

# Segmentación de cauce

Estructura de Computadores  
Semanas 10 y 11

## Bibliografía:

- |                 |   |
|-----------------|---|
| [HAM03] Cap.8   | Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 2003<br>Signatura ESIIT/ <a href="#">C.1 HAM org</a>                |
| [STA8] Sec.12.4 | Organización y Arquitectura de Computadores, 7ª Ed. Stallings. Pearson Educación, 2008.<br>Signatura ESIIT/ <a href="#">C.1 STA org</a> |

# Guía de trabajo autónomo (4h/s)

## ■ Lectura

- Cap.8 Hamacher
- Sección 12.4 Stallings

## Bibliografía:

[HAM03] Cap.8

Organización de Computadores. Hamacher, Vranesic, Zaki. McGraw-Hill 2003

Signatura ESIIT/[C.1 HAM org](#)

[STA8] Sec.12.4

Organización y Arquitectura de Computadores, 7ª Ed. Stallings. Pearson Educación, 2008.

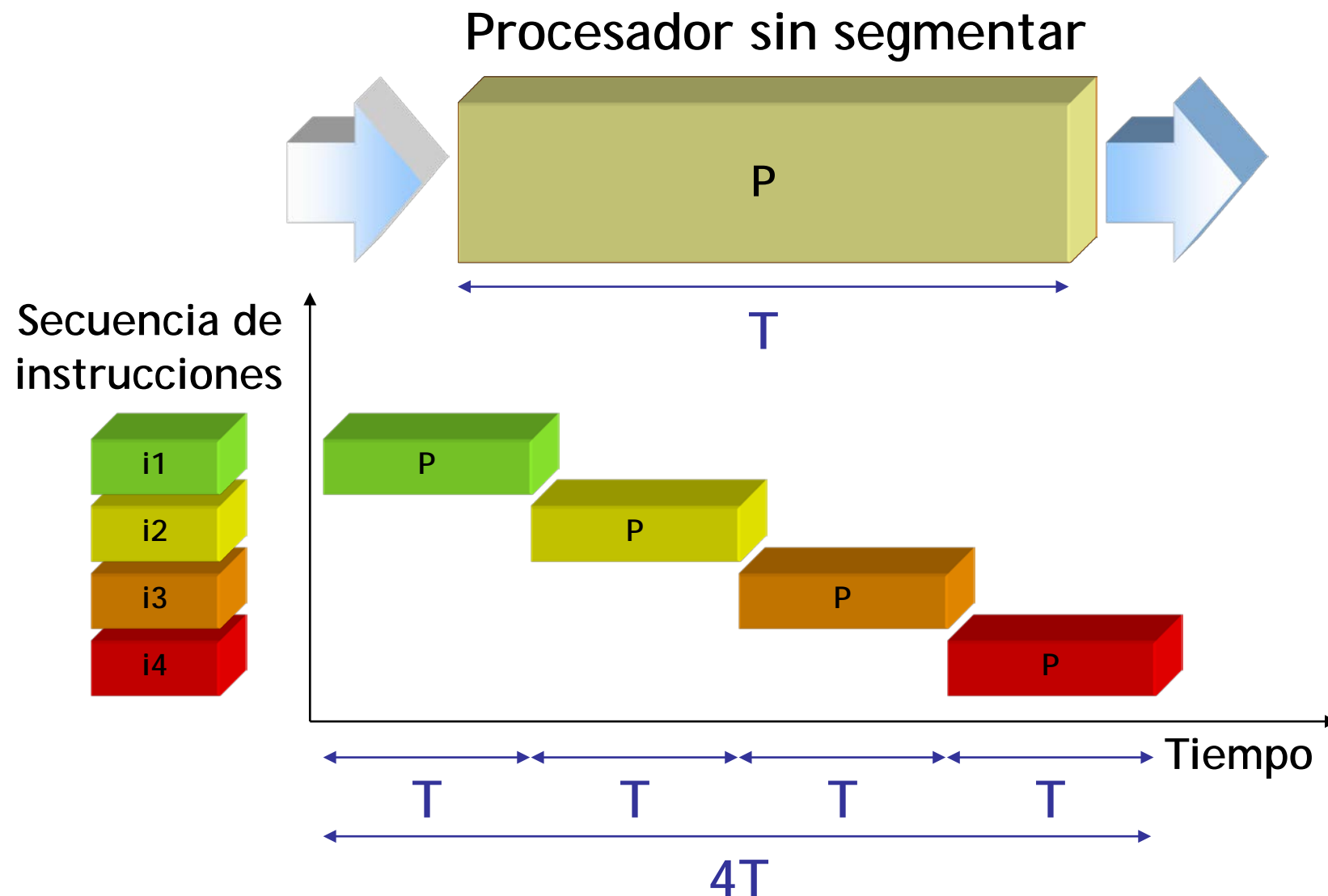
Signatura ESIIT/[C.1 STA org](#)

[

# Segmentación de cauce

- **Concepto de segmentación**
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

# Concepto de segmentación



# Concepto de segmentación

¿Es posible incrementar la velocidad de procesamiento, al margen de mejoras tecnológicas, sin incrementar el número de procesadores?

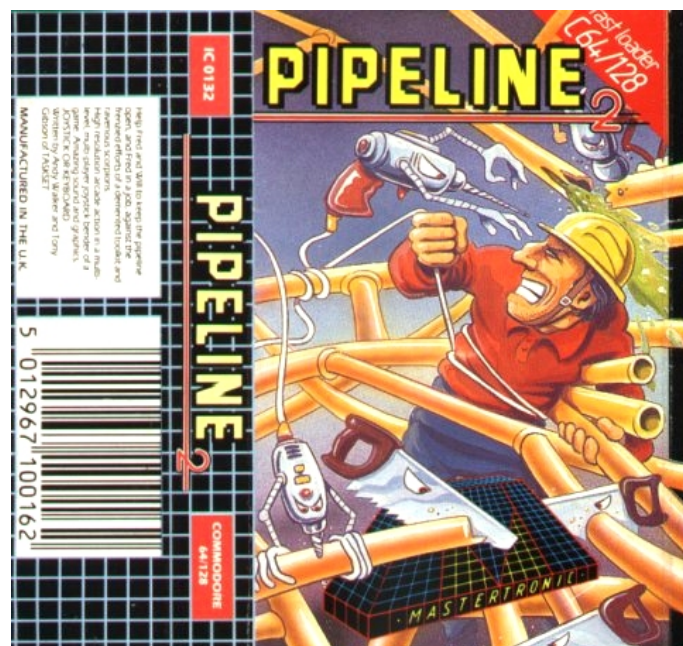


**SOLUCIÓN**



Segmentación de cauce  
(*pipelining*)

# Concepto de segmentación

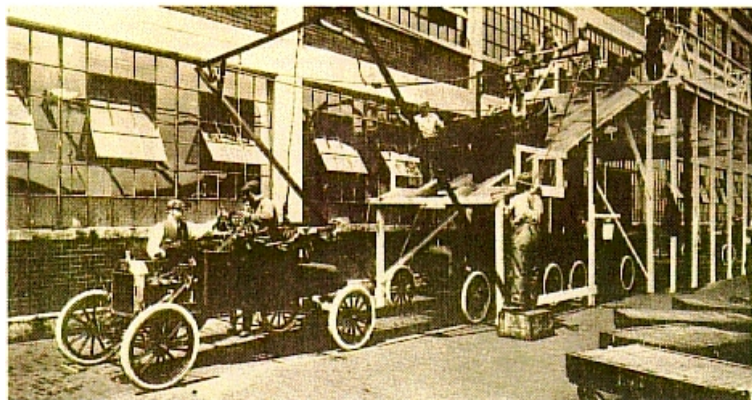


SOLUCIÓN

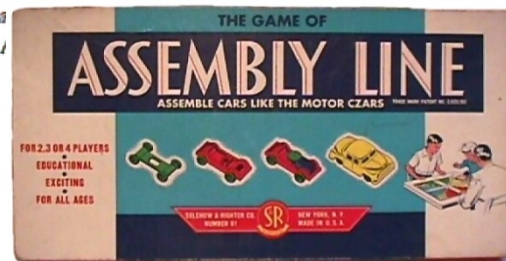
Segmentación de cauce  
(*pipelining*)



