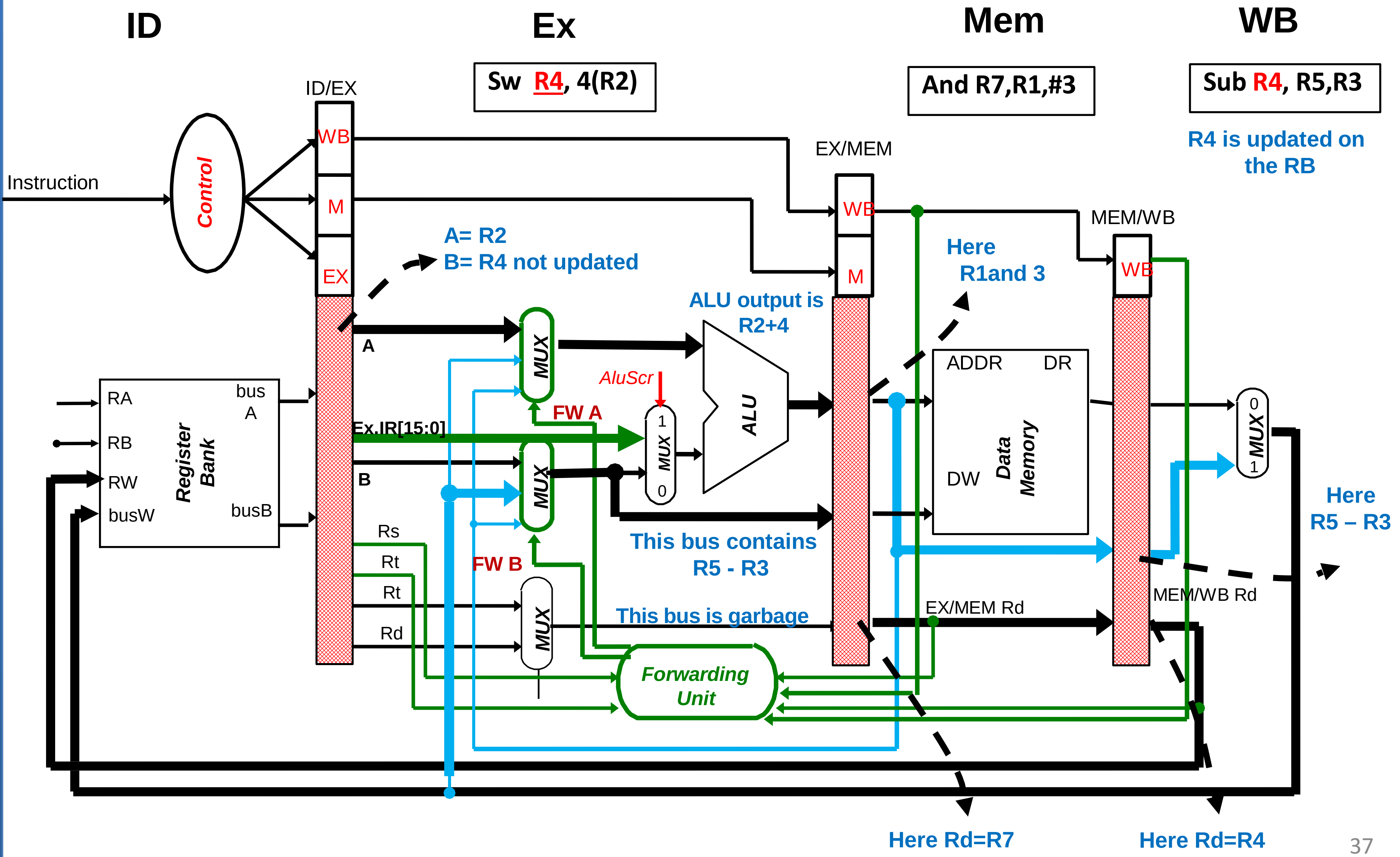
Garbage

Here Rd=R7

Here Rd=R4

5. Pipelining : Data Hazards

Execution of example: Cycle 5



R4 is updated on the RB

A= R2
B= R4 not updated

ALU output is R2+4

Here R1and 3

This bus contains R5 - R3

This bus is garbage

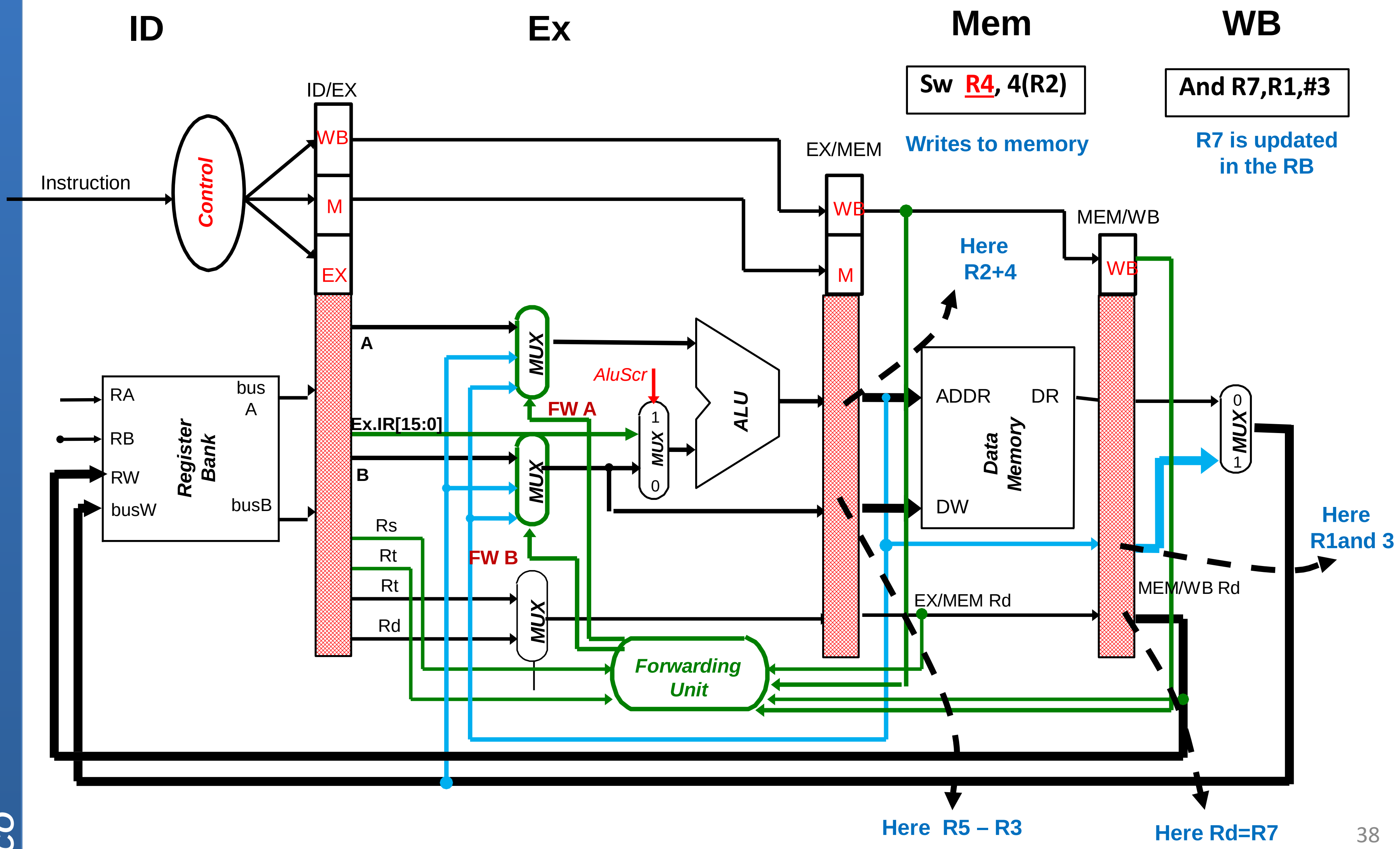
Here R5 – R3

Here Rd=R7

Here Rd=R4

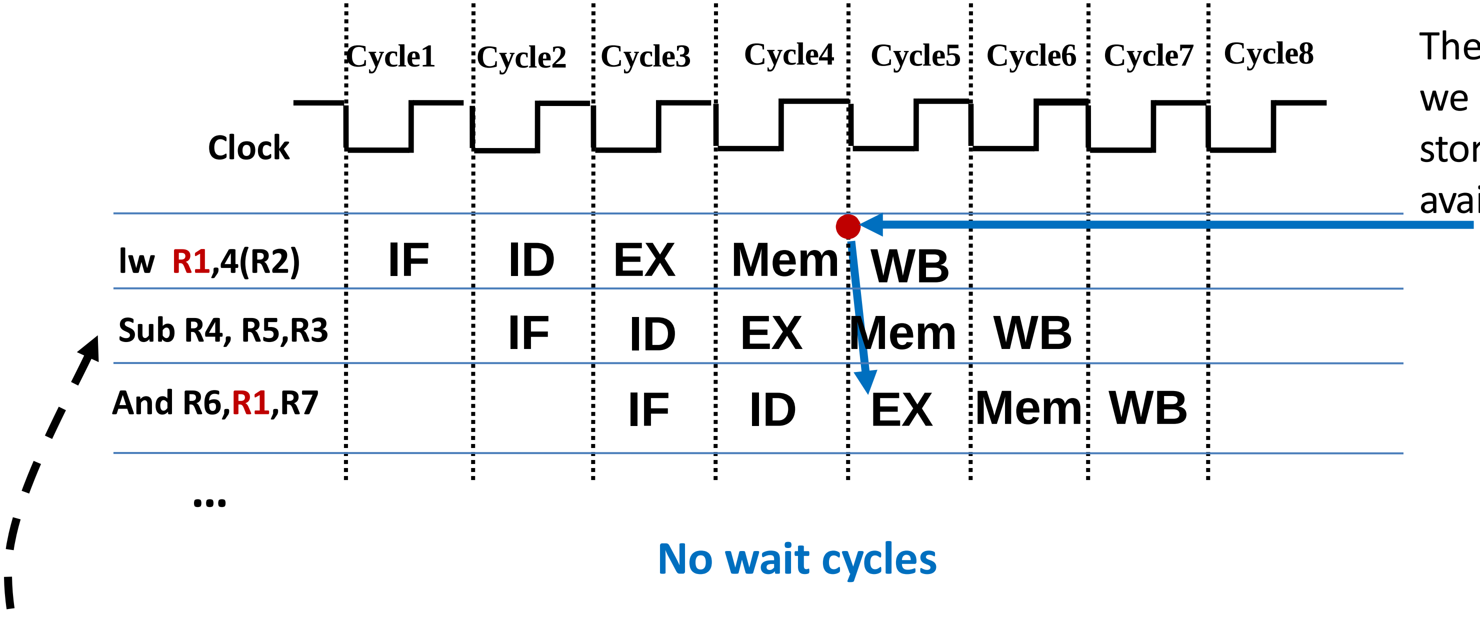
5. Pipelining : Data Hazards

Execution of the example: Cycle 6



5. Pipelining: Data Hazards

RAW Hazard: Data provided by a previous lw instruction



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
lw R1,4(R2)	IF	ID	EX	Mem	WB			
Sub R4, R5,R3		IF	ID	EX	Mem	WB		
And R6,R1,R7			IF	ID	EX	Mem	WB	
...								

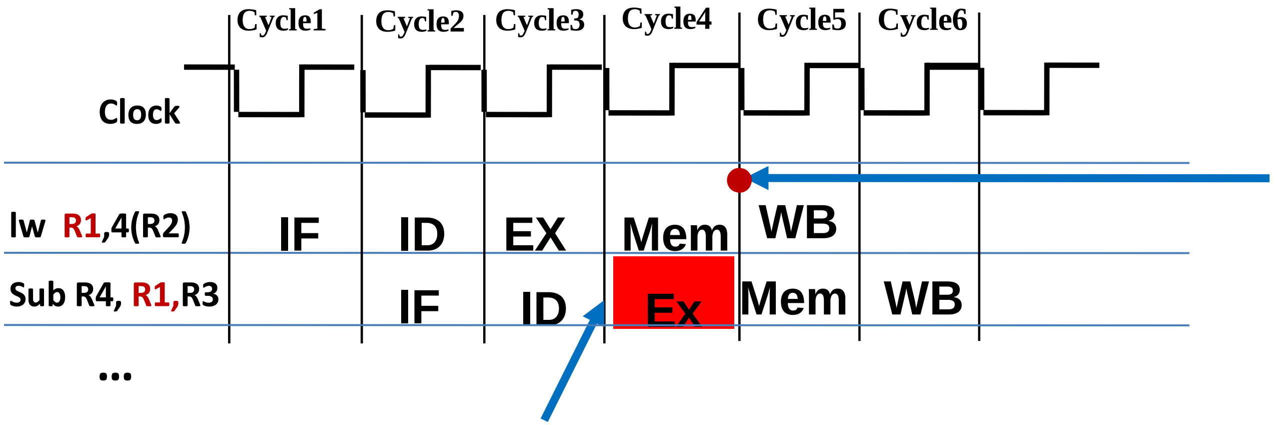
