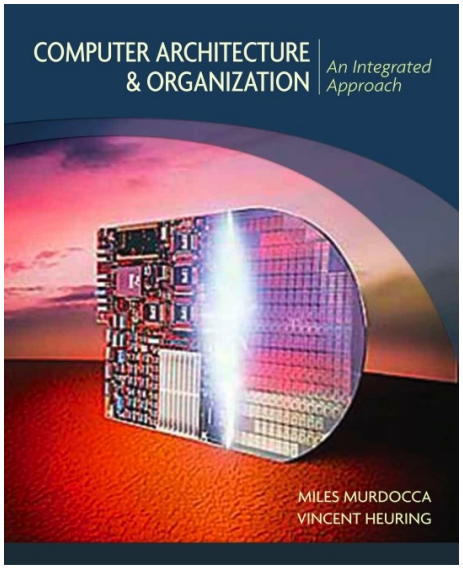
Organización de las Computadoras

Leonardo Giovanini



Microarquitectura segmentada

Contenidos

5.1 Arquitectura frente a implementación

5.2 Microarquitectura de la CPU - Implementación pipeline

5.3 Riesgos de datos

5.4 Riesgos de control

Arquitectura frente a implementación

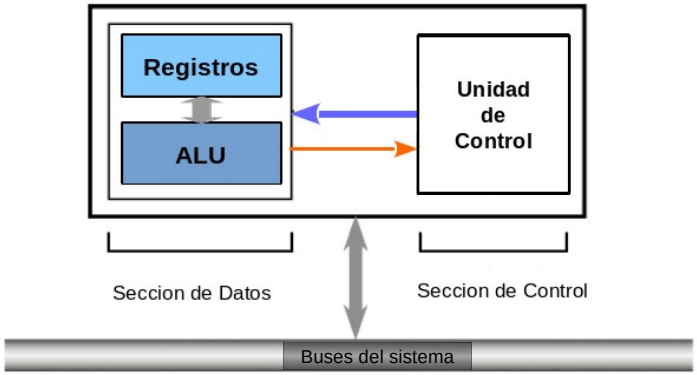
Arquitectura frente a implementación

Microarquitectura del procesador

La microarquitectura **describe cómo se implementa el comportamiento descrito por el ISA** a través de un sistema digital. La microarquitectura define cómo **cada una de las señales digitales se enruta y manipula**.

La microarquitectura se **representa como diagramas de bloques** que describen las interconexiones de los diversos elementos de la microarquitectura.

Las unidades de ejecución (***unidades lógicas aritméticas, unidades de puntos flotantes, unidades de carga / almacenamiento, periféricos*** *y* ***predicción de salto***, entre otras) son esenciales para la definición de la microarquitectura, ya que realizan las operaciones del procesador.



La **elección del número de unidades de ejecución**, su **interconexión**, su **latencia** y **rendimiento** es una tarea central en el diseño de microarquitectura.

Las decisiones de **diseño de la microarquitectura** afectan directamente al **desempeño general del sistema**, prestando atención a temas tales como el costo, el consumo de energía, la conectividad y la capacidad de depuración.

Arquitectura frente a implementación

Microarquitectura del procesador

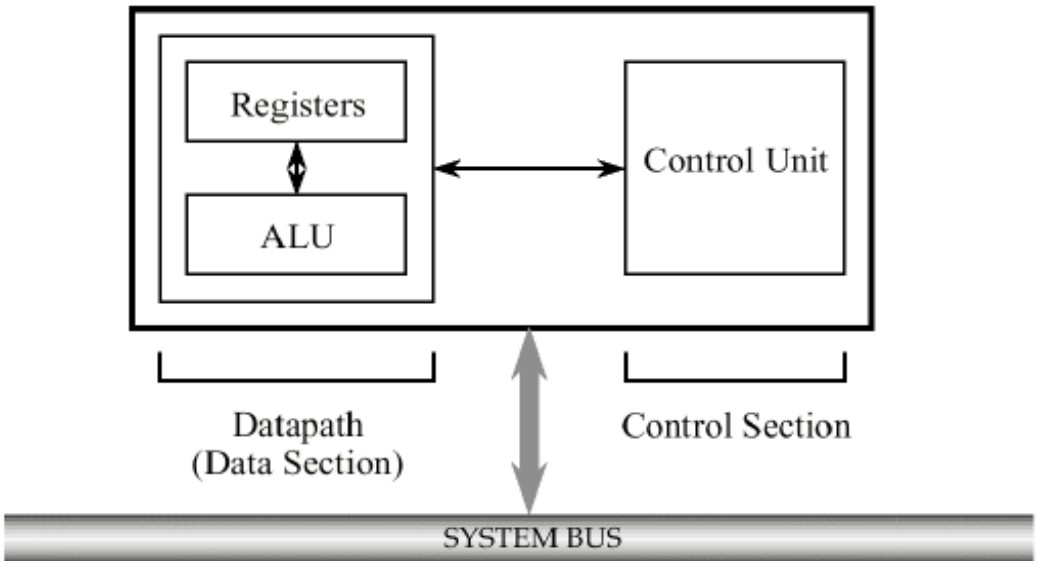
La CPU está compuesta por tres bloques:

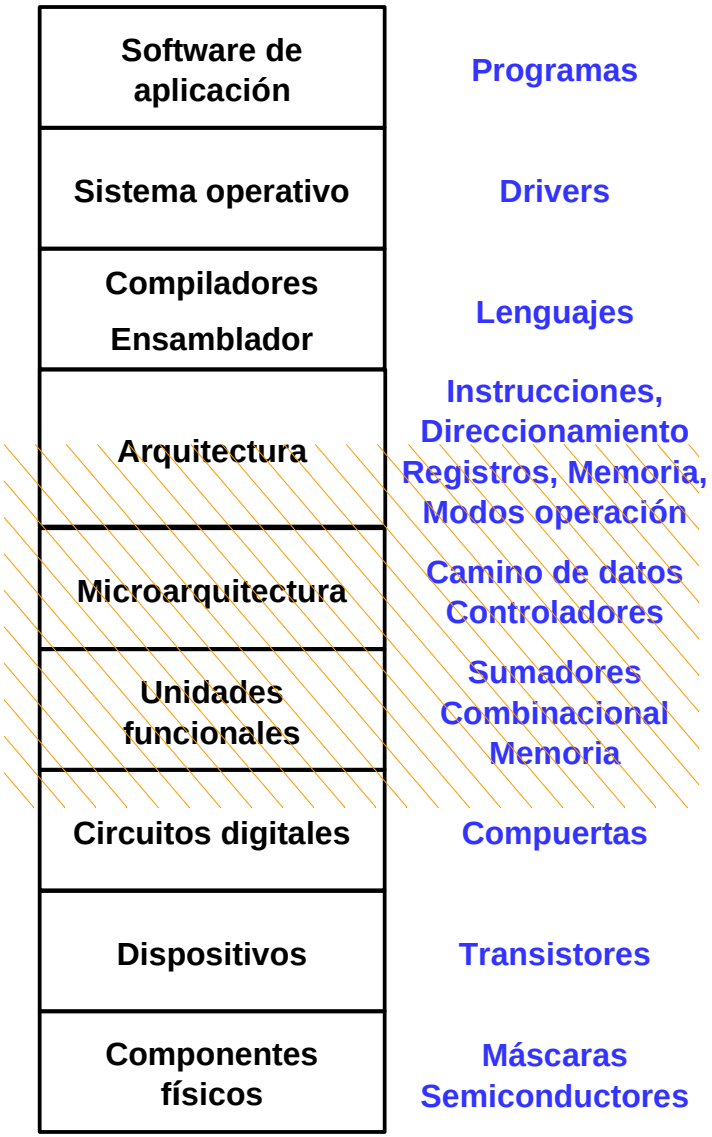
Un banco de registros donde se almacenan los datos e información;

Una ALU donde se procesan los datos; y

Una unidad de control que interpreta las instructiones y ejecuta las acciones necesarias para su ejecución.

La sección de datos (registros y ALU) es conocida como ***datapath***.





Arquitectura frente a implementación

Microarquitectura del procesador

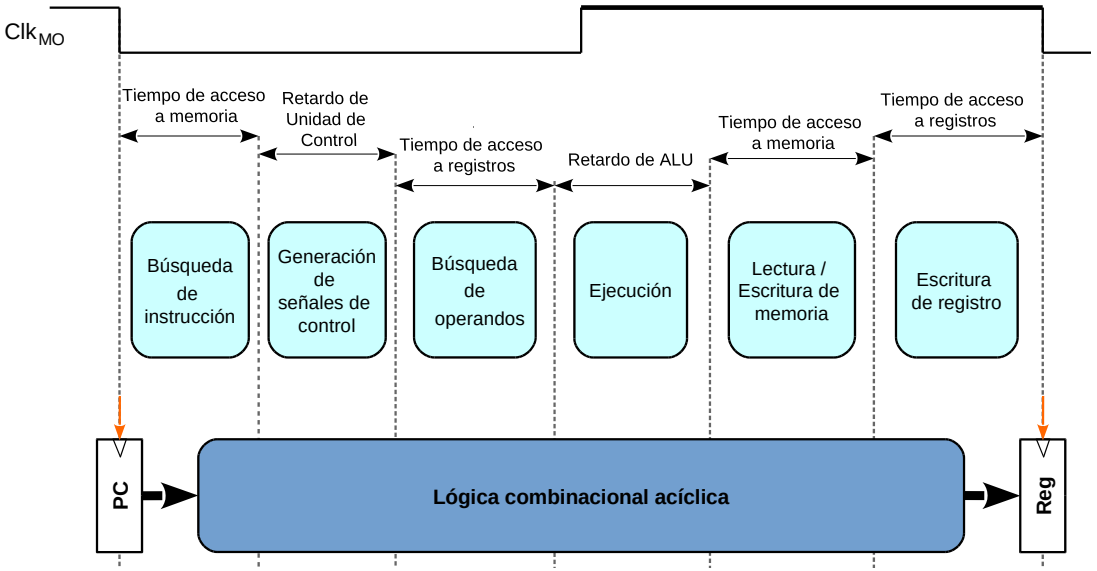
La microarquitectura más simple es la **implementación monociclo**.

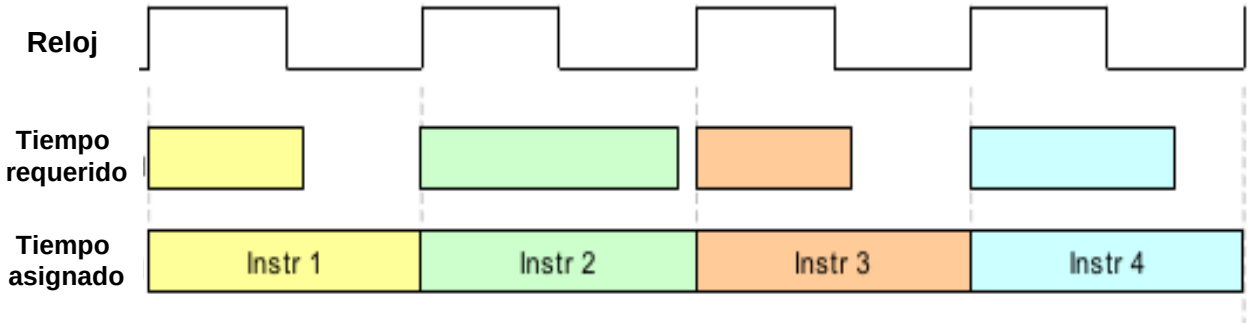
La unidad de control se implementa a través de un circuito combinacional.

Esto **simplifica su diseño e implementación circuital** (circuito digital muy simple).

Pero, **todas las instrucciones** se ejecutan en un **sólo ciclo de reloj**.

Por lo que el reloj debe tener un periodo igual tiempo de ejecución de la **instrucción más lenta**.





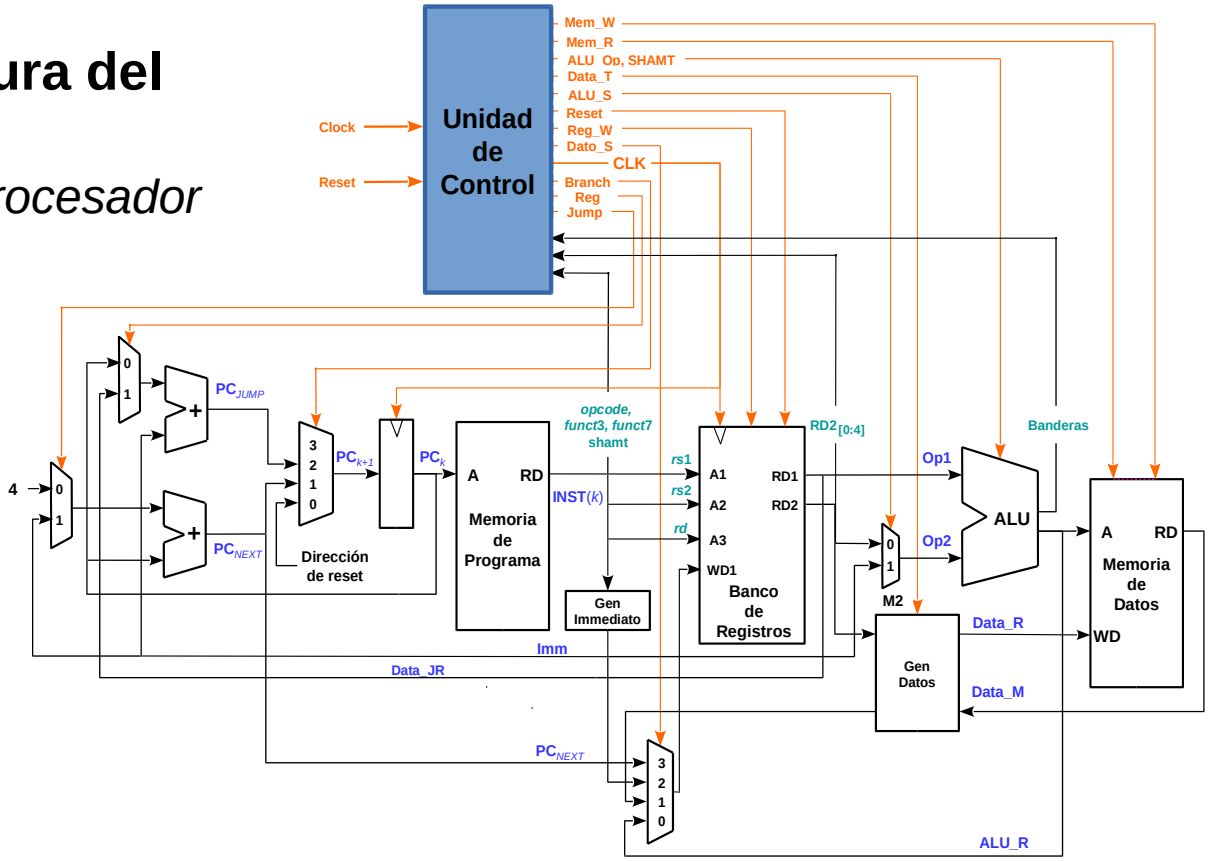
Por lo que las instrucciones rápidas desperdician tiempo de ejecución.

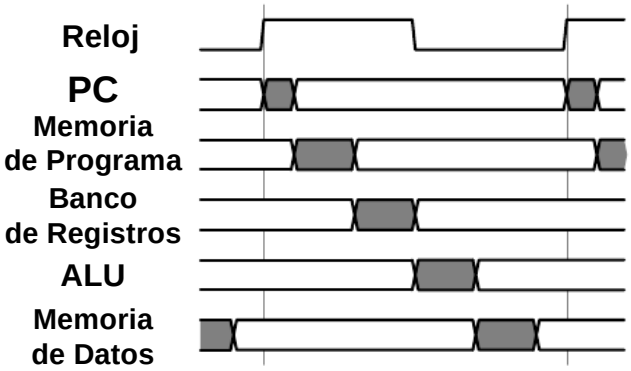
La microarquitectura del procesador

Microarquitectura del procesador

Las instrucciones **más lentas** son las instrucciones son las instrucciones de **carga** (*tipo I*) **y almacenamiento de datos** (*tipo S*).

Estas instrucciones involucran las todas las etapas de la ejecución de un instrucción, incluyendo el acceso a la memoria de datos.





Las instrucciones **más rápidas** son las instrucciones son las instrucciones de **salto** (*tipo S*) **y *operación con datos en registros*** (*tipo U*). Estas instrucciones sólo involucran las etapas de la decodificación y ejecución.

Con un periodo fijo, las instrucciones rápidas desaprovechan tiempo.

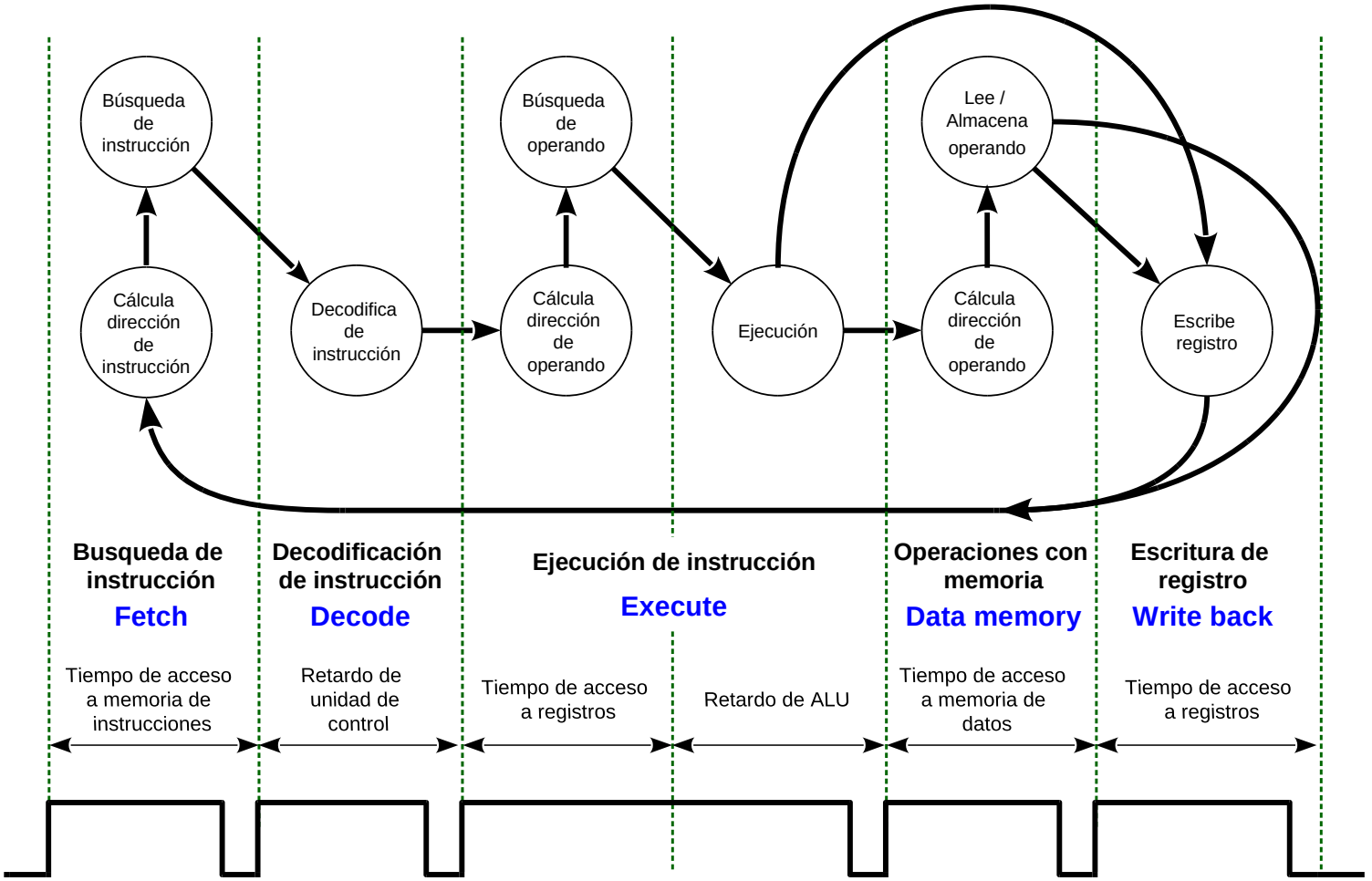
En los repertorios reales existen coexisten instrucciones lentas con muy rápidas.

Implementación segmentada

La microarquitectura del procesador

Implementación segmentada

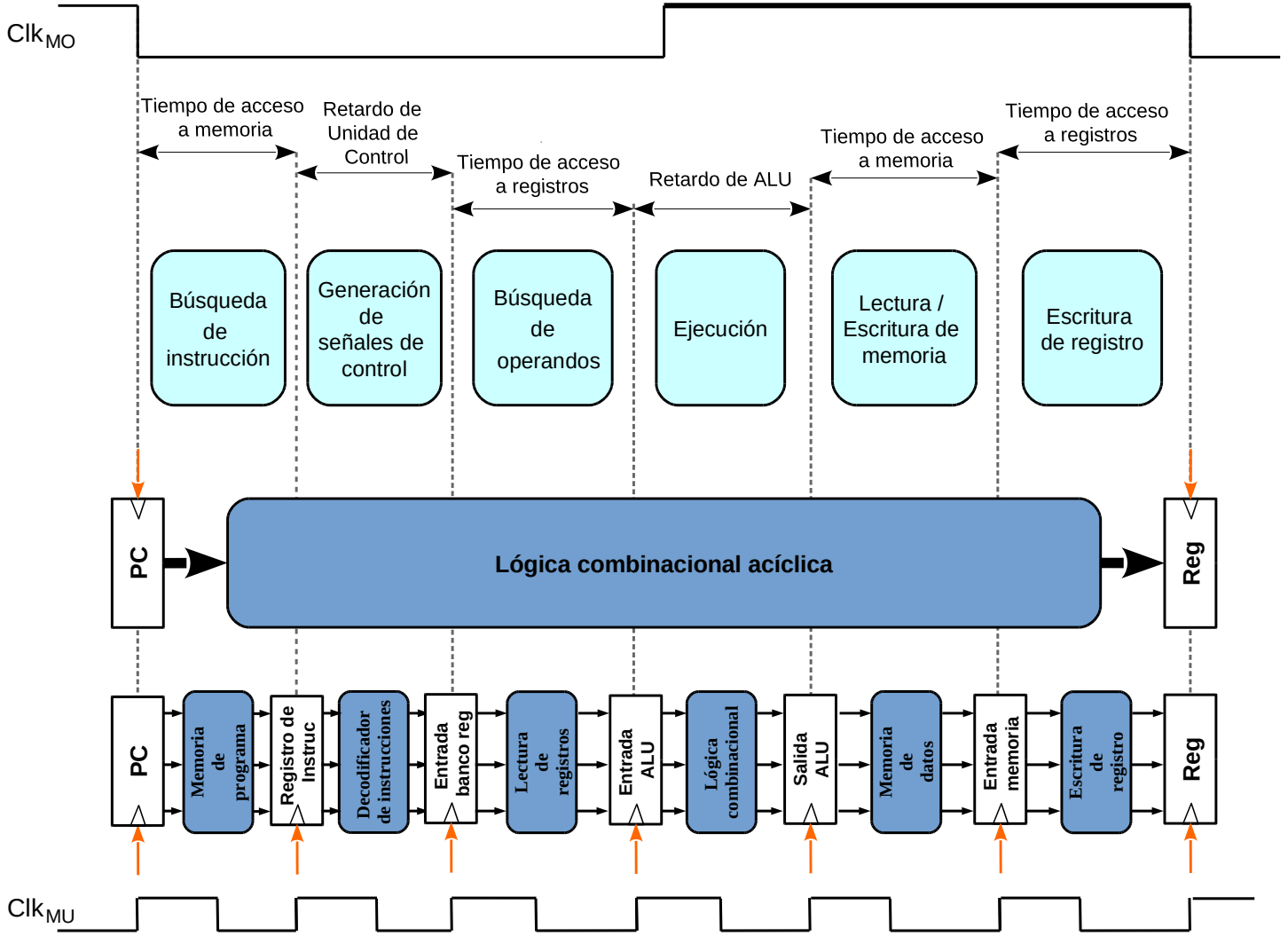
Para mejorar el desempeño del procesador se **divide la ejecución de la instrucción en etapas** más pequeñas, de modo **sólo utilicen las etapas necesarias** y el periodo del reloj esté determinado por **la etapa más lenta.**



La microarquitectura del procesador

Implementación segmentada

Esta idea requiere de elementos para almacenar valores generados por los diferentes bloques del procesador.

