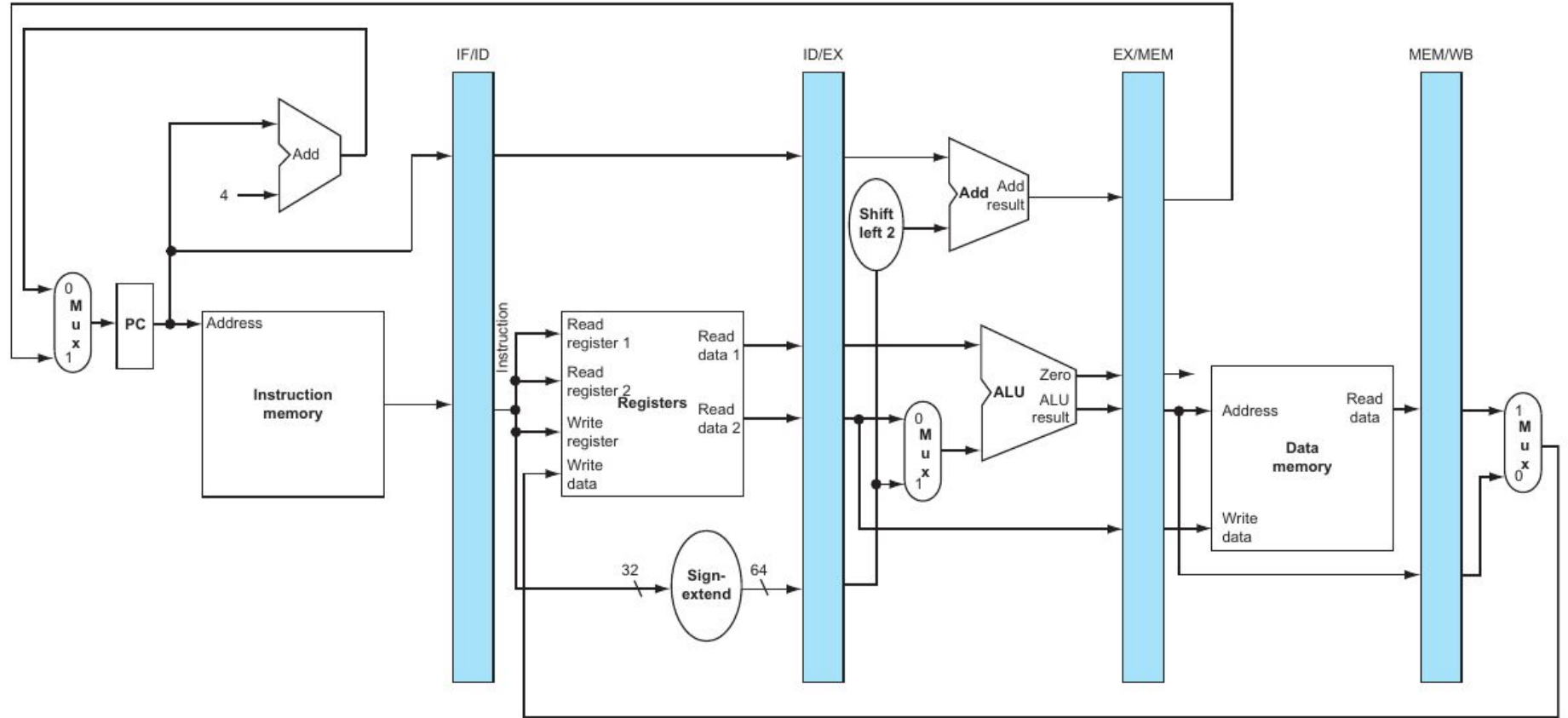


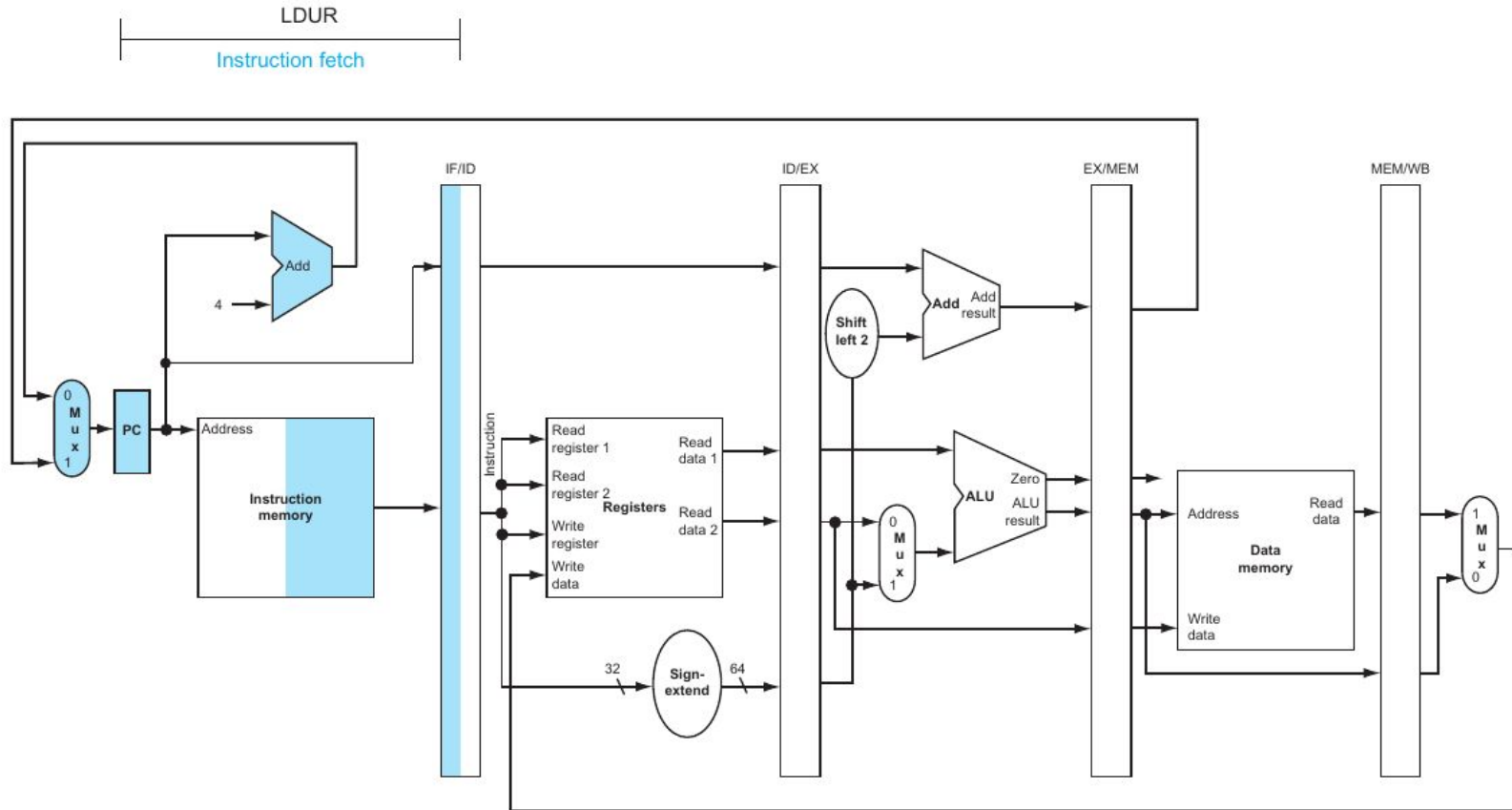
# Práctico N° 4: Procesador con pipeline

