Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación


# IIC2343 – Arquitectura de Computadores

## Paralelismo a nivel de instrucción (ILP)

**Profesor:** Jurgen Heysen

# Aumentar al velocidad del clock no es la única manera de mejorar el rendimiento

- ¿Qué problemas trae aumentar el clock de un computador?
- ¿Es posible utilizar otro esquema para acelerar las instrucciones?
- Analizaremos el proceso secuencial de ejecución de las instrucciones y veremos que bajo ciertas condiciones, estas podrán ejecutarse en una fracción del tiempo original.

# ¿Cuál es el flujo de la instrucción ADD A, B en el computador básico?

1. Lectura de instrucción desde memoria (Fetch)
2. Decodificar instr. en unidad de control (Decode)
3. Ejecutar en ALU (Execute)
4. Escribir resultado en registro (Write Back)

# ¿Cuál es el flujo de la instrucción MOV A, (B) en el computador básico?

1. Lectura de instrucción desde memoria (Fetch)
2. Decodificar instr. en unidad de control (Decode)
3. Acceder a memoria (Mem)
4. Escribir resultado en registro (Write Back)

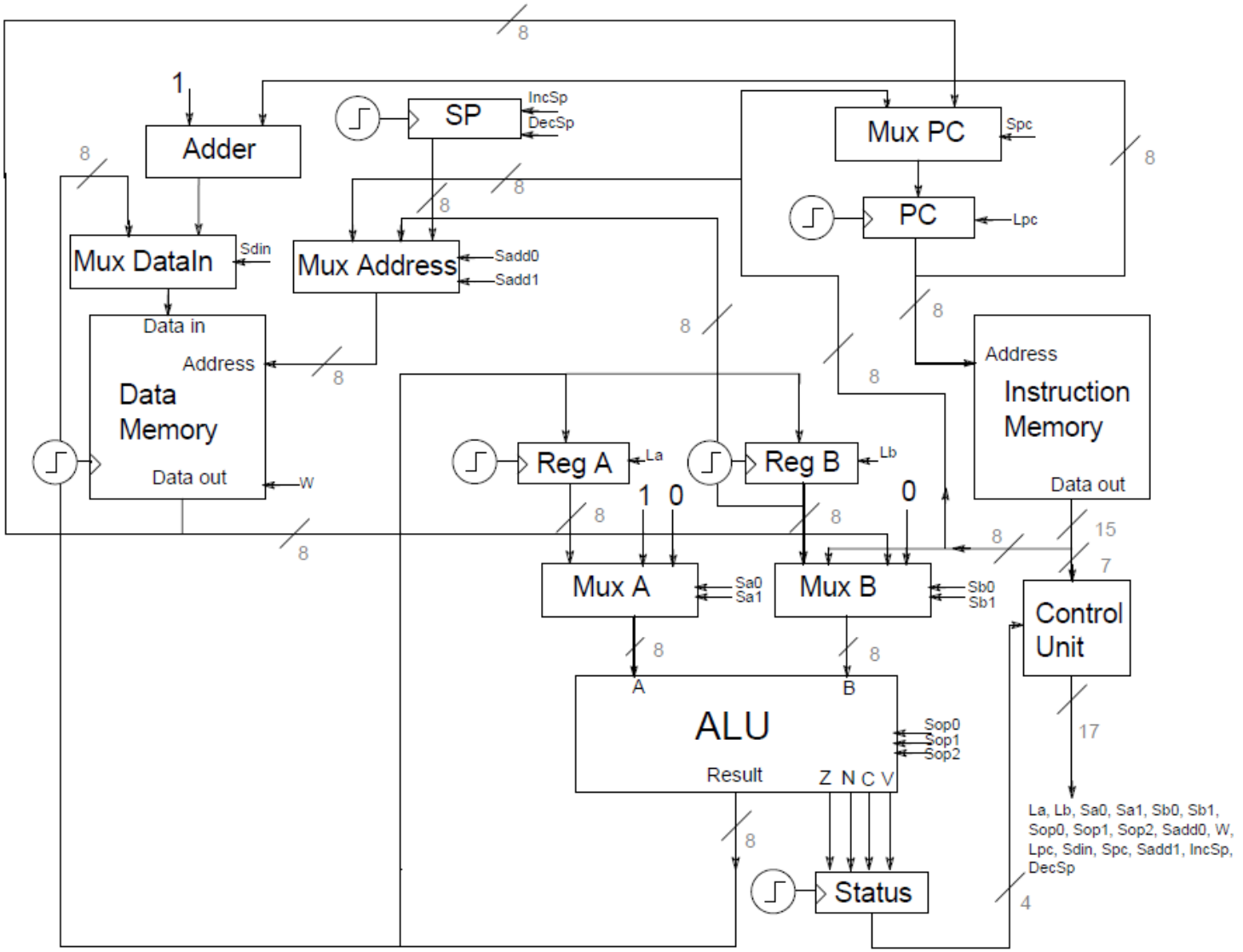
# ¿Cuál es el flujo de la instrucción MOV (B), A en el computador básico?

1. Lectura de instrucción desde memoria (Fetch)
2. Decodificar instr. en unidad de control (Decode)
3. Acceder a memoria (Mem)

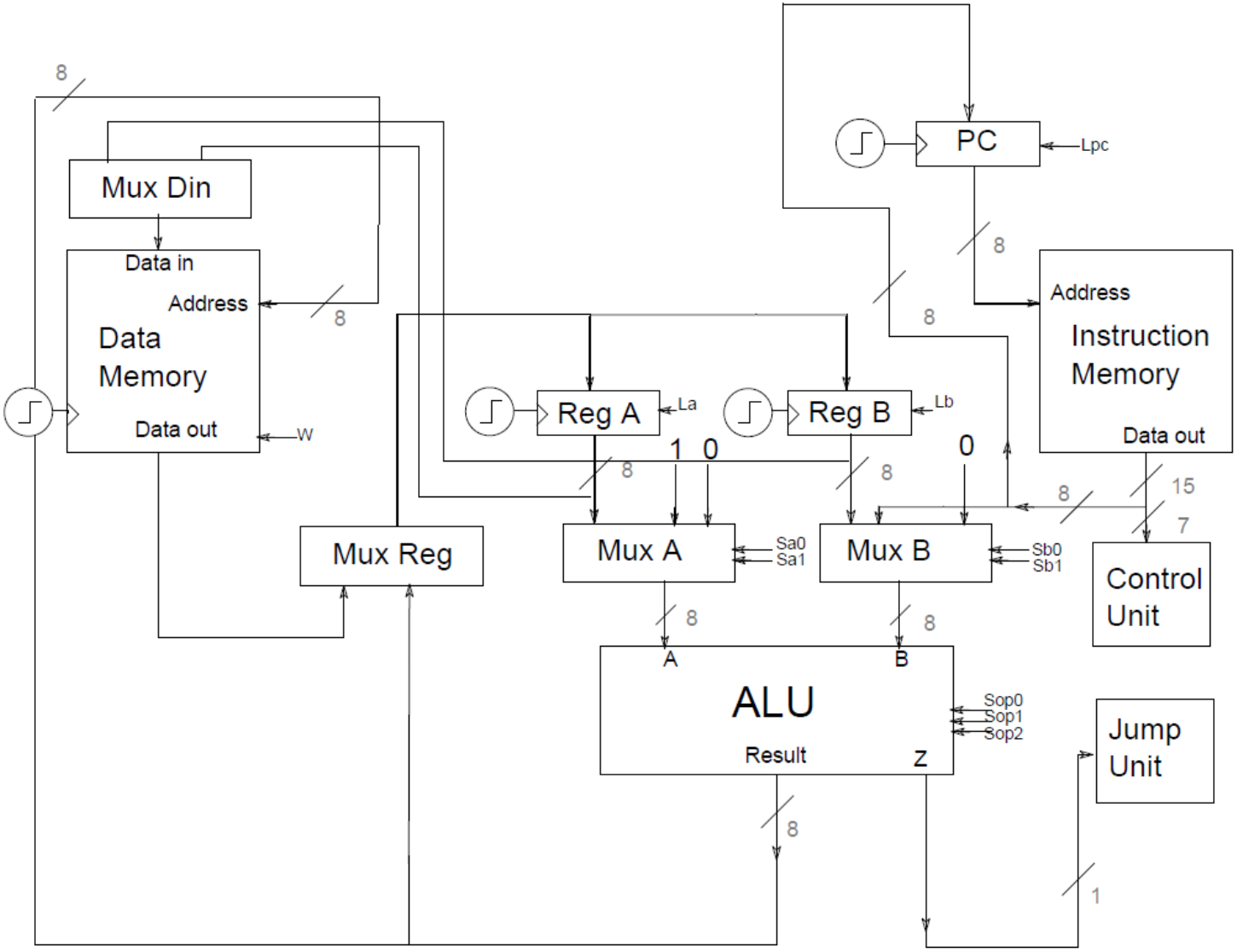
# Utilizaremos la siguiente secuencia de pasos

1. Lectura de instrucción desde memoria (Fetch)
2. Decodificar instr. en unidad de control (Decode)
3. Ejecutar en ALU (Execute)
4. Acceder a memoria (Mem)
5. Escribir resultado en registro (Write Back)