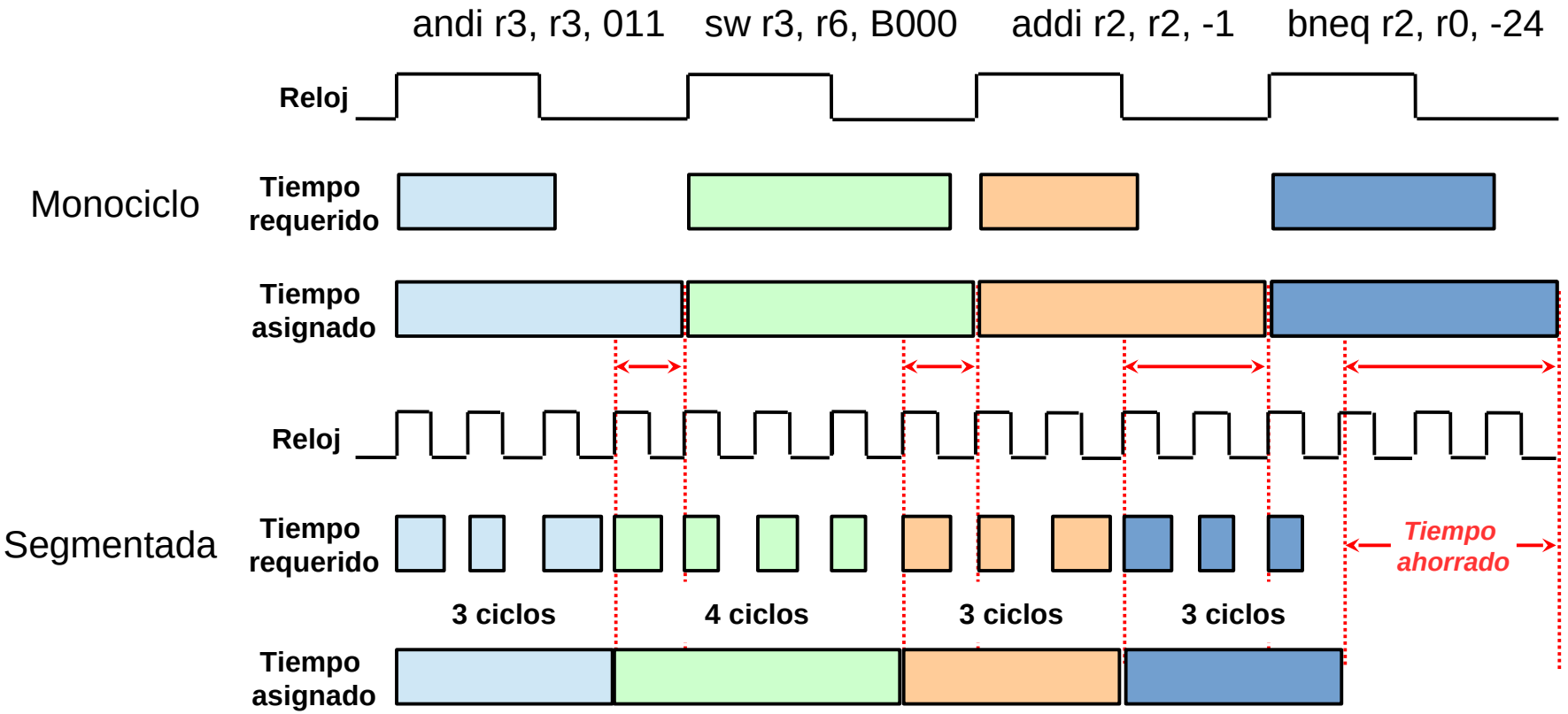


La microarquitectura del procesador

Implementación segmentada

De manera de aprovechar la diferencia de tiempos de ejecución de las instrucciones. Por ejemplo,

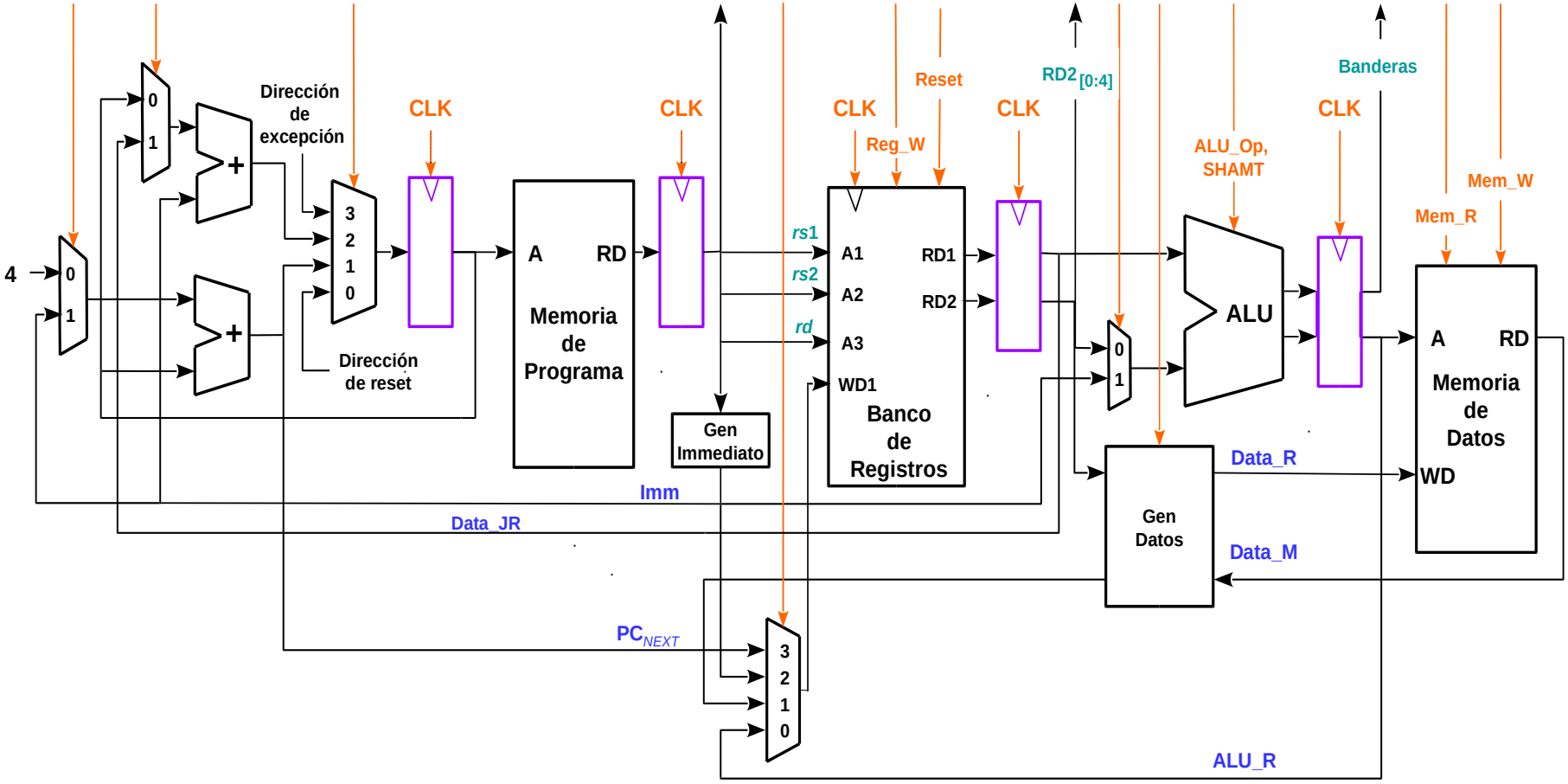
- Una instrucción **tipo R sólo utilizará 3 ciclos** (Fech, Decode y Execute),
- Una instrucción **tipo S utilizará 5 ciclos** (Fech, Decode, Execute, Data Memory y Writeback) y
- Una instrucción **tipo B sólo utilizará 3 ciclos** (Fech, Decode y Execute).



La microarquitectura del procesador

Implementación segmentada

Esta idea requiere de **elementos para almacenar valores** generados por los bloques de la implementación monociclo del procesador.

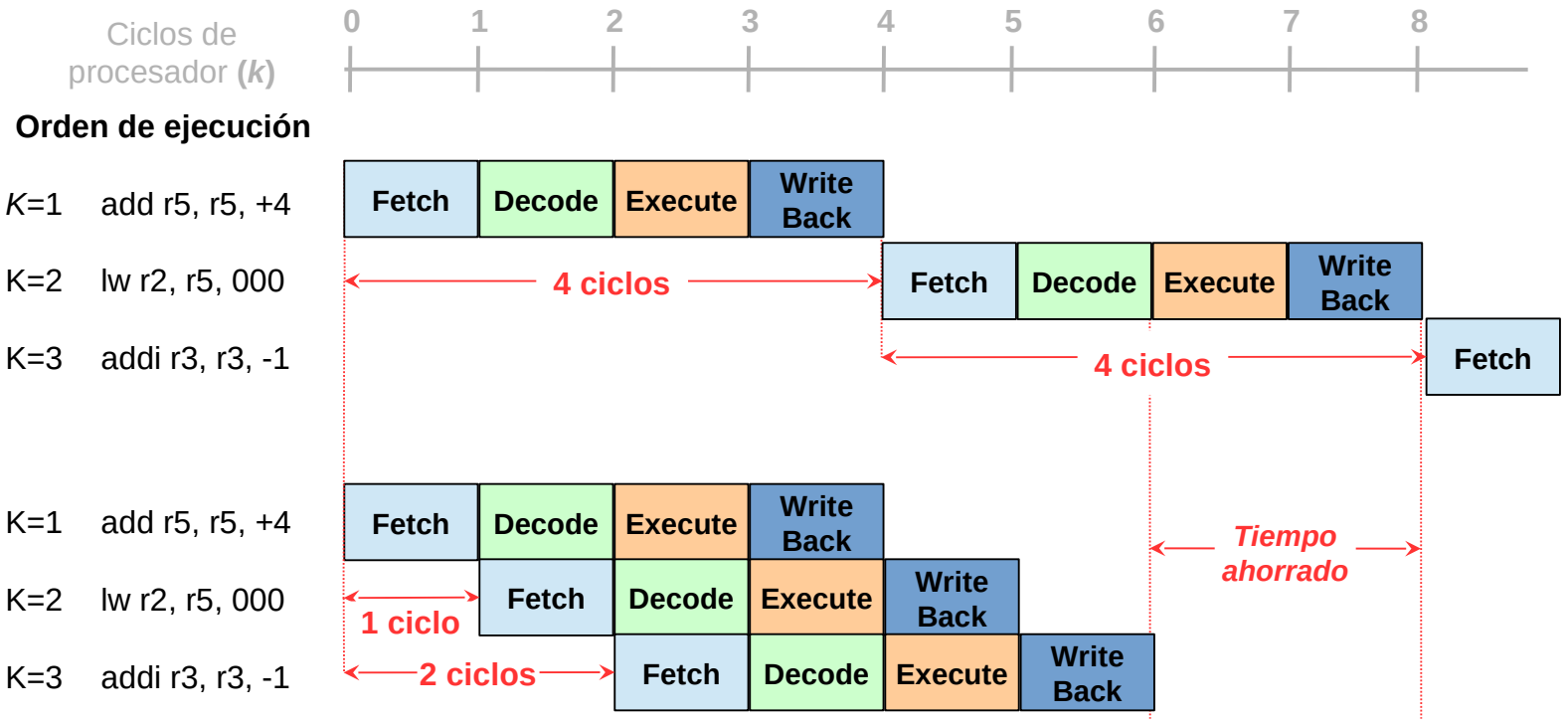


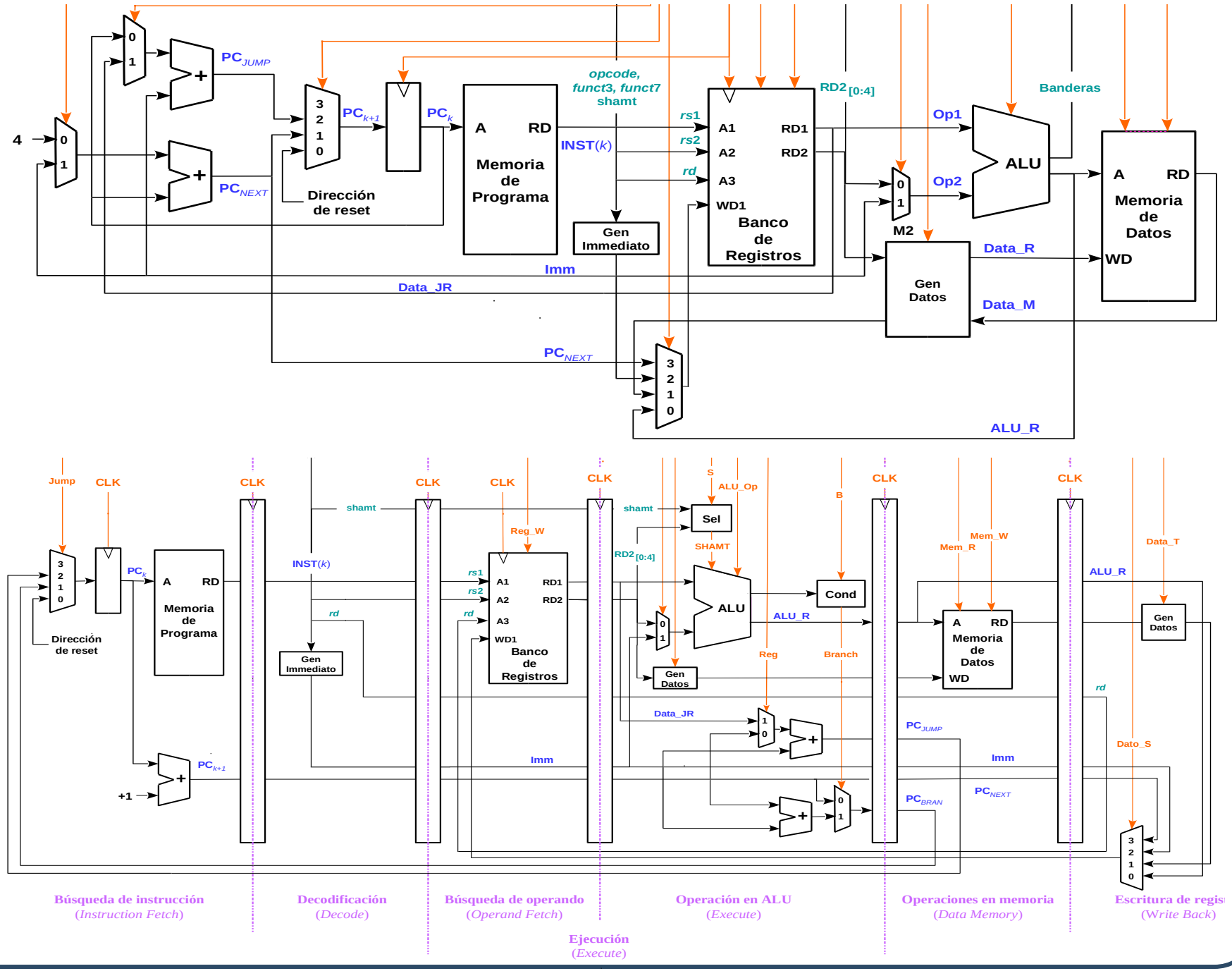
La microarquitectura del procesador

Implementación segmentada

Como cada bloque del procesador queda **libre una vez finalizada cada etapa de la instrucción**, este puede **utilizarse para iniciar la ejecución de la próxima instrucción**.

La idea detrás de la implementación segmentada es aprovechar los recursos del procesador disponibles para paralelizar la ejecución de las instrucciones.

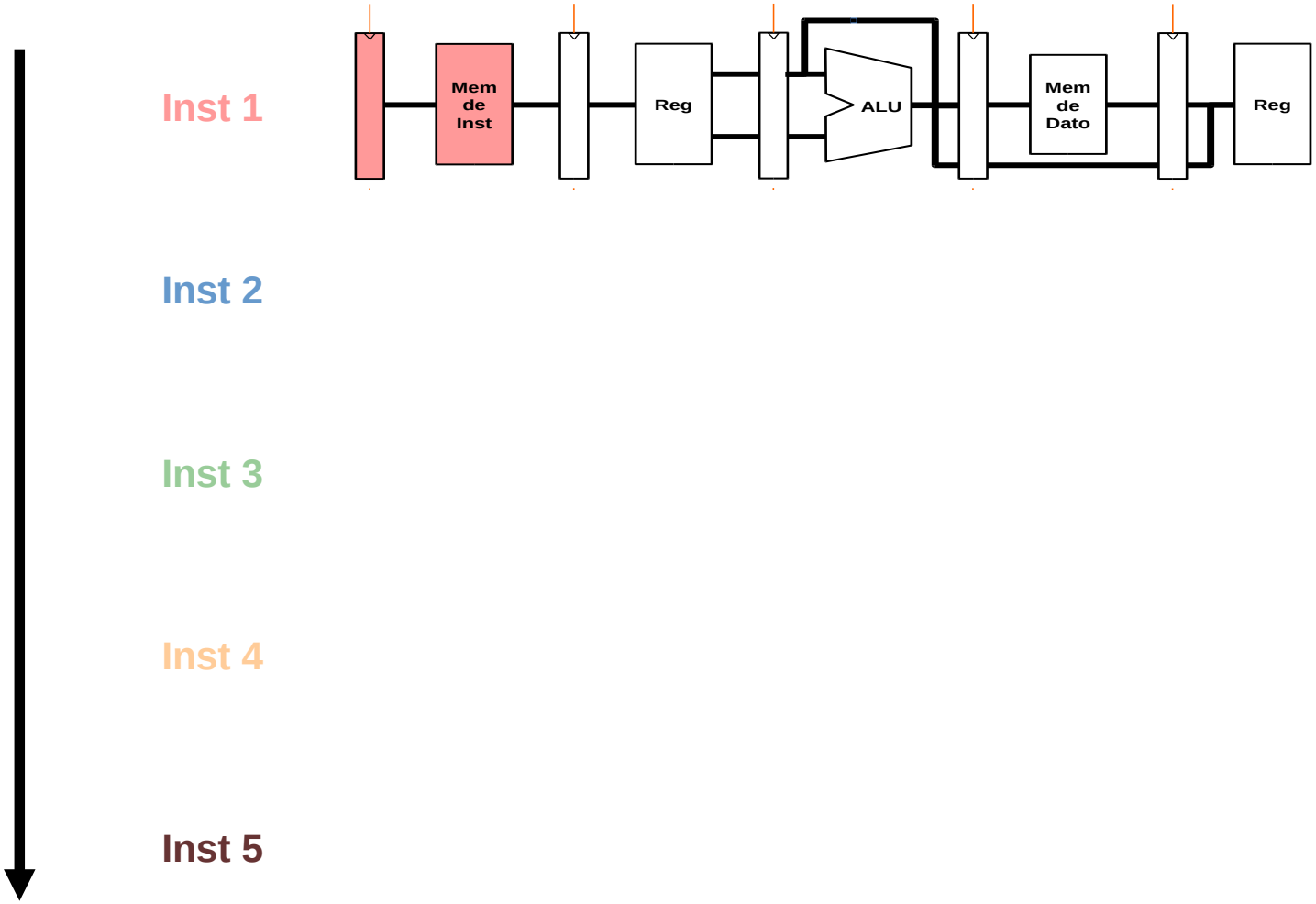



4
PC_JUMP
PC_NEXT
PC_k+1
PC_k
Dirección de reset
Memoria de Programa
A
RD
INST(k)
opcode, funct3, funct7 shamt
Gen Immediato
rs1
rs2
rd
Banco de Registros
A1
A2
A3
WD1
RD1
RD2
RD2 [0:4]
Banderas
Op1
Op2
ALU
M2
Gen Datos
Data_R
Data_M
Memoria de Datos
WD
Imm
Data_JR
PC_NEXT
ALU_R
Jump
CLK
shamt
Reg_W
Sel
ALU_Op
S
SHAMT
B
Cond
Reg
Branch
Mem_W
Mem_R
Data_T
Dato_S
PC_JUMP
PC_BRAN
Búsqueda de instrucción (Instruction Fetch)
Decodificación (Decode)
Búsqueda de operando (Operand Fetch)
Operación en ALU (Execute)
Operaciones en memoria (Data Memory)
Escritura de regis (Write Back)
Ejecución (Execute)

La microarquitectura del procesador

Implementación segmentada

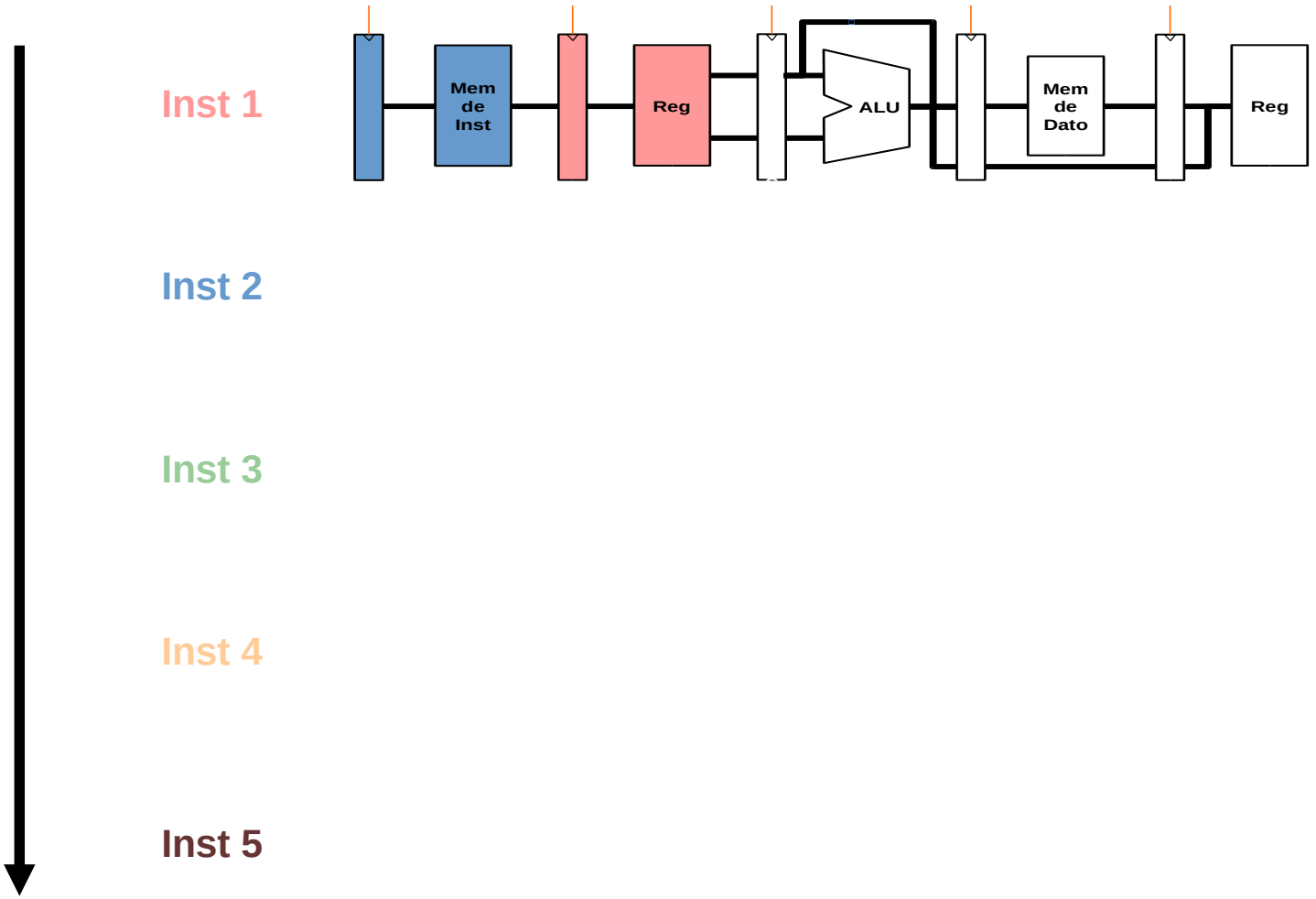
Visualizando la segmentacion: **Ciclo 1**



La microarquitectura del procesador

Implementación segmentada

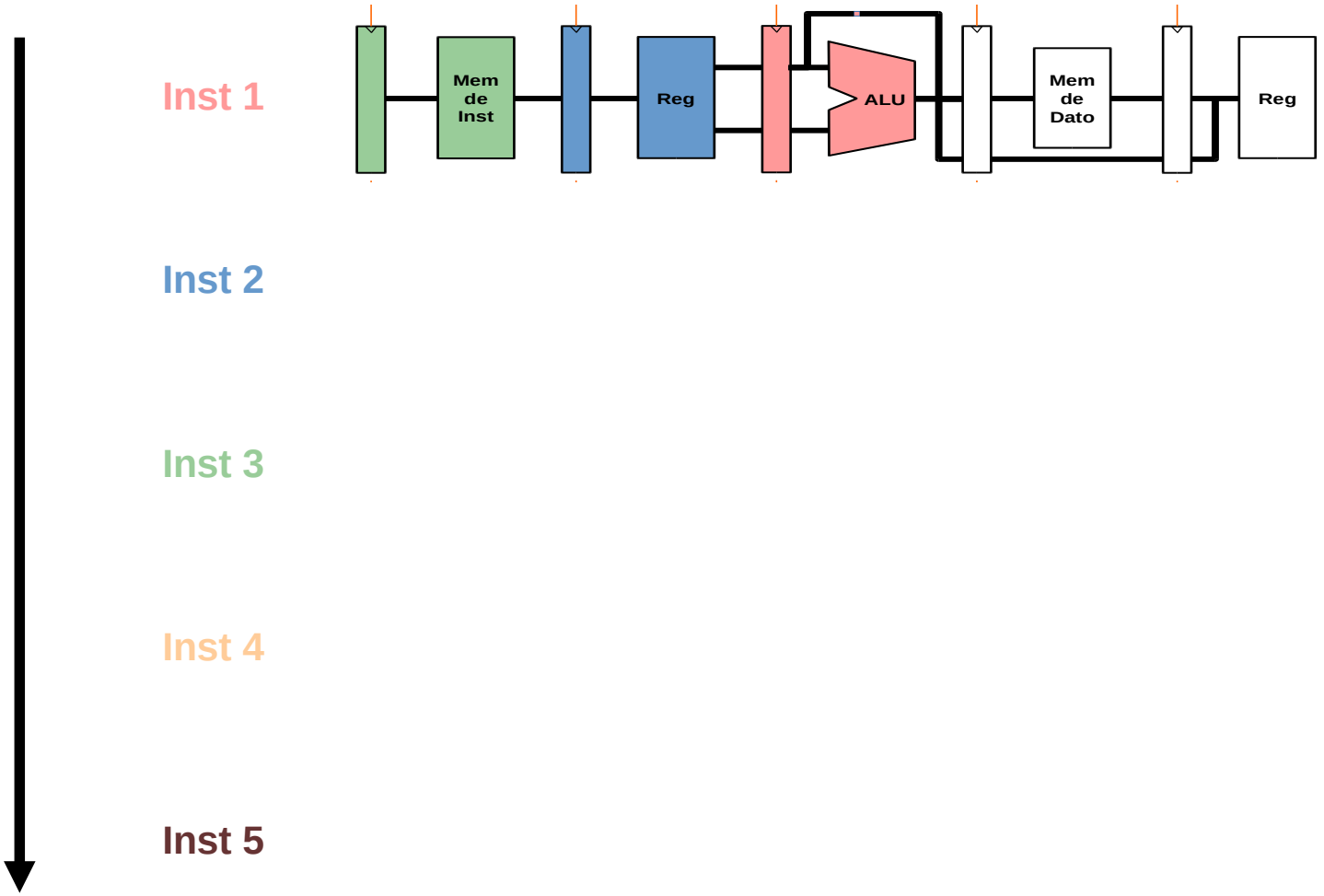
Visualizando la segmentacion: **Ciclo 2**