The value that we want to store in R1 is available

No wait cycles

– Can be solved by forwarding if there is one instruction in between

5. Pipelining: Data Hazards

RAW Hazard: Data provided by a previous load instruction



The value that we want to store in R1 is available

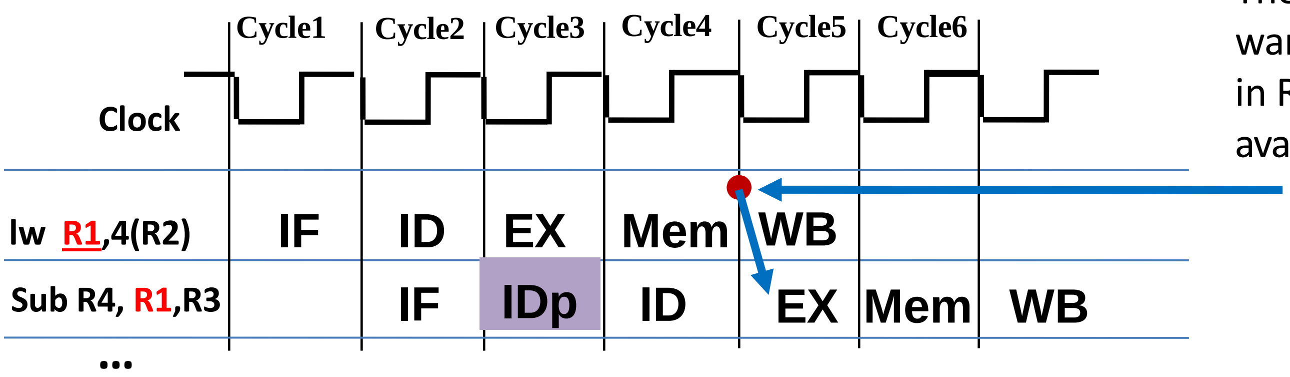
It needs the updated value of R1, but it is not available, as it comes from memory on the next cycle

– **Cannot be solved by forwarding, a wait cycle is required**

5. Pipelining: Data Hazards

RAW hazards: Data provided by a previous load instruction

- **HW Solution:** Hazard detection and pipeline stall for one cycle



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	
lw R1,4(R2)	IF	ID	EX	Mem	WB		
Sub R4, R1,R3		IF	IDp	ID	EX	Mem	WB
...							