# Concepto de segmentación



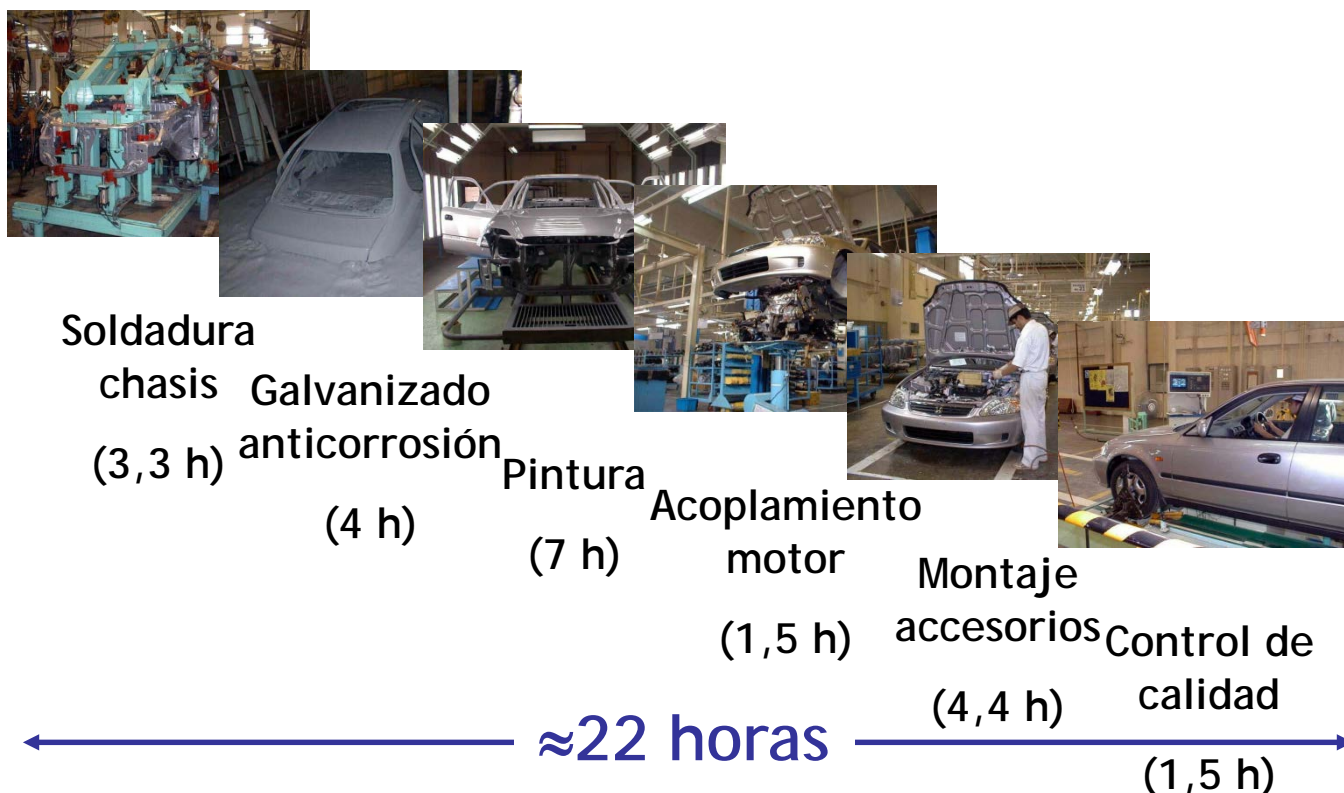
*Montaje final del vehículo del modelo T en la fábrica de Highland Park (Detroit, EUA), propiedad de Ford. En ella se ensambla grandes unidades constructivas (chasis y carrocería) que forman el vehículo.*



SEGMENTACIÓN EN UNA FÁBRICA DE AUTOMÓVILES

Cadena de montaje

# Concepto de segmentación



## SEGMENTACIÓN EN UNA FÁBRICA DE AUTOMÓVILES

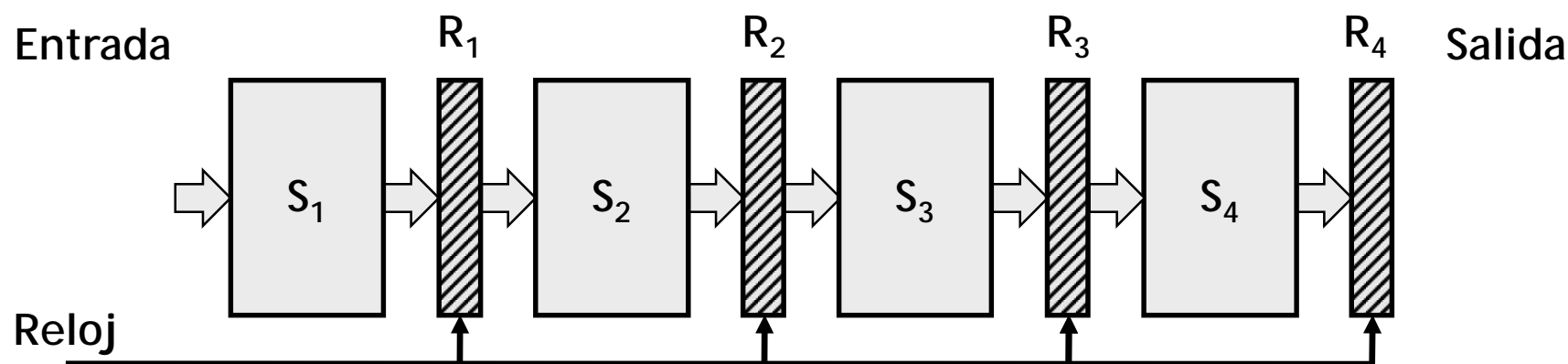
Subdividir el proceso en  $n$  etapas, permitiendo el solapamiento en la fabricación de automóviles



# Concepto de segmentación

$S_i$ : etapa de segmentación  $i$ -ésima

$R_i$ : registro de segmentación de la etapa  $i$ -ésima

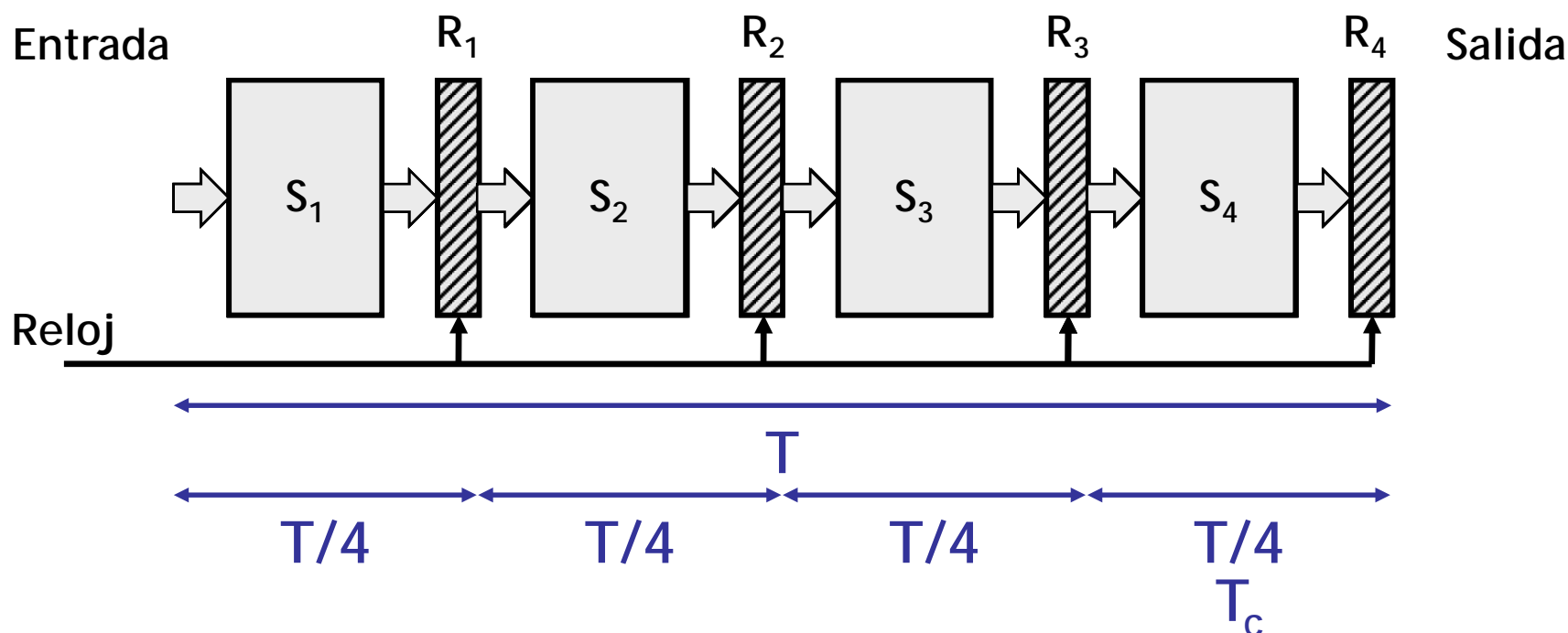


## SEGMENTACIÓN EN UN PROCESADOR

Subdividir el procesador en  $n$  etapas, permitiendo el solapamiento en la ejecución de instrucciones

# Concepto de segmentación

Las instrucciones entran por un extremo del cauce, son procesadas en distintas etapas y salen por el otro extremo.



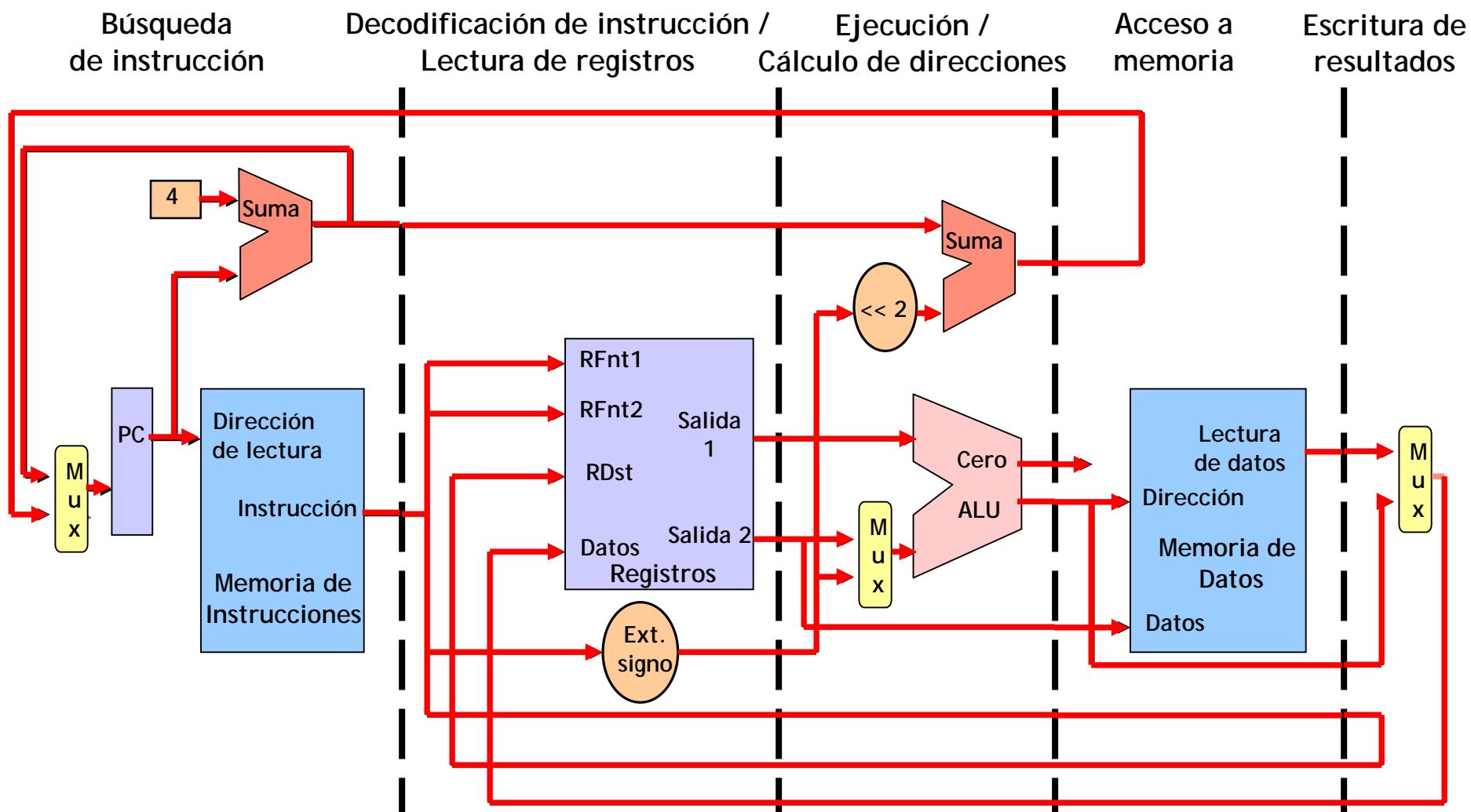
Cada instrucción individual se sigue ejecutando en un tiempo  $T$ ...