La microarquitectura del procesador

Implementación segmentada

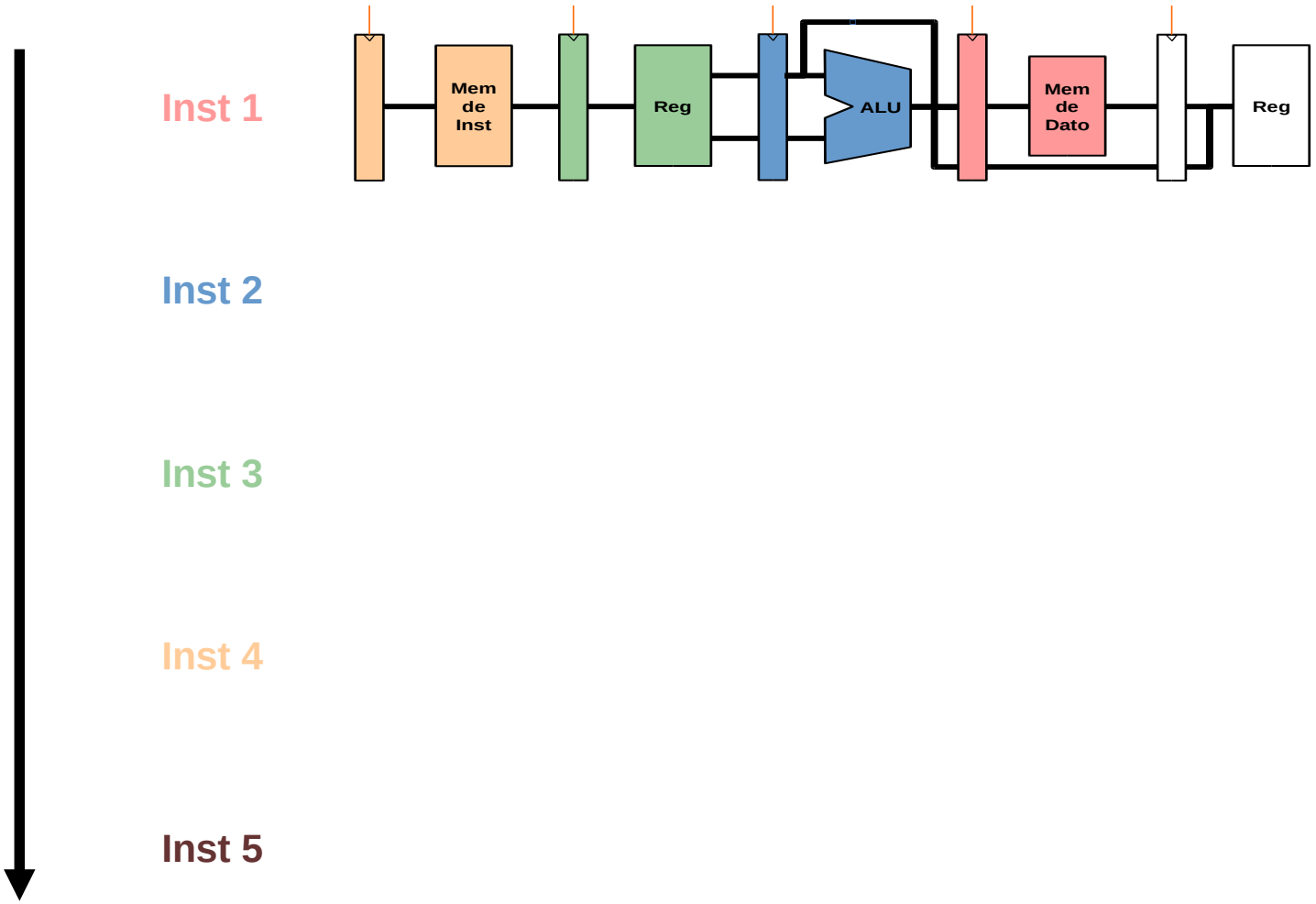
Visualizando la segmentacion: **Ciclo 3**



La microarquitectura del procesador

Implementación segmentada

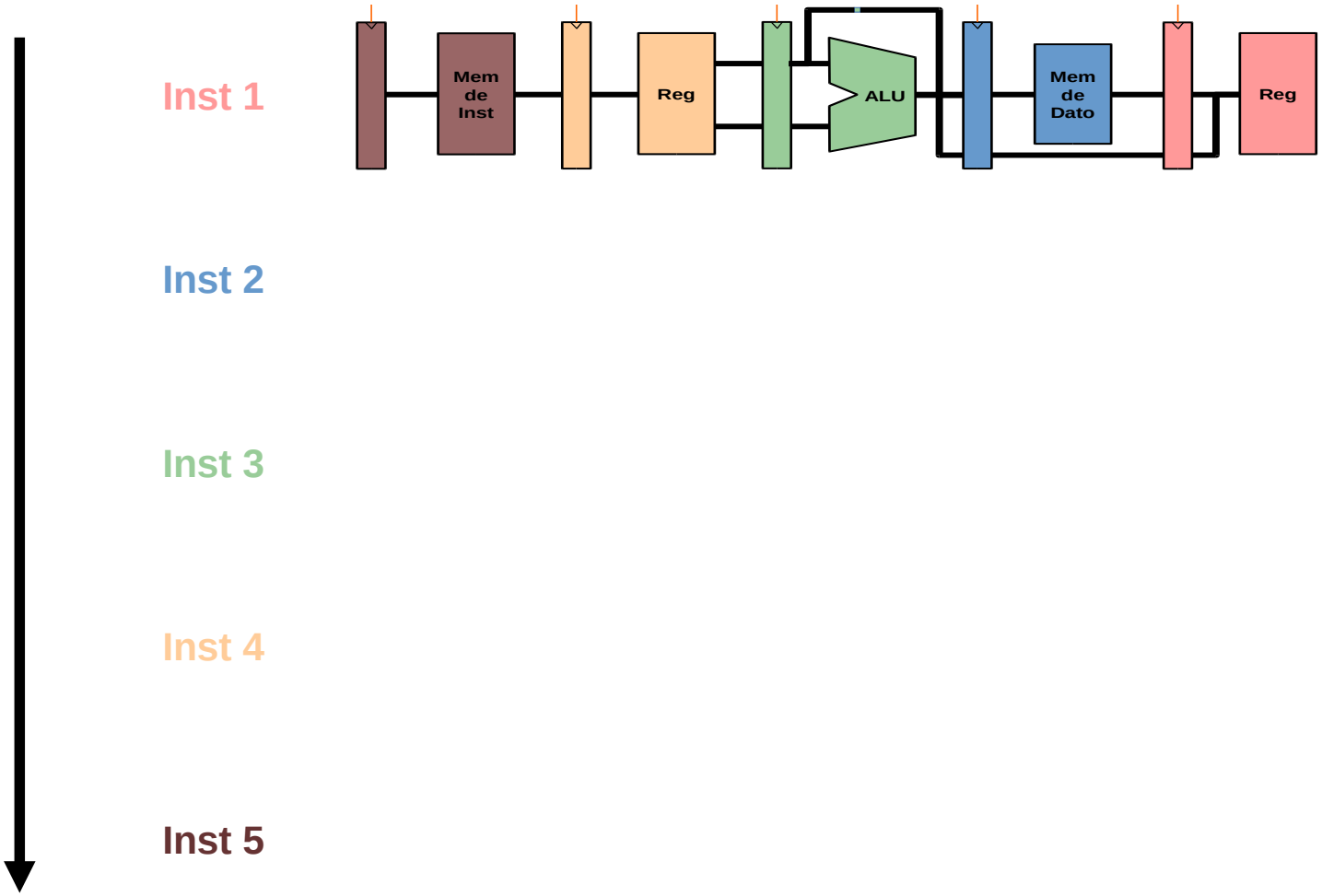
Visualizando la segmentacion: **Ciclo 4**



La microarquitectura del procesador

Implementación segmentada

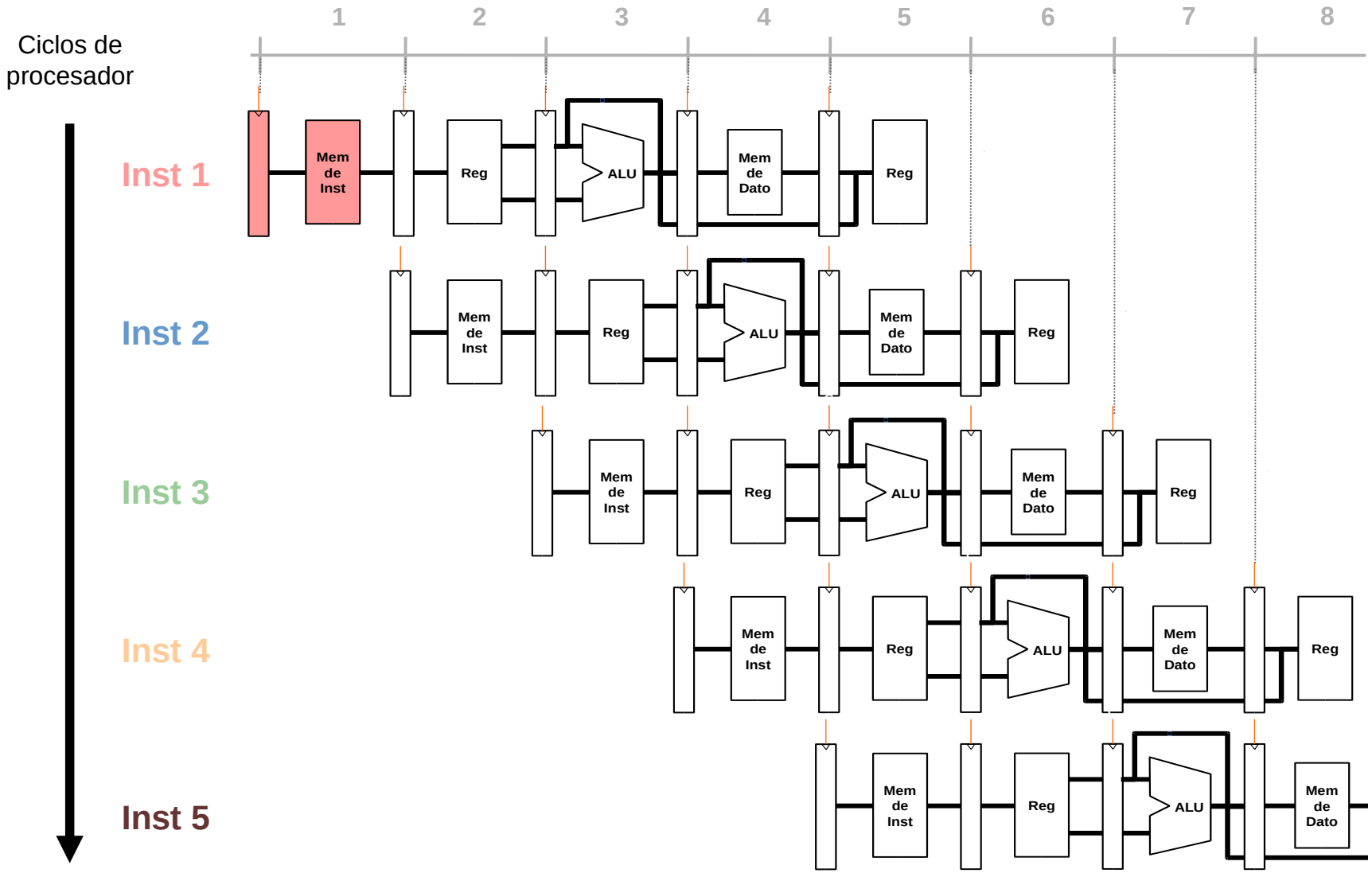
Visualizando la segmentacion: **Ciclo 5**



La microarquitectura del procesador

Implementación segmentada

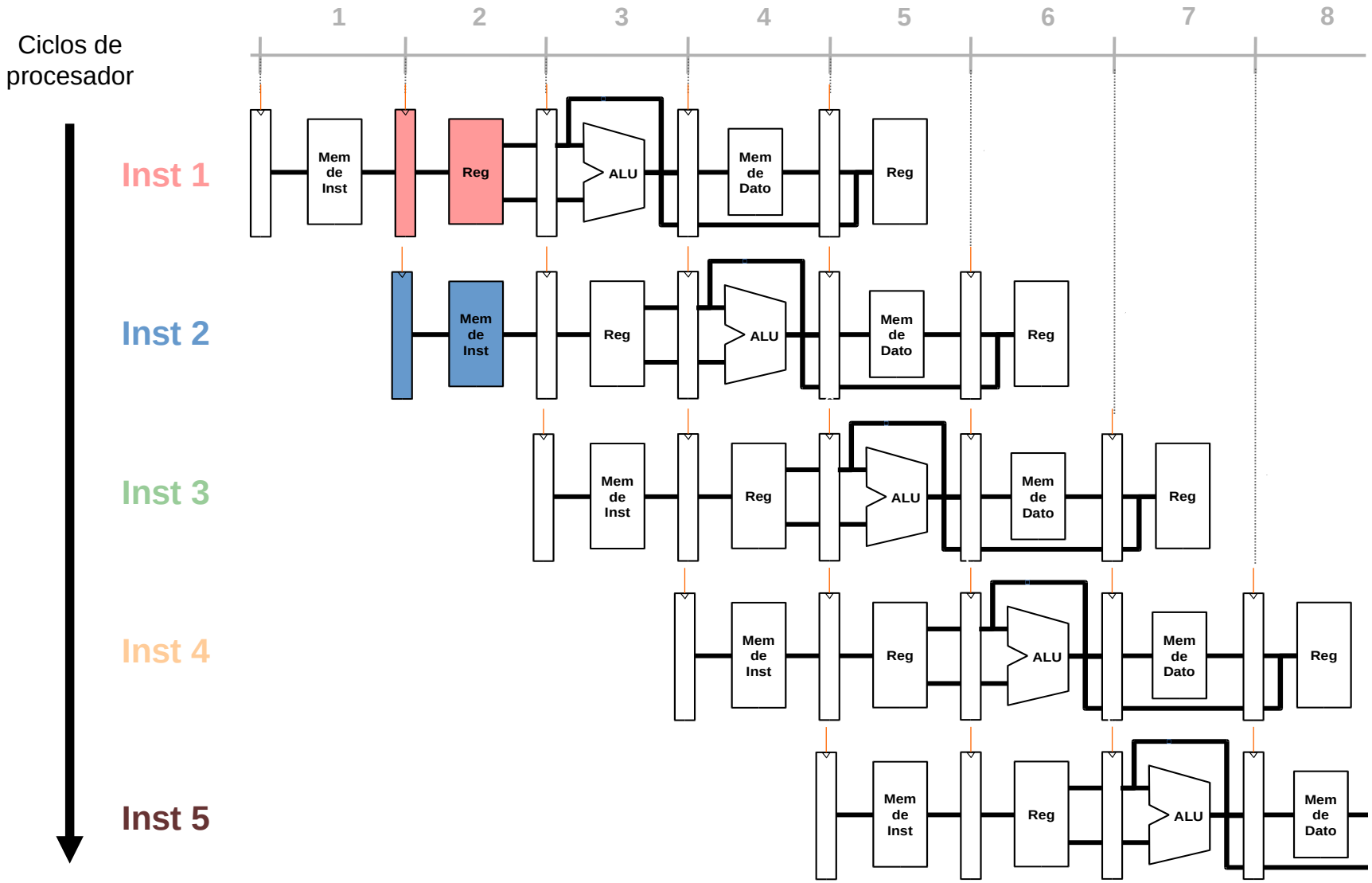
Visualizando la segmentacion: **Ciclo 1**



La microarquitectura del procesador

Implementación segmentada

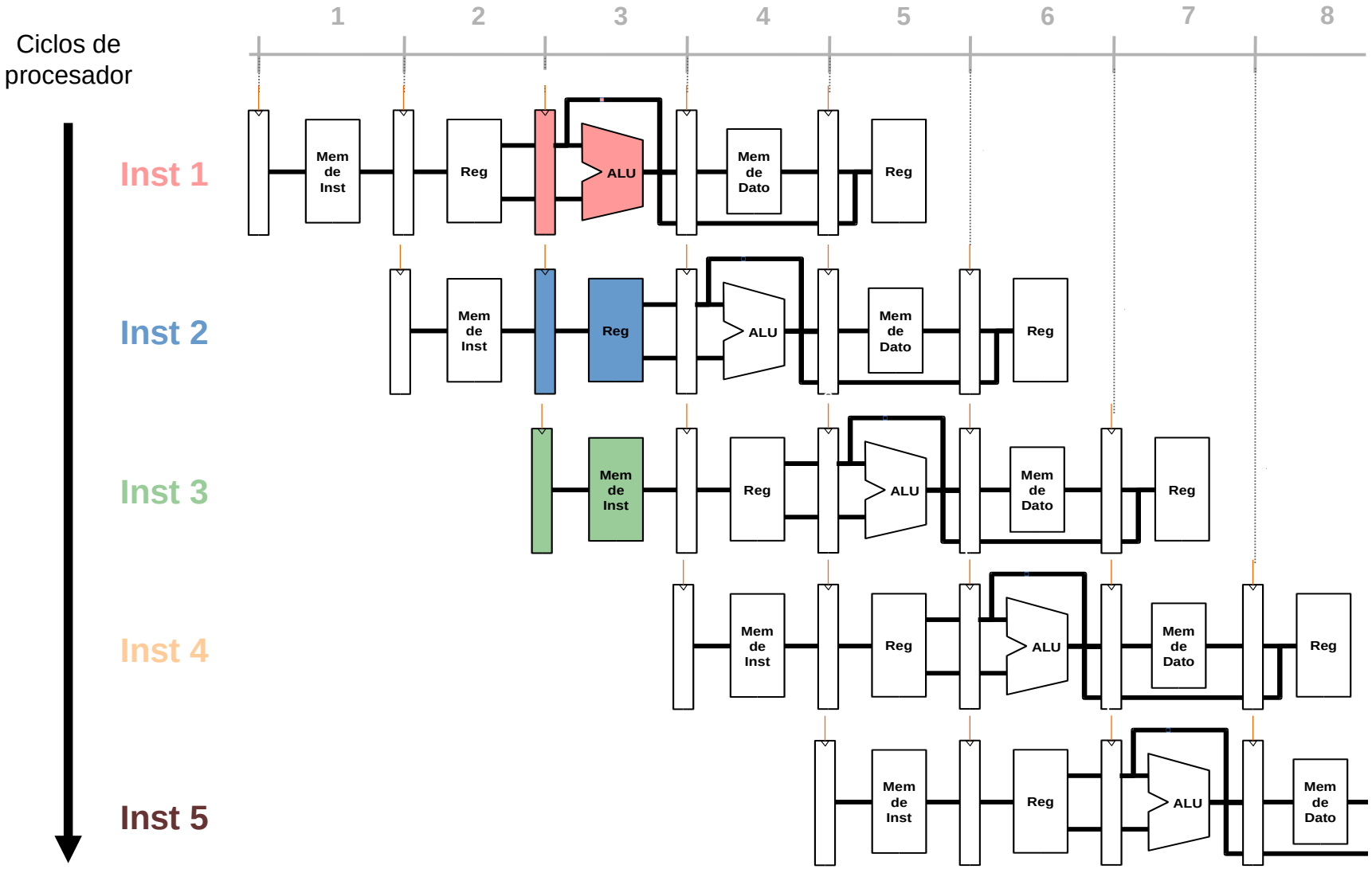
Visualizando la segmentacion: **Ciclo 2**



La microarquitectura del procesador

Implementación segmentada

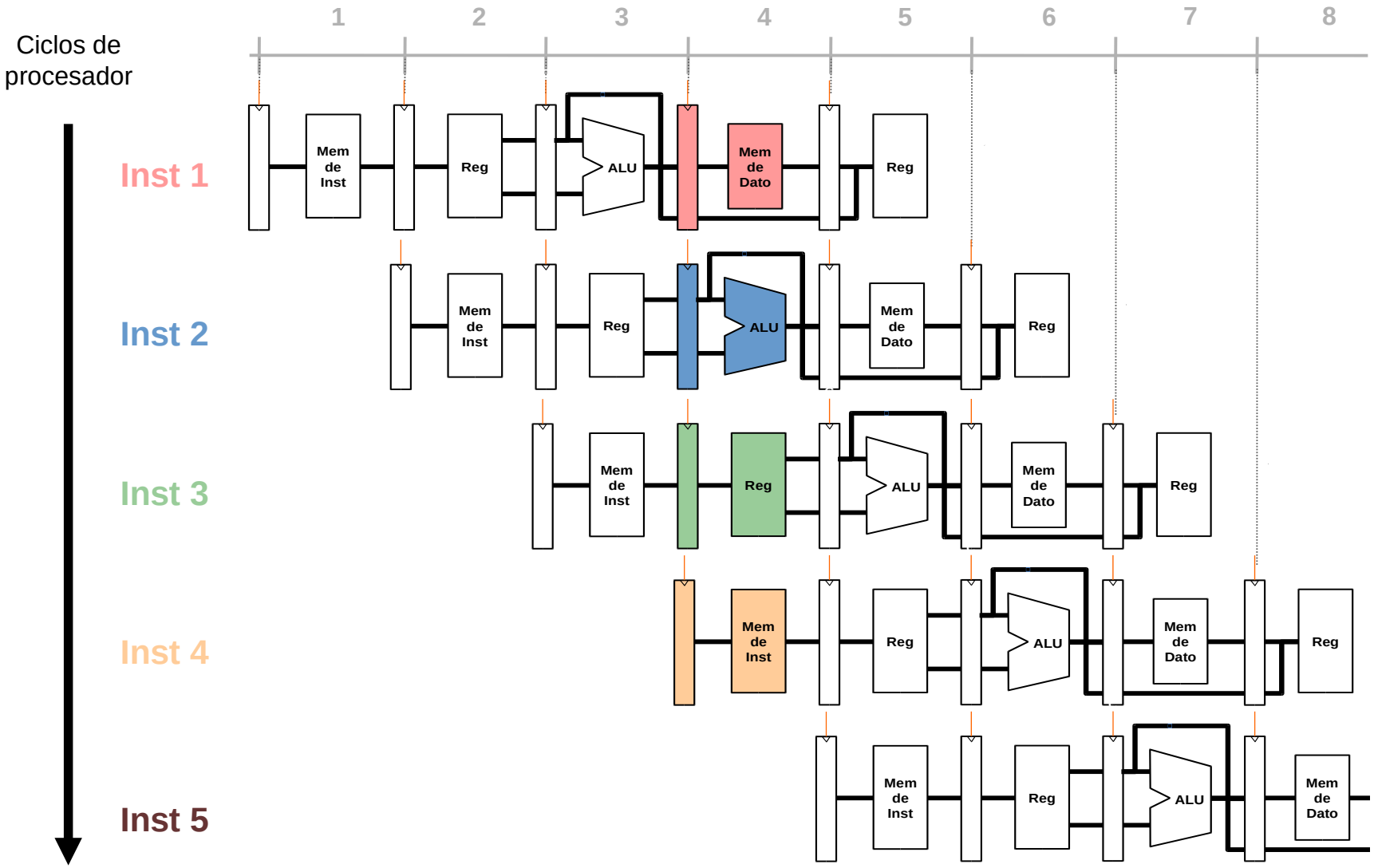
Visualizando la segmentacion: **Ciclo 3**