The value we want to store in R1 is available

1 wait cycle

IDp	Wait cycle because of a RAW hazard between instruction 1 and instruction 2

5. Pipelining : Data Hazards

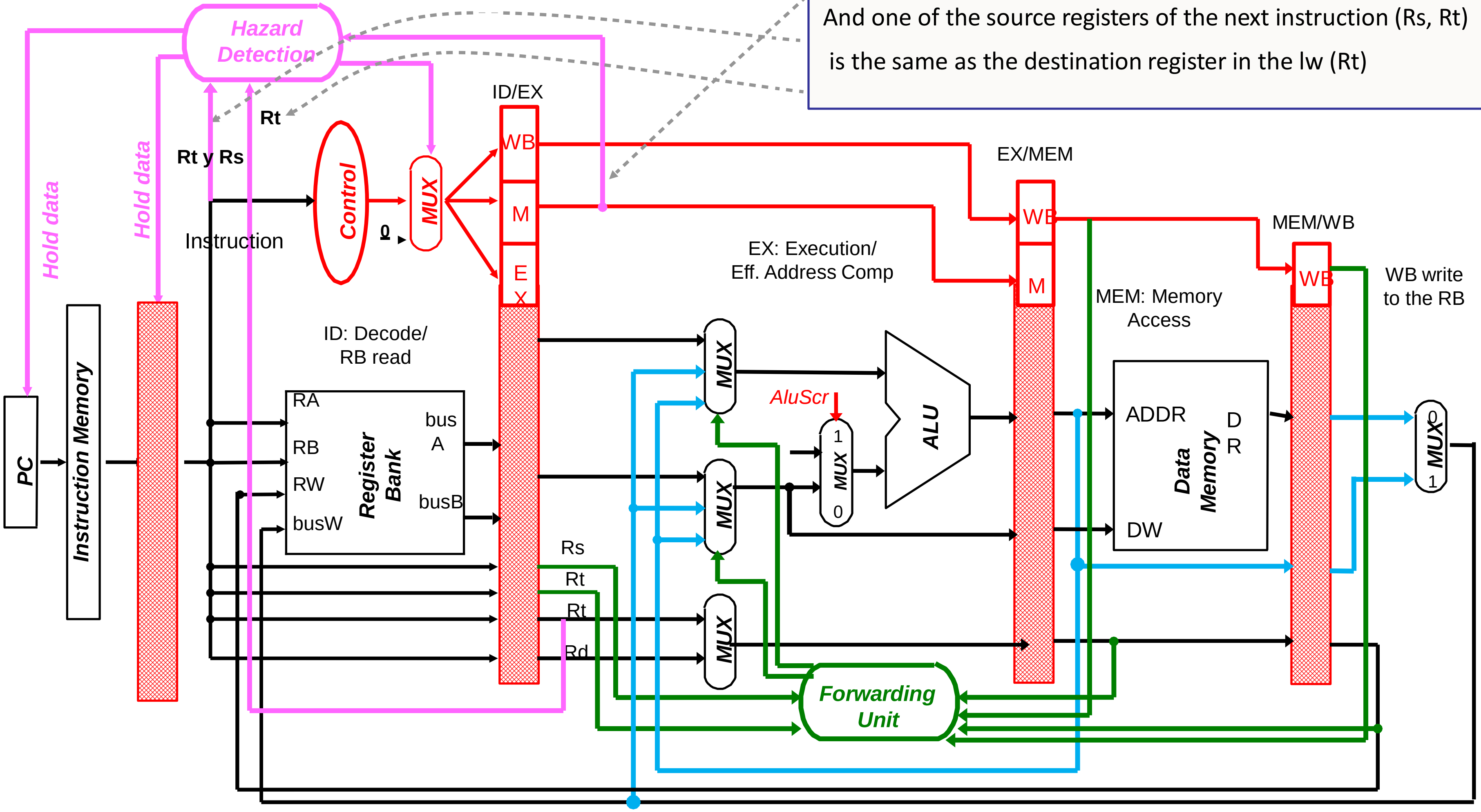
Hardware Solution implementation:

- Hazard detection and pipeline stall

If the instruction in EX is a lw (MemRead=1)

And one of the source registers of the next instruction (Rs, Rt)

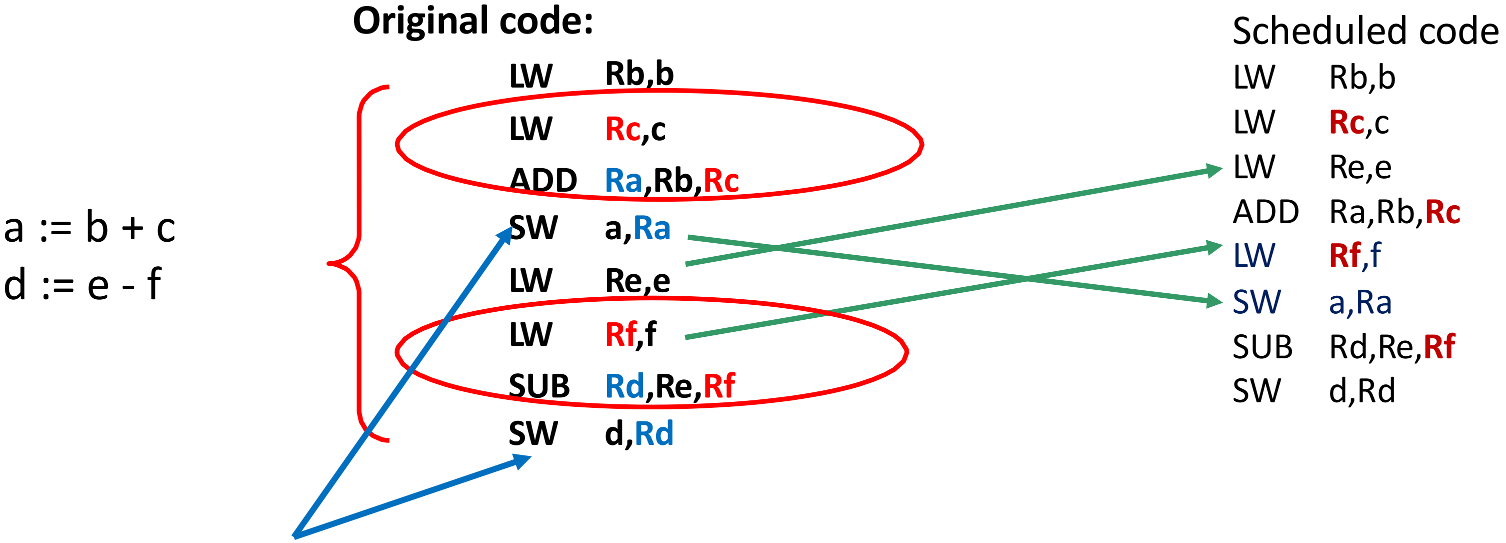
is the same as the destination register in the lw (Rt)



5. Pipelining: Data Hazards

RAW hazards: Data provided by a previous load

- **SW solution**: Instruction scheduling stage in the compiler tries to separate the load from the instruction that consumes the data



a := b + c
d := e - f

Original code:

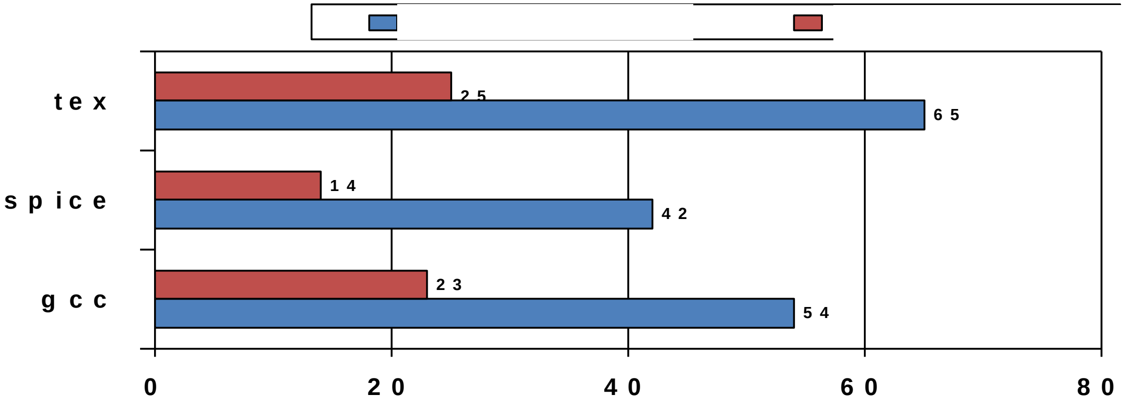
```
LW    Rb,b
LW    Rc,c
ADD   Ra,Rb,Rc
SW    a,Ra
LW    Re,e
LW    Rf,f
SUB   Rd,Re,Rf
SW    d,Rd
```