...pero hay varias instrucciones ejecutándose simultáneamente

# Segmentación de cauce

- Concepto de segmentación
- **Ejemplo de segmentación**
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

# Ejemplo de segmentación



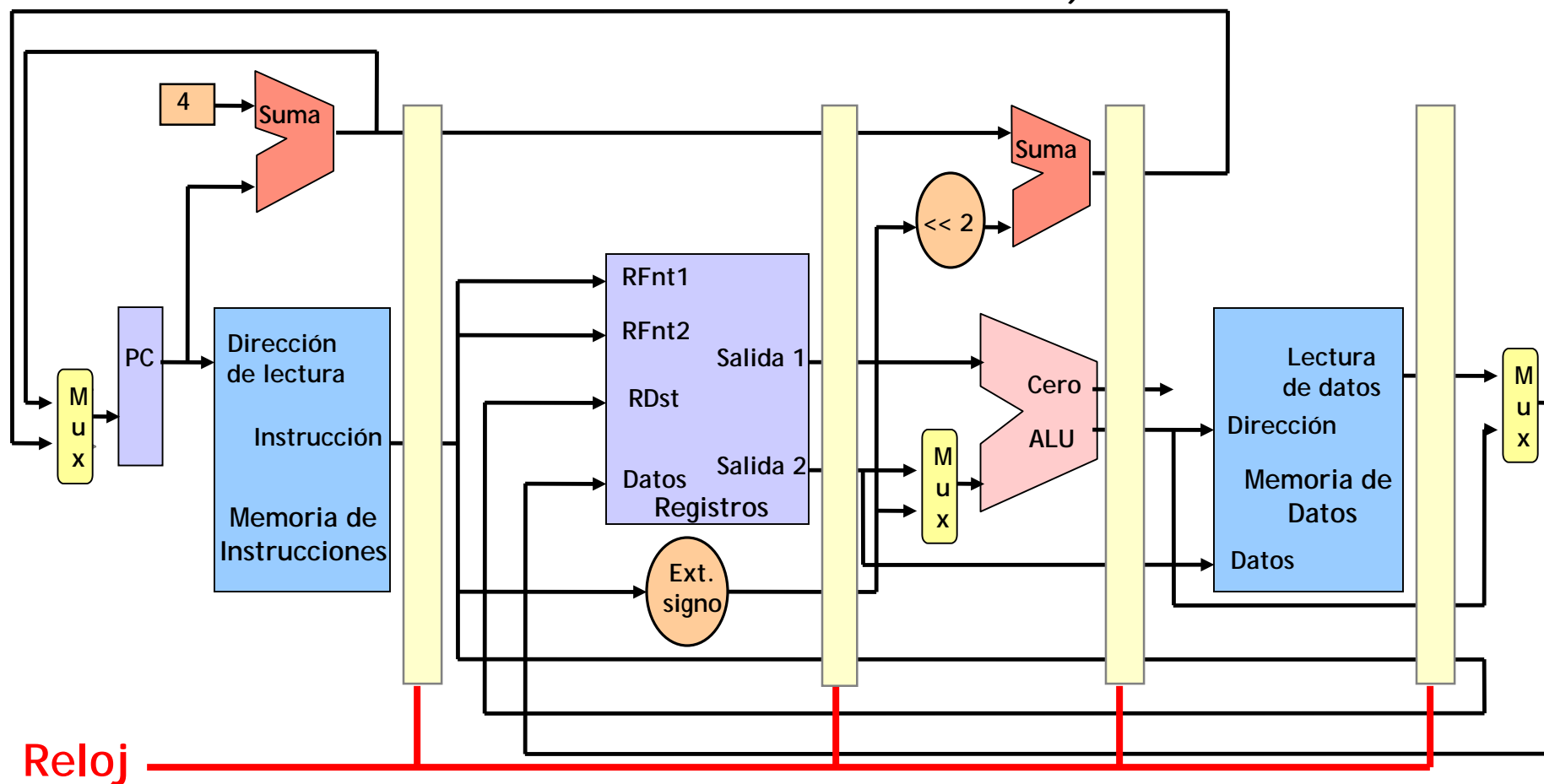
IF  
(Instruction Fetch)

ID  
(Instruction Decode and register fetch)

EX  
(EXecute and effective address calculation)

MEM  
(MEMory access)

WB  
(Write back)



# Ejemplo de segmentación

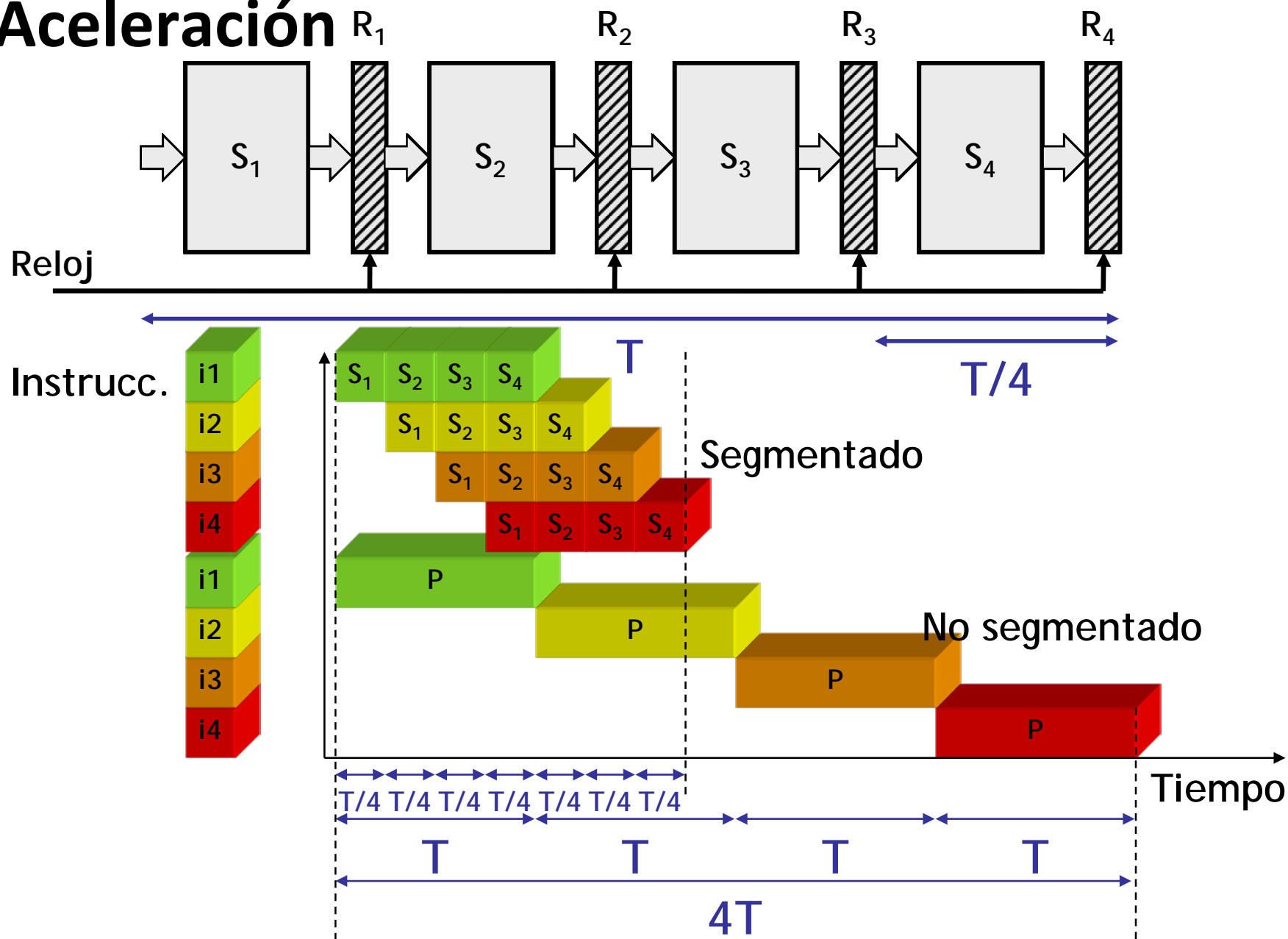
- Cada etapa del cauce debe completarse en un ciclo de reloj
- Fases de captación y de memoria
  - Si acceden a memoria principal, el acceso es varias veces más lento
  - La caché permite acceso en un único ciclo de reloj
- El periodo de reloj se escoge de acuerdo a la etapa más larga del cauce