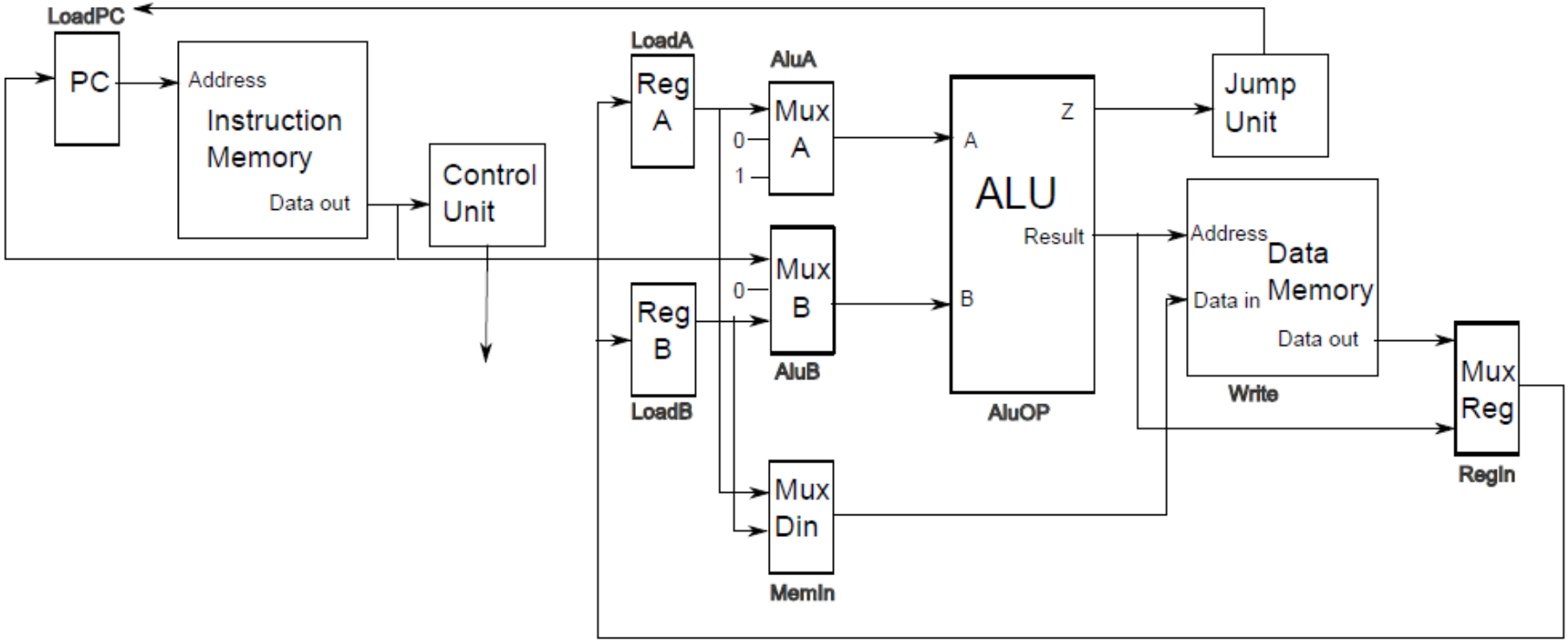
# Revisemos nuevamente el computador básico



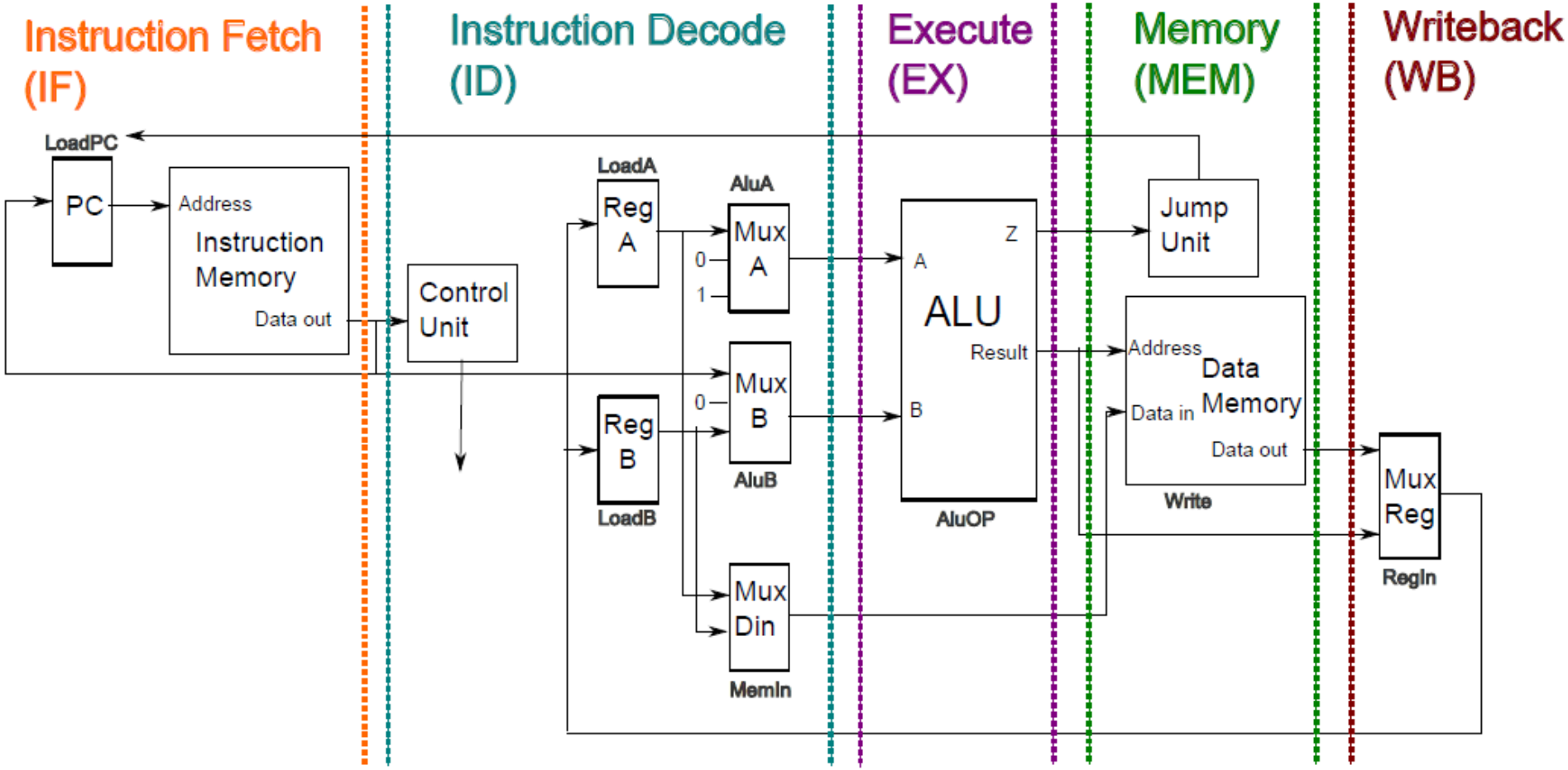
# Simplifiquemos la estructura para alinearla con el nuevo flujo de las instrucciones