Scheduled code

```
LW    Rb,b
LW    Rc,c
LW    Re,e
ADD   Ra,Rb,Rc
LW    Rf,f
SW    a,Ra
SUB   Rd,Re,Rf
SW    d,Rd
```

RAW in SW is solved by forwarding

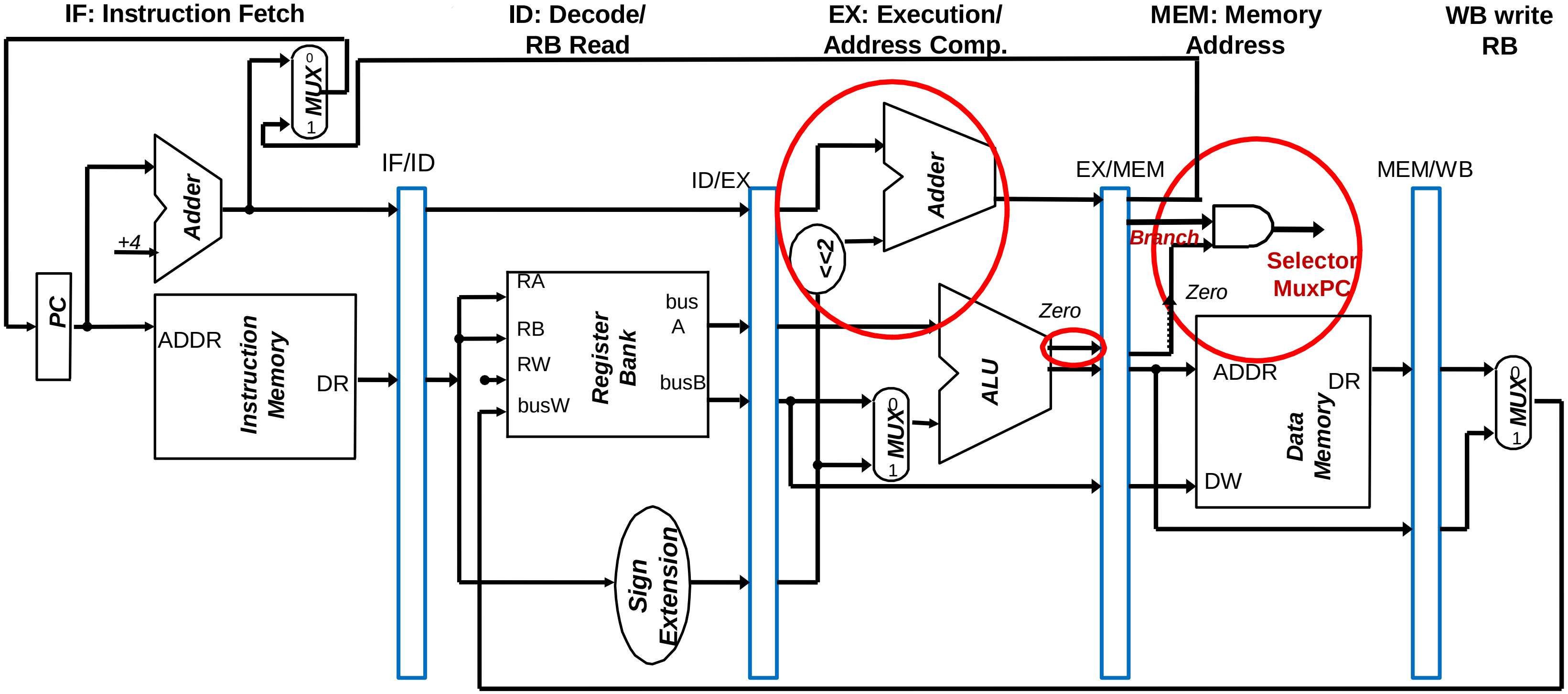
% of lw that introduce a delay cycle (pipeline stall)



6. Pipelining : Control Hazards

Control Hazards: **Why do they appear?**

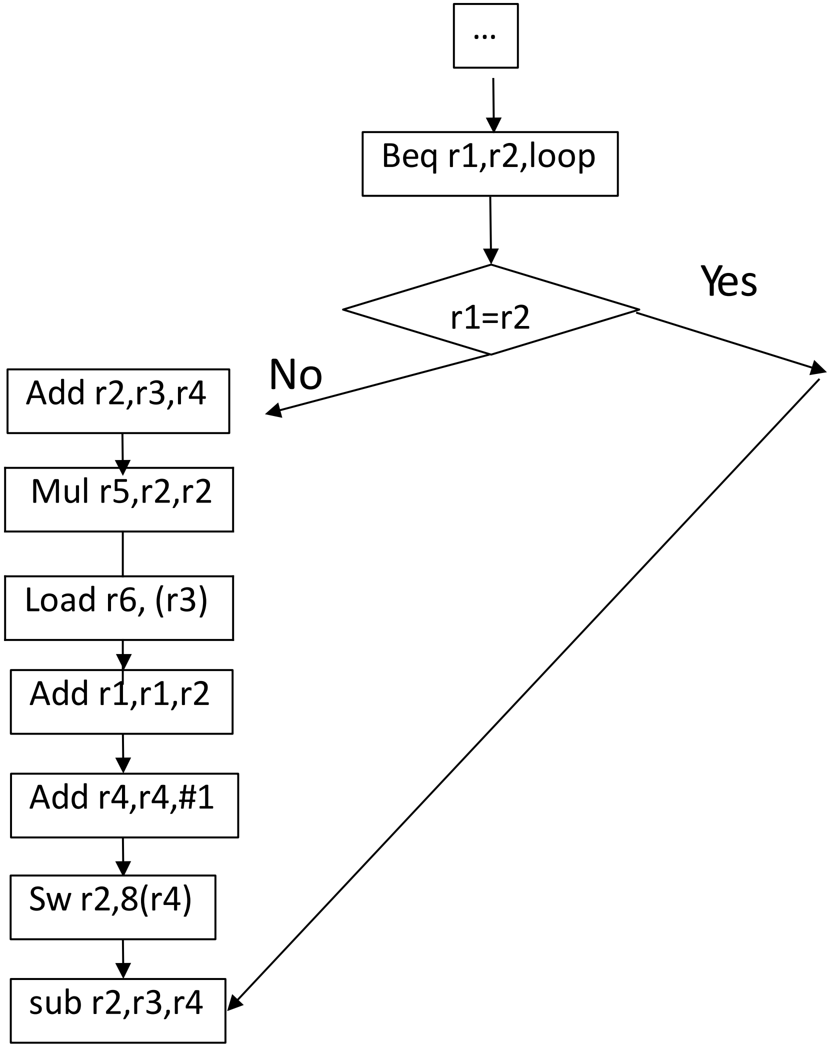
- A conditional branch needs to:
  - Compute the **destination address**
  - Compute **the condition** for the branch
- **The hazard appears because:**
  - In the data path studied so far **both computations take place in the Ex stage**
  - **If the branch is taken, the destination address is stored on PC at the end of the Mem stage,** on the rising edge of the clock



6. Pipelining : Control Hazards

Example:

```
…
          Beq r1,r2,loop
          Add r2,r3,r4
          Mul r5,r2,r2
          load r6, (r3)
          Add r1,r1,r2
          Add r4,r4,#1
          Sw   r2,8(r4)
Loop      sub r2,r3,r4
```