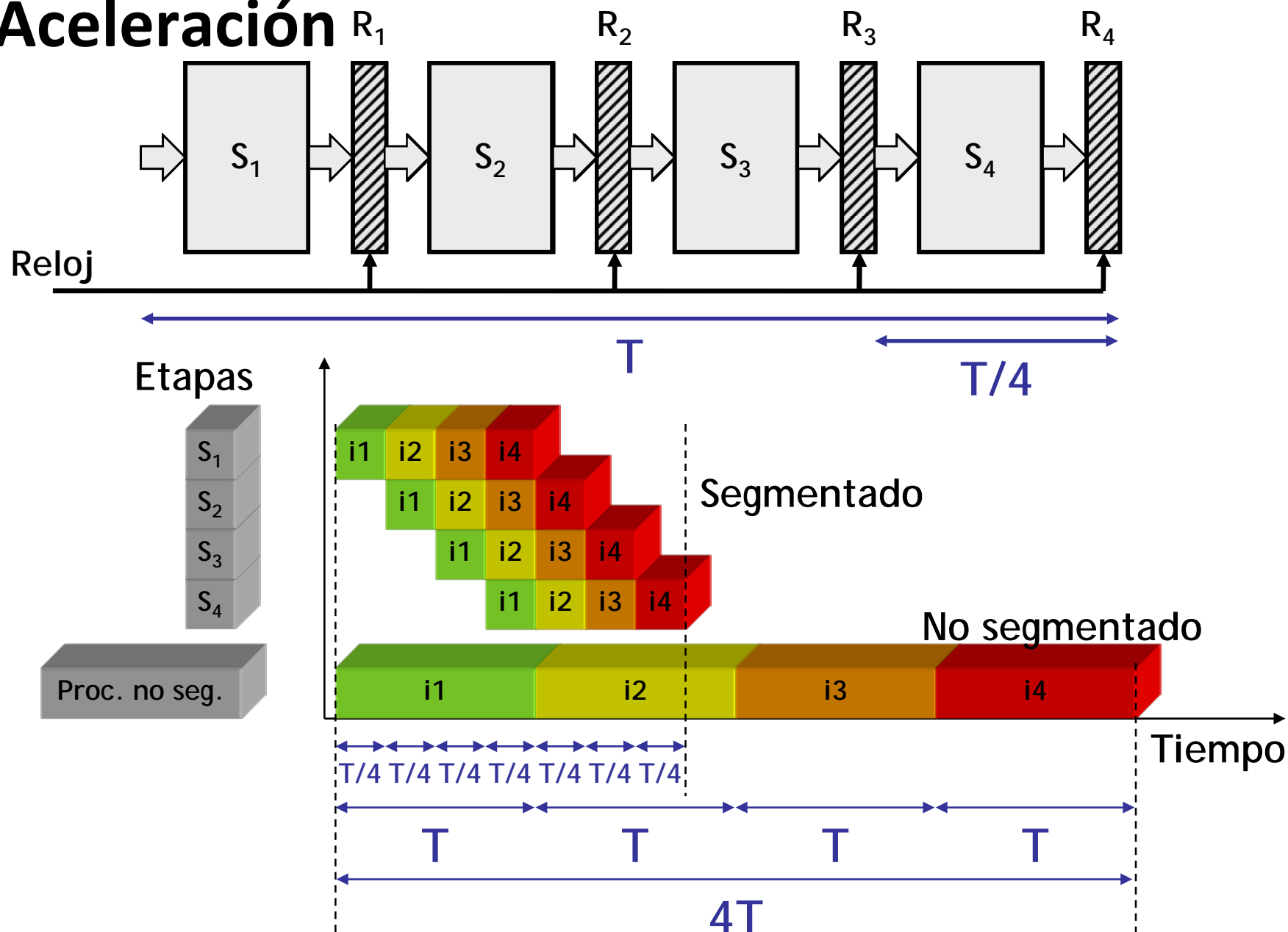
# Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- **Aceleración**
- Riesgos
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar

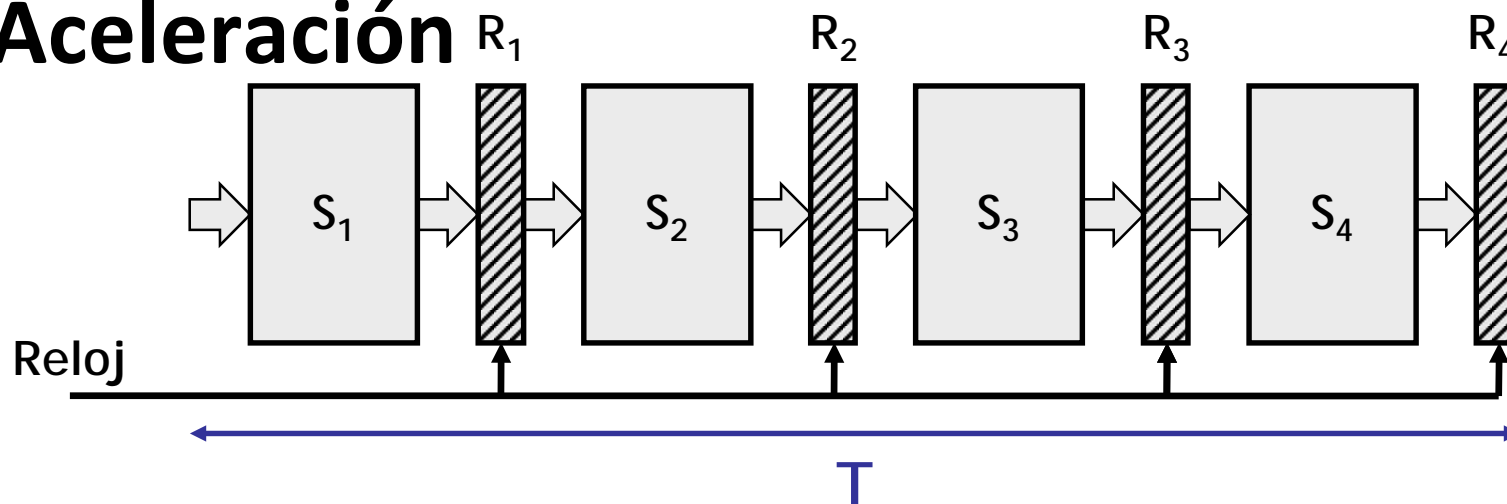
# Aceleración



# Aceleración



# Aceleración

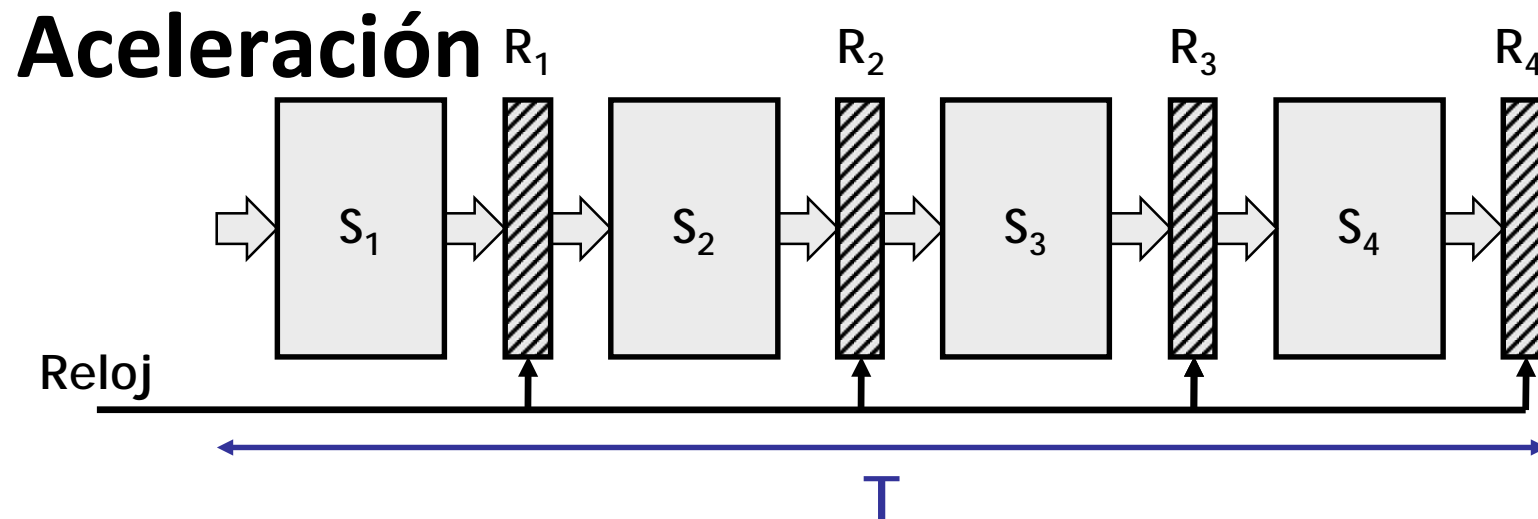


$$\text{Aceleración en el ejemplo} = \frac{\text{Tiempo máquina no-segmentada}}{\text{Tiempo máquina segmentada}} = \frac{4T}{7T/4}$$

$$\text{Aceleración ideal} = \frac{\overbrace{k \text{ instrucciones}}^{kT}}{(n-1+k)T/n} = \frac{nkT}{(n+k-1)T} \underset{k \rightarrow \infty}{=} n$$

$n$  etapas

La aceleración ideal coincide con el número de etapas de segmentación



## Causas que disminuyen la aceleración

- ✗ Coste de la segmentación
  - ✗ Duración del ciclo de reloj impuesto por etapa más lenta
  - ✗ Riesgos (*hazards*)  $\Rightarrow$  Bloqueo del avance de instrucciones
- $\Rightarrow T_c > T/n$

# Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- **Riesgos**
- Influencia en el repertorio de instrucciones
- Funcionamiento superescalar



# Riesgos

## Riesgo

Situación que impide la ejecución de la siguiente instrucción del flujo del programa durante el ciclo de reloj designado



Obliga a modificar la forma en la que avanzan las instrucciones hacia las etapas siguientes