# Simplifiquemos la estructura para alinearla con el nuevo flujo de las instrucciones

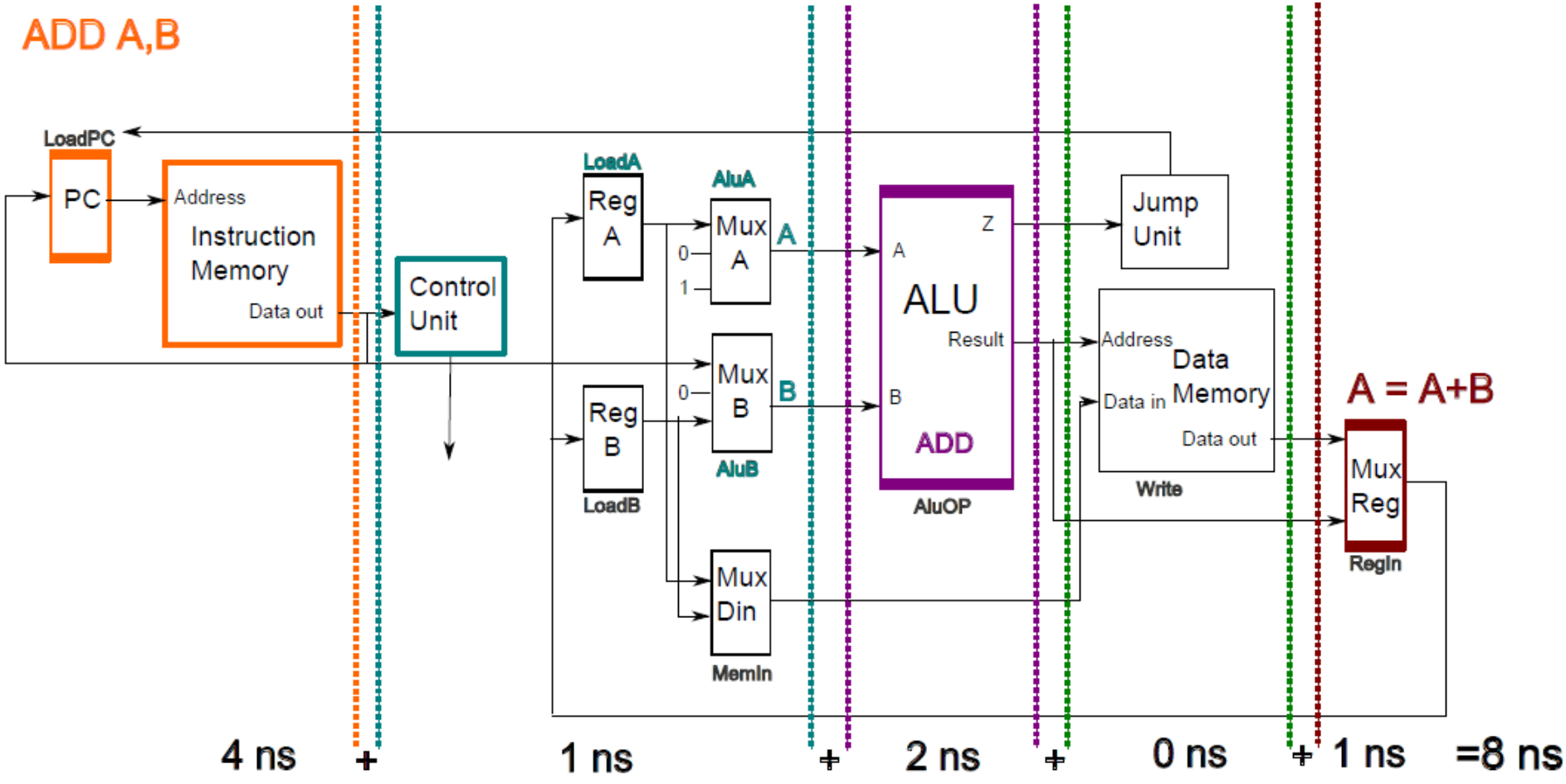


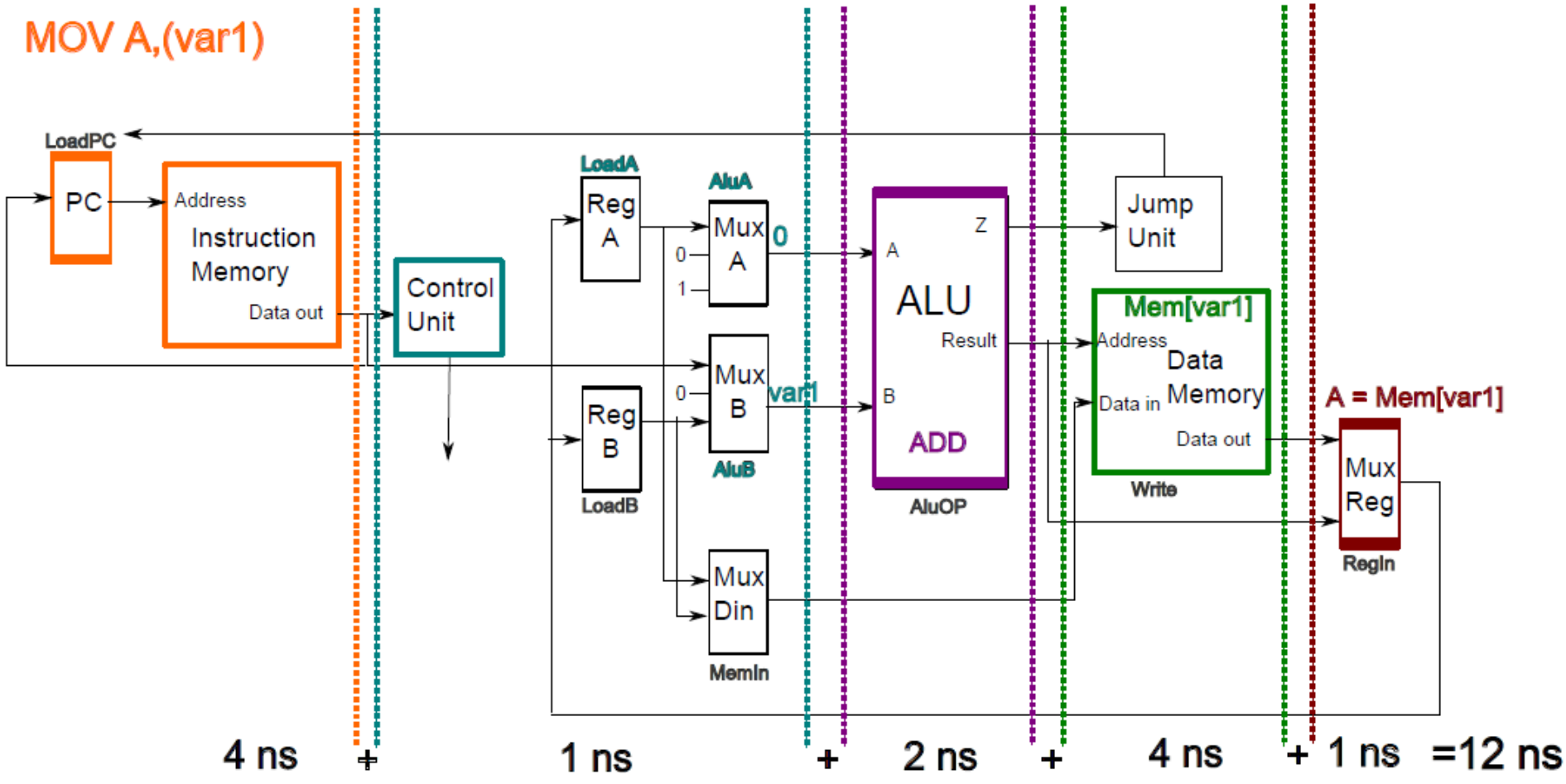
La nueva disposición nos permite dividir el ciclo de cualquier instrucción en 5 partes independientes



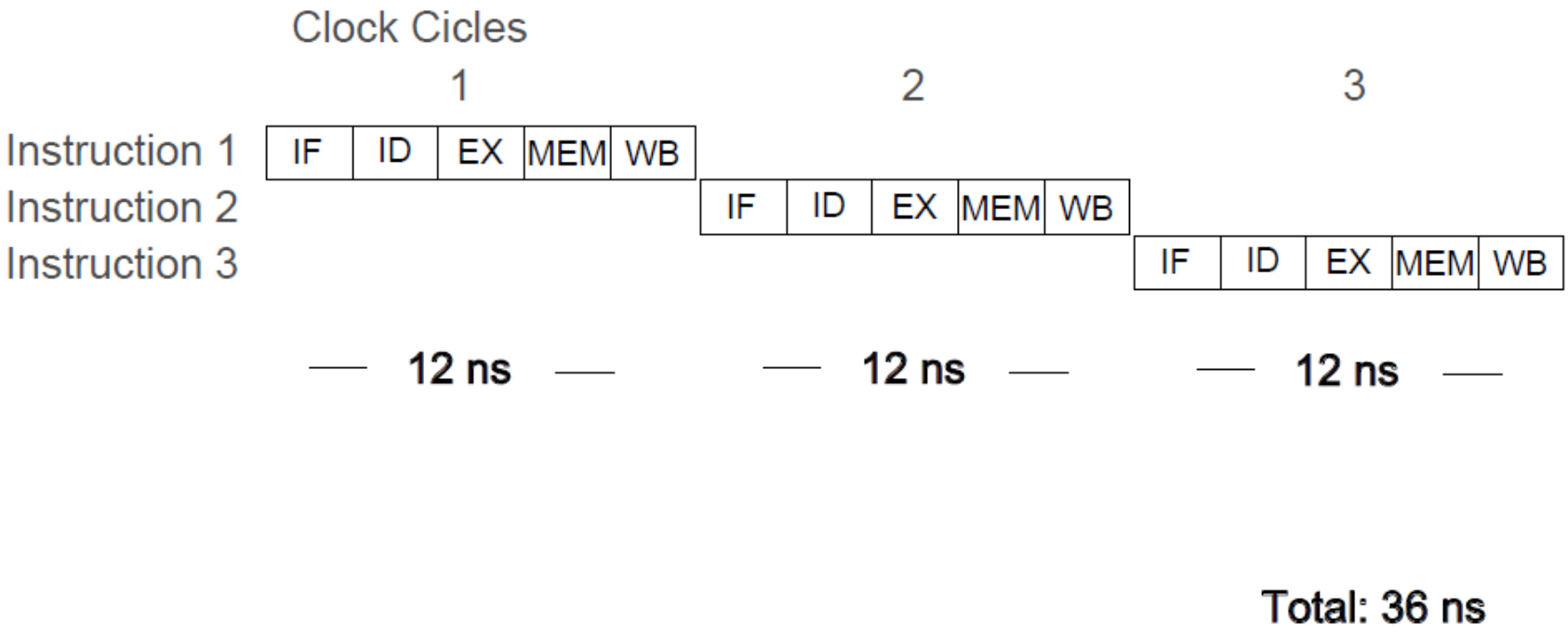
# Pipeline es paralelizable, tal como una línea de producción

- Los procesos secuenciales con etapas independientes son conocidos como pipeline.
- El diagrama con la nueva disposición del computador, nos permite ver el ciclo de una instrucción como un pipeline
- Esto permitiría procesar eventualmente múltiples instrucciones en paralelo (¿cuántas?), aprovechando las unidades libres.