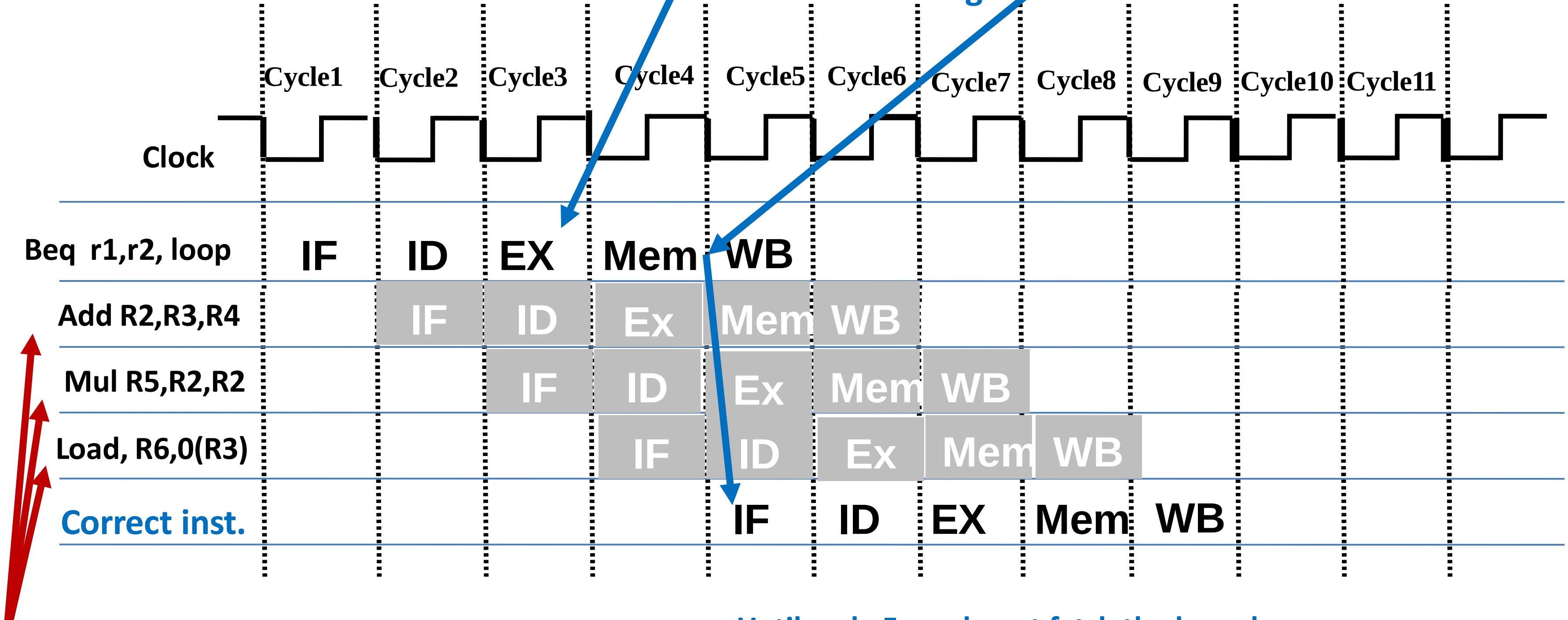


6. Pipelining: Control Hazards

Example:

The computation of the **destination address** and the **condition** evaluation are performed on the **EX stage**

The new address is stored on PC on the falling edge of the clock **that starts the WB stage**



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9	Cycle10	Cycle11
Beq r1,r2, loop	IF	ID	EX	Mem	WB						
Add R2,R3,R4		IF	ID	Ex	Mem	WB					
Mul R5,R2,R2			IF	ID	Ex	Mem	WB				
Load, R6,0(R3)				IF	ID	Ex	Mem	WB			
Correct inst.					IF	ID	EX	Mem	WB		

Still don't know if the branch will be taken, the next instructions enter the pipeline

Until cycle 5 we do not fetch the branch target instruction

3 instructions entered the pipeline and should have not

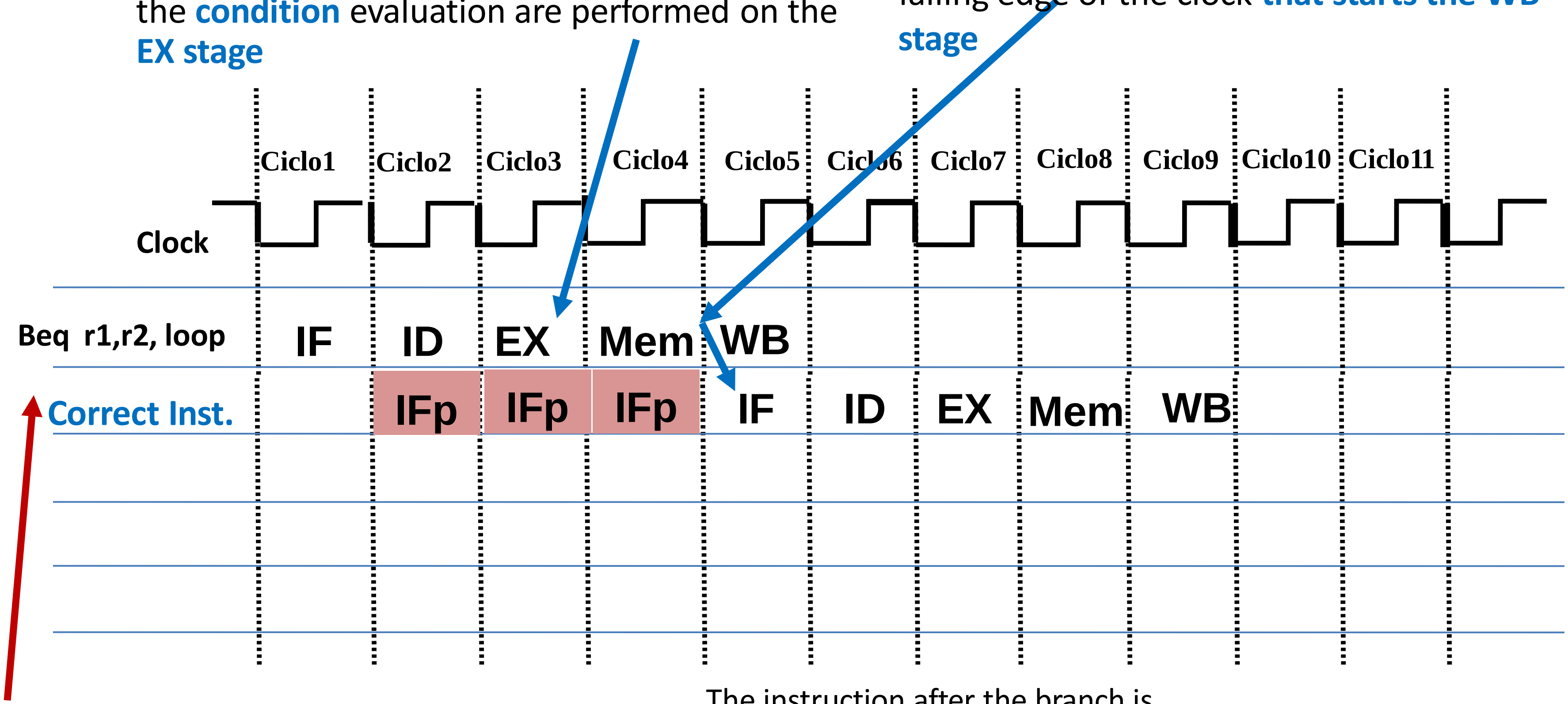
6. Pipelining : Control Hazards

How can we solve the control hazards?