La microarquitectura del procesador

Implementación segmentada

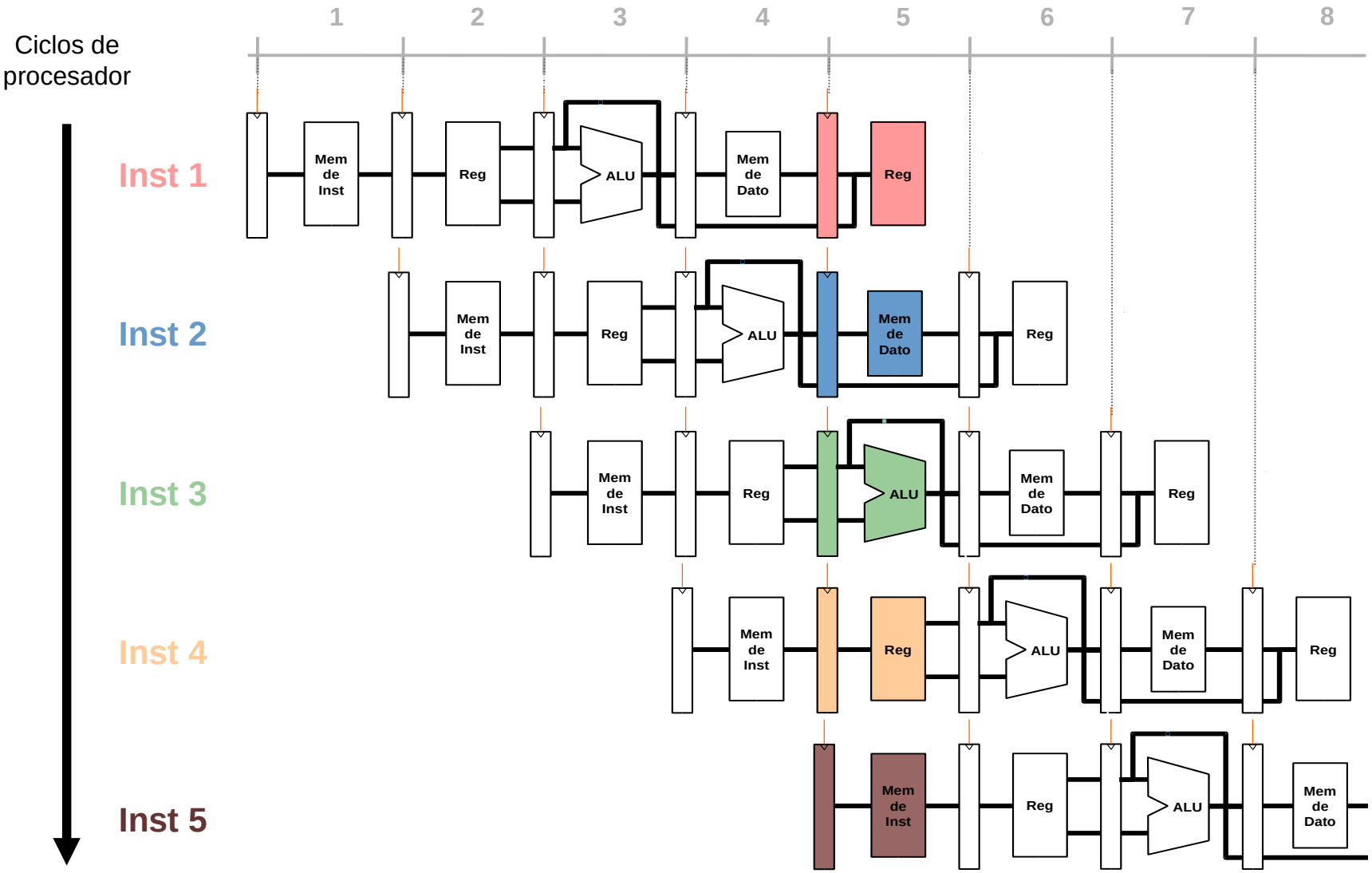
Visualizando la segmentacion: **Ciclo 4**



La microarquitectura del procesador

Implementación segmentada

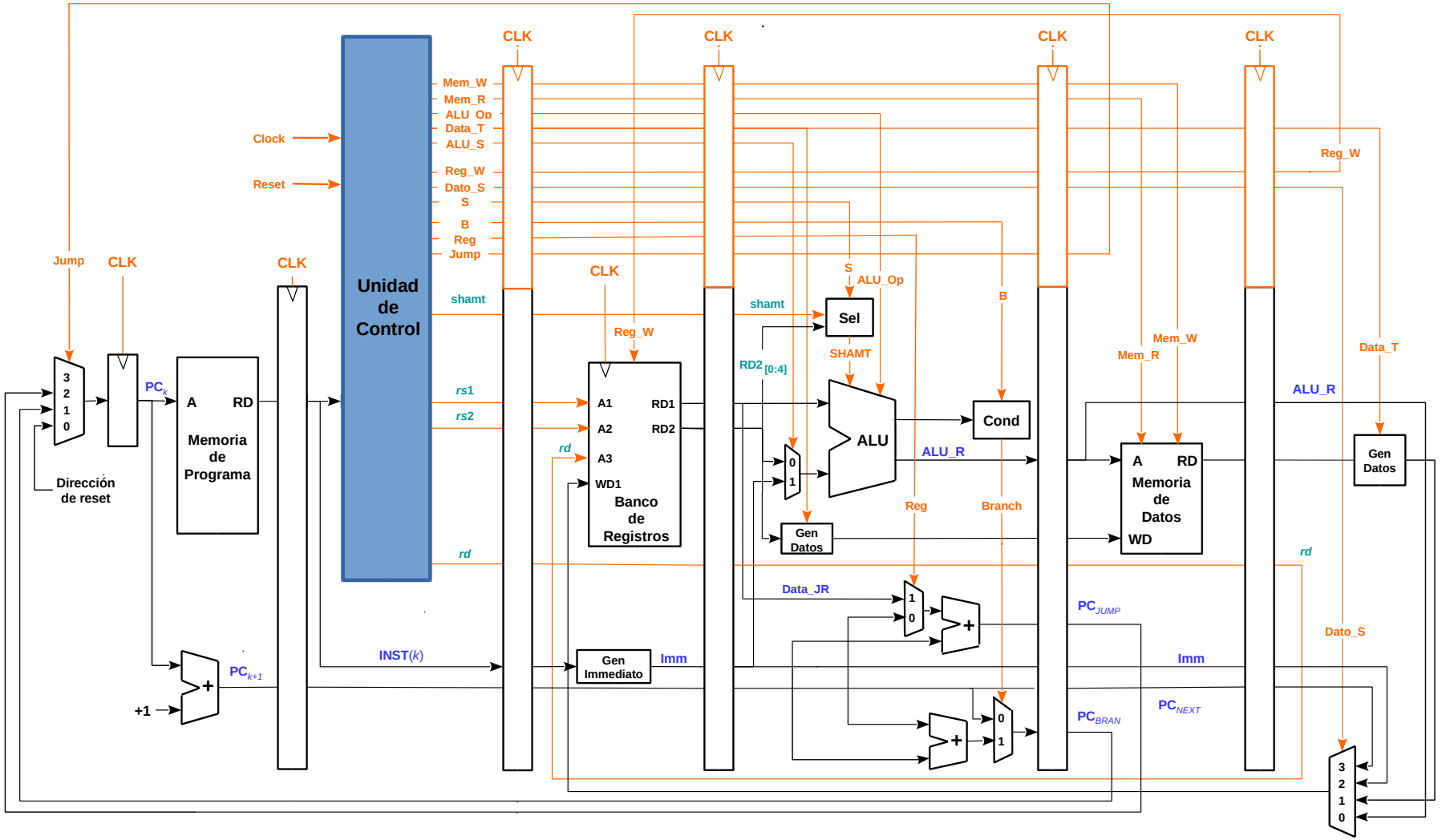
Visualizando la segmentacion: **Ciclo 5**



La microarquitectura del procesador

Implementación segmentada

La unidad de control es similar que la implementación monociclo (**circuito combinacional**) pero con las señales retardadas a lo largo de los bloques.



La microarquitectura del procesador

Implementación segmentada - Supersegmentación

El tiempo necesario para ejecutar una instrucción es

$$t = t_{Fetch\ Inst} + t_{Decode} + t_{Fetch\ Op} + t_{Execution} + t_{Writeback}$$

Un pipeline de *n*-etapas permite la ejecucion "*simultánea teórica*" de *n* instrucciones.

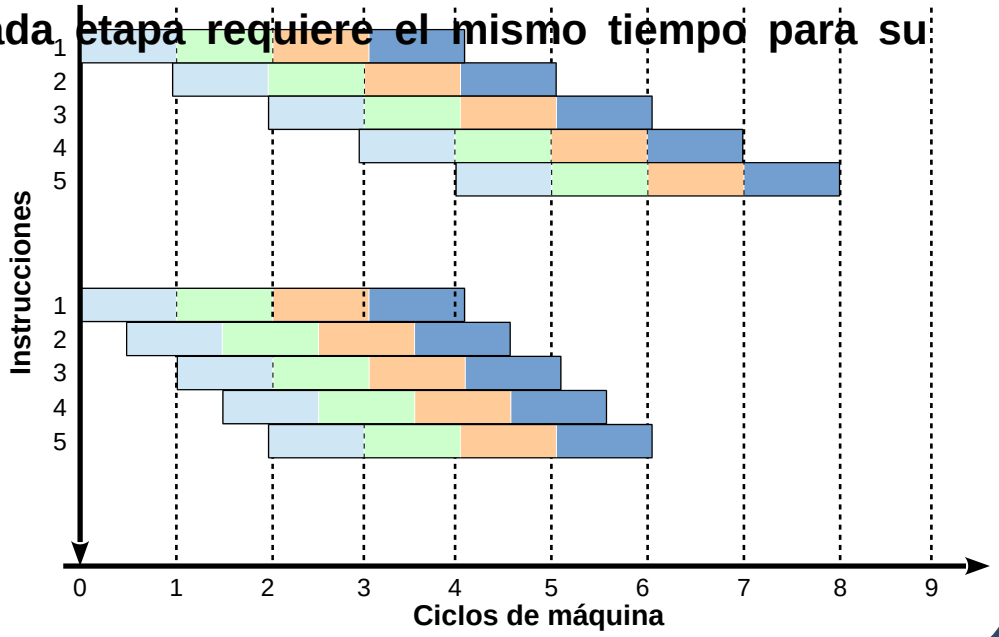
Cada etapa del pipeline realiza 1/*n* del trabajo total necesario. Por lo tanto

- Cada etapa solo requiere un tiempo *t/n* suponiendo que la microarquitectura esta perfectamente **balanceada**.

  Por balanceado se entiende que cada etapa requiere el mismo tiempo para su ejecucion.

- La frecuencia de reloj del procesador es $f_{pipe} = 1/(t/n) = n/t$

¿Si incrementamos *n*, incrementamos la potencia del procesador?



La microarquitectura del procesador

Implementación segmentada - Supersegmentación

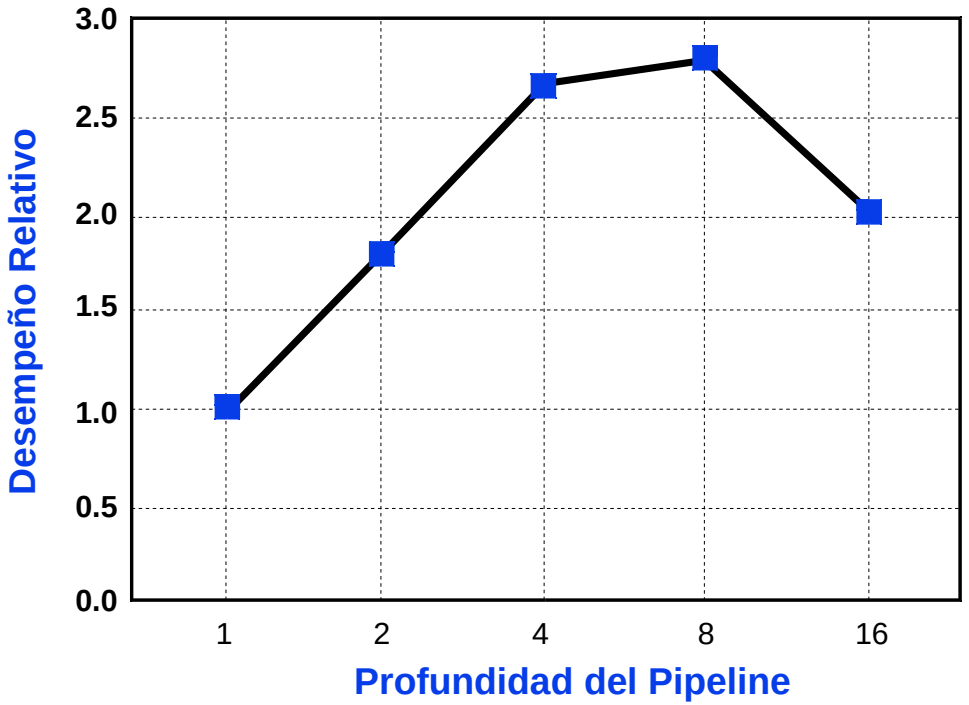
Incrementar la longitud del pipeline puede conducir a deterioros del desempeño global del procesador por que

Los pipelines necesitan mas tiempo para llenarse; y

Necesitan mas espacio de circuiteria en el circuito integrado.

Las instrucciones mas comunes (aritmeticas, movimientos de datos, saltos condicional) pueden iniciarse y ejecutarse de manera independiente.

La mayoria de las operaciones involucran cantidades escalares, por lo que mejorar el desempeño de estas operaciones conduce a una mejora al desempeño general del procesador



Riesgos de datos

La microarquitectura del procesador

Los riesgos

Los ***riesgos*** son situaciones que imposibilitan el comienzo de la ejecución de la próxima instrucción en el próximo ciclo.

Se clasifican en:

- **Riesgos estructurales:** Un recurso requerido para la ejecución esta ocupado;
- **Riegos de Datos:** Se necesita esperar a que se complete una instrucción previa para disponer del dato necesario en la instrucción actual. Por ejemplo

```
sub x2, x1, x3   // Register z2 written by sub
and x12, x2, x5   // 1st operand(x2) depends on sub
or  x13, x6, x2   // 2nd operand(x2) depends on sub
add x14, x2, x2   // 1st(x2) & 2nd(x2) depend on sub
sd   x15, 100(x2)  // Base (x2) depends on sub
```

- **Riegos de Control:** La decision sobre una acción de control de flujo de programa depende de una instrucción previa. Por ejemplo

```
36 sub x10, x4, x8
40 beq x1, x3, 16 //  PC-relative branch to 40+16*2=72
44 and x12, x2, x5
48 or x13, x2, x6
52 add x14, x4, x2
56 sub x15, x6, x7
```

La microarquitectura del procesador

Riesgos de datos

Los riesgos de datos ocurren cuando una instrucción se refiere a un resultado que aún no se ha calculado o recuperado.

```
sub x2, x1, x3   // Register z2 written by sub
and x12, x2, x5   // 1st operand(x2) depends on sub
or  x13, x6, x2   // 2nd operand(x2) depends on sub
add x14, x2, x2   // 1st(x2) & 2nd(x2) depend on sub
sd   x15, 100(x2)  // Base (x2) depends on sub
```

En estas situaciones, las instrucciones muestran la dependencia de los datos que modifican los datos en diferentes etapas del datapath.

Hay tres situaciones en las que puede ocurrir un peligro de datos:

* **Lectura despues de una escritura** (*Read-After-Write*) refiere a una situación en la que una instrucción se refiere a un resultado que aún no se ha calculado o recuperado. Esto puede ocurrir porque aunque una instrucción se ejecuta después de una instrucción anterior que ha sido procesado parcialmente por el dataph (***Dependencias de datos***).

* **Escritura despues de una lectura** (*Write-After-Read*) refiere a una situación en la que una instrucción escribe antes de que otra la lea (***Dependencias de nombre***).

* **Escritura despues de una escritura** (*Write-After-Write*) refiere a una situación en la que dos instrucciones sucesivas escriben el mismo registro, presentando una dependencia de nombre (***Reutilización de registros***).

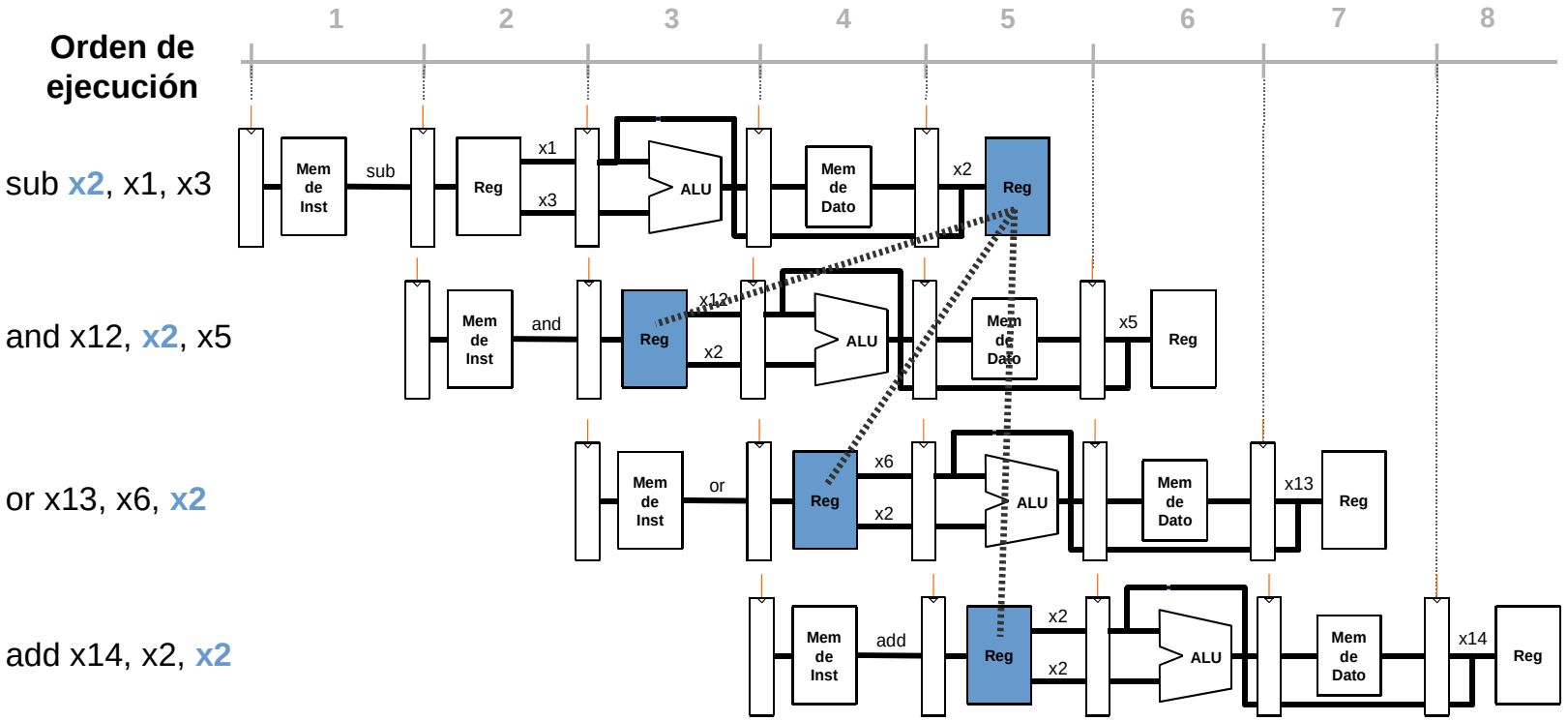
Los mecanismos de ejecución deben preservar el orden del programa.

Mismo resultado que en ejecución secuencial.

La microarquitectura del procesador

Riesgos de datos

Dependencias canalizadas en una secuencia de cinco instrucciones que utilizan rutas de datos simplificadas para mostrar las dependencias.



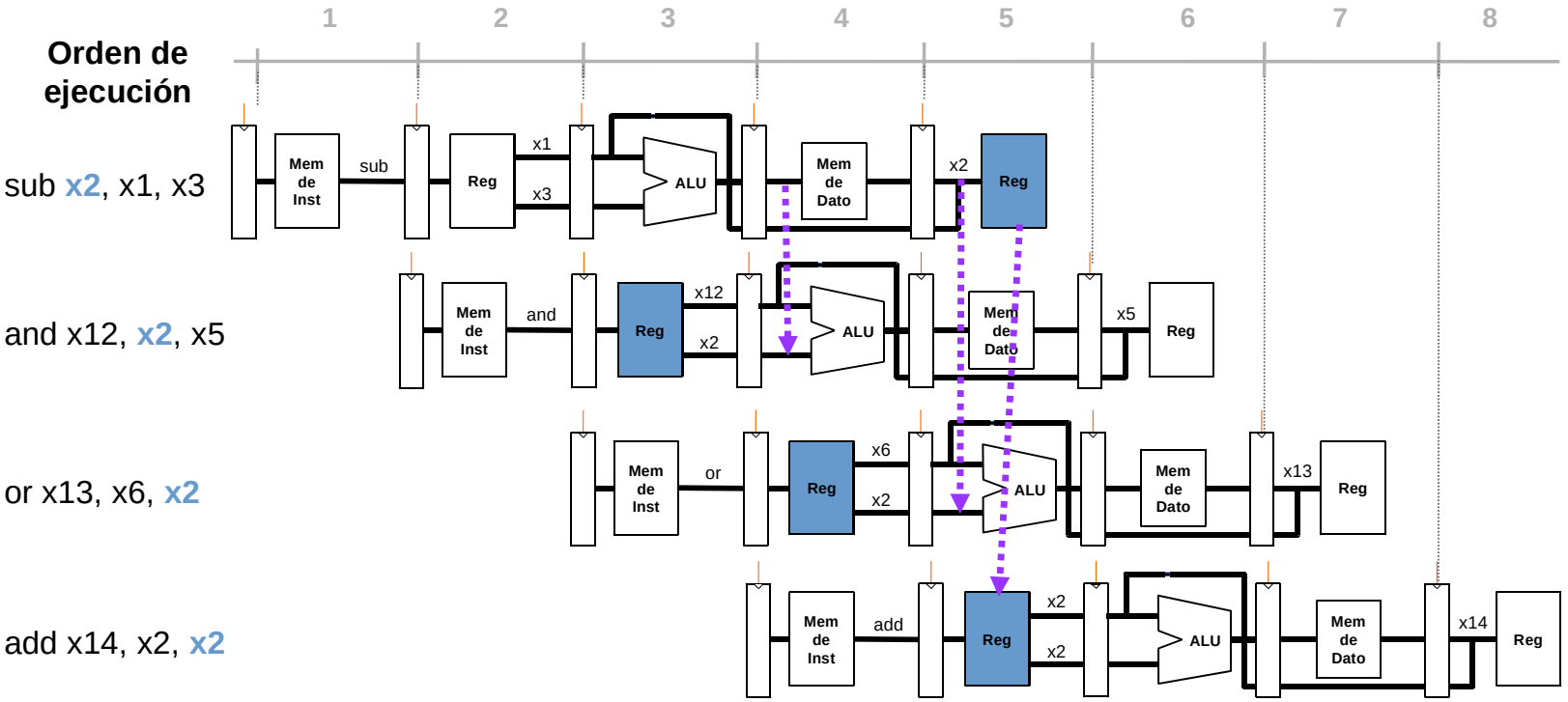
La microarquitectura del procesador

Riesgos de datos

Las dependencias entre los registros avanzan en el tiempo, por lo que es posible suministrar a la ALU las entradas requeridas por las instrucciones adelantando los resultados en el datapath.

Los valores en los registros del datapath muestra que el valor deseado está disponible antes de que se lo escriba en el registro. Suponemos que el registro reenvía los valores que se leen y escriben durante el mismo ciclo de reloj, pero los valores provienen del registro en lugar de un registro del datapath.

La solución al riesgo generado por una lectura antes de una escritura es el ***adelanto de los datos***.



La microarquitectura del procesador

Riesgos de datos – Adelanto de los datos

La **unidad de adelanto** de datos estará en la etapa ejecución porque la ALU se encuentra en esta etapa. Por lo tanto, debemos pasar los registro de operandos desde la etapa de decodificación a través del registro de canalización para determinar si se deben adelantar los datos.

Antes de incluir la unidad de adelanto de datos, el registro no tenía necesidad de incluir espacio para guardar rs1 y rs2.