Arquitectura de Computadoras 2022

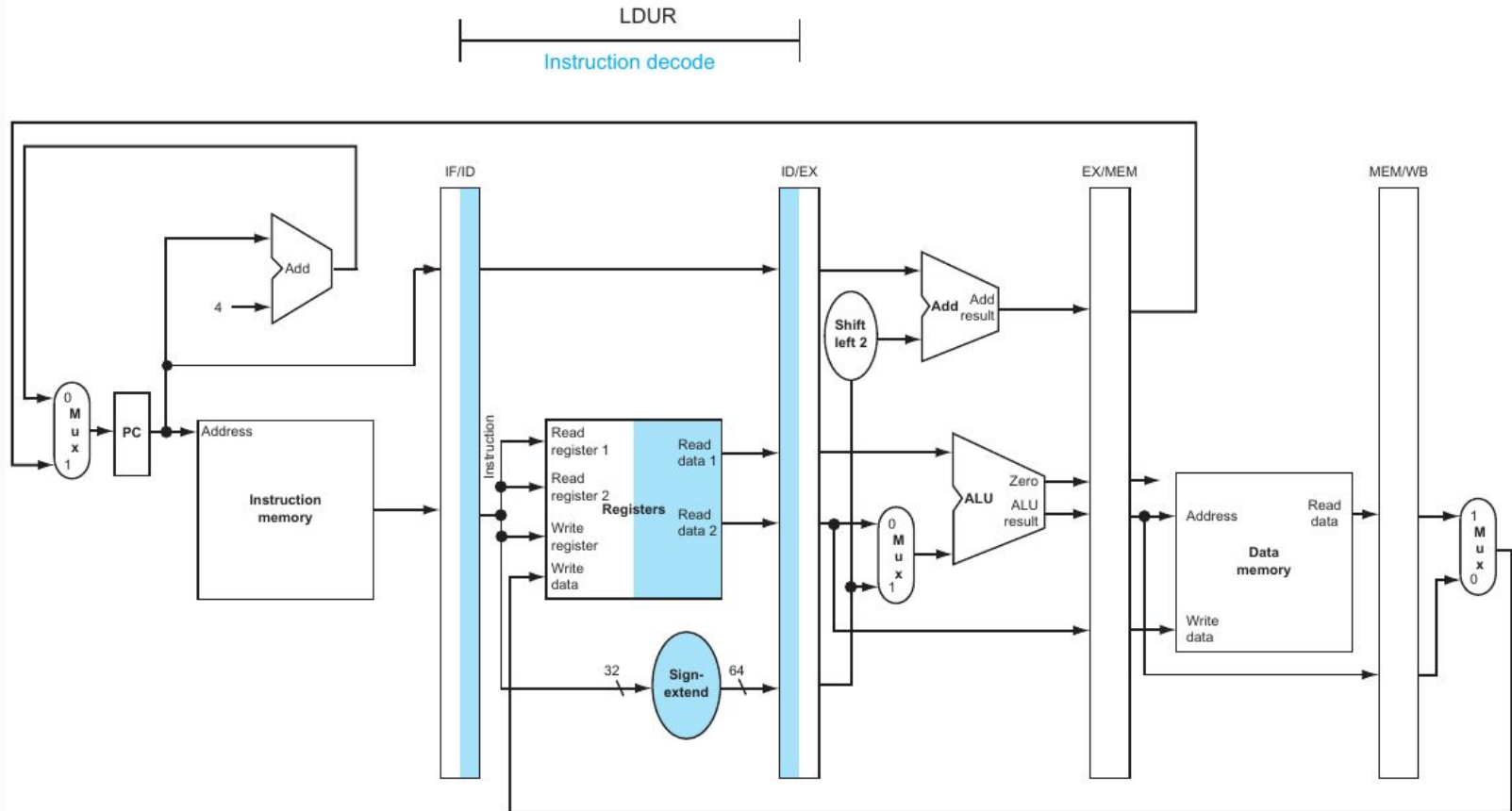
# Pipelined datapath



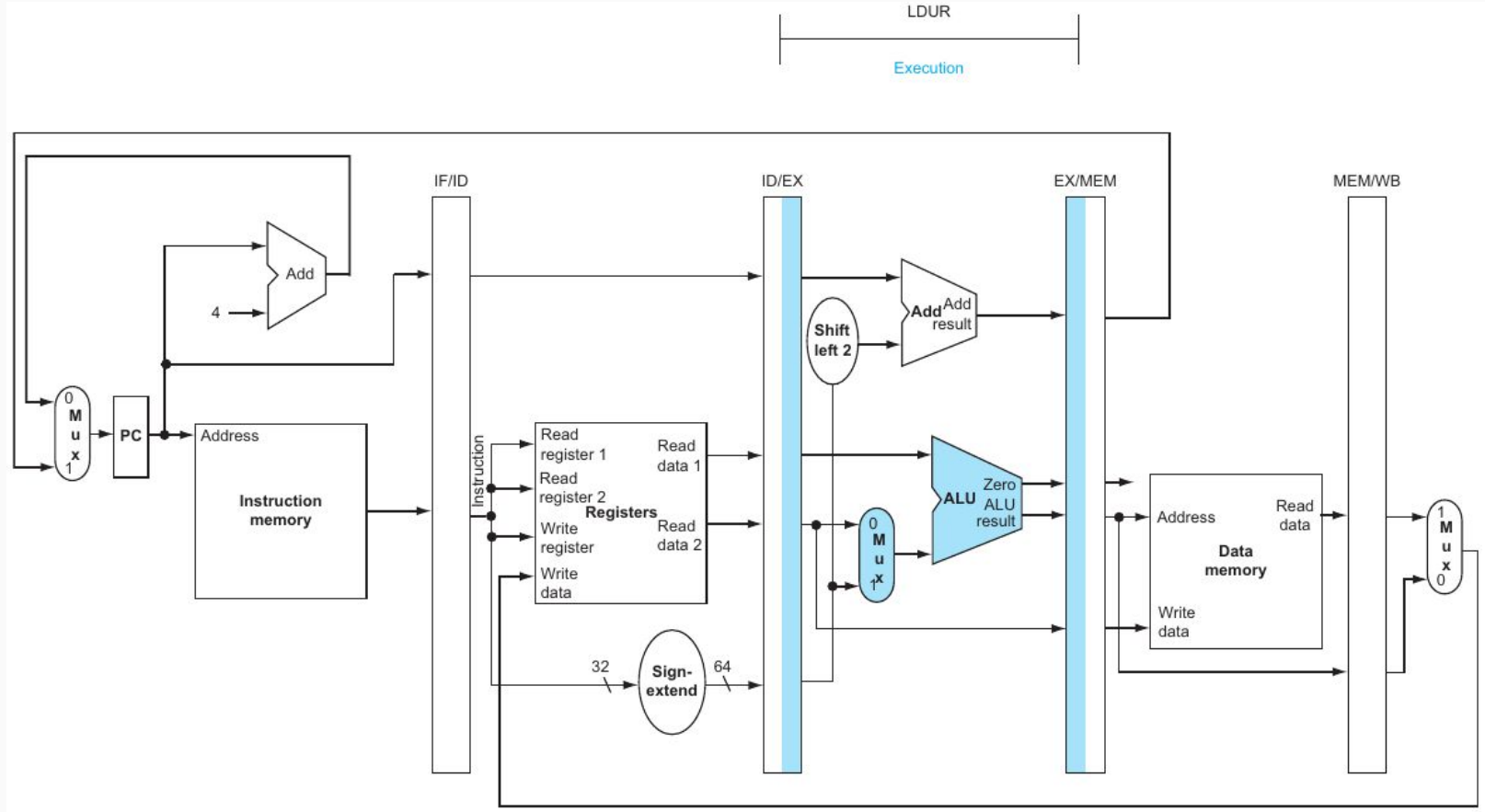
# Pipelined datapath



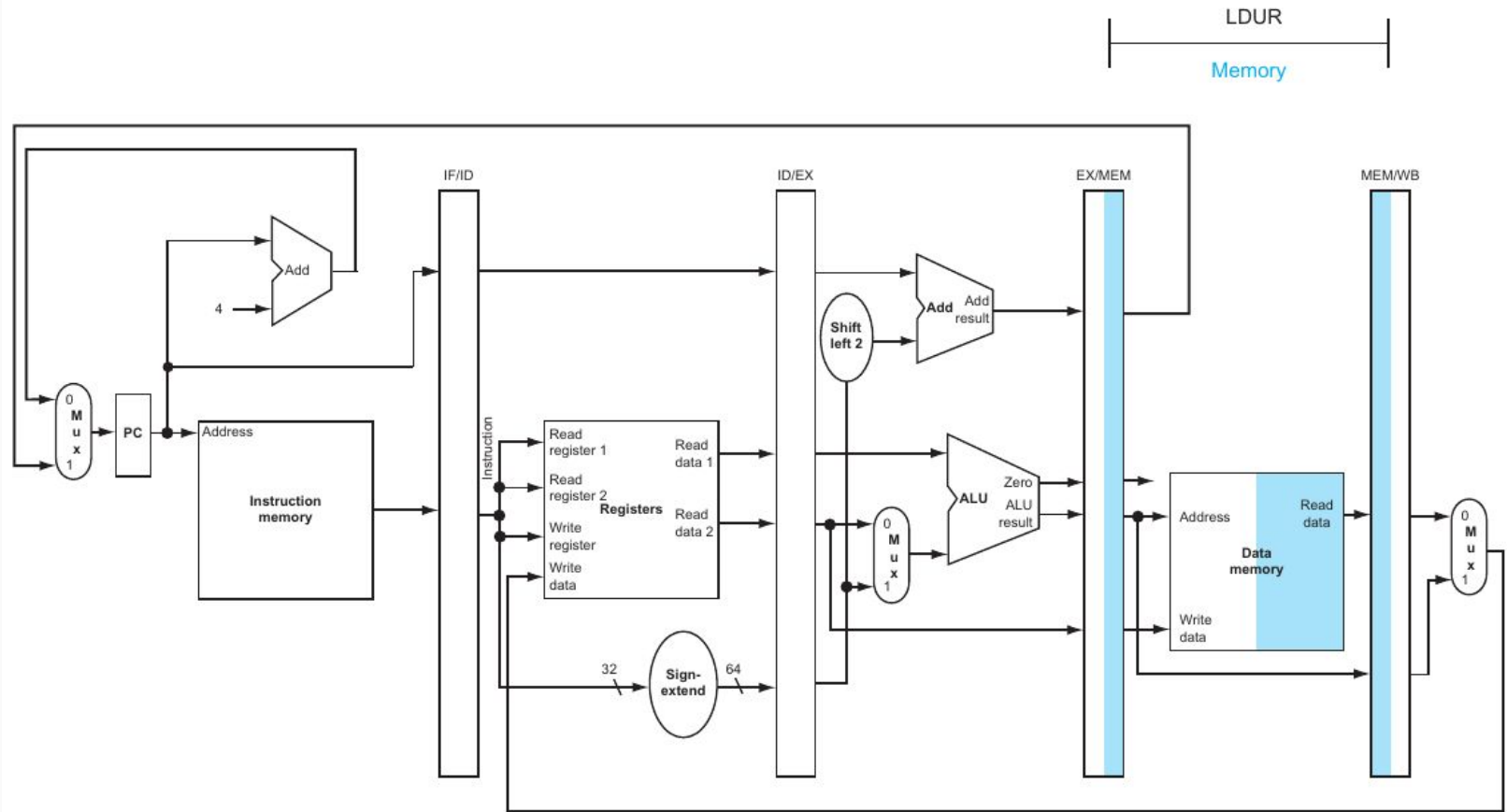
# Pipelined datapath



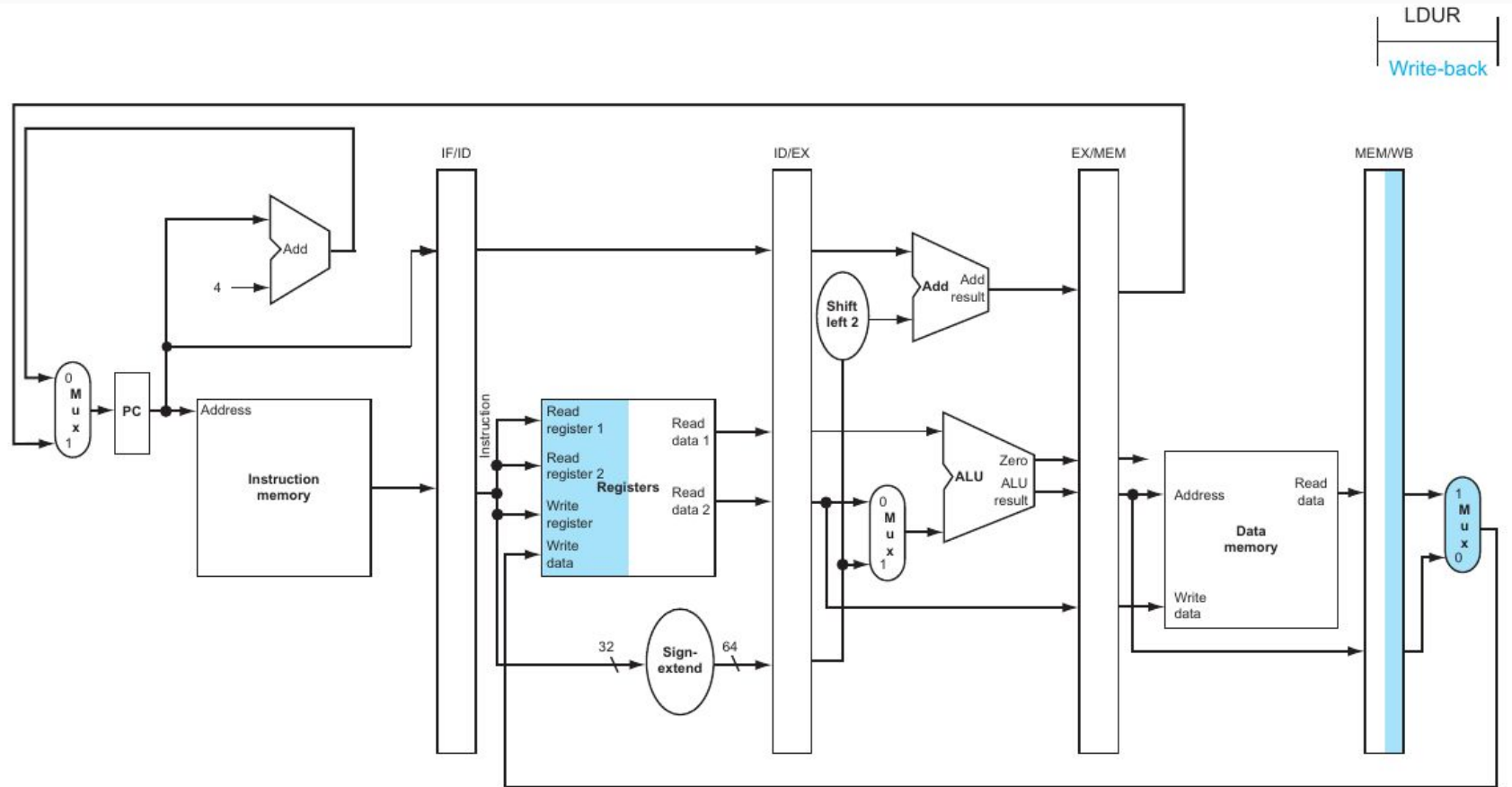
# Pipelined datapath

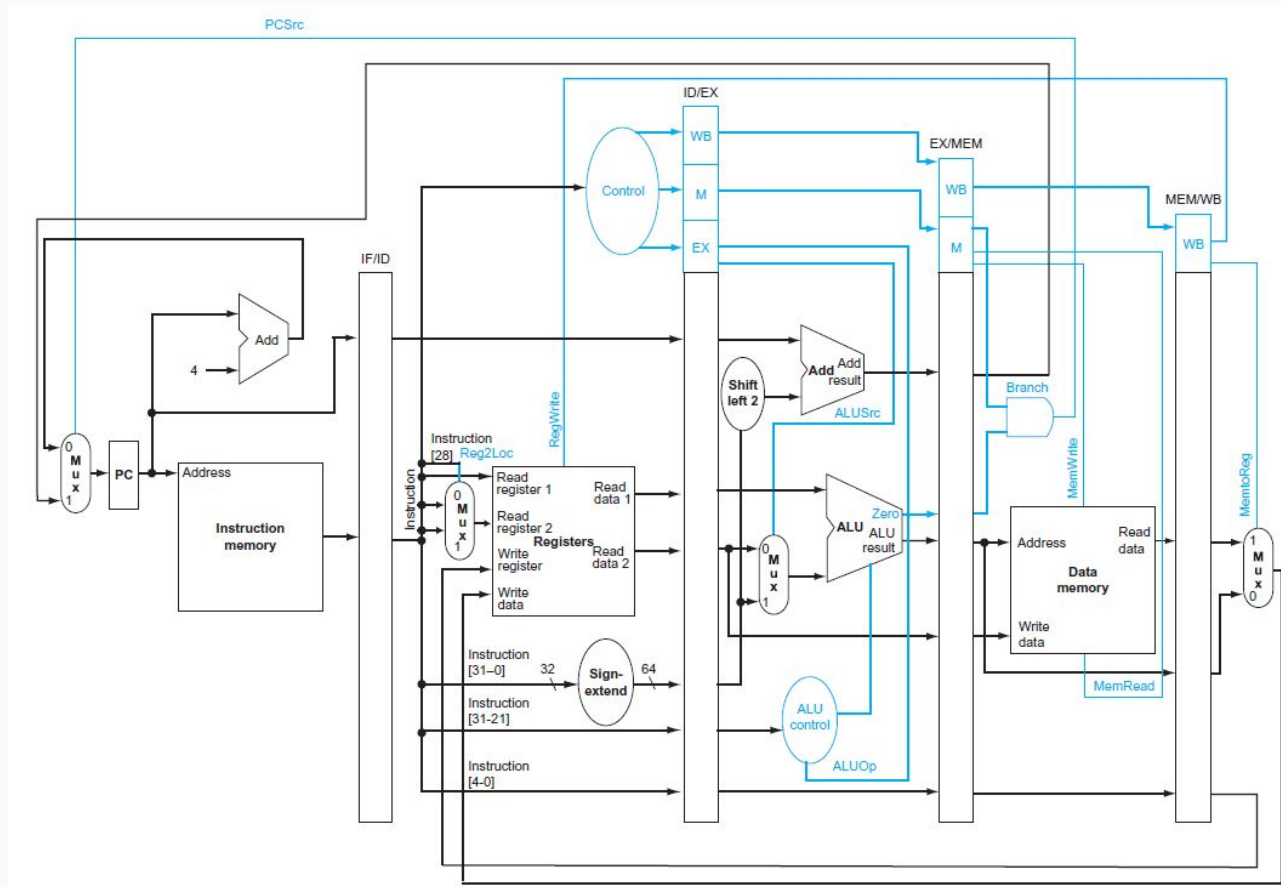


# Pipelined datapath



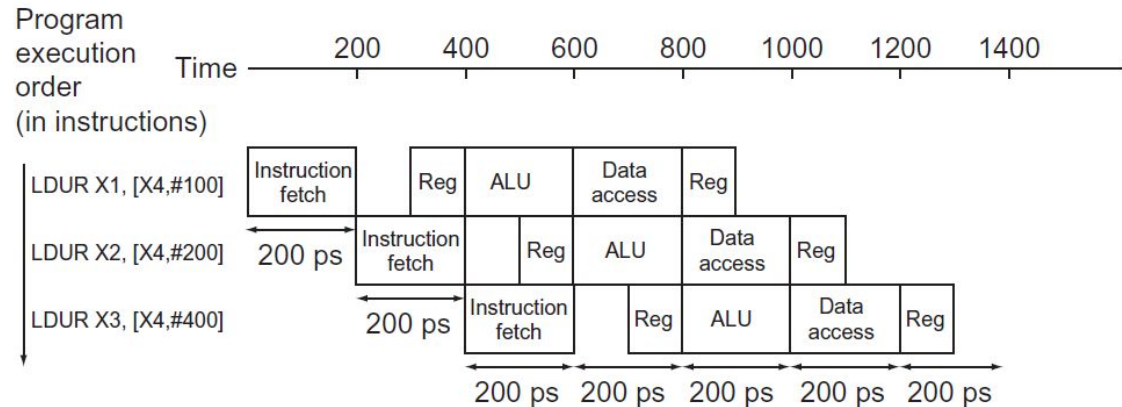
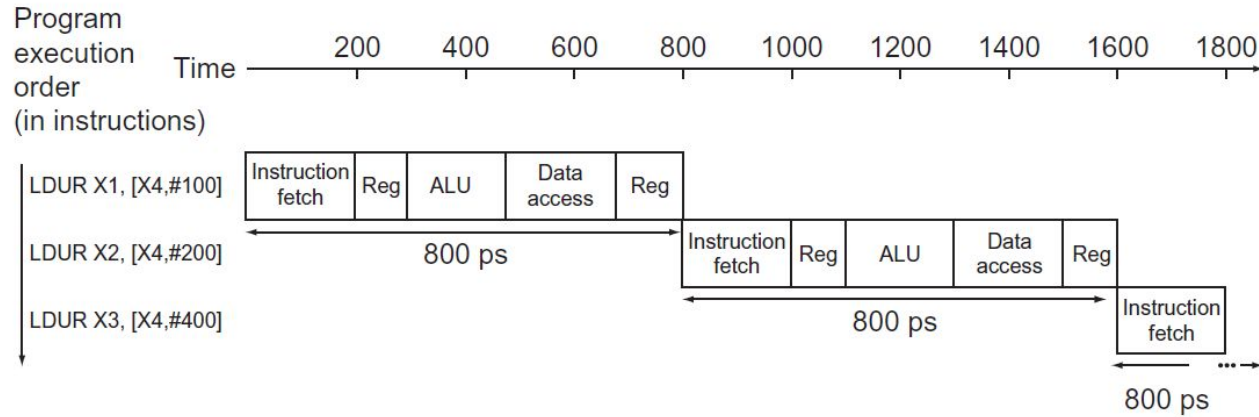
# Pipelined datapath







# Single-cycle, nonpipelined execution (top) versus pipelined