- **Solution 1**: **Wait for three cycles** to know which is the next instruction after the branch
  - **Pipeline stalling**

The computation of the **destination address** and the **condition** evaluation are performed on the **EX stage**

The new address is stored on PC on the falling edge of the clock **that starts the WB stage**



	Ciclo1	Ciclo2	Ciclo3	Ciclo4	Ciclo5	Ciclo6	Ciclo7	Ciclo8	Ciclo9	Ciclo10	Ciclo11
Beq r1,r2, loop	IF	ID	EX	Mem	WB						
Correct Inst.		IFp	IFp	IFp	IF	ID	EX	Mem	WB		

The pipeline is stalled until we know which is the next instruction

The instruction after the branch is fetched **in cycle 5**

3 penalty cycles

6. Pipelining : control hazards

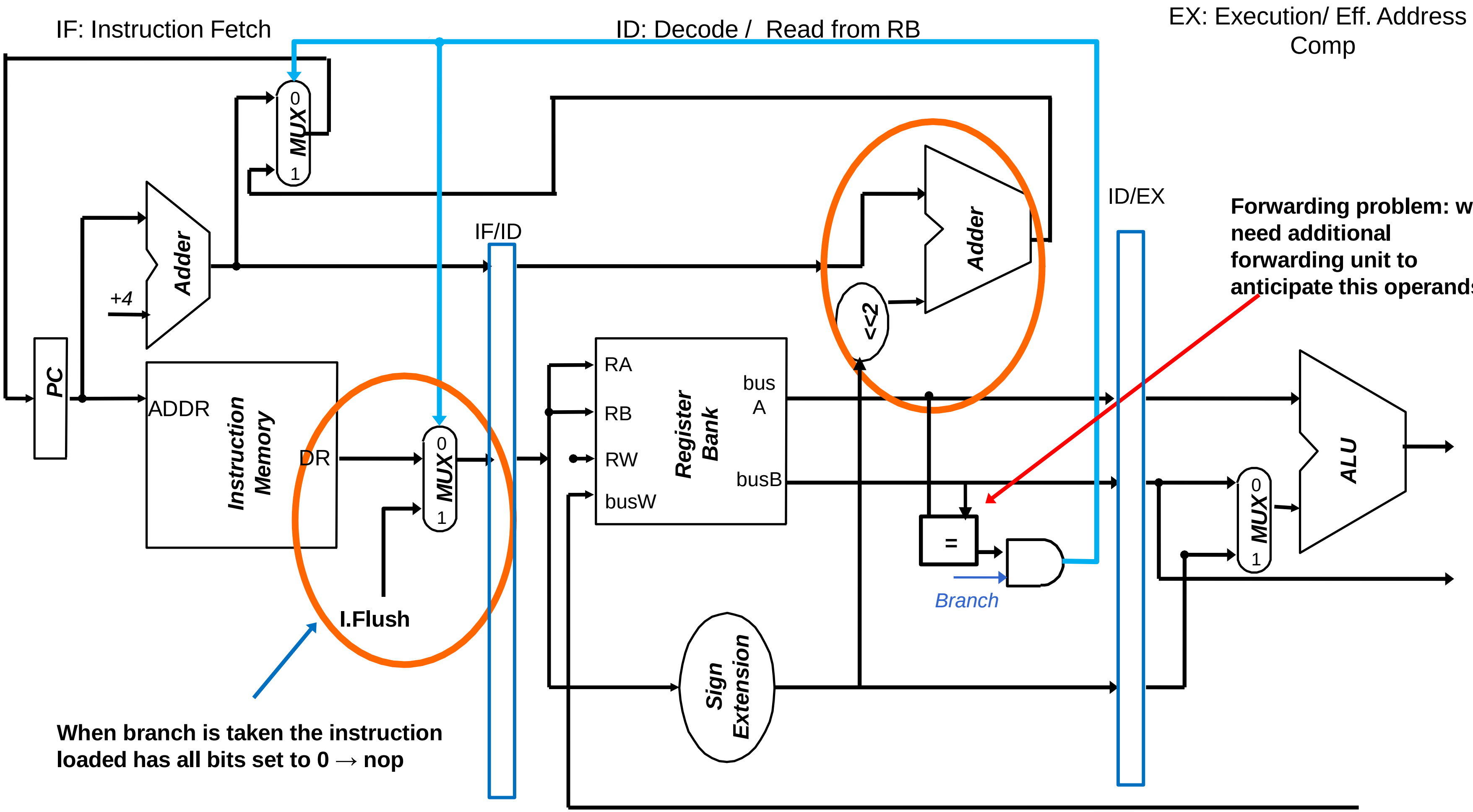
How can we solve the control hazards?

- **Solution 2**:
  - Move the **address computation** and **condition** evaluation at the **ID stage**   Only **one wait cycle** is needed to know the address of the instruction after the branch. This wait cycle can be implemented by:
    - **HW: Transforming the instruction fetched (the one following the branch) into a nop**
      - **If the branch is taken:** the instruction fetched after the branch is converted to a nop
        - **All control lines set to "0", no operation performed**
        - ***1 penalization cycle***
      - **If the branch is not taken:** the fetched instruction after the branch enters the pipeline as usual
        - ***No wait cycles***
    - **SW: Delay slot**
      - The instruction after the branch is always executed
      - The compiler places after the branch an instruction that is independent of the branch, which will be executed in the "delay slot"
      - *Free of penalty cycles if the compiler can find instructions that do not depend on the branch*

6. Pipelining : Control Hazards

Data path that implements the HW solution for Control Hazards:

- Computes the **branch target address** and evaluates the **condition** in the **ID stage**
- Transforms the fetched instruction into a nop (flush) if branch is taken



Forwarding problem: we need additional forwarding unit to anticipate this operands

When branch is taken the instruction loaded has all bits set to 0 → nop

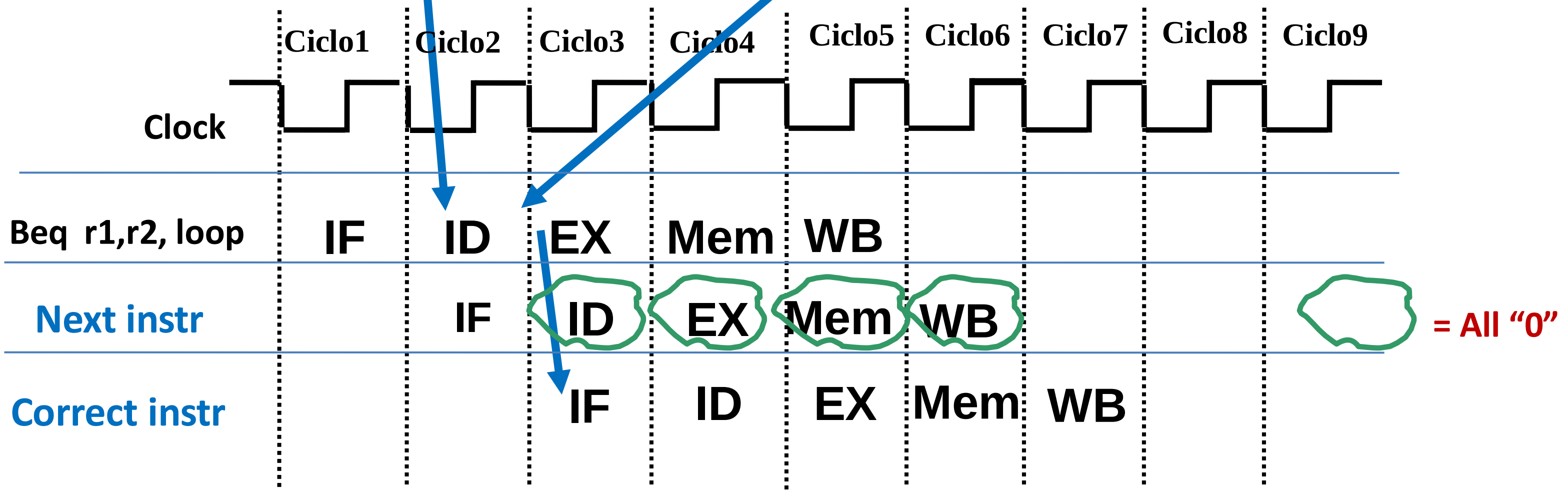
6. Pipelining : Control Hazards

Example: Applying solution 2 with HW implementation

If the branch is taken

Destination address computation and **condition** evaluation are performed on the **ID stage**

The new adrress is stored in PC on the falling edge of the clock that starts the **Ex stage**



	Ciclo1	Ciclo2	Ciclo3	Ciclo4	Ciclo5	Ciclo6	Ciclo7	Ciclo8	Ciclo9
Clock									
Beq r1,r2, loop	IF	ID	EX	Mem	WB				
Next instr		IF	ID	EX	Mem	WB			
Correct instr			IF	ID	EX	Mem	WB		

= All "0"

The instruction following the branch is NOT the correct instruction (set to "0")