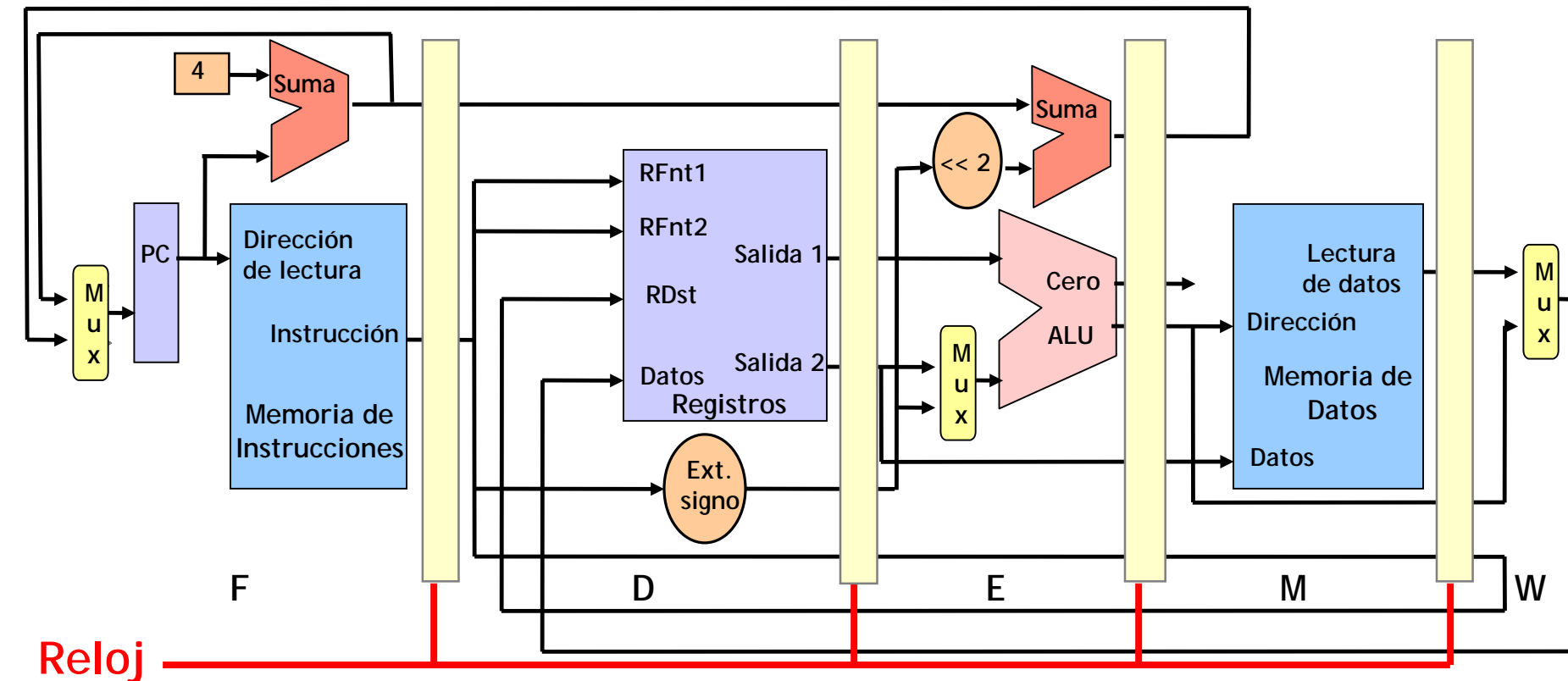
Reducción de las prestaciones logradas por la segmentación

# Riesgos

## ■ Supongamos las siguientes etapas:

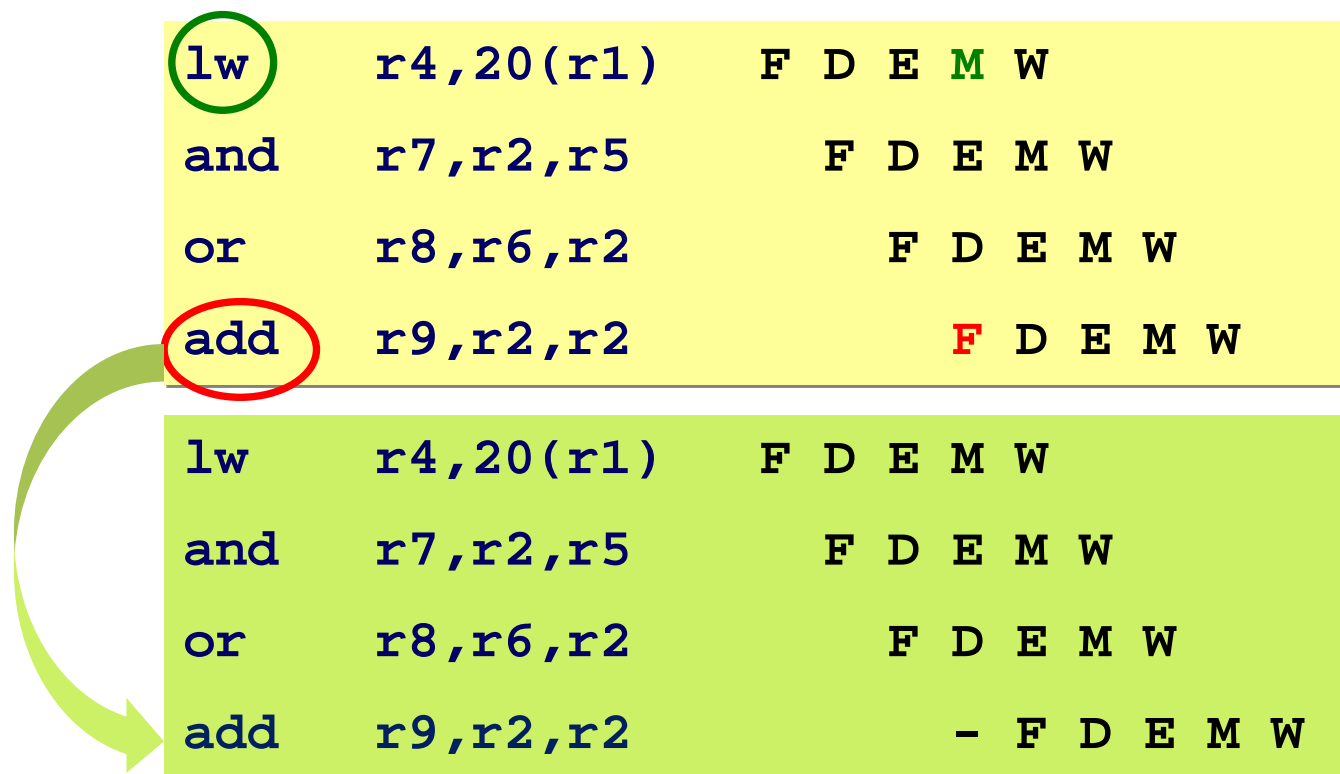
- **F**: búsqueda (**f**etch) de instrucción.
- **D**: **d**ecodificación de instrucción / lectura de registros.
- **E**: **e**jecución / cálculo de direcciones
- **M**: acceso a **m**emoria.
- **W**: escritura (**w**rite) de resultados.

# Riesgos



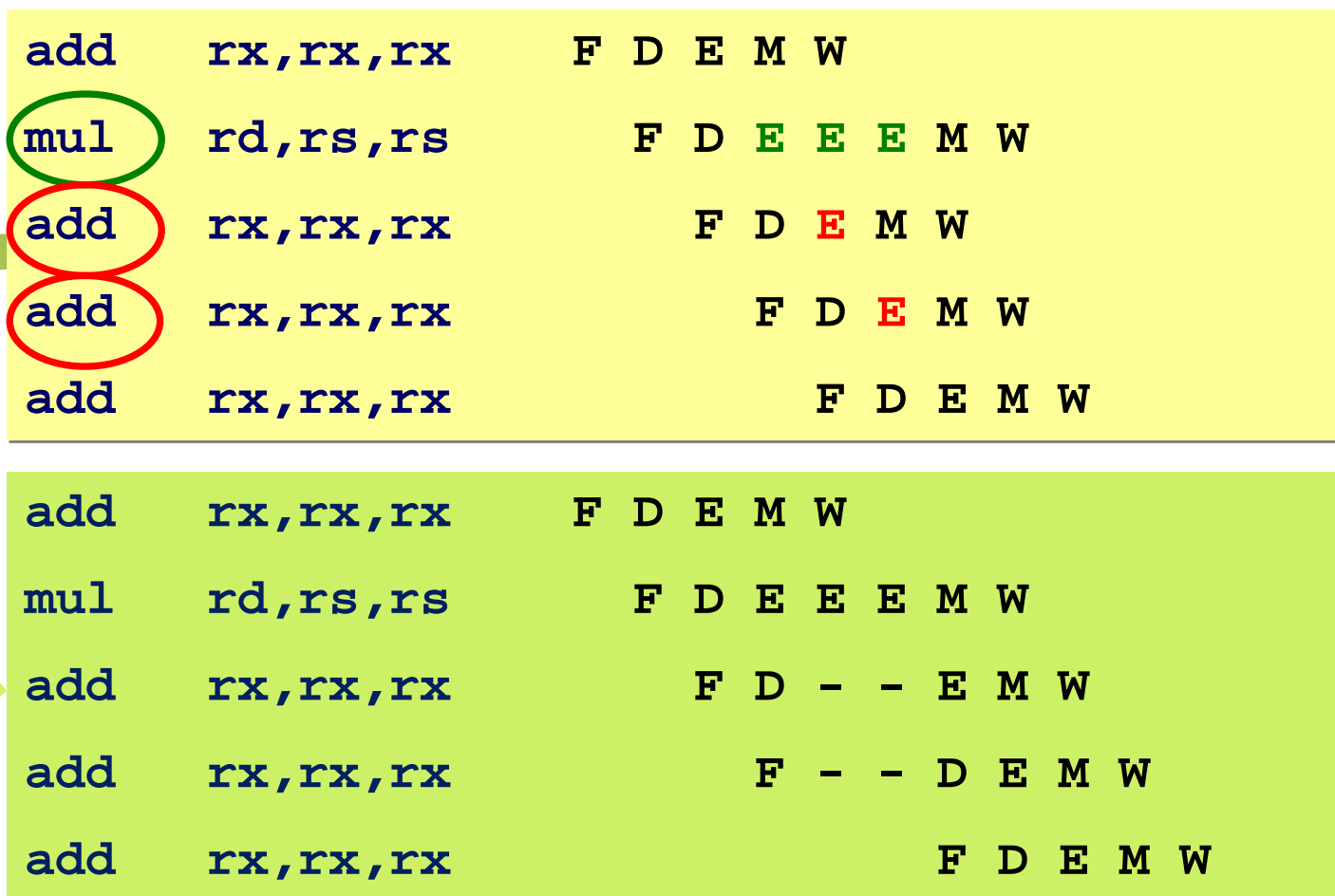
# Riesgos estructurales

- Conflicto por el **empleo de recursos**, dos instrucciones necesitan un mismo recurso.
- Ej. 1: lectura de dato + captación suponiendo **una única memoria** para datos e instrucciones.