Las condiciones para detectar los riegos y las señales de control para resolverlas son:

Operando 1

```
IF Reg_W Mem = 1 THEN
        IF rs1 Mem = rs1 THEN F Op1 = 01
        IF rs1 WB = rs1 THEN F Op1 = 10
ELSE
        F Op1 = 00
END
```

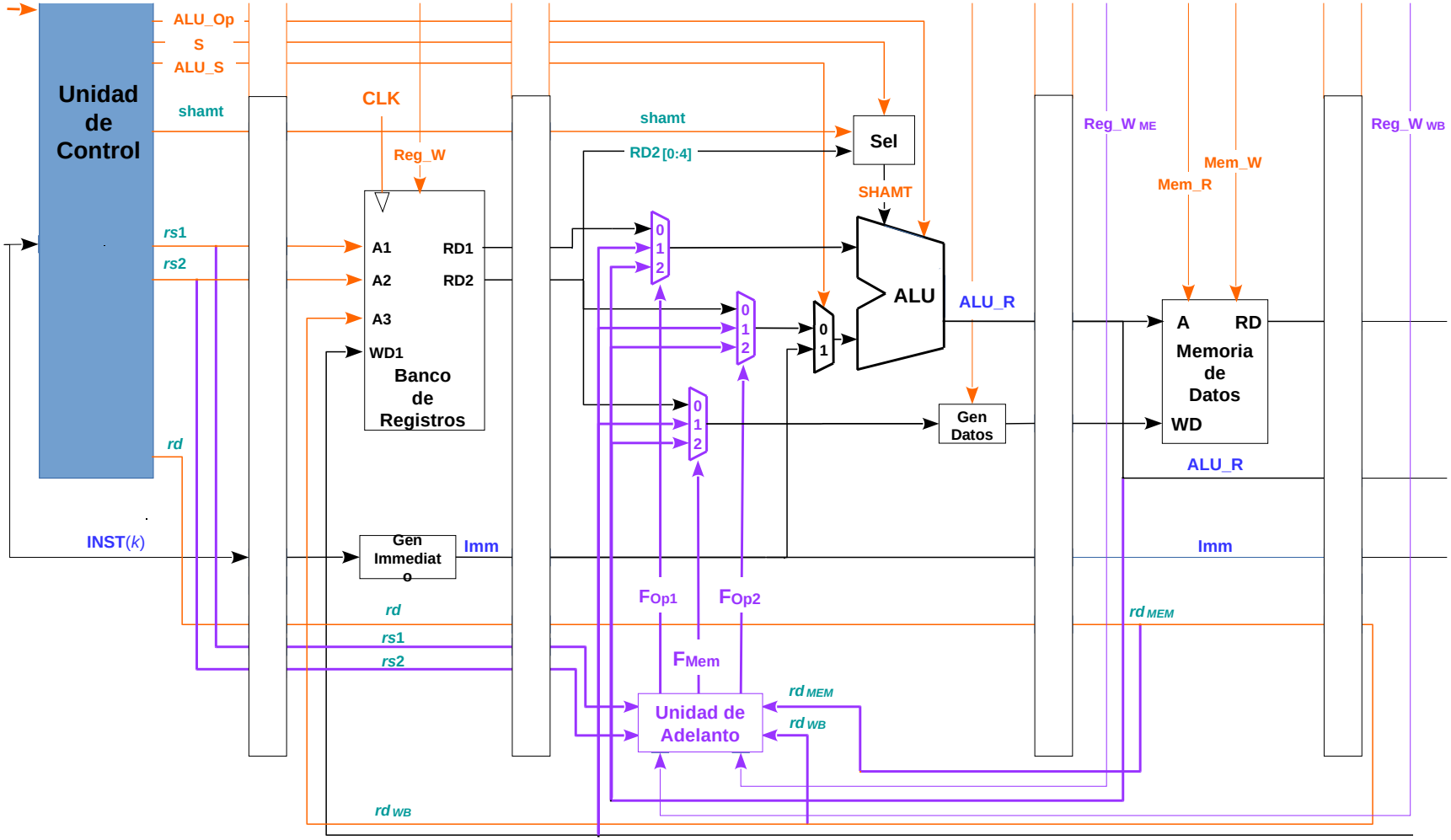
Operando 2

```
IF Reg_W Mem = 1 THEN
        IF rs2 Mem = rs2 THEN F Op2 = 01
        IF rs2 WB = rs1 THEN F Op2 = 10
ELSE
        F Op2 = 00
END
```

La microarquitectura del procesador

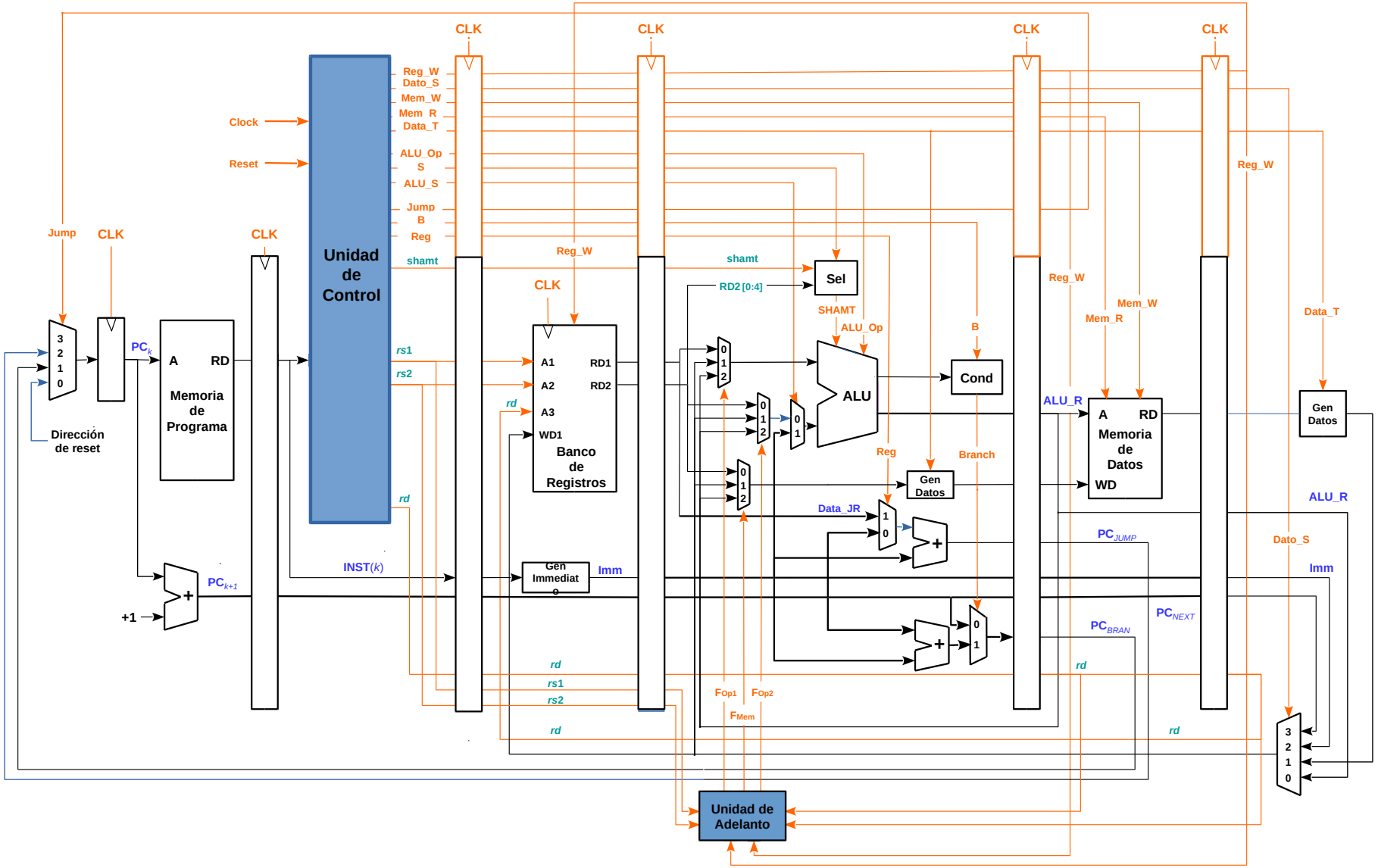
Riesgos de datos – Adelanto de los datos

Estas reglas se implementan de la siguiente manera



La microarquitectura del procesador

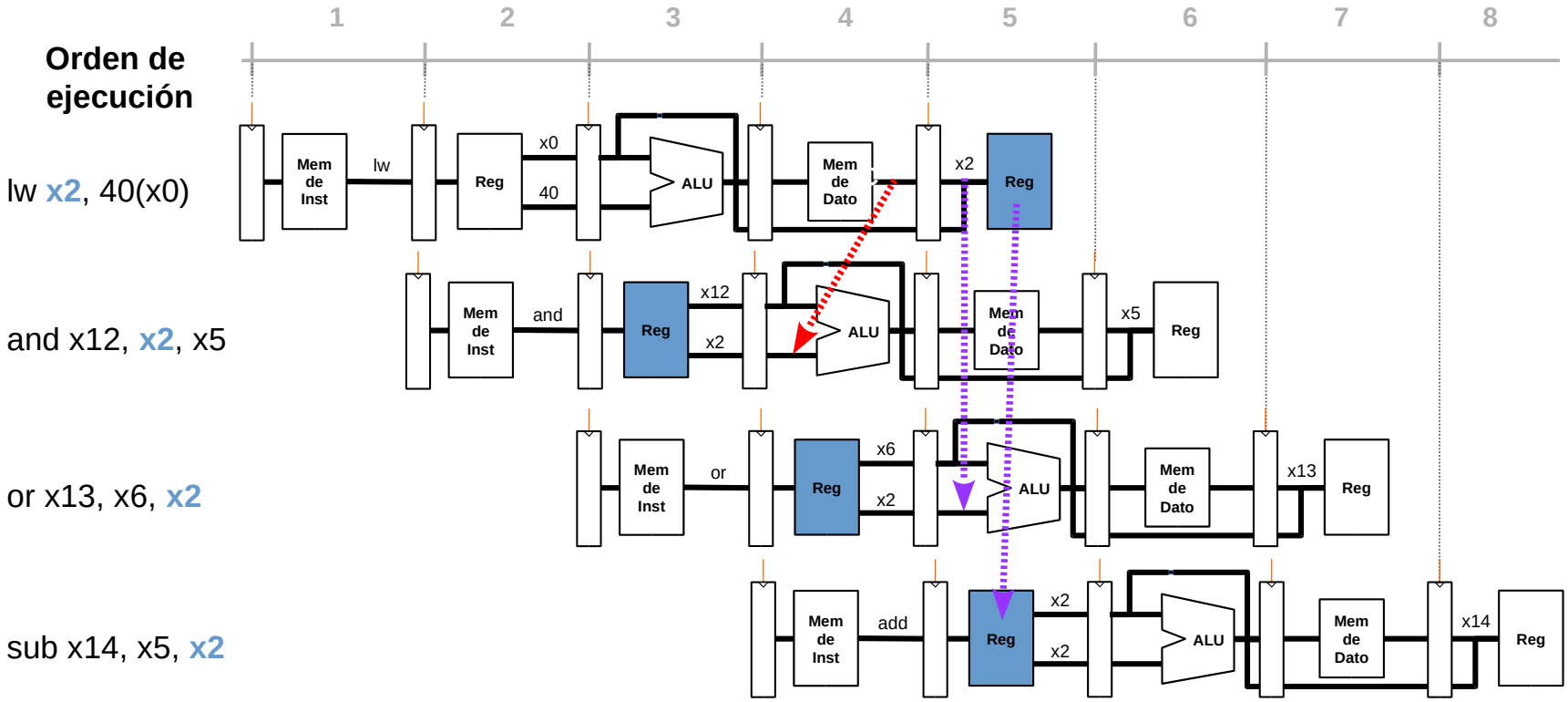
Riesgos de datos – Adelanto de los datos



La microarquitectura del procesador

Riesgos de datos - Detención

Los riesgos generados por una lectura antes de una escritura no pueden ser solucionados con adelanto de datos por que el dato no está disponible.



Estos riesgos se eliminan **deteniendo de la etapa** (*stall*) que presenta la dependencia, de modo que el dato pueda ser leido.

La microarquitectura del procesador

Riesgos de datos - Detención

Hay dos tipos de detenciones :

Burbujas: son instrucciones **NOP** insertadas entre dos instrucciones en el datapath.

- Evita que las instrucciones anteriores del datapath (adelantadas en el código) progresen por el datapath en un ciclo (las adelanta con las señales de control de escritura);
- Inserta una **NOP** poniendo a cero los bits de control en el registro del datapath en las fases adecuadas;
- Hace que las instrucciones posteriores en el datpath (anteriores en el código) progresen normalmente.

Vacíado: todas las instrucciones del datapath son sustituidas con instrucciones **NOP**.

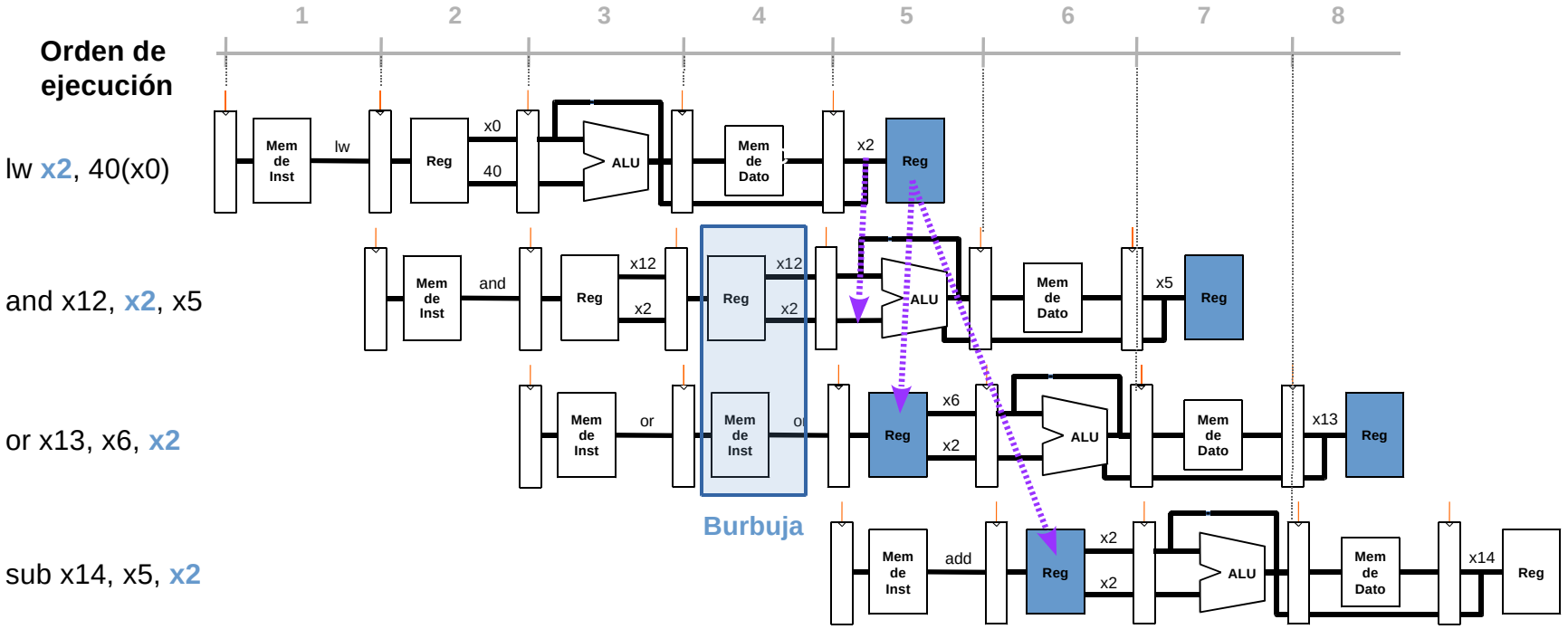
- Resetea los bits de control para la instrucciones que deben ser eliminadas

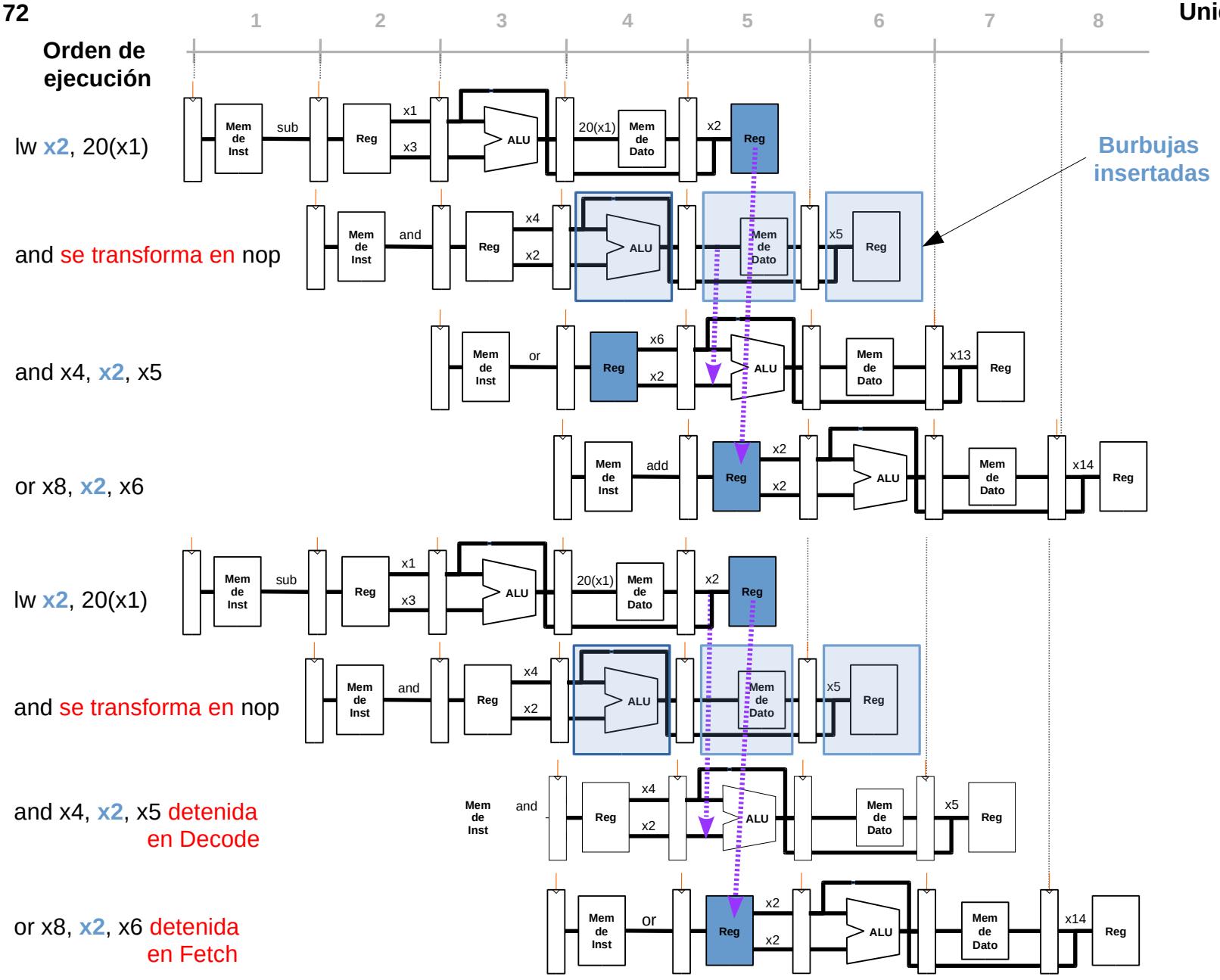
La microarquitectura del procesador

Riesgos de datos - Detención

Una secuencia instrucciones en un datapath segmentado en el cual la dependencia entre las instrucciones de carga (*lw*) y la siguiente (*and*) retrocede en el tiempo.

Por lo tanto, no se puede resolver mediante el adelanto de datos. Por lo tanto, esta combinación debe dar lugar a una detención del datapath por el riesgo generado.




1
2
3
4
5
6
7
8
Orden de ejecución
lw x2, 20(x1)
and se transforma en nop
and x4, x2, x5
or x8, x2, x6
lw x2, 20(x1)
and se transforma en nop
and x4, x2, x5 detenida en Decode
or x8, x2, x6 detenida en Fetch
Burbujas insertadas
Mem de Inst
Reg
ALU
Mem de Dato
sub
and
or
add
x1
x2
x3
x4
x5
x6
x13
x14
20(x1)

La microarquitectura del procesador

Riesgos de datos - Detención

Necesitamos una unidad de detección de peligros que funcione durante la etapa de decodificación para que pueda insertar el bloqueo entre la carga y las instrucciones que dependen de ella.

Comprobando las instrucciones de carga esta condición

```
IF Mem_R = 1
      IF rd = rs1 OR rd = rs1 THEN En = 0
ELSE
      En = 1
END
```

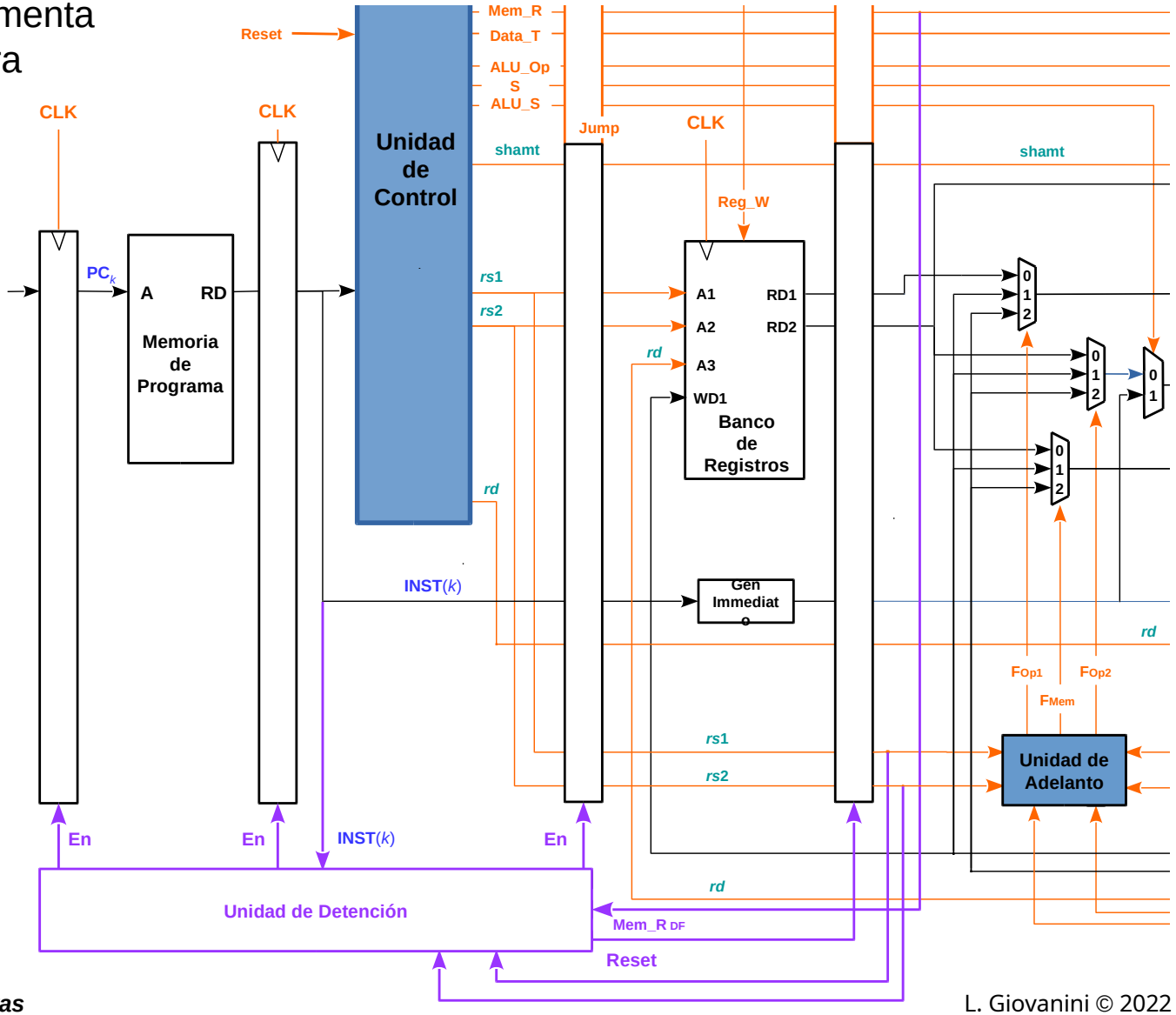
Si la instrucción en la etapa de decodificación es detenida, la instrucción en la etapa de búsqueda también debe ser detenida, de lo contrario, perderíamos la instrucción leída de la memoria.

Evitar que estas dos instrucciones avancen se logra simplemente impidiendo que el registro del contador de programa PC y el registro del datapath también.

La microarquitectura del procesador

Riesgos de datos - Detención

Esta regla se implementa de la siguiente manera



La microarquitectura del procesador

Riesgos de datos

El orden en que se ejecutan determinan el tiempo necesario para ejecutarlo en un procesador con arquitectura segmentada.

Por ejemplo, el **programa 1** necesita 13 ciclos, en lugar de los 11 ciclos teoricos, debido a que el procesador debe introducir dos detenciones (*stall*) debido a los riesgos de datos originados por las intrucciones de carga (lw).