At the start cycle 3 the correct instruction enters the pipeline

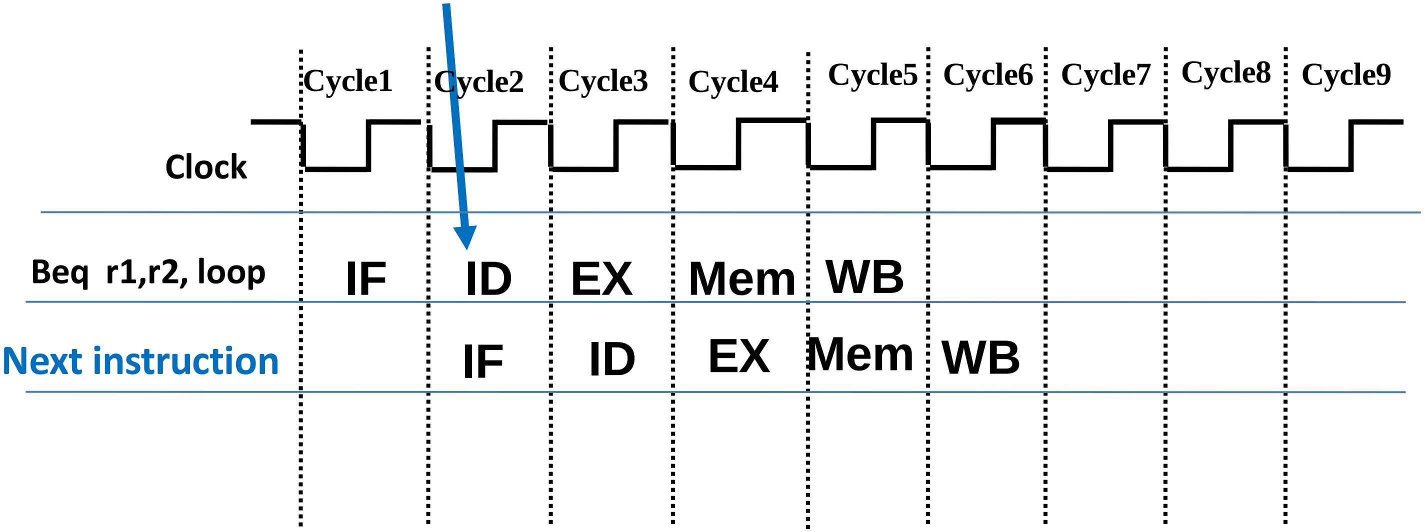
1 penalty cycle

6. Pipelining : Control Hazards

Example: Applying Solution 2 with HW implementation

If the branch is not taken

The **branch target address** and the **condition** evaluation are performed in **ID stage**



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8	Cycle9
Beq r1,r2, loop	IF	ID	EX	Mem	WB				
Next instruction		IF	ID	EX	Mem	WB			

The instruction following the branch is the correct one

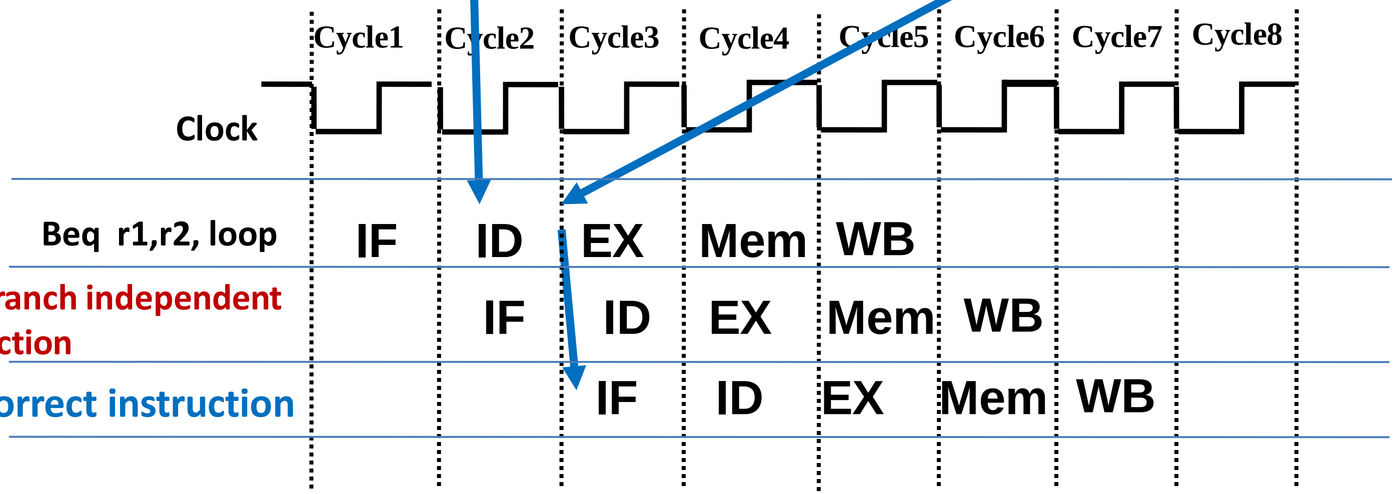
No penalty cycles

6. Pipelining : Control Hazards

Example: Applying Solution 2 with SW implementation: Delay slot

The **branch target address** computation and **condition** evaluation are performed in **ID stage**

The new address is stored in PC on the falling edge of the clock **that starts Ex stage**



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
Clock								
Beq r1,r2, loop	IF	ID	EX	Mem	WB			
Any branch independent instruction		IF	ID	EX	Mem	WB		
Correct instruction			IF	ID	EX	Mem	WB	

No penalty cycles

The ideal is to choose an instruction that must be always executed

If not possible, try to find an instruction that does not affect the program if it is executed and should have not

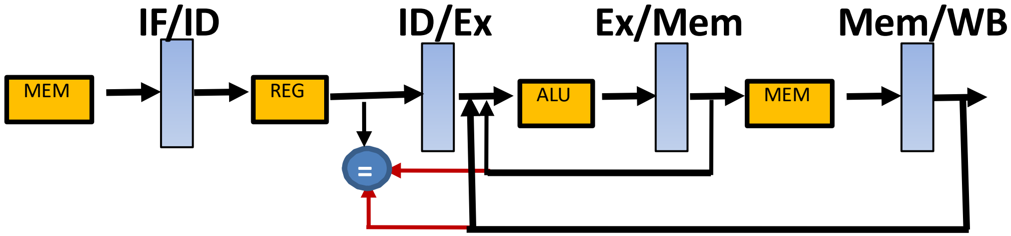
7. Control Hazards + RAW Hazards

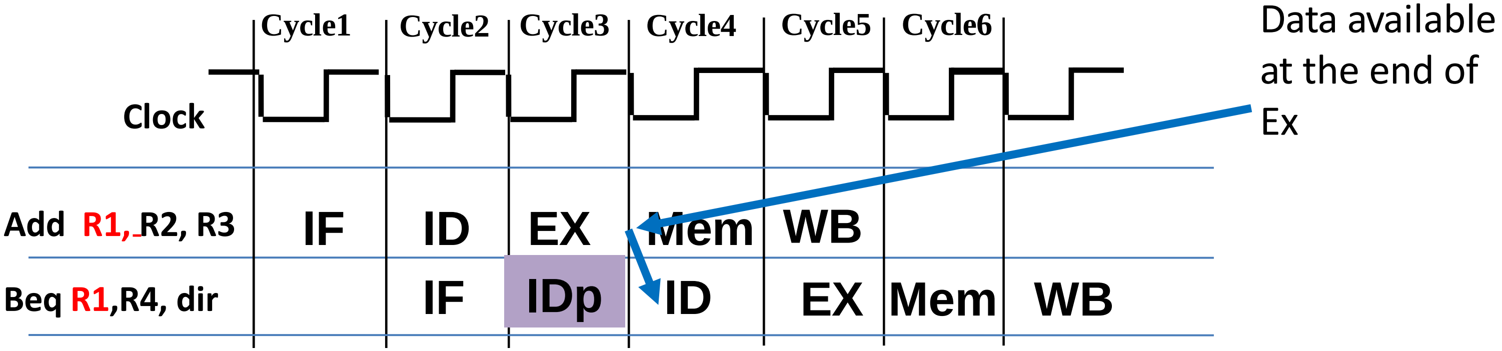
When the branch instruction requires a data generated by the previous instruction