execution (bottom)



# Multiple-clock-cycle diagram and single-clock-cycle figure

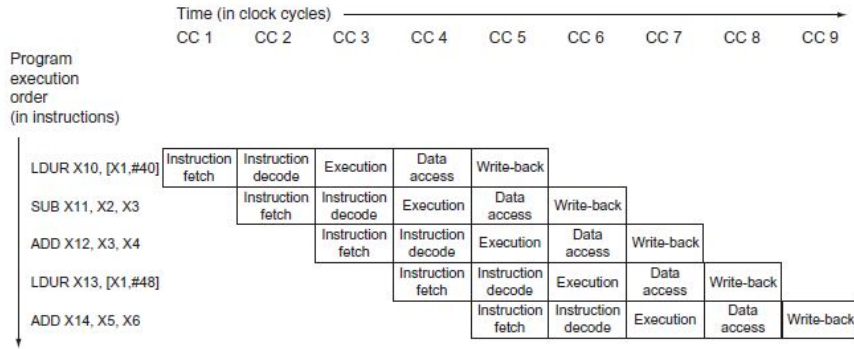


FIGURE 4.43 Traditional multiple-clock-cycle pipeline diagram of five instructions in Figure 4.42.

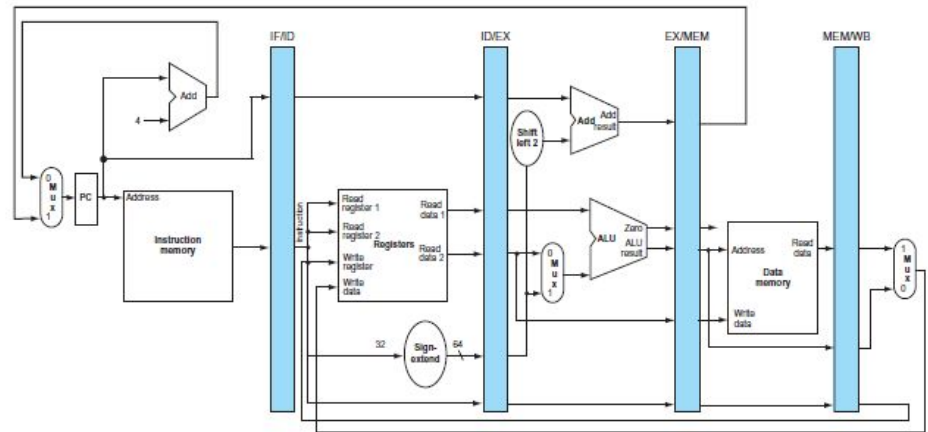
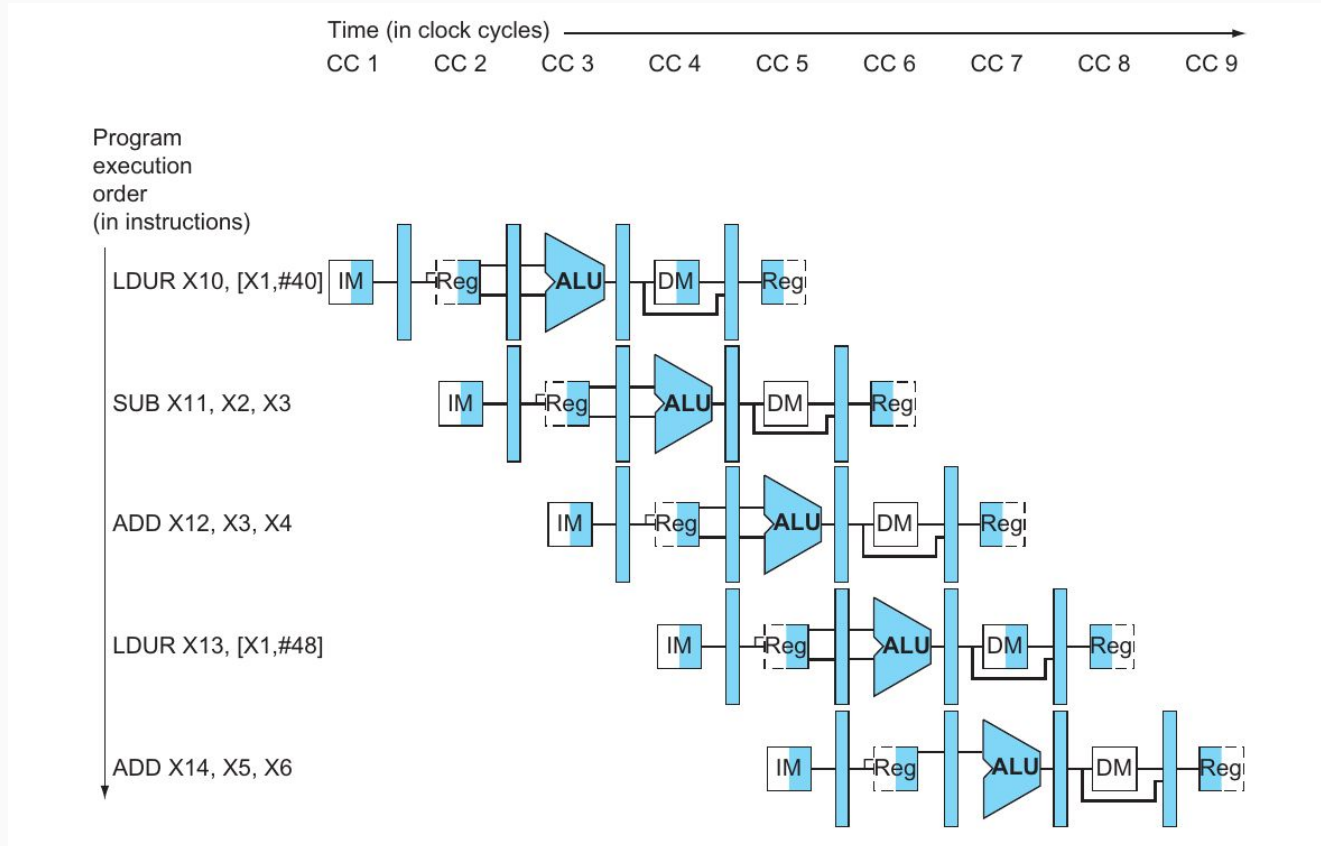


FIGURE 4.44 The single-clock-cycle diagram corresponding to clock cycle 5 of the pipeline in Figures 4.42 and 4.43. As you can see, a single-clock-cycle figure is a vertical slice through a multiple-clock-cycle diagram.