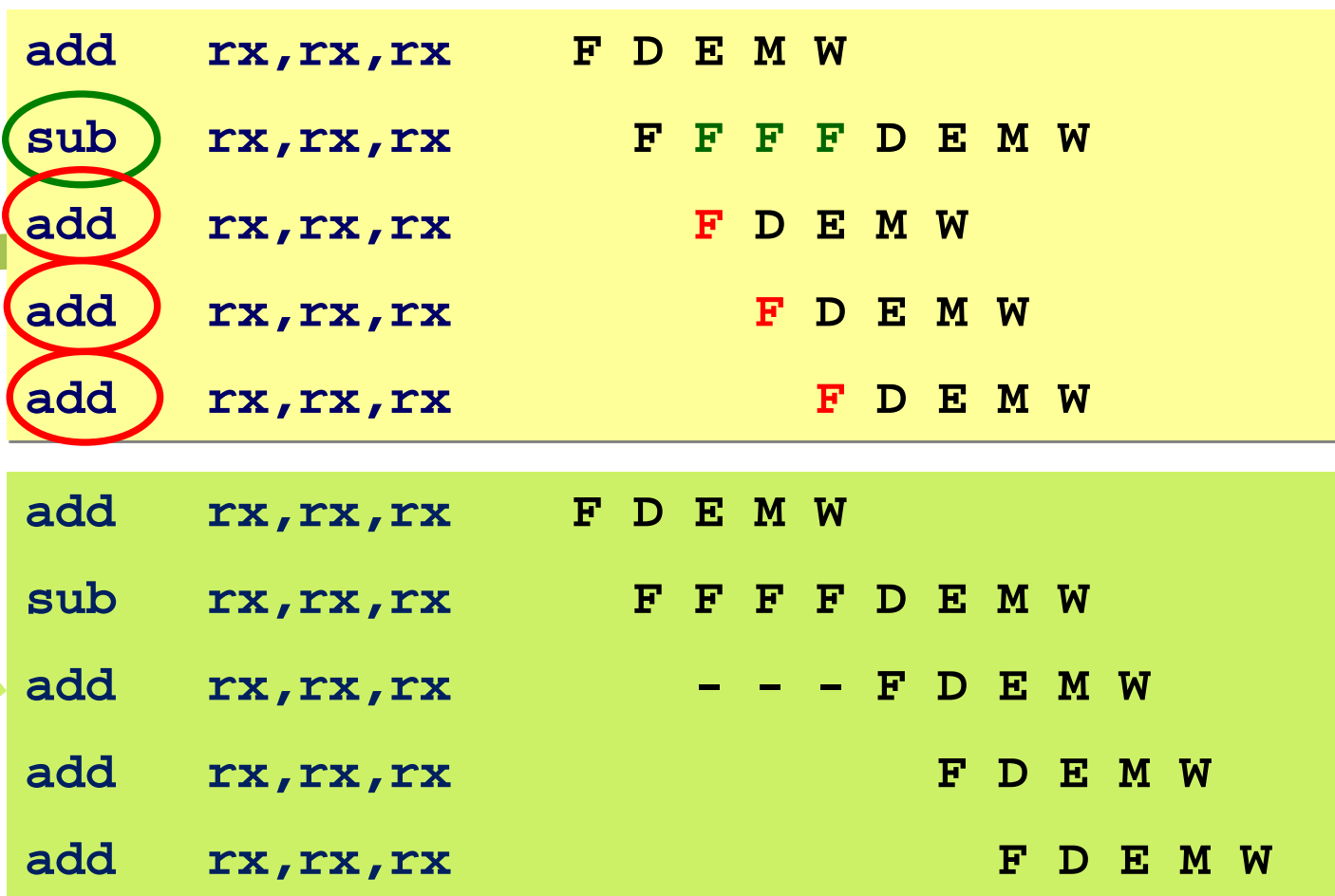
# Riesgos estructurales

- Ej. 2: ejecución de una operación con **más de un ciclo** en E.



# Riesgos estructurales

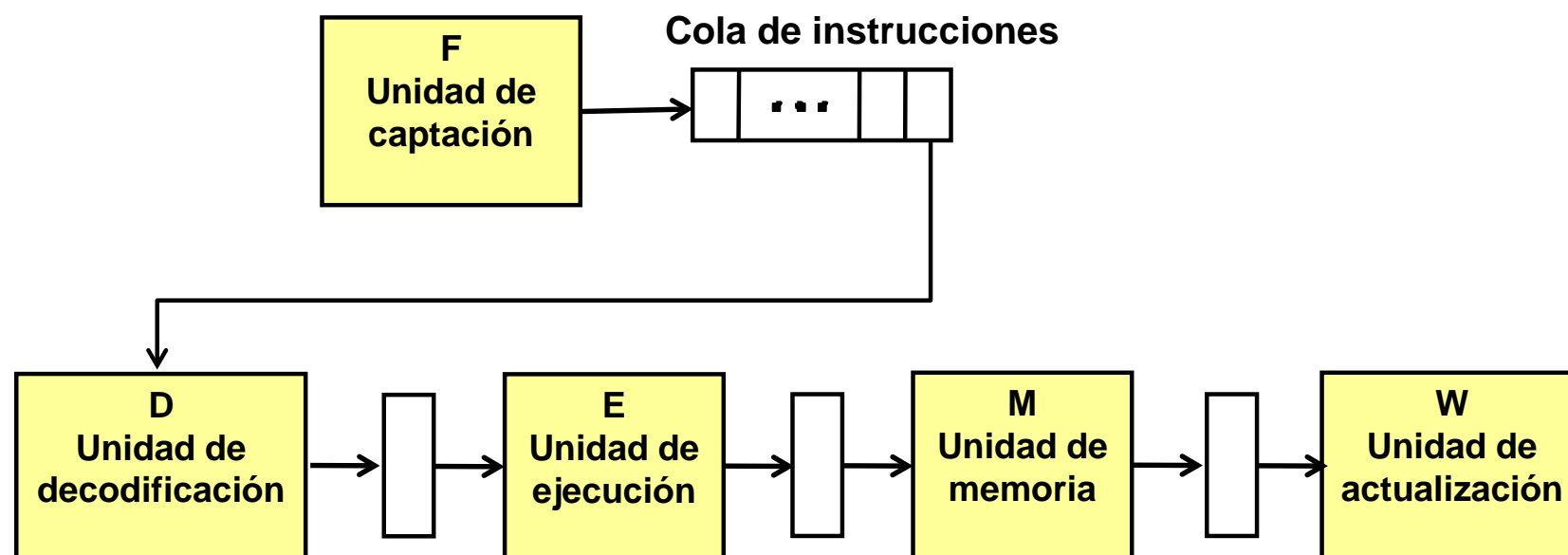
- Ej. 3: **fallo de caché** al captar una instrucción.






# Riesgos estructurales

- Para reducir el efecto de los fallos de caché se suelen captar instrucciones antes de que sean necesarias (precaptación) y se almacenan en una cola de instrucciones.



# Riesgos (por dependencias) de datos

- Acceso a datos cuyo valor actualizado depende de la ejecución de instrucciones precedentes.