Programa 1

```
lw    $x1, 40($x10)
lw    $x2, 44($x10)
add   $x3, $x1, $x2
sw    $x3, 52($x10)
lw    $x4, 48($x10)
add   $x5, $x1, $x4
sw    $x5, 56($x10)
```

Riesgo

Riesgo

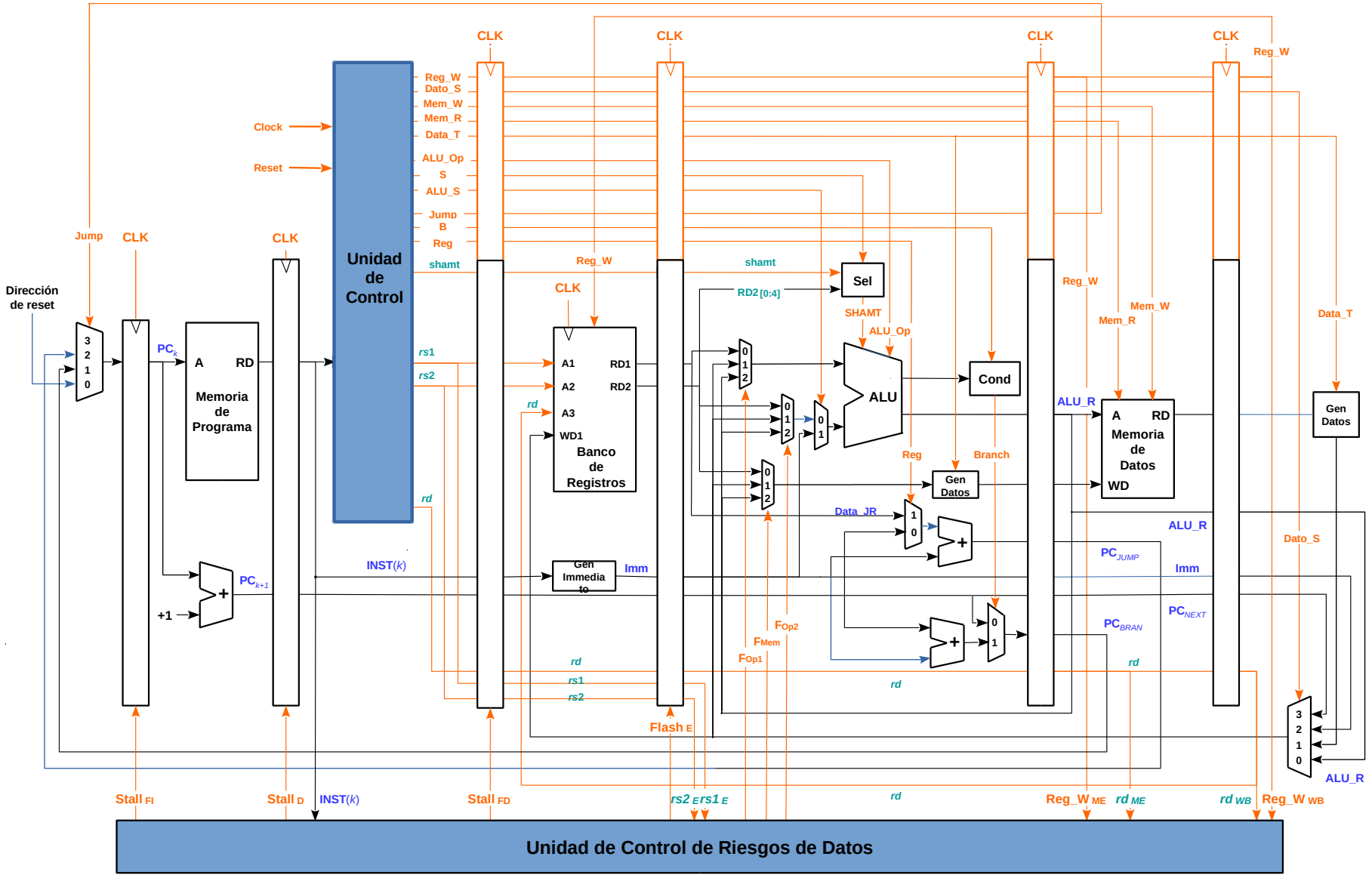
Programa 2

```
lw    $x1, 40($x10)
lw    $x2, 44($x10)
lw    $x4, 48($x10)
add   $x3, $x1, $x2
sw    $x3, 52($x10)
add   $x5, $x1, $x4
sw    $x5, 56($x10)
```

Mientras que el **programa 2** solo necesita 11 ciclos, en coincidencia con los ciclos teoricos requeridos, debido a que el programa ha sido reordenado para evitar los riesgos de datos originados por las intrucciones de carga (lw).

La microarquitectura del procesador

Riesgos de datos



Riesgos de control

La microarquitectura del procesador

Riesgos de control

Dependencias de control

Cada instrucción depende de un conjunto de saltos y esta dependencia, *orden de ejecución*, debe preservarse para preservar el *orden del programa*;

Las dependencias de control ***pueden violarse si no afectan*** los resultados del programa.

***LO IMPORTANTE* es preservar el comportamiento del flujo de datos.**

Dependencias de control y excepciones

Comportamiento de excepciones debe preservarse. Cualquier cambio en el orden de ejecución no debe cambiar como las excepciones son atendidas en la ejecución.

Dependencias de control y flujo de datos

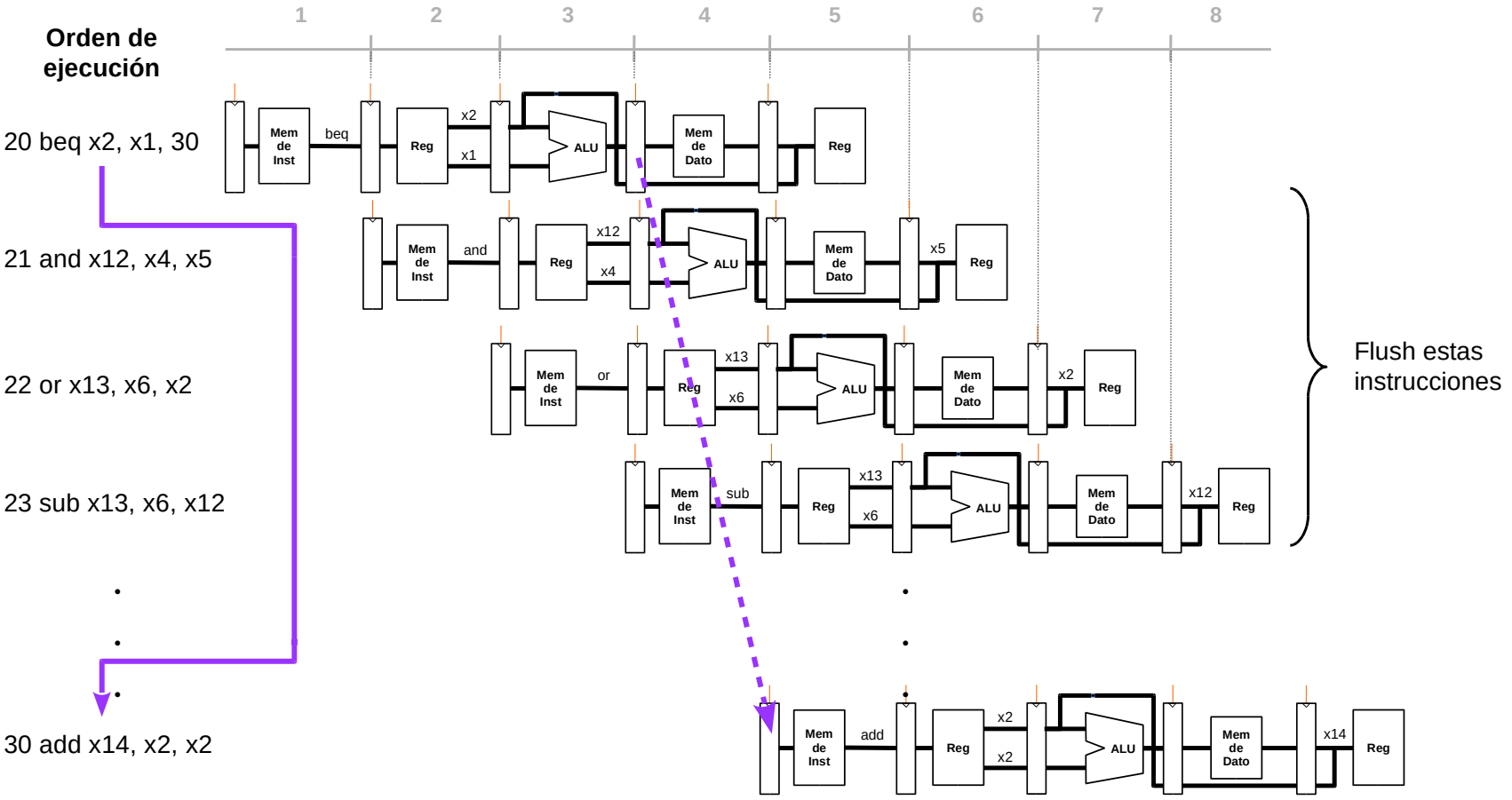
Se debe mantener el flujo de datos entre instrucciones productoras y consumidoras de datos.

La microarquitectura del procesador

Riesgos de control

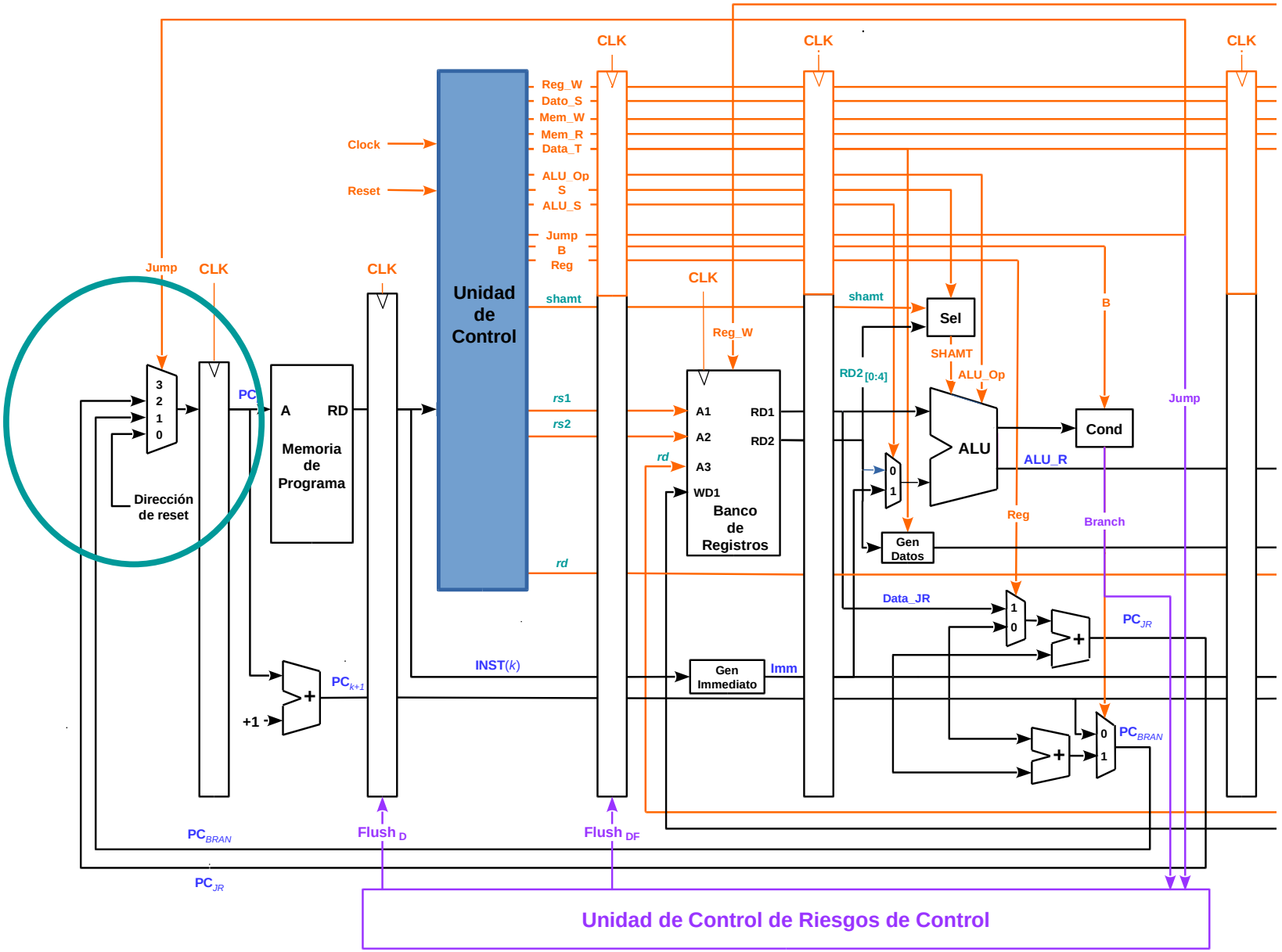
Solución del riesgo generado por un salto **través de reseteo** de la ejecución del datapath (*flushing*)

IF Jump = 1 AND Branch = 1 THEN $Flush_D$ = 1; $Flush_{DF}$ = 1 END



La microarquitectura del procesador

Riesgos de control



La microarquitectura del procesador

Riesgos de control

A partir de un análisis estadístico del uso de intrucciones de saltos en programas surge que

En promedio

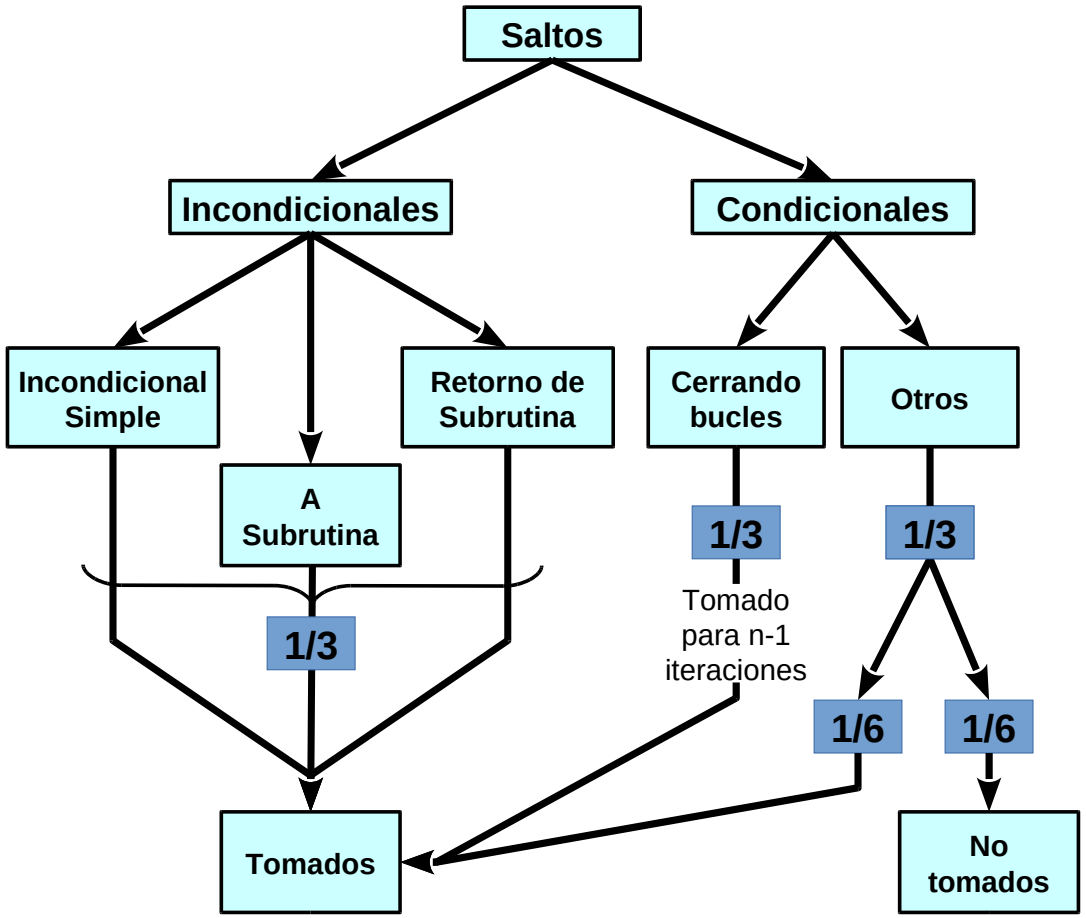
Instrucciones de salto:	**1 de cada 5**
Saltos condicionales:	**2 de cada 3**
Saltos incondicionales:	**1 de cada 3**
Saltos tomados:	**5 de cada 6**
Saltos condicionales:	**3 de cada 4**
Saltos incondicionales:	**Todos**

Conclusión

1 de cada 6 instrucciones es un salto tomado;

1 de cada 8 instrucciones es un salto condicional;

1 de cada 10 instrucciones es un salto condicional y tomado;



La microarquitectura del procesador

Riesgos de control

Cuando se detecta un instrucción de salto se debe determinar el tipo de salto a ejecutar, la informacion necesaria para ejecutarlo y la direccion del salto.

Cuando se detecta un **salto incondicional** debe determinarse la fuente de informacion necesaria para calcular la direccion final del salto, ya que el salto se toma siempre.

Cuando se detecta un **salto condicional** primero debe determinarse si el salto se ejecuta o no y luego si se ejecuta la fuente de informacion necesaria para calcular la direccion final del salto.

La fuente de informacion necesaria para calcular la direccion final del salto se puede encontrar en

- La instrucción misma; o
- Se obtiene componiendo informacion disponibles en algun registro y parte de la instrucción.

En promedio

Instrucciones de salto: **1 de cada 5**

Saltos condicionales: **2 de cada 3**

Saltos incondicionales: **1 de cada 3**

Saltos tomados: **5 de cada 6**

Saltos condicionales: **3 de cada 4**

Saltos incondicionales: **Todos**

Cuando se **dispone de toda la información**, el salto puede **ejecutarse durante la etapa de búsqueda** de modo de **acelerar su ejecución**.

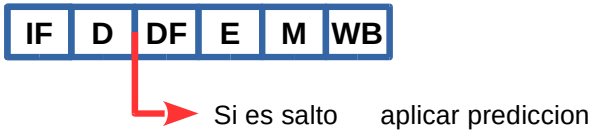
Cuando **no se dispone** de toda la información, el salto **no puede ejecutarse** hasta disponer de la información necesaria de modo que la ejecución **se retraza hasta la etapa de ejecución**.

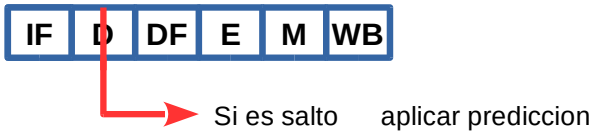
La microarquitectura del procesador

Riesgos de control

El problema es que no se conoce el tipo de instrucción hasta que finaliza la etapa de decodificacion (Decode), lo que introduce un retardo en la deteccion del salto.

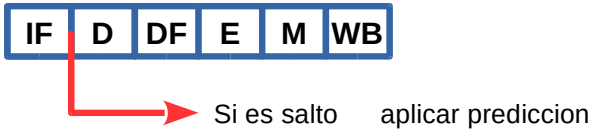
Hay tres alternativas

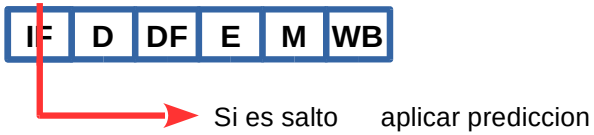




Deteccion en paralelo con la decodificacion: utiliza un decodificador de saltos dedicado para detectar las instrucciones de salto antes del final de la etapa de decodificacion;

Deteccion despues de la busqueda: detecta las instrucciones de salto en el buffer de instrucciones antes de que sean decodificadas; y





Deteccion durante la busqueda: detecta las instrucciones de salto al tiempo que se leen la memoria de instrucciones

La microarquitectura del procesador

Riesgos de control

Una primera aproximación al problema consiste en separar la ejecución de los saltos,

- Ejecutar los saltos con información completa en la **etapa de búsqueda de instrucción**; y
- Ejecutar los saltos con **informción incompleta** en la **etapa de búsqueda de datos**, permitir al proceesador continuar la búsqueda de instrucciones y vaciar el dataptah si el salto es tomado.

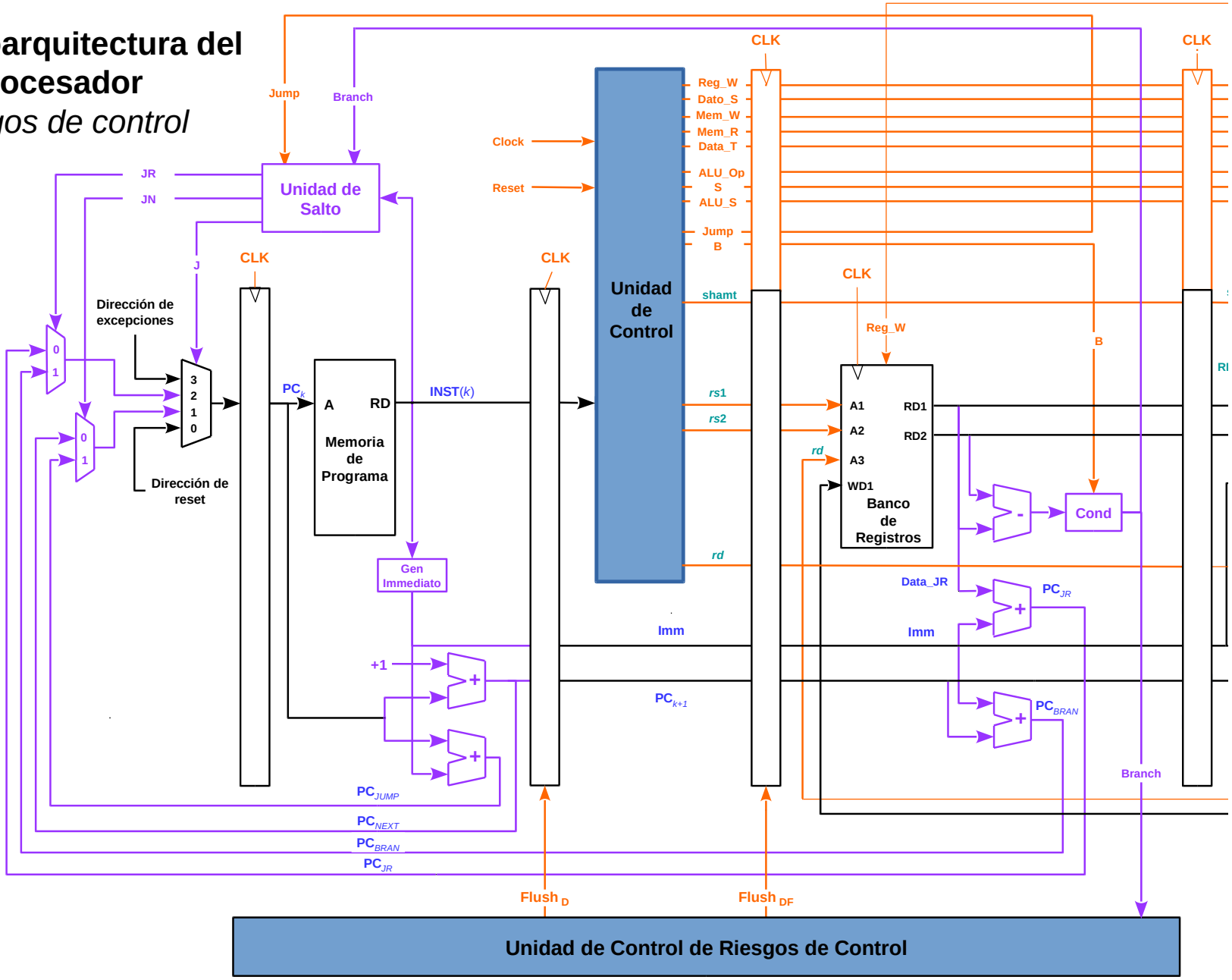
Esta idea implica intrroducir una **unidad de salto** que ejecute las instrucciones relacionadas con el primer grupo.

La unidad de salto implementa las siguientes regla para determinar el valor del contador de programa

```
IF Jump = 00          THEN J = 00
IF Opcode = BEQ AND rs1 = rs1 THEN J = 01; JN = 1
IF Opcode = JAL       THEN J = 01; JN = 1
IF Opcode = AUIPC THEN J = 01; JN = 1
IF Opcode = JALR   THEN J = 10; JR = 0
IF Opcode = BRANCH AND Branch = 1 THEN J = 10; JR = 1
IF Jump = 11          THEN J = 11
ELSE
        J = 01; JN = 0;
END
```

La microarquitectura del procesador

Riesgos de control



La microarquitectura del procesador

Riesgos de control – Predicción de saltos

Cuando se detecta un instrucción de salto se debe determinar el tipo de salto a ejecutar, la informacion necesaria para ejecutarlo y la direccion del salto.

Cuando se detecta un **salto incondicional** debe determinarse la fuente de informacion necesaria para calcular la direccion final del salto, ya que el salto se toma siempre.

- La direccion final se encuentra en la instrucción misma (**salto absoluto**); o
- La direccion fina se obtiene componiendo informacion disponibles en algun registro y parte de la instrucción (**salto relativo**)

Cuando se detecta un **salto condicional sin resolver** deben determinarse i) el camino del salto y ii) la fuente de informacion necesaria para calcular la direccion final del salto. El camino a seguir depende de la estructura del lazo considerado.