# Multiple-clock-cycle diagram and single-clock-cycle figure



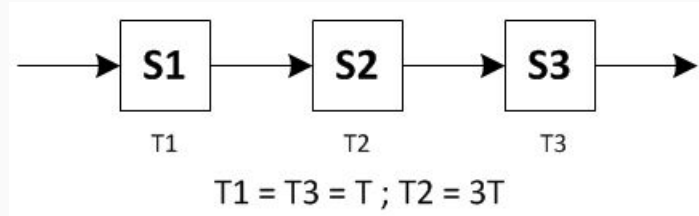
**FIGURE 4.42 Multiple-clock-cycle pipeline diagram of five instructions.** This style of pipeline representation shows the complete

## Ejercicio 1-a

En un microprocesador con tres etapas de pipeline: S1 -> S2 -> S3, con tiempos de ejecución

$$T1 = T3 = T$$

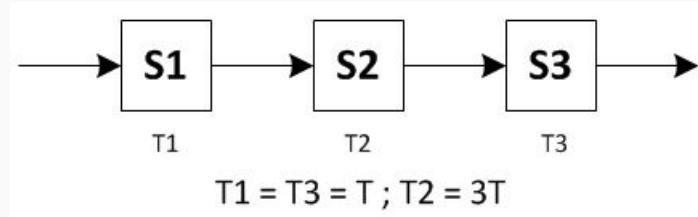
$$T2 = 3T.$$



a) ¿Cuál de los tres segmentos o etapas causa la congestión? (el cuello de botella).

## Ejercicio 1-a

En un microprocesador con tres etapas de pipeline:  $S1 \rightarrow S2 \rightarrow S3$ , con tiempos de ejecución  $T1 = T3 = T$  y  $T2 = 3T$ .

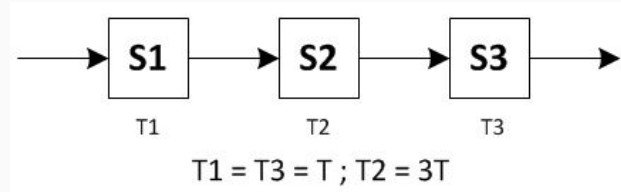


a) ¿Cuál de los tres segmentos o etapas causa la congestión? (el cuello de botella).

-> El segmento **S2**, que tiene mayor tiempo de ejecución.

## Ejercicio 1-b

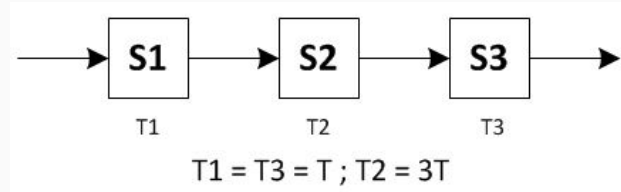
En un microprocesador con tres etapas de pipeline:  $S1 \rightarrow S2 \rightarrow S3$ , con tiempos de ejecución  $T1 = T3 = T$  y  $T2 = 3T$ .



b) Asumiendo que el segmento problemático se puede dividir en dos etapas consecutivas, ninguna de ellas con duración menor que  $T$ , ¿cuál sería la mejor partición posible? ¿Cuál sería el período de clock resultante para este nuevo pipeline de 4 etapas?

## Ejercicio 1-b

En un microprocesador con tres etapas de pipeline: S1 -> S2 -> S3, con tiempos de ejecución  $T_1 = T_3 = T$  y  $T_2 = 3T$ .



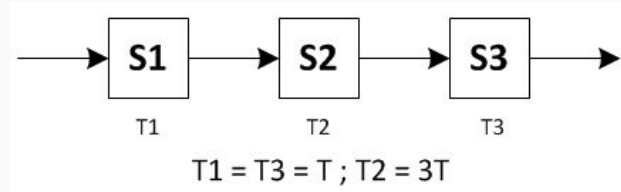
b) Asumiendo que el segmento problemático se puede dividir en dos etapas consecutivas, ninguna de ellas con duración menor que  $T$ , ¿cuál sería la mejor partición posible? ¿Cuál sería el período de clock resultante para este nuevo pipeline de 4 etapas?

->  $T_{\text{nuevo}} = \mathbf{3T/2}$

-> Nuevo período de clock =  $\mathbf{1.5T}$

## Ejercicio 1-c

En un microprocesador con tres etapas de pipeline:  $S1 \rightarrow S2 \rightarrow S3$ , con tiempos de ejecución  $T1 = T3 = T$  y  $T2 = 3T$ .

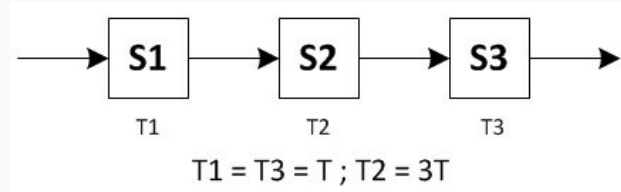


c) Asumiendo que el segmento problemático se puede dividir en varias etapas consecutivas de duración  $T$ , ¿cuál es el período de clock del microprocesador? ¿Cuál es el tiempo de ejecución entre instrucciones?



## Ejercicio 1-c

En un microprocesador con tres etapas de pipeline: S1 -> S2 -> S3, con tiempos de ejecución  $T_1 = T_3 = T$  y  $T_2 = 3T$ .



c) Asumiendo que el segmento problemático se puede dividir en varias etapas consecutivas de duración  $T$ , ¿cuál es el período de clock del microprocesador? ¿Cuál es el tiempo de ejecución entre instrucciones?

-> Período de clock del microprocesador =  **$T$**

-> Tiempo de ejecución entre instrucciones =  **$T$**

## Ejercicio 2-a

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>MEM</b>	<b>WB</b>
30ns	10ns	9ns	40ns	20ns

a) ¿Cuánto tiempo se requiere en un microprocesador sin pipeline para completar la ejecución de la instrucción de mayor latencia, es decir, la latencia del procesador completo?

## Ejercicio 2-a

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

a) ¿Cuánto tiempo se requiere en un microprocesador sin pipeline para completar la ejecución de la instrucción de mayor latencia, es decir, la latencia del procesador completo?

-> Latencia del procesador sin pipeline = 30ns + 10ns + 9ns + 40ns + 20ns = **109ns**

## Ejercicio 2-b

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>MEM</b>	<b>WB</b>
30ns	10ns	9ns	40ns	20ns

b) Si se requiere ejecutar esa instrucción en un microprocesador con pipeline, ¿a qué velocidad debería trabajar el clock?

## Ejercicio 2-b

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

b) Si se requiere ejecutar esa instrucción en un microprocesador con pipeline, ¿a qué velocidad debería trabajar el clock?

-> Período de clock del procesador con pipeline ( $T$ ) = 40ns

-> Velocidad del clock =  $1/T$  = **25MHz**

## Ejercicio 2-c

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

<b>IF</b>	<b>ID</b>	<b>EX</b>	<b>MEM</b>	<b>WB</b>
30ns	10ns	9ns	40ns	20ns

c) ¿Cuál es el tiempo de ejecución de una instrucción en un microprocesador con pipeline? ¿Cada cuanto se ejecuta una nueva instrucción en este procesador?

## Ejercicio 2-c

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

c) ¿Cuál es el tiempo de ejecución de una instrucción en un microprocesador con pipeline? ¿Cada cuanto se ejecuta una nueva instrucción en este procesador?

-> Tiempo de ejecución de una instrucción con pipeline =  $40\text{ns} * 5 \text{ etapas} = \mathbf{200\text{ns}}$

-> Tiempo entre instrucciones = **40ns**

## Ejercicio 2-d

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

d) Si un microprocesador con pipeline ejecuta 3 instrucciones consecutivas ¿cuál es la ganancia de velocidad de un procesador con pipeline respecto de uno sin pipeline? ¿Y si se ejecutan 1000 instrucciones consecutivas?



## Ejercicio 2-d

Asumiendo que las etapas de un procesador ARM tienen las siguientes latencias:

IF	ID	EX	MEM	WB
30ns	10ns	9ns	40ns	20ns

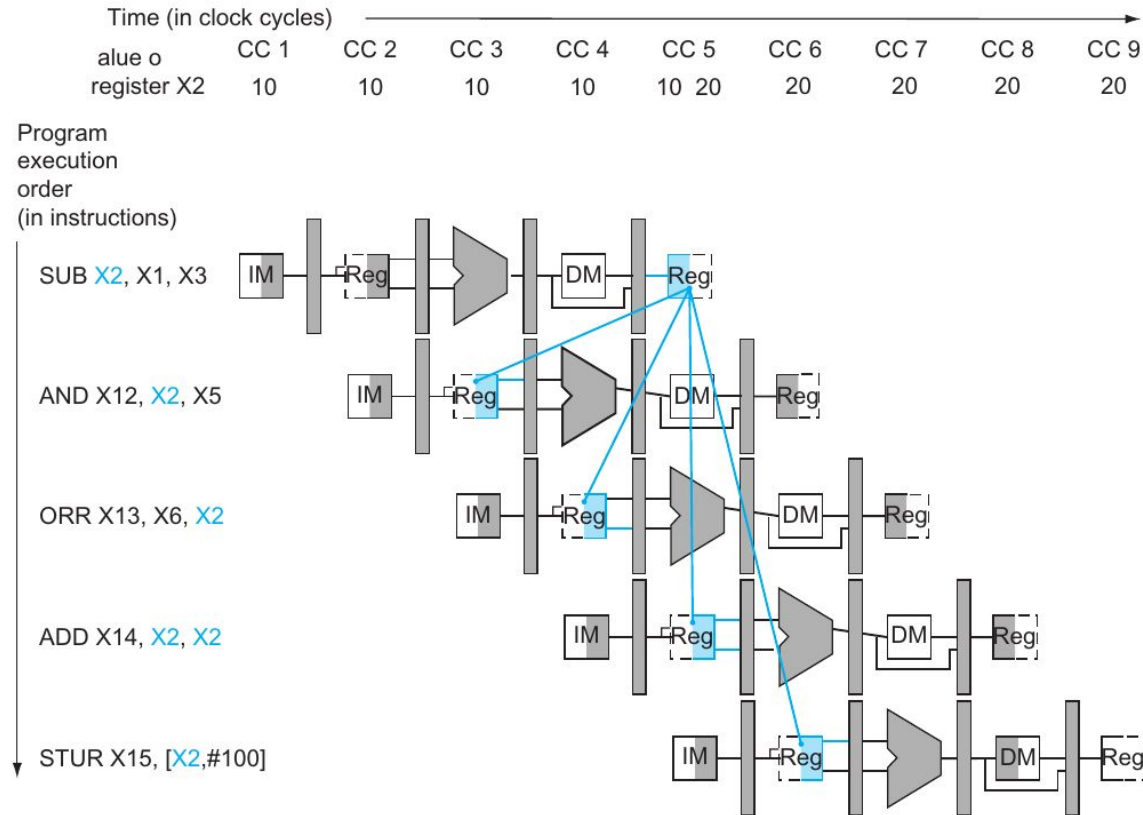
d) Si un microprocesador con pipeline ejecuta 3 instrucciones consecutivas ¿cuál es la ganancia de velocidad de un procesador con pipeline respecto de uno sin pipeline? ¿Y si se ejecutan 1000 instrucciones consecutivas?

$$\text{Ganancia de velocidad} = \frac{\text{Tiempo de ejecución sin pipeline}}{\text{Tiempo de ejecución con pipeline}}$$

$$\rightarrow \text{Ganancia}_{3\_instrucciones} = (3 \cdot 109\text{ns}) / (5 \cdot 40\text{ns} + 2 \cdot 40\text{ns}) = \mathbf{1.168}$$