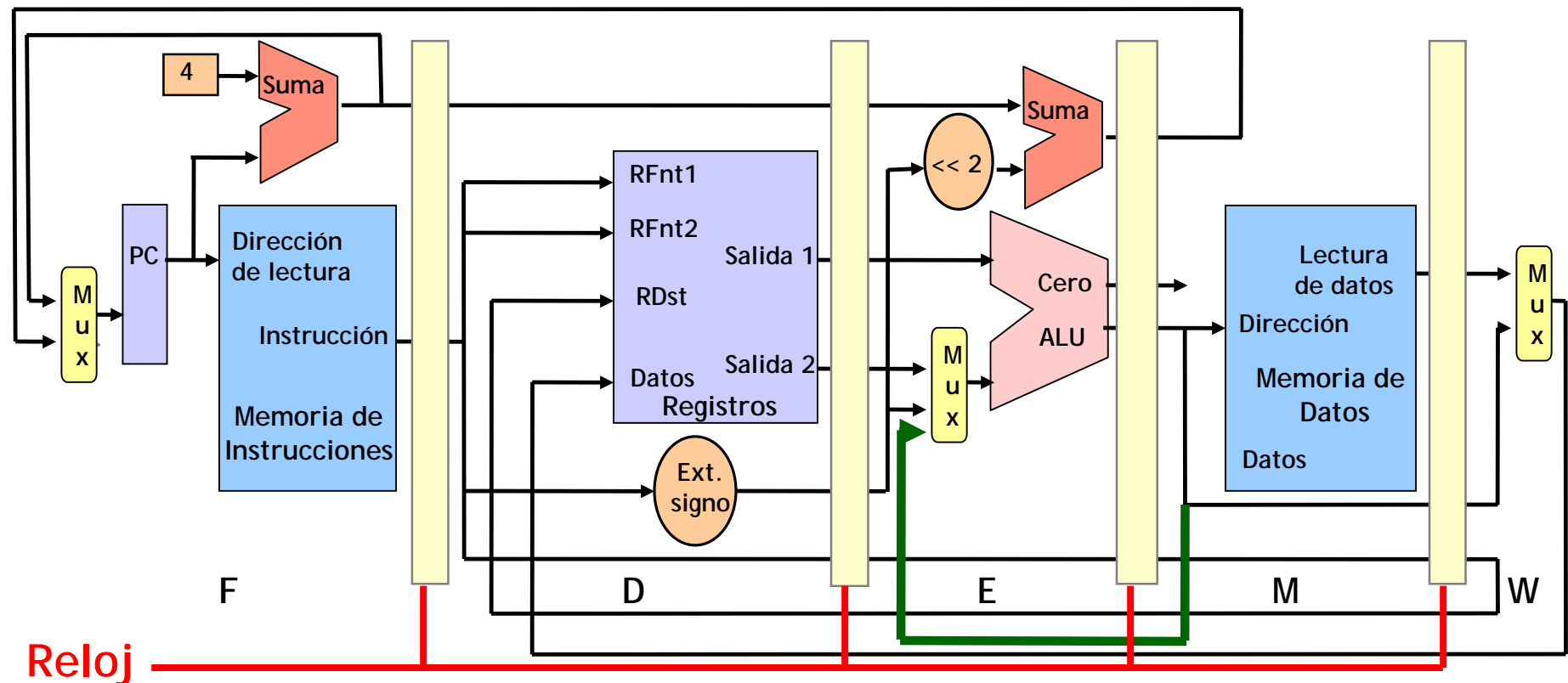
sub	r2,r1,r3	F	D	E	M	W
and	r7,r2,r5	F	D	E	M	W
or	r8,r6,r2	F	D	E	M	W
add	r9,r2,r2	F	D	W	M	W

sub	r2,r1,r3	F	D	E	M	W
and	r7,r2,r5	F	D	-	-	D E M W
or	r8,r6,r2	F	-	-	-	D E M W
add	r9,r2,r2					F D E M W

# Riesgos (por dependencias) de datos

- El nuevo r2 está a la salida de la ALU al terminar E.
- Si r2 se envía de nuevo a la ALU se elimina el retardo (*register forwarding*).



# Riesgos (por dependencias) de datos

- Las dependencias de datos las descubre el hardware al decodificar las instrucciones.
- Alternativamente puede resolverlas el compilador:

```
sub    r2,r1,r3    F D E M W
nop                                F D E M W
nop                                F D E M W
nop                                F D E M W
and    r7,r2,r5                                F D E M W
```

- **Ventajas:**
  - Hardware más simple
  - Reorganizar instrucciones para hacer trabajo útil en lugar de NOP
- **Inconveniente:**
  - Aumenta el tamaño del código

# Riesgos de control

- Consecuencia de la ejecución de instrucciones de salto.
- Salto **incondicional**:

<b>br</b>	L1	F	D	<b>E</b>	M	W
<b>and</b>	r2,r1,r4	<b>F</b>	<b>D</b>	-	-	-
<b>sub</b>	r5,r6,r7	<b>F</b>	-	-	-	-
or	r8,r1,r6					
L1:add	r6,r1,r4	F	D	E	M	W

- Se pierden 2 ciclos (**huecos de retardo de salto\***), ya que tras captar la instrucción br, hasta después del 3º ciclo (es decir, pasados otros 2 ciclos) no se conoce la dirección de salto.

# Riesgos de control

- Importante averiguar la dirección de salto **lo antes posible**, por ej. en la etapa de decodificación:

br	L1	F	D	E	M	W
and	r2,r1,r4	F	-	-	-	-
sub	r5,r6,r7					
or	r8,r1,r6					
L1:add	r6,r1,r4	F	D	E	M	W

- Se pierde 1 ciclo, ya que tras captar la instrucción br, después del 2º ciclo ya se conoce la dirección de salto.



# Riesgos de control

## ■ Salto **condicional**:

beq	r2,r3,L1	F	D	E	M	W
and	r2,r1,r4	F	-	-	-	-
sub	r5,r6,r7	F	-	-	-	-
or	r8,r1,r6					
L1:add	r6,r1,r4					
		F	D	E	M	W

- Si se produce el salto se pierden 2 ciclos.
- Si no se produce el salto, no se pierden ciclos.

# Riesgos de control

## ■ Degradación de prestaciones debida a los saltos.

- Supongamos algún mecanismo hardware que permita descartar la ejecución de las instrucciones siguientes si se produce el salto.
  - $b$ : nº de ciclos desperdiciados cuando se produce el salto.
  - $p_b$ : probabilidad de que se ejecute una instrucción de salto
    - (entre 0,15 y 0,30 normalmente)
  - $p_t$ : probabilidad de que realmente se produzca el salto cuando se ejecuta una instrucción de salto
  - $p_e = p_b p_t$ : probabilidad efectiva de que se produzca un salto
  - CPI: nº de ciclos de reloj por instrucción (suponer CPI = 1 sin saltos)

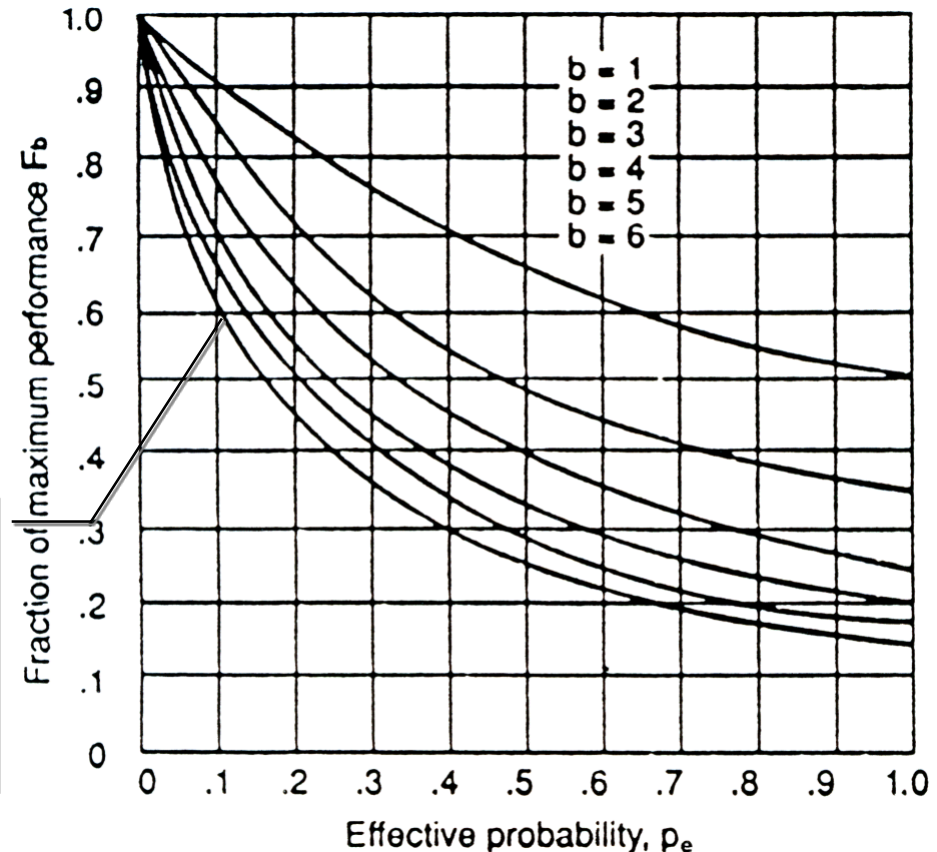
$$\text{CPI} = (1 - p_b) (1) + p_b [ p_t (1 + b) + (1 - p_t) (1) ] = 1 + p_b p_t b = 1 + p_e b$$

# Riesgos de control

- $F_b$ : fracción de máximas prestaciones (relación entre el nº de ciclos CPI si no hubiera saltos y el nº de ciclos CPI con saltos)

$$F_b = \frac{1}{1 + p_e b}$$

La degradación crece rápidamente al crecer  $p_e$  (más rápidamente a medida que  $b$  es mayor)



# Salto retardado (*delayed branch*)

- En lugar de desperdiciar las etapas posteriores a la de salto, una o más instrucciones parcialmente completadas se completarán antes de que el salto tenga efecto.
- El compilador busca instrucciones **anteriores** lógicamente al salto que pueda colocar tras el salto.
- Si el salto es condicional, las instrucciones colocadas detrás no deben afectar a la condición de salto.

Antes:

```
mov r1,#3  
jmp etiq  
nop
```

Después:

```
jmp etiq  
mov r1,#3
```

# Salto retardado (*delayed branch*)

- Otra posibilidad es colocar la(s) instrucción(es) **destino** de un salto tras el salto.

Antes:

```
    call subr
    nop
    ...
subr:
    mov r3,#100
```

Después:

```
    call subr+4
    mov r3,#100
    ...
subr:
    mov r3,#100
```

- Esto no funcionaría para saltos condicionales
- No podemos “subir” una instrucción que sólo tiene que ejecutarse algunas veces (cuando el salto se produce) a una posición donde siempre se ejecuta.

# *Annulling branch*

- Un salto de este tipo ejecuta la(s) instrucción(es) siguiente(s) sólo si el salto se produce, pero la(s) ignora si el salto no se produce.
- Con un salto de este tipo, el destino de un salto condicional sí puede colocarse tras el salto.