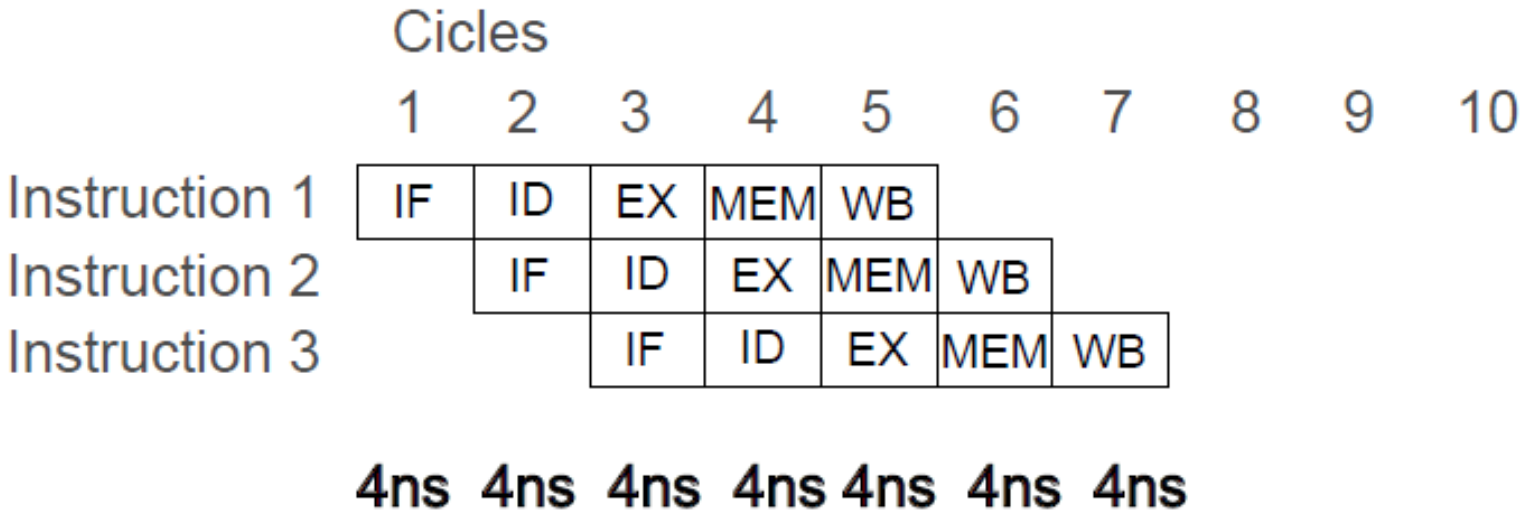

ADD A,B
LoadPC
PC
Address
Instruction Memory
Data out
Control Unit
LoadA
Reg A
AluA
Mux A
A
0
1
Reg B
LoadB
Mux B
B
AluB
Mux Din
MemIn
A
Z
ALU
Result
B
ADD
AluOP
Jump Unit
Address
Data in
Data Memory
Data out
Write
A = A+B
Mux Reg
RegIn
4 ns
+
1 ns
+
2 ns
+
0 ns
+
1 ns
=8 ns


MOV A,(var1)
LoadPC
PC
Address
Instruction Memory
Data out
Control Unit
LoadA
Reg A
AluA
Mux A
0
1
0
Reg B
LoadB
Mux B
0
var1
AluB
Mux Din
MemIn
A
Z
ALU
Result
B
ADD
AluOP
Jump Unit
Mem[var1]
Address
Data in
Data Memory
Data out
Write
A = Mem[var1]
Mux Reg
RegIn
4 ns
+
1 ns
+
2 ns
+
4 ns
+
1 ns
=12 ns

# Clock no puede ser más rápido que la instrucción más lenta (¿por qué?)



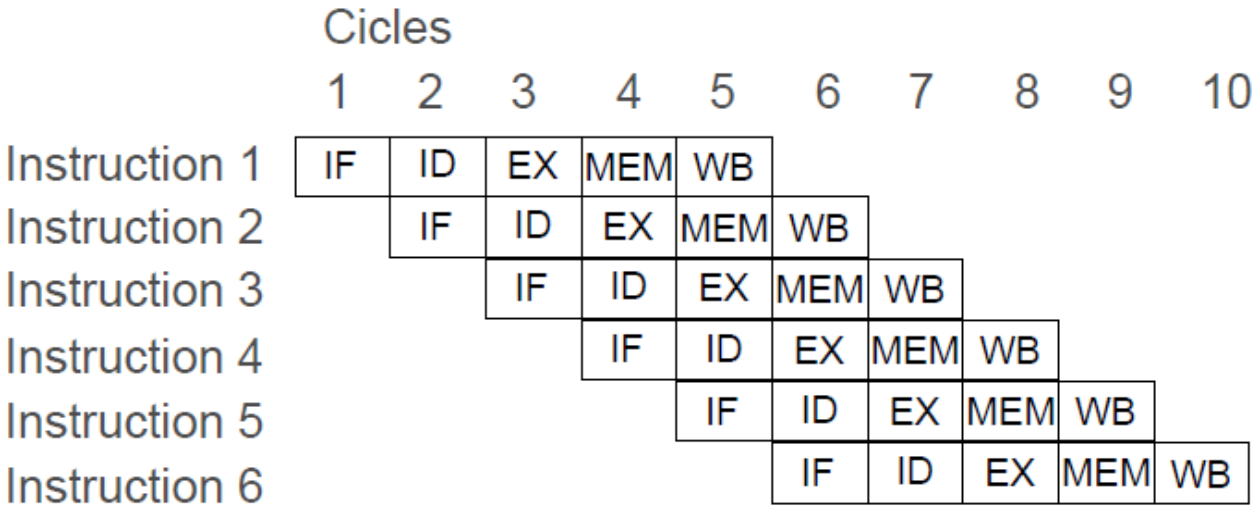
Con ILP, el clock ahora no puede ser más rápido que la etapa más lenta




¿Cómo es esto posible si ahora cada instrucción demora 20ns?