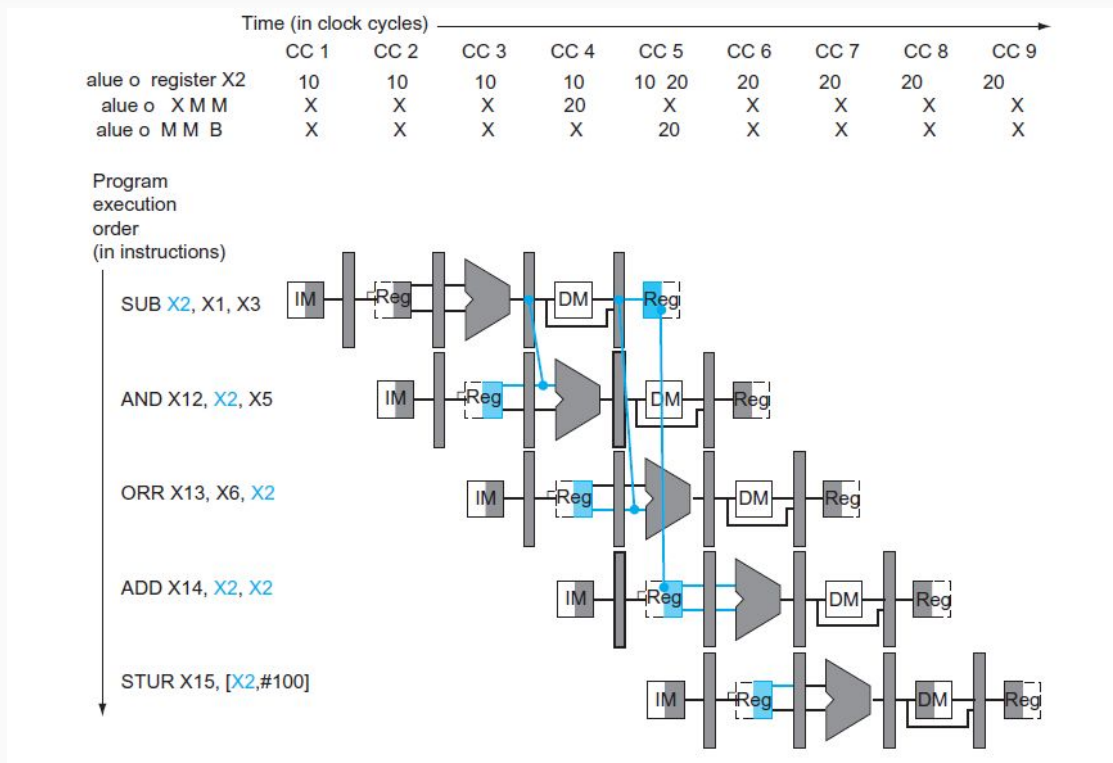
$$\rightarrow \text{Ganancia}_{1000\_instrucciones} = (1000 \cdot 109\text{ns}) / (5 \cdot 40\text{ns} + 999 \cdot 40\text{ns}) = \mathbf{2.714}$$

# Dependencias de datos



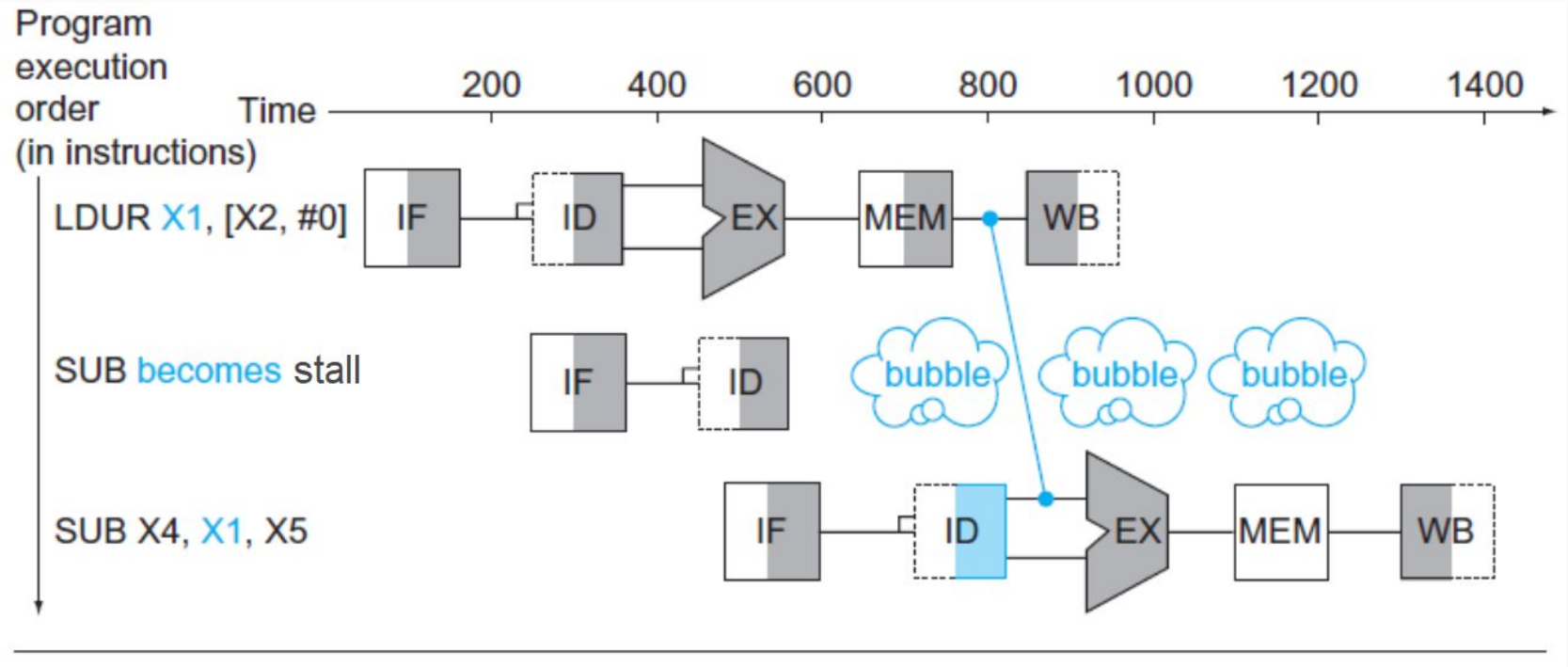
**FIGURE 4.51 Pipelined dependences in a five-instruction sequence using simplified datapaths to show the dependences.** All the dependent actions are shown in color, and “CC 1” at the top of the figure means clock cycle 1. The first instruction

# Data Hazards: Forwarding



**FIGURE 4.52 The dependences between the pipeline registers move forward in time, so it is possible to supply the inputs to the ALU needed by the AND instruction and ORR instruction by forwarding the results found in the pipeline registers.** The values in the pipeline registers show that the desired value is available before it is written into the register file. We assume that the register file forwards values that are read and written during the same clock cycle, so the ADD does not stall, but the values come from the register file instead of a pipeline register. Register file “forwarding”—that is, the read gets the value of the write in that clock cycle—is why clock cycle 5 shows register X2 having the value 10 at the beginning and –20 at the end of the clock cycle.

## Data Hazards: Forwarding stall



We need a stall even with forwarding when an R-format instruction following a load tries to use the data.

## Ejercicio 4 - 2

- a) Analizar en el código las dependencias de datos. En cada caso indicar: los números de las instrucciones involucradas y el operando en conflicto.
- b) Mostrar el orden de ejecución de las instrucciones y determinar cuales generan *data hazards*. En cada caso indicar en qué etapa se genera el hazard.
- b) Mostrar el orden de ejecución de las instrucciones utilizando un procesador con *forwarding stall*.
- c) Reescribir la sección de código alterando el orden de las instrucciones para evitar *stalls* innecesarios. Mostrar el nuevo orden de ejecución.
- d) ¿Cuántos ciclos toma la ejecución del código en cada caso?

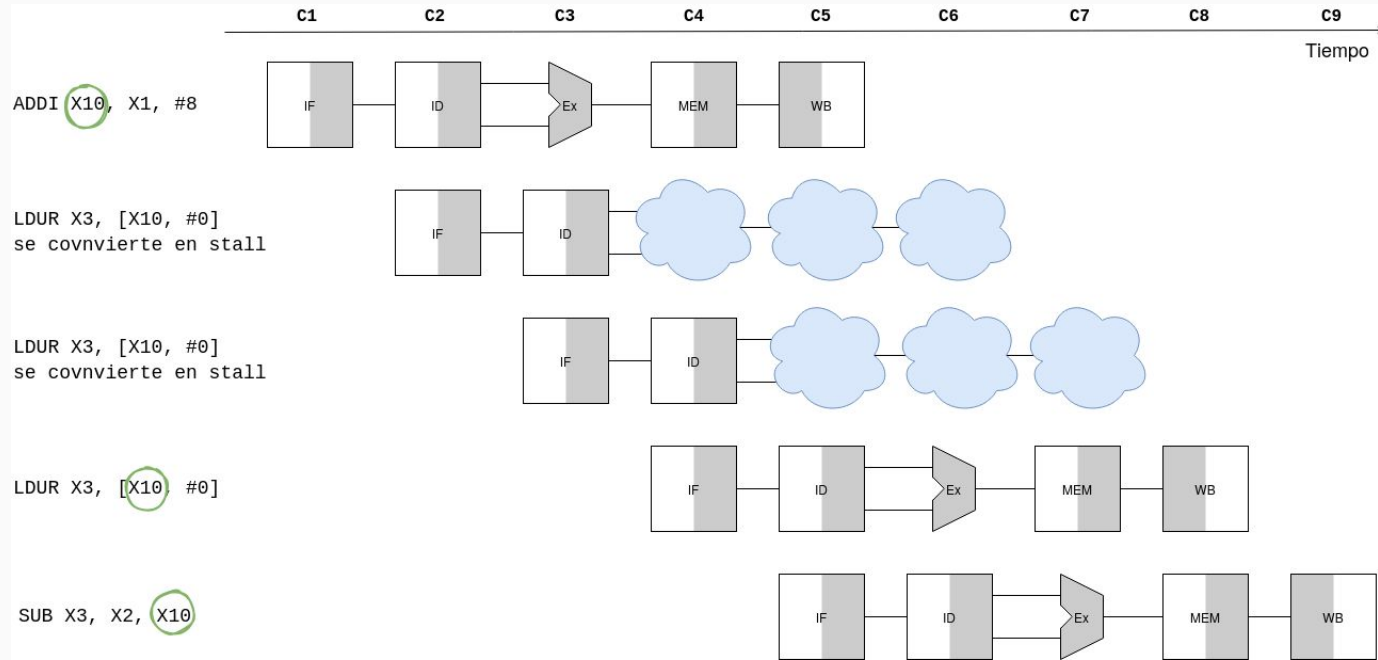
```
1> ADDI X10, X1, #8
2> LDUR X3, [X10, #0]
3> SUB X3, X2, X10
```

## Ejercicio 4 - 2

```
ADDI X10, X1, #8
LDUR X3, [X10, #0]
SUB X3, X2, X10
```

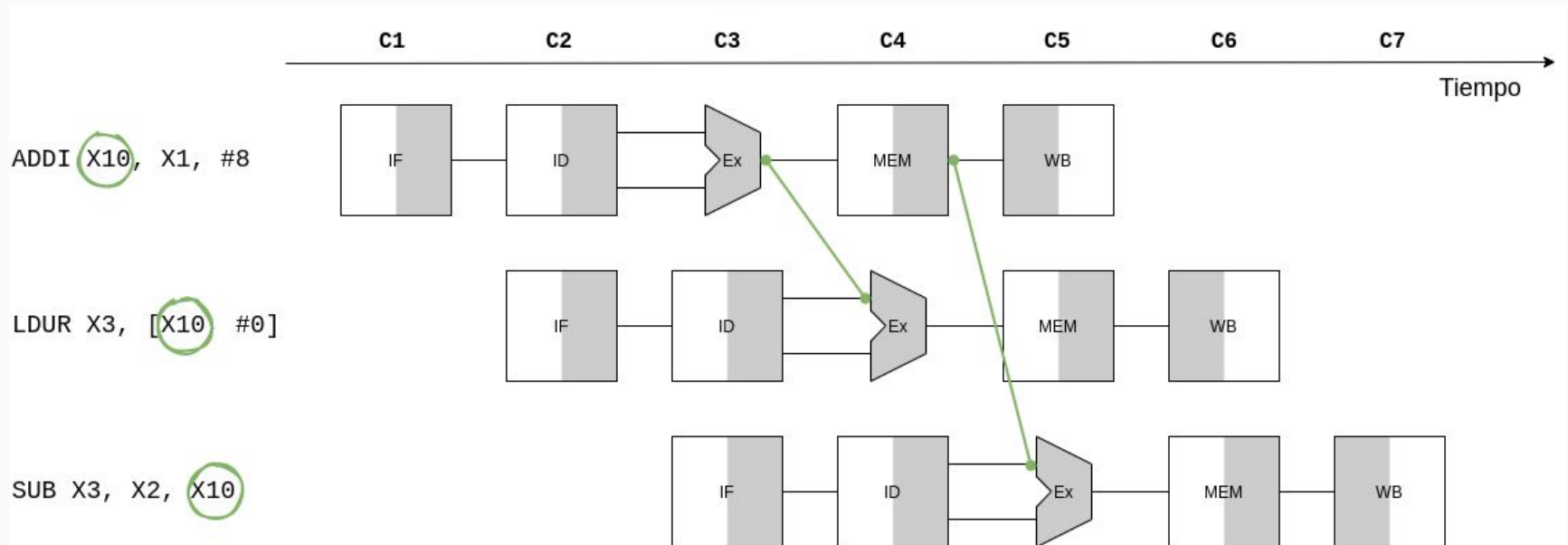
Dependencia Tipo	instr. 1, instr. 2, reg.
Datos	1, 2, X10
Datos	1, 3, X10