Antes:

```
    bz else
    nop
    ; código then
    ...
else:
    mov r3,#100
```

Después:

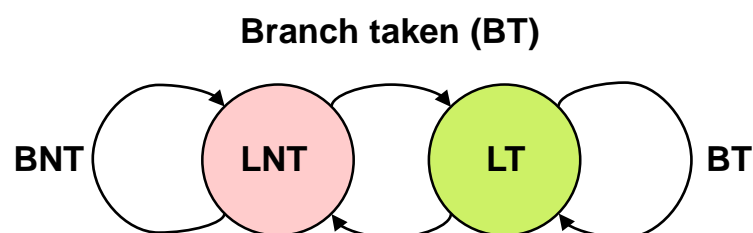
```
    bz,a else+4
    mov r3,#100
    ; código then
    ...
else:
    mov r3,#100
```

# Predicción de saltos

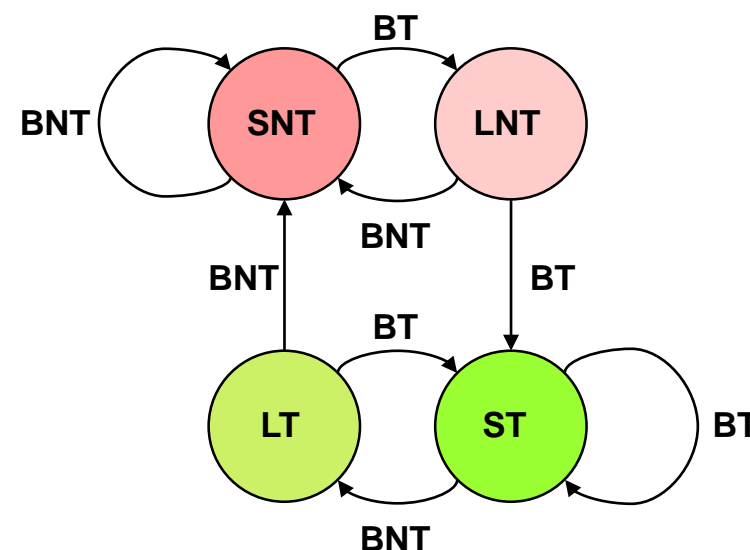
- Intentar predecir si una instrucción de salto concreta dará lugar a un salto (**branch taken**) o no (**branch not taken**).
  - Si los resultados de las instrucciones de salto condicional fueran aleatorios, comenzar a ejecutar las instrucciones siguientes al salto desperdiciaría ciclos en la mitad de las ocasiones.
  - Se pueden minimizar las pérdidas de ciclos inútiles si para cada instrucción de salto se puede predecir con un acierto  $> 50\%$  si el salto se producirá o no.
  - Se pueden hacer predicciones distintas si el salto es hacia direcciones menores o mayores.
- Tipos de predicción:
  - **Estática**: se toma la misma decisión para cada tipo de instrucción
  - **Dinámica**: cambia según la historia de ejecución de un programa

# Predicción dinámica de saltos

- ST: Muy probable saltar
- LT: Probable saltar
- LNT: Probable no saltar
- SNT: Muy probable no saltar



Branch taken (BT)  
Branch not taken (BNT)  
Algoritmo de dos estados  
(1 bit para cada  
instrucción)



Algoritmo de cuatro estados  
(2 bits para cada instrucción)



# Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- **Influencia en el repertorio de instrucciones**
- Funcionamiento superescalar

# Modos de direccionamiento

- Es deseable que un operando no necesite más de **un acceso a memoria**
  - Constante
  - Registro
  - Indirecto a través de registro
  - Indexado ( $EA = \text{reg.} + \text{desp.}$ , o  $EA = \text{reg.} + \text{reg.}$ )
- Es deseable que sólo puedan acceder a memoria las instrucciones de **carga y almacenamiento**

# Modos de direccionamiento

- Modo de direc. complejo, 2 accesos a memoria

lw	r4,(20(r1))	F	D	E	M	M	W
and	r7,r2,r5	F	D	E	-	M	W

- Modo de direc. más simple, 1 acceso a memoria

lw	r3,20(r1)	F	D	E	M	W
lw	r4,(r3)	F	D	E	M	W
and	r7,r2,r5	F	D	E	M	W

- Idéntica duración, pero hardware más sencillo

# Códigos de condición

- Las dependencias que introducen los bits de condición dificultan al compilador reordenar el código (deseable para evitar riesgos).
- En el ejemplo siguiente, la decisión de saltar se produce tras la etapa E de la instrucción `cmp`
- Es deseable **elegir en cada instr. si afecta o no a los cód. de condición**, para reordenar el código:

Antes:

```
add r1,r2  
cmp r3,r4  
jz eti
```

Después:

```
cmp r3,r4  
add r1,r2  
jz eti
```

Al reordenar el código, se puede tomar la decisión de salto un ciclo antes, y desperdiciar un ciclo menos tras el salto

# Segmentación de cauce

- Concepto de segmentación
- Ejemplo de segmentación
- Aceleración
- Riesgos
- Influencia en el repertorio de instrucciones
- **Funcionamiento superescalar**

# Funcionamiento superescalar

## ■ Procesamiento segmentado

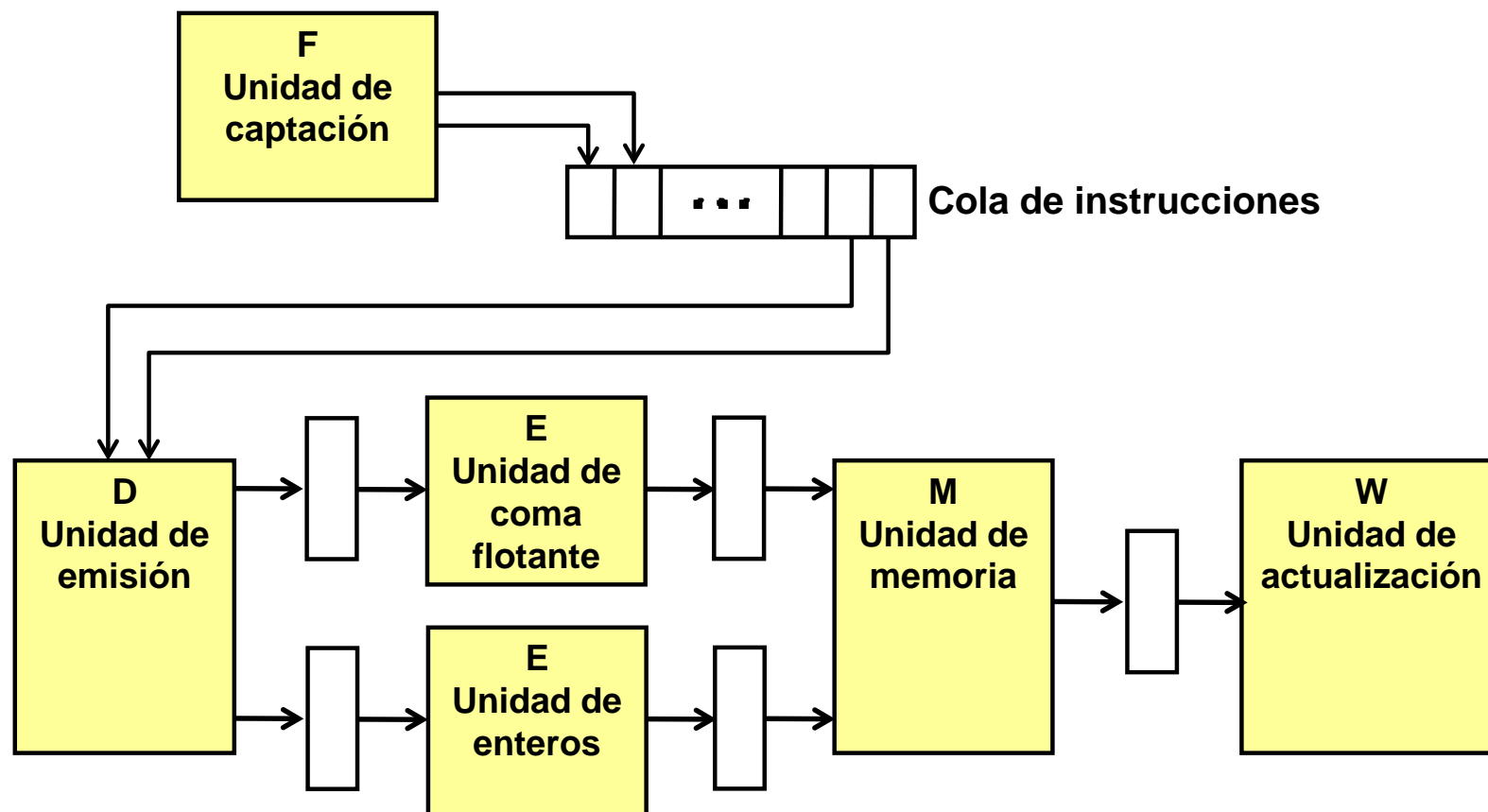
- Una instrucción tras otra
- Rendimiento ideal: una instrucción por ciclo

## ■ Procesamiento superescalar

- Varias instrucciones en **paralelo**
- Necesidad de **varias unidades funcionales**
- **Emisión múltiple**: se puede comenzar a ejecutar más de una instrucción por ciclo de reloj
- Rendimiento: **más de una instrucción por ciclo**
- Es fundamental poder captar instrucciones rápidamente: conexión ancha con caché + cola de instrucciones

# Funcionamiento superescalar

- Ej.: Procesador con dos unidades de ejecución:



# Funcionamiento superescalar

- El efecto negativo de los riesgos es más pronunciado.
- El compilador puede reordenar instrucciones para evitar riesgos.

<code>fadd</code>	<code>rx,rx,rx</code>	F D $E_1E_2E_3$ M W
<code>add</code>	<code>rx,rx,rx</code>	F D E M W
<code>fsub</code>	<code>rx,rx,rx</code>	F D $E_1E_2E_3$ M W
<code>sub</code>	<code>rx,rx,rx</code>	F D E M W

- Las instrucciones pueden emitirse en orden y finalizar de forma desordenada (ej. `add` finaliza antes que `fadd`)
  - Múltiples problemas y soluciones, que se verán en Arquitectura de Computadores



# Core i7 (Nehalem)

- **4 cores/chip**
- **Segmentación:**
  - 16 etapas
- **Superescalar:**
  - 4 instrucciones en paralelo
- **Caches:**
  - L1: 32KB I + 32KB D
  - L2: 256KB
  - L3: 8MB, compartida por los 4 cores