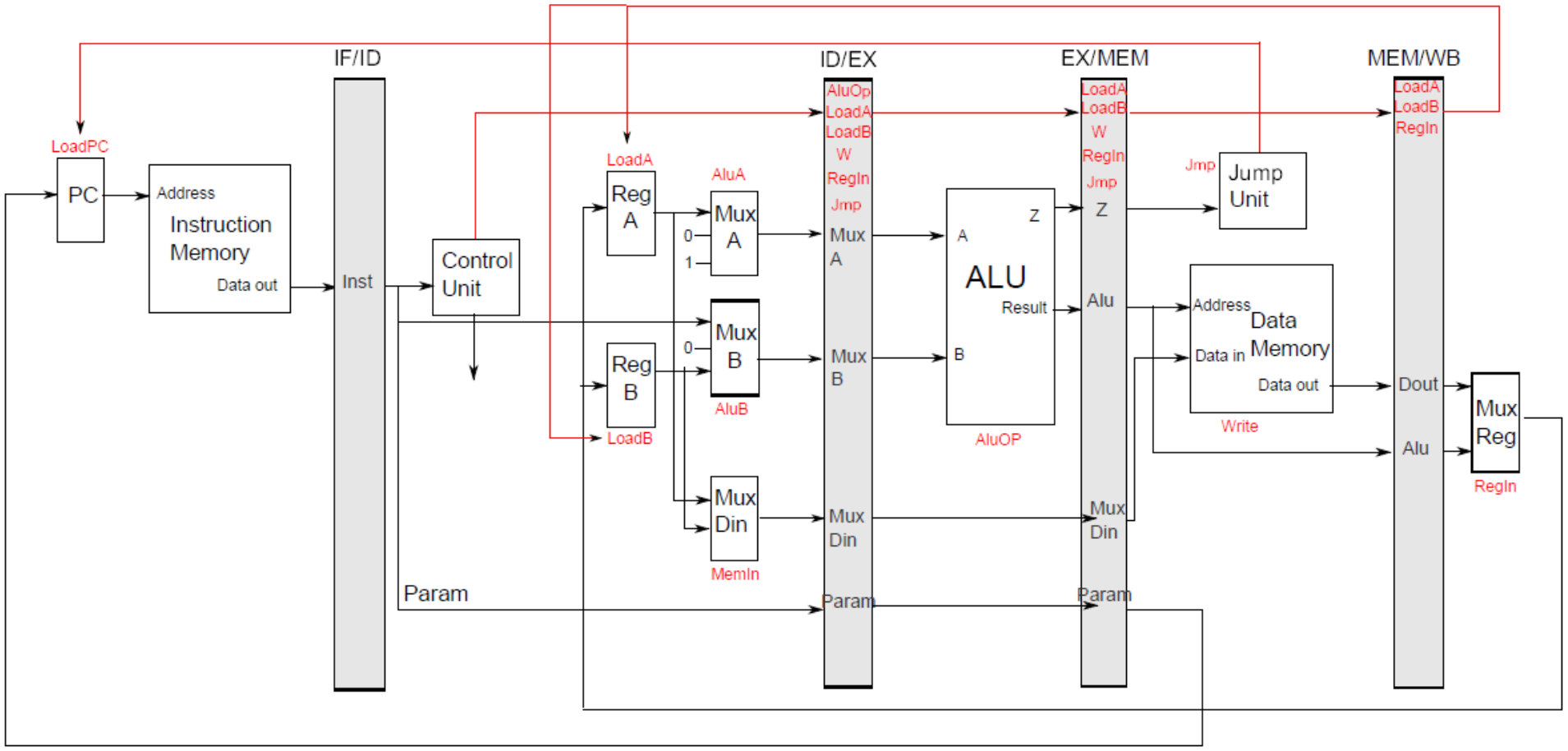
**Total: 28 ns**

# Mejora es ostensible si se aumenta el número de instrucciones




- 1MM instr. sin ILP = 12MM ns
- 1MM instr. con ILP ≈ 4MM ns

# Pipeline necesita soporte de HW para almacenar y propagar resultados



# Pipeline requiere mecanismos para asegurar consistencia

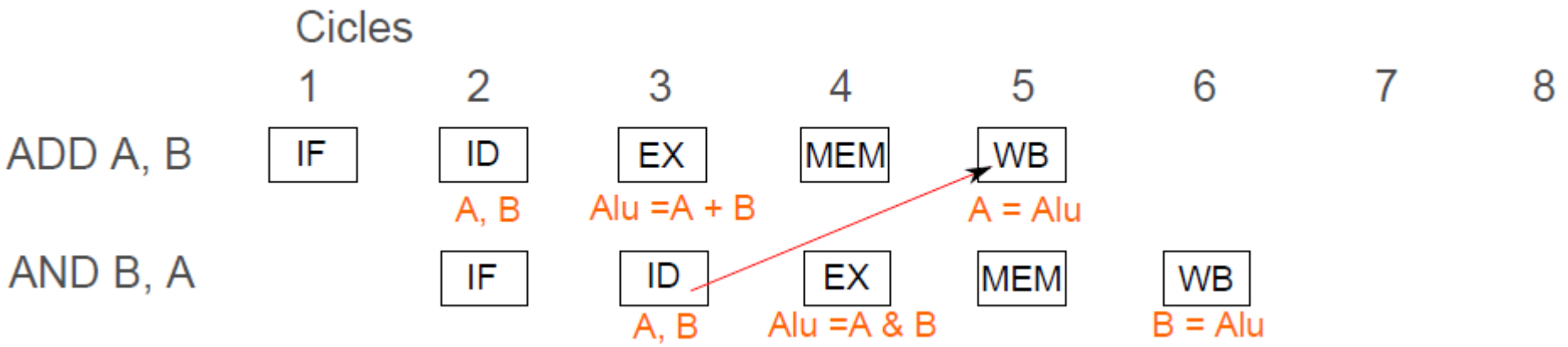
- El uso de un pipeline puede generar problemas con las dependencias entre instrucciones:

  ADD A, B

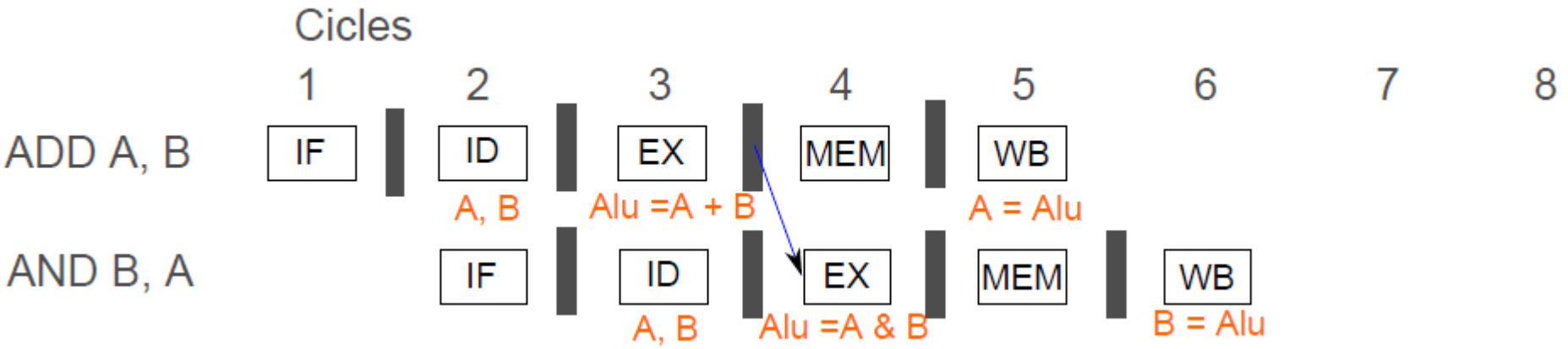
  AND B, A

- Estas situaciones son llamadas *hazards*. Existen 3 tipos: datos, control y estructurales.

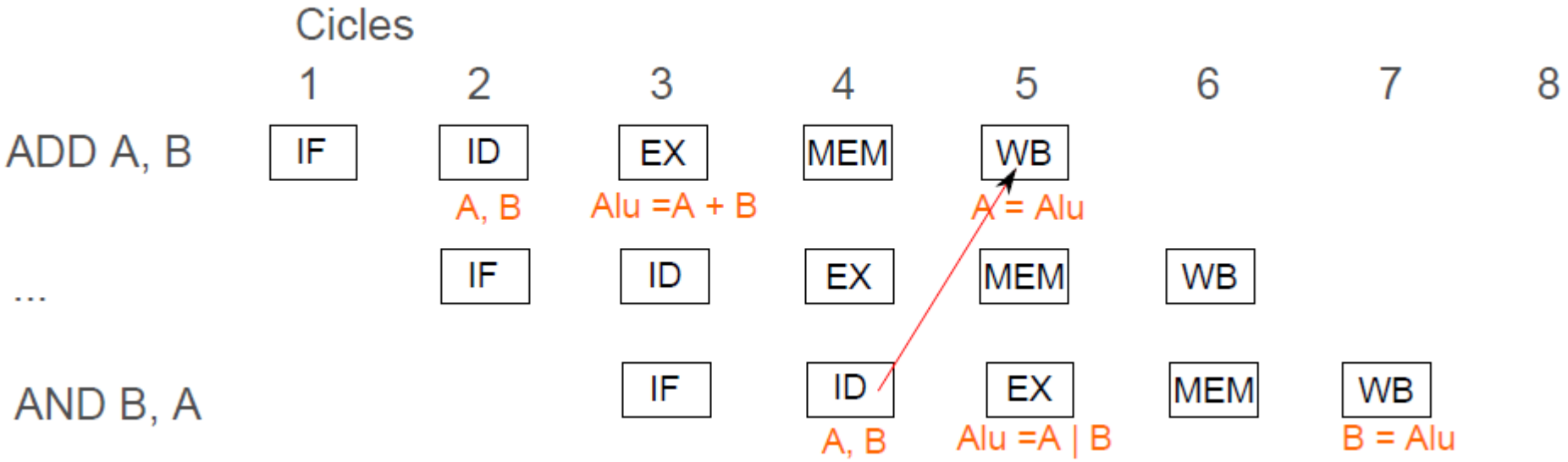
# La mayoría de los *hazards* de datos tienen solución simple



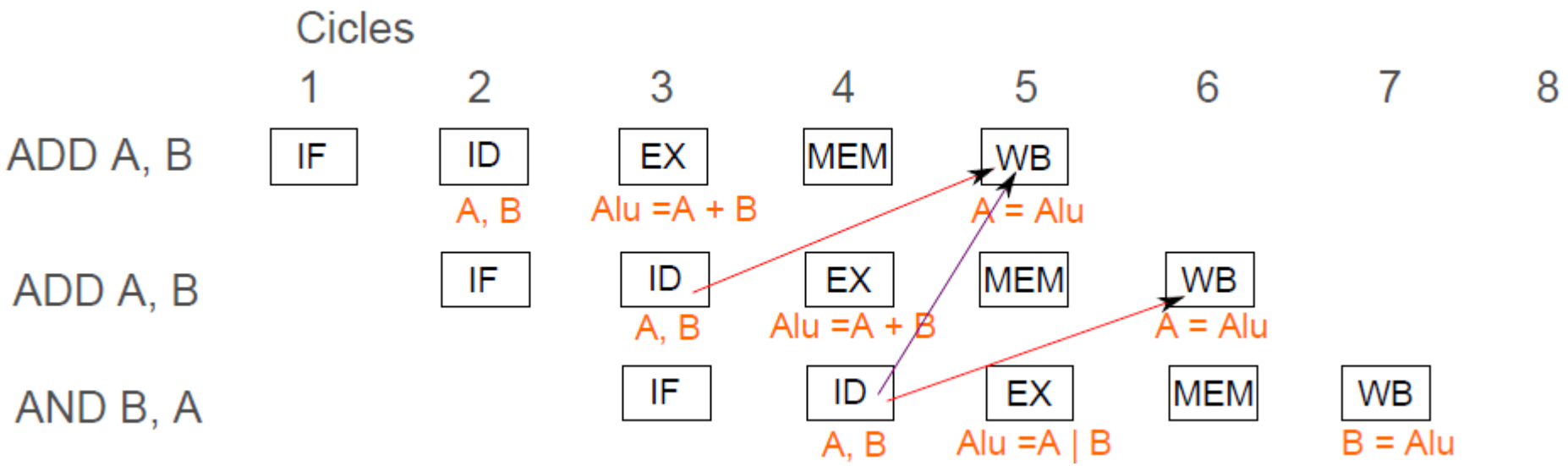
La mayoría de los *hazards* de datos tienen solución simple