Ejecución con stalls



## Ejercicio 4 - 2

### Ejecución con forwarding - stalls



## Ejercicio 4 - 3

Encontrar las dependencias de datos (RAW):

```
1> LDUR X1, [X10, #0]
2> LDUR X2, [X10, #8]
3> ADD X3, X1, X2
4> STUR X3, [X10, #24]
5> LDUR X4, [X10, #16]
6> ADD X5, X1, X4
7> STUR X5, [X10, #32]
```



## Ejercicio 4 - 3

Encontrar las dependencias de datos (RAW):

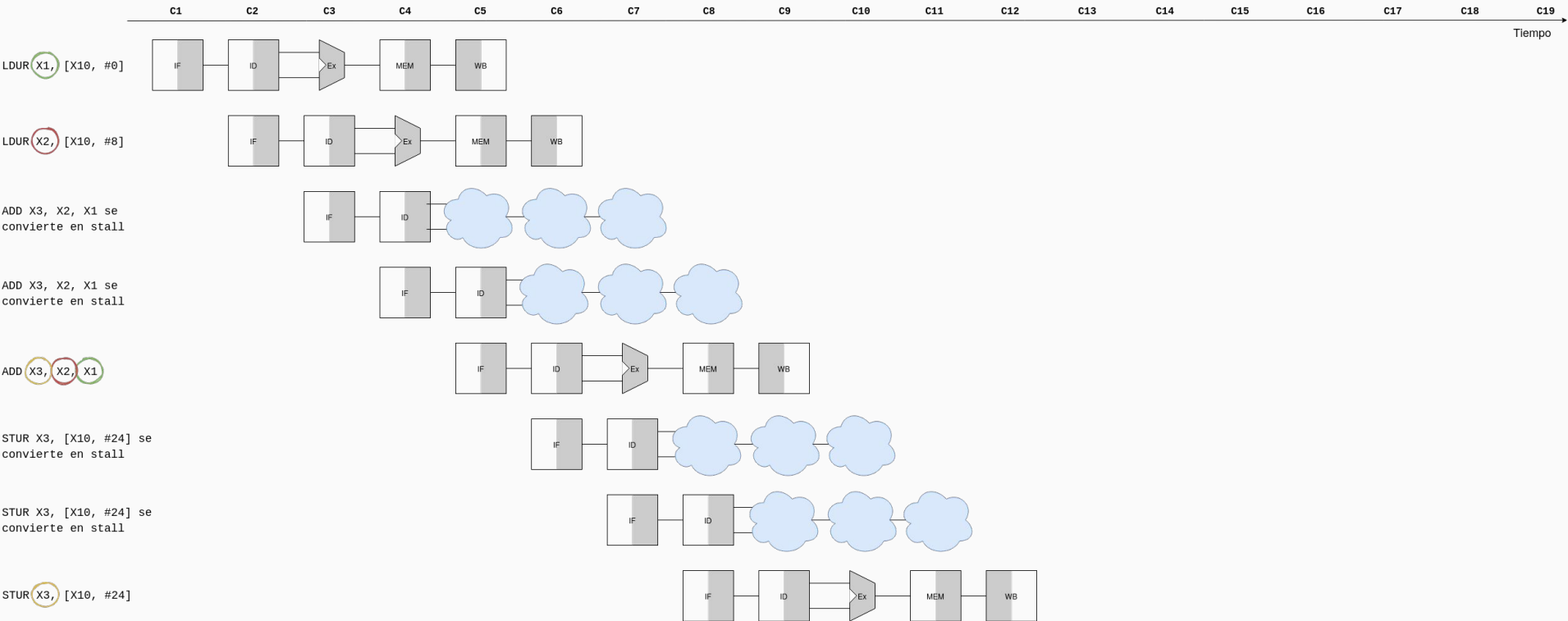
```
graph TD; I1[1> LDUR X1, [X10, #0]] --> I2[2> LDUR X2, [X10, #8]]; I2 --> I3[3> ADD X3, X2, X1]; I3 --> I4[4> STUR X3, [X10, #24]]; I4 --> I5[5> LDUR X4, [X10, #16]]; I5 --> I6[6> ADD X5, X1, X4]; I6 --> I7[7> STUR X5, [X10, #32]];
```

1> LDUR X1, [X10, #0]  
2> LDUR X2, [X10, #8]  
3> ADD X3, X2, X1  
4> STUR X3, [X10, #24]  
5> LDUR X4, [X10, #16]  
6> ADD X5, X1, X4  
7> STUR X5, [X10, #32]

Dependencia Tipo	instr. 1, instr. 2, reg.
Datos	1, 3, X1
Datos	2, 3, X2
Datos	3, 4, X3
Datos	5, 6, X4
Datos	6, 7, X5
Datos	1, 6, X1

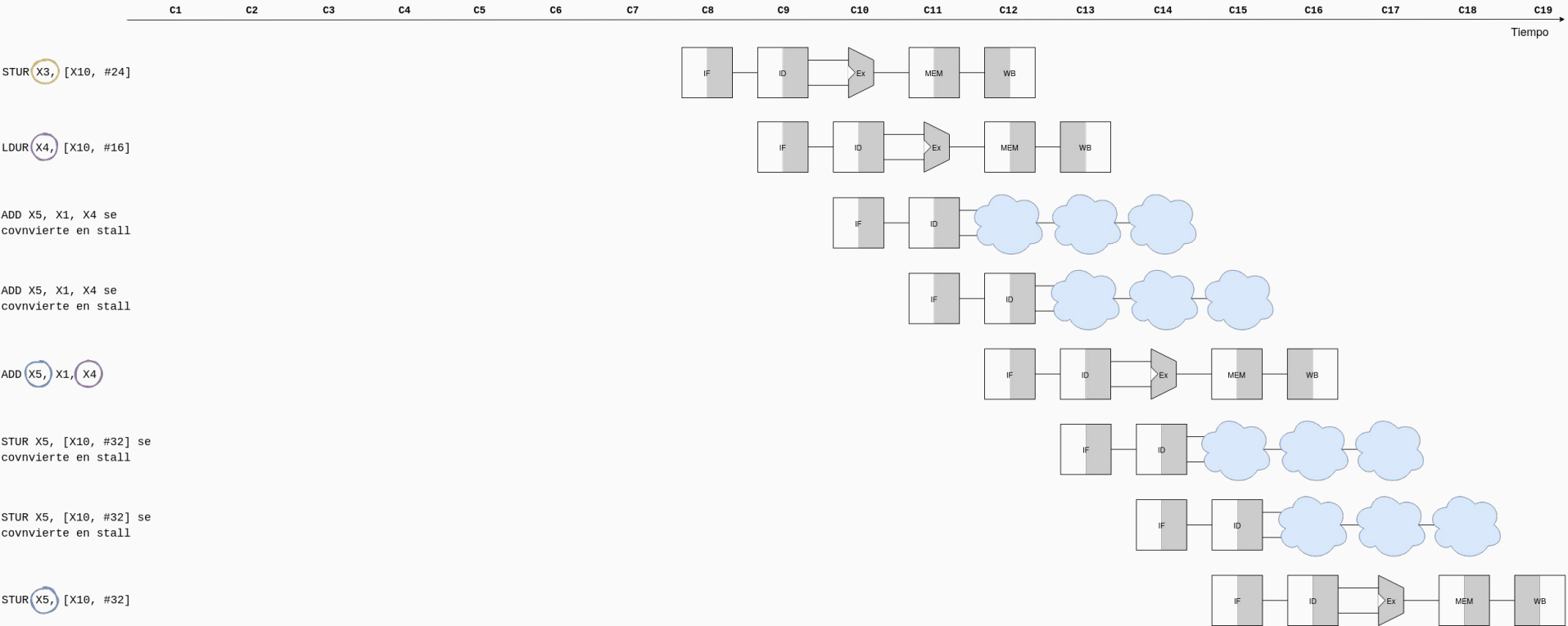
# Ejercicio 4 - 3

## Ejecución con stalls



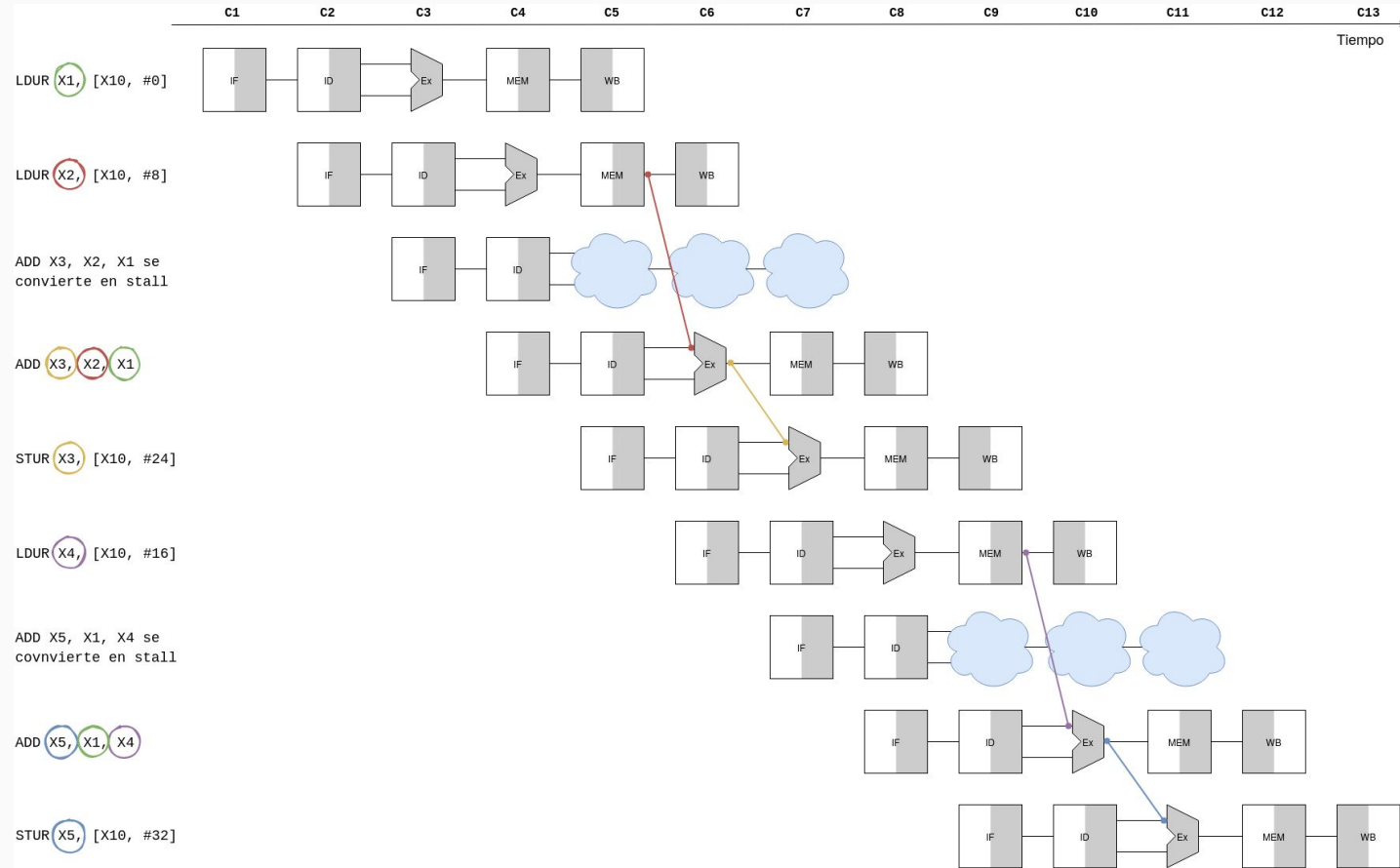
# Ejercicio 4 - 3

## Ejecución con stalls



## Ejercicio 4 - 3

### Ejecución con forwarding - stalls



## Ejercicio 4 - 3

### Orden inicial

```
1> LDUR X1, [X10, #0]
2> LDUR X2, [X10, #8]
3> ADD X3, X2, X1
4> STUR X3, [X10, #24]
5> LDUR X4, [X10, #16]
6> ADD X5, X1, X4
7> STUR X5, [X10, #32]
```