- Si se **predice como tomado** se calcula la direccion de destino y la **ejecucion continua de forma especulativa** a partir de dicha direccion; o
- Si se **predice como no tomado** la **ejecucion continua de forma especulativa.**

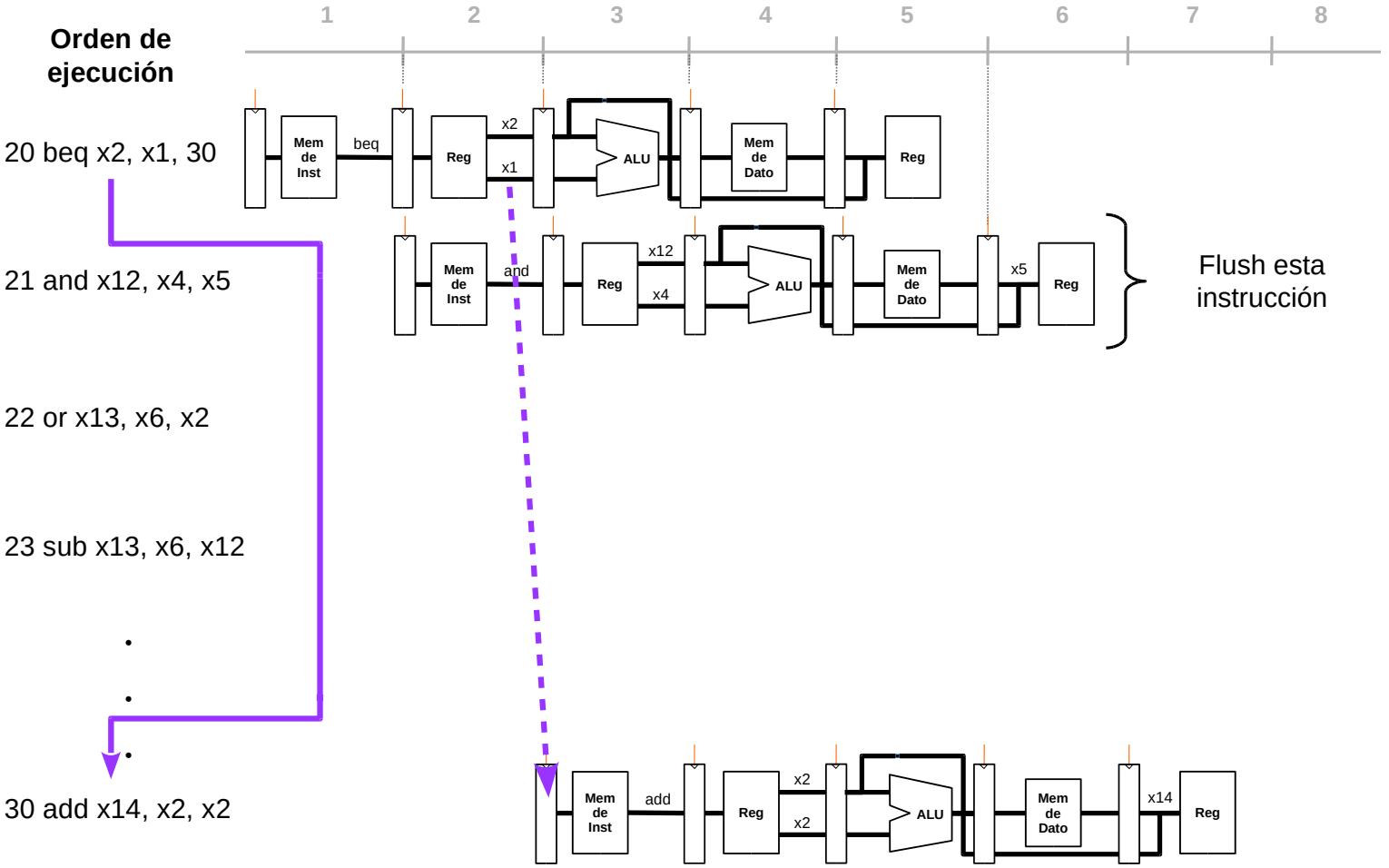
Cuando se **resuelve la condicion del salto**

- Si la prediccion **fue correcta** se confirma la ejecucion y **continua normalmenta**;
- Si la prediccion **no fue correcta se descartan** las instrucciones ejecutadas especulativamente (*flushing*) y se reanuda la ejecucion por el camino correcto.

La microarquitectura del procesador

Riesgos de control – Predicción de saltos

La solución al riesgo generado por un salto condicional consiste en **la ejecución anticipada del salto condicional** (*branch prediction*).



La microarquitectura del procesador

Riesgos de control – Prediccion de direccion

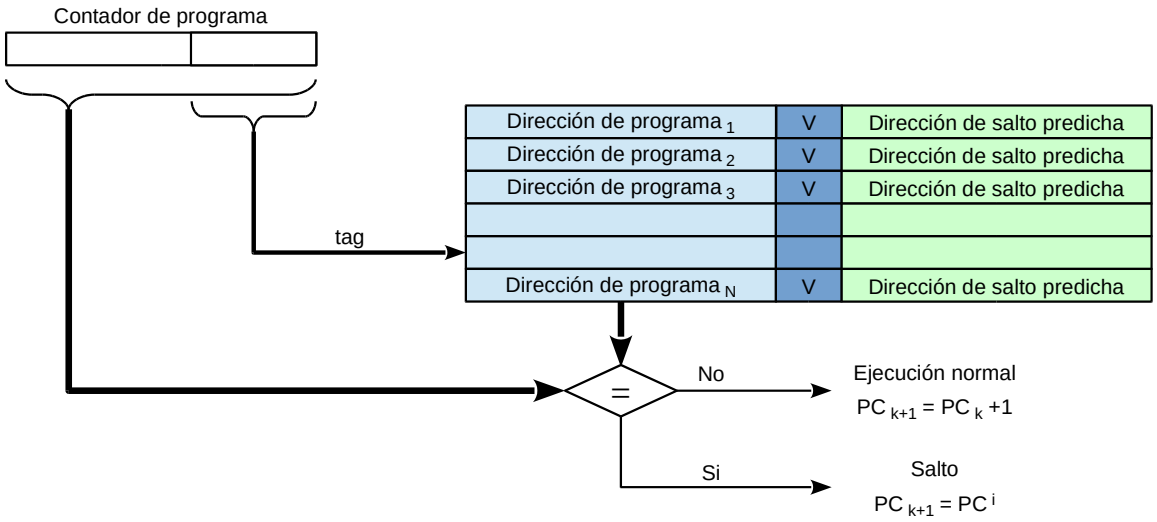
Cuando se detecta una instrucción de salto se debe determinar **la informacion necesaria para calcular la direccion final** del salto.

En ciertas instrucciones la direccion final se **encuentra en la instrucción misma** por lo que la instrucción puede ejecutarse inmediatamente.

En otras, la direccion final se obtiene **componiendo informacion disponible en un registro y parte de la instrucción**. La información del registro no se halla disponible hasta la etapa de búsqueda de datos (data fetch), lo cual retraza la ejecución de la instrucción al menos dos ciclos.

Para ello se introduce un **buffer de destino de saltos** (Branch Traget Buffer – BTB –), que almacena la dirección de destino del salto. La BTB is una memoria de doble puerto que en cada ciclo

- Lee los bits de predicción asociados a una dirección en la etapa de búsqueda;
- Actualiza los bits de predicción asociados a una dirección en la etapa de decodificación; y
- Escribe los bits de predicción asociados a una dirección que no esta en la tabla.



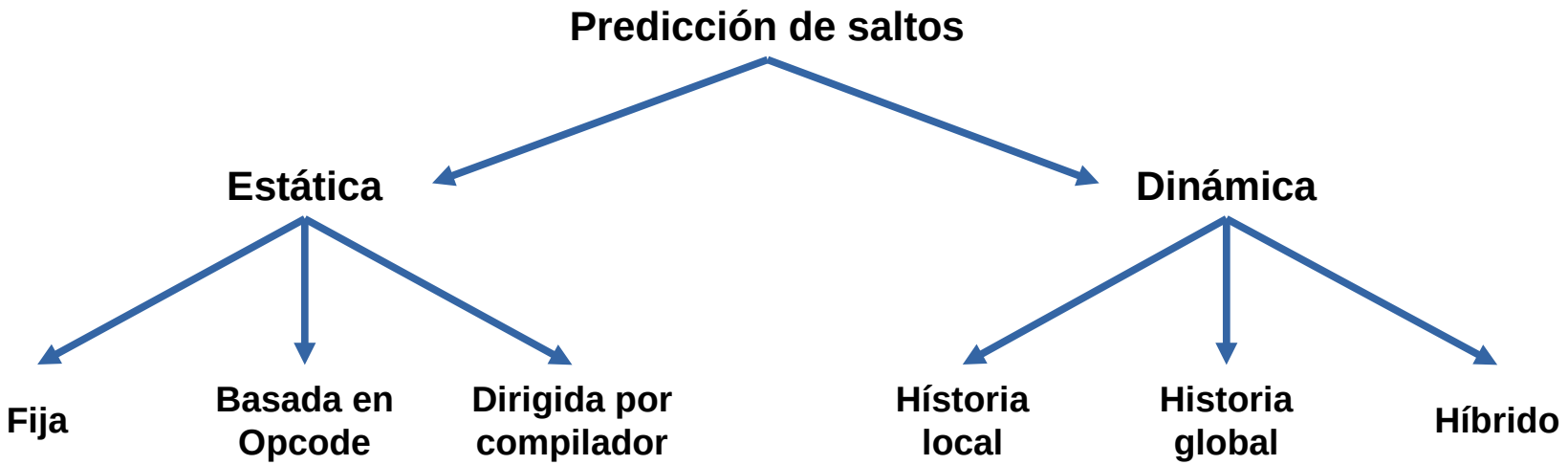
La microarquitectura del procesador

Riesgos de control – Predicción de saltos

Sin la predicción de saltos, el procesador tendría que esperar hasta que la instrucción de salto condicional haya pasado la etapa de ejecución antes de que la siguiente instrucción pueda ingresar a la etapa de búsqueda. El predictor de salto intenta evitar esta pérdida de tiempo al tratar de predecir si es más probable que se tome o no el salto condicional.

El tiempo que se pierde en el caso de una predicción errónea es igual al número de etapas en el procesador desde la etapa de búsqueda hasta la ejecución. Los microprocesadores modernos tienden a tener arquitecturas bastante largas lo que aumenta la necesidad de un predictor de saltos más avanzado.

Las tecnicas de prediccion de saltos se clasifican en



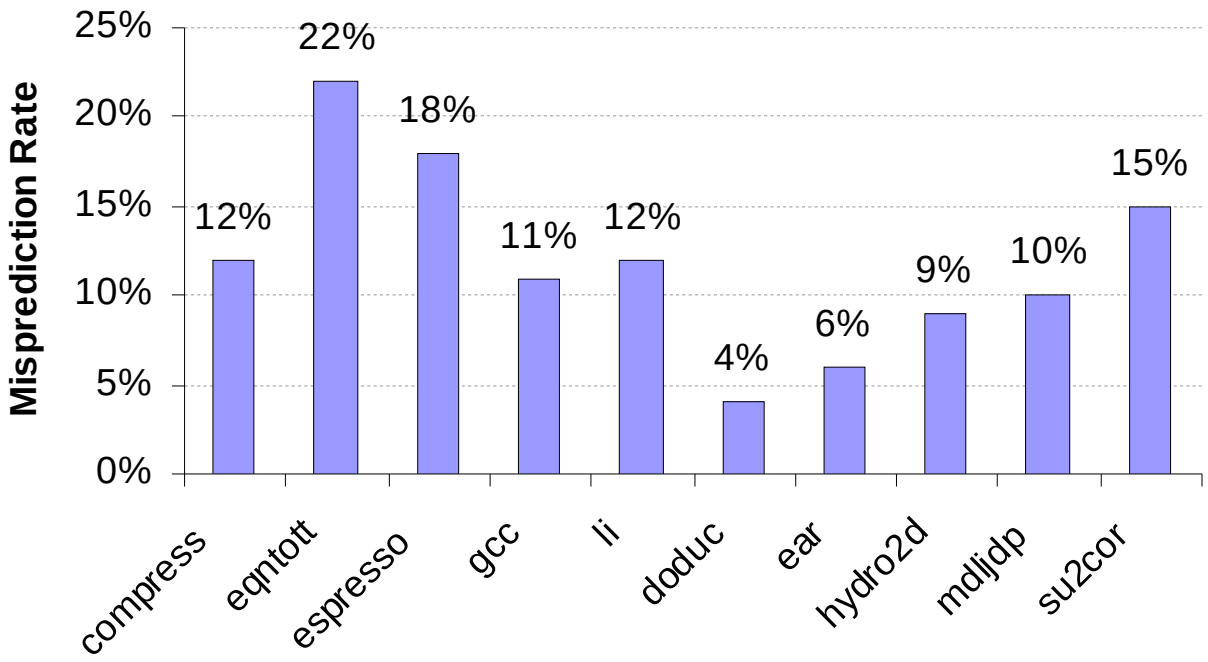
La microarquitectura del procesador

Riesgos de control – Predicción de saltos

Los algoritmos de predicción de saltos funcionan porque los algoritmos, y los datos utilizados por los mismos, presentan regularidades que pueden ser detectadas por los algoritmos de predicción.

Además, las secuencia de instrucciones tiene redundancias que son artefactos de manera en que los humanos / compiladores piensan acerca de los problemas.

Sin embargo, hay un **pequeño número de saltos** en programas que determinan el **comportamiento dinámico del programa**.



La microarquitectura del procesador

Riesgos de control – Predicción de saltos

A partir de un análisis estadístico del uso de intrucciones de saltos en programas surge que

En promedio

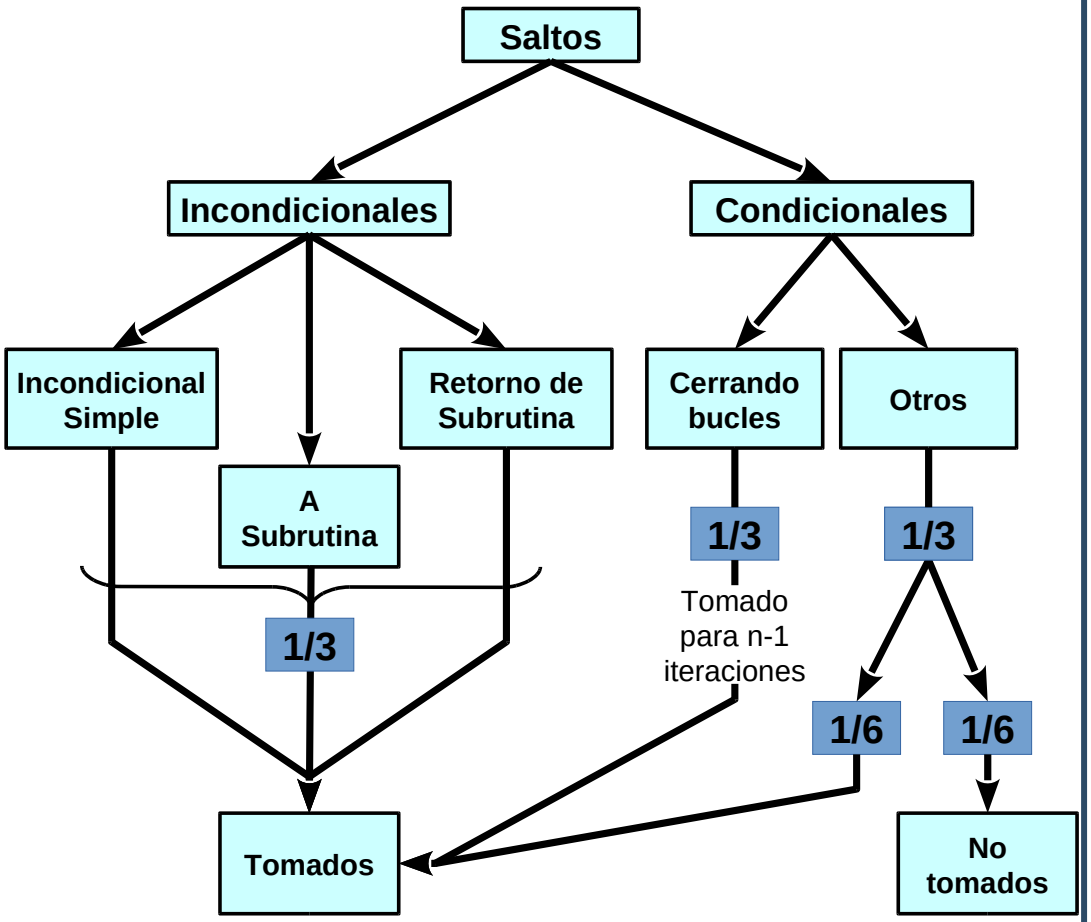
Instrucciones de salto:	**1 de cada 5**
Saltos condicionales:	**2 de cada 3**
Saltos incondicionales:	**1 de cada 3**
Saltos tomados:	**5 de cada 6**
Saltos condicionales:	**3 de cada 4**
Saltos incondicionales:	**Todos**

Conclusión

1 de cada 6 instrucciones es un salto tomado;

1 de cada 8 instrucciones es un salto condicional;

1 de cada 10 instrucciones es un salto condicional y tomado;



La microarquitectura del procesador

Riesgos de control – Predicción fija

La predicción estática es la técnica más sencilla que no utiliza información sobre el historial dinámico de ejecución del código, sino que predice el resultado basándose sólo en la instrucción.

Hay dos posibilidades:

Se ejecuta Cada instrucción de salto se ejecuta;
Tiene un número mayor de aciertos; pero
El hardware necesario para implementarlo es más complejo.

No se ejecuta Cada instrucción de salto no se ejecuta;
Tiene un número menor de aciertos; pero
El hardware necesario para implementarlo es más sencillo.

La predicción estática se basa en la dirección del salto a ejecutar;

- **Se ejecuta siempre** si la dirección destino es menor que la que se encuentra ela instrucción de salto (*salto hacia atrás*). Este tipo de salto generalmente se **utiliza en bucles**.
- **No se ejecuta siempre** si la dirección destino es mayor que la que se encuentra ela instrucción de salto (*salto hacia adelante*). Este tipo de salto generalmente se **utilizan en instrucciones IF – THEN - ELSE**.

Mal comportamiento en programas con pocos bucles y muchos IF-THEN-ELSE

La microarquitectura del procesador

Riesgos de control – Predicción fija

Estas ideas se implementan a partir de las siguientes reglas

- Si el salto es condicional y su desplazamiento es negativo **se toma**; y
- Si el salto es condicional y su desplazamiento es positivo **no se toma**.

La implementación de la **unidad de salto** ya incluye a los saltos incondicionales por lo que sólo falta incorporar los saltos condicionales con información completa a partir de modificar la regla de saltos relativos incondicionales.

```
IF Jump = 00 THEN J = 00
IF Opcode = BEQ AND rs1 = rs1 THEN J = 01; JN = 1
IF Opcode = JAL THEN J = 01; JN = 1
IF Opcode = AUIPC THEN J = 01; JN = 1
IF Opcode = BEQ⋁BNE⋁BLT⋁BGE THEN J = 01; JN = 1
IF Jump = 01 AND Branch = 1 THEN J = 10; JR = 1
IF Jump = 10 AND Branch = 0 THEN J = 10; JR = 0
IF Jump = 11 THEN J = 11
ELSE
        J = 01; JN = 0;
END
```

Reset

Predictor estático

Excepción

Ejecución normal

La microarquitectura del procesador

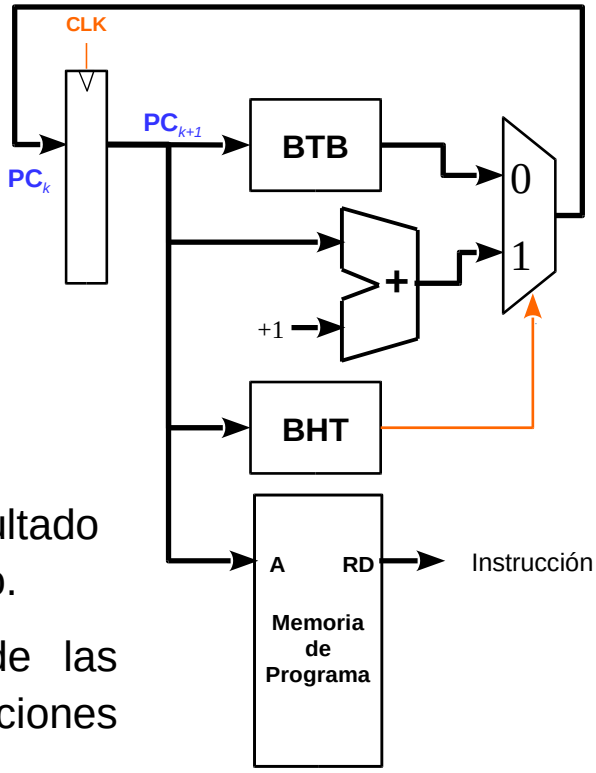
Riesgos de control – Predicción dinámica

Un predictor dinámico trabaja en tiempo de ejecución intentando aprender el comportamiento del programa para predecir, **con la máxima tasa de aciertos**, si un salto será o no tomado.

La información que se intercambia entre las etapas de búsqueda de instrucción, decodificación y ejecución se almacena junto con el resultado en la **tabla de historia de saltos** (Branch History Table – BHT -) que está direccionada por los bits mas bajos del contador de programa.

La BHT es una memoria de doble puerto que:

- Lee la dirección de destino asociada a una dirección en la etapa de búsqueda;
- Actualiza la dirección de destino asociada a una dirección en la etapa de decodificación; y
- Escribe la dirección de destino asociad a una dirección que no esta en la tabla.



Para ejecutar un salto se lee la BHT para ver el resultado esperado, se comienza a leer instrucciones a partir del mismo.

Si la decision es incorrecta, el datapath es vaciado de las instrucciones incorrectas y reicinicializado con las instrucciones correctas, y la informacion de la BHT actualizada.

La microarquitectura del procesador

Riesgos de control – Predicción dinámica

Un predictor de un bit solo registra el resultado del último salto, indicando el comportamiento de la última ejecución de la instrucción de salto.

Dado que en un bit sólo almacena el estado del salto anterior, su actualización es muy violenta, cambiando la predicción de un salto sólo por un comportamiento puntual.

Predicción

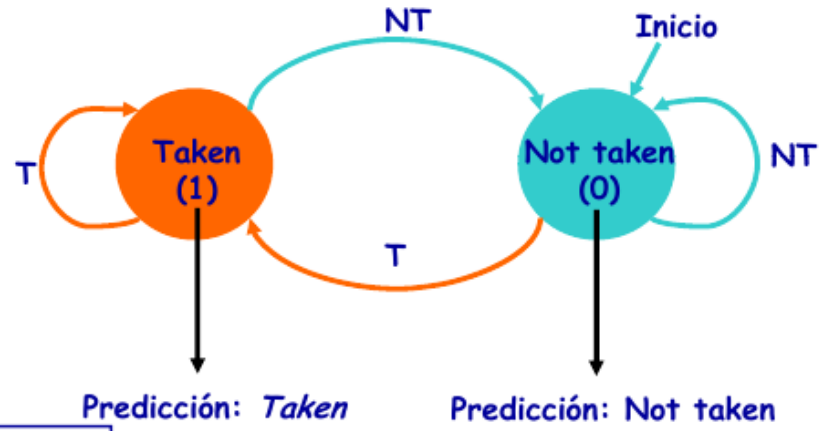
- El salto se predice como *Taken* si en la última ejecución fue tomado
- El salto se predice como *Not Taken* si en la última ejecución no fue tomado

FUNCIONAMIENTO

- *Máquina de dos estados:*
  - Not taken (0)
  - Taken (1)
- *Registro de historia*
  - Contador saturado de 1 bit
- *Predicción*
  - Valor del registro de historia

LIMITACIÓN

- Sólo se registra el comportamiento de la última ejecución del salto
- Dos malas predicciones en los cambios

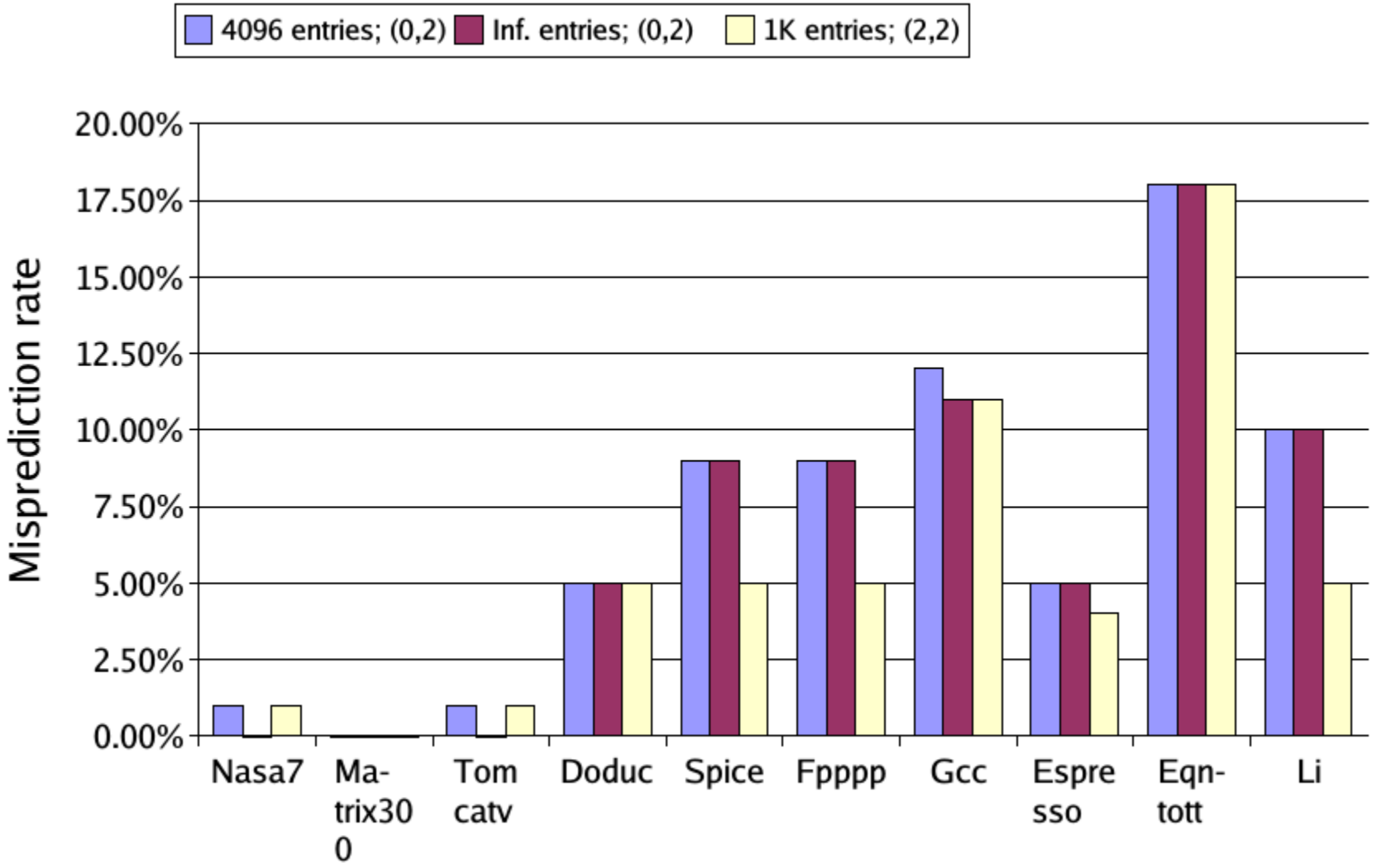


Más Bits

Cambios de estado:
T: el salto ha sido tomado
NT: el salto no ha sido tomado

La microarquitectura del procesador

Riesgos de control – Predicción dinámica



La microarquitectura del procesador

Riesgos de control – Predicción dinámica

Un predictor de dos bits solo registra el resultado de los últimos cuatro saltos a través de ua máquina de estados, indicando el comportamiento de las últimas instrucciones de salto.