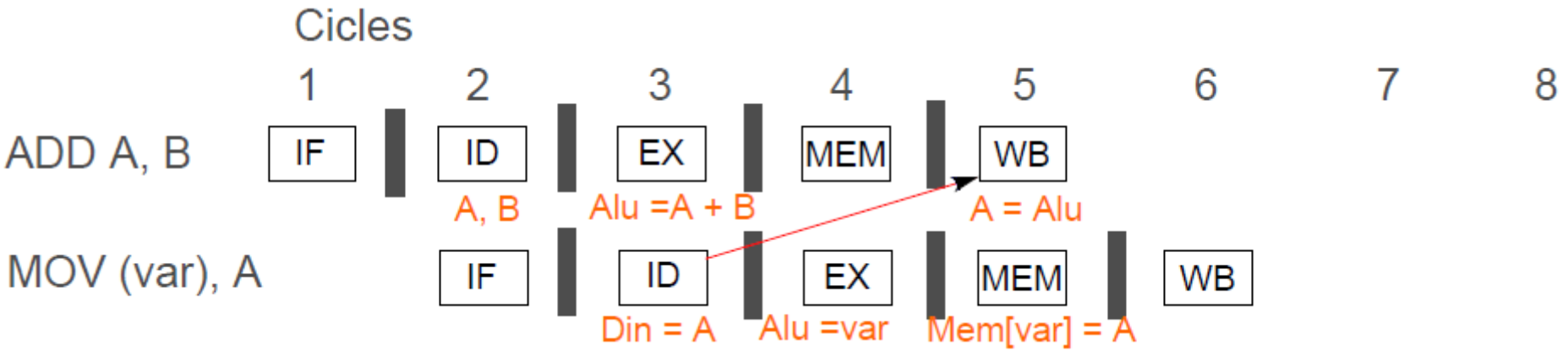
Existen más casos que requieren análisis



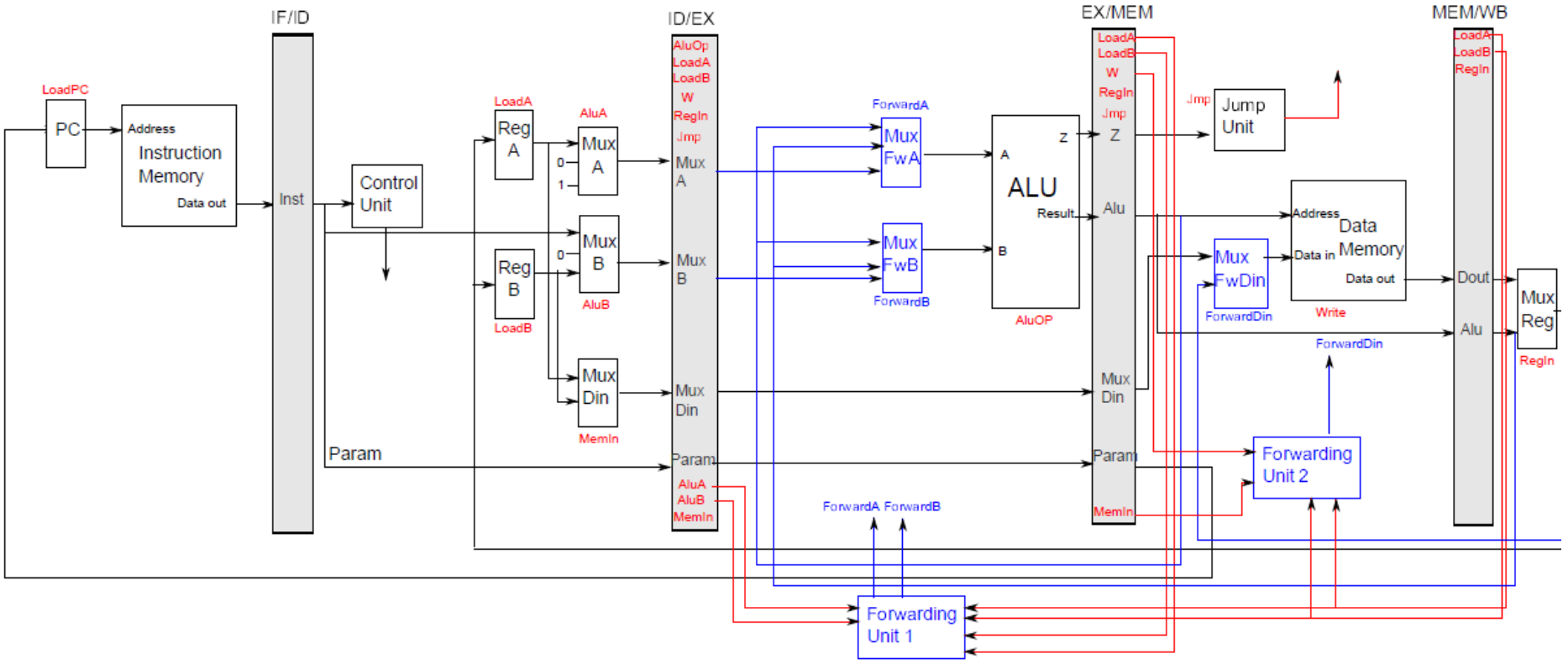
Existen más casos que requieren análisis



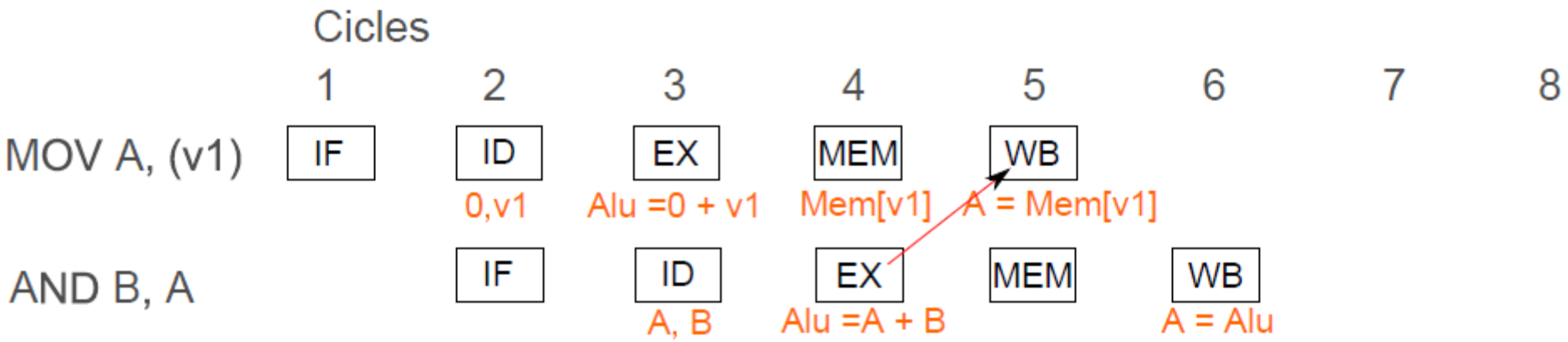
# Veamos un último caso



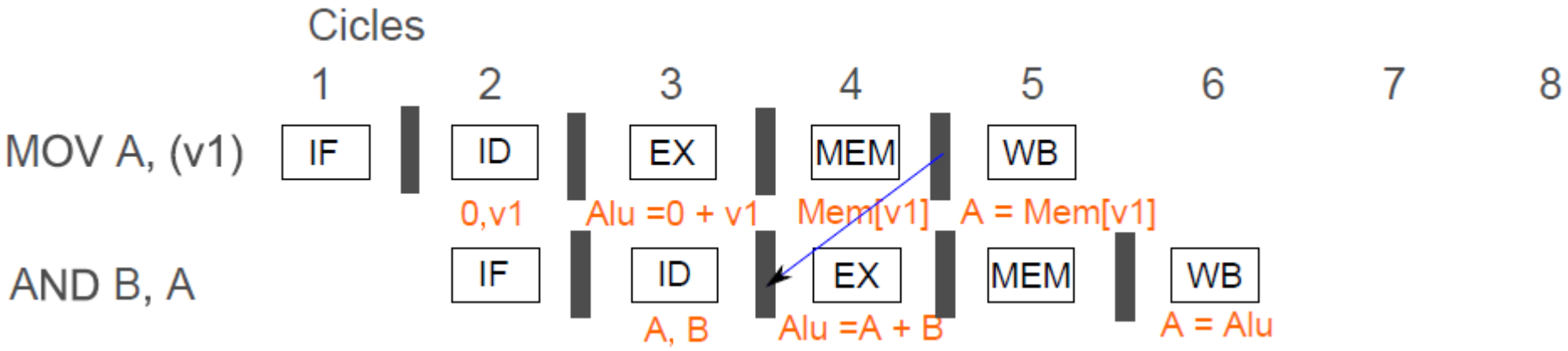
# Dos forwarding units se encargan de solucionar estos data *hazads*



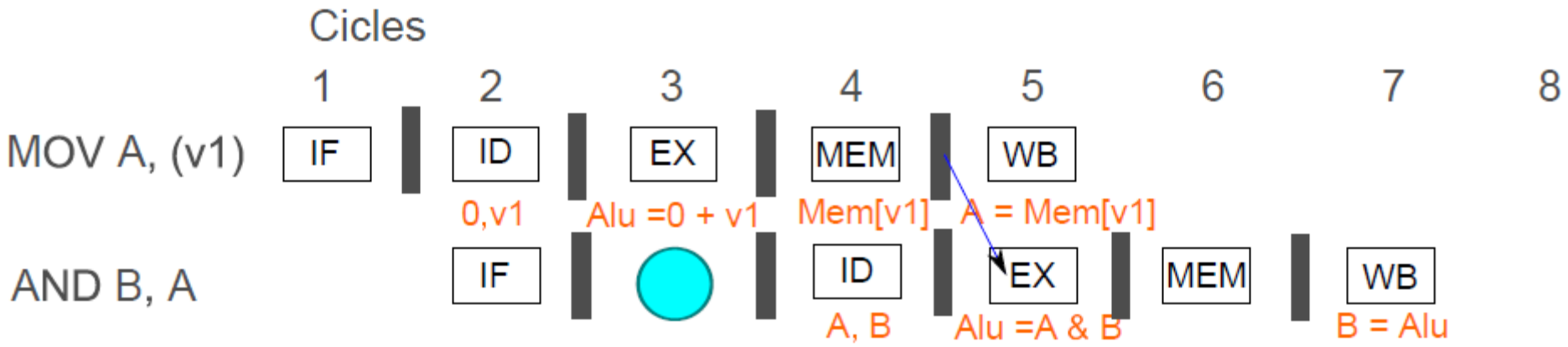
# ¿Qué hacemos en este caso?