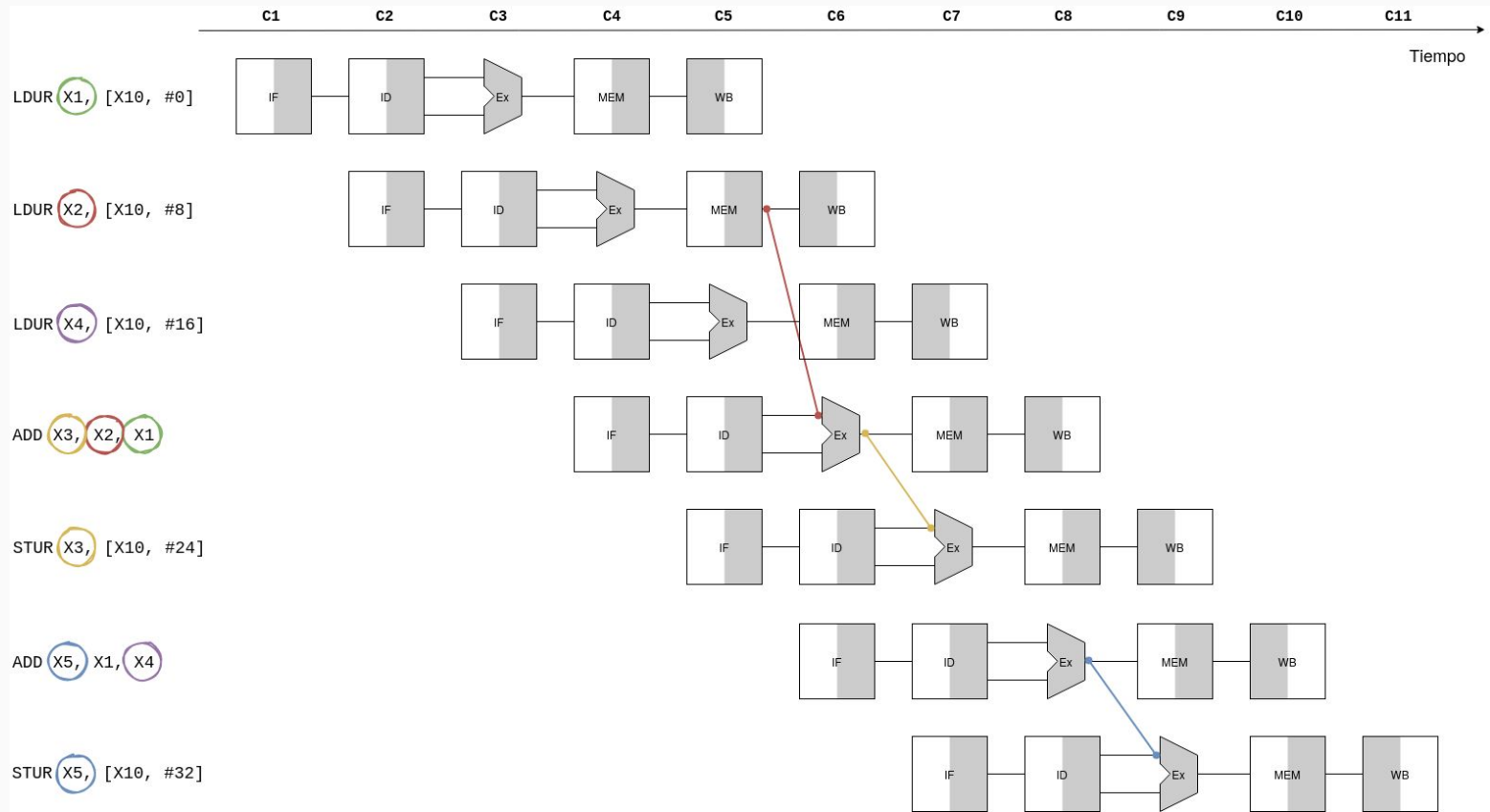


### Reordenamiento

```
1> LDUR X1, [X10, #0]
2> LDUR X2, [X10, #8]
3> LDUR X4, [X10, #16]
4> ADD X3, X2, X1
5> STUR X3, [X10, #24]
6> ADD X5, X1, X4
7> STUR X5, [X10, #32]
```

# Ejercicio 4 - 3

## Ejecución con forwarding - stalls luego de reordenar el código



# Hazard de control

**Técnicas estáticas:** El procesador siempre toma la misma decisión por defecto

- **Not taken (NT):** Por defecto se asume que el salto no se va a tomar y se carga la siguiente instrucción, en caso que el salto deba tomarse, se hace *flush* y se realiza el fetch de la instrucción correcta. La penalidad es de 3 ciclos de clock.

- **Taken (T):** Por defecto el procesador asume que el salto se toma y carga la instrucción que indica el salto. Esto en un procesador como el que venimos estudiando tiene una penalidad de 3 clocks siempre (la dirección se calcula en la etapa de memory).

**Técnicas dinámicas:** El procesador decide si saltar o no considerando el historial de saltos.

## Ejercicio 7

Para el siguiente fragmento de código de instrucciones LEGv8 que se ejecuta en un procesador con forwarding stall:

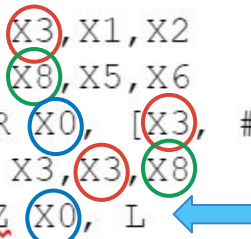
```
1>      SUB X3,X1,X2
2>      ORR X8,X5,X6
3> L:    LDUR X0, [X3, #0]
4>      ADD X3,X3,X8
5>      CBNZ X0, L
6>      ADD X8,X0,X3
```

- a) Analizar en el código las dependencias de datos y de control, determinar cuales generarían *hazards* y mediante qué técnica se evita. En cada caso indicar: los números de las instrucciones involucradas, el operando en conflicto y el tipo de hazard.
- b) Mostrar el orden de ejecución y los recursos utilizados por las instrucciones para el segmento de código dado, suponiendo que CBNZ no se toma.
- c) Mostrar el orden de ejecución y los recursos utilizados por las instrucciones para el segmento de código dado, suponiendo que CBNZ se toma una vez.



## Ejercicio 7-a

```
1>      SUB X3, X1, X2
2>      ORR X8, X5, X6
3> L:   LDUR X0, [X3, #0]
4>      ADD X3, X3, X8
5>      CBNZ X0, L
6>      ADD X8, X0, X3
```



Dependencia Tipo	Instrucciones	Genera Hazzard? Se evita?
Datos X3	1-3	Si, forwarding
Datos X3	1-4	No
Datos X8	2-4	Si, forwarding
Datos X0	3-5	Si, forwarding
Control	5	Sólo si salta, no

## Ejercicio 7-a (el salto NO se toma)

```
1>      SUB X3,X1,X2
2>      ORR X8,X5,X6
3> L:   LDUR X0, [X3, #0]
4>      ADD X3,X3,X8
5>      CBNZ X0, L
6>      ADD X8,X0,X3
```

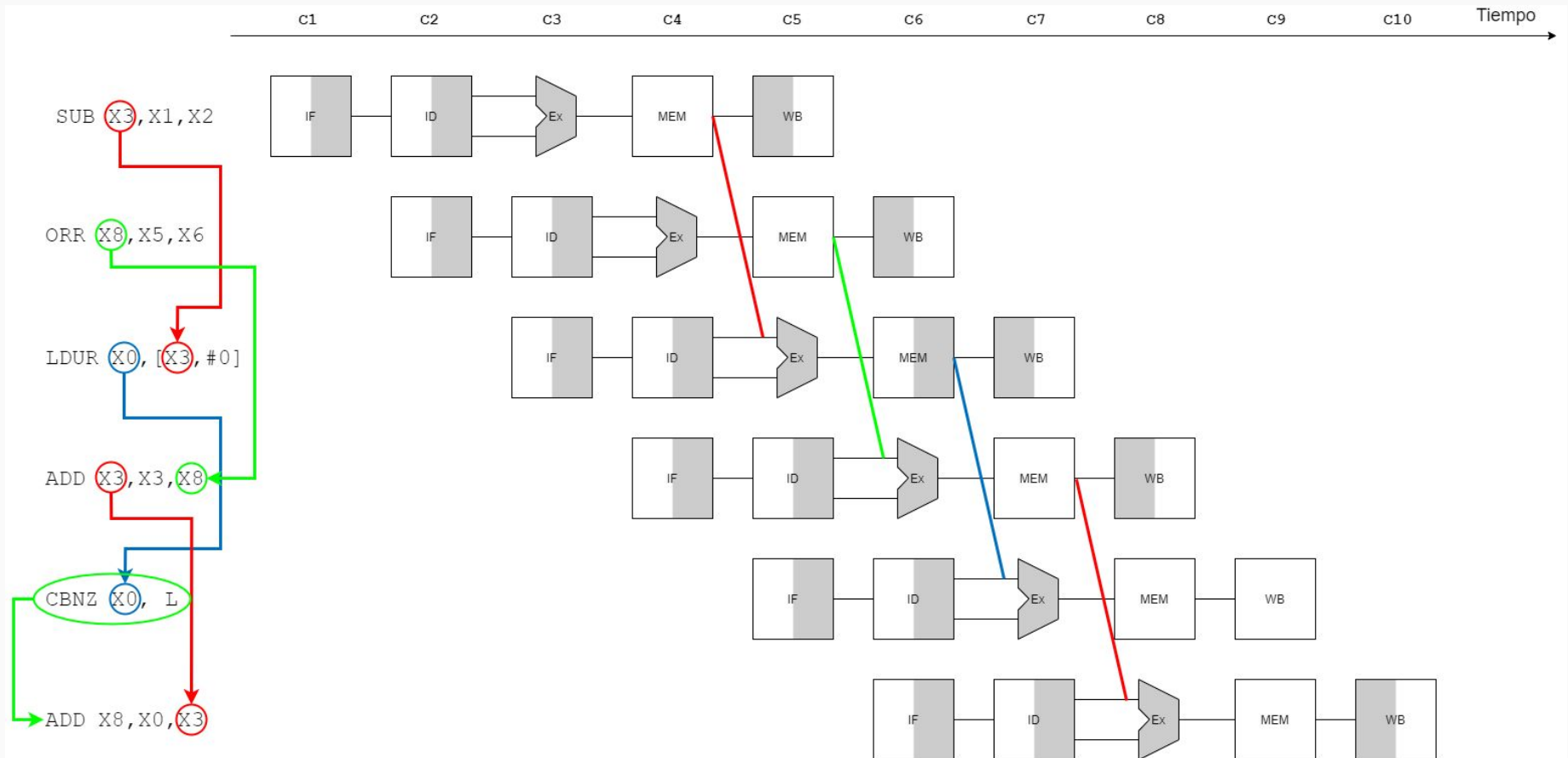
Dependencia Tipo	Instrucciones	Genera Hazzard? Se evita?
Datos X3	1-3	Si, forwarding
Datos X3	1-4	No
Datos X8	2-4	Si, forwarding
Datos X0	3-5	Si, forwarding
Control	5	Sólo si salta, no
Datos X0 (cond)	3-6	No
Datos X3 (cond)	4-6	Si, forwarding