Dado que almacena el estado de varios saltos, su actualización es suave y consigue mayor número de aciertos.

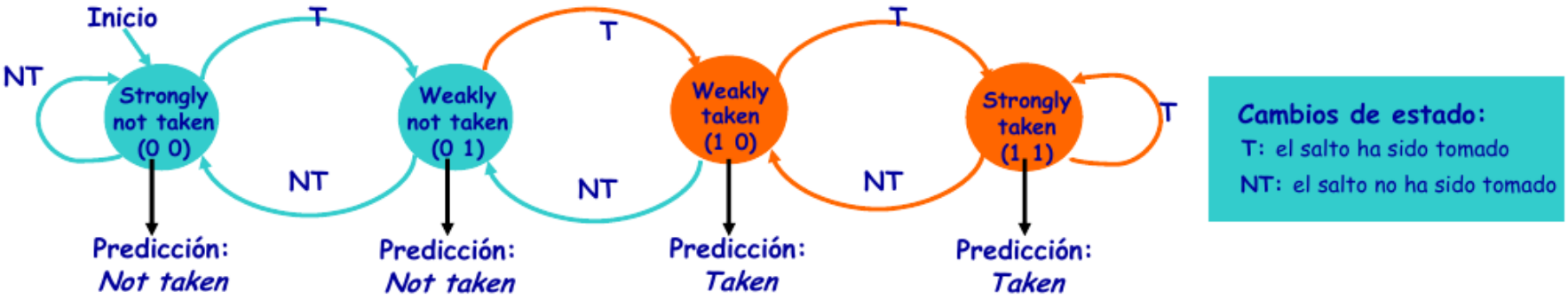
Predicción

- Un salto que se toma repetidamente se predice como *Taken*
- Un salto que no se toma repetidamente se predice como *Not taken*
- Si un salto toma una dirección inusual una sola vez, el predictor mantiene la predicción usual

Funcionamiento

Máquina de cuatro estados:
- Strongly not taken (00)
- Weakly not taken (01)
- Weakly taken (10)
- Strongly taken (11)

- *Registro de historia*
  - Contador saturado de 2 bits
- *Predicción*
  - bit más significativo del registro de historia



La microarquitectura del procesador

Riesgos de control – Predicción dinámica

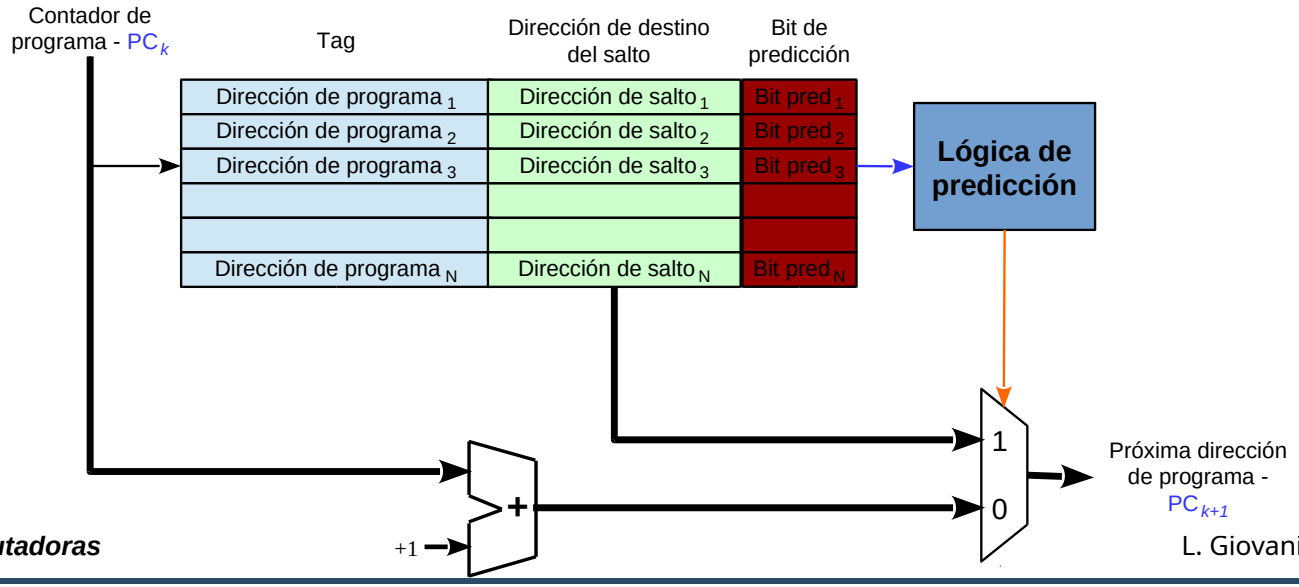
Una posible implementación integra el buffer de destino de saltos (BTB) con los bits de predicción en una única tabla.

Los campos de la tabla dirección de destino y bits de historia se actualizan después de la ejecución de los saltos.

Su principal desventaja es que solo se pueden predecir los saltos que están en la tabla

***Predicción Implícita** (sin bits de predicción)*

Aplicable con un sólo bit de predicción
- Sí la instrucción de salto está en la BTB
  ⇒ El salto se predice como tomado
- Sí la instrucción de salto no está en la BTB
  ⇒ El salto se predice como no tomado



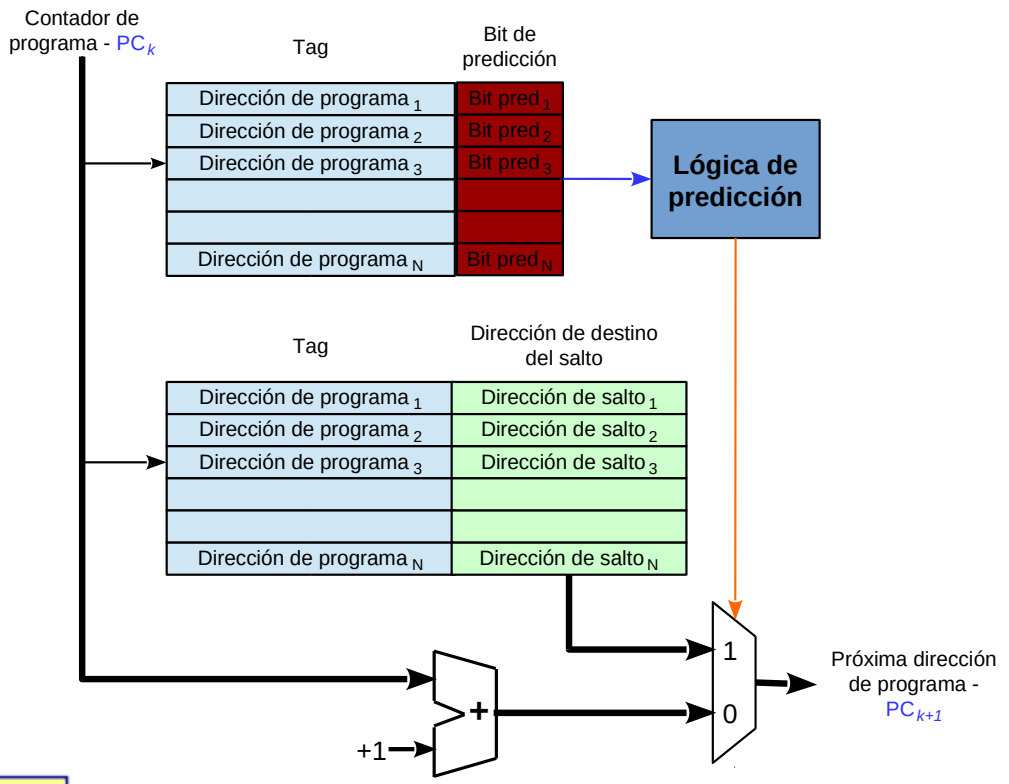
La microarquitectura del procesador

Riesgos de control – Predicción dinámica

Otra implementación consiste en desacoplar el buffer de destino de saltos con los bits de predicción en una única tabla.

La tabla de historia de saltos (BHT) contiene información sobre el comportamiento de los saltos mientras que la tabla de destino de saltos (BTB) sobre los saltos (direcciones de origen y destino).

Su principal ventaja es que se pueden predecir los saltos que no están en BTB (saltos cuya información está en la instruccion).



- Usando los bits menos significativos de la dirección
  - Sin TAGs ⇒ Menor coste (opción + habitual)
  - Compartición de entradas
    ⇒ Se degrada el rendimiento
- Asociativa por conjuntos
  - Mayor coste ⇒ Tablas pequeñas
  - Para un mismo coste hardware
    ⇒ Peor comportamiento

Mientras que su principla deseventaja es la cantidad de hardware necesario para implementarlas.

La microarquitectura del procesador

Riesgos de control – Predicción dinámica

Los errores de **predicción del salto** se detectan en la etapa de búsqueda de datos por lo que se be vaciar las etapas anteriores (decodificación y búsqueda de instrucción) y **corregir el contador de programa** utilizando la dirección calculada en esta etapa y la **tabla de historia de saltos** (BHT) .

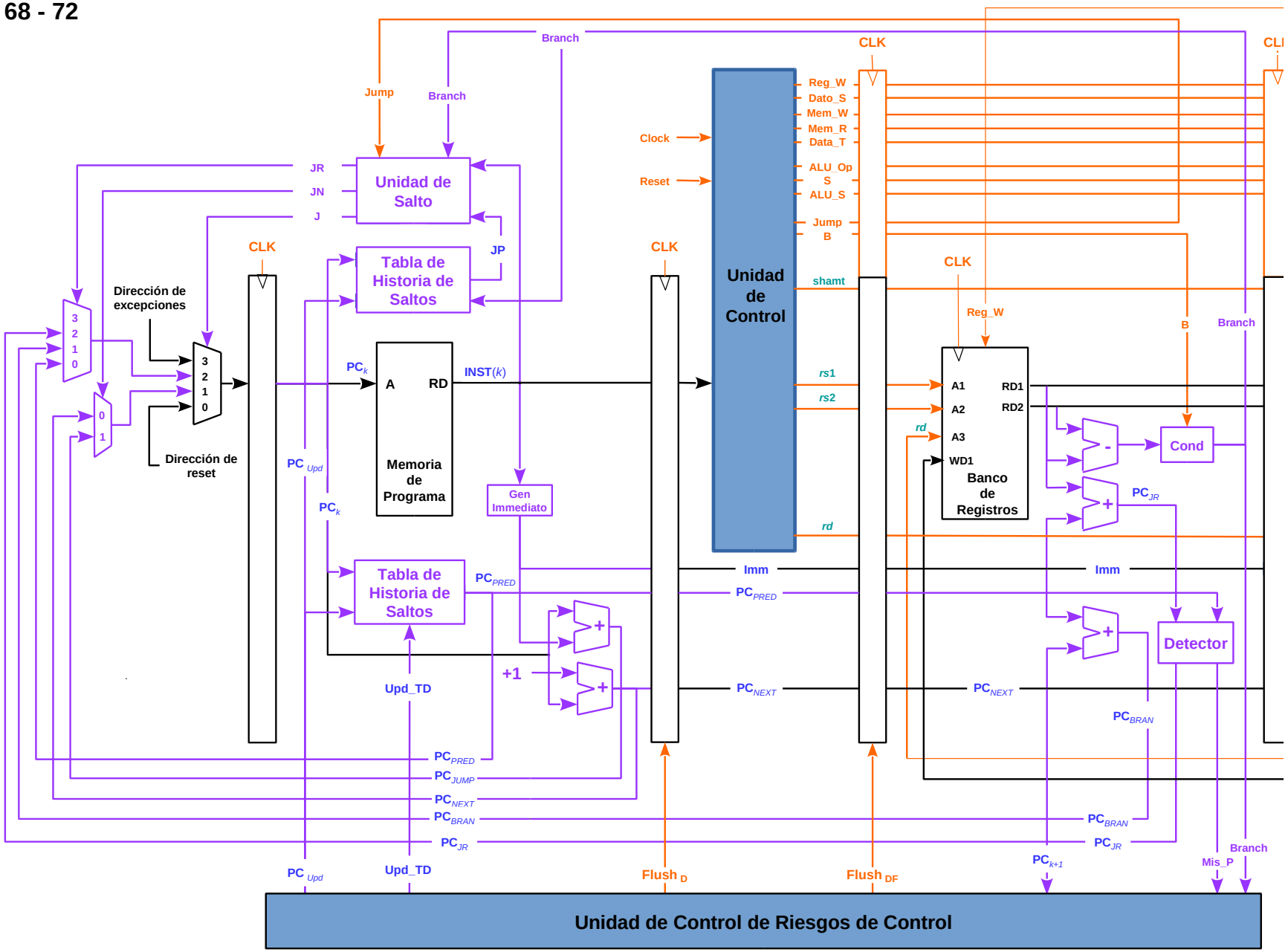
Si la **predicción del salto es correcta**, debe verificarse que la dirección de destino es correcta también. Hay dos posibles situaciones:

- La dirección es correcta se continua con la ejecución sin modificaciones.
- La dirección es incorrecta se vacia el datapath, se corrige el contador de programa y la información de la tabla.

La correción de la dirección de destino se puede implementar de dos formas:

Opción 1: Detectar el error en la dirección destino en la **etapa de búsqueda de instrucción** permite corregir la ejecución y evitar pérdidas de tiempo, a expensas del número de ciclos de ejecución de la instrucción. Esto se logra incorporando una lógica de comparación después de la BTB.

Opción 2: Detectar el error en la dirección destino en la **etapa de decodificación** permite mantener constante el número de ciclos de ejecución de la instrucción pero se debe corregir la ejecución a partir de burbujas en la etapa de búsqueda y restaurando el contador de programa con el valor calculado en la etapa de ejecución.



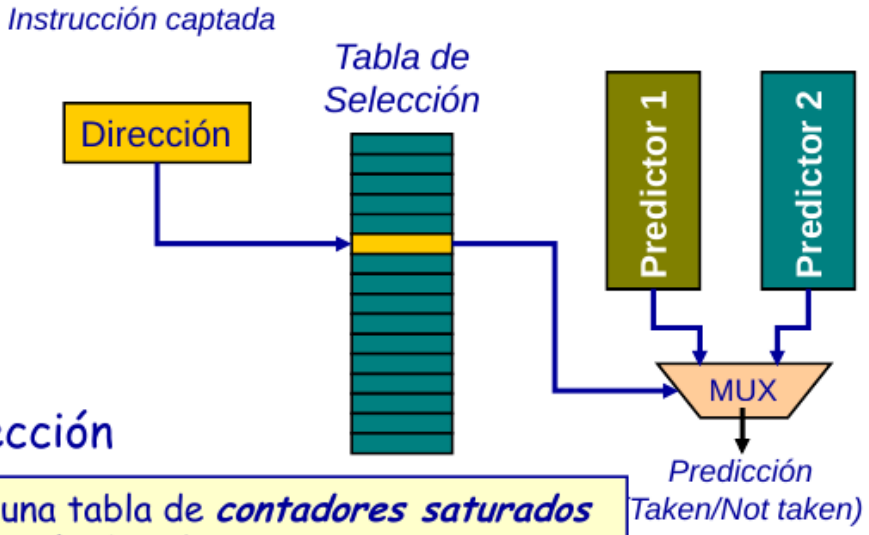
La microarquitectura del procesador

Riesgos de control – Predicción híbrida

- Cada uno de los predictores estudiados tiene sus ventajas y sus inconvenientes
- Combinando el uso de distintos predictores y aplicando uno o otro según convenga, se pueden obtener predicciones mucho más correctas

Predictor híbrido

Mezcla varios predictores y añade un mecanismo de selección del predictor



Mecanismo de selección

Elige, en cada caso, el predictor que haya dado mejores resultados hasta el momento

Implementación del mecanismo de selección

Para combinar dos predictores, P1 y P2, se utiliza una tabla de ***contadores saturados de dos bits*** indexada por la dirección de la instrucción de salto

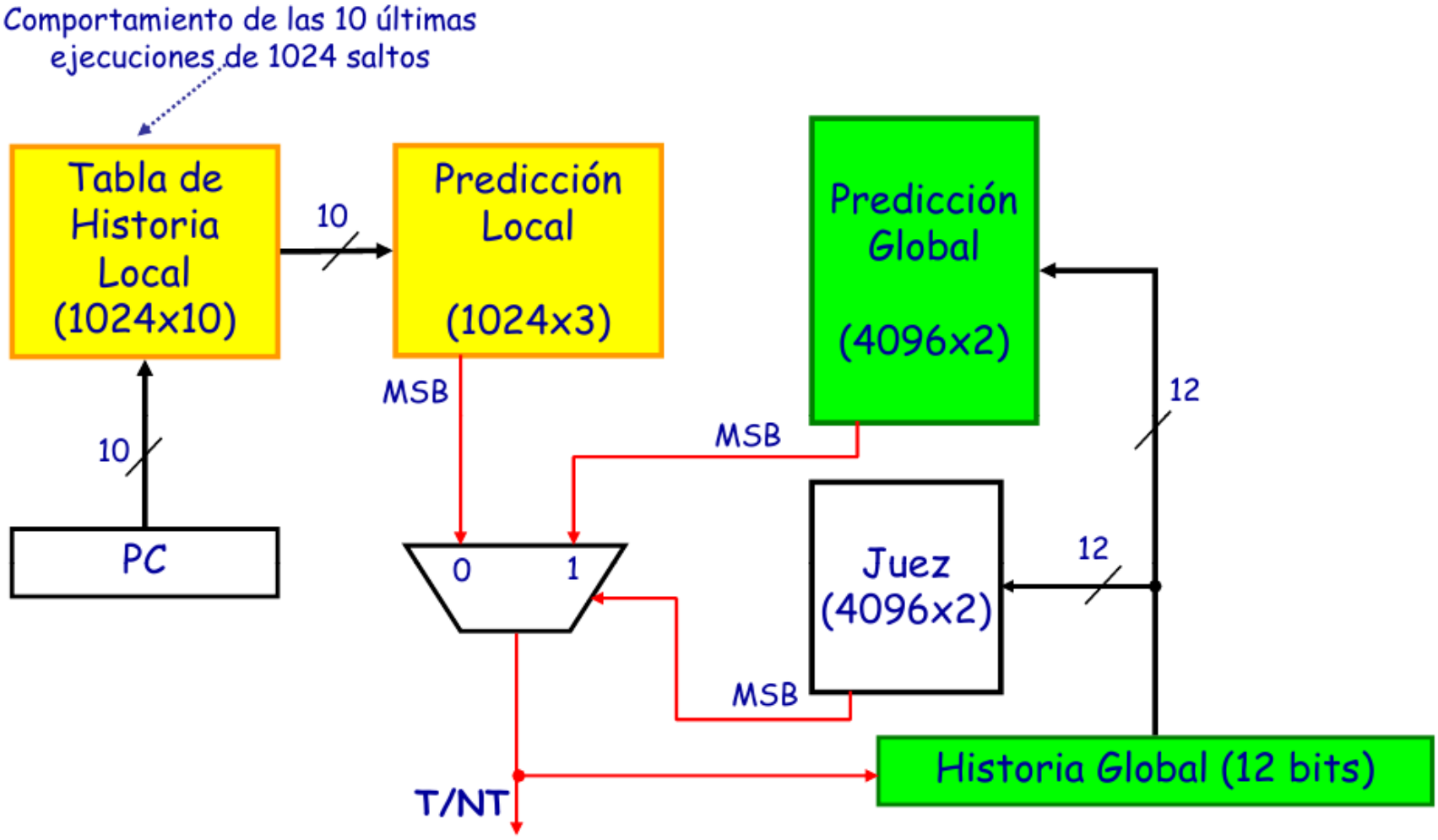
P1	P2	Actualiz. del contador
Fallo	Fallo	Cont no varía
Fallo	Acierto	Cont = Cont +1
Acierto	Fallo	Cont = Cont -1
Acierto	Acierto	Cont no varía

- Si P2 acierta más que P1 ⇒ *Cont* aumenta
- Si P1 acierta más que P2 ⇒ *Cont* disminuye

Bit más signif. del contador	Predictor seleccionado
0	P1
1	P2

La microarquitectura del procesador

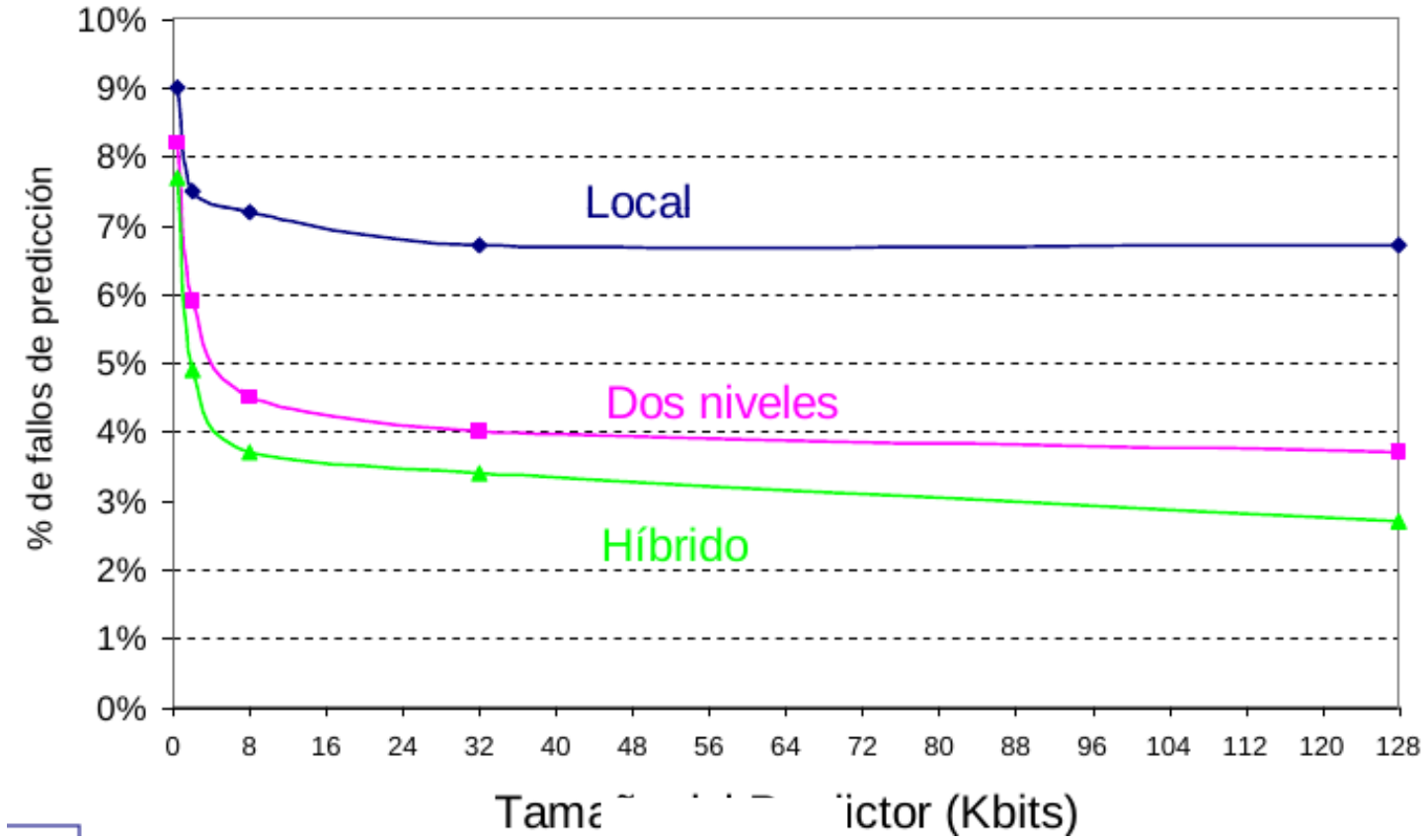
Riesgos de control – Predicción híbrida



La microarquitectura del procesador

Riesgos de control – Predicción híbrida

La ventaja del predictor híbrido es su capacidad de seleccionar el predictor correcto para cada salto, ya que pude ajustarse a las características de cada uno de ellos.



La microarquitectura del procesador

Implementación segmentada