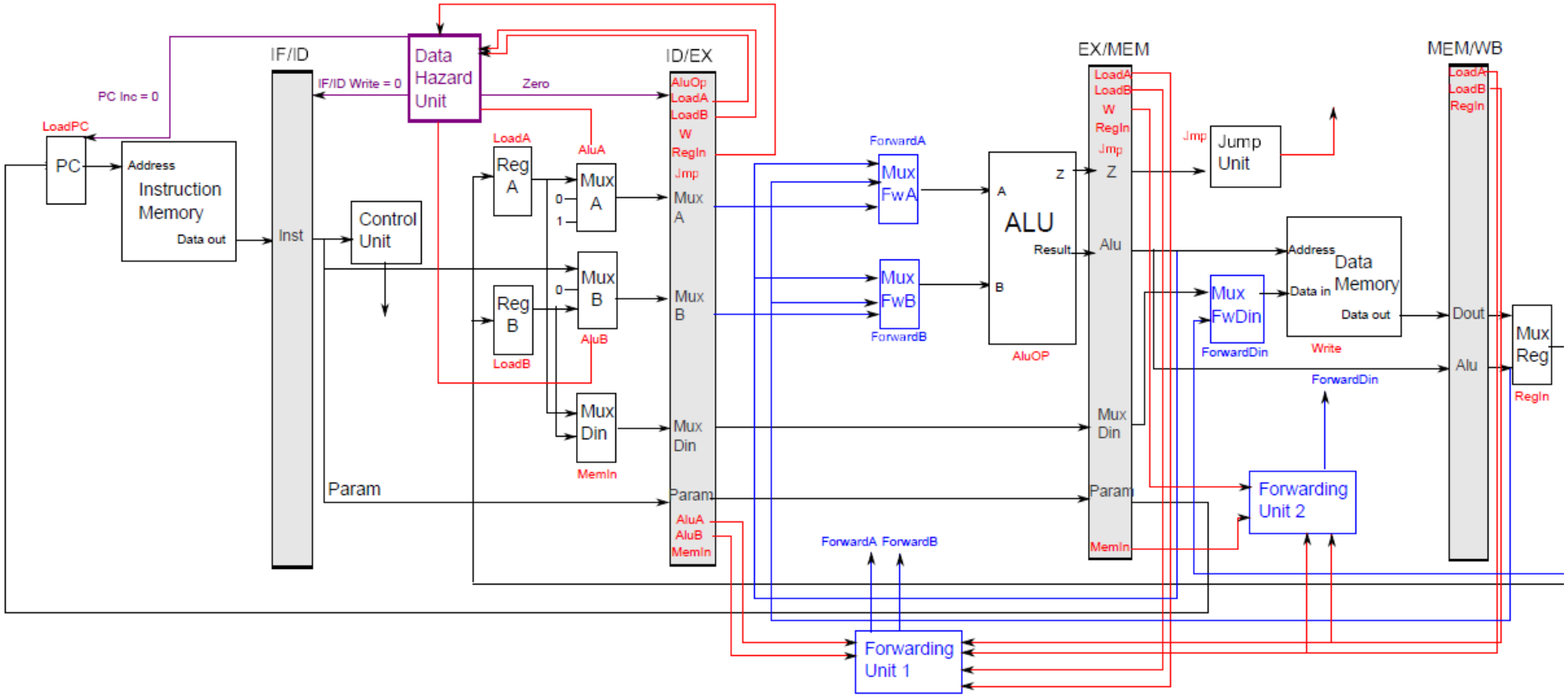
# Forwarding llega muy tarde, no es una solución factible



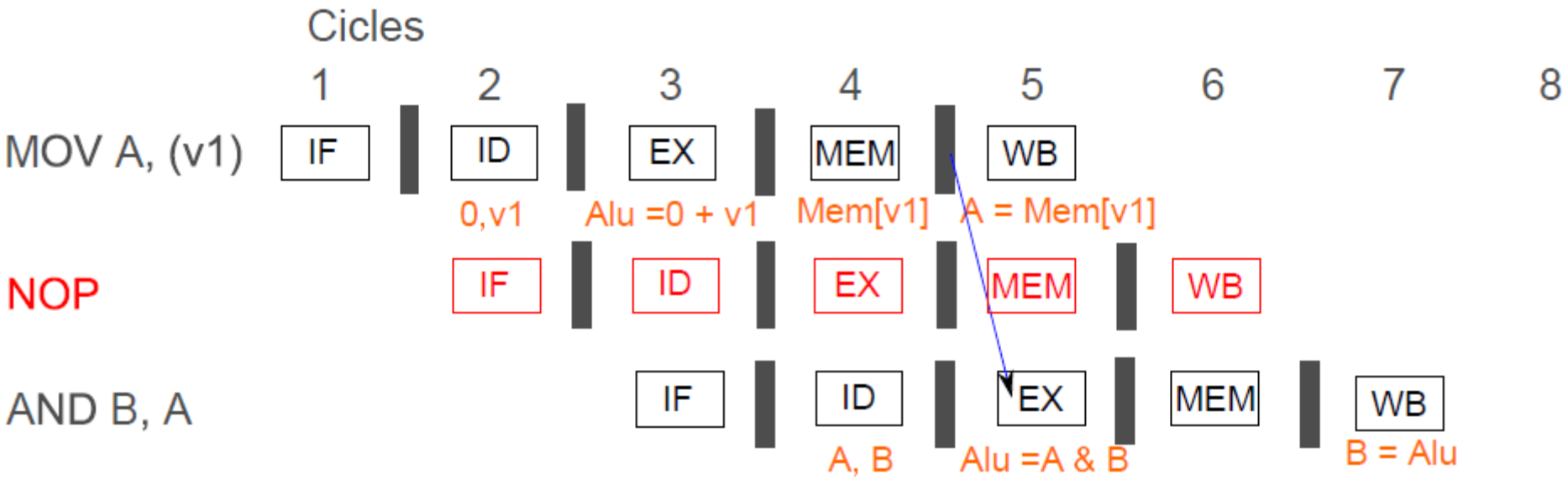
Necesitamos que el procesador espere,
lo que se conoce como *stalling*



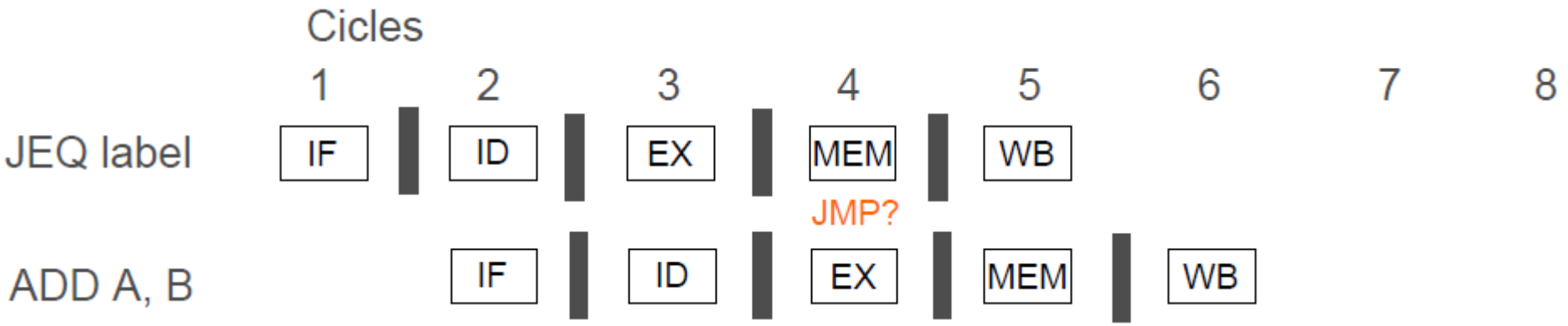
# Agregamos una unidad de detección de *hazards* en la etapa ID.