## Ejercicio 7-a (el salto SI se toma)

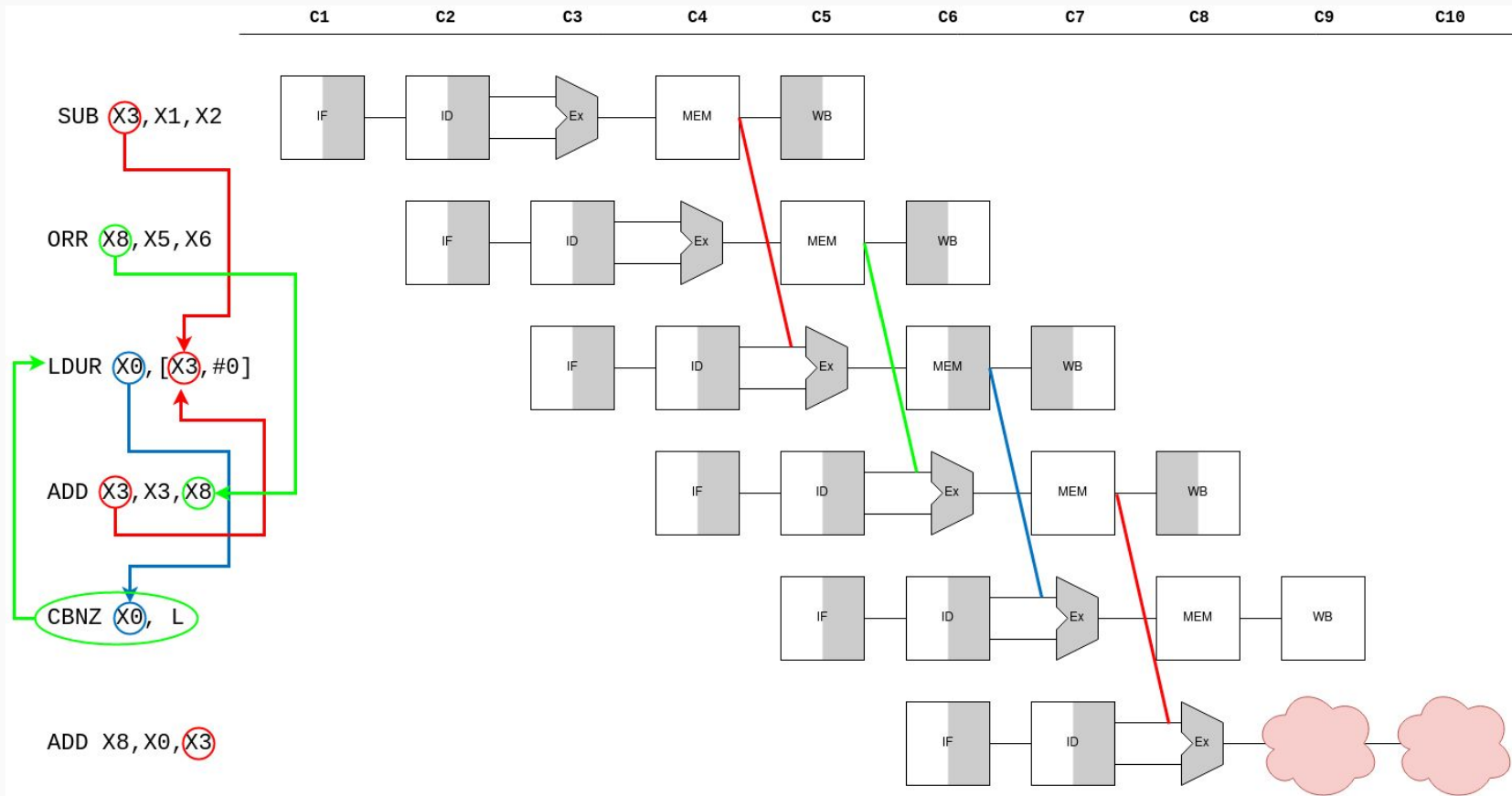
```
1>      SUB X3,X1,X2
2>      ORR X8,X5,X6
3> L:   LDUR X0, [X3, #0]
4>      ADD X3,X3,X8
5>      CBNZ X0, L
6>      ADD X8,X0,X3
```

Dependencia Tipo	Instrucciones	Genera Hazzard? Se evita?
Datos X3	1-3	Si, forwarding
Datos X3	1-4	No
Datos X8	2-4	Si, forwarding
Datos X0	3-5	Si, forwarding
Control	5	Sólo si salta, no
Datos X0 (cond)	3-6	No
Datos X3 (cond)	4-6	Si, forwarding
Datos X3 (cond)	4-3	Si, forwarding
Datos X3 (cond)	4-4 (en distintas iteraciones)	No

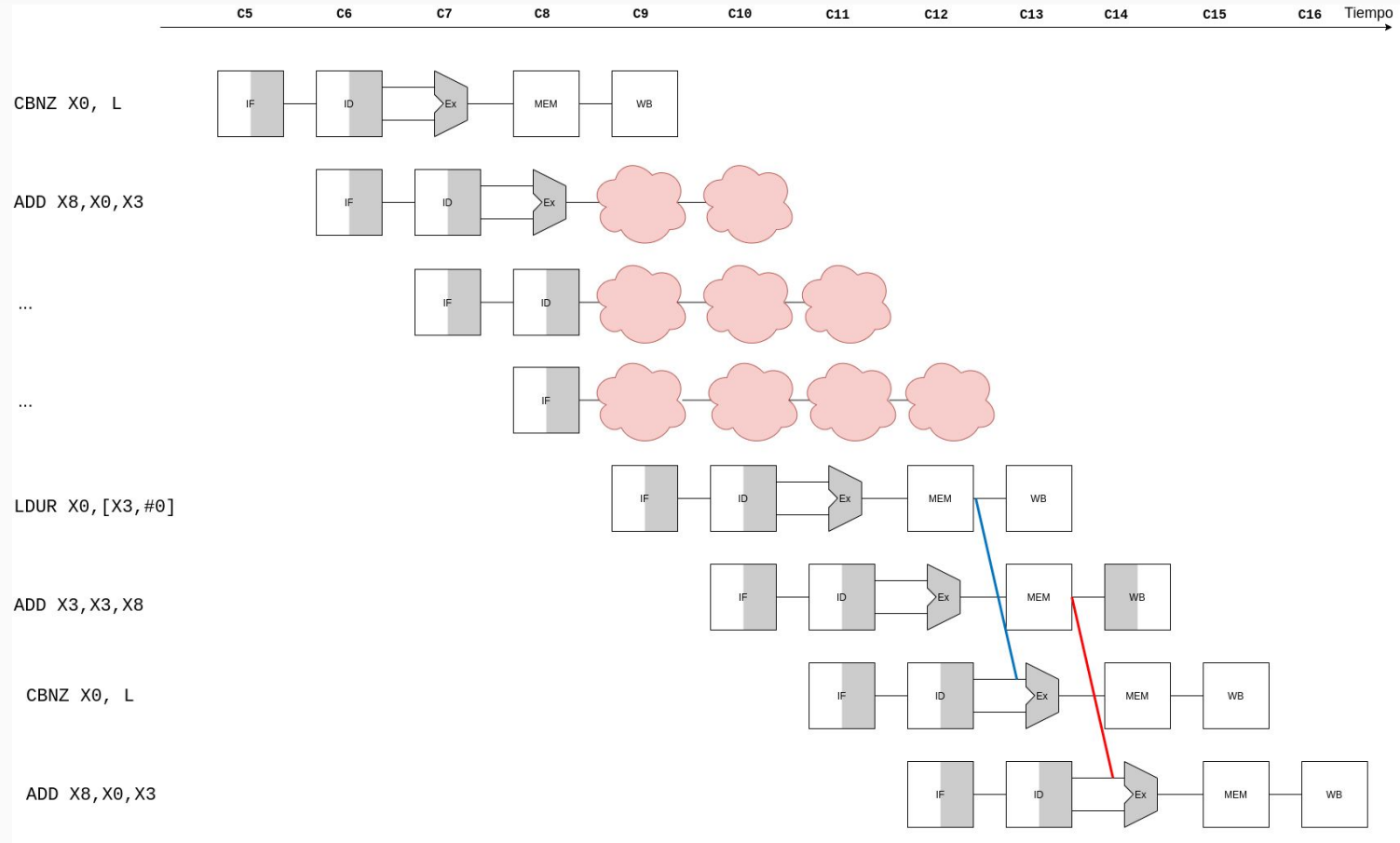
## Ejercicio 7-b (El salto NO se toma)



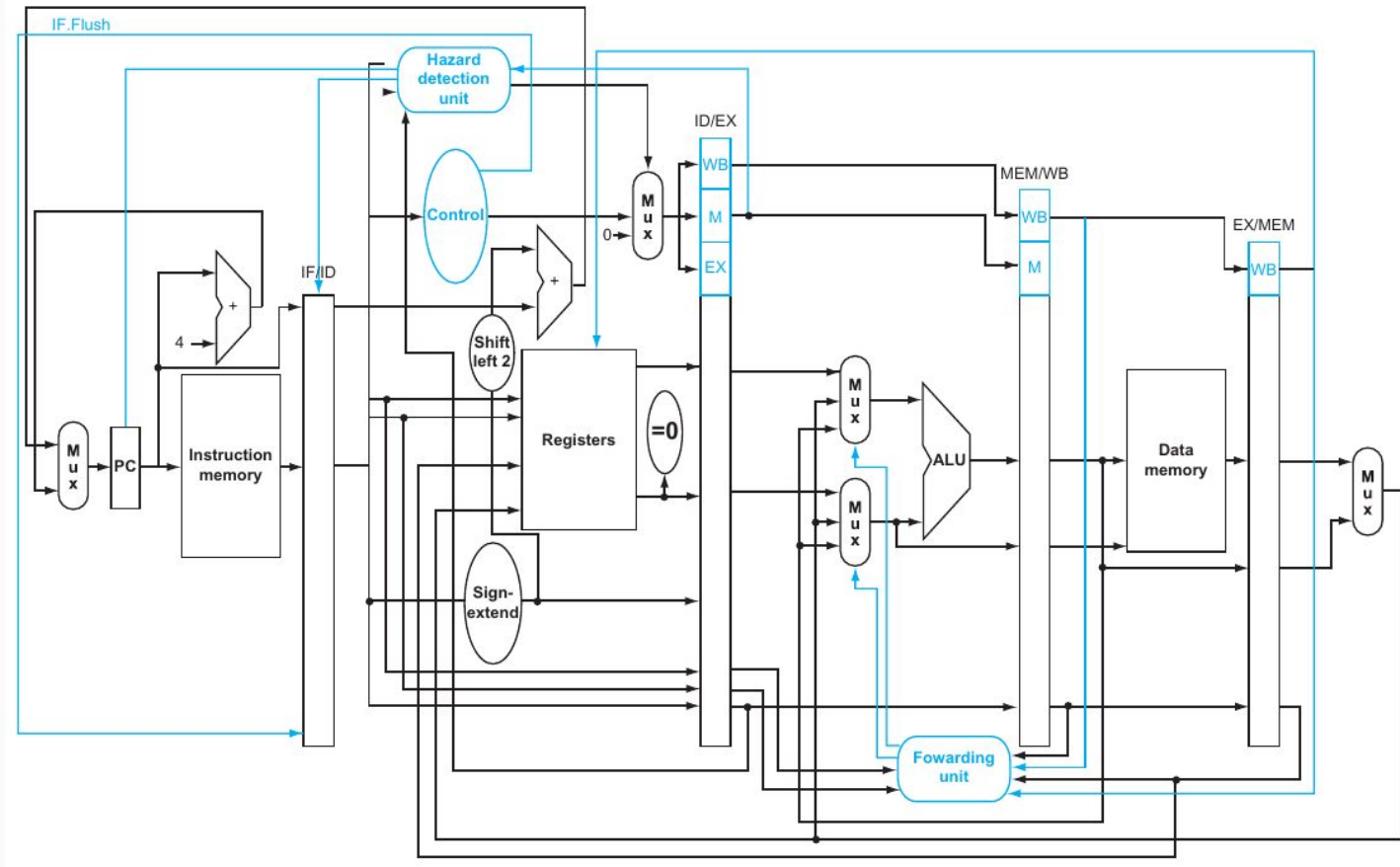
## Ejercicio 7-c (El salto SI se toma)



## Ejercicio 7-c (El salto SI se toma)



# El procesador optimizado para saltos



# Bibliografía

- PATTERSON, David A.; HENNESSY, John L. *Computer Organization and Design ARM Edition: The Hardware Software Interface*. Morgan kaufmann, 2016 - CH 4.