– **Problem**:

- The branch evaluates the condition in the ID stage
- The forwarding, as is implemented by now, does not provide the data

– **Solution**: **Add forwarding on ID stage**





	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	
Add R1, R2, R3	IF	ID	EX	Mem	WB		
Beq R1,R4, dir		IF	IDp	ID	EX	Mem	WB

Data available at the end of Ex

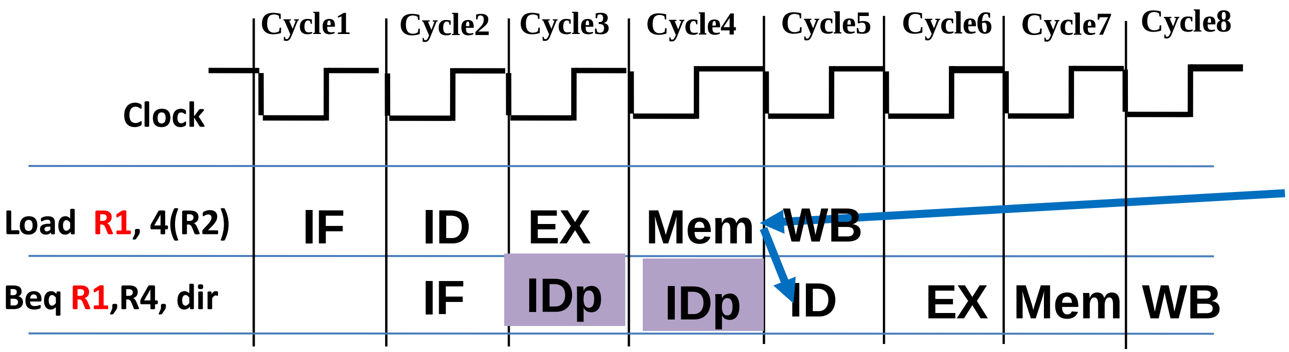
1 wait cycle

IDp Stall because of RAW hazard between instruction 1 and instruction 2

7. Control Hazard + RAW Hazard

The branch instruction needs a data provided by the previous instruction

– If the instruction that generates the data is a LOAD



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	Cycle7	Cycle8
Load R1, 4(R2)	IF	ID	EX	Mem	WB			
Beq R1,R4, dir		IF	IDp	IDp	ID	EX	Mem	WB

Data available at the end of Mem stage

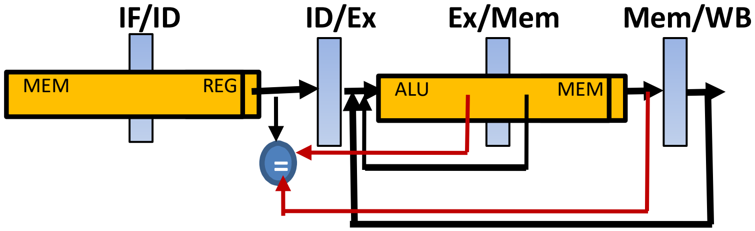
2 wait cycles

IDp **Wait because of RAW hazard** between instruction 1 and instruction 2

7. Control Hazard + RAW Hazard

The branch instruction requires a data provided by the previous instruction

- If forwarding is combinational
- The data is not forwarded from the pipelining registers but from the ALU output or the Data Memory output



	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6
Clock						
Add R1, R2, R3	IF	ID	EX	Mem	WB	
Beq R1,R4, dir		IF	ID	EX	Mem	WB

No wait cycles

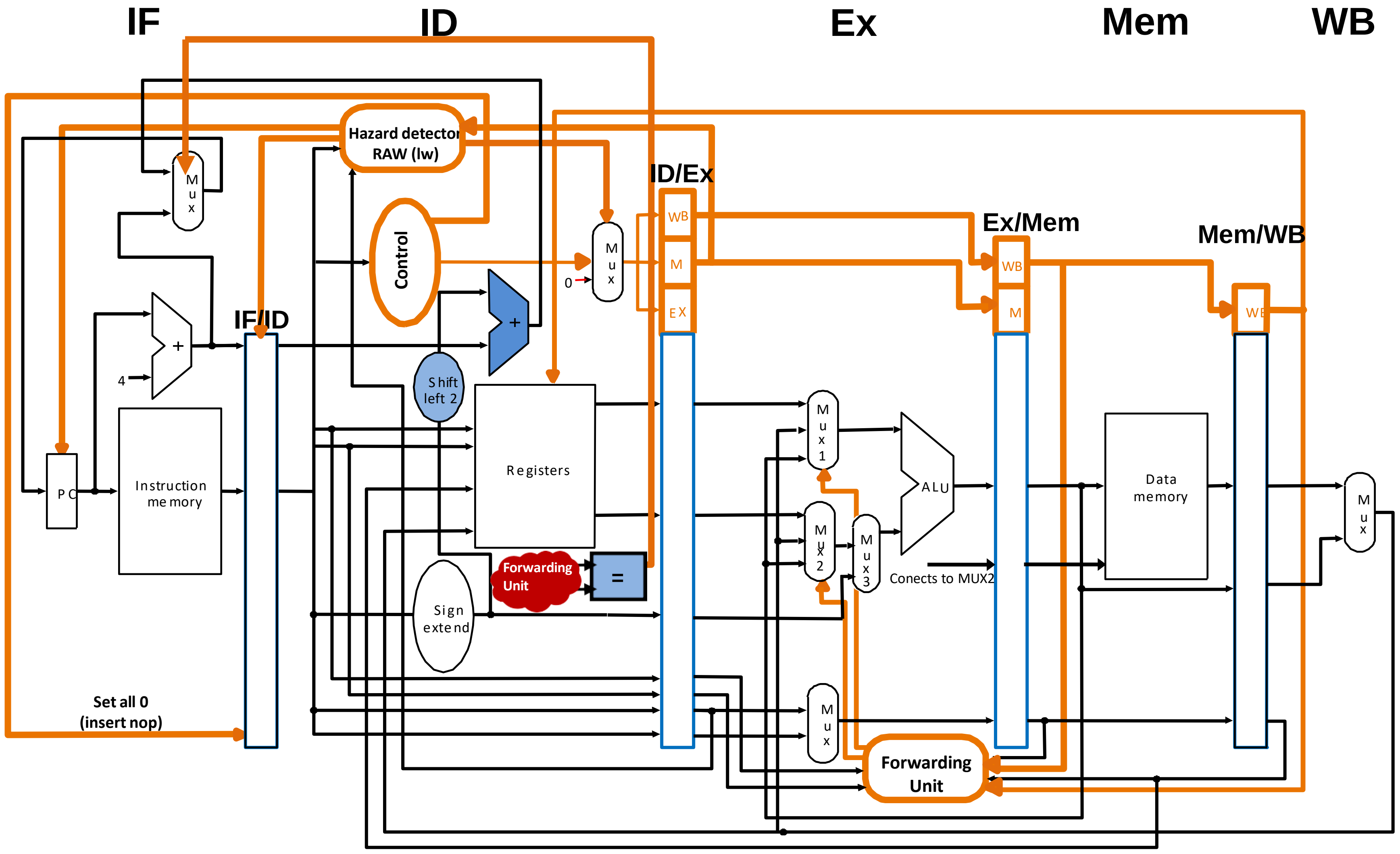
	Cycle1	Cycle2	Cycle3	Cycle4	Cycle5	Cycle6	
Clock							
Load R1, 4(R2)	IF	ID	EX	Mem	WB		
Beq R1,R4, dir		IF	IDp	ID	EX	Mem	WB

1 wait cycle

8. Summary

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**



8. Summary

Pipelined processor

All instructions need the same amount of cycles

Ideal performance, one instruction per cycle CPI=1

Structural Hazards and the WAW and WAR solved by construction

RAW hazards on type-R instructions are solved by forwarding

RAW hazards with lw require one wait cycle (pipeline stall)

Compiler can help with smart instruction scheduling

Control Hazards can be solved:

- HW: If branch is taken the hw inserts one NOP (equivalent to one penalty cycle)
- SW: Delay slot, depends on compiler scheduling

Instructions start and finish in order