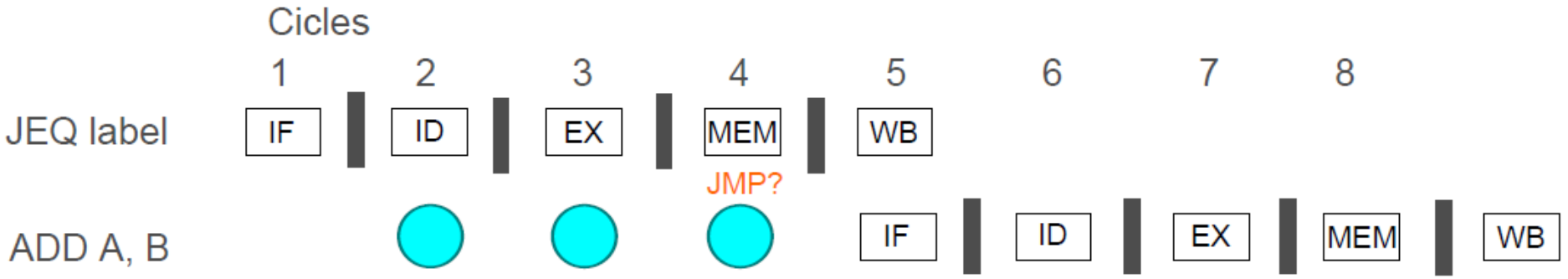
Otra opción es insertar una  instrucción NOP, que no hace nada, entre instrucciones



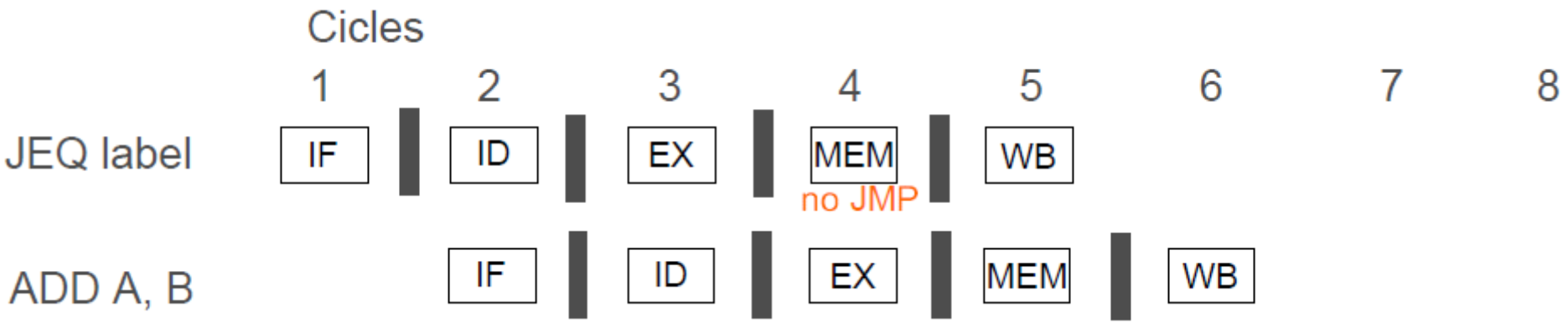
# *Hazards* de control ocurren cuando existen saltos



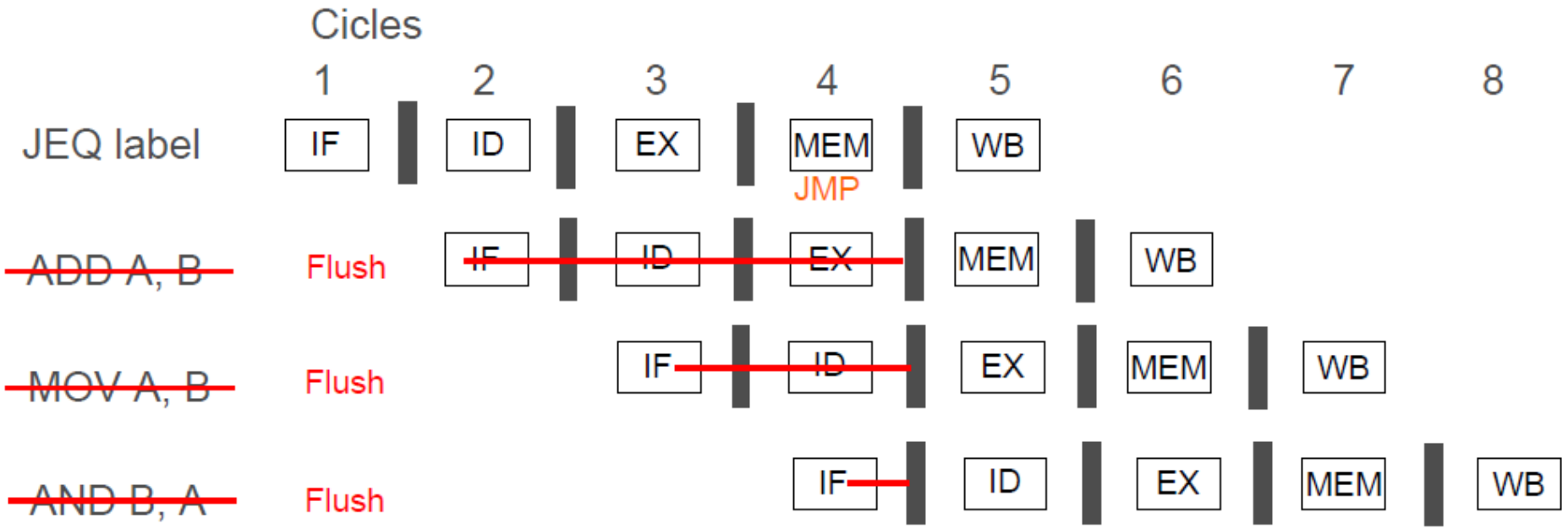
Una opción simple es hacer *stalling* del pipeline los ciclos que sean necesarios



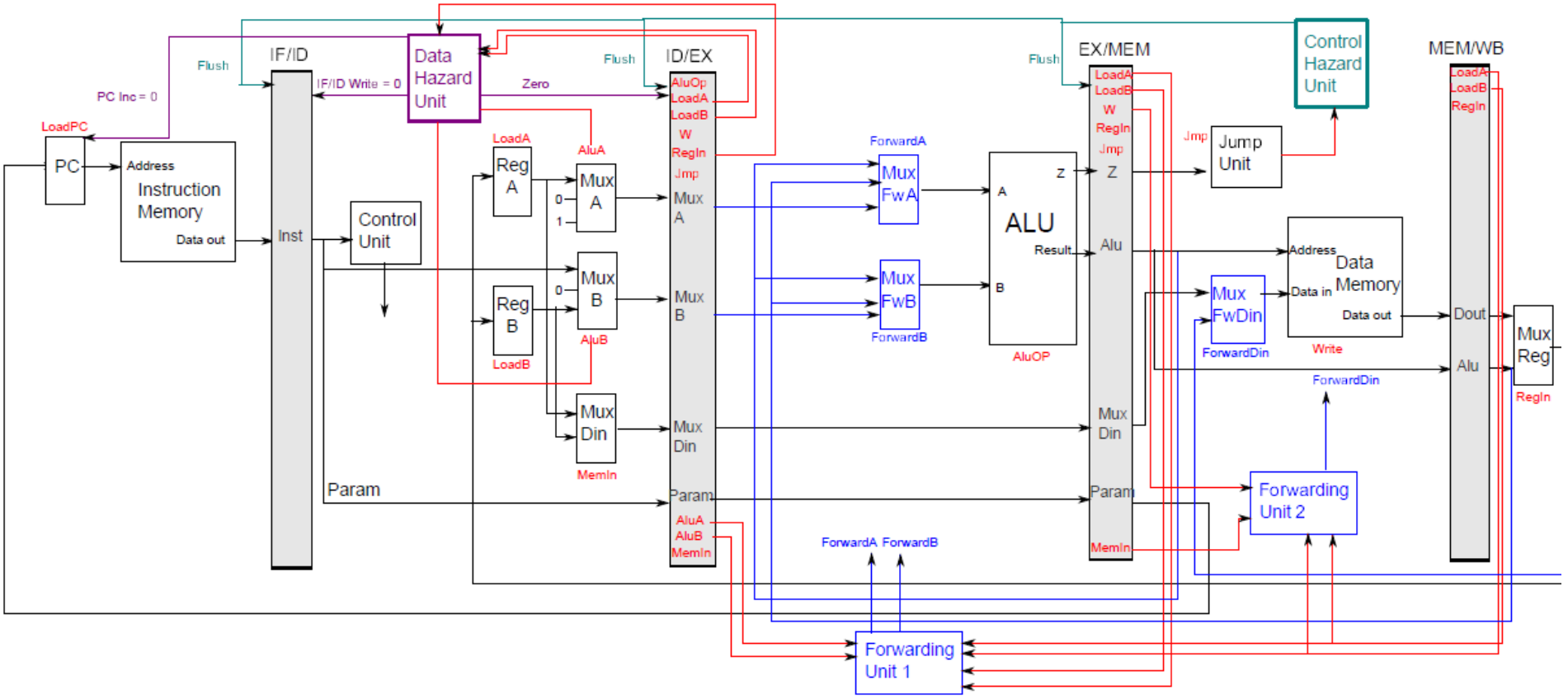
Otra opción es utilizar una predicción de saltos



# Otra opción es utilizar una predicción de saltos



# Esta solución requiere agregar una nueva unidad a la CPU



# *Hazards* estructurales ocurren cuando dos etapas necesitan usar la misma unidad funcional

- Un ejemplo ocurre en un computador Von Neumann, donde la memoria única de instrucciones y datos es accedida tanto en la etapa IF como en la etapa MEM.

- Una opción es agregar burbujas, que hagan que una de las instrucciones que quería ocupar la unidad en un cierto ciclo espere hasta el siguiente.

## *Hazards* estructurales ocurren cuando dos etapas necesitan usar la misma unidad funcional

- Otra solución puede ser agregar unidades funcionales extra.
- ¿Cómo solucionamos fácilmente el hazard estructural de acceso a memoria en una arquitectura Von Neumann?
- En el caso de las memorias, tener una caché split de primer nivel transforma la arquitectura del computador en Harvard y se soluciona el problema.

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación


# IIC2343 – Arquitectura de Computadores

## Paralelismo a nivel de instrucción (ILP)

**Profesor:** Jurgen Heysen